Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Ingunn Johanne Vallestad

Path Following and Collision Avoidance for Marine Vessels with Deep Reinforcement Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2019

NTNU
Norwegian University of
Science and Technology

Ingunn Johanne Vallestad

Path Following and Collision Avoidance for Marine Vessels with Deep Reinforcement Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

This thesis is the result of my work at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology during the spring semester of 2019. It has utilised work carried out in my own project thesis of autumn 2018, in which a study of reinforcement learning methods was conducted and algorithms were implemented in the Python programming language. In the final stage of the project thesis, a DRL algorithm taken from the work of Lillicrap et. al. [1] was implemented and applied to a vehicle steering task. The DRL algorithm is reused here, but it has been adapted for application with two different marine craft models. Content relevant for this thesis has been rewritten and included here, and is comprised of background material in deep reinforcement learning (DRL) and artificial neural networks (ANNs), and implementation of the DRL algorithm.

This thesis is concerned with path following and collision avoidance algorithms for marine vessels with the use of DRL, applied to two different vessel models. Recent results within DRL path following in the Master's thesis of Martinsen [2] has been a source of inspiration and has provided a starting point for my own development of path following and collision avoidance algorithms.

A marine craft simulator was made available by DNV GL for the purpose of vessel modelling. Assistance in utilisation of the simulator was provided by Jon Arne Glomsrud, as well as example code on starting the simulator and how to communicate with it using the Python programming language. Furthermore, a model of a container vessel from the Marine Systems Simulator (MSS), specifically the MSS GNC Toolbox, of Fossen and Perez [3] was ported from MATLAB to Python.

The Tensorflow library, which is an open source software library for high performance numerical computation with strong support for machine learning and deep learning, was used for implementation of function approximators in the form of ANNs. Tensorflow provides building blocks for implementation and training of ANNs and includes automatic backpropagation and optimisa-

tion techniques. Additionally, the NumPy library for scientific computing with Python was used for performing numerical computations such as linear algebra.

Finally, invaluable insights into DRL, and ideas and discussion of how collision avoidance DRL algorithms may be developed, has been supplied by my supervisor, Anastasios Lekkas.

Ingunn Johanne Vallestad
Trondheim, June 17, 2019

Abstract

Interest in fully autonomous vehicle control has increased rapidly in recent years, motivated by promises of higher efficiency as well as reduced cost and environmental impact. Within vessel control, collision avoidance is a vital component of full autonomy, as it usually entails the following of a path as well as detection and avoidance of unforeseen obstacles. In marine navigation in particular, the vessel is also required to follow the International Regulations for Preventing Collision at Sea (COLREGS). The regulations were created to suit human reasoning and have not yet been fully adapted to the fixed nature of computers, thus complicating the development of autonomous marine vessels.

Advances within artificial intelligence and deep learning have supported the claims that intelligent autonomous systems are achievable, and deep reinforcement learning (DRL) is one of the fields that have shown great promise. DRL methods optimise behaviour based on a user-specified performance measure and require no a priori knowledge of dynamics of the controlled vessels or the world they operate in, and are therefore well suited for complex tasks involving environmental disturbances and inaccuracy in modelling. In this thesis, a DRL algorithm suitable for continuous systems will be used in the implementation of a path following system with surge control, which is applied to two different vessels. Results show a successful control system that is able to optimise its control input to achieve approximate path convergence.

The path following system is further developed to include collision avoidance, and experimental results in a collision avoidance situation with a container vessel show promising results. This illustrates the potential DRL has in solving complicated control tasks and indicates that completely autonomous collision avoidance can be developed using DRL.

Sammendrag

Interessen for fullt autonome kjøretøy har økt raskt i løpet av de siste årene, motivert av løfter om økt effektivitet samt reduserte kostnader og miljøpåvirkning. Innenfor fartøystyring er kollisjonsunngåelse en viktig del av full autonomi, da slike oppgaver vanligvis innebærer å følge en sti i tillegg til deteksjon og unngåelse av uforutsette hindringer. Spesielt for maritim navigasjon er at fartøyet også må følge de internasjonale reglene for kollisjonsunngåelse på sjøen (COLREGS). Reglene ble utarbeidet for å passe til menneskelig resonnering og har ennå ikke blitt tilpasset maskiners fastsatte natur, noe som gjør det utfordrende å utvikle autonome marine fartøy.

Fremskritt gjort innen kunstig intelligens og dyp læring støtter påstanden om at intelligente autonome systemer er innen rekkevidde, og dyp forsterkende læring (eng. deep reinforcement learning, DRL) er et av feltene som er svært lovende. DRL-metoder optimerer oppførsel basert på et brukerdefinert ytelsesmål og krever ingen tidligere kunnskap om de kontrollerte fartøyenes dynamikk eller om verdenen som opereres i, og er derfor velegnet for komplekse oppgaver der miljøforstyrrelser og modelleringsunøyaktigheter er til stede. I denne oppgaven vil en DRL-algoritme egnet for kontinuerlige systemer anvendes på to ulike fartøy i et sti-følgesystem med hastighetskontroll. Resultatene viser at kontrollsystemet er vellykket i den forstand at det kan optimalisere pådraget for å oppnå sti-følging.

Sti-følgesystemet har blitt videreutviklet for inkludering av kollisjonsunngåelse, og eksperimenter med et containerskip i en typisk situasjon innen kollisjonsunngåelse viser lovende resultater. Det foregående illustrerer dyp forsterkende lærings potensiale innen kompliserte oppgaver, og tyder på at DRL kan anvendes til utvikling av fullstendig autonome systemer for kollisjonsunngåelse.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

AI	Artificial intelligence
ANN	Artificial neural network
CA	Collision avoidance
COLREGS	The International Regulation for Preventing Collisions at Sea
DDPG	Deep deterministic policy gradient
DL	Deep learning
DNN	Deep neural network
DOF	Degree of freedom
DP	Dynamic programming
DRL	Deep reinforcement learning
IMO	International Maritime Organisation
LOS	Line-of-Sight
MDP	Markov decision process
ML	Machine learning

MPC	Model predictive control
NED	North-East-Down frame
POMDP	Partially observable Markov decision process
PSV	Platform supply vessel
ReLU	Rectified linear unit
RL	Reinforcement learning
TD	Temporal difference

Symbols

α_p	Path tangential angle
$\mathbf{T}$	Angular velocity transformation matrix
$\mathcal{A}$	Action space
$\mathcal{R}$	Reward function
$\mathcal{S}$	State space
$\mathcal{T}$	Transition model
$\boldsymbol{\eta}$	Pose vector of marine craft
$\boldsymbol{\nu}$	Velocity vector of marine craft
$\boldsymbol{\tau}$	Force vector of marine craft
$\boldsymbol{\theta}$	Parameter vector for a function approximator
$\boldsymbol{\Theta}_{nb}$	Euler angles
$\boldsymbol{a}$	Action vector in a deep reinforcement learning system
$\boldsymbol{b}$	Bias vector in artificial neural network
$\boldsymbol{p}$	Position vector

$\boldsymbol{R}_b^n$	Rotation matrix from BODY to NED frame
$\boldsymbol{s}$	State vector in a deep reinforcement learning system
$\boldsymbol{W}$	Weight matrix in artificial neural network
$\mu(\cdot)$	Deterministic policy
$\pi(\cdot)$	Stochastic policy
$A(s, a)$	Advantage function
d_{safe}	Safe distance in collision avoidance
L_{pp}	Length of marine craft
$Q(s, a)$	Estimated action-value function
$q(s, a)$	Action-value function
U	Total speed
$V(s)$	Estimated value function
$v(s)$	Value function

Subscripts and Superscripts

$*$	Optimality property
b	Coordinate in BODY frame
c	Commanded value
d	Desired value
e	Error
k	Current waypoint
n	Coordinate in North-East-Down frame
r	Relative value
t	Time step

Chapter 1

Introduction

1.1 Motivation

Autonomous systems are entering many areas of today's society, such as self-driving cars, household items like robotic lawn mowers and vacuum cleaners [7], autopilots in aircraft, and space exploration vehicles [8], to name a few. One of the advantages of autonomous vehicles is the possibility of increased efficiency and safety of operations. This is particularly prevalent in environments unsuitable for humans – such as prolonged underwater missions, areas exposed to radiation or toxic waste, and even planets like Mars – although the benefits of autonomy are present in all areas where autonomy is pushed.

In marine navigation, a significant incentive for making the transition to autonomous vehicles is the high density of large marine craft used for shipping of trade goods (which itself makes up 90% of world trade transport [9]), transportation of passengers, and the fishing industry, where accidents can have serious consequences for humans, the environment and other assets. Humans may make erroneous decisions or misjudge dangerous situations under stress, and this is reported to cause as many as 60% of accidents in ocean navigation [10], thus the development of fully autonomous vessels may reduce the risk of collisions, and consequently reduce cost and environmental impact. An additional improvement of autonomous vessels compared to human-operated ones is the potential for optimising the path travelled, for instance with respect to minimal

fuel consumption, or minimum time or distance.

In autonomous ship navigation, there are many challenges to deal with. A few examples include low-level control of under-actuated or fully-actuated vessels, path following, trajectory tracking and collision avoidance. Development of controllers include design and tuning of heading and speed controllers, specific for each vessel, and these implement a translation from a heading and speed error to control inputs to the vehicle. Such inputs can be e.g. rudder angles or thruster forces and directions, depending on the vessel construction. Path following and tracking involve development of heading and speed guidance laws that allow the vessel to follow or track a path, minimising position error. In the path following problem there are no temporal constraints imposed on the vehicle, thus the position error includes only a cross-track error, while the trajectory tracking problem includes the desired position of the vehicle along the path at each time instant.

The collision avoidance problem is among the most complicated navigation challenges, as it involves an assessment of the risk of collision with target vessels, and creating a plan for how to avoid them. This usually also includes an underlying path following task, in order to steer the vessel towards its final destination. To create some structure for this kind of navigation, the International Maritime Organization (IMO) have developed regulations that specify appropriate behaviour in various scenarios, known as COLREGS [11]. These regulations were originally intended as a guide for crew and captain manually controlling ships [12] and are therefore inherently vague, both in their description of appropriate actions as well as in distinguishing potential collision scenarios from each other. These kinds of rules are suited for human reasoning, but are required to be more specific in order to be tackled by computers in autonomous systems. In spite of the vague nature of COLREGS, obeying them is necessary for making the interaction between autonomous and human control systems predictable, and therefore COLREGS plays a crucial role in paving the way for autonomy in marine navigation.

Usually, previous works in areas of marine vessel navigation have been based on models that represent vessel dynamics and kinematics, which are then used to develop guidance and control laws that achieve a control objective, util-

ising suitable methods from linear or nonlinear control theory. A common approach is to place the vessel model and guidance and control systems in a cascaded structure, where the control system drives the vessel actuators, while the guidance system provides input to the control system. Some examples include Line-of-Sight (LOS) systems for path following [13, 14], and collision avoidance (CA) control systems that separate the CA module from the rest of the system, either by switching between path following and CA guidance systems [15], or by letting the CA system modify the nominal guidance law [12]. An accurate model with respect to the vessel and environmental forces it is based on can result in completely predictable behaviour when the control laws are employed in the real vessel, while a model with fewer non-linearities, and that is placed in a cascaded structure, may simplify the stability analysis during design of control laws. Thus, a compromise must be made in the design of the system, sometimes trading off accuracy for less complicated control laws.

Reinforcement learning (RL) is a branch of artificial intelligence developed for optimal system performance where there is uncertainty in environment and/or the system itself. As opposed to the previously mentioned guidance and control laws which are based on instructive feedback, RL is based on evaluative feedback. This feedback is called the *reward signal*, which encapsulates the control objective by giving high reward for actions resulting in something good and low reward, or penalty, for undesired outcomes. The RL algorithm then adjusts its *policy* (the mapping from observations to actions) through exploration of the possible solutions, in order to find the optimal solution to the control objective. An advantage of RL algorithms is their ability to solve complex problems without, or with only partial, knowledge of the system it controls and its surroundings. This is usually referred to as being *model-free*, and means that vessel models and descriptions of environmental disturbances need not be known for the RL agent to solve a task, as long as the reward signal, actions and observations are defined suitably.

In recent years, a combination of RL and deep neural networks has emerged, and received attention as a theory for employing RL in problems with high-dimensional or continuous state space and action space. Neural networks are

used for representation purposes (e.g. for the policy), which allow the RL algorithm to approximate functions effectively without visiting every combination of states and actions. This is called deep reinforcement learning (DRL). Significant results include agents learning to play Atari games [16] and the achievement of super-human performance in the game of Go, without any knowledge beyond game rules [17], in which DRL architectures are applied to systems of high-dimensional state spaces. The results of [1, 18–20] have shown success in applications such as vessel path following, trajectory tracking for AUVs (autonomous underwater vehicles), and control tasks such as cartpole swing-up, dexterous manipulation, legged locomotion and car driving, which include continuous state and action spaces.

The advantages of RL and DRL, and the following literature review, support the possibility of utilising DRL in the development of complicated control systems for use in autonomous vessels. This thesis will focus on path following and collision avoidance.

1.2 Literature review

For underactuated marine surface vessels, the 3 degree of freedom (3 DOF) path following and trajectory tracking problem has received much attention. The problems are interesting because conventional ships usually have two available controls – propellers for surge control and rudders for turning control – and thus sideways speed (sway) is uncontrolled. In other words, the vessels are underactuated because they are steered in 3 DOF using two control inputs. The controllers for yaw and speed are usually decoupled, where the focus lies in development of yaw controllers that solve the path following problem. An example of a 3 degrees of freedom path following controller for an underactuated marine vessel is given in [13]. Here, a path was generated as straight line segments connecting several waypoints, and a Line-of-Sight (LOS) guidance system with yaw and surge controllers was derived using backstepping in order to obtain a dynamic feedback controller that includes the uncontrolled sway mode. For curved paths, both path planning and guidance is addressed in [14]. A monotone cubic Hermite spline interpolation (CHSI) method is used for generating a path

between waypoints that avoids zigzag behaviour when switching between waypoints. The LOS guidance law that is used includes a time-varying look-ahead distance, which results in less oscillatory behaviour around the desired path compared to a constant look-ahead distance. When unknown environmental forces such as current are included, the LOS guidance law is often augmented to account for these disturbances, which may be challenging due to the modelling uncertainties that arise. An example is to use adaptive feedback linearisation combined with sliding mode [21].

The above mentioned path following results have in common that they do not consider the possibility of obstacles and the need to avoid these. Since the development of fully autonomous ships require the incorporation of collision avoidance, a review of some CA approaches follows.

One of the first COLREGS-compliant control systems for autonomous marine vehicles was introduced by Benjamin et. al. [22] in 2004, and the presented paper addresses challenges related to interpretation of the COLREGS and uses a behaviour-based control architecture to encapsulate both the precision and flexibility inherent to the rules.

Previous works based on model predictive control (MPC) have shown that by simulation of a finite set of control behaviours in a collision avoidance setting, the optimal behaviour can be selected by comparison of an associated cost function based on hazard [12, 23]. MPC is a powerful framework for the CA problem because constraints of operation, the minimisation of danger, and other objectives can be formalised as a cost function and constraints in a numerical optimisation problem, and it can be applied to systems with nonlinear vehicle models and uncertain environmental forces. However, solving numerical optimisation problems may take time, and collision avoidance scenarios require real-time solutions. Therefore care must be taken to ensure a solution is found as quickly as possible. In order to reduce computational complexity, the numerical optimisation problem is avoided by replacing it with finite-horizon simulation, where a fixed set of scenarios are evaluated and compared. Additionally, only a single change in control input is considered per prediction horizon, since increasing the number of control input changes will cause the number of simulation scenarios to increase exponentially. The CA system is designed as its own module

that communicates with a guidance system by modifying the course and speed commands proposed by the guidance, before passing them to the vessel control system. In the most recent approach [23], the CA module does not simulate the guidance system to predict its proposed commands, but rather obtains them directly. This module can thus be inserted in existing systems with any guidance module. The vessel used to illustrate the results of the approach has relatively fast dynamics, thus it is argued that the vessel model used by the CA system to predict behaviour can be reduced to the kinematic equation, implying instant turning and no drift due to wind and ocean current. If one wishes to apply this to larger ships where the instant turn assumption is infeasible, it is possible that a more complex vessel model should be applied. This would in turn increase the computational cost of the approach. The proposed CA method adheres to COLREGS by including a penalty component in the MPC cost function for violating COLREGS, as well as somthing called a *COLREGS-transitional* cost, penalising control behaviours that abort a COLREGS-compliant manoeuvre.

An alternative approach is using a set-based method for switching between a path following mode and a collision avoidance mode [15]. This architecture enables the user to decide which combination of path following controller and CA controller to be used, independently of the switching mechanism. The path following controller used here is LOS with extensions that ensure compensation of disturbances caused by environmental forces, without knowledge of their magnitude or direction. It is shown that the proposed controllers depend only on absolute velocity measurements, circumventing the need for estimation of ocean currents, and that the collision avoidance guidance law guarantees the tracking of a circular path around moving obstacles with a safe radius. No information about the obstacle dimension or dynamics is required, only position and velocity. It is proven that the vessel converges to the desired path when in path following mode, and that obstacles are avoided in a COLREGS-compliant manner.

The reviewed CA approaches cover only a fraction of the research on the field. Some other commonly applied methods include artificial potential fields [24, 25]; where potential fields attracting the vehicle to a goal and repelling it from obstacles are defined, dynamic window [26]; where a search for velocities that are reachable for the vessel under its dynamic constraints and safe with

respect to obstacles is made, and velocity obstacles [27, 28]; which generates a cone in velocity space consisting of all velocity vectors a vessel can choose that can lead to a collision, and then ensures the velocity vector of the vessel lies in the safe region outside this cone.

The above discussed methods include some drawbacks. The computation time in the MPC-based methods is one of them, where the number of possible behaviours must be severely limited in order to compute the optimal one in the limited time between detection of an obstacle and a potential collision. All approaches discussed in any depth here also require complex control laws to be able to comply with the navigational rules of COLREGS. An advantage of the approaches is that mathematical proof can be found which show guaranteed avoidance of collision if avoidance is possible. However, the proofs assume the models of vessel dynamics and environmental forces used in derivation of control laws is accurate, and changes to these may cause the algorithms to fail. A potential solution to the issue of online computational complexity is to use reinforcement learning. The reward function of the RL framework is similar to the cost function of MPC, but RL algorithms may be preferred in CA systems due to their low computational cost at run-time. That is, once an RL system has found a good control behaviour through learning, it requires very little effort to compute. Another one of the major assets of reinforcement learning is, as pointed out in Section 1.1, their applicability in systems with uncertainty regarding vessel dynamics and behaviour of environmental disturbances. This makes DRL algorithms good candidates for applications in marine control problems, and a few recent works are summarised below. The focus is on DRL for path following and collision avoidance.

When it comes to path following, it has been demonstrated that a deep reinforcement learning based controller for path following can replace the guidance system and controller of conventional underactuated ships, applying appropriate rudder angle and propulsion commands directly to the vessel [18, 19]. Experimental results show that the controller is able to compensate for unknown disturbances, can be applied to different vessel types, and that it outperforms traditional Line-of-Sight. The reward signal design is simple while also ensuring smooth rudder commands.

A DRL obstacle avoidance algorithm for underactuated marine vessels is shown in [29], in which the control behaviours are defined as discrete increments or decrements of propeller force, and application of force to either sides of the rudder, causing a yaw moment that increases or decreases the rate of turn. The experiments illustrate that the vessel successfully finds a collision-free path through an area of obstacles, however all obstacles are stationary. Additionally, the control system is not designed to abide by COLREGS. An approach outlined by Shen et. al [30] uses a deep Q-learning framework, with discrete action space, for automatic collision avoidance of multiple ships. Human knowledge and experience is incorporated by several techniques, including a polygon shaped “bumper" defining detection distances to obstacles surrounding the vessel, with larger required distance in the front. The bumper is then used to decide when to switch between normal mode and collision avoidance mode. Similarly, the reward signal is designed so that differing distances are defined as collision, depending on where the obstacle is located. Results indicate that the proposed algorithm can avoid collision in cluttered environments while accounting for navigational rules, both in simulated environments and an experimental setting using miniature ships. However, the many heuristics used to incorporate existing knowledge into the controller makes the design slightly cluttered and introduces a fair amount of design parameters.

Trajectory tracking [20] and position tracking [31] of autonomous underwater vehicles has been achieved using the *deep deterministic policy gradient* algorithm. In [31], a goal-driven architecture is applied, translating the goal and vehicle position to a relative position. The agent learns thruster inputs for six thrusters directly from measurements. A cooperative multi-vehicle collision avoidance system is created in [32], where a value function is learned through approximate value iteration with least-squares regression. Two vehicles cooperate to avoid collision between them and return to their respective paths. To choose from continuous actions, a search algorithm chooses the best turning speed for both vessels.

Evidently, DRL for contnuous control is a rapidly expanding research field, thus only a few examples of specific application in marine vessel control and collision avoidance are addressed here. It is expected that by the time of thesis

completion, a growing number of publications will be available for review.

1.3 Objective and contributions

The ultimate objective of this thesis is to implement an autonomous collision avoidance control system for a marine vessel by using deep reinforcement learning, and analysing the performance. The suggested method can provide an alternative way to solve the collision avoidance problem, and has the advantage of being model-free while also optimising a performance measure. This calls for time-consuming computations ahead of deployment, but is nonetheless applicable in real-time situations requiring short reaction time once learning is completed and the system is deployed. DRL also includes the possibility of creating end-to-end control laws, eliminating the need for cascaded system structures where performance of one module is highly dependent on performance of others. However, DRL does not eliminate the *possibility* of cascaded structures, and we will see that DRL and already developed control laws can be used concurrently.

The intention was to use a provided vessel model with accompanying simulation interface given by DNV GL for all experiments, due to the simulator being equipped with functionality for environmental disturbances, obstacle vessels and implementations of path following and collision avoidance algorithms. However, complexity of the simulation interface provided a disadvantage in training of DRL algorithms, and it was deemed necessary to include a second vessel model in the thesis. The second vessel is a container type, and the first one is a platform supply vessel, from now on referred to as a PSV.

A step-by-step approach is used in order to arrive at the final goal, starting with an initial implementation of a plain DRL guidance system for a platform supply vessel. The next step is a path following system tested on two vessel types with the inclusion of surge control, and finally the thesis culminates in application of DRL in a collision avoidance system.
The main contributions of this thesis are as follows:

- Create a DRL guidance system for a PSV, with vessel model and simula-

tor provided by DNV GL, feeding heading and surge references to corresponding controllers included with the PSV.

- Implementation, tests and analysis of DRL straight-path following control structure originally presented in [2, 18] for two vessel types:
  - For the kind of container vessel also used in [2, 18].
  - For a PSV.
- Adaptation of said path following method to include (i) surge control in the case of a container and (ii) guidance system for surge in the case of a PSV.
- Creation of a collision avoidance system trained for a head-on situation using DRL, and analysis of results in said situation.
- Adaptation of the CA system to follow one of the rules in COLREGS, and suggestions on how to comply with other rules.

1.4 Outline of report

The thesis consists of five chapters, including this introductory chapter whose aim has been to give an introduction to motivation behind the thesis, an overview of related work and a specification of the main objective and contributions. Chapter 2 presents relevant theoretical background, by briefly explaining main concepts within modelling and guidance for marine vessels, before moving on to a thorough description of deep reinforcement learning and the algorithm used in the following chapters. In Chapter 3 we address the objectives of the thesis by building a control algorithm step by step, where the end goal is to perform collision avoidance and path following for a marine vessel. Simulation results of the implemented control systems are presented and discussed in Chapter 4. This chapter ends with a brief summary of the findings and a discussion of suggested future work. Chapter 5 gives a brief conclusion of the thesis.

A description of the container vessel model can be found in Appendix A, while parameter values for the performance measures of controllers and a few additional simulation plots are placed in Appendix B and C, respectively.

Chapter 2

Theory

The following chapter contains relevant background theory for this thesis. First, an introduction to modelling of marine vessels is given in Section 2.1, before essential concepts and notation within straight-path following for marine vessels are described in Section 2.2. Section 2.3 provides some insight into rules and regulations for navigation at sea, called COLREGS. Finally, deep reinforcement learning is introduced and thoroughly explained in Section 2.4. Here, fundamental ideas within reinforcement learning are gone through first in Section 2.4.1, then deep neural networks are introduced as a powerful function approximation tool in Section 2.4.2, before a description of the deep deterministic policy algorithm is given in Section 2.4.3.

2.1 Marine vessel model

Kinematics is the analysis of geometrical aspects of motion, and are important for the development of equations of motion for marine craft. The equations of motion describe both kinematics and kinetics (which describe the forces causing the motion), and are needed in order to simulate and analyse a vessel. Usually, this model of the system is necessary for the design of control laws and guidance systems, but as will be seen, a controller based on DRL may not require a model at all. However, an introduction is included here for the sake of completeness, to describe notation and because a vessel model has been used for simulation.

The following is a summary of relevant sections in Fossen's *Handbook of Marine Craft Hydrodynamics and Motion Control* [5].

For marine craft that can move along and rotate around all 3 coordinate axes (x, y, and z), we say that it moves in six *degrees of freedom* (DOF), and thus six coordinates are needed to determine position and orientation, called *surge*, *sway*, *heave*, *roll*, *pitch* and *yaw*. The notation used here is given by Table 2.1 and an illustration is given in Figure 2.1.

DOF		Forces / moments	Linear / angular velocities	Positions / Euler angles
1	surge	X	u	x
2	sway	Y	v	y
3	heave	Z	w	z
4	roll	K	p	ϕ
5	pitch	M	q	θ
6	yaw	N	r	ψ

Table 2.1: The notation of SNAME [4] for marine vessels [5]

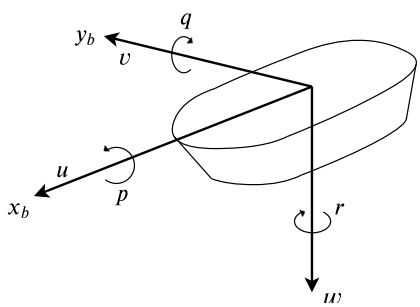


Figure 2.1: Linear and angular velocities in BODY frame

The North-East-Down (NED) coordinate system $\{n\} = (x_n, y_n, z_n)$ is defined as the tangent plane on the surface of the earth moving with the craft, but for vessels operating in a local area, a frame fixed to a point on the earth's surface can be used as an inertial coordinate system. The x-axis points to the true North, the y-axis points towards East, and the z-axis points downwards.

A body-fixed coordinate frame $\{b\} = (x_b, y_b, z_b)$ (denoted as BODY frame)

is a moving coordinate frame with origin fixed to the craft, x-axis pointing from aft to fore, y-axis pointing starboard and z-axis in the direction from top to bottom of the craft. It is used for expressing the linear and angular velocities of a craft, whereas the position and orientation are described relative to the inertial frame.

By defining vector notation for variables in BODY and NED frame, where o_b is the origin of $\{b\}$:

$\boldsymbol{p}^n_{b/n}$	position of the point o_b with respect to $\{n\}$ expressed in $\{n\}$
$\boldsymbol{v}^b_{b/n}$	linear velocity of the point o_b with respect to $\{n\}$ expressed in $\{b\}$
$\boldsymbol{f}^b_b$	force with line of action through the point o_b expressed in $\{b\}$
$\boldsymbol{\Theta}_{nb}$	Euler angles between $\{n\}$ and $\{b\}$
$\boldsymbol{\omega}^b_{b/n}$	angular velocity of $\{b\}$ with respect to $\{n\}$ expressed in $\{b\}$
$\boldsymbol{m}^b_b$	moment about the point o_b expressed in $\{b\}$

the components of Table 2.1 can be grouped in vectors:

NED position	$\boldsymbol{p}^n_{b/n} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$	Attitude	$\boldsymbol{\Theta}_{nb} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \in \mathcal{S}^3$
Body-fixed linear velocity	$\boldsymbol{v}^b_{b/n} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \in \mathbb{R}^3$	Body-fixed angular velocity	$\boldsymbol{\omega}^b_{b/n} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \in \mathbb{R}^3$
Body-fixed force	$\boldsymbol{f}^b_b = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \in \mathbb{R}^3$	Body-fixed moment	$\boldsymbol{m}^b_b = \begin{bmatrix} K \\ M \\ N \end{bmatrix} \in \mathbb{R}^3$

where $\mathbb{R}^3$ is the three-dimensional Euclidean space and $\mathcal{S}^3$ is the sphere given by three angles on the interval $[0, 2\pi]$. This gives the pose vector $\boldsymbol{\eta}$, the velocity vector $\boldsymbol{\nu}$, and a force vector $\boldsymbol{\tau}$ that describe the general motion of a marine craft

in 6 DOF:

$$\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{p}^n_{b/n} \\ \boldsymbol{\Theta}_{nb} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} \boldsymbol{v}^b_{b/n} \\ \boldsymbol{\omega}^b_{b/n} \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{f}^b_b \\ \boldsymbol{m}^b_b \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ K \\ M \\ N \end{bmatrix} \tag{2.1}$$

Then the equations of motion for a marine craft moving in 6 DOF, without wind and current forces, can be written as [5]:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{2.2}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{\eta}) + \boldsymbol{g}_0 = \boldsymbol{\tau} \tag{2.3}$$

where $\boldsymbol{M}$, $\boldsymbol{C}(\boldsymbol{\nu})$, and $\boldsymbol{D}(\boldsymbol{\nu})$ are the inertia matrix, Coriolis–centripetal matrix and dampening matrix of the craft, respectively. The matrix $\boldsymbol{J}(\boldsymbol{\eta})$ describes a transformation from BODY to NED frame, and consists of a linear velocity rotation matrix $\boldsymbol{R}^n_b(\Theta_{nb})$ and an angular velocity transformation $\mathbf{T}(\Theta_{nb})$:

$$\boldsymbol{J}(\boldsymbol{\eta}) = \begin{bmatrix} \boldsymbol{R}^n_b(\Theta_{nb}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{T}(\Theta_{nb}) \end{bmatrix} \tag{2.4}$$

where (if we abbreviate $\sin$ to s, $\cos$ to c and $\tan$ to t)

$$\begin{aligned} \boldsymbol{R}^n_b(\Theta_{nb}) &= \boldsymbol{R}_{z,\psi}\boldsymbol{R}_{y,\theta}\boldsymbol{R}_{x,\phi} \\ &= \begin{bmatrix} \mathrm{c}\psi & -\mathrm{s}\psi & 0 \\ \mathrm{s}\psi & \mathrm{c}\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{c}\theta & 0 & \mathrm{s}\theta \\ 0 & 1 & 0 \\ -\mathrm{s}\theta & 0 & \mathrm{c}\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathrm{c}\phi & -\mathrm{s}\phi \\ 0 & \mathrm{s}\phi & \mathrm{c}\phi \end{bmatrix} \end{aligned} \tag{2.5}$$

$$\mathbf{T}(\Theta_{nb}) = \begin{bmatrix} 1 & \mathrm{s}\psi\mathrm{t}\theta & \mathrm{c}\phi\mathrm{t}\theta \\ 0 & \mathrm{c}\phi & -\mathrm{s}\phi \\ 0 & \mathrm{s}\phi/\mathrm{c}\theta & \mathrm{c}\phi/\mathrm{c}\theta \end{bmatrix} \tag{2.6}$$

The components $\boldsymbol{g}(\boldsymbol{\eta})$ and $\boldsymbol{g}_0$ are hydrostatic forces. For marine craft mov-

ing in the horizontal plane, the hydrostatic forces can be neglected, and the equations are reduced to

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{2.7}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{2.8}$$

Course, *heading* and *sideslip* are angular variables useful for manoeuvring of marine craft in the horizontal plane, and the relationship between them is given by (2.9) and illustrated in Figure 2.2. The course angle χ is the angle from the x^n-axis of $\{n\}$ to the velocity vector of the vessel, while the heading (yaw) angle ψ is the angle from the x^n-axis to the x^b-axis of $\{b\}$. Sideslip β can be explained as the difference between a craft's orientation and its direction of travel, hence it can become nonzero when a vessel is exposed to ocean currents or during a turning motion.

$$\chi = \psi + \beta \tag{2.9}$$

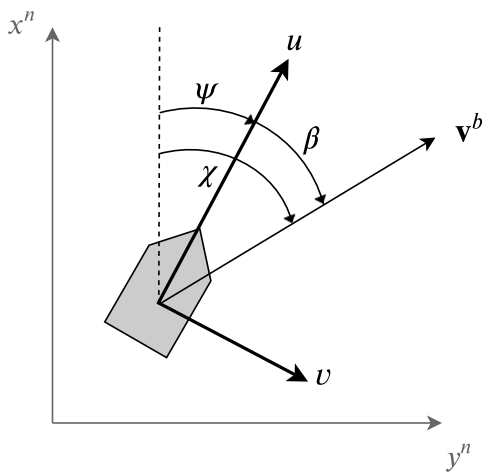


Figure 2.2: Relationship between course χ, heading ψ and sideslip β

2.2 Straight-line path following

This section will introduce terms and notation which are important when discussing path following on the ocean surface.

Path following is the task of controlling a vessel such that it converges to a predefined path, and the simplest task is to follow straight-line paths. There are no restrictions placed on temporal propagation along the path, so the path following objective in the horizontal plane can be achieved by choosing any speed $U = \|\boldsymbol{v}_{b/n}^b\| = \sqrt{u^2 + v^2} > 0$.

A path consisting of straight line segments is defined by a collection of waypoints, where each one is specified using coordinates in $\{n\}$: $\boldsymbol{p}_k^n = [x_k, y_k]^\top$ for $k = 1, ..., n$. If we consider a path decided by the straight line between two waypoints $\boldsymbol{p}_k^n$ and $\boldsymbol{p}_{k+1}^n$, its positive angle relative to the x^n-axis is given by

$$\alpha_p = \operatorname{atan2}\left(y_{k+1} - y_k, x_{k+1} - x_k\right) \tag{2.10}$$

where $\operatorname{atan2}(y, x) \in [-\pi, \pi]$ is the same as $\arctan(y/x) \in [-\pi/2, \pi/2]$ defined in four quadrants. To achieve a positive angle from this, one must wrap the resulting α_p to $[0, 2\pi]$.

Then, coordinates of the vessel can be computed in a path-fixed coordinate frame rotated by α_p around the z^n-axis and with origin placed in $\boldsymbol{p}_k^n$. An illustration of the path-fixed coordinate system and its relation to the NED system can be found in Figure 2.3. The rotation matrix corresponding to this transformation is

$$\boldsymbol{R}(\alpha_p) = \begin{bmatrix} \cos\alpha_p & -\sin\alpha_p \\ \sin\alpha_p & \cos\alpha_p \end{bmatrix} \tag{2.11}$$

and the path-fixed coordinates of a vessel at position $\boldsymbol{p}^n = [x, y]^\top$ can be computed by

$$\boldsymbol{p} = \begin{bmatrix} x_e \\ y_e \end{bmatrix} = \boldsymbol{R}(\alpha_p)^\top \left(\boldsymbol{p}^n - \boldsymbol{p}_k^n\right) \tag{2.12}$$

where x_e is the along-track position of the vessel, and y_e is the cross-track position, usually named *cross-track error*. Thus, the control objective for straight-line

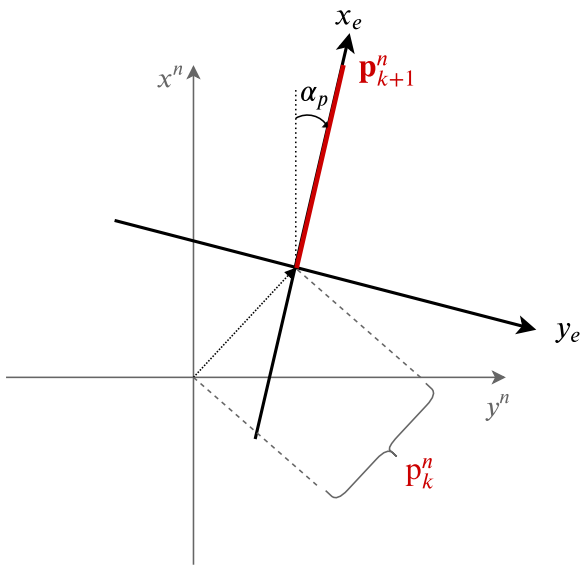


Figure 2.3: Illustration of path-fixed coordinate system. The NED frame is grey, the path-fixed frame is black, and the path given by waypoints $\boldsymbol{p}_k^n$ and $\boldsymbol{p}_{k+1}^n$ is shown in red.

path following is reducing the cross-track error to zero:

$$\lim_{t \to \infty} y_e(t) = 0 \tag{2.13}$$

In order to move along a path made up of straight lines connected by waypoints, it is necessary to design a switching mechanism for selecting which straight line segment to follow. Since a line is decided by two consecutive waypoints $\boldsymbol{p}_k^n$ and $\boldsymbol{p}_{k+1}^n$ in a database consisting of n points, the switching can be made when the craft lies within a circle of acceptance with radius R_{k+1} around $[x_{k+1}, y_{k+1}]^\top$. At this point, the waypoints of the path switch to $\boldsymbol{p}_{k+1}^n = \boldsymbol{p}_{k+2}^n$ and $\boldsymbol{p}_k^n = \boldsymbol{p}_{k+1}^n$. The switching condition is then

$$(x_{k+1} - x)^2 + (y_{k+1} - y)^2 \leq R_{k+1}^2 \tag{2.14}$$

where a guideline for choosing the radius of acceptance is $R_{k+1} = 2L_{pp}$, and L_{pp} is the ship length.

A commonly used path following method is Line-of-Sight (LOS) guidance, which produces a desired yaw angle ψ_d or course angle χ_d. The performance of LOS is therefore dependent on the performance of the system's heading controller, which translates a desired heading or course to a low-level control input (i.e. rudder angle or thruster configuration). The steering law can be selected as

$$\chi_d = \alpha_p + \chi_r \tag{2.15}$$

where χ_r is a velocity-path relative angle describing the angle between the path and velocity vector of the vessel, chosen to ensure convergence to the path. In the *lookahead-based* steering scheme, this angle is given by

$$\chi_r = \arctan\left(\frac{-y_e}{\Delta_{y_e}}\right) \tag{2.16}$$

and the lookahead distance $\Delta_{y_e} > 0$ represents the point on the path towards which the velocity is directed, a distance Δ_{y_e} ahead of the projection of $\boldsymbol{p}^n$ on to the path [5].

2.3 Collision avoidance and COLREGS

COLREGS [11] describe a set of traffic rules that vessels at sea must follow, established by the International Maritime Organisation in 1972. The rules make other vessels' future trajectories more predictable for the crew/control system aboard ships. However, one must be prepared for the situation where others are unable, or unwilling, to follow these rules.

Due to the rules being developed for human operators, some of the rules may appear vague when it comes to applying them to autonomous ships. They are intended for use in many different vessels, and with the subjective opinions of many different people, thus defining exact rules are a challenge. When a rule states that a vessel shall "take early and substantial action to keep well clear [of the other vessel]", it is important to acknowledge that difficulties may arise in letting an autonomous collision avoidance algorithm know what exactly this means, in terms of e.g distance, speed and direction of travel.

2.3.1 Relevant rules

The rules directly relevant to this thesis are given here, taken from the International Regulations for Preventing Collisions at Sea [11] from 1972. An illustration of Rule 14 can be found in Figure 2.4.

Rule 14: Head-on situation

> *(a). When two power-driven vessels are meeting on reciprocal or nearly reciprocal courses so as to involve risk of collision each shall alter her course to starboard so that each shall pass on the port side of the other.*
>
> *(b). Such a situation shall be deemed to exist when a vessel sees the other ahead or nearly ahead and by night she could see the masthead lights of the other in a line or nearly in a line and/or both sidelights and by day she observes the corresponding aspect of the other vessel.*
>
> *(c). When a vessel is in any doubt as to whether such a situation exists she shall assume that it does exist and act accordingly.* ([11])

Rule 16: Action by give-way vessel

> *Every vessel which is directed to keep out of the way of another vessel shall, so far as possible, take early and substantial action to keep well clear.* ([11])

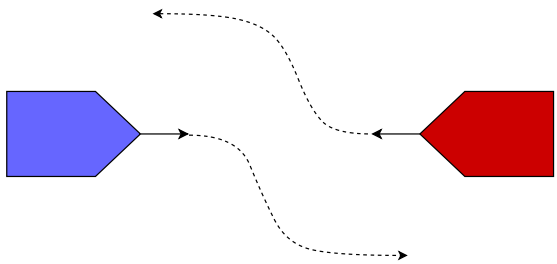


Figure 2.4: Head-on situation. Vessels should alter their course to starboard.

2.4 Deep reinforcement learning

The control approaches studied in this thesis are based on deep reinforcement learning (DRL), therefore this section will provide the necessary material in order to understand what DRL is, how it can be implemented, and what its advantages may be.

Deep reinforcement learning refers to methods that use deep artificial neural networks to find an approximately optimal solution to a reinforcement learning problem. This section is therefore structured as follows: Section 2.4.1 introduces the reinforcement learning problem and some significant solution approaches, such as policy gradient and actor-critic methods. Furthermore, it is explained that using a function approximator is often crucial when applying RL to real-world problems, since their states can be continuous – with infinitely many possible values – and an approximator can represent the full state-space from a finite set of parameters. Then, neural networks are introduced in Section 2.4.2, as an efficient way to perform function approximation. Finally, the algorithm used in this thesis is detailed in Section 2.4.3.

2.4.1 Reinforcement learning

The core of reinforcement learning is learning from experience. The RL agent initially has no knowledge of its environment or objective, and in order to achieve its objective it explores the world around it to learn what to do. Along the way it receives some evaluative feedback in the form of a *reward signal* on whether its current choice of action was good or not. This feedback signal thus defines the objective of the agent, and instructs the agent in what it should keep doing, and what kind of behaviour should be avoided. Eventually, the agent should achieve near-optimal behaviour with respect to maximising the reward signal.

Figure 2.5 illustrates the main components of reinforcement learning. It includes an *agent* and an *environment*, and they interact through signals called *actions*, *states* and *rewards*. The agent consists of a decision making algorithm with states as inputs and actions as outputs, controlling for instance a ship or a car. This is the part that learns from the observed reward signal. The environment can be described as the world the agent operates within. The agent's

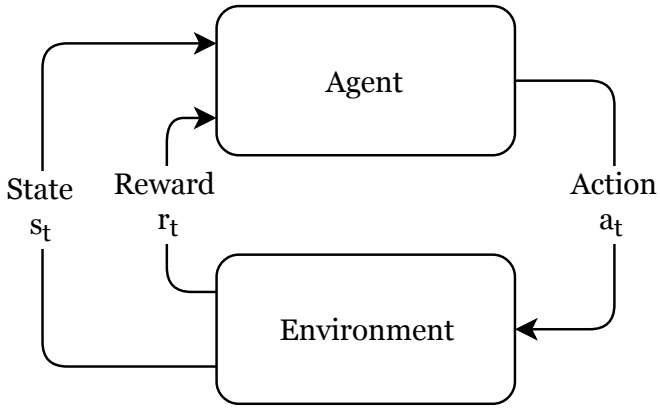


Figure 2.5: Reinforcement learning

knowledge of the world is incorporated into the state, while the action is used to influence the environment in order to change the state and receive a new reward. The details of how this is computed are unknown to the agent.

2.4.1.1 Markov decision processes

When formalising reinforcement learning problems, a framework called Markov Decision Processes (MDPs) is used. An MDP describes a sequential decision problem where the environment is fully observable (i. e. the observed state is the true state of the environment), the transition model $\mathcal{T}$ is stochastic and Markovian, and there is a reward function $\mathcal{R}$ associated with the transition from state s to s' through the action a.

For $\mathcal{T}$ to be Markovian, the Markov property must be assumed for all states. This means that the current state depends on only a finite fixed number of previous states [33]. To simplify this, it can be argued that a state depending on any number of previous states can be converted into depending only on the one immediately preceding it by changing the state to incorporate the appropriate history. Thus, a state in an MDP contains all information about past states that make a difference for the future.

Then, the MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of all states, $\mathcal{A}(s)$, $s \in \mathcal{S}$ is the set of actions associated with each state, $\mathcal{T}$ is the environment's stochastic transition model, and $\mathcal{R}$ is the reward function. $\mathcal{T}$ is

given by

$$\mathcal{T}(s, a, s') = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right] \tag{2.17}$$

and $\mathcal{R}$ is the expected immediate reward when performing an action a in state s, as given below.

$$\mathcal{R}(s, a) = \mathbb{E}\left[R_t | S_t = s, A_t = a\right] \tag{2.18}$$

A partially observable MDP (POMDP) is different from MDPs in that the environment is only partially observable. In other words, the true state s is not available to the agent, which must rely on a partial observation o instead, in which some information is hidden from the agent. In such a process, the state space $\mathcal{S}$ is replaced by the observation space $\mathcal{O}$ to clarify that the true state is not observed.

2.4.1.2 Policies, value functions and the Bellman equation

In order to understand how an RL agent learns, first we must define *what* it learns. The agent can learn a policy, a value representation, or both, that define its behaviour, as will be detailed in the following sections. A policy is a function describing the rules an agent follows when deciding actions, depending on the current state. It can be deterministic, usually denoted by μ as $a = \mu(s)$, which gives a single action corresponding to each state. Another alternative is a stochastic policy, which gives a probability distribution over actions, denoted π. To choose an action the stochastic policy must be sampled, as $a \sim \pi(\cdot|s)$.

The value function provides the agent with some measure of how good it perceives being in a particular state (and performing and action) to be. The value thus gives information about what cumulative reward is expected when starting in this state or state-action pair, given that the agent follows a chosen policy in the future.

The term *return* is usually used when talking about cumulative reward in RL. The return can be defined as the sum of rewards in a fixed window of time steps (called an episode), such as

$$G_t = \mathbb{E}\left[\sum_{t=0}^{T} R_t\right] \tag{2.19}$$

or it can be an infinite-horizon discounted return,

$$G_t = \mathbb{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}\right] \tag{2.20}$$

which is used in cases where there is no natural end to a sequence of states and actions, or we simply do not care as much about future reward as immediate reward. Because of the discount factor $\gamma \in [0, 1)$, the sum converges to a finite value, and is a mathematically convenient notation. Unless otherwise stated, this thesis will use the discounted return from now on.

Returning to the value function, we can now define some functions. The *state-value* function $v^\pi(s)$ is the expected return given if the agent starts in state s and follows policy π:

$$v^\pi(s) = \mathbb{E}_{a\sim\pi}\left[G_t | S_t = s\right] \tag{2.21}$$

The *action-value* function $q^\pi(s, a)$ gives the expected return when starting in state s and taking action a, and thereafter following policy π:

$$q^\pi(s, a) = \mathbb{E}_{a\sim\pi, s\sim\mathcal{T}}\left[G_t | S_t = s, A_t = a\right] \tag{2.22}$$

From this, it is clear that the value can be viewed as the sum of the immediate reward of being in the current state and the discounted value of the next state, as shown in Equations (2.23), (2.24).

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_{a\sim\pi}\left[R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...) \mid S_t = s\right] \\ &= \mathbb{E}_{a\sim\pi}\left[R_t + \gamma G_{t+1} \mid S_t = s\right] \\ &= \mathbb{E}_{a\sim\pi}\left[R_t + \gamma v^\pi(S_{t+1}) \mid S_t = s\right] \end{aligned} \tag{2.23}$$

$$q^\pi(s, a) = \mathbb{E}_{a\sim\pi, s\sim\mathcal{T}}\left[R_t + \gamma v^\pi(S_{t+1}) \mid S_t = s, A_t = a\right] \tag{2.24}$$

A policy π is defined as better than or equal to another policy π' if $v^\pi(s) \geq v^{\pi'}(s)\ \forall s \in \mathcal{S}$, thus by following an optimal policy π^*, one can obtain optimal value functions, $v^*(s) \triangleq \max_\pi v^\pi(s)$ and $q^*(s, a) \triangleq \max_\pi q^\pi(s, a)$. The superscript * denotes optimal solutions.

These definitions bring us to the Bellman optimality equations, which are recursive equations describing the relationship between the value of a state and its successor states [34]. These equations are important in this context because their solution is the solution to the reinforcement learning problem, as they uniquely define the optimal value function for a given MDP. The basis of Bellman optimality is the following relationship,

$$v^*(s) = \max_{a \in \mathcal{A}(s)} q^*(s, a) \tag{2.25}$$

stating that the state-value when following an optimal policy π^* from a state must be equal to the best (maximum) action-value in the same state. This gives the Bellman optimality equations for v^* and q^*:

$$\begin{aligned} v^*(s) &= \max_{a \in \mathcal{A}(s)} q^*(s, a) \\ &= \max_a \mathbb{E}_{s \sim \mathcal{T}} \left[R_t + \gamma v^*(S_{t+1}) \mid S_t = s, A_t = a \right] \\ &= \max_a \sum_{s', r} \mathcal{T}(s, a, s') \left[r + \gamma v^*(s') \right] \end{aligned} \tag{2.26}$$

$$\begin{aligned} q^*(s, a) &= \mathbb{E}_{a \sim \pi, s \sim \mathcal{T}} \left[R_t + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} q^*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} \mathcal{T}(s, a, s') \left[r + \gamma \max_{a' \in \mathcal{A}(s')} q^*(s', a') \right] \end{aligned} \tag{2.27}$$

where s', a' denote the next state and action after being in state s taking action a, and r is the immediate reward received from this transition, given by $\mathcal{R}$. Once the value function is found, one can establish the policy by choosing the action that moves the agent to the neighbouring state with the highest value.

2.4.1.3 Learning approaches

As mentioned, solving the Bellman optimality equations solves the RL problem. There are several ways to solve them explicitly, such as *value iteration* and *policy iteration* within dynamic programming. However, since the equations are a set of nonlinear equations corresponding to each state or state-action pair,

solving all the equations quickly becomes computationally expensive and time-consuming in large state or action spaces. Additionally, the transition model of the MDP must be known in order to compute the value function. Thus, these methods are only applicable when the state and action spaces are discrete and finite and the environment model is known, and they will not be explained further in this thesis. The reader is referred to the book *Reinforcement Learning: An Introduction* by R. Sutton and A. Barto [34] for further information.

The reinforcement learning methods are usually divided into three separate approaches – policy-based or *actor-only* methods, in which a policy is learned explicitly; value-based or *critic-only* methods, where the agent learns the value function (implicitly defining the policy); and *actor-critic* methods, where both policy and value functions are learned. This chapter will focus on actor-only and actor-critic methods, since the critic-only methods have the drawback of needing to iterate through all available actions at each state in order to determine which action gives highest expected return. They are thus unsuitable in continuous action spaces.

Usually, approximate solution methods must be employed. This means that the value function or policy, or both, must be estimated based on observed transitions rather than knowledge of transition models. The experience can be sampled from real or simulated transitions, and estimates can be updated after each step in the environment, after n steps, or after an episode is completed. The latter case is called *Monte Carlo* learning, and assumes that episodes terminate after a finite number of time steps. This way of sampling full episodes is called *off-line* training, while the opposite (updates after every step) is called *online* training. Methods that update their estimates before the episode is completed are part of *temporal-difference* (TD) learning, and they work as such: observe the reward received by following a policy for n steps, and combine this (discounted) sum with the estimated value of the state at step number $n + 1$, to be denoted as the *TD target*. Then, the estimate of the value function is moved towards the target using a step size α. With $n = 1$, we get the expressions

$$V(s) = V(s) + \alpha \left[r + \gamma V(s') - V(s) \right] \tag{2.28}$$

$$Q(s,a) = Q(s,a) + \alpha \underbrace{\Big[\underbrace{r + \gamma Q(s',a')}_{\text{TD target}} - Q(s,a) \Big]}_{\text{TD error, } \delta_t} \tag{2.29}$$

where V, Q are estimates of v^π, q^π.

An example of a TD algoritm that finds the optimal policy is **SARSA**, which is an *on-policy* method. Thus, the action-value q^π is estimated for the policy π, while simultaneously π is changed towards greediness with respect to the action-value [34, p. 129]. This means that the next action, a' of Equation (2.29) is decided by the policy. According to Sutton and Barto [34], the algorithm converges to an optimal π^* and q^* with probability 1 if the following conditions are met: (i) all state-action pairs are visited an infinite number of times and (ii) the policy converges to the greedy policy.

Q-learning is an example of an *off-policy* TD algorithm, in which the optimal q^* is approximated directly, independently of the policy. In terms of Equation (2.29), a' is chosen greedily as the action that maximises Q: $\delta_t = r + \gamma \max_{a'} Q(s',a') - Q(s,a)$. The policy's purpose in this case is to determine which actions and states to visit so that the environment is explored. Q converges to q^* with probability 1 as long as all state-action pairs continue to be visited and thus its estimate is updated.

SARSA and Q-learning are tabular solution methods, used in discrete problems, meaning that they keep track of Q and π as tables, where each entry corresponds to a state-action pair. They both learn an action-value function, and are thus critic-only methods.

When the action and/or state space is continuous, tabular methods can not be applied. A simple solution to this challenge is to discretise the spaces, but a too fine-grained discretisation may lead to a slow, inefficient algorithm, while information can be lost from very coarse discretisation. Thus, another approach is investigated here – **function approximation**. This entails parameterising functions by parameter vectors, rather than using lookup tables. The parameter vector is usually denoted $\boldsymbol{\theta} \in \mathbb{R}^d$. If we have a function $F(x)$, an example parameterisation is the linear combination of a set of features $f_1(x), f_2(x), ..., f_d(x)$, which becomes $F^\theta(x, \boldsymbol{\theta}) = \theta_1 f_1(x) + \theta_2 f_2(x) + ... + \theta_d f_d(x)$. By choosing

appropriate features and parameters, one can achieve $F^\theta \approx F$.

Thus, if a value function or policy is parameterised by $\boldsymbol{\theta}$, and assuming that the chosen parameter space makes it possible to approximate the value or policy sufficiently well, an RL agent can learn the elements of $\boldsymbol{\theta}$ to approximate the true value or policy. This results in a drastic reduction in the number of learned values, from infinitely many (in continuous systems) to the number of parameters in $\boldsymbol{\theta}$. Additionally, the agent is no longer required to visit every state-action pair in order to learn a good approximation of the value or policy. The combination of these traits makes function approximators suitable for solving reinforcement learning problems in continuous spaces.

2.4.1.4 Policy gradient methods

A branch of methods that use function approximators is called *policy gradient* methods. These methods learn a parameterised policy $\pi(\cdot|s, \boldsymbol{\theta})$ without checking the value function first. They are thus policy-based, or actor-only, methods, which allow the agent to sample from the continuous action space. The policy π must be parameterised such that its gradient $\nabla_\theta \pi(a|s, \boldsymbol{\theta})$ exists and is finite for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, $\boldsymbol{\theta} \in \mathbb{R}^d$.

The performance of the policy parameterisation is measured by a *score function* $J(\boldsymbol{\theta})$, which depends on the parameters. Then, we can maximise the score by gradient ascent on $\boldsymbol{\theta}$,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_\theta J(\boldsymbol{\theta}_t) \tag{2.30}$$

where α is the learning rate and $\nabla_\theta J(\boldsymbol{\theta})$ is the gradient of J with respect to $\boldsymbol{\theta}$. Usually, the score is chosen as the true value of the start state of the episode,

$$J(\boldsymbol{\theta}) \triangleq v^{\pi_\theta}(s_0) = \mathbb{E}_{a \sim \pi_\theta}\left[G_0 | S_0 = s_0\right] \tag{2.31}$$

since this gives a true measurement of the performance. However, the true value function is unknown, therefore the gradient update of Equation (2.30) must be approximated by sampling transitions in the environment. The estimated gradient should be proportional to the true gradient.

By the policy gradient theorem [35], we have the relationship of Equation

(2.32), where we have omitted the θ subscript from the ∇ operator because there is no ambiguity regarding which parameter is used in the differentiation. This notation is used from now on.

$$\nabla J(\boldsymbol{\theta}) = \nabla v^{\pi}(s_0) \propto \sum_s \rho^{\pi}(s) \sum_a q^{\pi}(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \tag{2.32}$$

Here, ρ^{π} is the *on-policy distribution under* π, which means the fraction of time spent in each state if following policy π. Then the sum over states can be replaced by an expectation under π, from which states are sampled. The sum over actions can be modified by following a similar argument, as outlined below.

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \rho^{\pi}(s) \sum_a q^{\pi}(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\
&= \mathbb{E}_{s\sim\rho^{\pi}} \left[\sum_a q^{\pi}(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \right] \\
&= \mathbb{E}_{s\sim\rho^{\pi}} \left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q^{\pi}(S_t,a) \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})} \right] \\
&= \mathbb{E}_{a\sim\pi, s\sim\rho^{\pi}} \left[q^{\pi}(S_t,A_t) \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] \\
&= \mathbb{E}_{a\sim\pi, s\sim\rho^{\pi}} \left[q^{\pi}(S_t,A_t) \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \right]
\end{aligned} \tag{2.33}$$

Then we have a formula for the policy gradient, and by estimating $q^{\pi}(s,a)$, the gradient update can be applied to the parameter vector $\boldsymbol{\theta}$. This is called the *stochastic* policy gradient and produces an unbiased estimate of the true gradient. For an intuitive understanding of this gradient, Sutton and Barto [34] give a well-formulated explanation, recounted here: look at the second to last line of Equation (2.33). The numerator of the fraction is the gradient of the probability of taking action A_t in state S_t, and the expression in the denominator gives the probability of taking that action. This means that the numerator ensures that the update changes the parameter vector $\boldsymbol{\theta}$ in the direction of highest increase in probability of taking this action on future visits to state S_t. Dividing by the probability prevents actions that occur frequently to be favoured over actions that yield better returns, yet are rarely visited. Multiplying this by the action-value makes the update largest in the direction of actions that give higher value,

i. e. are better.

A well-known policy gradient method that uses Monte Carlo sampling is called **REINFORCE** [36], in which the action-value function is estimated by the expected episode return. From (2.22), we have $Q(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a]$, and get

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \rho^\pi(s) \sum_a q^\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \mathbb{E}_{a\sim\pi, s\sim\rho^\pi} \left[G_t \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \right]. \end{aligned} \tag{2.34}$$

The procedure is outlined in Algorithm 1. An episode is sampled from start at $t = 0$ to end at $t = T - 1$ by following the policy, before any updates are made. Then, a separate return is computed from each encountered state to the end of the episode, and an update is made for each of these returns.

Algorithm 1 REINFORCE

Ensure: A differentiable policy parameterisation $\pi(a|s,\boldsymbol{\theta})$
Require: Step size $\alpha > 0$
1: Initialise policy parameter $\boldsymbol{\theta} \in \mathbb{R}^d$ (e.g to $\mathbf{0}$)
2: **for** each episode **do**
3: Generate an episode $S_0, A_0, R_0, ..., S_{T-1}, A_{T-1}, R_{T-1}$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
4: **for** each step $t = 0, 1, ..., T-1$ of the episode **do**
5: $G \leftarrow \sum_{k=t}^{T-1} \gamma^{k-t} R_k$
6: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$
7: **end for**
8: **end for**

A modification of REINFORCE is to add a *baseline* function $b(s)$ that is subtracted from the action-value. As mentioned, the policy gradient described above gives an unbiased estimate of $\nabla J(\boldsymbol{\theta})$, but the estimates can still have large variance. A bias can lead to the algorithm being unable to converge to a solution, or it can converge to a poor one. The effect of high variance, on the other hand, is the need for many samples before the algorithm can converge. Thus, both low bias and low variance is desired, and using a baseline in the policy gradient estimate can achieve lower variance while maintaining low bias [34, p. 329].

The baseline can be any function that does not depend on the action. The expected subtracted quantity then becomes zero, and the expected gradient re-

mains the same as before:

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \sum_s \rho^\pi(s) \sum_a (q^\pi(s,a) - b(s)) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \sum_s \rho^\pi(s) \left(\sum_a q^\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) - b(s) \nabla \sum_a \pi(a|s,\boldsymbol{\theta}) \right) \\ &= \sum_s \rho^\pi(s) \sum_a q^\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})\end{aligned} \tag{2.35}$$

A common choice of baseline is an estimate of the state-value function, $V(s)$. The general expression for the estimated gradient is then

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_{a\sim\pi, s\sim\rho^\pi} \left[(Q(S_t, A_t) - V(S_t)) \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \right] \\ &= \mathbb{E}_{a\sim\pi, s\sim\rho^\pi} \left[A(S_t, A_t) \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \right]\end{aligned} \tag{2.36}$$

where $A(s,a)$ is the *advantage function*, which gives a measure of how much better or worse action a in state s is compared to following the policy. The value function is a good baseline because it varies with s, giving high value in states where all actions give a high value, and opposite. Hence, the advantage, and the gradient, does not depend as heavily on where in the state space the gradient is computed.

The described modification of REINFORCE is simply called **REINFORCE with baseline**. The outline is the same as in Algorithm 1, with the addition of estimating a parameterised value function, $V(s,\boldsymbol{\phi})$. It can for example be estimated using gradient updates in a similar way as the policy gradient. An example score function is the squared advantage

$$J(\boldsymbol{\phi}) = \frac{1}{2} A(s_t, a_t, \boldsymbol{\phi})^2 = \frac{1}{2} \left(G_t - V(s_t, \boldsymbol{\phi}) \right)^2 \tag{2.37}$$

which has the gradient

$$\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) = A \nabla_{\boldsymbol{\phi}} A = -A \nabla_{\boldsymbol{\phi}} V(s, \boldsymbol{\phi}) \tag{2.38}$$

The score function based on advantage should be minimised, since nonzero advantage represents a possible improvement to the policy. Thus, gradient *descent* should be applied in this case, as $\boldsymbol{\phi} = \boldsymbol{\phi} - \alpha_\phi \nabla J(\boldsymbol{\phi})$.

The policy gradient is given by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = (G_t - V(s_t, \boldsymbol{\phi}))\nabla_{\boldsymbol{\theta}} \ln \pi(a_t|s_t, \boldsymbol{\theta}) = A\nabla_{\boldsymbol{\theta}} \ln \pi(a_t|s_t, \boldsymbol{\theta}) \tag{2.39}$$

and the parameter update equations for REINFORCE with baseline become

$$\begin{aligned} \boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \alpha_{\boldsymbol{\phi}} A \nabla V(s_t, \boldsymbol{\phi}) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma^t A \nabla \ln \pi(a_t|s_t, \boldsymbol{\theta}) \end{aligned} \tag{2.40}$$

2.4.1.5 Actor-critic methods

The term *actor-critic* covers reinforcement learning methods that combine policy and value function approximation. The policy is the *actor*, which chooses and applies actions to the environment based on the current state, while the value function is called the *critic* because it measures the performance of the policy and uses this to improve both the policy and its own estimate. The framework of such methods is illustrated in Figure 2.6.

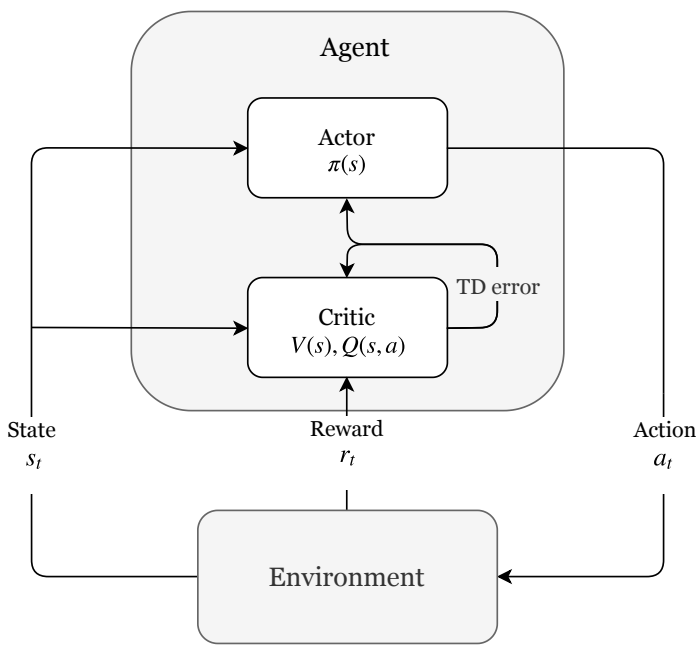


Figure 2.6: The actor-critic architecture

The REINFORCE with baseline method detailed above learns an approximated policy and state-value function, however, the value is not used as a critic. In that case, the action-value was estimated by the Monte Carlo return G_t, while the state-value approximation was only used as a baseline. Thus, REINFORCE with baseline is not considered an actor-critic method.

If the actor is parameterised by $\boldsymbol{\theta}$ and the critic is parameterised by $\boldsymbol{\phi}$, they are given by $\pi_\theta(\cdot|s, \boldsymbol{\theta})$ and $Q_\phi(s, a, \boldsymbol{\phi})$, respectively. When both functions are parameterised like this, they are both applicable when states and actions are continuous, because the approximators generalise from observed states and actions to unseen ones.

The policy parameters $\boldsymbol{\theta}$ are adjusted as previously, according to the policy gradient of Equation (2.33). In place of the true, unknown action-value q, the parameterised approximation of the critic, $Q_\phi(s, a, \boldsymbol{\phi})$, is used: $\nabla_\theta J(\boldsymbol{\theta}) = \mathbb{E}\left[Q_\phi(s, a, \boldsymbol{\phi})\nabla_\theta \ln \pi_\theta(a|s, \boldsymbol{\theta})\right]$.

The critic's parameters can be estimated by any appropriate method, and an example is temporal-difference learning. The squared TD error can be used as the critic's score function, which should be minimised, giving

$$\begin{aligned} J(\boldsymbol{\phi}) &= \frac{1}{2}\left(y_t - Q_\phi(s_t, a_t, \boldsymbol{\phi})\right)^2 \\ &= \frac{1}{2}\left(r_t + \gamma Q_\phi(s_{t+1}, a_{t+1}, \boldsymbol{\phi}) - Q_\phi(s_t, a_t, \boldsymbol{\phi})\right)^2 = \frac{1}{2}\delta_t^2 \end{aligned} \tag{2.41}$$

By ignoring y_t's dependence on $\boldsymbol{\phi}$, the gradient update rules are as follows,

$$\begin{aligned} \boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \alpha_{\boldsymbol{\phi}}\delta_t \nabla Q_\phi(s_t, a_t, \boldsymbol{\phi}) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}}\gamma^t Q_\phi(s_t, a_t, \boldsymbol{\phi})\nabla \ln \pi_\theta(a_t|s_t, \boldsymbol{\theta}) \end{aligned} \tag{2.42}$$

An off-policy deterministic actor-critic algorithm is the **deterministic policy gradient (DPG)** algorithm [37], which learns a deterministic policy $a = \mu(s, \boldsymbol{\theta})$ parameterised by $\boldsymbol{\theta}$ by following a stochastic behaviour policy $\beta(\cdot|s)$. Learning about a deterministic policy results in an intuitive handling of continuous actions - the output of the policy is simply a real value, and can thus take on any value. However, since there is no stochasticity inherent to the policy, one cannot follow this policy while learning and expect to explore the environ-

ment sufficiently. So, an off-policy approach must be used. By letting the critic approximate the action-value function, thus learning the values of all actions in each state, we enable the agent to learn about the deterministic policy while following an exploratory behaviour policy during training.

Furthermore, it has been shown that the deterministic policy gradient can be estimated more efficiently than the stochastic gradient [37], by avoiding a sum/integral over actions. By assuming a policy $a = \mu(s, \boldsymbol{\theta})$ with parameter vector $\boldsymbol{\theta}$, an action-value function $q^{\mu}(s, a)$ and a performance measure $J(\boldsymbol{\theta}) = \mathbb{E}\left[G_0 | S_0 = s\right]$, the Deterministic Policy Gradient Theorem [37] gives

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \rho^{\mu}(s) \nabla_{\theta} q^{\mu}(s, \mu(s, \boldsymbol{\theta})) \\ &= \sum_s \rho^{\mu}(s) \nabla_a q^{\mu}(s, a) \nabla_{\theta} \mu(s, \boldsymbol{\theta}) \\ &= \mathbb{E}_{s \sim \rho^{\mu}} \left[\nabla_a q^{\mu}(S_t, a) \nabla_{\theta} \mu(S_t, \boldsymbol{\theta}) | S_t = s, a = \mu(s, \boldsymbol{\theta})\right] \end{aligned} \tag{2.43}$$

Consequently, there is no need to compute the expectation over actions, and so the policy gradient of Equation (2.43) is suitable when the action space is continuous or high-dimensional.

When implementing an off-policy deterministic policy gradient, states are sampled from a behaviour policy $\beta(\cdot|s)$ rather than the deterministic policy, giving the gradient

$$\nabla J(\boldsymbol{\theta}) \approx \mathbb{E}_{s \sim \rho^{\beta}} \left[\nabla_a q^{\mu}(S_t, a) \nabla_{\theta} \mu(S_t, \boldsymbol{\theta}) | S_t = s, a = \mu(s, \boldsymbol{\theta})\right] \tag{2.44}$$

By applying an off-policy algorithm for estimating the critic, such as Q-learning or gradient temporal-difference learning, the off-policy deterministic actor-critic algorithm is completed.

These past two sections have shown that using function approximators to represent value functions and policies allows reinforcement learning to be applied to problems where action and state space may be high-dimensional and continuous. However, a major challenge in this approach is how to decide which representation, and which features, to use. The chosen representation and features must be able to provide an adequate approximation to the real function.

Nonetheless, typical function approximators such as linear or sigmoidal ones, can represent only a limited set of functions, and additionally rely on features to be hand-designed to suit specific problems [34].

Using large, nonlinear function approximators, such as deep artificial neural networks (deep ANNs, or DNNs), can mitigate some of the challenges mentioned above – they can approximate any continuous function [38], and they can learn from raw sensory input rather than relying on pre-designed features. Despite this, early results of combining temporal-difference reinforcement learning algorithms with DNNs, or any nonlinear function approximators, showed that it could cause the value of the approximator and its parameters to diverge [39]. Additionally, one cannot theoretically guarantee convergence to optimal performance [1], thus DNNs were often avoided in reinforcement learning prior to the success of Deep Q-learning [16]. This algorithm, however, has shown that DNNs can be applied in RL with good results, and the combination is called *deep reinforcement learning (DRL)*. A significant DRL algorithm will be presented in Section 2.4.3, while the following section will explain artificial neural networks more thoroughly.

2.4.2 Deep neural networks

The structure of artificial neural networks allows them to approximate nonlinear functions using a relatively small set of parameters. As we saw in Section 2.4.1, function approximation is usually necessary when applying reinforcement learning to real-valued control problems, and it has been shown that deep artificial neural networks can be used for this purpose [16, 17, 40]. This section will give some insight into the structure of ANNs and the theory behind them.

The main idea of ANNs is to use training data to learn an approximation, $y_{pred} = \tilde{g}(x)$, of a function $g(x)$. The training data consists of input-output pairs (x, y_{target}) corresponding to inputs and outputs of the real function, and the ANN adjusts its parameters so that the network's predicted output converges towards the target output provided by the training data for all inputs.

This section describes the key components of ANNs below, before providing a brief overview of the way ANNs learn in Sections 2.4.2.2 and 2.4.2.3.

2.4.2.1 Feed-forward networks

The ANNs that will be focused on in this thesis are *feed-forward* neural networks, meaning that there are no loops in the network, i.e. connections are only in one direction. A different kind of network that can also be applied in RL is called a *recurrent* neural network, where the output depends on both the current input and previous outputs. The feed-forward network structure is illustrated by Figure 2.7.

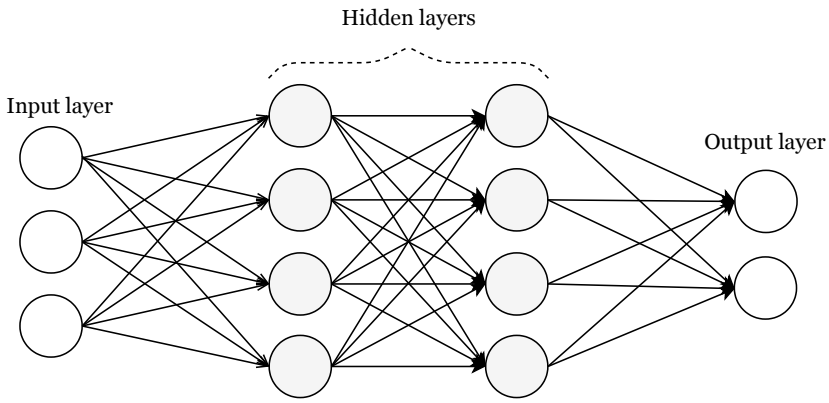


Figure 2.7: Feed-forward artificial neural network

The networks consist of neurons called *nodes*, or *units*, arranged into *layers*. The first layer, consisting of the input, is called the *input layer*, and the last layer is called the *output layer*. All layers in between are referred to as *hidden*, because the values of nodes in the hidden layers are internal to the function approximator. There exist several ways to implement the connections between layers in a feed-forward network, where one of the most commonly used are fully-connected layers. A fully-connected neural network has connections between every unit in subsequent layers, so that the value of a unit influences all the units coming after it. Other typical layer types include convolutional layers and pooling layers, used in networks that process images.

Until now, ANNs and DNNs have been mentioned somewhat interchangeably in this thesis, although there is a slight difference in their meaning. The main difference between the two terms is that an ANN refers to a function approximator inspired by networks of neurons found in humans, of the form in

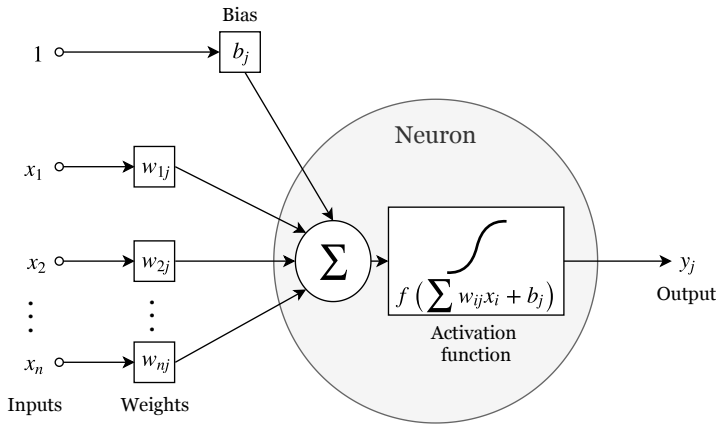


Figure 2.8: Node or unit of an artificial neural network

Figure 2.7, while a DNN is usually described as an ANN with at least one hidden layer. The more hidden layers a network has, the deeper the network is. A DNN with a single hidden layer and enough nodes can approximate any non-linear function [38], but deeper networks may simplify the approximation task for very complex functions.

The structure of units is shown in Figure 2.8. The layering structure means that the inputs of the units in a layer, denoted $\boldsymbol{x}$, are the outputs of the preceding layer's units. Each unit's output is given by an *activation function*, $y = f(\boldsymbol{x}, \boldsymbol{w}, b)$, which is a nonlinear function - usually an s-shaped (e.g. *sigmoid* or *tanh*) function or a rectified linear unit (ReLU). The activation depends on the input vector $\boldsymbol{x}$, the bias b and the parameters $\boldsymbol{w}$ in that a weighted sum of the input is computed before being passed to the activation, shown in Equation (2.45).

$$y = f(x_1w_1 + x_2w_2 + ... + x_nw_n + b) = f(\sum_{i=1}^{n} x_iw_i + b) = f(\boldsymbol{w}\boldsymbol{x} + b) \quad (2.45)$$

Thus, an entire layer can be represented by the activation function f of its units, a weight matrix $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$, containing the weight vectors and the scalar biases associated with the nodes in the layer, respectively. Then, the output of a layer is given by $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{x}$ is a vector of outputs

from the previous layer. Then an element w_{ij} in $\boldsymbol{W}$ represents the connection from node i in the previous layer to node j in the current layer, and the bias and output of the current node are b_j and y_j, respectively.

An important note is that units in the same layer use the same activation function. Additionally, all hidden layers usually share activation as well. The output layer may use a different activation to reflect the characteristics of the output, which depends on the task at hand. A $softmax$-function is often used at the output in classification tasks, to assign decimal probabilities to different classes that sum up to 1. The hyperbolic tangent, on the other hand, gives an output in the range $(-1, 1)$. One of the most popular hidden layer activations is the Rectified Linear Unit (ReLU).

2.4.2.2 Backpropagation

The training of DNNs is done by adjusting the weights of the network so that the predicted output becomes closer to the desired output for all inputs, by utilising training data consisting of reference input-output pairs (x, y_{target}). An error function typically used to measure the distance between predictions and correct values is the squared error:

$$E(y_{pred} - y_{target}) = \frac{1}{2}(y_{pred} - y_{target})^2 \tag{2.46}$$

where y_{pred} is the output predicted by the network, and y_{target} is the real output which the network's output should be equal to.

To learn the weights, two phases must be carried out, called the forward propagation and backpropagation [41] phases. The forward propagation simply passes the input of a data pair to the network, receiving a predicted output and the computed error. Backpropagation then updates the weights of the network by a gradient descent algorithm to minimise error. The gradient of the error with respect to each weight in $\boldsymbol{W}$ tells the network how the weights should be updated in order to reduce error. The update rule is give by

$$w_{ij} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}} \tag{2.47}$$

where $\alpha > 0$ is referred to as the learning rate. If $\frac{\partial E}{\partial w_{ij}} < 0$, the error goes down when the weight increases, thus the weight should be increased by the update. If $\frac{\partial E}{\partial w_{ij}} > 0$, the error goes up when the weight increases, thus the weight should be decreased by the update.

To calculate the derivatives, the backpropagation algorithm uses the chain rule to move backwards in the network, from the output to the input. Considering the weight between neurons i and j in subsequent layers, w_{ij}, we get

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} \\ &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j} y_i \\ &= \frac{\partial E}{\partial y_j} f_j'(z_j) y_i \end{aligned} \tag{2.48}$$

where y_i denotes the output of neuron i, $z_j = \sum_{k=1}^{n} w_{kj} y_k + b_j$ is the total input to neuron j, and $f_j(\cdot)$ is the activation function of neuron j. For neurons in the last layer, the first factor becomes $\frac{\partial E}{\partial y_{pred}} = (y_{target} - y_{pred})$, and this derivative is propagated backwards in the layers to calculate the rest of the derivatives. For neuron i in layer $L-1$, we can use the calculations from layer L:

$$\frac{\partial E}{\partial y_i} = \sum_{l \in L} \left(\frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l} w_{il} \right) \tag{2.49}$$

where l denotes units in layer L. Similar arguments can be made for derivatives with respect to the biases, where $\frac{\partial z_j}{\partial b_j} = 1$.

There exist several optimisation algorithms that implement the gradient updates of backpropagation, and some commonly used in neural networks are AdaGrad [42], RMSProp [43], and Adam [44], with Adam being the most recent.

A challenge that arises when training DNNs is overfitting, in which the network fits the output only to the presented training examples, but is unable to generalise when faced with new examples. Several solutions to this has been proposed, such as *dropout* [45], *L1* or *L2 weight regularisation* [46] or *early stopping*. *Batch normalisation* [47] is a technique that improves stability of neural network training, thus allowing higher learning rates and faster convergence.

2.4.2.3 Transfer learning

An inherent part of human learning is how they recognise similarities between tasks and are able to apply the relevant knowledge in order to master something new. Similarly, transfer learning in machine learning means to make use of knowledge gained when learning about one task to improve learning of a different, related, task. The approach moves the learning process of machine learning algorithms, which traditionally handle isolated tasks, towards how humans transfer knowledge from previous settings when encountering unknown tasks.

In general, transfer learning can improve the initial performance in a new task compared to starting from zero knowledge. Additionally, it can reduce the amount of time and data needed to converge to a good solution, and may sometimes give a better final performance than if transfer learning was not used. The effectiveness of the transfer learning approach depends on the relationship between the source task and target task – the more similar they are, the easier it becomes to transfer knowledge between them.

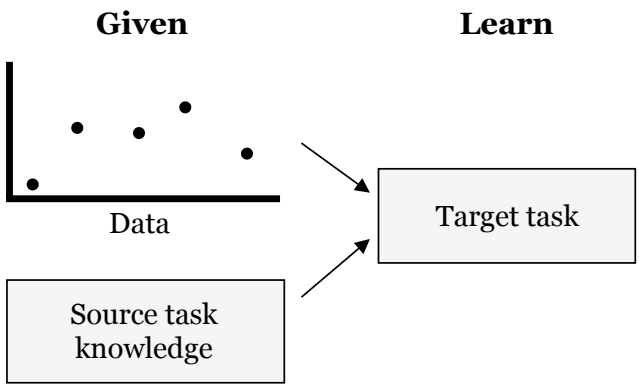


Figure 2.9: Transfer learning from source task knowledge and new data [6]

In the context of artificial neural networks, there are two common approaches to transfer learning – the final weights of the source task network can be used as initial values for the weights of the new network, or a number of the first layers of the source task network can be used as a feature extractor for the new task. In the latter case, the weights of the first layers are frozen in the new task,

meaning that they will not be affected by backpropagation. This can be useful when the available training data for the target task is limited, since restricting the number of trainable weights may reduce chances of overfitting of the network. The choice of how many layers to freeze depends on the similarity of the tasks, as earlier layers produce more general features than the later ones – thus more layers can be reused when the differences are small. In the first case, all weights may be fine-tuned during learning, which is applicable when a larger amount of training data is available for the target task.

2.4.3 Deep deterministic policy gradients

The *deep deterministic policy gradient* (DDPG) algorithm was first introduced in 2015, and has shown that the contribution of Deep Q-learning [16, 40], which could solve problems with high-dimensional state spaces, can be expanded to suit continuous action spaces [1]. The procedure is outlined in Algorithm 2.

It is an off-policy actor-critic DRL algorithm, where the policy and action-value functions are approximated using DNNs. As the name suggests, a policy gradient approach is used for estimating the policy network. The policy is deterministic rather than stochastic, and the effects of this were introduced in Section 2.4.1.5. Additionally, DDPG uses experience replay [48] to sample transitions in the environment from a replay buffer, rather than learning from transitions that were encountered in sequence and are thus temporally correlated. This is useful when using DNNs, since good approximation is usually dependent on uncorrelated training data and exploration of different parts of the state space. An experience is saved as a tuple, (s, a, r, s'), consisting of the current state, action, observed reward and the next encountered state. The experiences are uniformly sampled in mini-batches and passed to the networks when training.

The exploratory behaviour policy is computed by adding a noise process $\mathcal{N}$ to the deterministic policy. In the original DDPG algorithm [1], an Ornstein-Uhlenbeck noise process [49] was used. This noise process results in noise values centred around 0, which are also temporally correlated. The behaviour policy is given by (2.50).

$$a = \mu(s, \boldsymbol{\theta}) + \mathcal{N} \tag{2.50}$$

In order to stabilise learning of parameters in neural networks, using the network representing the critic, $Q(s, a, \boldsymbol{\phi})$, for calculating the TD target values in (2.41) and simultaneously updating the parameters of Q should be avoided. To prevent the updates from diverging, DDPG uses separate *target networks* for the critic and actor, $Q'(s, a, \boldsymbol{\phi}')$ and $\mu'(s, \boldsymbol{\theta}')$ respectively, whose weights are updated by slowly tracking the parameters of the learned networks $Q(s, a, \boldsymbol{\phi})$ and $\mu(s, \boldsymbol{\theta})$. These target updates are

$$\begin{aligned} \boldsymbol{\phi}' &\leftarrow \tau\boldsymbol{\phi} + (1-\tau)\boldsymbol{\phi}' \\ \boldsymbol{\theta}' &\leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}' \end{aligned} \tag{2.51}$$

where $\tau \ll 1$ is the rate of updates for the target networks. By reducing the rate of change of the target values of the critic in this way, the problem of learning the action-value is moved closer to a supervised learning problem, in which the backpropagation of error is computed based on an approximately stationary target y_i. This results in improved stability of learning.

DDPG is the method used for the vehicle control, path following and collision avoidance tasks in Chapters 3 and 4.

Algorithm 2 DDPG

Randomly initialise critic $Q(s, a, \boldsymbol{\phi})$ and actor $\mu(s, \boldsymbol{\theta})$ networks with weights $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$.
Initialise target network Q' and μ' with weights $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}$, $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
Initialise replay buffer R
for $episode = 1, ..., M$ **do**
    Initialise random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for $t = 1, ..., T$ **do**
        Select action $a_t = \mu(s_t, \boldsymbol{\theta}) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}, \boldsymbol{\theta}'), \boldsymbol{\phi}')$ for $i \in 1...N$
        Update critic by minimising loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i, \boldsymbol{\phi}))^2$
        Update actor policy using the sampled policy gradient:

$$\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_{a_i} Q(s_i, a_i, \boldsymbol{\phi}) \nabla_\theta \mu(s_i, \boldsymbol{\theta})$$

        Update the critic target network: $\boldsymbol{\phi}' \leftarrow \tau \boldsymbol{\phi} + (1 - \tau) \boldsymbol{\phi}'$
        Update the actor target network: $\boldsymbol{\theta}' \leftarrow \tau \boldsymbol{\theta} + (1 - \tau) \boldsymbol{\theta}'$
    end for
end for

Chapter 3

Design and implementation

The main contribution of this thesis is to investigate how deep reinforcement learning can be used in developing control systems for marine vessels, specifically path following and rudimentary collision avoidance algorithms. To achieve this, a stepwise process is followed, where the complexity of the tasks we aim to solve increases throughout the thesis. This approach was chosen due to the novelty of the ultimate goal of the work, namely designing a fully autonomous collision avoidance system using DRL – it was deemed essential to confirm the success of path following and surge control before attempting collision avoidance. In this chapter, the design decisions and implementation details related to each control problem are presented and discussed. First, Section 3.1 gives an overview of the DRL algorithm used in all experiments.

Two ship types have been used when implementing control systems, where the first is a platform supply vessel (PSV) and the second is a container type vessel. The PSV was provided as part of a simulator given by DNV GL for the work of this thesis, and is introduced in Section 3.2 below. The container was originally described by Son and Nomoto in 1981 [50], and has been ported from the Marine Systems Toolbox (MSS) [3] to Python. This means that the full description of the container's dynamics is available to us, while the same cannot be said about the PSV. Therefore, the container model is described in detail in Appendix A, while the PSV is superficially described in this chapter. For the reinforcement learning agent, there will be no apparent difference between the

two vessels, and it will treat the ship and its dynamics as a part of the environment which it must learn how to interact with. In other words, the agent has no knowledge of any of the two vessels before training is started.

The main reason for applying learning algorithms to the container vessel as well as the PSV is the time needed for simulation of the two different ship models. The PSV requires the provided simulator interface, which is computationally heavy, thus resulting in an excessive amount of time needed just to simulate the ship's movement. The container, on the other hand, is implemented in Matlab as simple equations without a graphic interface or additional modules, and therefore its computations are quick compared to the PSV. This results in less time used on simulating ship movement when using the container, and more time available for training of the neural networks.

In Section 3.3, we report on how to teach a reinforcement learning agent to control the PSV. The two tasks are surge control in Section 3.3.1 and simultaneous surge and heading control in Section 3.3.2. Implementing controllers for relatively simple tasks on the PSV have played a vital part in understanding which considerations should be made when creating a DRL agent for this particular vessel. These experiments are the first in a series of increasingly complex control problems, that will eventually allow us to design an autonomous collision avoidance agent.

Section 3.4 moves on to implementation of a path following system, and discusses the performance measure, states and actions necessary for the task. The pure path following task is also extended to include surge control. The design of these controllers is based on the works of Martinsen [2] and Martinsen and Lekkas [18], which have shown promising results when applying a proposed path following agent to different vessel models from the MSS Toolbox. The vessel types were a mariner, a tanker and a container ship. Here, an equivalent path following framework is applied to a container vessel of the same kind as in [2], and to the PSV introduced in Section 3.2, with the extension of surge control.

Finally, a collision avoidance system using DRL is outlined in Section 3.5.

3.1 DRL algorithm details

The DRL learning algorithm used in this thesis is the DDPG algorithm of Section 2.4.3 shown in Algorithm 2. The reason for choosing this algorithm is its suitability for environments where the possible state values and action values are continuous, and discretising may discard important information. Furthermore, DDPG has previously shown promise when applied to straight-line [18] and curved [19] path following tasks for marine vessels.

The algorithm consists of a policy and an action-value function, known as an actor and critic, both estimated by DNNs. The functions are implemented as fully-connected neural networks with two hidden layers of 400 and 300 units each, and all hidden layers use the ReLU function as activation, which is implemented as $\mathrm{relu}(x) = \max(0, x)$. The input to the actor is the state vector $\boldsymbol{s}$, and its output is the action vector $\boldsymbol{a}$, restricted to the range $(-1, 1)$ by a $\tanh$ activation function. To produce the necessary control actions for the system, this output is then scaled by a linear transform to the range $(\boldsymbol{a}_{min}, \boldsymbol{a}_{max})$, where $\boldsymbol{a}_{min}$ and $\boldsymbol{a}_{max}$ give the saturating limits of the actuators of the system.

The critic uses both the state and action as input, and produces a scalar action-value at the output. The state vector is an input to the first hidden layer, thus passed through both layers, while the action is added as input to the second hidden layer. The structures are outlined in Figure 3.1 below, where $Q(\boldsymbol{s}, \boldsymbol{a})$ is the action-value and $\boldsymbol{\mu}(\boldsymbol{s})$ is the output from the policy. The notation of Section 2.4.2 is used, where $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ denote weights and biases associated with layer i, and the activation function is given by $\boldsymbol{f}(\boldsymbol{x}) = \mathrm{relu}(\boldsymbol{x})$.

For learning the neural network parameters, the learning rates 10^{-4} and 10^{-3} were used in the actor and critic, respectively, with the Adam optimizer [44]. The discount factor was $\gamma = 0.99$, and the target update rate was set to $\tau = 0.001$. The maximum replay buffer size was 10^6 transitions and the parameters of the noise process utilised for exploration were $\theta = 0.15$ and $\sigma = 0.2$. These parameter choices and network structures were based on the work of Lillicrap et. al. [1].

In order to allow effective training of the neural networks, we implement linear scaling of network inputs to ensure their possible values lie within ± 1. This

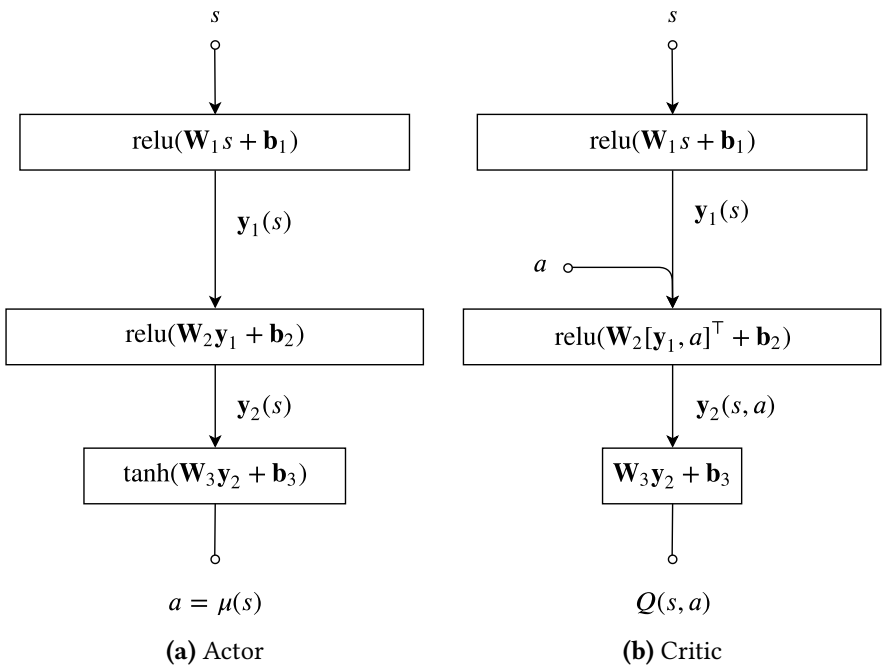


Figure 3.1: The DRL algorithm's implemented actor and critic structure

will be especially important when training DRL agents for path following tasks, where we will implement the state vector to include information about distance to the path (see Sections 3.4–3.5). This distance can be up to several thousand meters, making these state vector entries disproportionately large compared to heading, for instance, which is limited to $\pm\pi$. If standardisation of input data is not carried out, the propagation of values through the networks will result in some input features affecting the output more than others. Additionally, weights can become very large, leading to unstable learning, or very small, slowing down learning significantly.

For this to work as intended, it is crucial that the transitions saved in the replay buffer consists of these processed states and unscaled actions, so that training is performed with intended network inputs.

Training was carried out in mini-batches of size 256. The size is increased compared to 64, which was the case in [1], due to the length of episodes and na-

ture of vessels in the tasks of this thesis. Training episodes were of length 1000 or 1500 steps, and it was found that the first 64 transitions in this context were often similar to each other. By increasing the mini-batch size, more variation could be included in the first training batch.

Training of the policy and critic was performed by placing the vessel on the map with randomised position, heading and velocity, before simulating for a specified number of time steps. Simultaneously, the state, action and reward at each time step was recorded and stored in the replay buffer, and training was carried out in batches at each time step by randomly sampling the replay buffer and updating network weights through backpropagation. A time step corresponded to one second in simulation time, and one action was applied per time step, resulting in a control rate of 1 Hz. Further details concerning episode length and initial conditions will be given in Chapter 4.

3.2 The platform supply vessel simulation interface

A simulator implementing the dynamics and visualisation of a platform supply vessel (PSV) was provided for this thesis. Examples on how to communicate with the simulator were also given, and this was modified to suit the needs of the work done here. All relevant signals are read from the simulator as ground truth values, thus it is assumed that these signals can be measured accurately in the following experiments. A screen shot illustrating the graphic user interface is shown in Figure 3.2. The ship has two thrusters located aft of the ship and has length 94m and width 21m.

A heading controller and speed controller have been provided with the simulator, along with corresponding allocation of thruster forces. Thus the vessel can be controlled through two inputs - by setting the currently desired heading and surge. The speed controller includes a reference model, which smooths out the surge reference signal before feeding it to the controller. By controlling the vessel in this manner, the resulting DRL control system will not be *end-to-end*, from measurements to thruster configuration. It may, however, simplify the analysis of controller behaviour once learning is complete, on the grounds that the relationship between commands and resulting heading and surge can

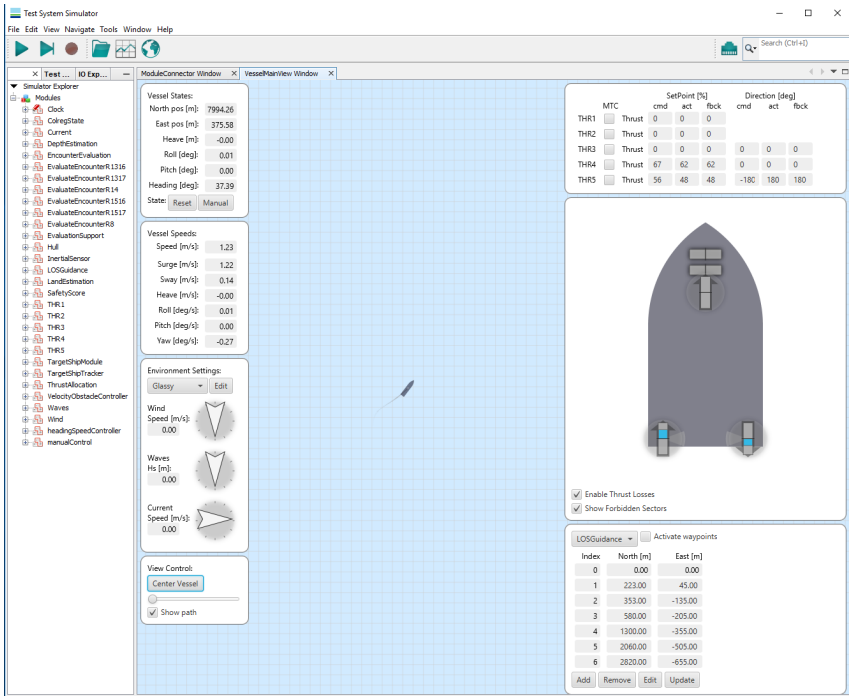


Figure 3.2: Graphic user interface of the PSV simulator

be understood more intuitively in this case.

There is an abundance of possible controllers and guidance systems that could be made available to the simulator configuration, which is one of the main reasons the use of this simulator is considered interesting for this thesis. For instance, a LOS guidance system is available, where the user needs only specify the waypoints of a path and initial conditions of the vessel. There is also the possibility of including a collision avoidance algorithm with modelling of a number of obstacles, which would make it possible to compare the DRL collision avoidance system proposed here to a trusted algorithm. Additionally, environmental forces, such as wind, current and waves are modelled and can easily be turned on or off.

The model of the PSV's dynamics are unknown to us. For the purposes of this thesis however, this is not a problem. In reinforcement learning, the ship is considered part of the unknown environment, and thus is treated as a black box. Combining this with the fact that the ship's simulator comes with controllers

that are already tuned appropriately, such as heading and surge controllers, explicit knowledge of equations of motion is redundant.

The dynamics of our ship are rather slow, and a high control rate may be needlessly complex in this case. The vessel's heading and surge responses will not be fast enough to be able to react to several substantial control input changes per second, and the result may be rapid changes in thruster configurations that yield barely noticeable change in movement of the ship. The control rate is set to 1 Hz in the following experiments. Reducing it further should be considered.

3.3 Control of a platform supply vessel using deep reinforcement learning

Even though it has been argued that knowledge of equations of motion is unnecessary from a reinforcement learning perspective, some insight into how a vessel behaves can still be helpful when designing state vectors, actions, and particularly reward signals. In order to gain this insight, this section is dedicated to the design of relatively simple control systems for the platform supply vessel.

The first case is a surge control system, laid out in Section 3.3.1, where a DRL agent learns to reach a specified surge speed through feeding reference values to a surge controller. In Section 3.3.2, heading is added to the system. From now on we will refer to these guidance systems as performing *surge control* and *surge and heading control.* Although not technically correct, this notation simplifies the discussion laid out in the thesis, since referring to a "guidance system which feeds surge and heading reference trajectories to the controllers of a PSV" can be thought of as long and winding, and takes away from the main arguments of the thesis.

The control input from the DRL agent to the vessel is a reference heading, given to the heading controller, and a reference surge speed which is fed to the speed controller. Then the action vector of our learning agent (the output of the policy network) becomes

$$\boldsymbol{a} = \begin{bmatrix} \psi_c & u_c \end{bmatrix}^\top \tag{3.1}$$

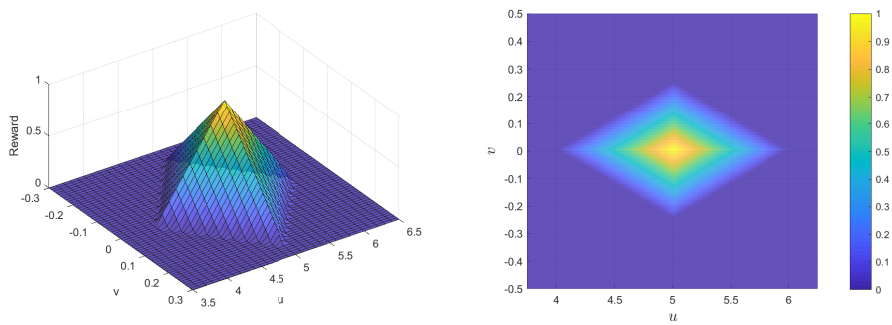
3.3.1 Surge control

The first attempt at learning to control the vessel entails learning to reach and maintain a surge velocity of u_d. The observation vector for the first experiment was chosen as in (3.2), and the corresponding reward signal is shown in Equation (3.3) and Figure 3.3, where $\tilde{u} = u - u_d$ and $\tilde{v} = v - v_d$.

$$\boldsymbol{s} = \begin{bmatrix} \psi & r & u & v \end{bmatrix}^\top \tag{3.2}$$

$$\mathcal{R}(u, v) = \max\left(0, 1 - g_u|\tilde{u}| - g_v|\tilde{v}|\right) \tag{3.3}$$

The reward is shaped as a pyramid, so that the maximum is reached when $u = u_d$ and $v = v_d = 0$. The gains define the size of the pyramid's base, and the gains used to generate the signal in Figure 3.3 were $g_u = 1$ and $g_v = 4$. The desired velocity was $[u_d, v_d] = [5, 0]$.



(a) Surface plot of reward signal as function of u and v, where $u_d = 5$ and $v_d = 0$

(b) Contour plot of reward signal as function of u and v, where $u_d = 5$ and $v_d = 0$

Figure 3.3: Shape of reward signal given by Equation (3.3)

It can often be beneficial to include more information in the reward signal, so that the performance measure becomes a weighted sum of several goals. Our main objective is to reach a surge u_d, and an additional desire is to limit "bang-bang" control, which is often unnecessary and can lead to needless wear on actuators. Our previous reward gave no indication of desired heading behaviour, so we propose a penalty based on change in ψ_c, as in (3.5). To make this objective easier to achieve, the observation vector is extended to include the previous

heading command and its derivative, shown in (3.4).

$$\boldsymbol{s} = \begin{bmatrix} \psi & r & u & v & \psi_c & \dot{\psi}_c \end{bmatrix}^\top \tag{3.4}$$

$$\mathcal{R}(u, v, \dot{\psi}_c) = \max\left(0, 1 - g_u|\tilde{u}| - g_v|\tilde{v}|\right) - c_{\dot{\psi}_c}\dot{\psi}_c^2 \tag{3.5}$$

The constant $c_{\dot{\psi}_c}$ can be seen a a weight factor, deciding the relative importance of smooth heading actions versus reaching the desired surge.

This kind of penalty term in the reward signal is not strictly necessary in order to achieve smooth control inputs. An inherent part of DRL algorithms is the randomness introduced by exploration, and due to this random process an agent may find a solution with smooth actions even though the penalty related to smoothness is omitted from reward calculations. However, such an agent will see little difference between solutions with extremely erratic actions and smooth actions, as the two cases often lead to similar vessel movement. With this in mind, one can argue that a small penalty allows the agent to differentiate between the mentioned cases, and can make sure aggressive control behaviour is discouraged.

It is also worth mentioning that the addition of ψ_c and $\dot{\psi}_c$ to the state vector is a technique used in this case to help the learning agent calculate similar actions at consecutive time steps, but is in many cases not required. It will be seen in Chapter 4 that control of the PSV using a DRL agent proved to be challenging, and this is the main reason for augmenting the state vector when including penalties.

3.3.2 Surge and heading control

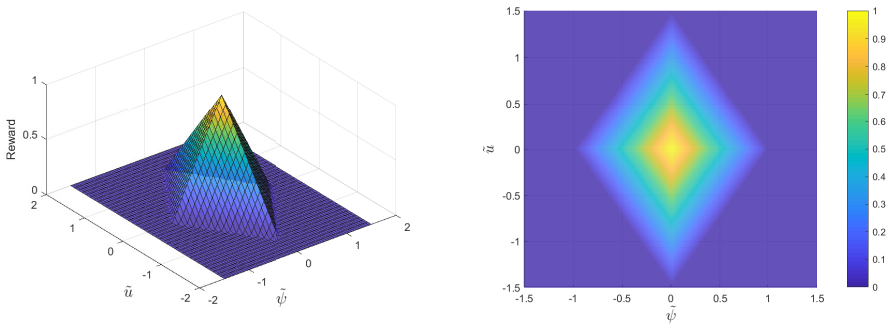
In this case, the objective is to travel at a constant forward speed of u_d, in a certain direction ψ_d. When choosing an observation vector for this problem, it could be beneficial to make sure the optimal values of the elements are the same for all choices of u_d and ψ_d. This can be implemented by introducing error terms, such as $\tilde{\psi} = \psi - \psi_d$, $\tilde{r} = r - r_d$, $\tilde{u} = u - u_d$ and $\tilde{v} = v - v_d$, which all become zero when the desired surge and heading is reached and maintained.

As long as ψ_d is constant, $r_d = \dot{\psi}_d = 0$ and $v_d = 0$ can be assumed.

$$\boldsymbol{s} = \begin{bmatrix} \tilde{\psi} & \tilde{r} & \tilde{u} & \tilde{v} \end{bmatrix}^\top = \begin{bmatrix} \psi - \psi_d & r - r_d & u - u_d & v - v_d \end{bmatrix}^\top \tag{3.6}$$

The reward signal is given by (3.7). Its shape is illustrated in Figure 3.4, where it is shown that a deviation of up to 1 rad and 1.5 m/s from desired heading and surge, respectively, gives nonzero reward. These limits were generated by setting $g_u = \frac{2}{3}$ and $g_\psi = 1$.

$$\mathcal{R}(\tilde{u}, \tilde{\psi}) = \max\left(0, 1 - g_u|\tilde{u}| - g_\psi\left|\tilde{\psi}\right|\right) \tag{3.7}$$



(a) Surface plot of reward signal as function of $\tilde{u}$ and $\tilde{\psi}$

(b) Contour plot of reward signal as function of $\tilde{u}$ and $\tilde{\psi}$

Figure 3.4: Shape of reward signal given by Equation (3.7)

By the same arguments as in Section 3.3.1, a penalty can be added to limit the occurrence of erratic control actions. For this task, it may be useful to place the penalty on change in u_c, which gives a state vector and reward signal shown below.

$$\boldsymbol{s} = \begin{bmatrix} \psi & r & u & v & u_c & \dot{u}_c \end{bmatrix}^\top \tag{3.8}$$

$$\mathcal{R}(u, v, \dot{u}_c) = \max\left(0, 1 - g_u|u - u_d| - g_v|v - v_d|\right) - c_{\dot{u}_c}\dot{u}_c^2 \tag{3.9}$$

It would have been possible to penalise $\dot{\psi}_c$ in the same way as in the surge tracking task, or even penalise both $\dot{u}_c$ and $\dot{\psi}_c$. However, it was found through experiments that the current design was most efficient.

3.4 Path following using deep reinforcement learning

As has previously been mentioned, the path following DRL controller developed in [2, 18] provides the basis for this work, and the reward function design and the elements of the state vector are the same. However, since all components have been re-implemented for this thesis, they are presented here to illustrate the reproduction of the results for the container vessel, and to confirm the applicability of the same path following system for the PSV. Furthermore, this will provide insight into design of the extended path following and surge guidance system, as well as the collision avoidance system of Section 3.5.

3.4.1 Control input

The two vessel types for which control systems have been developed in this thesis expect different kinds of control inputs. This section clarifies their characteristics, and how the action vector of each control system is designed.

3.4.1.1 Platform supply vessel

The PSV expects a control input vector of two elements, a *heading command* ψ_c and *surge command* u_c. These signals are sent to the heading and surge controller of the vessel, and have units radians and m/s, respectively. For the pure path following task, a constant surge command is sent to the surge controller to achieve constant speed, and thus only the heading ψ must be controlled in order to achieve the path following objective. This means the output of the policy network is a 1-dimensional action,

$$\boldsymbol{a} = \psi_r = \psi_c - \alpha_p \tag{3.10}$$

which is chosen so that the output of the policy is invariant to path orientation. The heading command ψ_c is obtained by adding the path angle α_p to ψ_r, and then given to the heading controller to steer the craft. This steering law is inspired by lookahead-based steering schemes, in which a course angle assignment χ_d is separated into a path-tangential angle α_p and a velocity-path relative

angle χ_r, by $\chi_d = \alpha_p + \chi_r$. The velocity-path relative angle thus defines the direction of the velocity vector of the vessel. Here, a heading angle command is given in place of course angle, thus ψ_r is the orientation of the BODY x^b-axis relative to the path. Letting the policy learn this value rather than a direct heading assignment allows it to be applied to any straight path, and thus need not be exposed to several path orientations during training. This results in faster training of the path following system.

Once surge tracking is included in the system, the policy network's output must include a surge command in addition to heading. This gives an action vector of two elements:

$$\boldsymbol{a} = [\psi_r \quad u_c]^\top = \left[\psi_c - \alpha_p \quad u_c\right]^\top \tag{3.11}$$

The heading command is extracted in the same manner as in the pure path following case, and both heading and surge commands are fed to their respective controllers as desired set-points.

3.4.1.2 Container vessel

The container vessel expects a control input vector of two elements, a *rudder angle* δ_c and *shaft speed* n_c, given in radians and rpm, respectively. Resulting rudder angle and shaft speed is described by (A.3) and (A.4) in Appendix A.

In path following, a constant shaft speed command is sent to the propeller to achieve approximately constant surge speed, and thus only the rudder angle δ must be controlled. This means the policy network output is a 1-dimensional action,

$$\boldsymbol{a} = \delta_c \tag{3.12}$$

For the path following task with surge control a shaft speed command must be included in the action vector:

$$\boldsymbol{a} = [\delta_c \quad n_c]^\top . \tag{3.13}$$

3.4.2 Performance measure

A Gaussian shaped reward signal dependent on the cross-track error is chosen as the performance measure for the path following task. It encourages path convergence, since it increases monotonically as the position error approaches zero. To ensure the vessel is travelling along the path in the desired direction, reward is given only when the vessel's yaw angle deviates from the path tangential angle by less than 90°. Equation (3.14) shows the resulting function, where c_{pf} is the amplitude, and σ decides the width (standard deviation) of the bell-shaped curve.

$$\mathcal{R}(y_e, \tilde{\psi}) = \begin{cases} c_{pf} e^{-y_e^2/2\sigma^2}, & \text{if } \left|\tilde{\psi}\right| < \frac{\pi}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

An example of this kind of reward function, with parameters $c_{pf} = 1$ and $\sigma = 30$, is illustrated in Figure 3.5. The reward signal presented here will be used as a *base reward*, which provides a foundation on which the extensions in following sections can build.

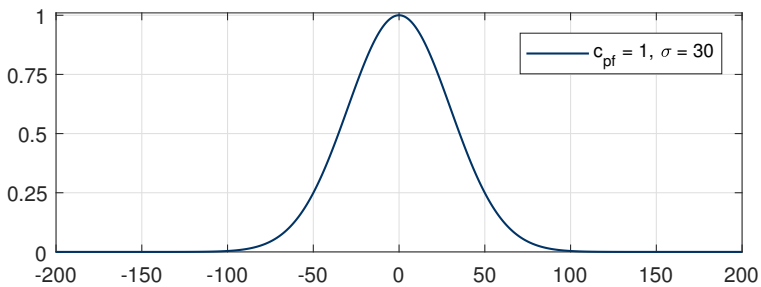


Figure 3.5: SHape of reward signal given by equation (3.14)

3.4.2.1 Additional surge control objective

Here, simultaneous path following and surge control is implemented, and therefore the reward signal must include two objectives – let cross-track error converge to zero and let surge equal a desired surge. There exists several possible ways to implement this. For instance, a positive reward may be given only when the vessel travels along the path while maintaining the correct velocity. However, this approach may restrict the agent's learning abilities due to a too sparse

reward signal. Rewards associated with each objective may be added together and weighted, equally or otherwise – this may result in a control system where only one of two goals have been reached, due to finding a local optimum of reward maximisation. Experience shows that the task of travelling at approximately correct surge speed is easier than converging to a path. This makes sense when considering the expected amount of necessary exploration in the two tasks: if one assumes constant heading, the surge control task simplifies to giving a constant input to the vessel which translates to a surge speed equal to the desired, thus little exploration is needed before this relationship can be approximated. To follow a path, on the other hand, the control input related to heading should not be constant – the vessel must make a turn towards the path in order to reach it before adjusting to the constant heading corresponding to path direction.

Motivated by the above discussion, the following reward signal for path following and surge control is suggested:

$$\mathcal{R}(y_e, \tilde{\psi}, \tilde{u}) = \mathcal{R}(y_e, \tilde{\psi}) + \begin{cases} c_u e^{-\tilde{u}^2/2\sigma_u^2}, & \text{if } \left|\tilde{\psi}\right| < \frac{\pi}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

where a Gaussian reward is added to the base reward of (3.14), but only when the vessel is travelling in the general direction of the path, with heading error within $\pm 90°$. This facilitates learning of some behaviour related to path following before giving too much velocity-related reward, possibly reducing chances of settling into a locally optimal solution. Parameters c_u and σ_u specify the amplitude and standard deviation of the curve. A depiction of the shape of this reward signal when $\left|\tilde{\psi}\right| < \pi/2$ can be found in Figure 3.6, where parameters were chosen as $c_{pf} = c_u = 0.5$, $\sigma = 30$ and $\sigma_u = 0.25$.

3.4.2.2 Counteracting action noise

In the case of the container vessel, a limitation on unnecessary changes to the rudder angle can be introduced to reduce wear on actuators and increase comfort. This can be added to the reward function as a supplementary objective, where the original reward function can be any of the ones proposed above. By

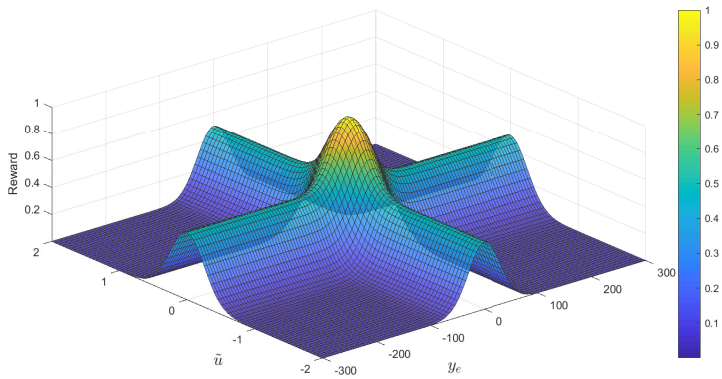


Figure 3.6: Shape of reward signal given by equation (3.15)

subtracting a small penalty $c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2$ from the proposed reward, the agent will prefer smoother control actions. This gives the following reward signal for path following:

$$\mathcal{R}(y_e, \tilde{\psi}, \dot{\delta}) = -c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2 + \mathcal{R}(y_e, \tilde{\psi}) \tag{3.16}$$

and an equivalent design when surge control is added:

$$\mathcal{R}(y_e, \tilde{\psi}, \tilde{u}, \dot{\delta}) = -c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2 + \mathcal{R}(y_e, \tilde{\psi}, \tilde{u}) \tag{3.17}$$

An alternative penalty term is needed for the PSV, due to the vessels' differing control input characteristics. However, a system with penalty for the PSV path following task will not be presented. The results laid out in Section 4.1 and 4.2.2–4.3.1 point out the difficulties of designing an appropriate performance measure for the PSV including counteraction of input noise, and it followed that a decent reward function design was not found in this case.

3.4.3 State vector

The state vector used in the path following system, with both vessel models, is

$$\begin{aligned} \boldsymbol{s} &= \begin{bmatrix} y_e & \dot{y}_e & \tilde{\psi} & \dot{\tilde{\psi}} & u & v \end{bmatrix}^\top \\ &= \begin{bmatrix} y_e & \dot{y}_e & \psi - \alpha_p & r & u & v \end{bmatrix}^\top \end{aligned} \tag{3.18}$$

where y_e is the cross-track error of the vessel and $\tilde{\psi}$ is the vessel's heading relative to the path angle α_p, called *heading error*. The cross-track error and its derivative $\dot{y}_e$ give information about the distance to the path and how fast the vessel is moving towards it, while the heading error and heading error rate $\dot{\tilde{\psi}} = r$ give a measure of the current orientation and angular velocity of the vessel. By choosing elements of the state vector with respect to the path, they become invariant to the path orientation. As discussed previously, the effect of this is faster training of the path following system, because the need to train the agent on several path orientations is eliminated.

In [2] an augmented state vector where course information is included, was tested and compared to the state vector of Equation (3.18). It was found that course information, which entails telling the agent about the direction of its velocity vector, could increase the system's robustness, especially its ability to compensate for disturbances such as ocean currents, and in the case of following a curved path. However, it was also found that the added value of extending the state vector was modest, if present, in the case of following straight-line paths with no disturbances.

In this thesis, the effect ocean currents or other disturbances may have on the performance is not investigated, and neither is curved path following. Therefore, the state vector is kept as the one shown above.

3.4.3.1 Augmentation for surge control

In order to achieve the additional objective of surge control, it can be useful to provide the DRL algorithm with supplementary information regarding surge and sway and their relation to the desired velocity, as well as knowledge about how these values change. The state vector for this task is presented below:

$$\begin{aligned} \boldsymbol{s} &= \begin{bmatrix} y_e & \dot{y}_e & \tilde{\psi} & r & \tilde{u} & v & \dot{\tilde{u}} & \dot{v} \end{bmatrix}^\top \\ &= \begin{bmatrix} y_e & \dot{y}_e & \psi - \psi_d & r & u - u_d & v & \dot{u} - \dot{u}_d & \dot{v} \end{bmatrix}^\top \end{aligned} \tag{3.19}$$

where $\tilde{u}$ is the *surge error*. It is assumed that the desired sway is always zero, therefore the error term $\tilde{v} = v$ is omitted. When desired surge is constant, the surge error derivative is reduced to $\dot{\tilde{u}} = \dot{u}$.

The additional elements describing error terms assist the algorithm in learning an optimal policy efficiently, by transforming absolute velocity measurements (u, v) into signals related to the objective before feeding them to the algorithm, thus reducing the amount of transformations to be approximated by the policy.

3.5 Collision avoidance using deep reinforcement learning

In order to achieve collision avoidance of ships, a heading control system based on deep reinforcement learning is proposed here. The container vessel [50] model is used in simulations of ship movement, and the algorithm takes in a low-dimensional vector of measurement signals and provides a commanded rudder angle to the vessel. The goal is to follow a predefined path while there is no danger of collision, and perform suitable evasive manoeuvres whenever there is a risk of colliding with a detected obstacle. The decision to control heading, and not surge, was made due to the large size of the container vessel. A ship with large mass makes changes to forward speed less desirable than heading changes, since there are significant constraints placed on forward acceleration [51]. This may also simplify the design of the DRL performance measure, as the need for an objective with respect to surge is eliminated, and consequently speed up training of the controller. Thus, the vessel's shaft speed command is held constant here. It follows that the action given by the policy network is given by (3.20).

$$\boldsymbol{a} = \delta_c \tag{3.20}$$

For this task, it is essential to include information about obstacle position and velocity in the observation vector of the DRL agent, which is shown in (3.21). The first six elements of the resulting vector are identical to the pure path following task, while four elements are augmented which describe position and motion of the obstacle with respect to the controlled vessel. The agent's own vessel position is defined in a coordinate frame aligned with the path, with origin placed at the first waypoint of the straight-line segment, and the notation

$\boldsymbol{p} = [x_e, y_e]^\top$ is used. This means $\boldsymbol{p}$ is actually the along-track and cross-track position of the vessel. The obstacle position is defined in a similar manner, as $\boldsymbol{p}_o = [x_{eo}, y_{eo}]^\top$.

$$\boldsymbol{s} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \tilde{\psi} \\ \dot{\tilde{\psi}} \\ u \\ v \\ x_{er} \\ y_{er} \\ \dot{x}_{er} \\ \dot{y}_{er} \end{bmatrix} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \psi - \alpha_p \\ r \\ u \\ v \\ x_e - x_{eo} \\ y_e - y_{eo} \\ \dot{x}_e - \dot{x}_{eo} \\ \dot{y}_e - \dot{y}_{eo} \end{bmatrix} \tag{3.21}$$

In other words, the relative x-position x_{er} and velocity $\dot{x}_{er}$ are the differences between positions and velocities along the path, called *along-track* distance and relative velocity. The *cross-track* distance y_{er} and relative velocity $\dot{y}_{er}$ are thus defined normal to the path.

The collision avoidance task includes at least three sub-goals that can be arranged according to priority, from higher to lower priority:

1. Avoid collision by keeping a safe distance d_{safe} to the obstacle at all times:

$$\|\boldsymbol{p}(t) - \boldsymbol{p}_o(t)\| \leq d_{safe} \quad \forall t \leq t_0$$

2. Converge to the desired path:

$$\lim_{t \to \infty} y_e(t) = 0$$

3. Limit abrupt changes to control input δ_c

An essential part of these goals is that in the case of a possible collision in the vicinity of the path, the first two goals contradict each other. The CA controller would in this case need to waver from the path in order to keep the safe distance to the obstacle, and for a DRL algorithm in which there is a single scalar perfor-

mance measure, learning this can be problematic and lead to unstable learning. With this in mind, a few potential design approaches will be discussed here.

A simple way to implement a DRL collision avoidance system when a framework for a successful path following system is already developed, as is the case here, is to expand that system with an additional term in the reward signal related to collision. By appropriate tuning of the weight factors in the performance measure, one can end up with a successful CA algorithm. However, a drawback is the issue of stability already mentioned, due to conflicting objectives. Additionally, a complete CA control system should behave according to COLREGS, which are not addressed by the prioritised list above, and including specific rules in a scalar reward signal may turn out to be challenging.

Another approach is to make use of the trained path following guidance system directly, by implementing the collision avoidance separately as its own path planner. This allows the complete DRL control system – which would be a combination of path planning for collision avoidance and path following – to plan a new safe path when there is risk of collision, and utilising the path following controller to follow the new path. This system should preferably plan curved paths around obstacles, and thus requires a path following system for curved paths. The currently developed path follower of this thesis is only directly applicable to straight-line paths, but can be further developed for curved paths through transfer learning [2]. By eliminating the path following objective from the CA controller itself, the inclusion of COLREGS-compliant behaviour can be made with less risk of unstable learning.

The first of these methods is implemented in this thesis, due to a combination of time limitations and the simplicity of the approach. The training procedure is outlined in Section 4.4.1.

3.5.1 Performance measure

In order to accommodate the priorities of a collision avoidance task, the performance measure must incorporate a large penalty term related to collision, in addition to the path following reward and rudder penalty terms applied previ-

ously. A simple reward signal is proposed here:

$$\mathcal{R}(y_e, \tilde{\psi}, \dot{\delta}, \boldsymbol{p}_r) = \begin{cases} -c_{collision}, & \text{if } \mathcal{S}_{collision}(\boldsymbol{p}_r) \\ -c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2 + \begin{cases} c_{pf}e^{-\frac{y_e^2}{2\sigma^2}}, & \text{if } \left|\tilde{\psi}\right| < \frac{\pi}{2} \\ 0, & \text{otherwise} \end{cases}, & \text{otherwise} \end{cases} \tag{3.22}$$

where $c_{collision}$ is a weight factor determining how heavily an unsafe distance to obstacles is penalised. To enforce the priorities listed above, this term should be much larger in magnitude than the others. In addition, a termination criteria $\mathcal{S}_{collision}$ is added to the system, shown in (3.23),

$$\mathcal{S}_{collision}(\boldsymbol{p}_r) = \begin{cases} \text{True}, & \text{if } \|\boldsymbol{p}_r\| < d_{safe} \\ \text{False}, & \text{otherwise} \end{cases} \tag{3.23}$$

which states that if $\|\boldsymbol{p}_r\| = \|\boldsymbol{p} - \boldsymbol{p}_o\| < d_{safe}$, the episode is terminated due to the vessel moving too close to an obstacle. This means that when the controlled vessel is outside an unsafe region surrounding an obstacle, reward is given according to the path following task, and otherwise a penalty of $c_{collision}$ is given before terminating.

3.5.1.1 Accommodating selected regulations

The central scenario that will be investigated in this thesis is a head-on situation. Its definition, as well as navigational rules imposed on vehicles in this situation can be found in Rule 14 in 2.3.1. Furthermore, compliance with Rule 16 (action by give-way vessel) is attempted. The reward function presented in (3.22) does not take Rule 14 into account, as it will give the same evaluative feedback to the DRL agent independently of whether it alters its course to starboard or port. Whether Rule 16 is accommodated is open for discussion due to the vague definition of appropriate action – the rule simply states that *early and substantial action* should be taken, and the vessel shall *keep well clear [of the other vessel]*. Thus it can be argued that by appropriately large selection of d_{safe}, the above proposed reward signal will result in following this rule.

However, by modifying (3.22) with additional penalty within a certain shape surrounding the obstacle and its unsafe circular area, it may be possible to achieve *earlier* and/or *more substantial* actions. The shape of this area of penalty can dictate which trajectory the agent chooses when avoiding obstacles. For instance, if the goal is conforming to Rule 14 it may be designed as a an ellipse or rectangle stretching out on the port side of the obstacle (from the viewpoint of the controlled vessel), which would encourage passing the obstacle on starboard side.

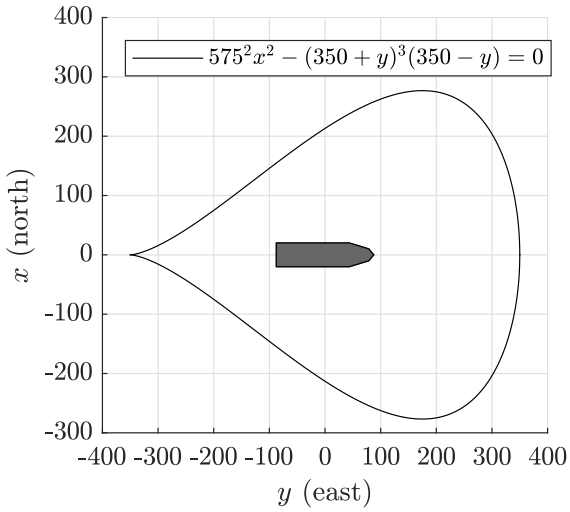


Figure 3.7: Teardrop-shaped penalty region for encouraging substantial actions in collision avoidance

A suggested region of penalty has the shape shown in Figure 3.7. This choice is motivated by creating a wide area ahead of the obstacle which is more expensive in terms of penalty, therefore encouraging the agent to take earlier action to avoid this region. The corresponding reward function is given by (3.24),

$$\mathcal{R}(y_e, \tilde{\psi}, \dot{\delta}, \boldsymbol{p}_r) = \begin{cases} -c_{collision}, & \text{if} \quad \mathcal{S}_{collision}(\boldsymbol{p}_r) \\ -c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2 + \begin{cases} -c_{16}, & \text{if } \mathcal{P}_{16}(\boldsymbol{p}_r) \leq 0 \\ c_{pf}e^{-\frac{y_e^2}{2\sigma^2}}, & \text{if } \left|\tilde{\psi}\right| < \frac{\pi}{2} \end{cases} \end{cases} \tag{3.24}$$

where the teardrop-shaped region is defined by the function $\mathcal{P}_{16}(\boldsymbol{p}_r) = 575^2 x_{er}^2 -$

$(350 + y_{er})^3(350 - y_{er}) \leq 0$. The penalty inside the region of $\mathcal{P}_{16}$ is given by a weight factor c_{16}, named after the rule in COLREGS which inspired it. The same termination criteria as before is used.

An important aspect of the kind of obstacle expansion performed here is that entering the expanded region should result in penalty, but not termination of the episode. In other words, entering the penalty-region should not be prohibited, merely discouraged. This suits the regulations defined by COLREGS, because the reward function design will result in an agent following the rules we have imposed unless an emergency situation arises – when the only collision free trajectory violates the rules, it must be possible to choose this trajectory over collision.

3.6 Overview of controllers

Table 3.1 shows a summary of the implemented control structures which have been described in this chapter. The state vectors, actions, reward functions and vessel types are included, as well as references to corresponding implementation details (under *Info*) and results. Penalties are omitted from reward functions in the table for simplicity.

Controller	State	Reward function	Vessel	Action	Info	Results
Surge	$[\psi, r, u, v]^\top$	$\max(0, 1 - g_u\lvert\tilde{u}\rvert - g_v\lvert\tilde{v}\rvert)$	PSV	$\begin{bmatrix}\psi_c \\ u_c\end{bmatrix}$	3.3.1	4.1.2
Heading and surge	$\left[\tilde{\psi}, \tilde{r}, \tilde{u}, \tilde{v}\right]^\top$	$\max(0, 1 - g_u\lvert\tilde{u}\rvert - g_v\left\lvert\tilde{\psi}\right\rvert)$	PSV	$\begin{bmatrix}\psi_c \\ u_c\end{bmatrix}$	3.3.2	4.1.3
Path follow	$\left[y_e, \dot{y}_e, \tilde{\psi}, r, u, v\right]^\top$	$c_{pf}e^{-y_e^2/2\sigma^2}$ if $\left\lvert\tilde{\psi}\right\rvert < \frac{\pi}{2}$	PSV Container	ψ_r δ_c	3.4 3.4	4.2.2 4.2.3
Path follow and surge	$\left[y_e, \dot{y}_e, \tilde{\psi}, r, \tilde{u}, v, \dot{\tilde{u}}, \dot{v}\right]^\top$	$c_{pf}e^{-y_e^2/2\sigma^2}$ $+c_u e^{-\tilde{u}^2/2\sigma_u^2}$ if $\left\lvert\tilde{\psi}\right\rvert < \frac{\pi}{2}$	PSV Container	$\begin{bmatrix}\psi_r \\ u_c\end{bmatrix}$ $\begin{bmatrix}\delta_c \\ n_c\end{bmatrix}$	3.4 3.4	4.3.1 4.3.3
Collision avoidance	$\left[y_e, \dot{y}_e, \tilde{\psi}, r, u, v, x_{er}, y_{er}, \dot{x}_{er}, \dot{y}_{er}\right]^\top$	$\begin{cases} -c_{collision} \text{ if } \mathcal{S}_{collision}(\boldsymbol{p}_r) \\ c_{pf}e^{-\frac{y_e^2}{2\sigma^2}} \text{ if } \left\lvert\tilde{\psi}\right\rvert < \frac{\pi}{2} \end{cases}$ $\begin{cases} -c_{collision} \text{ if } \mathcal{S}_{collision}(\boldsymbol{p}_r) \\ \begin{cases} -c_{16} \text{ if } \mathcal{P}_{16}(\boldsymbol{p}_r) \leq 0 \\ c_{pf}e^{-\frac{y_e^2}{2\sigma^2}} \text{ if } \left\lvert\tilde{\psi}\right\rvert < \frac{\pi}{2} \end{cases} \end{cases}$	Container Container	δ_c δ_c	3.5 3.5	4.4.2 4.4.3

Table 3.1: Overview of DRL control systems

Chapter 4

Simulations

In this chapter, results of the proposed deep reinforcement learning control systems of Chapter 3 are presented and analysed. For each system, specifications of parameters used in training are documented, and representative simulation results are presented and discussed. The final section provides an overview of the findings.

4.1 Control of a platform supply vessel (PSV)

In these case studies, the DRL agent acts as the guidance system of a PSV by feeding a surge command u_c and a heading command ψ_c to the surge and heading controllers of the PSV. The goal is to learn to choose these values so that the actual surge and heading converge to desired values, u_d and ψ_d. In the first case, only a desired surge is chosen as $u_d = 5m/s$, whereas in the second case both a desired surge and desired heading are specified, as $u_d = 5m/s$ and $\psi_d = 45°$, respectively. In other words, only the first case allows heading to drift.

This iterative method, where objectives are added to the guidance system one by one, was mainly used because the PSV behaviour was unknown beforehand and it was necessary to confirm that straightforward tasks could be performed using DRL in this vessel. If it had been attempted to learn a complicated task first, it would have been more difficult to find errors in the implementation. Simulations of these case studies are included in this chapter because they

were applied to a vessel for which DRL had never been tried, and the results therefore give relevant insight, both to specific characteristics of the PSV, and to issues that may arise when implementing DRL algorithms.

4.1.1 Training

Training was carried out using the DDPG algorithm with specifications given in Section 3.1. In both case studies, the vessel was placed on the map with random initial position and heading, and the episode length was 1000s. The initial surge was sampled from a uniform distribution in the range $u_{init} \in [2, 6]$ m/s. Actions were limited to the values $\psi_c \in [-\pi, \pi]$ and $u_c \in [0, 6]$. Parameters used in the reward functions for each of the case studies are given in Appendix B, Table B.1.

4.1.2 Surge control

Section 3.3.1 saw a description of two different state vectors and reward signals for the surge control task – one where there is no limitations on the behaviour of heading, found in (3.2)–(3.3), and another where we have added a penalty related to change in inputs to the heading controller, see (3.4)–(3.5). DRL agents

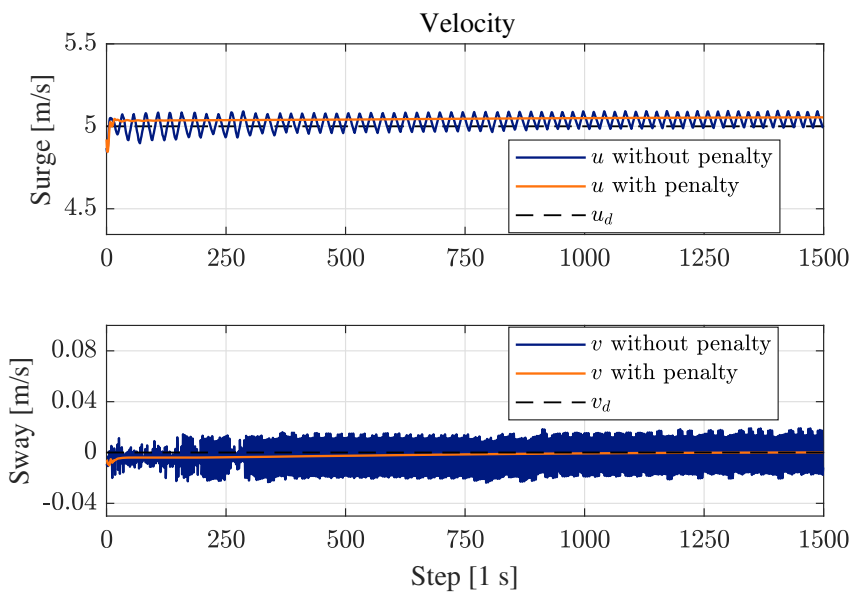


Figure 4.1: Velocities of a DRL agent performing surge control on a PSV

for both variants have been trained and the results are shown here.

With a desired surge velocity of $u_d = 5$ m/s, we get the velocities for the ship shown in Figure 4.1. The blue lines represent the system where the reward does not include any penalty term for $\dot{\psi}_c$, while the orange lines are from the system with penalty. It is clear that the penalty term reduces fluctuations in the system. Even though the penalty is placed on heading command, one can expect that smoother directional commands affects the behaviour of surge and sway due to the coupling between heading and velocities.

The effect of a penalty term in the reward signal is most visible in the heading. As seen in Figure 4.2, the heading command sent to the controller is very erratic when there exists no penalty to prevent such behaviour. When there is penalty, on the other hand, the results show a slowly drifting commanded heading, demonstrating the significant reduction of erratic behaviour by the modified reward signal.

Trajectories and surge responses can be found in Figures 4.3 and 4.4. The trajectories of the two systems are similar to each other, but the result of the $\dot{\psi}_c$-penalty is evident when examining commanded and resulting surge.

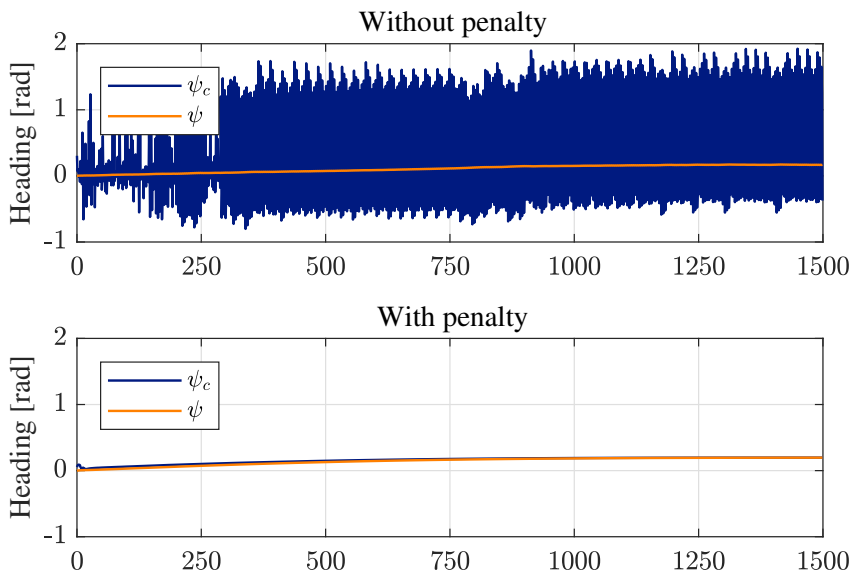


Figure 4.2: Heading commands and responses of a DRL agent performing surge control on a PSV

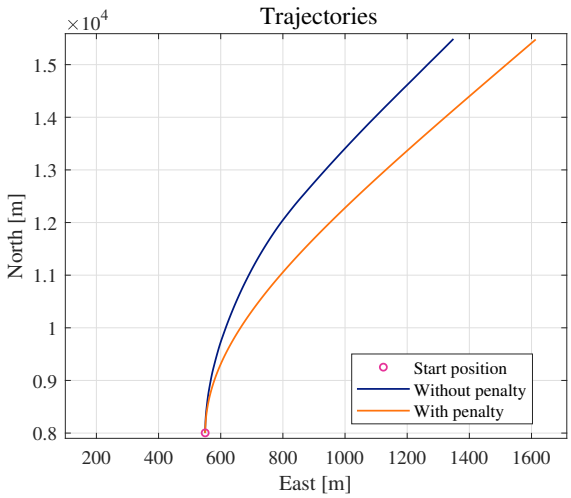


Figure 4.3: Trajectories of a DRL agent performing surge control on a PSV

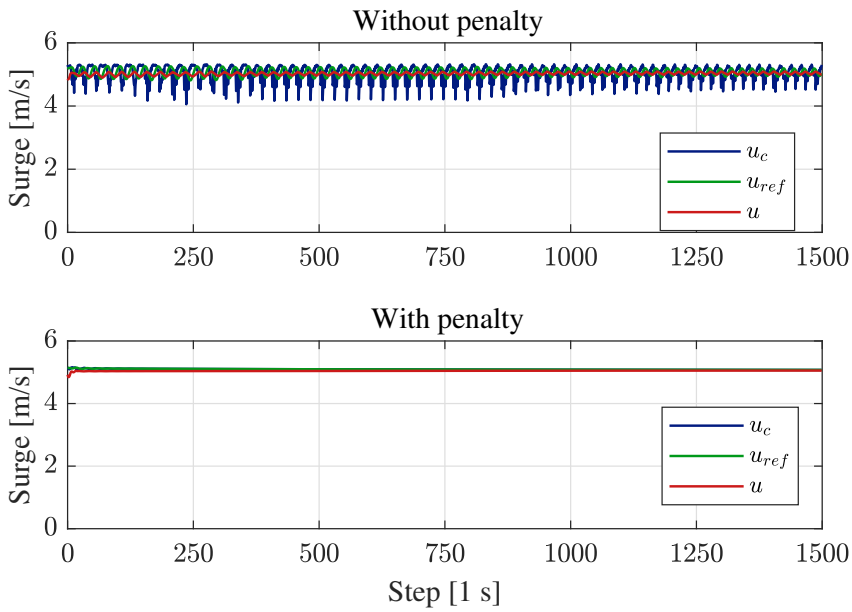


Figure 4.4: Surge commands and responses of a DRL agent performing surge control on a PSV

The figures include the surge command given by the agent, u_c, the real input to the surge controller generated by a reference model, and the resulting surge u. The bottom plot in Figure 4.4 depicts a smooth surge command, which results in

a nice-looking surge response very close to the desired u_d, while in the topmost plot both u_c and u oscillate in the proximity of u_d.

4.1.3 Surge and heading control

With desired surge and heading set to $u_d = 5$ m/s and $\psi_d = 45°$, the results presented in Figures 4.5–4.10 were obtained. Commanded surge and its response can be found in Figure 4.6, while velocities in surge and sway are shown together in Figure 4.8. Heading command and response is presented in Figure 4.9, and resulting trajectories in Figure 4.5. All figures present the results of two different reward signals – without and with penalties on change in control input, given by (3.7) and (3.9), respectively.

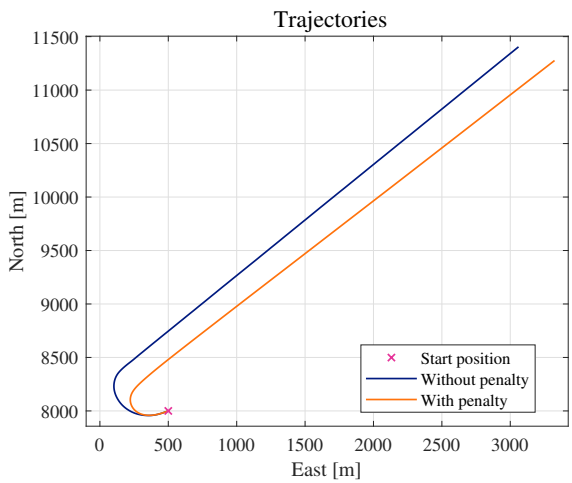


Figure 4.5: Trajectories of a DRL agent performing surge and heading control on a PSV

When comparing results with and without added penalty, it is found that both systems fulfil their main objective approximately equally well – they converge to $\psi \approx \psi_d$ and $u \approx u_c$. The most apparent difference is the behaviour of u_c and how this affects ψ_c and sway v. Figure 4.7 shows a comparison of surge commands u_c in the cases with and without penalty. It illustrates how the surge commands differ in the two agents – when penalty is added, consecutive changes to u_c become slightly smaller, and the shape of the surge command appears similar to a periodic signal.

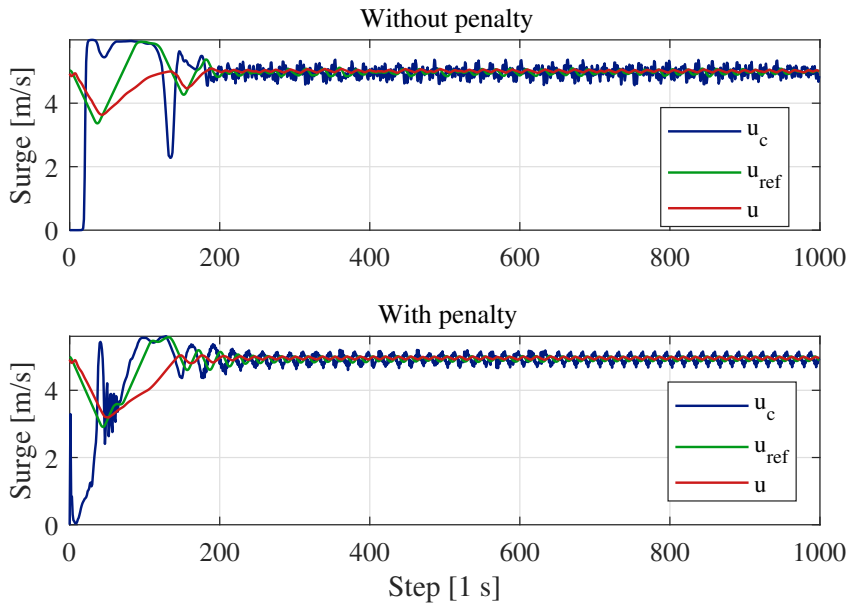


Figure 4.6: Surge commands and responses of a DRL agent performing surge and heading control on a PSV

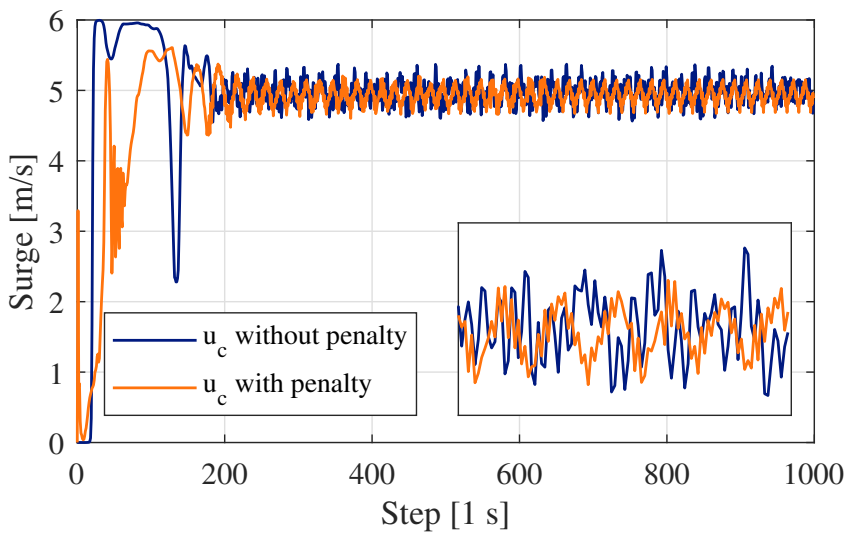


Figure 4.7: Comparison of u_c with and without penalty proportional to $\dot{u}_c$, of a DRL agent performing surge and heading control on a PSV

The effect of this is not obvious in the surge response, but one can see a clear improvement to sway and heading. The commanded heading becomes slightly smoother, and this in turn results in less adjustment to orientation of the vessel. This again leads to less sway velocity, due to the fact that sway is induced when a marine craft is turning. However, the added penalty has not resulted in completely smooth control inputs, and oscillations are still present in the resulting u. Tuning the weight factors, as well as general design of the reward signal, was found to be challenging in the tasks where the DRL algorithm controlled heading and surge simultaneously, resulting in the presented results being the best obtained solutions.

A significant remark that must be made in this context is that the thruster allocation method of the PSV is unknown to both the DRL algorithm and to the author of this thesis. This means that explaining the coupling between desired and actual surge, sway and yaw is challenging, and that there may exist some combination of actions which result in contradictory behaviour that the algorithm is unable to approximate well.

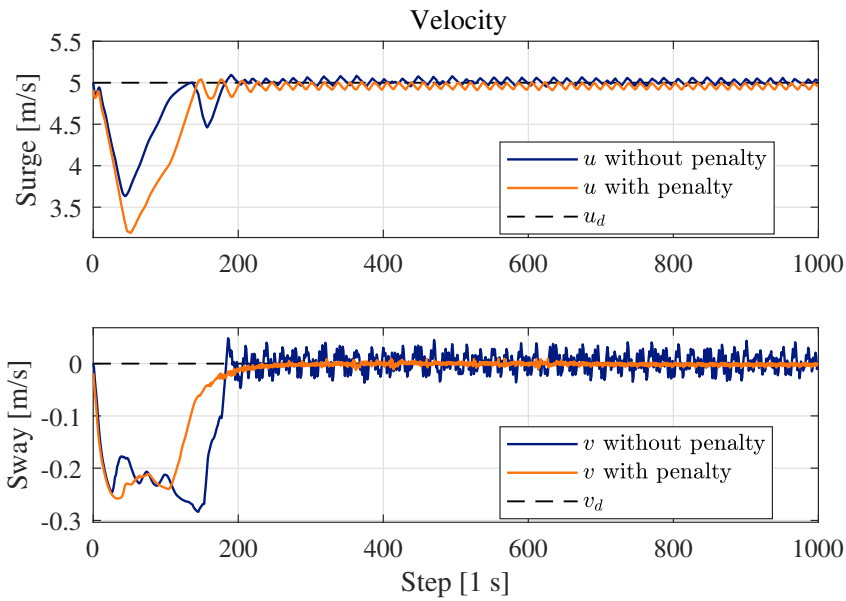


Figure 4.8: Velocities of a DRL agent performing surge and heading control on a PSV

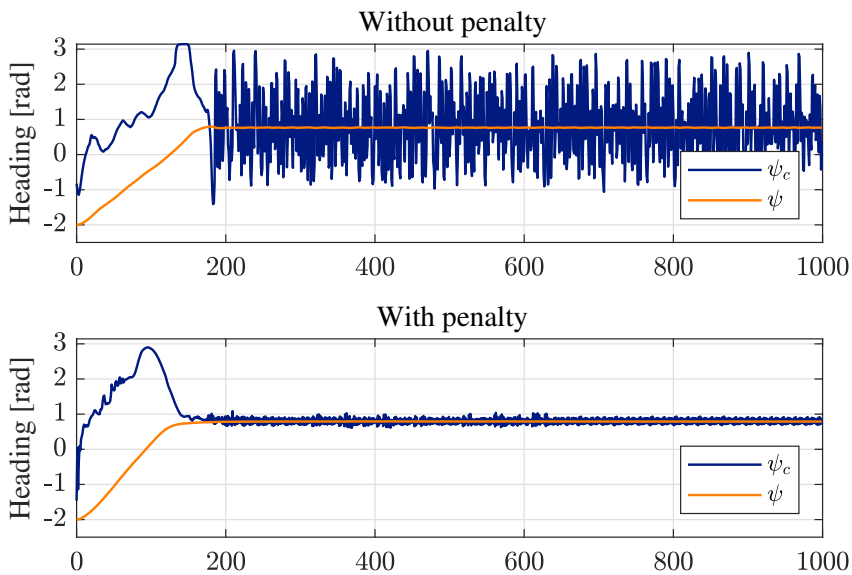


Figure 4.9: Heading commands and responses of a DRL agent performing surge and heading control on a PSV

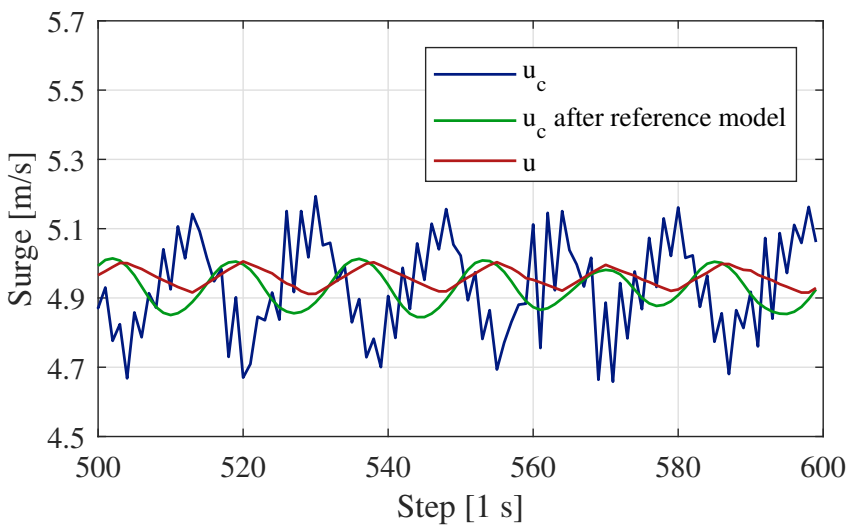


Figure 4.10: Close-up of surge command and response of a DRL agent performing surge and heading control on a PSV (with penalty proportional to $\dot{u}_c$)

A concern is raised regarding the response time in forward speed – from u_c to u. The PSV's surge controller includes a reference model serving as a filter for the surge commands, which delays the response in surge significantly. A close-up of this effect is shown in Figure 4.10, which exhibits a delay of close to 10 seconds between the time u_c begins to increase until u increases. When creating a DRL controller, this could result in considerably delayed rewards: the controller must wait too long before receiving a reward for its action, making it challenging to assign credit to the correct action.

These complications will characterise the PSV simulations of later sections as well.

4.2 Path following

Results from the path following task in two vessel models is presented here, where simulations of the PSV and container vessel are shown in Section 4.2.2 and 4.2.3, respectively. The container vessel results are included for the sake of comparison, although similar results were obtained and discussed in [2, 18]. For the PSV, a constant surge command $u_c = 5$ m/s is given to the controller, and a similar surge speed of $u \approx 5$ m/s is achieved in the container by letting the shaft speed be $n_c = 50$ rpm. The path is then followed by adjusting vessel heading: the PSV uses heading commands ψ_c and the container uses rudder angle δ_c for this.

Waypoint	Position (x, y)
1	$(0, 600)$
2	$(3880, -4680)$
3	$(2248, -7662)$
4	$(3878, -10644)$
5	$(2246, -13626)$

(a) For the PSV

Waypoint	Position (x, y)
1	$(0, 600)$
2	$(3880, -6680)$
3	$(2248, -10662)$
4	$(3878, -14644)$
5	$(2246, -20626)$

(b) For the container

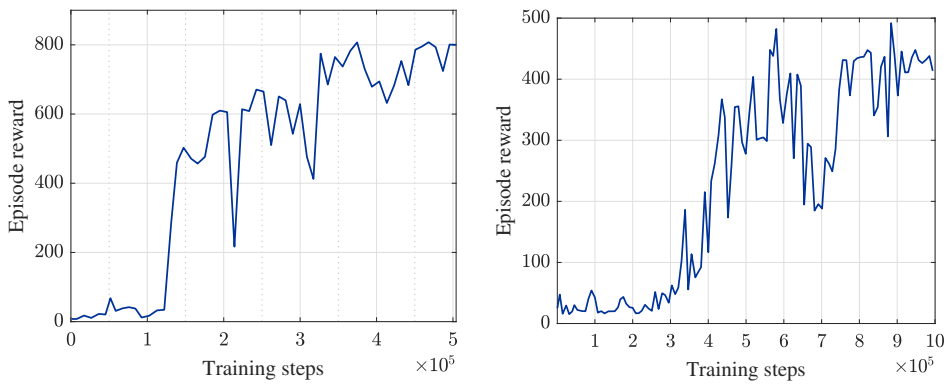
Table 4.1: Waypoints used for generating test paths

The paths used to obtain results presented in this section consist of the waypoints in Table 4.1. The path for the container vessel is more stretched out than

for the PSV to accomodate differences in the vessels' maximum turning rates.

4.2.1 Training

Training was performed by placing the vessel in a random position within 500m of the path, with random heading. For the container vessel, the initial value of the rudder angle was in the range $\delta_{init} \in [-2°, 2°]$. The maximum episode length was 1000s, and an upper limit on the cross-track error was used to reduce the size of the state space. The limit was $y_{e,max} = 2000$m. The action space for the PSV was defined by $\psi_r \in [-\pi, \pi]$ and the action space for the container was $\delta_c \in [-10°, 10°]$.



(a) PSV **(b)** Container

Figure 4.11: Reward history for the path following task

Evaluation of the trained agent was performed every 10 episodes by turning off exploration and running the agent for a full episode from a representative set of initial conditions, and then averaging the results. Training progress when using the PSV and container can be found in Figure 4.11. The agents converge to a good policy after about 370, 000 training steps for the PSV, and about 600, 000 steps for the container. Differences in training time and achieved accumulative reward between the vessels is likely due to the limited turning rate of the container vessel, meaning that the PSV is able to turn towards, and consequently reach the path, faster than the container, which in turn results in higher total reward. It should be pointed out that training of the PSV was the slowest measured in *time* spent, despite needing less training steps than the container. This was

discussed in Section 3.2 as one of the reasons the container had to be included in this thesis.

Parameters used in the reward functions for each agent are given in Appendix B, Tables B.2 and B.3.

4.2.2 Platform supply vessel simulations

Figure 4.12: Trajectory of two DRL agents performing path following on a PSV

Two solutions are presented in order to illustrate some of the challenges one is faced with when training DRL algorithms. For this vessel, a satisfactory reward signal designed to counteract unpredictable action noise was not found, hence these results were obtained using the base reward of Equation (3.14). Some discussion related to difficulties in tuning of penalty terms for the platform supply vessel was laid out in Section 4.1.3.

It has been pointed out that a penalty in the reward signal placed on the derivative of actions is not the only way to achieve smooth control inputs from a DRL algorithm. A learning agent may find such a solution without the penalty terms, and the main reason for including these terms is thus to make it easier for the agent to differentiate between solutions. Here, a solution with erratic actions (referred to as *case 1*) is shown and compared to a similar solution with smooth actions (*case 2*), and it is shown that they receive approximately the

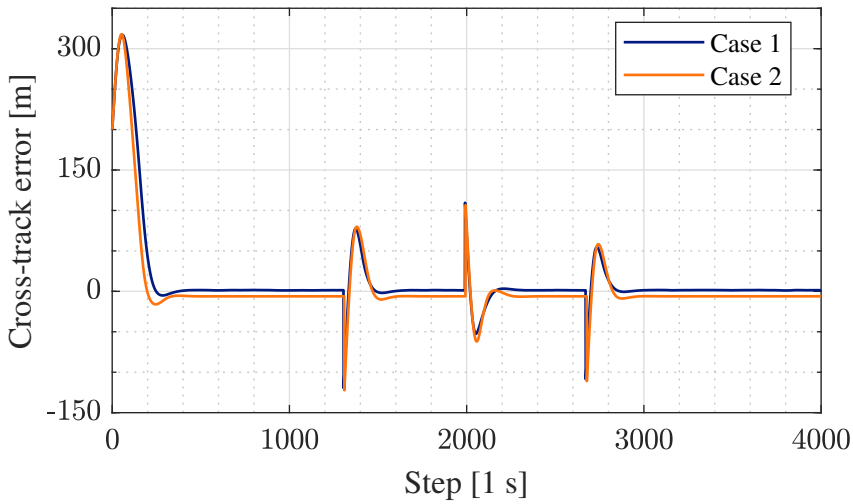


Figure 4.13: Cross-track error of two DRL agents performing path following on a PSV

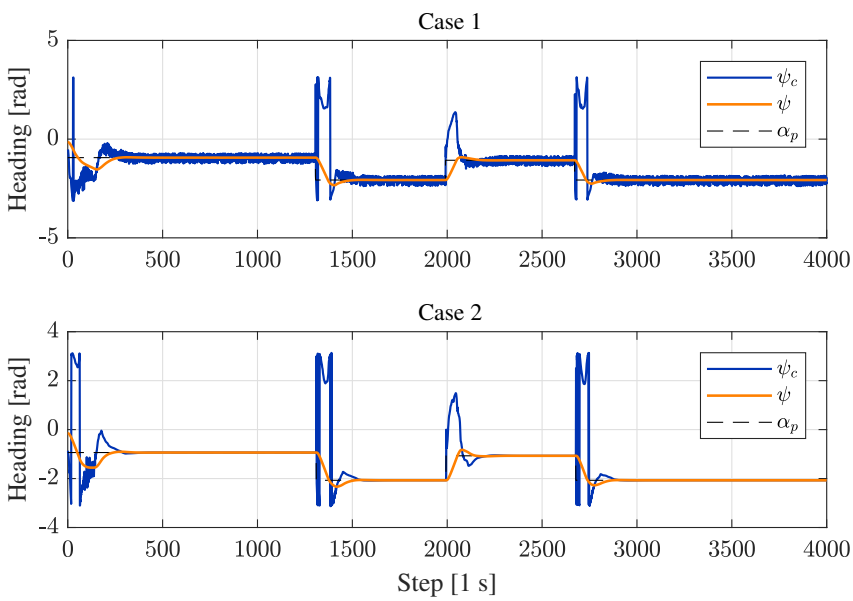


Figure 4.14: Heading command and response of two DRL agents performing path following on a PSV

same reward.

Figure 4.12 clearly shows that the craft is able to follow the provided path in both cases. Examining Figure 4.13, however, it is evident that the cross-track error in case 1 is closer to zero than in case 2, meaning that the agent whose actions are *not* smooth is objectively better at solving the path following task. This is one of the reasons tuning of weight factors in a reward function can be challenging – one must make sure the importance of smooth control inputs is high enough to prefer case 2 over case 1, while also being low enough to encourage reaching the path in the first place. As shown in Figure 4.14, the change in control input is significant when the vessel must change its orientation to follow the path, which illustrates that considerable control input changes should not always be discouraged.

Figures of thruster forces and orientations for the two solutions are placed in Appendix C. They are included to show the consequences of unnecessarily frequent control input adjustments, which subjects thrusters to needless wear and tear, possibly reducing their lifespan.

4.2.3 Container vessel simulations

The container simulations shown in this section were obtained using the reward with penalty for counteraction of rudder noise, of Equation (3.16), with $c_{\dot{\delta}\dot{\delta}} = 10$. The trajectory of the craft as it attempts to follow a given path is shown in Figure 4.15, Figure 4.16 shows cross-track error, and the the rudder input along with resulting rudder angle and heading of the vessel can be found in Figure 4.17.

The controller is able to converge to a small cross-track error by adjusting its rudder angle appropriately and without an unreasonable amount of changes to rudder input. However, the steady-state value of y_e is nonzero, as seen in Figure 4.16, so convergence to the path is not achieved. This was addressed in [2, 18], where compensation for this effect was achieved by estimating the steady-state error as $\hat{y}_{e,ss}$ using integration of the cross-track error, and then replacing y_e in the state vector with $y_e + \hat{y}_{e,ss}$. This was applied to a trained policy, i.e. not implemented during training, and anti-windup action was applied in the integration strategy. The result was path convergence. This steady-state error compensation strategy was not applied here, yet it can be expected

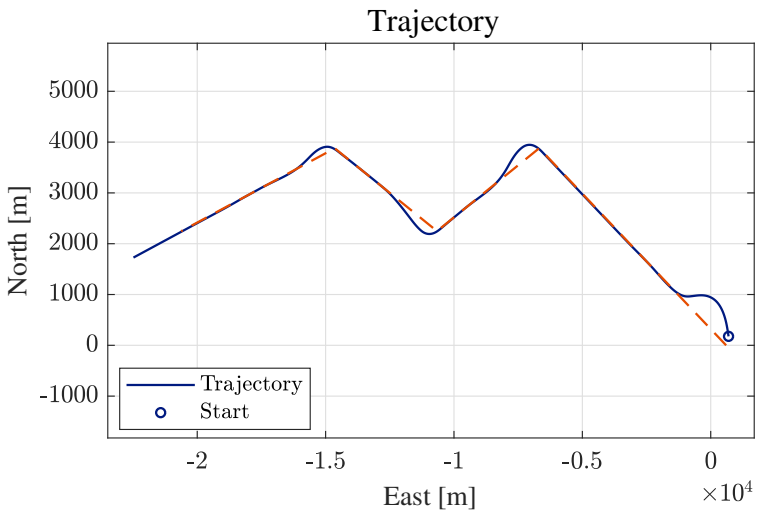


Figure 4.15: Trajectory of a DRL agent performing path following on a container vessel

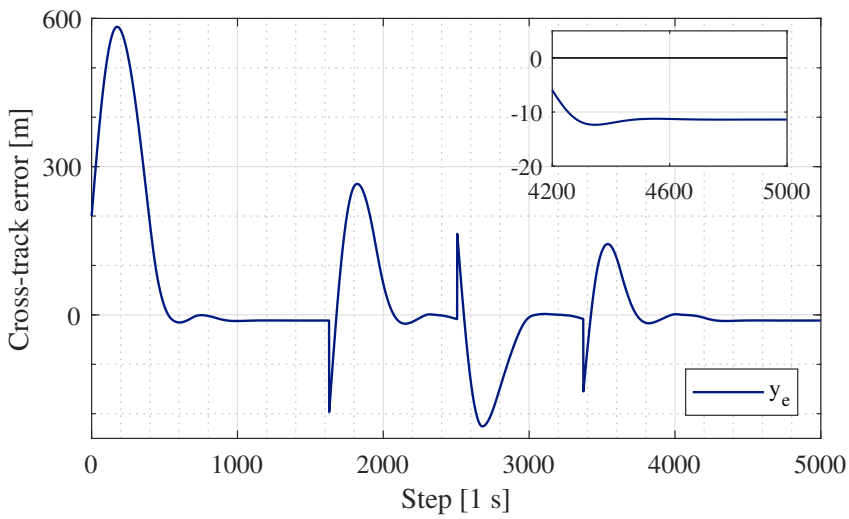


Figure 4.16: Cross-track error of a DRL agent performing path following on a container vessel

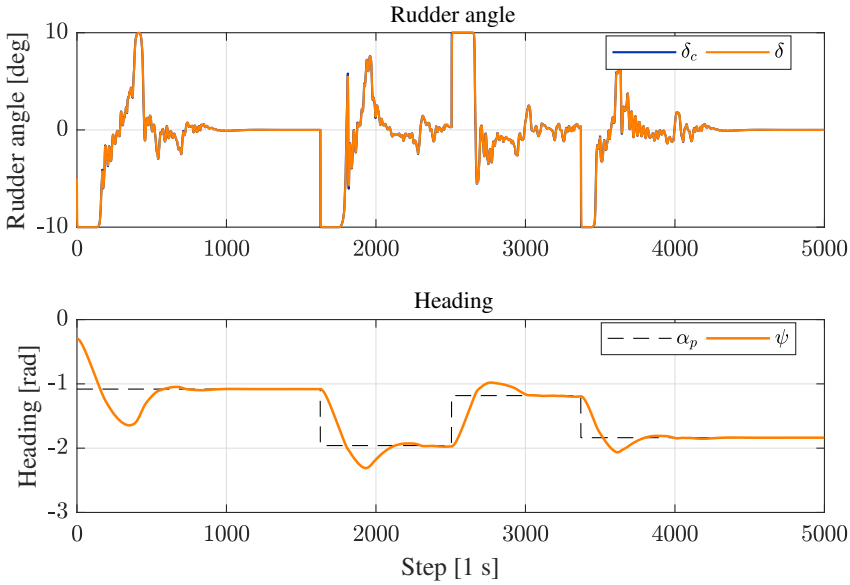


Figure 4.17: Rudder input and heading of a DRL agent performing path following on a container vessel

to have the desired effect due to the DRL algorithm designs being equivalent. The phenomenon where the DRL controller does not accomplish path convergence, even though the Gaussian reward function has its maximum at $y_e = 0$, is typical in DRL algorithms – their random nature of exploration usually result in converging to a sub-optimal solution by virtue of being unable to visit the full state space, as well as possibly using too high learning rates.

Comparing the trajectory of Figure 4.15 with Figure 4.12, it can be seen that the PSV is able to perform significantly sharper turning manoeuvres than the container. This is mainly due to the limitation on rudder angle for the container, which ensures δ is less than 10°, whereas the PSV does not include the same kind of restriction – it can position its thrusters sideways to achieve higher turning rate.

4.3 Path following and surge control

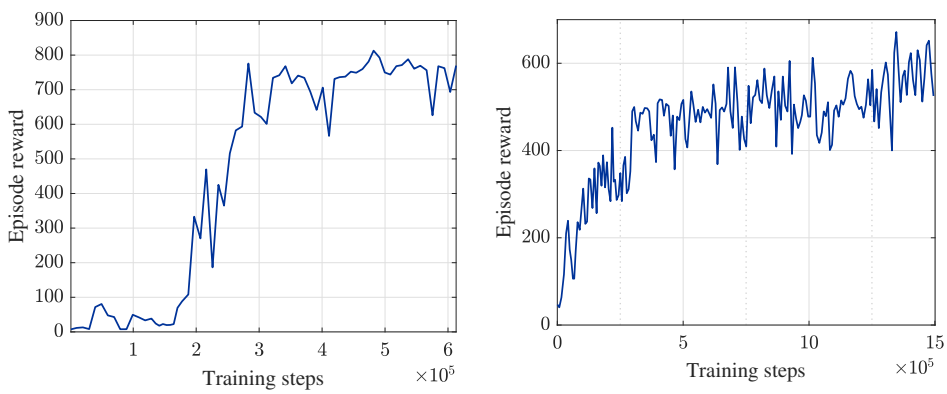
The following section presents results from simultaneous path following and surge control in the two vessel models, the PSV (Section 4.3.1) and container (Section 4.3.3). A path is followed by adjusting heading in the same manner as before, while a desired surge is held by adjusting surge command u_c in the PSV, and shaft speed n_c in the container. Desired surge was chosen as $u_d = 5$ m/s.

The simulation results of this task were obtained using the same test paths as in the path following task. The waypoints can be found in Table 4.1.

4.3.1 Training

Training was executed in the same way as laid out in Section 4.2.1, except that a randomly initialised surge within $u_{init} \in [u_d - 2, u_d + 2]$ m/s was implemented. The action limits in the case of the PSV were $\psi_r = [-\pi, \pi]$ and $u_c = [0, 10]$ m/s. In the container, actions were constrained to $\delta_c = [-10°, 10°]$ and $n_c = [0, 100]$ rpm.

Evaluations were performed every 10th episode, and training progress when using the PSV and container can be found in Figure 4.18. In this task, the agents converged to a good policy after about $500,000$ training steps for the PSV, and about $1,300,000$ steps for the container. Parameters used in the reward functions for each agent are given in Appendix B, Tables B.2 and B.3.



(a) PSV

(b) Container

Figure 4.18: Reward history for the path following and surge control task

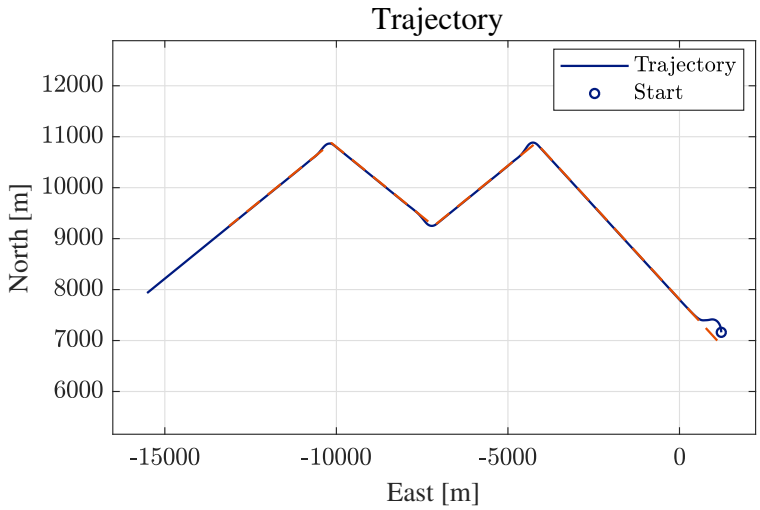


Figure 4.19: Trajectory of DRL agent performing path following with surge control on a PSV

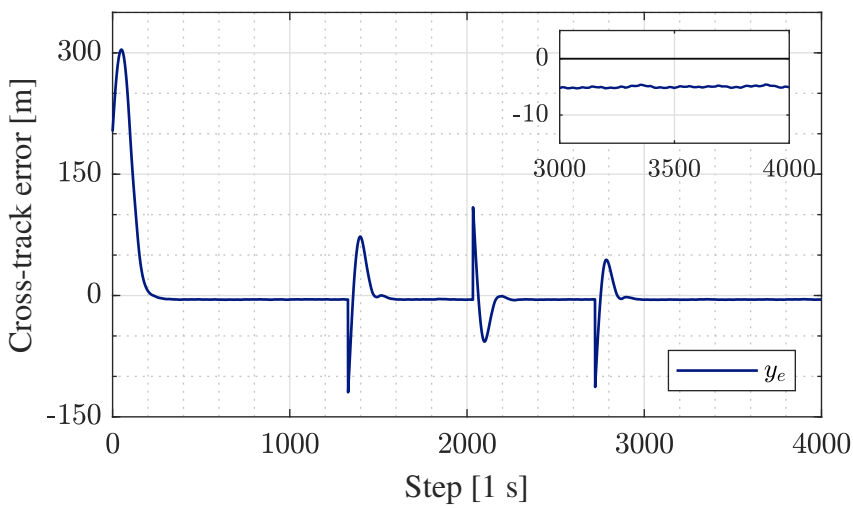


Figure 4.20: Cross-track error of DRL agent performing path following with surge control on a PSV

4.3.2 Platform supply vessel simulations

Representative results after training a DRL agent with the reward of Equation (3.15) and using the state in Equation (3.19) are shown here. Figures 4.19 and 4.20 illustrate that the path following task is solved similarly well as in Section 4.2.2, in that the vessel converges to a steady-state cross-track error between 0 and 10m within a fairly short amount of time of around 400 seconds.

The control inputs and resulting heading and surge can be found in Figure 4.21 (ψ) and Figure 4.22 (u). It is clear that smooth control inputs were not achieved here, and a solution accomplishing this feat was not found within the time available for this thesis. The heading command is relatively steady compared to the surge command – u_c almost spans its entire space of possible values between 0 and 10. This is reflected in velocities of Figure 4.23, where the amplitude of oscillations in u has larger magnitude than in v. One can also observe that the cross-track error is not converging to a perfectly constant value, possibly a result of small perturbations in heading.

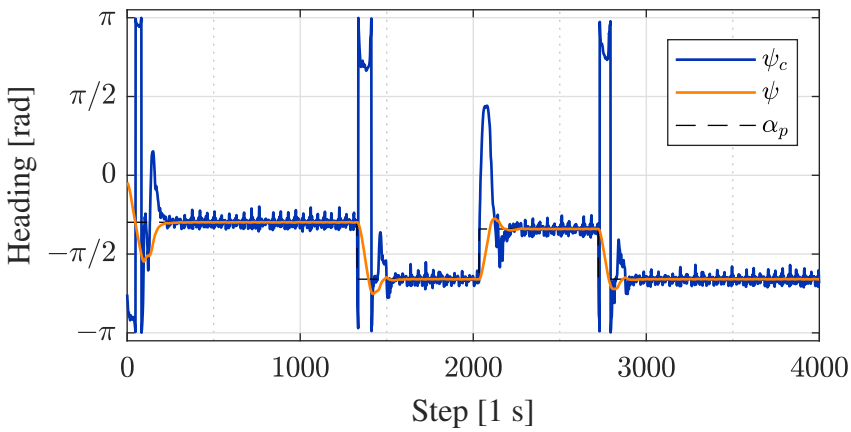


Figure 4.21: Heading command and response of DRL agent performing path following with surge control on a PSV

Despite these drawbacks, it is demonstrated that simultaneous path following and surge control can be achieved for the PSV, at least to a certain degree. If it is found in the future that this DRL algorithm cannot learn to give a reliably constant surge command, it may be beneficial to apply a low-pass filter after training, effectively removing oscillations from u_c. However, due to the system

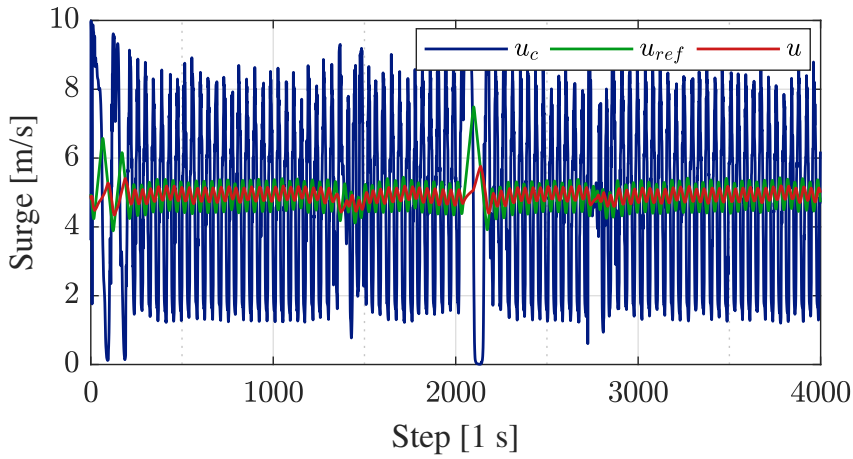


Figure 4.22: Surge command and response of DRL agent performing path following with surge control on a PSV

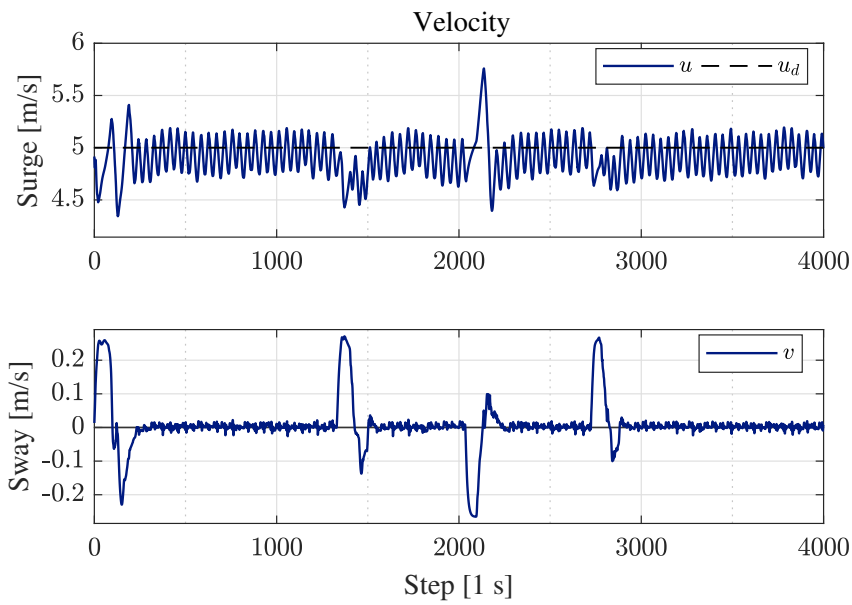


Figure 4.23: Velocity of DRL agent performing path following with surge control on a PSV

already being equipped with a surge controller, one could argue that forcing a DRL algorithm to learn which surge control inputs to apply is redundant. Equivalent results can be obtained by calculating or choosing the desired surge and feeding this to the controller. For these reasons, further investigation of DRL algorithms for the PSV is not conducted here.

For the interested reader, figures of thruster forces and orientations for these simulations can be found in Appendix C.

4.3.3 Container vessel simulations

The results obtained in the path following with surge control task for the container vessel can be found here. Equation (3.17), which includes rewards for small cross-track error and surge error, and penalty for shaky actions, was used in training of the DRL agent. The state vector is given by Equation (3.19).

From the trajectory of Figure 4.24 and corresponding cross-track error of Figure 4.25, one can see that the path following task is almost accomplished, with a steady-state error of less than 10m, which is comparable to the result of Section 4.2.3. However, slightly wider turns are made by the vessel in this case. A possible reason for this behaviour is the fact that the surge is now controlled,

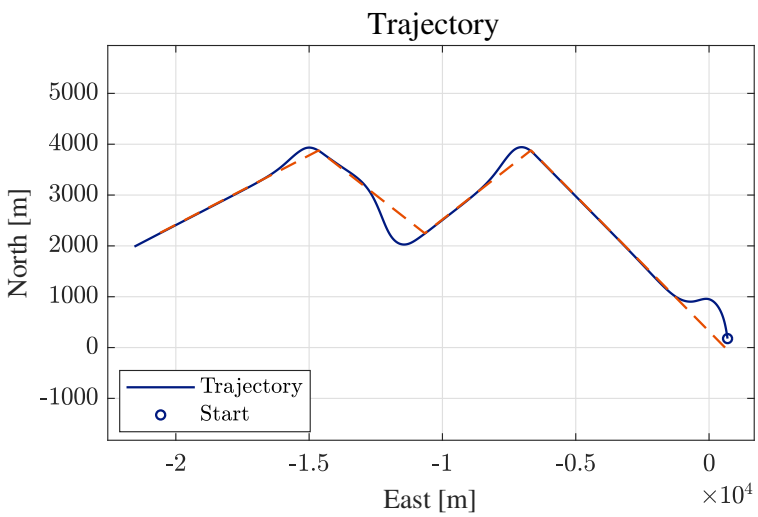


Figure 4.24: Trajectory of DRL agent performing path following with surge control on a container vessel

meaning that only insignificant amounts of surge speed is lost during turning. In Section 4.2.3 on the other hand, a fixed shaft speed was applied, resulting in transfer of speed from the surge component to sway as a result of sideways movement. This means that when a surge controller with constant reference is applied, the vessel usually moves at higher total speed during turning motion – and wider turns are produced. As a sidenote, if it is of importance to remain as close as possible to the path at all times, it is often beneficial to design the desired surge to be reduced whenever abrupt turning is required. This would allow the vehicle to be controlled more precisely, but is not addressed in this thesis.

Figure 4.26 shows the rudder input δ_c together with actual rudder angle δ in the top figure, and resulting heading ψ in the bottom figure. The rudder input has been successfully discouraged from being erratic, and the result is that the rudder angle follows the command with very little deviation. Heading angle converges to the path angle α_p once cross-track error has reached its steady-state value, which is as expected.

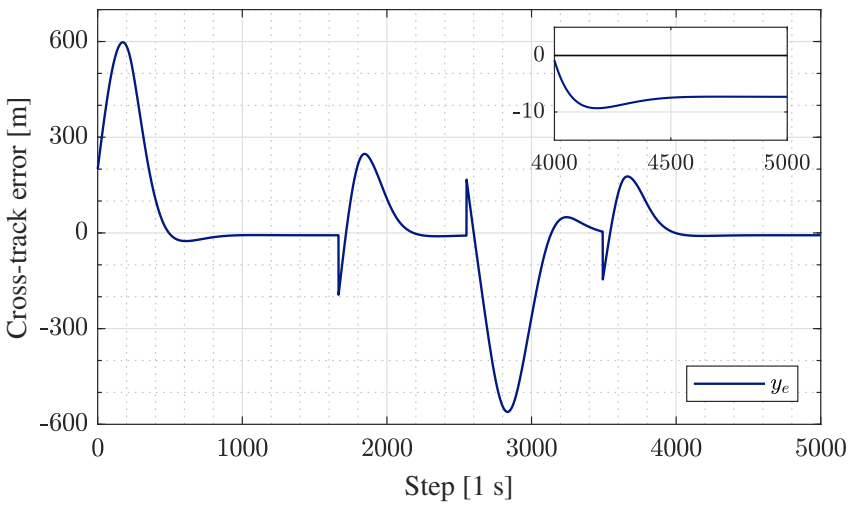


Figure 4.25: Cross-track error of DRL agent performing path following with surge control on a container vessel

The shaft speed input n_c and corresponding shaft speed n can be found in Figure 4.27. This figure also shows the surge and sway of the vessel, which demonstrate how sway is nonzero when the vessel is turning, and how this is

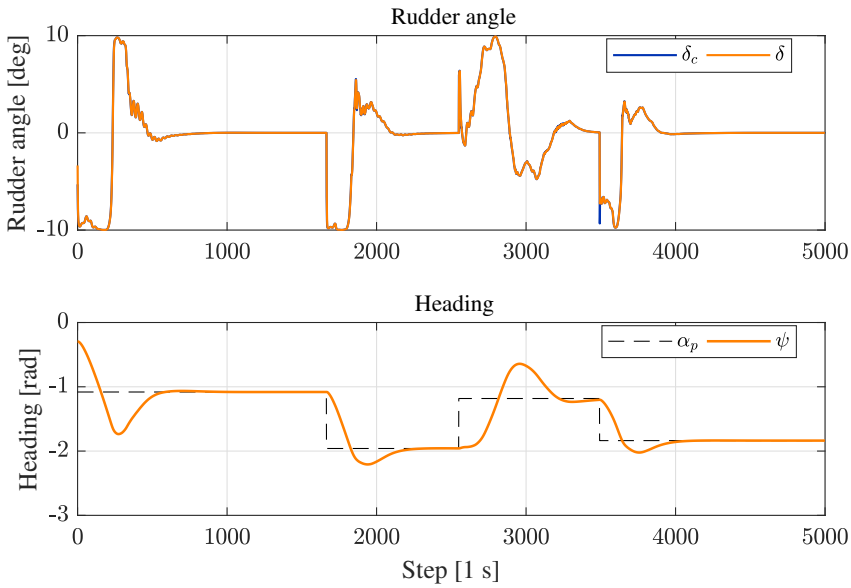


Figure 4.26: Rudder input and heading of DRL agent performing path following with surge control on a container vessel

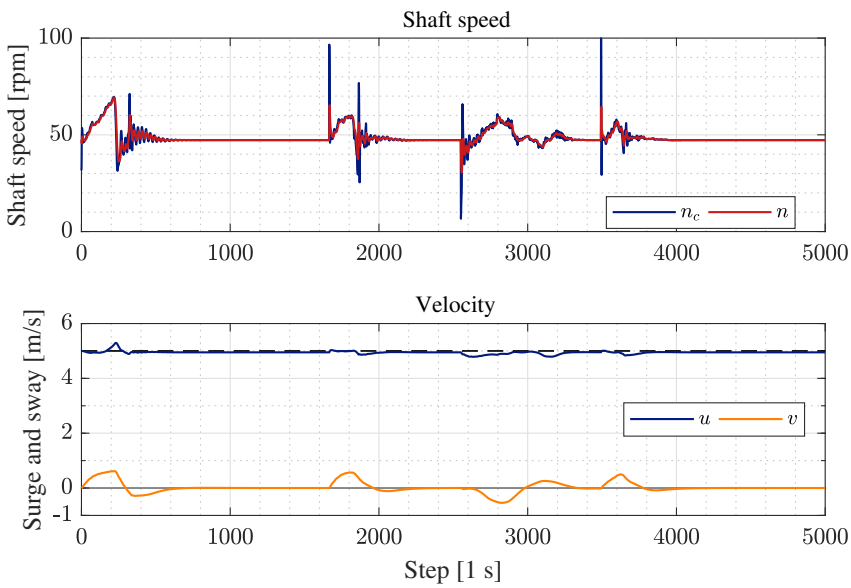


Figure 4.27: Shaft speed input and velocity of DRL agent performing path following with surge control on a container vessel

compensated for by the surge controller: the commanded shaft speed changes to increase or reduce total speed, producing near constant surge u.

There was no penalty placed explicitly on $\dot{n}_c$, but the shaft speed command still converges to a constant value when the vessel is moving in a straight line. This is in contrast to the behaviour of the PSV, in which oscillations in surge command could not be counteracted easily, if ever. This indicates that it is easier for a DRL algorithm to find an adequate policy approximation for the container than for the PSV in the case of surge control. Since the container uses a rudder and rotating propeller for directional and speed control, the two control inputs can be considered as practically decoupled, and the relationship between control inputs and changes to direction and velocity is unambiguous. For the PSV, which is controlled at the lowest level by two thrusters that can produce different forces at different angles, we can not assume the same. Certain combinations and sequences of heading and surge commands may counteract each other, for instance.

4.4 Collision avoidance

This section presents simulation results obtained after training a DRL agent to perform collision avoidance on a container vessel as described in Section 3.5, using the state vector of (3.21), performance measures of (3.22) and (3.24), and commanded rudder angle δ_c as control input. The obstacle is modelled as a container vessel moving in a straight line with constant velocity.

Two cases are investigated. Both entail avoiding an obstacle vessel approaching from a reciprocal course, in other words a head-on situation. In the first case, the reward signal of (3.22) was used without any modifications dictating the conventions of COLREGS. These simulation results can be found in Section 4.4.2. Secondly, the reward signal is expanded to include a penalty inside the region illustrated in Figure 3.7, given by (3.24). This change was made for the purpose of encouraging early and substantial action to avoid collision, which is the essence of Rule 16 of COLREGS. Results of this training is shown in Section 4.4.3. These two case studies will be referred to as *simple head-on* and *extended head-on* for easy distinction.

4.4.1 Training

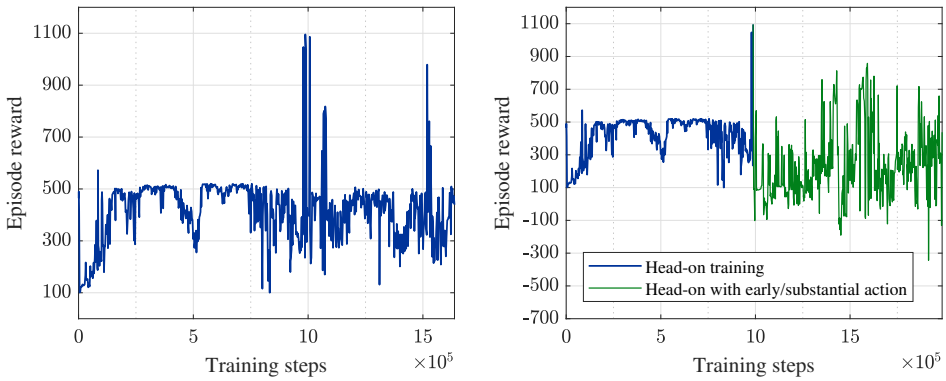
Training was carried out as follows. First, a pure path following guidance system was trained using the full state vector needed for collision avoidance, but with the reward signal designed for path following with penalty, as shown in (3.16). Then transfer learning was utilised by applying the trained actor and critic networks of the path following agent as initial values for the simple head-on collision avoidance agent. Since path following is a vital part of a complete collision avoidance guidance system, this approach enables the agent to refine the policy to include avoidance of obstacles rather than needing to build a path follower and obstacle avoidance simultaneously from scratch, consequently reducing the necessary training time. When moving on to the extended head-on case, the best performing simple head-on agent was used as the starting point through transfer learning once again. An alternative for the extended head-on CA agent is to begin training with transfer learning directly from the path following agent in the same manner as for the simple head-on case. The reason for using a simple head-on CA agent as starting point was an attempt to reduce training time.

The path following agent was trained by treating the obstacle vessel as a make-believe vessel where small distances had no effect on reward or episode termination. This kind of training would thus not be feasible when working with real vessels. It can be discussed whether this is the best approach, or if the path following agent should only be exposed to states where there would be no possibility of collision, to avoid any contradictory learning.

Initial values during training of the path following system were chosen in the same manner as in Section 4.2.1, in addition to letting shaft speed and heading of the obstacle vessel be $n_o = 20$ rpm and $\psi_o \in \left[\alpha_p - 4\frac{\pi}{180}, \alpha_p + 4\frac{\pi}{180}\right]$, respectively. Initial values for distances to the obstacle were chosen in the ranges $x_{er} \in [-4500, 4500]$m and $y_{er} \in [-100, 100]$m, which allow the agent to experience several phases of a head-on scenario in early stages of training. A limit was applied to the measured along-track distance to simulate a detection range, so that when $x_{er} > 2500$m, x_{er} is assumed equal to 2500. The maximum episode length was 1000s.

When using transfer learning to train the collision avoidance agent for head-

on situations, it is assumed the agent has learned to follow the path. Thus, the initial cross-track error range was reduced slightly in this phase and set to $y_e \in [-300, 300]$m. The initial heading range was also reduced, to $\psi \in \left[\alpha_p - \pi/4, \alpha_p + \pi/4\right]$. For collision avoidance, we expect an evasive manoeuvre to take longer than reaching a path, therefore the maximum episode length was 1500s. The safe distance imposed on the vessel was $d_{safe} = L_{pp} = 175$m and the action limits were $\delta_c = [-10°, 10°]$. Reward function parameters of (3.22) and (3.24) can be found in Table B.4 in Appendix B.



(a) Training progress for head-on situation

(b) Training progress for head-on situation with early and substantial action

Figure 4.28: Reward history for the collision avoidance task

Evaluation of the agent was performed every other episode for this task, which was needed in order to reduce chances of missing out on a truly good agent. Figure 4.28 shows the training progress of the CA agent training for a head-on scenario, where Figure 4.28a is the progress in the case of the simple head-on case and the green curve of Figure 4.28b illustrates the progress made in the extended head-on case. It can be seen that the training is not stable. At some points the agent performs well, passing the obstacle vessel and re-converging to the desired path, which corresponds to the spikes seen in the training curve. Most of the time, however, finding a solution to the collision avoidance task becomes a challenge for the agent, and it often ends up in a local maximum where it believes that following the path is a good solution regardless of the obstacle vessel position and velocity. The vessels crash after about 500 seconds,

thus the accumulative reward received in these cases was also 500 in the simple head-on case. These observations suggest the magnitude of the penalty given when colliding may be too low. It could also suggest that the chosen system architecture and reward function design can be improved.

4.4.2 Head-on situation

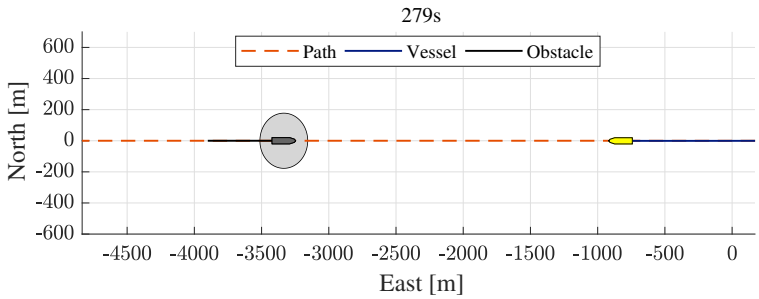
In a head-on situation, two vessels meet when travelling in approximately opposite directions, and each vehicle must steer clear of the other to avoid colliding. The simulation results presented here were obtained by testing the trained DRL agent of Section 3.5 with reward signal of (3.22). The controlled vessel was placed in an initial position on the path with heading equal to the path angle α_p, and shaft speed $n = 50$ rpm (translating to a BODY-fixed velocity $\boldsymbol{v}^b_{b/n} \approx [5, 0]^\top$). The obstacle vessel was placed on the path with an along-track distance of 4500m to the controlled vessel, with heading $\psi_o = \alpha_p + \pi$ and shaft speed $n_o = 20$ rpm. The initial values of the test scenario are summarised in Table 4.2.

Vessel	Position (x, y)	Heading (ψ)	Shaft speed (n)	Waypoints
Controlled	$(0, 600)$	$-\pi/2$ rad	50 rpm	$(0, 700), (0, -5000)$
Obstacle	$(0, -3900)$	$\pi/2$ rad	20 rpm	$(0, -5000), (0, 700)$

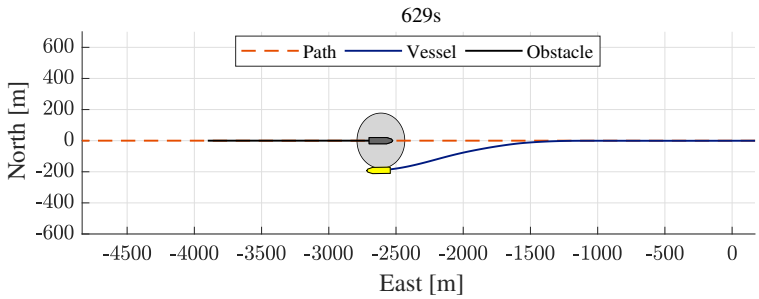
Table 4.2: Head-on initial values

In the simulation only one of the vessels, denoted the *controlled vessel*, is controlled by the DRL agent, whereas the *obstacle vessel* travels straight ahead with no attempts at avoiding collision. The main reason for disregarding Rule 14 of COLREGS in the obstacle vessel is simplicity. Additionally, exposing the DRL agent to situations where the obstacle behaves according to the rules during training, could result in the agent learning to continue along the path rather than taking action to avoid the other vessel, as this would give maximum reward with the current design.

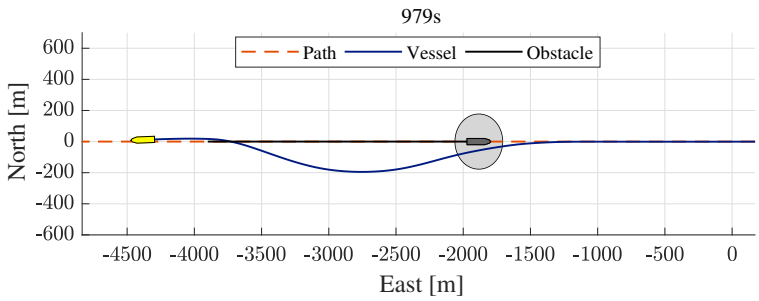
Figure 4.29 shows the simulation of the collision avoidance manoeuvre, with obstacle vessel in black and controlled vessel in yellow (with blue line showing



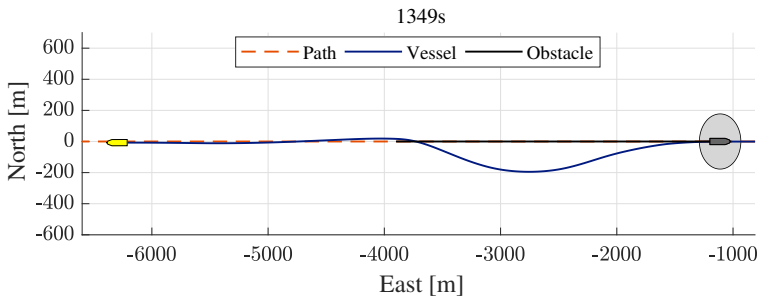
(a) Initial path following



(b) During avoidance



(c) Converging to path after avoidance



(d) Converged to path

Figure 4.29: Simulation of DRL agent performing collision avoidance in head-on situation

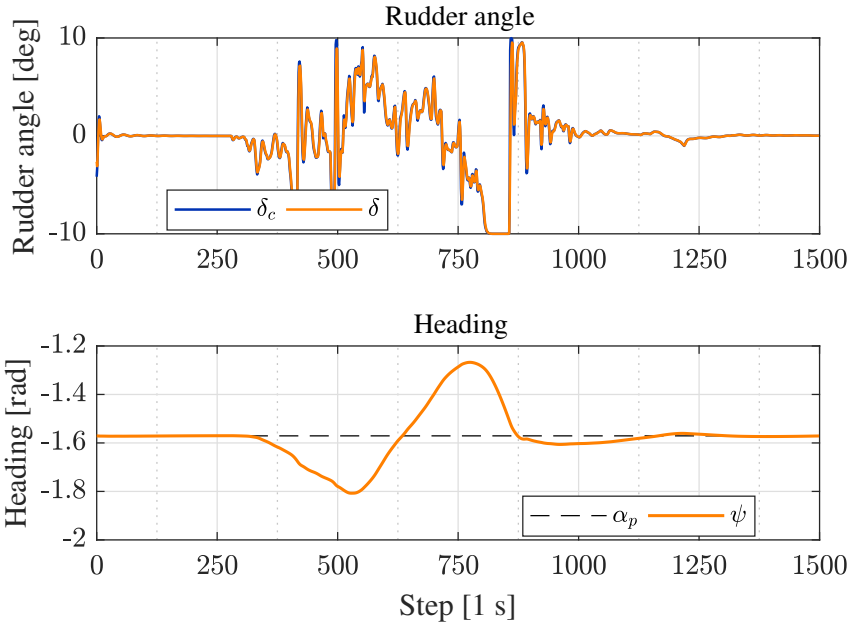


Figure 4.30: Rudder input and heading of DRL agent performing collision avoidance in head-on situation

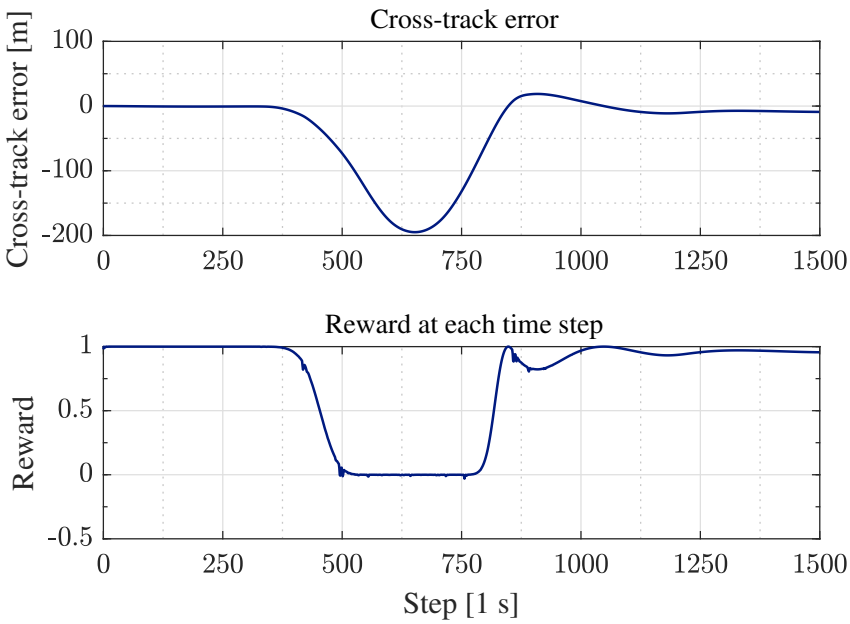


Figure 4.31: Cross-track error and instantaneous reward of DRL agent performing collision avoidance in head-on situation

its trajectory). The rudder angle and heading of the controlled vessel during simulation is shown in Figure 4.30, and its cross-track error and received reward per time step can be found in Figure 4.31. Figure 4.32 depicts position of the obstacle relative to controlled vessel in the path-fixed coordinate frame.

Rudder angle and heading converge nicely to 0 and α_p, respectively, when the vessel follows the given path. This corresponds to the desired behaviour of a CA control system. The agent manages to adjust the heading to let the vessel move away from the path, before converging again once it has passed the obstacle. The point where ψ crosses α_p at close to 625s lines up with the trajectory of Figure 4.29b, which shows the vessel position in the middle of its evasive manoeuvre at 629s.

At first sight, the plotted reward of Figure 4.31 suggests that the agent is not behaving optimally, due to the long period where zero reward is given to the agent. However, the alternative would have been to collide with the obstacle and receiving a considerable penalty. The distances between vessels shown in Figure 4.32 confirm that the vehicles pass each other in the period with reduced

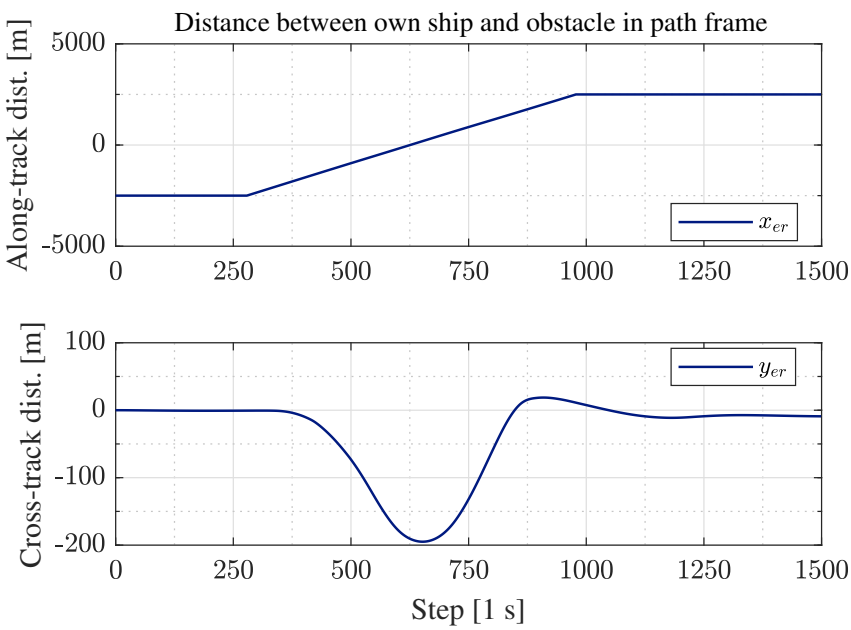


Figure 4.32: Relative positions of DRL agent performing collision avoidance in head-on situation

reward, and that this movement is therefore close to optimal behaviour.

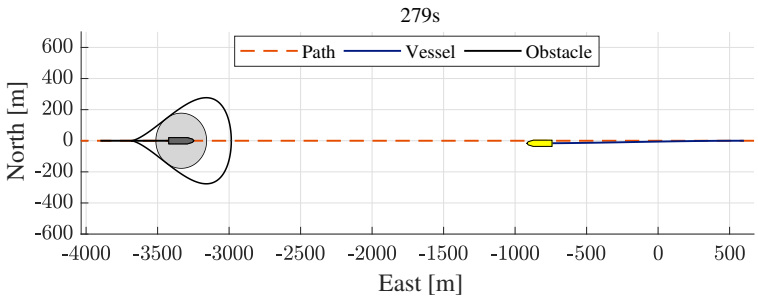
In conclusion, a collision avoidance control system successfully steering clear of an obstacle in a head-on situation was found. A drawback of this solution is that it violates COLREGS, as the correct behaviour in a head-on scenario would have been alteration of course to starboard rather than to port. The lack of information regarding COLREGS in the performance measure is to blame for this result.

4.4.3 Head-on situation with early and substantial action

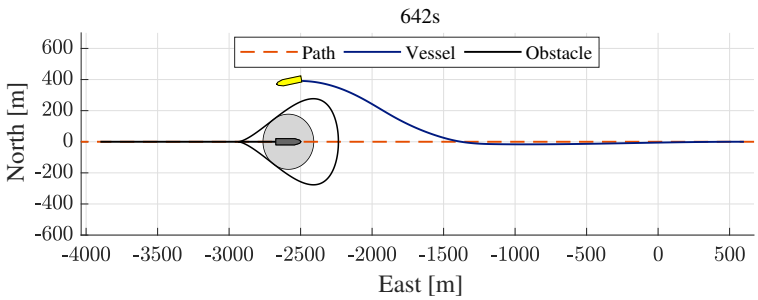
Section 3.5.1 suggested implementing a teardrop-shaped penalty region, shown in Figure 3.7, to accommodate Rule 16 of COLREGS. Rule 16 describes guidelines for actions taken by the give-way vessel in a collision avoidance situation, and is repeated in Section 2.3.1. In this Section, simulation results were realised by testing the DRL agent trained for the extended head-on case with the reward given by (3.22), with an additional penalty of -5 inside the region of Figure 3.7. Initial conditions of the obstacle and controlled vessels are equal to the previous section, and can be found in Table 4.2.

The movement of the controlled vessel can be found in Figure 4.33, and it is shown that collision is successfully avoided in that the vessel initially follows the path, then performs an avoidance manoeuvre before going back to the path after passing the obstacle. Figures 4.34–4.36 show the rest of the simulation results. The actions of the controlled vessel are noticeably more substantial here, which Figure 4.34 illustrates. The rudder angle is at its limit more often than previously, indicating that the agent has learned to utilise the maximum turning rate. The largest cross-track error of Figure 4.35 is about 400m, twice the distance reached in Section 4.4.2, which confirms that a wider region of penalty does inspire larger course changes and path deviation.

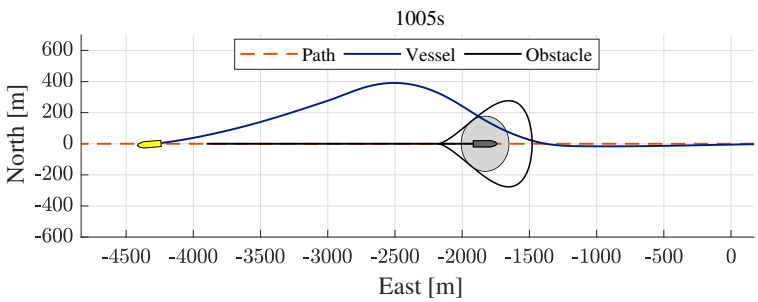
The first action to avoid collision was taken at approximately 300s, which was also the case in simulations of the simple head-on case. The reason for this is likely to be the maximum measured along-track distance between vessels. It can be observed in both Figure 4.36 and 4.32 that this distance is believed to be 2500m until about 300s have passed, at which point the agent begins to dodge the obstacle.



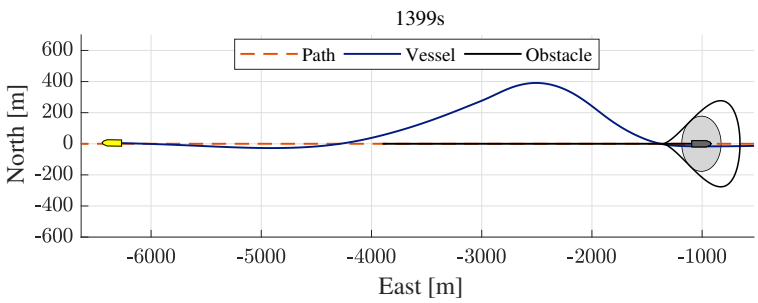
(a) Initial path following



(b) During avoidance



(c) Converging to path after avoidance



(d) Converged to path

Figure 4.33: Simulation of DRL agent performing collision avoidance in head-on situation following convention of Rule 16

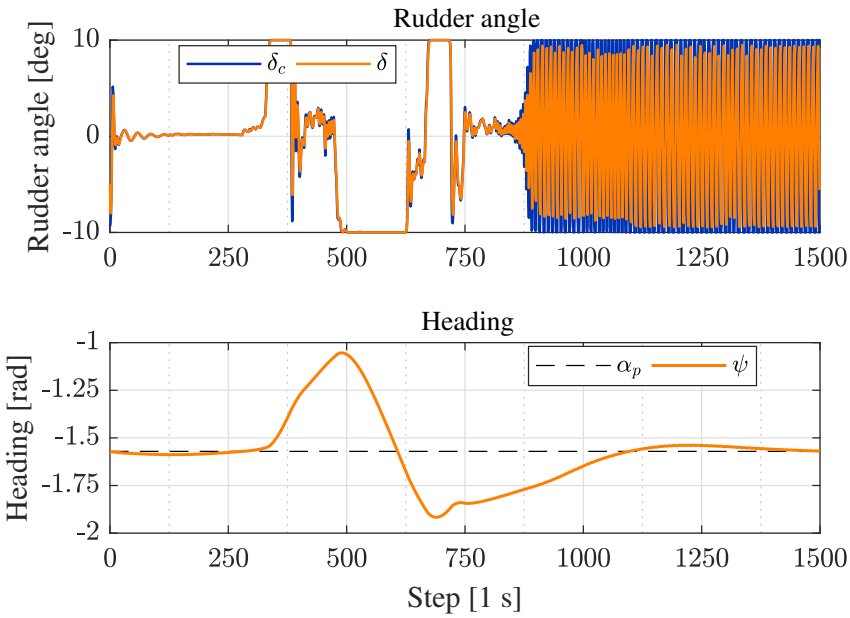


Figure 4.34: Rudder input and heading of DRL agent performing collision avoidance in head-on situation following convention of Rule 16

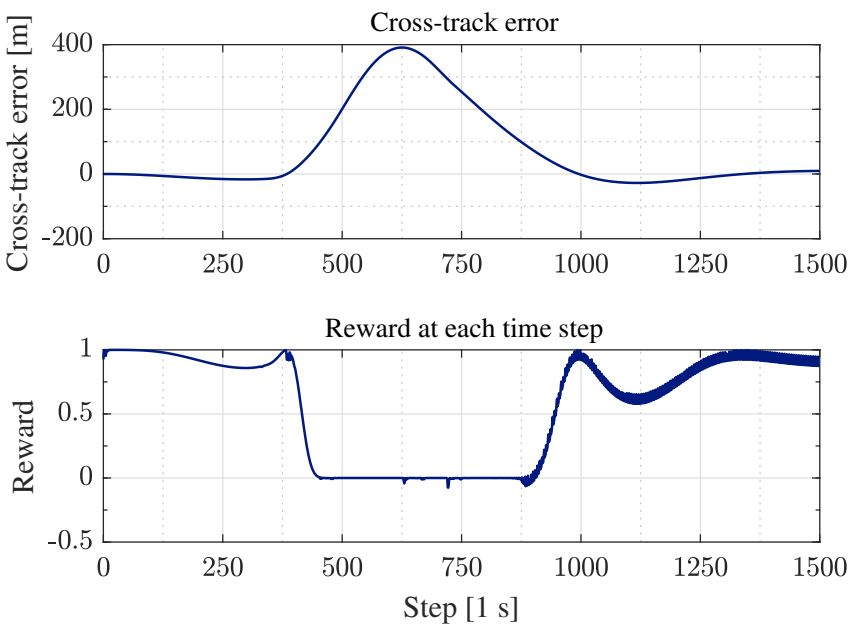


Figure 4.35: Cross-track error and instantaneous reward of DRL agent performing collision avoidance in head-on situation following convention of Rule 16

This collision avoidance agent ended up with severe fluctuations in rudder input in the last phase, where the vessel has passed the obstacle and is allowed to converge to the path once again. This can be seen in the rudder angle in Figure 4.34 as well as in the received reward shown in Figure 4.35, where the erratic rudder angle results in reduced reward.

In conclusion, the results have shown that shaping a region of penalty appropriately and letting the region surround the obstacle, can guide the collision avoidance agent in adapting to conventions of COLREGS. The region can have any shape, and thus, in theory, the agent can be trained to comply with any rule. However, it was found that the agent was prone to giving shaky control input. Such behaviour is often the result of training a DRL agent for too long, sometimes resulting in saturation of the policy network output. This could perhaps have been counteracted by increasing the weight of the $\dot{\delta}$-penalty, providing even more incentive to reduce changes in δ.

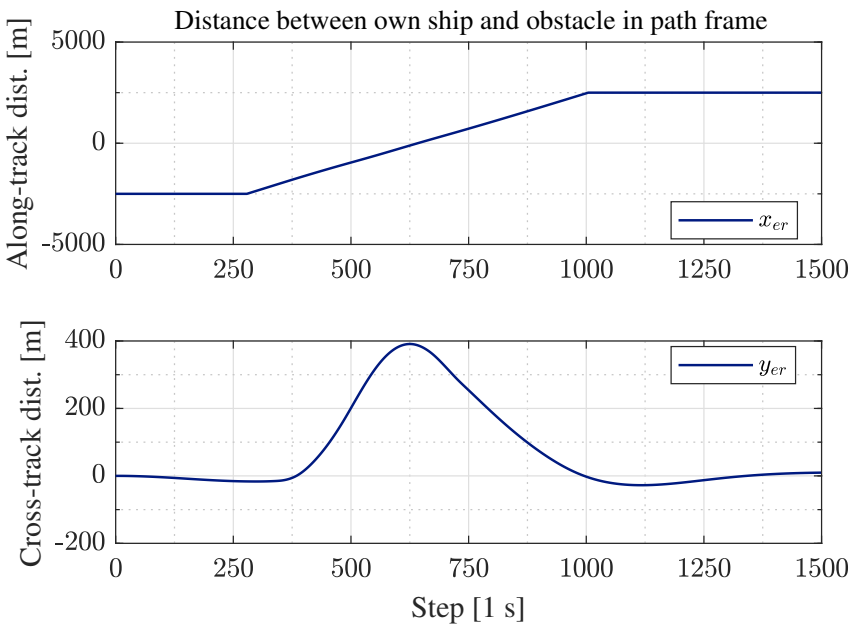


Figure 4.36: Relative positions of DRL agent performing collision avoidance in head-on situation following convention of Rule 16

4.5 Summary of results

This chapter has presented several results within deep reinforcement learning control of different types of marine vessels. The control tasks were implemented as an iterative process, because finding solutions in later tasks were dependent on success within earlier tasks.

A DRL guidance system for the PSV was tested in Section 4.1, which gave reference signals to heading and surge controllers embedded in the vessel. It was shown that DRL can provide suitable inputs causing the vessel to reach its desired heading and surge quickly, by letting the reference signals overshoot desired values. Guiding the surge towards a constant value was found to be challenging, however, and requires more investigation before control systems based on DRL can be utilised in the PSV.

DRL control systems for path following, with and without surge control, were presented for two vessel types in Section 4.2 and 4.3. The trained agents performed particularly well in simulations of the container vessel. Due to control allocation differences between the two vessel types, it was found that the PSV was able to reach a path more quickly than the container, but that reward function tuning for the PSV was more challenging.

Performance of a DRL collision avoidance system, shown in Section 4.4, was promising as a first result. The task was constrained to head-on situations with a single obstacle, but was found to satisfactorily avoid collision. An experiment with obstacle expansion for COLREGS compliance was conducted, which illustrated the possibilities of extending the system with different rules of navigation.

The investigation performed in this thesis has been extensive, while at the same time limited. The effects of environmental disturbances have not been considered, and neither has extension to path following for curved paths. A single collision scenario with one obstacle has been examined, and no more than one rule from COLREGS was directly incorporated in the final system. Due to the experimental procedure of the research in deep reinforcement learning for continuous control systems, these expansions were not prioritised, but are nonetheless requirements in a successfully deployed collision avoidance system and should be investigated further.

4.6 Future work

The work done in this thesis can be taken further in a number of directions. Some of the most obvious have already been mentioned, such as investigation of robustness to environmental disturbances and application with curved paths, and also expansion for handling of several obstacles and further investigation of control structures specific for the PSV. This section will focus on three ways to improve the collision avoidance system: COLREGS compliance, explainability and choosing a different system architecture.

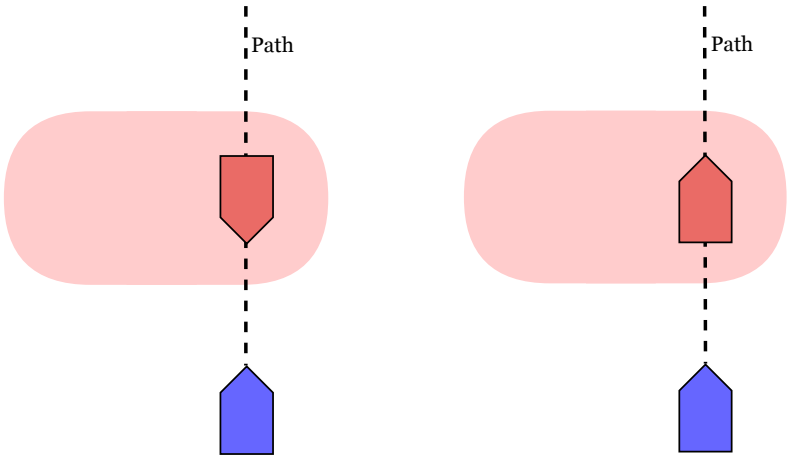
COLREGS compliance

Expanding the obstacle as explained in Section 3.5.1.1 can guide the DRL agent to prefer actions that comply with COLREGS over actions that result in violation of the rules. In a head-on situation the vessel is instructed to pass on the port side of the obstacle (from the point of view of the obstacle), thus the expansion should be in a direction perpendicular to the obstacle's velocity vector, on its starboard side. In an overtaking situation, passing on the obstacle's starboard side is preferred, therefore its port side should induce penalty from the expanded region. Demonstration of obstacle expansion in a head-on and an overtaking situation is shown in Figure 4.37. Thus, depending on the situation, different penalty regions can be defined such that the system becomes COLREGS compliant.

Explainability

A significant driving force behind utilisation of DRL in continuous control is the potential to encompass a wide set of instructions and decisions by learning from examples rather than explicit programming. However, this means that the inner workings of a trained system is often too complicated for humans to easily understand. This is why AI, within which DRL belongs, is often referred to as a *black box*, and analysing artificially intelligent systems is a challenge due to complexity.

Explainability in AI is important in establishing trust – a human interacting with an artificially intelligent machine is often naturally sceptic, and reluctant to



(a) Head-on **(b)** Overtaking

Figure 4.37: Extension of obstacle (red) in the direction of COLREGS violation

trust decisions made by the AI system without knowledge about its reasoning. Additionally, analysis of an agent's decision making process can help uncover inconsistencies, and is thus a strategy for checking whether the data (input to output) makes sense. LIME [52] and integrated gradients [53] are techniques for explaining the predictions of deep networks, which can assist users both in providing explanations for specific decisions and suggesting which features of input data are most important for decision making. For these reasons, applying explainable AI to the collision avoidance agent of this thesis could be helpful in determining strengths and weaknesses of the approach, and thus guide researchers in expanding this work.

It is important to distinguish between trusting a system to perform without fail and trust in the context of understanding why a decision was made. Explainable AI can provide the latter. It can also assist developers in uncovering inconsistent behaviour that should be corrected. However, it should be stressed that explainable AI cannot replace the need for extensive testing of systems before applying them in the real world.

Alternative system architecture

An alternative approach to development of a DRL collision avoidance system was discussed in Section 3.5, and it was suggested to combine a path planner with a path following system, both implemented and trained as DRL agents. The path planner can modify the originally intended path in the case of danger of collision, and the path follower can be used to follow the newly planned path. With this architecture, the already implemented DRL path following agent can be used.

By separation into two tasks, the reward function design can be simplified, making the full collision avoidance system potentially more robust to conflicting objectives. For instance, the path following objective will in this case be less likely to overrule collision avoidance due to path following being performed separately.

Another possible design change is to investigate other DRL algorithms than DDPG. Recent years has seen rapid development within DRL, and some promising algorithms were presented as late as summer 2018. Some examples include *twin delayed DDPG (TD3)* [54], which has been shown to improve stability of learning compared to DDPG, and *soft actor-critic* [55], which is a powerful stochastic off-policy DRL algorithm.

Chapter 5

Conclusion

This thesis has investigated the use of deep reinforcement learning in path following and collision avoidance for marine vessels with unknown models. The deep deterministic policy gradient algorithm was used.

Results of a previously presented DRL path following system [18] have been reproduced and extended to include surge control. Robustness of the proposed control structure has been confirmed by applying the same method to two vessel models with substantially differing control allocation and demonstrating their respective successes. Furthermore, DRL has been applied to a rudimentary collision avoidance task for a container vessel by building on the framework used for path following, and it was demonstrated that the proposed reward function could be extended to include guidelines in accordance with COLREGS. The resulting performance of the collision avoidance control system showed that transfer learning can be used as a tool to help solve a new control task only by changing the reward function of an agent trained to solve a similar task.

The simulation results presented for each problem indicate that DRL generally results in good performance, and is a suitable framework for the examined navigation tasks. To be considered as a completely autonomous collision avoidance system, more research, development and testing is needed. Some suggestions for future work have been given, based on the challenges encountered during this thesis.

Bibliography

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[2] A. B. Martinsen, "End-to-end training for path following and control of marine vehicles," Master's thesis, Norwegian University of Science and Technology, 2018.

[3] T. I. Fossen and T. Perez, "Marine Systems Simulator (MSS)," 2004. [Online]. Available: https://github.com/cybergalactic/MSS

[4] SNAME, The Society of Naval Architecture and Marine Engineers, "Nomenclature for treating the motion of a submerged body through a fluid," *Technical and Research Bulletin No. 1–5*, 1950.

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

[6] A. N. Gorban and A. Y. Zinovyev, "Transfer learning," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques.* IGI Global, 2009, pp. 242–264.

[7] H. Sahin and L. Guvenc, "Household robotics - autonomous devices for vacuuming and lawn mowing," *Control Systems, IEEE*, vol. 27, pp. 20 – 96, May 2007.

[8] M. Bajracharya, M. W. Maimone, and D. Helmick, "Autonomy for Mars rovers: Past, present, and future," *Computer*, vol. 41, no. 12, pp. 44–50, Dec 2008.

[9] D.-G. Kim, K. Hirayama, and T. Okimoto, "Ship collision avoidance by distributed tabu search," *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, vol. 9, no. 1, pp. 23–29, 2015.

[10] E. Demirel and D. Bayer, "The further studies on the COLREGs (collision regulations)," *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, vol. 9, pp. 17–22, March 2015.

[11] International Maritime Organization, "COLREGS -— Convention on the International Regulations for Preventing Collisions at Sea," 1972. [Online]. Available: http://www.jag.navy.mil/distrib/instructions/COLREG-1972.pdf

[12] T. A. Johansen, T. Perez, and A. Cristofaro, "Ship collision avoidance and COLREGS compliance using simulation-based control behavior selection with predictive hazard assessment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 12, pp. 3407–3422, Dec 2016.

[13] T. I. Fossen, M. Breivik, and R. Skjetne, "Line-of-sight path following of underactuated marine craft," in *6th IFAC Conference on Manoeuvring and Control of Marine Craft (MCMC)*, vol. 36, no. 21, 2003, pp. 211 – 216.

[14] A. M. Lekkas and T. I. Fossen, "Integral LOS path following for curved paths based on a monotone cubic hermite spline parametrization," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2287–2301, Nov 2014.

[15] S. Moe and K. Y. Pettersen, "Set-based line-of-sight (LOS) path following with collision avoidance for underactuated unmanned surface vessels under the influence of ocean currents," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, Aug 2017, pp. 241–248.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, October 2017.

[18] A. B. Martinsen and A. M. Lekkas, "Straight-path following for underactuated marine vessels using deep reinforcement learning," *11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles (CAMS)*, vol. 51, no. 29, pp. 329 – 334, 2018.

[19] A. B. Martinsen and A. Lekkas, "Curved path following with deep reinforcement learning: Results from three vessel models," in *OCEANS MTS/IEEE*, October 2018, pp. 1–8.

[20] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle," in *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 4958–4965.

[21] S. Moe, K. Y. Pettersen, T. I. Fossen, and J. T. Gravdahl, "Line-of-Sight curved path following for underactuated USVs and AUVs in the horizontal plane under the influence of ocean currents," in *2016 24th Mediterranean Conference on Control and Automation (MED)*, June 2016, pp. 38–45.

[22] M. R. Benjamin and J. A. Curcio, "Colregs-based navigation of autonomous marine vehicles," in *2004 IEEE/OES Autonomous Underwater Vehicles (IEEE Cat. No.04CH37578)*, June 2004, pp. 32–39.

[23] I. B. Hagen, D. K. M. Kufoalor, E. F. Brekke, and T. A. Johansen, "MPC-based collision avoidance strategy for existing marine vessel guidance systems," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7618–7623.

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *1985 IEEE International Conference on Robotics and Automation*, vol. 2, March 1985, pp. 500–505.

[25] H. Lyu and Y. Yin, "Ship's trajectory planning for collision avoidance at sea based on modified artificial potential field," in *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*, Dec 2017, pp. 351–357.

[26] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, March 1997.

[27] Y. Kuwata, M. T. Wolf, D. Zarzhitsky, and T. L. Huntsberger, "Safe maritime autonomous navigation with colregs, using velocity obstacles," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 1, pp. 110 – 119, Jan 2014.

[28] W. Zhang, S. Wei, Y. Teng, J. Zhang, X. Wang, and Z. Yan, "Dynamic obstacle avoidance for unmanned underwater vehicles based on an improved velocity obstacle method," *Sensors*, vol. 17, p. 2742, Nov 2017.

[29] Y. Cheng and W. Zhang, "Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels," *Neurocomputing*, vol. 272, pp. 63 – 73, 2018.

[30] H. Shen, H. Hashimoto, A. Matsuda, Y. Taniguchi, D. Terada, and C. Guo, "Automatic collision avoidance of multiple ships based on deep Q-learning," *Applied Ocean Research*, vol. 86, pp. 268 – 288, 2019.

[31] I. Carlucho, M. De Paula, S. Wang, B. V. Menna, Y. R. Petillot, and G. G. Acosta, "AUV position tracking control using end-to-end deep reinforcement learning," in *OCEANS 2018 MTS/IEEE Charleston*, Oct 2018, pp. 1–8.

[32] Q. Wang and C. Phillips, "Cooperative collision avoidance for multi-vehicle systems using reinforcement learning," in *2013 18th International Conference on Methods Models in Automation Robotics (MMAR)*, Aug 2013, pp. 98–102.

[33] S. J. Russel and P. Norvig, *Artificial Intelligence. A Modern Approach*, 3rd ed. Upper Saddle River, New Jersey, USA: Prentice Hall, 2010.

[34] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts, USA: The MIT Press, 2018.

[35] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *In Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 1057–1063.

[36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992.

[37] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, vol. 32. JMLR.org, 2014, pp. I–387–I–395.

[38] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec 1989.

[39] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G Bellemare, A. Graves, M. Riedmiller, A. K Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct 1986.

[42] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, July 2011.

[43] 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, 2012.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[46] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *ICML*, 2004.

[47] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[48] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992.

[49] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, pp. 823–841, Sep 1930.

[50] K. Son and K. Nomoto, "On the coupled motion of steering and rolling of a high speed container ship," *Journal of the Society of Naval Architects of Japan*, vol. 1981, pp. 232–244, 01 1981.

[51] M. S. Wiig, K. Y. Pettersen, and T. R. Krogstad, "A reactive collision avoidance algorithm for vehicles with underactuated dynamics," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec 2017, pp. 1452–1459.

[52] M. T. Ribeiro, S. Singh, and C. Guestrin, “"Why should i trust you?": Explaining the predictions of any classifier,” in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2016, pp. 1135–1144.

[53] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. JMLR.org, 2017, pp. 3319–3328.

[54] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., vol. 80. PMLR, Jul 2018, pp. 1587–1596.

[55] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., vol. 80. PMLR, Jul 2018, pp. 1861–1870.

[56] M. Abkowitz, *Lectures on Ship Hydrodynamics: Steering and Maneuvrability*, ser. Technical report Hy-5. Hydro- and Aerodynamics Laboratory, Hydrodynamics Department, 1964.

Appendix A

Container model

The Container vessel model has 4-DOF (degrees of freedom). A MATLAB model can be found in the MSS Toolbox [3], based on the work of Son and Nomoto [50]. In certain cases, a 3-DOF model is sufficiently accurate for describing the motion of a surface vessel, such as when motion in roll, pitch and heave can be neglected. However, since the roll and sway-yaw dynamics are often strongly coupled, including roll in the equations of motion can increase model accuracy. The state vectors of such a 4-DOF vessel model are $\boldsymbol{\eta} = [x \quad y \quad \phi \quad \psi]^\top$ and $\boldsymbol{\nu} = [u \quad v \quad p \quad r]^\top$, where $\boldsymbol{\eta}$ is a position vector in the NED (North-East-Down) coordinate frame, and $\boldsymbol{\nu}$ is a velocity vector given in the vessel's BODY frame. The control vector $\boldsymbol{\tau}$ consists of the desired rudder angle δ_c and desired shaft speed n_c, where rudder angle is specified in radians and shaft speed is in rotations per minute (rpm). The equations of motion can be written in vector form as:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{A.1}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{A.2}$$

$$\dot{\delta} = \begin{cases} \delta_c - \delta, & \text{if } \delta \leq \delta_{max} \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

$$\dot{n} = \frac{60}{T_m}(n_c - n) \tag{A.4}$$

where $\boldsymbol{M}$ is the system inertia matrix, consisting of rigid body and hydrodynamic added mass matrices, $\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A$. The Coriolis-centripetal matrix $\boldsymbol{C}(\boldsymbol{\nu})$ can be similarly decomposed into $\boldsymbol{C}_{RB}$ and $\boldsymbol{C}_A$, and $\boldsymbol{D}(\boldsymbol{\nu})$ is the damping matrix.

The nonlinear terms of Equation (A.2) can be modelled from first principles such as surge resistance and cross-flow drag [5], or by curve-fitting of experimental data to a Taylor-series. The latter approach results in the nonlinear model of Abkowitz [56], which describes the nonlinear terms as $\boldsymbol{N}(\boldsymbol{\nu}) = \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}$. If we include the control input $\boldsymbol{\tau}$, Equation (A.2) becomes

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{N}(\boldsymbol{\nu}, \boldsymbol{\tau}) = 0 \tag{A.5}$$

The matrix $\boldsymbol{J}(\boldsymbol{\eta})$ denotes a transformation between BODY and NED coordinate frames and is given by a rotation of ϕ about the x-axis followed by a rotation of ψ about the z-axis, and by an angular velocity transformation matrix $\mathbf{T}(\phi)$:

$$\boldsymbol{J}(\boldsymbol{\eta}) = \begin{bmatrix} \boldsymbol{R}_{z,\psi}\boldsymbol{R}_{x,\phi} & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{T}(\phi) \end{bmatrix} = \begin{bmatrix} \mathrm{c}\psi & -\mathrm{s}\psi\mathrm{c}\phi & 0 & 0 \\ \mathrm{s}\psi & \mathrm{c}\psi\mathrm{c}\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathrm{c}\phi \end{bmatrix} \tag{A.6}$$

The inertia matrix of the Container model is

$$\boldsymbol{M} = \begin{bmatrix} m + m_x & 0 & 0 & 0 \\ 0 & m + m_y & -m_y l_y & m_y \alpha_y \\ 0 & -m_y l_y & I_x + J_x & 0 \\ 0 & m_y \alpha_y & 0 & I_z + J_z \end{bmatrix} \begin{bmatrix} \frac{L}{U^2} & 0 & 0 & 0 \\ 0 & \frac{L}{U^2} & 0 & 0 \\ 0 & 0 & \frac{L^2}{U^2} & 0 \\ 0 & 0 & 0 & \frac{L^2}{U^2} \end{bmatrix} \tag{A.7}$$

where L is the length and $U = \sqrt{u^2 + v^2}$ is the speed of the ship. The nonlinearities $\boldsymbol{N}(\boldsymbol{\nu}, \boldsymbol{\tau}) = \left[X(\boldsymbol{\nu}, \boldsymbol{\tau}), Y(\boldsymbol{\nu}, \boldsymbol{\tau}), K(\boldsymbol{\nu}, \boldsymbol{\tau}), N(\boldsymbol{\nu}, \boldsymbol{\tau})\right]^\top$ consist of forces and moments modelled as polynomials, given below:

$$\begin{aligned} X(\boldsymbol{\nu}, \boldsymbol{\tau}) =& X_{uuu}^2 + (1-t)T + X_{vr}vr + X_{vv}v^2 + X_{rr}r^2 + X_{\phi\phi}\phi^2 + \\ & c_{RX}F_N \sin\delta + (m + m_y)vr \end{aligned} \tag{A.8}$$

$$\begin{aligned} Y(\boldsymbol{\nu}, \boldsymbol{\tau}) =& Y_v v + Y_r r + Y_p p + Y_\phi \phi + Y_{vvv}v^3 + Y_{rrr}r^3 + Y_{vvr}v^2 r + \\ & Y_{vrr}vr^2 + Y_{vv\phi}v^2\phi + Y_{v\phi\phi}v\phi^2 + Y_{rr\phi}r^2\phi + \\ & Y_{r\phi\phi}r\phi^2 + (1 + a_H)F_N \cos\delta - (m + m_x)ur \end{aligned} \tag{A.9}$$

$$\begin{aligned} K(\boldsymbol{\nu}, \boldsymbol{\tau}) =& K_v v + K_r r + K_p p + K_\phi \phi + K_{vvv}v^3 + K_{rrr}r^3 + \\ & K_{vvr}v^2 r + K_{vrr}vr^2 + K_{vv\phi}v^2\phi + K_{v\phi\phi}v\phi^2 + \\ & K_{rr\phi}r^2\phi + K_{r\phi\phi}r\phi^2 - (1 + a_H)z_R F_N \cos\delta + \\ & m_x l_x ur - WG_M\phi \end{aligned} \tag{A.10}$$

$$\begin{aligned} N(\boldsymbol{\nu}, \boldsymbol{\tau}) =& N_v v + N_r r + N_p p + N_\phi \phi + N_{vvv}v^3 + N_{rrr}r^3 + \\ & N_{vvr}v^2 r + N_{vrr}vr^2 + N_{vv\phi}v^2\phi + N_{v\phi\phi}v\phi^2 + \\ & N_{rr\phi}r^2\phi + N_{r\phi\phi}r\phi^2 + (x_r + a_H x_H)F_N \cos\delta \end{aligned} \tag{A.11}$$

$$\begin{aligned} v_R &= g_a v + c_{Rr}r + c_{Rrrr}r^3 + c_{Rrrv}r^2 v \\ u_P &= \cos v((1 - w_p) + \tau((v + x_p r)^2 + c_{pv}v + c_{pr}r)) \\ J &= u_P U/(nD) \\ K_T &= 0.527 - 0.455J \\ u_R &= u_P \epsilon \sqrt{1 + 8k_k K_T/(\pi J^2)} \\ \alpha_R &= \delta + \arctan v_R/u_R \\ F_N &= -(6.13\Delta/(\Delta + 2.25))(A_R/L^2)(u_R^2 + v_R^2)\sin\alpha_R \\ T &= 2\rho D^4/(U^2 L^2 \rho)K_T n|n| \\ W &= \rho g \nabla/(\rho L^2 U^2/2) \end{aligned}$$

and parameters are given in Table A.1.

Term	Value	Term	Value	Term	Value
m	0.00792	m_x	0.000238	m_y	0.007049
I_x	0.0000176	I_z	0.000456	l_x	0.0313
J_x	0.0000034	J_z	0.000419	l_y	0.0313
L	175	α_y	0.05		
g	9.81	∇	21222	A_R	33.0376
Δ	1.8219	D	6.533	G_M	$0.3/L$
ρ	1025	t	0.175		
X_{uu}	-0.0004226	X_{vr}	-0.00311	X_{rr}	0.00020
$X_{\phi\phi}$	-0.00020	X_{vv}	-0.00386		
K_v	0.0003026	K_r	-0.000063	K_p	-0.0000075
K_ϕ	-0.000021	K_{vvv}	0.002843	K_{rrr}	-0.0000462
K_{vvr}	-0.000588	K_{vrr}	0.0010565	$K_{vv\phi}$	-0.0012012
$K_{v\phi\phi}$	-0.0000793	$K_{rr\phi}$	-0.000243	$K_{r\phi\phi}$	0.00003569
Y_v	-0.0116	Y_r	0.00242	Y_p	0
Y_ϕ	-0.000063	Y_{vvv}	-0.109	Y_{rrr}	0.00177
Y_{vvr}	0.0214	Y_{vrr}	-0.0405	$Y_{vv\phi}$	0.04605
$Y_{v\phi\phi}$	0.00304	$Y_{rr\phi}$	0.009325	$Y_{r\phi\phi}$	-0.001368
N_v	-0.0038545	N_r	-0.00222	N_p	0.000213
N_ϕ	-0.0001424	N_{vvv}	0.001492	N_{rrr}	-0.00229
N_{vvr}	-0.0424	N_{vrr}	0.00156	$N_{vv\phi}$	-0.019058
$N_{v\phi\phi}$	-0.0053766	$N_{rr\phi}$	-0.0038592	$N_{r\phi\phi}$	0.0024195
k_k	0.631	ϵ	0.921	x_R	-0.5
w_p	0.184	τ	1.09	x_p	-0.526
c_{pv}	0	c_{pr}	0	g_a	0.088
c_{Rr}	-0.156	c_{Rrrr}	-0.275	c_{Rrrv}	1.96
c_{RX}	0.71	a_H	0.237	z_R	0.033
x_H	-0.48				

Table A.1: Container parameter values

Appendix B

Experiment details

B.1 Vessel control with PSV

Experiment		Parameter	Value
Surge control	No added penalty	g_u	1
		g_v	4
	Added penalty	g_u	2/3
		g_v	2
		$c_{\dot{\psi}_c}$	0.1
Surge and heading control	No added penalty	g_u	2/3
		g_ψ	1
	Added penalty	g_u	2/3
		g_ψ	1
		$c_{\dot{\psi}_c}$	0.02

Table B.1: Parameters used in PSV control

B.2 Path following and surge control with two vessel types

Experiment	Parameter	Value
Path following	c_{pf}	1
	σ_{pf}	30
Path following with surge control	c_{pf}	0.5
	σ_{pf}	30
	c_u	0.5
	σ_u	0.1

Table B.2: Parameters used in path following for PSV

Experiment		Parameter	Value
Path following	No added penalty	c_{pf}	1
		σ_{pf}	30
	Added penalty	c_{pf}	1
		σ_{pf}	30
		$c_{\dot{\delta}\dot{\delta}}$	10
Path following with surge control	No added penalty	c_{pf}	0.5
		σ_{pf}	30
		a_u	0.5
		σ_u	0.25
	Added penalty	c_{pf}	0.5
		σ_{pf}	30
		c_u	0.5
		σ_u	0.25
		$c_{\dot{\delta}\dot{\delta}}$	10

Table B.3: Parameters used in path following for container vessel

B.3 Collision avoidance

Experiment	Parameter	Value
Simple head-on situation	c_{pf}	1
	σ_{pf}	30
	$c_{\dot{\delta}\dot{\delta}}$	10
	$c_{collision}$	10
Extended head-on situation	c_{pf}	1
	σ_{pf}	30
	$c_{\dot{\delta}\dot{\delta}}$	10
	$c_{collision}$	10
	c_{16}	5

Table B.4: Parameters used in collision avoidance

Appendix C

Additional plots showing thruster input to the platform supply vessel

C.1 Path following

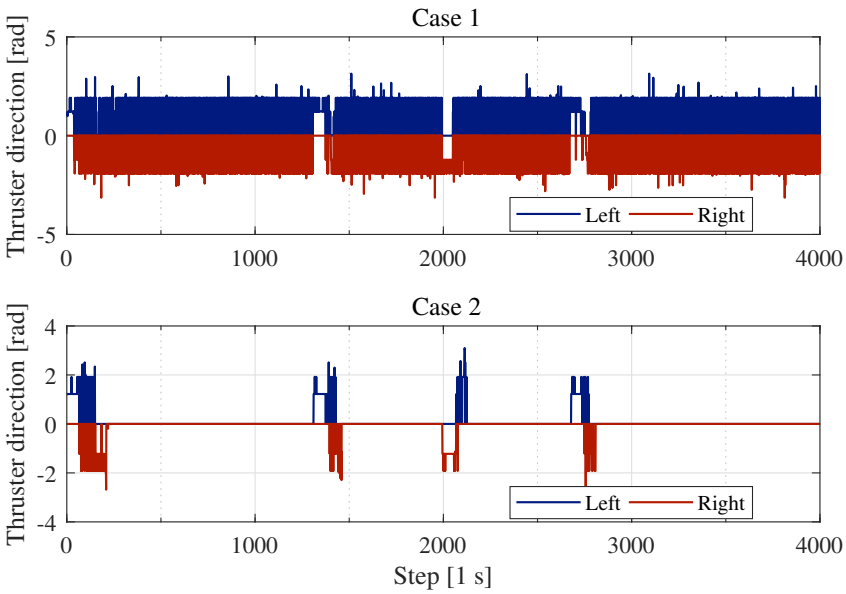


Figure C.1: Thruster angles of two DRL agents performing path following on a PSV

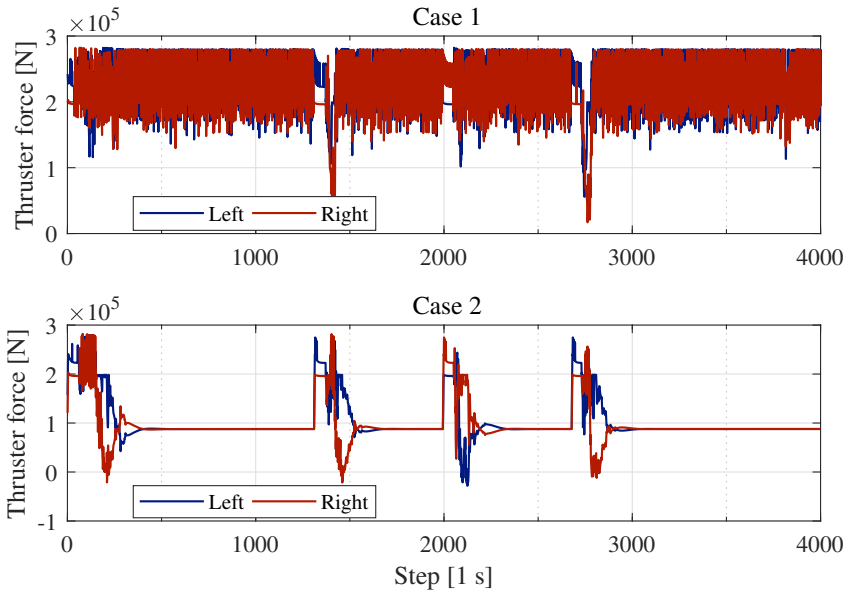


Figure C.2: Thruster forces of two DRL agents performing path following on a PSV

C.2 Path following and surge control

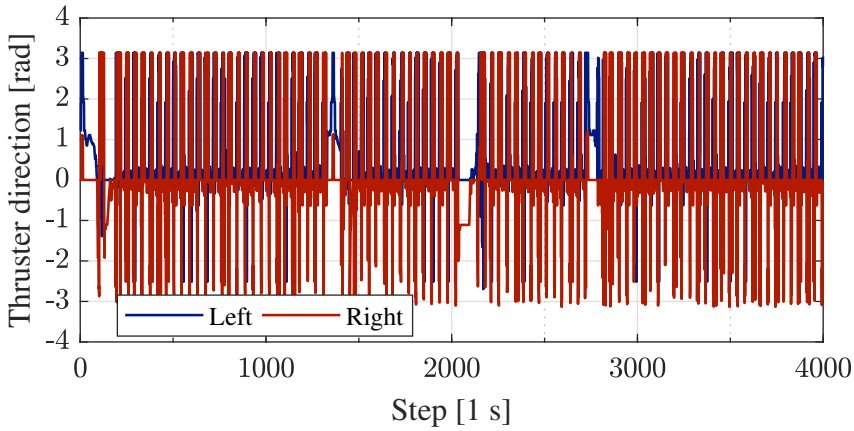


Figure C.3: Thruster angles of DRL agent performing path following and surge control on a PSV

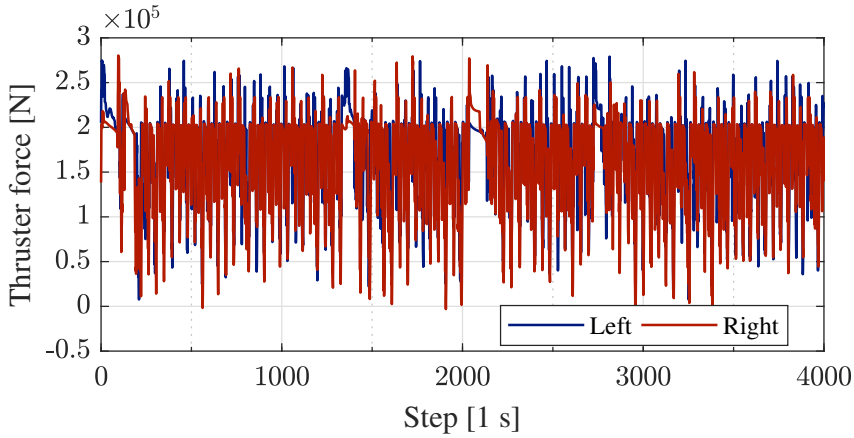


Figure C.4: Thruster forces of DRL agent performing path following and surge control on a PSV

Ingunn Johanne Vallestad

Path Following and Collision Avoidance for Marine Vessels with Deep Reinforcement Learning

NTNU
Norwegian University of
Science and Technology