NTNU
Norwegian University of
Science and Technology

End-to-end training for path following and control of marine vehicles

Andreas Bell Martinsen

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Anastasios Lekkas, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This thesis is the culmination of my work at the Norwegian University of Science and technology, under the supervision of Anastasios Lekkas, during the spring semester of 2018. The thesis is a summary of my findings, and the methods used, to apply Deep Reinforcement Learning (DRL) methods to perform path following for marine vessels in uncertain environments. The thesis was inspired by recent advances in the field of machine learning, and in particular DRL. Reinforcement Learning (RL) which has largely remained untouched as a viable control strategy due to high computational demand, and limits in terms of expressive power, has seen significant technological advances in recent years. This has caused a resurgence of interest in reinforcement learning, which is seen by some researchers as a stepping stone to the development of Artificial General Intelligence.

The thesis assumes the reader has prior knowledge of linear and nonlinear systems, and how to solve these types of systems numerically, as well as some knowledge on optimization. Familiarity with machine learning, and in particular function approximators such as Artificial Neural Networks (ANNs), is beneficial, however the theory needed to understand the specifics of DRL, will be presented. The thesis is intended to show how DRL can be used to perform path following on marine vessels, however it also gives an introduction to DRL, and is structured in such a way that it should give a reader with little or no knowledge on the topic both a theoretical and practical introduction.

The main contribution of this thesis is a method for applying DRL to the prob-

lem of path following in uncertain environments, including simulations on three different vessel models. The DRL algorithm which was used, was taken from the work of Lillicrap Et al. [1], and implemented in the Python programming language. Additionally the Tensorflow library for performing automatic differentiation, and the numpy library for performing Linear algebra were used. The vessel models that were used were ported from the Marine Systems Simulator (MSS) toolbox [2].

Acknowledgment

Research is never a solitary task, and my work has benefited significantly from the supervision of Anastasios Lekkas, whose motivation, and insights have helped me progress to the point where I am today. I would also like to thank my fellow graduate students for insightful and inspiring discussions, as well as my family for the love and support that they have given me throughout this journey.

04.06.2018

Andreas Bell Martinsen

Summary

The problem of following, or tracking a predefined path, has been a long standing problem in the control engineering community. In most cases, previous works utilized existing or newly-presented models to represent the vessel dynamics and kinematics before employing methods from nonlinear control theory for developing suitable cascading kinematic (i.e. guidance) and dynamic (i.e. control) laws for achieving the control objective. Although significant advances have been made in this field, the methods have mostly stayed the same, and usually require significant domain expertise, and experience to implement.

In recent years, with the advances of Machine Learning (ML), and in particular Deep Learning (DL), a number of problems that were previously thought impossible, have seen great success. One field that has shown great promise from these advances, is the field of Reinforcement Learning (RL), which in combination with DL has given rise to Deep Reinforcement Learning (DRL). These methods allow for learning to optimize decision making, by exploring environment and receiving evaluative feedback based on the performance. In this thesis we propose a general framework for using DRL algorithms in order to learn to optimize path following for marine vessels, by learning from exploration. Applying the proposed method on three different vessels, we were able to show how the DRL algorithm was able to learn to control the vessel, and optimize the control law in order to outperform the Line-of-Sight guidance law in its basic form, which is one of the most popular guidance methods used today.

Based on our results, DRL show great promise in performing complex guidance task, as it allows for optimizing a user specified performance measure, and doing so by requiring no knowledge of the internal dynamics of the vessel.

Sammendrag

Problemet med å følge eller spore en forhåndsdefinert sti har vært et langvarig problem i kontrollteori. I de fleste tilfeller utnytter tidligere arbeider eksisterende eller nylig presenterte modeller for å representere fartøydynamikken og kinematikken, før man anvender metoder fra ikke-lineær kontrollteori for å utvikle egnede kinematiske og dynamiske kontrollover for å oppnå kontrollmålet. Selv om det er gjort betydelige fremskritt i dette feltet har metodene for det meste forblitt det samme og krever vanligvis betydelig domenekompetanse og erfaring for å implementere.

I senere år har fremskritt innen maskinlæring (eng. Machine Learning), og særlig dyp læring (eng. Deep Learning), hatt stor suksess på en rekke problemer, som man tidligere ikke har kunnet løse. Et felt som har opplevd et stort løft fra disse fremskrittene, er feltet forsterkende læring (eng. Reinforcement Learning), som i kombinasjon med dyp læring har gitt opphav til dyp forsterkende lærning (eng. Deep Reinforcement Learning). Disse metodene optimaliserer beslutningstaking, gjennom å lære ved å utforske miljøet og motta tilbakemeldinger basert på ytelsen. I denne avhandlingen foreslår vi et generelt rammeverk for bruk av dyp forsterkende læring, for å optimalisere sti-følging for marine fartøy. Ved å anvende den foreslåtte metoden på tre forskjellige fartøy vil vi vise hvordan DRL-algoritmen kan lære å kontrollere fartøyet, og optimalisere kontrolloven. Dette resulterer i at DRL-algoritmen er i stand til å overgå Line-of-Sight kontrolloven som i sin grunnleggende form er en av de mest populære sti-følgings algoritmene som brukes i dag.

Basert på våre resultater, viser dyp forsterkende lærning seg å være en lovende

metode for sti-følging, da det muliggjør optimalisering av et brukerdefinert ytelsesmål, og gjør det uten å kreve at læringsalgoritmen har kjennskap til fartøyets indre dynamikk.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 2

ANN Artificial Neural Network. i, 2, 5, 6

CNN Convolutional Neural Network. 9

DDPG Deep Deterministic Policy Gradient. 3, 4

DL Deep Learning. v, 5, 7

DOF Degree Of Freedom. 28

DP Dynamic Programming. 14, 15

DRL Deep Reinforcement Learning. i, v, 18

MDP Markov Decision Process. 13

ML Machine Learning. v, 5

POMDP Partially Observable Markov Decision Process. 13

RL Reinforcement Learning. i, v, 2, 12

RNN Recurrent Neural Network. 9

SGD Stochastic Gradient Descent. 7

Symbols

α Learning rate. 7, 16

$\boldsymbol{\eta}$ Pose vector for marine vessel. 28

$f(x, u)$ Deterministic transition model, gives the next state when taking action u in state x . 13

$J(\boldsymbol{\theta})$ Loss function for parameter vector $\boldsymbol{\theta}$. 7

$\boldsymbol{\nu}$ Velocity vector for marine vessel. 28

$P(x'|x, u)$ Stochastic transition model, gives the probability of getting to state x' when taking action u in state x. 13

$Q(x, u)$ Action value function, giving the value of taking action u in state x. 16

$R(x, u)$ Reward function giving the reward of taking action u in the state x. 13

$\boldsymbol{\theta}$ Parameter vector for a function aproximator. 7

$\mathcal{U}$ Action constraint set. 13

$V(x)$ State value function, value of being in state x. 14

$\mathcal{X}$ State constraint set. 13

Chapter 1

Introduction

1.1 Background and motivation

Path following and path tracking for underactuated marine vehicles have attracted the attention of the marine control research community for a long time, resulting in a vast literature. The main task is to develop heading and speed control laws so as to follow or track a predefined path with minimum position error. More specifically, "following" pertains to the case where no temporal constraints are imposed, meaning the vessel should be on the path at any given time, whereas "tracking" implies the vessel should be at a specific point on the path at each time instant. In most cases, previous works utilized existing or newly-presented models to represent the vessel dynamics and kinematics, before employing methods from nonlinear control theory for developing suitable guidance and control laws for achieving the control objective.

In previous works, the guidance and control systems are often treated as a cascade, where guidance is the driving system, which in turn drives the control system responsible for driving the vessel actuators, this is illustrated in Figure 1. Using the cascaded structure, helps simplify the stability analysis, and has been used extensively in the past [5, 6, 7]. The problem however becomes more difficult when unknown environmental forces need to be taken into account [8, 9, 10, 11], this is often done by augmenting the original Line-of-Sight guidance law [12, 13]. In general this makes the path following problem challenging due to the large model

and environment uncertainties.

In the Artificial Intelligence (AI) community a theory for optimal system performance under uncertain conditions has been in development for a long time. This method, which is based on evaluative, rather than instructive feedback, is known as Reinforcement Learning (RL), or alternatively neuro-dynamic programming or approximate dynamic programming [14, 15, 16]. Reinforcement learning comes in different forms, which might, or not, include partial knowledge of the environment or the system. The most important element is the reward function which is directly related to the objective and dictates which actions are good and receive a reward, and which are undesired and receive a penalty. The algorithm then attempts to explore the space of possible solutions until an appropriate policy (series of control actions) that achieves the control objective in the best possible way is found. In the past, a challenge for RL has been to keep track of everything the algorithm learns about the system. Recently Deep Mind proposed a solution that turned out to be very efficient and involves using Deep Neural Networks (DNNs) for representation purposes, hence resulting in the field of Deep Reinforcement Learning (DRL). Methods in this field have given rise to some very interesting results. For problems with discrete action spaces we have seen these methods learn to play Atari games [17], and beating world champions at the game of GO [18]. For continuous action spaces we have seen the methods solve dozens of highly nonlinear physics problems such as robotic manipulation, bipedal locomotion, and game play [1, 19, 20].

1.2 Goal and method

The main goal of this thesis is to investigate how we can apply DRL to the complex problem of performing end-to-end path following and control of marine vessels exposed to unknown ocean currents. Our proposed method, illustrated in Figure 2, consists of two main components. The first component which is the environment, consists of a performance measure, which tells us how well we are doing at the path following task, and the vessel, which we are tasked with controlling by only being able to observe the state $\boldsymbol{x}$. The second component is the DRL agent, which consists of a control policy, mapping the input from the environment to a control action, and a value function, which gives a measure of how good a certain state is. Both the control policy and the value function are represented by Artificial Neural Networks (ANNs), and are learned by exploration. Our proposed method involves

shaping a performance measure in order to achieve the desired behaviour, augmenting the state representation to give the DRL agent useful information on the path following task, training the DRL agent in order to find a good policy, and choosing an architecture for the control policy and value function. In order to train the the agent, we use the DRL algorithm known as Deep Deterministic Policy Gradient (DDPG) [1].

The proposed approach constitutes an alternative solution to the path following problem by combining the ideas of optimizing performance similarly to model predictive control, with the benefits of being model-free. This sets it apart from most methods that are used today, as methods such as Line-of-Sight, which although it is model-free, does not optimize performance. While methods that optimize performance, such as model predictive control, are highly dependant on a good model. In addition, DRL allows for simplifying the control structure by creating a single control law, eliminating the need for a cascaded system such as the one illustrated in Figure 1.

This thesis is the culmination of the work of applying DRL methods to the path following problem. In addition to the thesis, the work has also resulted in two conference papers:

- One paper on straight-line path following [21], which has been accepted for the 11th IFAC Conference on Control Applications in Marine Systems, to be held in Opatija, Croatia from the 10th until 12th of September 2018.
- One paper on curved path following, which has been submitted to the OCEANS conference, to be held in Charleston from the 22nd until the 25th of October 2018.

1.3 Outline of report

The thesis is divided into five main chapters, we will start off in Chapter 2 by introducing some important concepts and notation used in the thesis. Here we will also give an introduction to machine learning, and in particular deep learning and artificial neural networks. We will also give a thorough introduction to reinforcement learning, as well as the main deep reinforcement learning methods, namely actor-only, critic-only, and actor-critic methods. Chapter 2 also discuss some of the main concepts of kinematics, motion control, and guidance for marine vessels, in

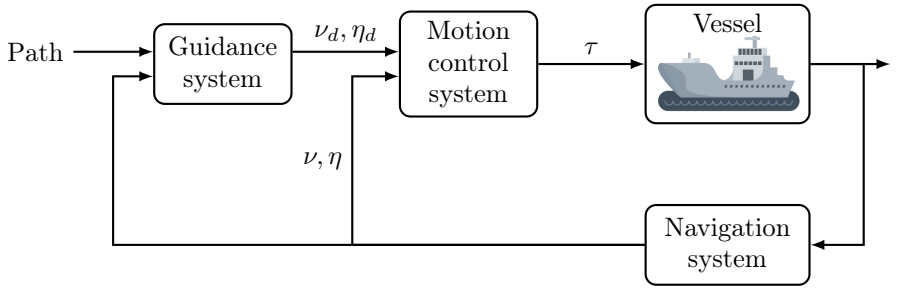


Figure 1: Block diagram of traditional cascaded guidance and control system.

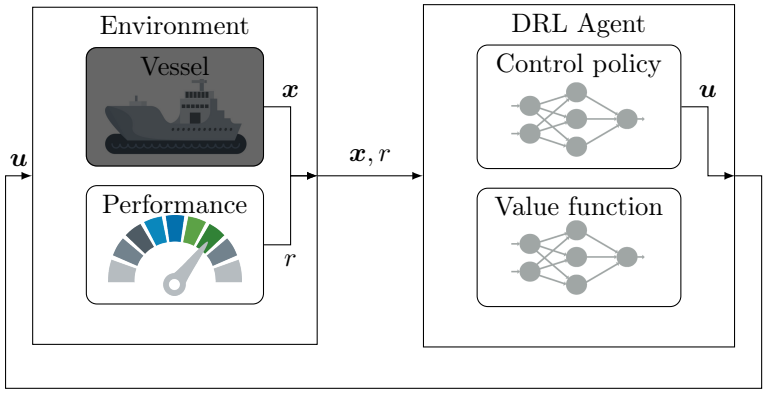


Figure 2: Illustration of the proposed control architecture, where vessel is considered a black box from, where only the given output can be observed.

order to give a good theoretical understanding. In Chapter 3 we will give a overview of how a Line-of-Sight algorithm can be implemented, as well as giving a comprehensive description of our proposed control algorithm, visualized in Figure 2, for performing end to end training and guidance of a marine vessel. In Chapter 4 we will present the main results from applying DDPG to our proposed method, and in Chapter 5 we will conclude the thesis.

Chapter 2

Background and theory

2.1 Machine Learning

The control approaches presented later in this thesis is based on Machine Learning (ML), which is a field in computer science which deals with the ability of computers to learn, without being explicitly programmed. ML is usually categorized into three main approaches, namely supervised learning, unsupervised learning and reinforcement learning. Our proposed method is based on reinforcement learning, however in this this section we will focus on supervised learning, as it has played an essential role in the recent developments in DRL.

In general, supervised learning can be described as learning from examples, where the machine is given training examples x which are a subset of the possible inputs $\mathcal{X}$, i.e. $x \in \mathcal{X}$. The machine is also given the known outputs y of an unknown function $y = f(x)$, and is then tasked with finding an approximation $\hat{f}(x)$ of the unknown function given the training data.

There are multiple approaches to learning function approximations depending on input and output data type (discrete, continuous binary), and expected complexity of the function (linear, nonlinear, logical), however we will mainly focus on Artificial Neural Network (ANN) and Deep Learning (DL) as these approaches are those best suited to working with continuous inputs and outputs, as well as highly nonlinear functions.

2.1.1 Artificial Neural Networks

Artificial Neural Network (ANN) is a learning algorithm which is inspired by biological neural networks found in animals in nature. One of the first implementations of neural networks was the perceptron introduced by Frank Rosenblatt[22]. The perceptron consists of a weighting vector $\boldsymbol{w}$ and a scalar bias b. Given an input vector $\boldsymbol{x}$ the output of the perceptron is given by

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}^\top \boldsymbol{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

This gives binary classification algorithm where the $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ represents the linear decision boundary. The simple single layer linear perceptron however has some significant drawbacks. The greatest of these drawback are that the perceptron learning algorithm does not terminate if the training set is not linearly separable, and the the output of the perceptron is limited to binary classes, these issues can be seen in Figure 3. In order to improve the perceptron, the idea of adding additional layers and activation functions were proposed in order to allow for continuous outputs, and the approximation of more complex nonlinear functions. These computing systems are what later became known as artificial neural networks.

In general artificial neural networks can be characterized by connected units or nodes, which are organized in layers. Each layer is represented by weighting matrix $\mathbf{W}$, a bias vector $\boldsymbol{b}$ and a activation function $f(\cdot)$. Given an input vector $\boldsymbol{x}$ the layer produces an output vector $\boldsymbol{y}$ by the the following operation.

$$\boldsymbol{y} = f(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.2}$$

where the activation function is applied to each element of the vector given by $\mathbf{W}\boldsymbol{x} + \boldsymbol{b}$. In a multi-layer ANN the input of each layer is the output of the previous layer. An example of an artificial neural network is given in Figure 4, and the function of each unit or node can be seen in Figure 5.

The framework described above gives a way of representing a nonlinear function with a given number of discrete inputs and outputs. Combining this with the universal function approximator theorem[23], it can be shown that a neural network with a single hidden layer, containing a finite number of neurons, can approximate any continuous function on compact subsets of $\mathbb{R}^n$. The challenge now becomes

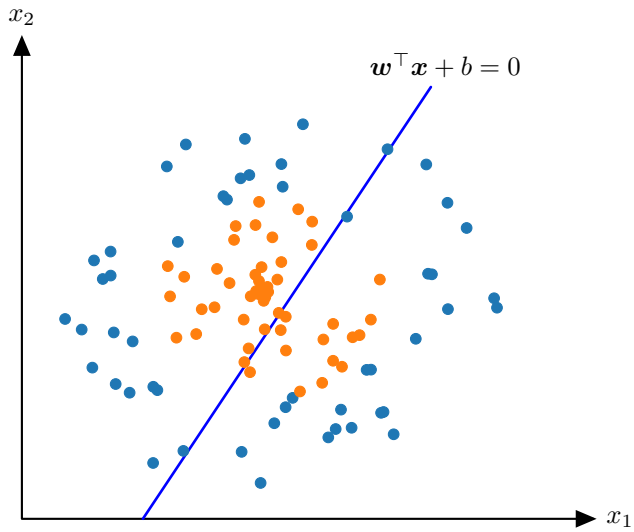


Figure 3: The perceptron learns a linear decision boundary for classifying the input data, this fails when the data is not linearly separable.

finding the weights and biases, which are collectively called parameters and denoted $\boldsymbol{\theta}$, in order to find the parameterization which best fits the artificial neural network to the target function. The most popular method for doing this is the use of gradient descent, where the gradient of loss function $J(\boldsymbol{\theta})$ with respect to the parameter vector $\boldsymbol{\theta}$, is used to augment the parameters in order to lower the loss on training data. Given the loss function, the gradient descent algorithm can be written as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.3}$$

where α is the learning rate. When gradient descent is performed on the loss function evaluated on randomly sampled data from from the training set, this is called Stochastic Gradient Descent (SGD).

2.1.2 Deep learning

Deep Learning (DL) can be considered an extension to the field of machine learning, in where a hierarchy of concepts enables a machine to learn complex concepts by building them out of simpler ones. Building concepts in such a way results in "deep" hierarchy with many layers where each consecutive layer represents more

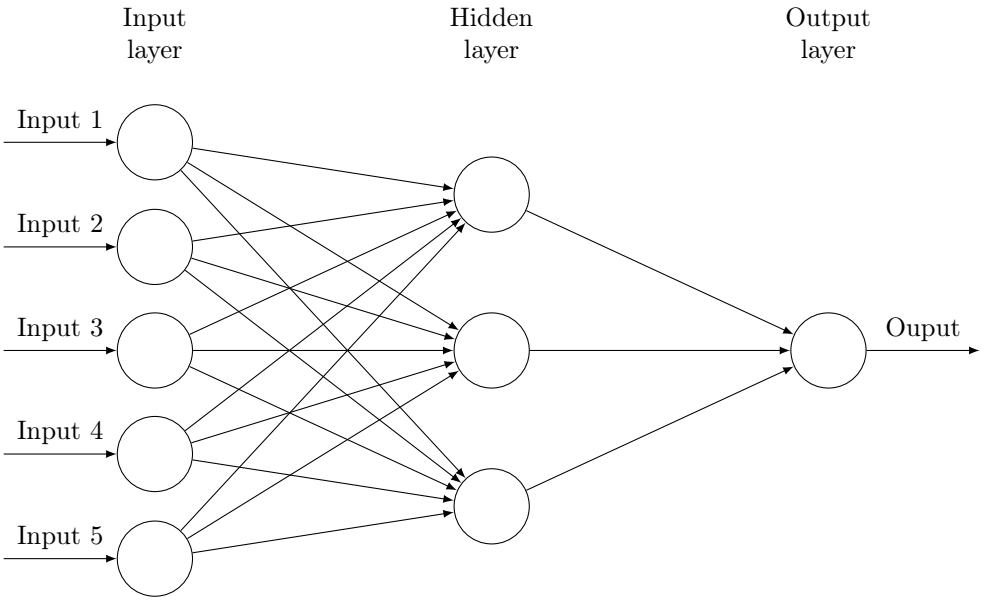


Figure 4: Artificial Neural Network

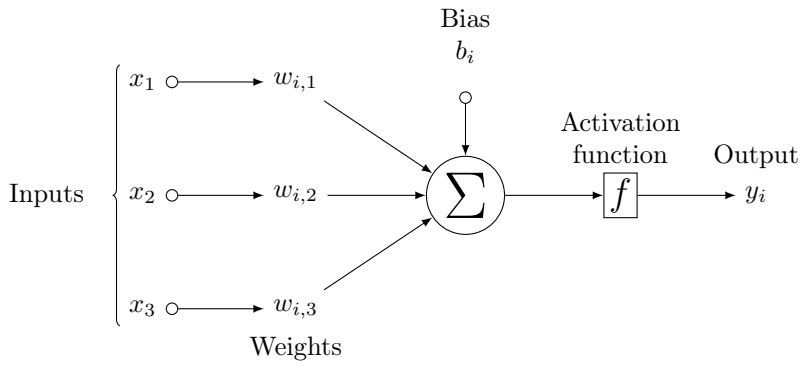


Figure 5: Unit or node in the artificial neural network

complex concepts. From a more mathematical point of view we can say that these techniques give more complex nonlinear parameterizations, and hence more complex functions can be modeled.

Where classical ML techniques rely on hand designed features to be selected as inputs to the learning algorithm, the additional layers in DL techniques allow for representation learning, where important features and abstractions are learned see Figure 6. This enabels DL to learn with little human intervention.

The simplest form of DL are Deep Feed Forward Networks or Deep Artificial Neural Networks, which are essentially ANNs, with multiple layers, as they consist of fully-connected layers with activation functions as illustrated in Figure 7. These are called feed forward because the information flows through the network from the input $\boldsymbol{x}$ through the intermediate hidden layers, until it reaches the output $\boldsymbol{y}$. In addition to fully-connected layers, seen in ANNs, DL also encompasses a number of other methods of creating network layers which handle specific kinds of data. Convolutional Neural Networks (CNNs) are one such method of adding convolution layers for processing data which is spatially connected. The architecture was first proposed as a method for classifying handwritten numbers from images[24]. An other technique is Recurrent Neural Network (RNN), for processing sequential data, such as dynamic and temporal data. RNNs work by saving information from previous passes in an internal hidden state, which it then uses in later passes.

2.1.3 Transfer learning

Transfer learning refers to the the situation where what has been learned in one setting, is used in order to improve generalization in an other, usually similar setting. Transfer learning is based on the idea of representation learning, in where the learning algorithm learns representations, or features, which can be useful when faced with a similar tasks. In problems such as image classification, a model which has learned to classify cats and dogs, can be adapted to classify ants and wasps, since the representations of edges and shapes learned in the first task, may be used in the second task. Similarly this can be used in RL, if the learning algorithm has been trained on one task, it can be retrained on a similar task in a shorter amount of time, by utilizing the representation learned from the previous task. In general transfer learning will significantly shorten training time for the new problem, as well as it requiring fewer training examples in order to learn a new task, however

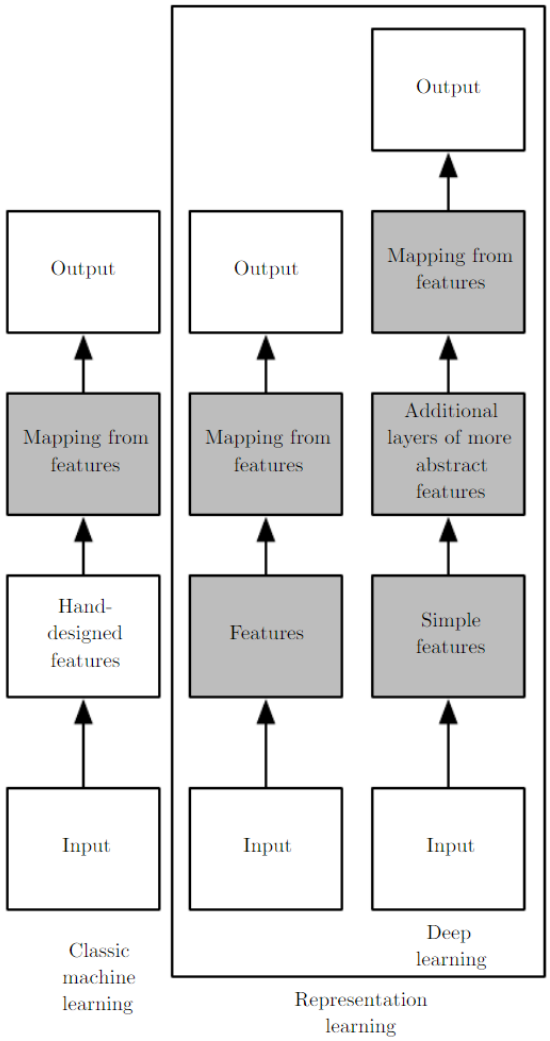


Figure 6: Flow chart of how AI systems relate [3]

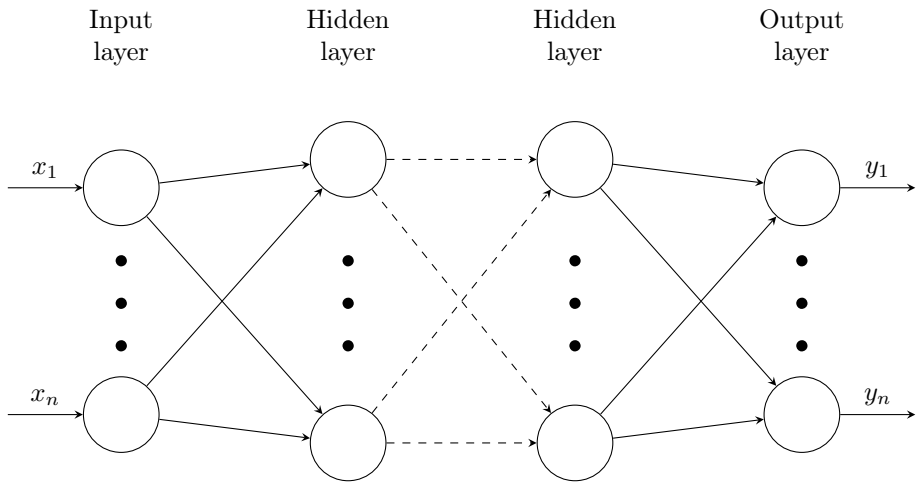


Figure 7: Illustration of deep feed forward neural network

the degree to which it will work is usually highly dependant on the similarity of the problems, and how well the learned representation generalizes to the new problem.

2.1.4 Training DL models

Training DL models is as mentioned earlier, usually done by performing SGD on the parameters of the network, where the gradients are calculated with respect to a loss function. The actual gradient calculation is performed using automatic differentiation, and backpropagation[25], where the gradient of the input of each mathematical operation in the network is computed as a function of the output, by utilizing the chain rule. Given the network $y = f(g(x))$, with the nested operations f, and g, backpropagation will go back layer by layer, in order to calculate the gradient as the product of the gradients of each layer with respect to its input in the following way.

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f'(g(x))g'(x) \tag{2.4}$$

Modeling the network as a computational graph from x to y, where each node represents a mathematical operation, the process of calculating the gradients using backpropagation is simply the passing of gradients back through the network in the reverse order, and multiplying with the gradient of the previous layer, this is illustrated in Figure 8.

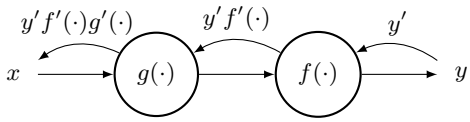


Figure 8: Illustration of backpropagation and forward pass of computational graph

Deep neural networks can often be difficult to train. Some of the major difficulties include overfitting, in where the network only learns how the correct responses to the training data, but does not learn any underlying concepts which it can use, when faced with new unseen examples. An other problem is underfitting, in where training is stopped before the network has reached its full potential. Training deep neural networks can also be difficult in terms of stability while training, causing oscillations, such that the network does not converge. Deep Learning proposes many different solutions to these problems, such as *weight regularization*[26], *dropout*[27], *early stopping*, *data augmentation*, and *noise injection*, which all seek to improve the performance of deep learning models.

2.2 Reinforcement learning

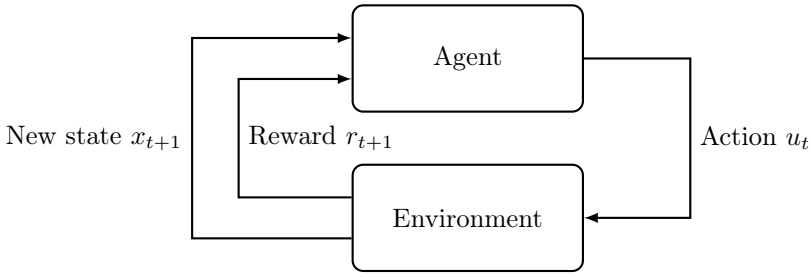


Figure 9: Visualization of reinforcement learning

The Reinforcement Learning (RL) problem is a way of framing the problem of a decision making agent interacting with its environment to learn how to act in the best possible way so as to obtain maximum reward. For this we frame the problem in terms of a Markov decision process presented in Section 2.2.1. Where the reinforcement learning agent interacts with its environment over a sequence of discrete time-steps as seen in Figure 9. In reinforcement learning the job of the

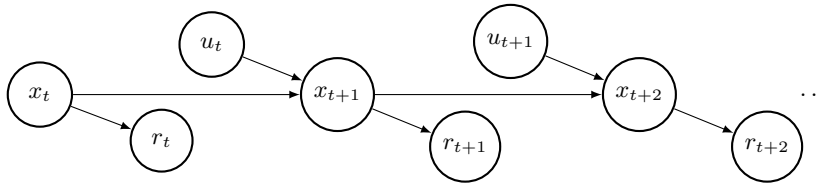


Figure 10: Markov decision process

agent is to learn how to act by continuing to interact with the environment, and learn what actions are good and what actions are bad with respect to the reward function, and in this way improve its performance.

2.2.1 Markov decision process

A sequential decision process which is fully observable, is in a stochastic environment, and has a transition model which satisfies the Markov property is called a Markov decision process (MDP). For simplicity we can denote the Markov decision problem as a tuple $(\mathcal{X}, \mathcal{U}, R, T)$ where $\mathcal{X}$ is the set of all states, $\mathcal{U}$ are the actions available to the agent (in general the actions available to the agent are dependant on the state, we denote this $\mathcal{U}(x)$ where $x \in \mathcal{X}$), R is the reward function (most often the reward is dependant on the state and action, we denote this $R(x, u)$), and T is the transition model given by the transition probability $P(x'|x, u)$. If the environment is deterministic the transition model can be written as $x' = f(x, u)$. The general structure of an MDP is visualized in Figure 10, where from each state an action is taken, leading to a new state and reward which can be observed [28].

Given a Markov decision process where we can only observe evidence of the state, and not the full state itself, we have a Partially Observable Markov Decision Process (POMDP). The umbrella world is a good example of a POMDP. In this example we imagine a person in a room with no windows, trying to figure out whether or not it is raining outside. The person is not able to directly observe the rain, however by studying the people entering the room, the person can make an educated guess based on whether or not the people entering are carrying umbrellas. Partially observable Markov decision processes are a useful concept in many situations. It should however be noted that all Partially observable Markov decision process can be converted into Markov decision processes by introducing a belief state which is inferred from the observable evidence.

2.2.2 Bellman equation

The Bellman equation, sometimes referred to as the dynamic programming equation, gives the value of a decision problem in a certain state in terms of the payoff from previous choices and the value of the remaining decision problem after making these choices, this is given according to Richard Bellman's Principle of Optimality:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision [29]

The justification for the Principle of Optimality is quite simple. If the policy we were following from state x_i was not optimal, then we would be able to reduce the cost by switching to an optimal policy for the subproblem when we reach state x_i. The Principle of Optimality suggests that an optimal policy can be constructed in a piecewise fashion, and hence the problem can be broken into simpler subproblems. This is known as Dynamic Programming (DP) and is the basis for solving optimization problems for maximizing the cumulative reward in a sequential decision problem.

Given a discrete deterministic decision problem we will go through the derivation of the Bellman equation. We start by looking at the value of being in a given state x as a value function $V(x)$, which we define as the sum of all the rewards we get while taking the action u_t at time t. This gives the following:

$$\begin{aligned} V(x_0) &= \sum_{t=0}^{\infty} R(x_t, u_t) \\ s.t. \quad & u_t \in \mathcal{U}(x_t), \quad x_{t+1} = f(x_t, u_t) \end{aligned} \tag{2.5}$$

where $R(x_t, u_t)$ is the reward from taking action u_t in state x_t. and $f(x_t, u_t)$ Is the transition model.

We now wish to maximize the reward over time, that is find the highest value for our current state, which we can do by always selecting the best action at every time step. We also assume impatience by introducing a discounting factor $0 < \gamma < 1$. This will give the optimal value function in terms of expected discounted future reward denoted V^* as:

$$\begin{aligned} V^*(x_0) &= \max_{\{u_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \gamma^t R(x_t, u_t) \\ s.t. \quad & u_t \in \mathcal{U}(x_t), \quad x_{t+1} = f(x_t, u_t) \end{aligned} \tag{2.6}$$

By using the Principle of Optimality, we separate the first decision from the rest of the decisions. This gives the equation:

$$\begin{aligned} V^*(x_0) = &\max_{u_0} \left\{ R(x_0, u_0) + \gamma \left[\max_{\{u_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \gamma^{t-1} R(x_t, u_t) \right] \right\} \\ &s.t. \quad u_t \in \mathcal{U}(x_t), \quad x_{t+1} = f(x_t, u_t) \end{aligned} \tag{2.7}$$

At first glance this seems to have made the problem uglier by separating the first decision from the rest. However by noticing that what is inside the square brackets is the value of the decision problem at $t = 1$ we can rewrite the problem as:

$$V^*(x) = \max_{u \in \mathcal{U}(x)} \left\{ R(x, u) + \gamma V^*(f(x, u)) \right\} \tag{2.8}$$

which is the Bellman equation in it's simplest form.

The Hamilton–Jacobi–Bellman equation is an extension of the bellman equation for continuous time systems. The equation is given as a differential equation, and for the simple continuous time system on the form:

$$\dot{x} = f(x, u)$$

the Hamilton–Jacobi–Bellman equation is given as:

$$\dot{V}^*(x, t) = \max_{u \in \mathcal{U}} \left\{ \frac{\mathrm{d}V^*(x, t)}{\mathrm{d}x} f(x, u) + R(x, u) \right\} \tag{2.9}$$

Solving either the Bellman equation or the Hamilton–Jacobi–Bellman equation means finding the value function $V^*(x)$ which gives optimal value of the objective as a function of the state x.

With a known transition model, and a finite and discrete state and action space, these problems can be solved using DP. This approach involves making an initial guess for the value function for all states, usually 0. Using the initial guess, the Bellman equation is used to solve for all the states using the initial value function guess. The updated value is then used for finding yet another value function. This is performed iteratively until the value function converges, and an optimal value function is found. Two popular methods for performing this are value iteration and policy iteration.

2.2.3 Temporal difference learning

In the previous section we saw that it was possible to use DP algorithms for solving sequential decision problems. However this tends to be the exception. In most cases a numerical solution is necessary. This is due to complexity of the problem causing the number of states and actions to become overwhelming, and requires excessive computational demands in order to solve the problem. Richard Bellman himself called this the "curse of dimensionality". Dynamic programming also requires the full model of the system to be known, which in most cases we do not. Temporal difference learning is one solution which tries to solve these problems by combing the ideas of dynamic programming and Monte Carlo methods. The main idea behind Temporal difference learning is the use of experience to solve the prediction problem, this means that as the agent interacts with the environment it gains experience which it can use to estimate the value function. The simplest Temporal difference method, known as $TD(0)$, is given as:

$$V(x) \leftarrow V(x) + \alpha\big[R(x', x, u) + \gamma V(x') - V(x)\big] \tag{2.10}$$

or alternatively when transitioning from state-action to state-action, we get the following update rule

$$Q(x, u) \leftarrow Q(x, u) + \alpha\big[R(x', x, u) + \gamma Q(x', u') - Q(x, u)\big] \tag{2.11}$$

where $Q(x, u)$ is the expected value of taking action u in state x. Temporal difference learning is based on taking an action u in the state x and observing the new state x' and the reward $R(x', x, u)$. It then uses this information to update its estimate of the state-value $V(x)$ or action-value $Q(x, u)$ of state x, as a weighted sum of its new experience and its old estimate of the value. Note α is the weighing factor called the learning rate.

The n-step prediction for temporal difference is an other way of updating the temporal difference equation, where n denotes how many future discounted returns we should take into account when computing the updated value. For a state reward sequence $x_t, r_{t+1}, x_{t+1}, \ldots, x_T, r_T$ with $r_{t+1} = R(x_{t+1}, x_t, u_t)$ and the discount rate γ, the backed up discounted reward can be written as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \gamma^{T-t-1} r_T \tag{2.12}$$

where T is the last step of the episode. For the one step backup this can be written as the reward plus the discounted estimated value of the next state given as:

$$G_t^{(1)} = r_{t+1} + \gamma V(x_{t+1})$$

Intuitively this makes sense, since $V(x_{t+1})$ is an estimate of it's future discounted reward. Generalizing this we can write the n-step truncated return as:

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \gamma^n V(x_{t+n}) \tag{2.13}$$

This gives the n-step update equation as:

$$V(x) \leftarrow V(x) + \alpha\big[G_t^{(n)} - V(x)\big] \tag{2.14}$$

This method however is rarely used as it requires waiting n steps to compute the update, which makes it inconvenient to implement, as well as being problematic in control applications for large n. It does however have a some benefits, which is that the value from later states are more quickly propagated back to earlier states, which reduces the number of iterations required to get sufficiently close to the value function of a given policy.

The general $TD(0)$ algorithm has been shown to converge to the optimal solution V^* [30] of the Bellman equation with a probability of one under the following assumptions.

1. The values are stored in a table (tabular method)
2. The learning rate $\alpha_t(x_t)$ satisfies the following conditions: $\sum_t \alpha_t(x_t) = \infty$, and $\sum_t \alpha_t(x_t)^2 < \infty$ (these conditions are upheld if $0 < \alpha_t(x_t) < 1 \; \forall t, x_t$)
3. $Var(r) < \infty$ (the variance of the reward is bounded, which holds true if r is bounded $r < \infty$)

This is a useful result, as it can be extended to tabular temporal difference methods, unfortunately the result does not hold for approximate methods such as deep reinforcement learning.

2.2.4 Deep reinforcement learning

We have so far discussed methods which assume we have value functions and policies that can be represented as tables with one entry for each state or state-action pair. This approach is however limited in that it only works for a finite number

of state or state-actions pairs. These methods also suffer form the "curse of dimensionality", that is that as the number of states or state-action pairs increase, the number of entries in the table increases exponentially. This becomes not only a problem in terms of memory, but also in terms of the computational time to accurately fill the value function and policy tables. A natural question that arises is whether or not it is possible to generalize experience from a limited subset of the state space in such a way that we find a good approximation for the entire state space. Fortunately the answer to this question is yes, these methods are often called function approximators, and have been extensively studied.

The popularity of approximate RL methods have increased significantly the last few years as computational power has increased, and the use of function approximators such as neural networks, and in particular deep neural networks have proven to be able to learn tasks such as playing ATARI games at super human levels[31], defeating the world champion in the game of GO[18], and learning advanced robotics manipulation tasks[32]. In this section we will look at some different approximate methods where we use function approximators to find the approximations of the value function and policy using ANNs and DL. These methods are collectively referred to as Deep Reinforcement Learning (DRL) methods.

DRL methods can be split into three main categories depending on architecture, these three categories are actor-only methods, critic-only methods, and actor-critic methods. The methods are categorized according to architecture, in where actor-only methods only consist of learning a policy $\pi(x)$, critic-only methods consist of learning a action-value function $Q(x, u)$, and actor-critic methods learn both a policy and value function. Due to the critic-only method, only having a action-value function available. It must enumerate through all possible actions u in order to find the optimal action which has the highest action-value. This means that the critic-only architecture is only suitable for discrete finite action spaces. Actor-critic and actor-only methods are both able to learn policies with continuous action spaces. For most dynamical systems, the action space is continuous, this is also true for path following and control, and we will therefor focus on these methods.

Policy gradient methods

Policy gradient methods are approximate methods which aim to find an estimate for the optimal policy function $\pi^*(x)$ by optimizing a parametrized policy function

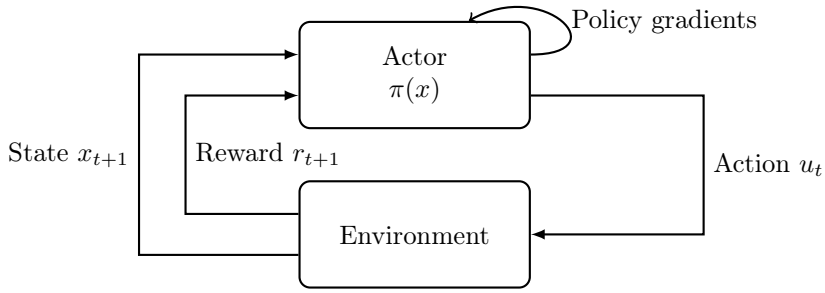


Figure 11: Actor-only architecture

with respect to the expected long term cumulative reward. The optimization itself takes place by using gradient descent to iteratively improve the policy with respect to the expected return. the policy gradients method is an on policy method, which manes it needs to follow the policy it is learning in order to improve.

In the policy gradient methods we parametrize the set of policies by a vector $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_n]$. We further assume that the policy can be written as a probability distribution $\pi_{\boldsymbol{\theta}}(u|x)$ which is the probability of taking action u in state x given the parameter vector $\boldsymbol{\theta}$. This is quite practical as it makes the policy inherently stochastic. One example where a stochastic policy is useful is in card games, where it is beneficial to be unpredictable, which a stochastic policy will allow. We will also see that modeling the policy as a probability distribution naturally incorporates exploratory actions which helps us train and optimize the policy. The general objective of reinforcement learning is as discussed earlier, to maximize the expected discounted reward. this can be expressed as maximizing expected cumulative discounted reward, giving the following objective function:

$$J(\boldsymbol{\theta}) = E\left[\sum_{t} \gamma^t R(x_t, u_t)\right]_{u_t \sim \pi(x_t;\boldsymbol{\theta})}$$

where the action u_t at time t is sampled from the policy π_θ. We can rewrite this problem as finding the parameters θ^* such that the policy gives the actions which

maximize the expected reward. This gives the optimization problem:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} E\left[\sum_t \sum_{u_t \sim \pi(x_t;\boldsymbol{\theta})} \gamma^t R(x_t, u_t)\right] \tag{2.15}$$

To find the optimal parameter vector $\boldsymbol{\theta}^*$ we now come to the heart of policy gradients which is to use gradient descent to iteratively improve the estimate of $\boldsymbol{\theta}$, where the gradient descent algorithm is given as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.16}$$

The question now becomes how to compute the gradient of the objective function. In order to do so there are tow main methods, finite difference approximation, and direct policy differentiation.

Finite Difference Policy Gradient

The simplest approach to finding policy gradients, is by using the finite difference method, which is one of the oldest policy gradient approaches. This is done by perturbing $\boldsymbol{\theta}$ slightly by an amount ϵ in the the kth dimension to find an approximation to the partial derivative with respect to θ_k. Using the forward difference method this can be written as:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_k} \approx \frac{J(\boldsymbol{\theta} + \epsilon v_k) - J(\boldsymbol{\theta})}{\epsilon}$$

where v_k is a unit vector with 1 in the kth component and 0 otherwise. We can alternatively use the backwards difference or central difference method. Using the partial derivatives we can compute the gradient as follows:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix} \tag{2.17}$$

The finite difference method uses n evaluations to compute the gradient for the forward and backward difference method, and $2n$ evaluations when using the central difference method. The finite difference method is often inefficient and noisy, however it works for arbitrary policies, even if the policy parameterization is not differentiable.

Direct policy differentiation

A more common approach today is direct policy differentiation where we find an analytical expression for the gradients. Due to the advance of deep neural nets, one can easily find the derivatives of parameters in a computation graph by using automatic differentiation and the chain rule during backpropagation. If we assume that we know the gradient $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(x, u)$, and that the policy gives likely hood ratios, we can exploit the following identity:

$$\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(x, u) = \pi_{\boldsymbol{\theta}}(x, u) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(x, u)$$

Using this we can use standard differential calculus[33] to compute the gradient of the objective function as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E\left[\left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(x_t, u_t) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}\right)\right)\right] \tag{2.18}$$

With a Monte-Carlo approach, where we compute multiple different trajectories or episodes given a policy, we can approximate the expected value of the gradients as the average of the gradients over the simulation, and we get the policy gradient as follows:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(x_{i,t}, u_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}\right)\right)$$

where N are the number episodes we average over, and T are the number of steps in the episodes. This update method has an intuitive appeal to it. Looking at the gradient we see that it is given as product of the return and the gradient of the probability of taking the action that lead to the return. This means that during gradient descent, if an action sequence resulted in a high return, then we move the parameters in the direction which make taking the actions more probable. In this sense we move the parameters in the direction of taking an action, proportional to the return, and hence the policy moves in the direction which favours actions which give higher returns.

In order to reduce the variance of the gradient estimator, a constant baseline can be subtracted from the gradients. The optimal baseline can be computed to be:

$$b = \frac{\sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(x_{i,t}, u_{i,t})\right)^2 \left(\sum_{t=1}^{T} \gamma^t r_t\right)}{\sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(x_{i,t}, u_{i,t})\right)^2}$$

which is just the expected reward, but weighted by gradient magnitudes. An approximation of the value function for each state, or the average reward may also be used as a baseline. The average baseline is given as:

$$b = \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma^t r_t$$

While the average baseline reduces the complexity in comparison to the optimal baseline, it has a larger variance. The third option is to approximate the value function, and use it as a baseline, this will usually be the best baseline, however it increases complexity. Using a value function approximation as a baseline, we get the following:

$$b(x_{i,t}) = V(x_{i,t}) \tag{2.19}$$

where the baseline becomes a function of the state $x_{i,t}$. When using a baseline the gradient of the loss is given as the following:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(x_{i,t}, u_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - b \right) \right) \tag{2.20}$$

The method described above is known as the REINFORCE algorithm [34] and is summarized in Algorithm 1

Algorithm 1 REINFORCE

1: **repeat**
2: Sample trajectories using $\pi_{\boldsymbol{\theta}}(x, u)$
3: Compute gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ from Equation 2.20
4: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: **until** sufficiently good policy

There are other methods which build on these the classical policy gradients above, such as Natural Policy Gradients [35], Trust Region Policy Optimization (TRPO)[36], and most recently Proximal Policy Optimization (PPO)[20]. These methods extend the classical policy gradient methods in order to improve the performance and stability.

Actor-critic methods

In the previous sections we looked at policy approximation. Policy approximation is an actor-only method since it learns a policy directly from the experience it

gets. Similarly to actor-only methods, it is possible to create critic-only methods, in which a value or action-value function approximation is used in order to approximate the value of a state. Critic-only methods however are limited in that in order to find the policy it is necessary to iterate through all possible actions, to find the action which results in the highest state-value. This however is not practical for problems with continuous action spaces.

Combining the policy approximation from actor-only methods, and value function approximation from critic-only methods, we get the actor-critic methods which learn both a value function and a policy. The actor-critic methods are based on actor (policy) which acts on the environment, given a state, and a critic which finds the value of being in a certain state when following the policy of the actor. The actor in turn gets feedback from the critic to improve the policy, while the critic continues to evaluate the performance of the actor, an illustration of the actor-critic architecture is given in Figure 12. The actor-critic methods in this way aim

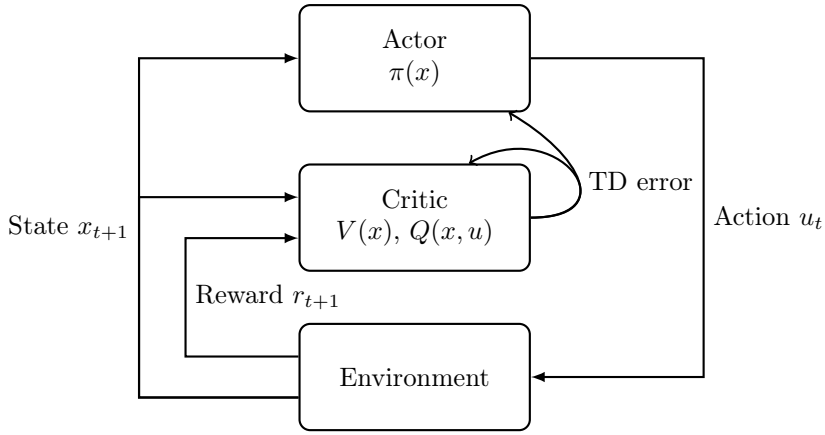


Figure 12: The actor-critic RL architecture consists of an actor which chooses which action to take, and a critic, which evaluates the actor by looking at the difference in expected and actual performance, in terms of the temporal difference error.

to combine the advantages of both actor-only methods such as policy gradients, and critic-only methods. Similarly to actor-only methods, actor-critic methods are capable of producing continuous actions, while the large variance in actor-only methods is reduced in the actor-critic methods by learning from the critic. Today

some of the most popular actor-critic algorithms are based on the use of policy gradients and value function approximation using ANNs [37]. These methods have been successfully used to play complex first person shooter video games [38, 39] and perform complex humanoid locomotion tasks [40].

Actor-critic methods are most often based on approximate methods for both the policy, and the value function. These methods parameterize the critic value function $V_{\boldsymbol{\theta}_c}(x)$ in terms of a parameter vector $\boldsymbol{\theta}_c$. Similarly the actor policy $\pi_{\boldsymbol{\theta}_a}(x)$, which is usually stochastic, is parameterized by the parameter vector $\boldsymbol{\theta}_a$. Since the goal of the actor-critic method is to find the best policy by learning from the critic, the critic needs some way of evaluating the policy. Using the Bellman equation, we find the error between the right, and left hand side of the equation, which gives us the temporal difference error:

$$\delta = R(x', x, u) + \gamma V_{\boldsymbol{\theta}_c}(x') - V_{\boldsymbol{\theta}_c}(x) \tag{2.21}$$

Since the temporal difference error should converge to 0 as the value function approximate converges to the true value function, the temporal difference error becomes a natural choice for updating the critic. Using the squared temporal difference error as a loss function, we get

$$J_c(\boldsymbol{\theta_c}) = \frac{1}{2}\delta^2 \tag{2.22}$$

we now assume the value function estimate of the next state in the temporal difference error is fixed and independent of the parameter vector $\boldsymbol{\theta}_c$. Taking the gradient of the loss function will then give.

$$\nabla_{\boldsymbol{\theta}_c} J_c(\boldsymbol{\theta_c}) = \delta \nabla_{\boldsymbol{\theta}_c} \delta = \delta \nabla_{\boldsymbol{\theta}_c} V_{\boldsymbol{\theta}_c}(x) \tag{2.23}$$

When using gradient descent on the value function parameters $\boldsymbol{\theta}_c$, we can write the update law as follows:

$$\boldsymbol{\theta}_c \leftarrow \boldsymbol{\theta}_c + \alpha_c \delta \nabla_{\boldsymbol{\theta}_c} V_{\boldsymbol{\theta}_c}(x)$$

where α_c is the learning rate of the critic. For the actor we wish to minimize the cost or maximize the reward of the policy, which means we need the gradient of the policy parameters with respect to the cost or reward. From the derivation of the policy gradients with direct policy differentiation, a one step horizon and a value

function as baseline, we get

$$\nabla_{\boldsymbol{\theta}_a} J(\boldsymbol{\theta}_a) \approx \frac{1}{N} \sum_i \nabla_{\boldsymbol{\theta}_a} \log \pi_{\boldsymbol{\theta}_a}(x_{i,t}, u_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - V(x_t) \right) \tag{2.24}$$

From the the Bellman equation we have that

$$\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} = r_t + V(x_{t+1}) \tag{2.25}$$

denoting the next state $x_{t+1} = x'$, and using a batch size $i = 1$, we get the following approximation for the gradient of the cost function.

$$\nabla_{\boldsymbol{\theta_a}} J(\boldsymbol{\theta_a}) \approx \delta \nabla_{\boldsymbol{\theta_a}} \log \pi_{\boldsymbol{\theta_a}}(x) \tag{2.26}$$

Using gradient descent we can write the parameter update equation for the actor as follows:

$$\boldsymbol{\theta_a} \leftarrow \boldsymbol{\theta_a} + \alpha_a \delta \nabla_{\boldsymbol{\theta_a}} \log \pi_{\boldsymbol{\theta_a}}(x)$$

where α_a is the learning rate of the actor. Intuitively we can think of this as guided descent, where we multiply the gradient with the temporal difference error in order to guide the gradient descent in such a way that we move towards actions which give a positive temporal difference error, i.e. actions that perform better then the current policy with respect to the value function.

The resulting actor-critic algorithm for the episodic case is describe in Algorithm 2. The method can also be extended to train on minibatches of experience which often stabilizes training, since noise in the experience is suppressed when normalizing over a batch of training data. Asynchronous actor-critic methods have also been proposed[41], and allow for training on multiple distributed machines, which has been shown to significantly speed up training on complex problems.

Deep Deterministic Policy Gradients

The method we settled on using for the path following tasks, is a hybrid method of actor-critic methods combined with some of the ideas of critic-only methods, called Deep Deterministic Policy gradients (DDPG)[1]. The general architecture of DDPG is similar to that of the actor-critic architecture. DDPG approximates both a policy and a value function, the major difference being the use of an action-value

Algorithm 2 Actor-critic

repeat
  Initialize x (first state in episode)
  while x not terminal **do**
    $u \sim \pi_{\boldsymbol{\theta_a}}(x)$
    Take action u, and observe x', r
    $\delta \leftarrow r + \gamma V_{\boldsymbol{\theta}_c}(x') - V_{\boldsymbol{\theta}_c}(x)$
    $\boldsymbol{\theta}_c \leftarrow \boldsymbol{\theta}_c + \alpha_c \delta \nabla_{\boldsymbol{\theta}_c} V_{\boldsymbol{\theta}_c}(x)$
    $\boldsymbol{\theta}_a \leftarrow \boldsymbol{\theta}_a + \alpha_a \delta \nabla_{\boldsymbol{\theta_a}} \log \pi_{\boldsymbol{\theta_a}}(x)$
    $x \leftarrow x'$
  end while
until training terminated

function $Q(x, u)$ in stead of a value or state-value function $V(x)$. Using a state-value function allows for the critic to learn the value of all action in all states, meaning the algorithm can learn the optimal policy even when it is not performing the optimal policy, i.e. the algorithm is off policy.

Given a action-value function $Q_{\boldsymbol{\theta}_Q}(x, u)$, parameterized by a parameter vector $\boldsymbol{\theta}_Q$. And a policy $\pi_{\boldsymbol{\theta}_\pi}(x)$, parameterized by a parameter vector $\boldsymbol{\theta}_\pi$ we define the loss function as the squared temporal difference error, similar to the actor-critic algorithm, as the following.

$$J_Q(\boldsymbol{\theta_Q}) = \frac{1}{2}\delta_t^2 = \frac{1}{2}\left(r_t + Q_{\boldsymbol{\theta}_Q}(x_{t+1}, \pi_{\boldsymbol{\theta}_\pi}(x_{t+1})) - Q_{\boldsymbol{\theta}_Q}(x_t, u_t)\right)^2 \tag{2.27}$$

For computing the loss of the policy, we utilize the fact that the action-value function encodes the value of taking a action in a given state, and hence we can use the action-value function directly as a loss function. This means we can compute the gradient of the the value function with respect to the policy parameterization $\boldsymbol{\theta}_\pi$ for a given state as

$$\begin{aligned} \nabla_{\boldsymbol{\theta}_\pi} J_\pi(\boldsymbol{\theta}_\pi) &= \nabla_{\boldsymbol{\theta}_\pi} Q_{\boldsymbol{\theta}_Q}(x, \pi_{\boldsymbol{\theta}_\pi}(x)) \\ &= \nabla_u Q_{\boldsymbol{\theta}_Q}(x, u) \nabla_{\boldsymbol{\theta}_\pi} \pi_{\boldsymbol{\theta}_\pi}(x) \end{aligned} \tag{2.28}$$

DDPG additionally adds several methods for stabilizing the training, such as the use of experience replay [31, 42], in where the training data (x, u, r, x') consisting of the state, action, and observed reward and next state, are sampled uniformly from the replay memory. This method increases sample efficiency, and allows for

experiences to be sampled multiple times. Additionally, experience replay allows for batch training on temporally uncorrelated mini batches, which increases computational efficiency, and increases stability during training.

In addition to experience replay, DDPG also utilizes soft parameter updates[1, 31], where a separate target policy and action-value function parameterization, $\boldsymbol{\theta}_{\pi'}$ and $\boldsymbol{\theta}_{Q'}$ are used to calculate the target values, and slowly track the learned network.

$$\begin{aligned} \boldsymbol{\theta}_{\pi'} &= (1-\tau)\boldsymbol{\theta}_{\pi'} + \tau\boldsymbol{\theta}_{\pi} \\ \boldsymbol{\theta}_{Q'} &= (1-\tau)\boldsymbol{\theta}_{Q'} + \tau\boldsymbol{\theta}_{Q} \end{aligned} \tag{2.29}$$

This means that the target values are constrained to change slowly, and further improving the stability of learning by avoiding feedback.

For training the DDPG policy, the algorithm must explore the environment. Following the learned policy directly would not lead to any exploration due to the policy being deterministic. In order to incorporate exploration, noise sampled from a noise process $\mathcal{N}$ is added to the policy.

$$u_t = \pi_{\boldsymbol{\theta}_\pi}(x_t) + \mathcal{N}_t \tag{2.30}$$

Combining everything we get the DDPG algorithm given in Algorithm 3

2.3 Guidance and control of marine vessels

2.3.1 Kinematics

In order to simulate and control marine vessels, it is necessary to study the kinematics, which treats the geometrical aspects of motion. In most control approaches, the kinematics are a central part of the controller, for DRL however a model is not necessary, however knowledge of the kinematics can be useful, when design a reward function, and as we will be working with a simulated vessel, a kinematic model is needed in order to perform simulations.

Using the SNAME notation in Table 1, the motion of a 3 Degree Of Freedom (DOF) surface vessel can be represented by the pose vector $\boldsymbol{\eta} = [x, y, \psi]^\top \in \mathbb{R}^2 \times \mathbb{S}$, and velocity vector $\boldsymbol{\nu} = [u, v, r]^\top \in \mathbb{R}^3$. Here, (x, y) describe the Cartesian position in a local North-East-Down (NED) reference frame, ψ is yaw angle, (u, v) is the body fixed linear velocities, and r is the yaw rate, an illustration is given in Figure 13.

Algorithm 3 Deep Deterministic Policy Gradients

Randomly initialize critic $Q_{\boldsymbol{\theta}_Q}(x,u)$ and actor $\pi_{\boldsymbol{\theta}_\pi}(x)$ with weights $\boldsymbol{\theta}_Q$ and $\boldsymbol{\theta}_\pi$.
Initialize target network $\boldsymbol{\theta}_{Q'}$ and $\boldsymbol{\theta}_{\pi'}$ with weights $\boldsymbol{\theta}_{Q'} \leftarrow \boldsymbol{\theta}_Q$, $\boldsymbol{\theta}_{\pi'} \leftarrow \boldsymbol{\theta}_\pi$
Initialize replay buffer
for episode $= 1, ..., M$ **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state x_1
    for $t = 1, ..., T$ **do**
        Select action $u_t = \pi_{\boldsymbol{\theta}_\pi}(x_t) + \mathcal{N}_t$
        Take action u_t and observe reward r_t and new state x_{t+1}
        Store transition (x_t, u_t, r_t, x_{t+1}) in replay buffer
        Sample N transitions (x_t, u_t, r_t, x_{t+1}) from replay buffer
        Set $y_i = r_i + \gamma Q_{\boldsymbol{\theta}_{Q'}}(x_{i+1}, \pi_{\boldsymbol{\theta}_{\pi'}}(x_{i+1}))$ for $i \in 1 \dots N$
        Update critic by minimizing loss: $\frac{1}{N}\sum_i \left(y_i - Q_{\boldsymbol{\theta}_Q}(x_i, u_i)\right)^2$
        Update policy with: $\frac{1}{N}\sum_i \nabla_{u_i} Q_{\boldsymbol{\theta}_Q}(x_i, u_i) \nabla_{\boldsymbol{\theta}_\pi} \pi_{\boldsymbol{\theta}_\pi}(x_i)$
        Update critic target network: $\boldsymbol{\theta}_{\pi'} = (1-\tau)\boldsymbol{\theta}_{\pi'} + \tau\boldsymbol{\theta}_\pi$
        Update actor target network:$\boldsymbol{\theta}_{Q'} = (1-\tau)\boldsymbol{\theta}_{Q'} + \tau\boldsymbol{\theta}_Q$
    end for
end for

DOF		Force/Moment	Velocity	Position/Angle
1	Surge	X	u	x
2	Sway	Y	v	y
3	Heav	Z	w	z
4	Roll	K	p	ϕ
5	Pitch	M	q	θ
6	Yaw	N	r	ψ

Table 1: The SNAME notation [4] for 6-DOF marine vessels

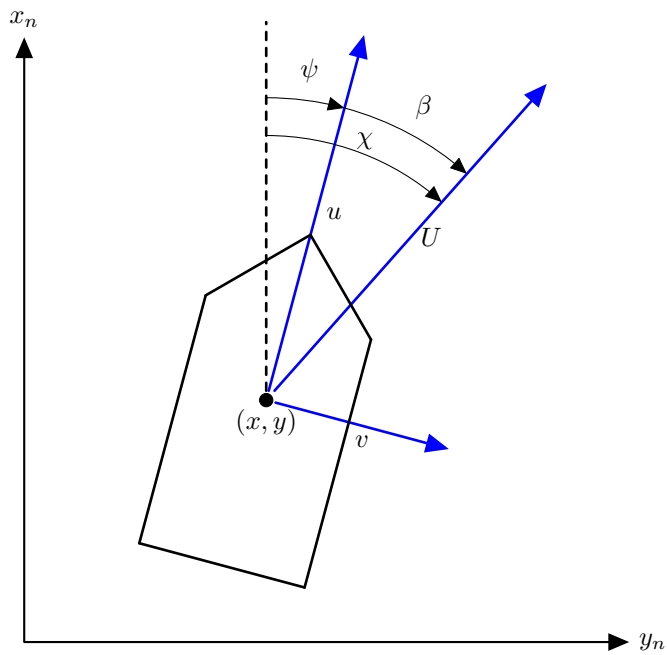


Figure 13: 3-DOF vessel centered at (x, y), with surge velocity u, sway velocity v, heading ψ, course χ and sideslip β, in a North-East-Down (NED) reference frame.

From [43] we can describe a 3-DOF vessel model on vectorial form as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{2.31}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{2.32}$$

where $\boldsymbol{M} \in \mathbb{R}^{3\times3}$, $\boldsymbol{C}(\boldsymbol{\nu}) \in \mathbb{R}^{3\times3}$, $\boldsymbol{D}(\boldsymbol{\nu}) \in \mathbb{R}^{3\times3}$ and $\boldsymbol{\tau}$ are the inertia matrix, Coriolis matrix, dampening matrix and control input vector respectively. The rotational matrix $\boldsymbol{R}(\psi) \in SO(3)$ is given by

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

With the edition of ocean currents, both the rigid-body and hydrostatic terms, as well as the hydrodynamic terms must be taken into consideration, this gives the following 3-DOF vessel model.

$$\underbrace{\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu}}_{\text{rigid-body and hydrostatic terms}} + \underbrace{\mathbf{M}_A\dot{\boldsymbol{\nu}}_r + \mathbf{C}_A(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r}_{\text{hydrodynamic terms}} = \tau \tag{2.34}$$

where $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c$ is the relative velocity vector, when given the current velocity vector $\boldsymbol{\nu}_c$. Assuming irrotational constant ocean currents, this simplifies to the following model

$$\mathbf{M}\dot{\boldsymbol{\nu}}_r + \mathbf{C}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r = \tau \tag{2.35}$$

where $\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A$ and $\mathbf{C} = \mathbf{C}_{RB} + \mathbf{C}_A$.

For maneuvering, some additional terms are important to know these include the course, and sideslip angle, as well as the speed of the marine vessel. The speed U of a marine vessel is defined as the absolute velocity of the vessel. From the model above, this can be calculated as:

$$U = \sqrt{u^2 + v^2} \tag{2.36}$$

The sideslip angle β is defined as the angel of the speed vector in the vessel body frame.

$$\beta = \sin^{-1}\left(\frac{v}{U}\right) \tag{2.37}$$

The sideslip angular velocity $\dot{\beta}$ can then be computed using the following:

$$\dot{\beta} = \frac{\mathrm{d}}{\mathrm{d}t}\sin^{-1}\left(\frac{v}{U}\right) = \frac{\frac{\dot{v}}{U} - \frac{v\dot{U}}{U^2}}{\sqrt{1 - \frac{v^2}{U^2}}} \tag{2.38}$$

The course angle χ is defined as the angle of the absolute velocity vector in the NED frame, which is the heading angle ψ pluss the sideslip angle χ.

$$\chi = \psi + \beta \tag{2.39}$$

A visual representation of the course angle χ, and sideslip angle β, as well as the speed U is given in Figure 13.

2.3.2 Guidance of marine vehicles

Guidance can be defined as "*The process for guiding the path of an object towards a given point, which in general may be moving.*"[44] In the simplest case guidance is the generation of a reference trajectory for time varying trajectory tracking, or a path for time invariant path following. In this section we will focus on the path following task, which involves controlling a vessel such that it converges to a desired path.

Straight-line path following

The simplest path following problem, is straight-line path following. In straight-line path following the objective of the control algorithm, is to directly follow a straight line, in other words, we want the x, and y direction of the vessel to lie on the path. Consider a straight-line path defined by two waypoints, $\boldsymbol{p_k} = [x_k, y_k]^\top$ and $\boldsymbol{p_{k+1}} = [x_{k+1}, y_{k+1}]^\top$, define in the the North east down coordinate system. Then the angle of the path defined by the waypoints can be computed by:

$$\gamma_p = \text{atan2}(y_{k+1} - y_k, x_{k+1} - x_k) \tag{2.40}$$

where $\text{atan2}(y, x)$ is the four quadrant version of $\arctan(y/x)$. The rotation matrix from the path fixed referance frame to the the NED frame is than given as

$$\mathbf{R}_p(\gamma_p) = \begin{bmatrix} \cos(\gamma_p) & -\sin(\gamma_p) \\ \sin(\gamma_p) & \cos(\gamma_p) \end{bmatrix} \tag{2.41}$$

Using this we can find the position of the vessel in the path centered coordinate frame given the vessel position $\boldsymbol{p} = [x, y]^\top$ in the NED frame as:

$$\boldsymbol{\epsilon} = [x_e, y_e]^\top = \mathbf{R}_p^\top (\boldsymbol{p} - \boldsymbol{p}_k) \tag{2.42}$$

where x_e is the along-track distance tangential to the path, and y_e is the cross-track error, which is normal to the path, this is illustrated in Figure 14.

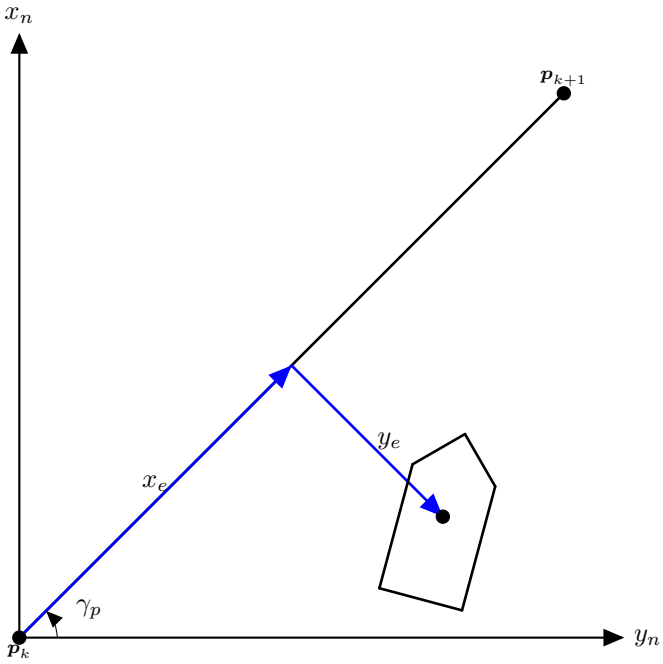


Figure 14: Path centered coordinate transformation

For the path following task, the objective is for the vessel to converge to the desired path, this means that we want the cross-track error to go to zero as time goes to infinity $\lim_{t\to\infty} y_e(t) = 0$. One of the most popular methods of achieving this behaviour is the use of Line-of-Sight (LOS) guidance, where the vessel is tasked with traveling from one waypoint to the next while constrained to following the straight-line path between the two waypoints. From Fossen 2011 [43] the steering law can be selected as:

$$\chi_d = \chi_p + \chi_r \tag{2.43}$$

where the path tangential angle which ensures the vessel travels along the path is given as:

$$\chi_p = \gamma_p = \text{atan2}(y_{k+1} - y_k, x_{k+1} - x_k) \tag{2.44}$$

and the path relative angle which ensures the vessel converges to the path, can be chosen as:

$$\chi_r = \text{atan}\left(\frac{-y_e}{\Delta_{y_e}}\right) \tag{2.45}$$

where $\Delta_{y_e} > 0$ represents the lookahead distance. This steering law can be interpreted as a saturating control law where the cross-track error $y_e \in \mathbb{R}$ is mapped to $\chi_r \in [-\pi/2, \pi/2]$ and where Δ_{y_e} is a tunable gain. The performance of this method depends on the performance of the motion control system which maps the heading to rudder actuation, as well as the Δ_{y_e} gain which controls the convergence speed to the path. We will later propose a different method of performing both the motion control, and guidance of a vessel using DRL, however the LOS approach is a good benchmark approach, as it is one of the most widely used guidance approaches.

Curved path following

Curved path following generalizes the ideas of straight-line path following, by introducing arbitrary paths. It also significantly increases the complexity of the problem, as the curvature of the path must be taken into consideration. In general this can cause problems for underactuated vessels, as some paths may be infeasible to converge to, due to the path exceeding the physical limits of the vessel. In this section we will discuss the dynamics of curved path following.

For curved paths, it is useful to describe the path in terms of a parameterization. These parameterized paths describe a geometric curve parameterized by a continuous path variable. For a surface vessel, a path parameterized by ω can be

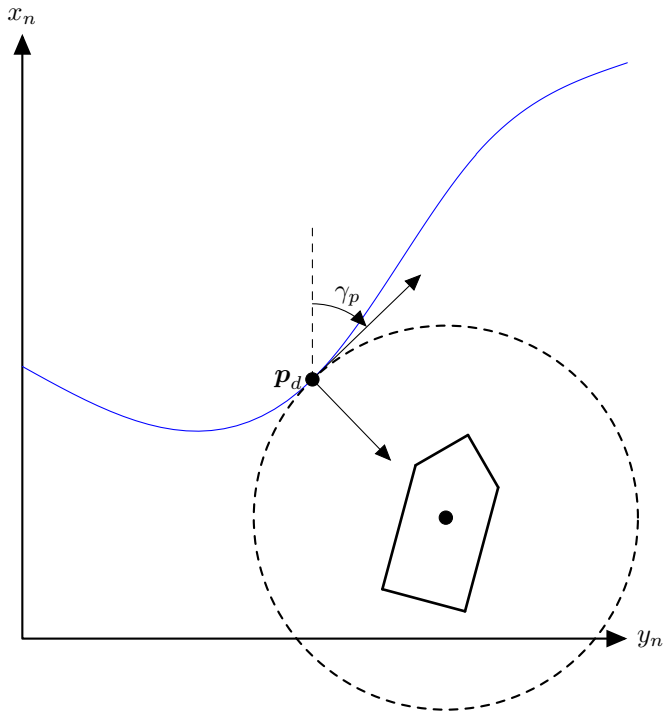


Figure 15: The origin of the path centered coordinate frame $\boldsymbol{p}_d$ is the point on the path the shortest Euclidean distance from the vessel, and the orientation γ_p is given by the directional derivative of the path at $\boldsymbol{p}_d$

written as

$$\boldsymbol{p}_d(\omega) = [x_d(\omega), y_d(\omega)]^\top \tag{2.46}$$

where $x_d(\omega)$ and $y_d(\omega)$ describe the position of the vessel in the horizontal plane for values of ω. The first and second order derivatives of a path $\boldsymbol{p}_d(\omega)$ with respect to the parameterization ω are denoted $\boldsymbol{p}'_d$ and $\boldsymbol{p}''_d$, and are useful in expressing the curvature of the path. The goal of path following for curved paths is the geometric task of converging to the path.

$$\lim_{t\to\infty} \boldsymbol{p}(t) - \boldsymbol{p}_d(\omega(t)) \tag{2.47}$$

Similarly to straight-line path following, the objective of the curved path following task is to minimize the cross-track error. Since the path is no longer a straight line, a new problem arises, which is to find the point along the curved path from which to calculate the cross-track error. A natural point to select is the point along the path which is closest to the vessel, this is illustrated in Figure 15. The problem of finding the position, can be written as finding the path variable ω which minimizes $(x - x_d(\omega))^2 + (y - y_d(\omega))^2$, given the vessel position $\boldsymbol{p} = [x, y]^\top$. This can be expressed as the following optimization problem.

$$\min_{\omega} f(\omega) = (x - x_d(\omega))^2 + (y - y_d(\omega))^2 \tag{2.48}$$

In order to solve the optimization problem above, there are many approaches one can take, however a good method is to use an iterative gradient descent approach, by approximating a Taylor series expansion of the function at a given point. Using the second order Taylor series expansion, we get the Newton method[45], giving the following iterative update rule

$$\omega_{k+1} = \omega_k - \left(\frac{\mathrm{d}^2 f(\omega)}{\mathrm{d}\omega^2}\right)^{-1} \frac{\mathrm{d}f(\omega)}{\mathrm{d}\omega}\bigg|_{\omega=\omega_k} \tag{2.49}$$

where k denotes the iteration. The first and second order derivative of the objective function are given as follows.

$$\frac{\mathrm{d}f(\omega)}{\mathrm{d}\omega} = -2(x - x_d(\omega))x'_d(\omega) - 2(y - y_d(\omega))y'_d(\omega) \tag{2.50}$$

$$\frac{\mathrm{d}^2 f(\omega)}{\mathrm{d}\omega^2} = 2\left(-(x - x_d(\omega))x''_d(\omega) - (y - y_d(\omega))y''_d(\omega) + x'_d(\omega)^2 y'_d(\omega)^2\right) \tag{2.51}$$

Using the newton method, we can compute the optimal ω iteratively until the estimate is sufficiently close to the optimal value, which is determined by how close the

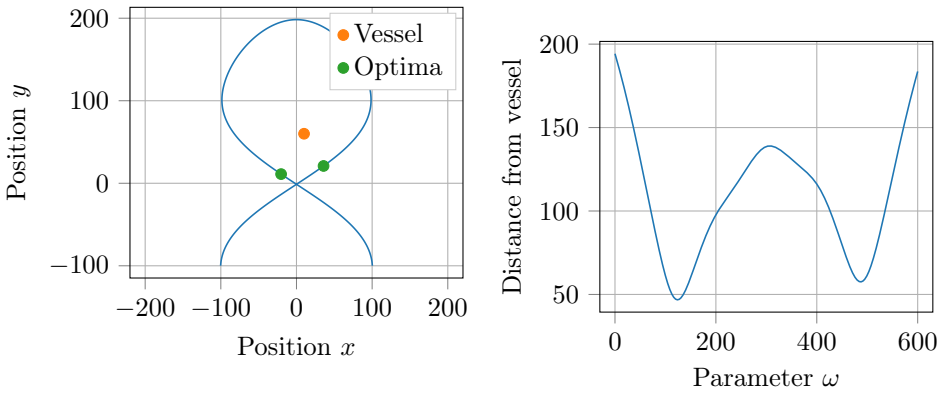


Figure 16: Calculating parameter value ω, as the local minimum ensures the parameter value only changes slightly at each time step, and avoids large jumps from one section of the path to another.

derivative gets to zero. It should be noted that the Newton method is only guaranteed to find local optima, this is however beneficial, since the vessel will only move a short distance from one time step to the next. Using the previous parameter value as the initial starting point for finding the next parameter value ensures that the closest local minimum is selected as the parameter value. This ensures the path is tracked smoothly, and avoids jumps in ω if we get closer to a different part of the path then the part of the path that is currently being tracked, this is illustrated in Figure 16.

When tracking the path, it is useful to know the evolution of the the parameter ω. Given a desired speed along the path $U_d(t)$, the parameter evolution $\dot{\omega}(t)$ is given by the following.

$$\begin{aligned} U_d(t) &= \sqrt{\frac{\mathrm{d}x_d(\omega(t))}{\mathrm{d}t}^2 + \frac{\mathrm{d}y_d(\omega(t))}{\mathrm{d}t}^2} = \sqrt{x_d'(\omega)^2 + y_d'(\omega)^2}\dot{\omega}(t) \\ \Rightarrow \dot{\omega}(t) &= \frac{U_d(t)}{\sqrt{x_d'(\omega)^2 + y_d'(\omega)^2}} \end{aligned} \tag{2.52}$$

The angle of the path is given by the direction the path is traveling. Given the derivatives of the position with respect to the parameterization $y_d'(\omega)$ and $x_d'(\omega)$

the path angle $\gamma_p(\omega)$ in the NED frame, can be computed using the following equation.

$$\gamma_p(\omega) = \text{atan2}(y'_d(\omega), x'_d(\omega)) \tag{2.53}$$

The path angular velocity with respect to the parameterization ω, gives a measure of the curvature of the path. The path angular velocity $\gamma'_p(\omega)$ is given by:

$$\frac{\mathrm{d}\gamma_p}{\mathrm{d}\omega} = \frac{x'_d(\omega)y''_d(\omega) - y'_d(\omega)x''_d(\omega)}{x'_d(\omega)^2 + y'_d(\omega)^2} \tag{2.54}$$

The path angular velocity with respect to time, gives a desired yaw rate for which we want the vessel to follow. Knowing the time evolution of the parameterization $\dot{\omega}$ the path angular velocity $\dot{\gamma_p}$ can be found using the following equation:

$$\dot{\gamma_p} = \frac{\mathrm{d}\gamma_p}{\mathrm{d}\omega}\frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{x'_d(\omega)y''_d(\omega) - y'_d(\omega)x''_d(\omega)}{x'_d(\omega)^2 + y'_d(\omega)^2}\dot{\omega} \tag{2.55}$$

The position of the vessel in the path centered coordinate frame given the vessel position $\boldsymbol{p} = [x, y]^\top$ in the NED frame is computed in the same way as for straight-line pats:

$$\boldsymbol{\epsilon} = [x_e, y_e]^\top = \mathbf{R}_p(\gamma_p(\omega))^\top(\boldsymbol{p} - \boldsymbol{p}_d(\omega)) \tag{2.56}$$

where $\mathbf{R}_p$ is the rotation matrix from the NED reference frame to the path centered reference frame given as:

$$\mathbf{R}_p(\gamma_p(\omega)) = \begin{bmatrix} \cos(\gamma_p(\omega)) & -\sin(\gamma_p(\omega)) \\ \sin(\gamma_p(\omega)) & \cos(\gamma_p(\omega)) \end{bmatrix} \tag{2.57}$$

Similarly, the path relative velocity can be calculated as follows:

$$\begin{aligned} \dot{\boldsymbol{\epsilon}} = [\dot{x_e}, \dot{y_e}]^\top &= \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{R}_p(\omega)^\top(\boldsymbol{p} - \boldsymbol{p}_d(\omega)) \\ &= \dot{\mathbf{R}}_p(\omega)^\top(\boldsymbol{p} - \boldsymbol{p}_d(\omega)) + \mathbf{R}_p(\omega)^\top(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_d(\omega)) \\ &= \mathbf{R}_p(\omega)^\top\left(\begin{bmatrix} 0 & \dot{\gamma_p} \\ -\dot{\gamma_p} & 0 \end{bmatrix}(\boldsymbol{p} - \boldsymbol{p}_d(\omega)) + \dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_d(\omega)\right) \end{aligned} \tag{2.58}$$

2.4 Tools and libraries

For implementing the algorithms and models used in evaluating the performance of the proposed control schemes, the python programming language was used. Additionally the scientific library scipy, and the numerical library numpy were used in order in order to do high level numerical calculations.

In order to implement the function approximators used to represent the policy and value functions, the automatic differentiation library Tensorflow[46] was used. Tensorflow gives a high level interface for building DL models as computation graphs, it also implements automatic differentiation and backpropagation, as well as a number of optimization techniques which can be used in order to efficently build and train train DL models.

The vessel models used to perform the simulations were ported over to python from the Marine Systems Simulator (MSS) toolbox [2]. The models that were ported include the a 3-DOF Mariner vessel [47], a 3-DOF Tanker vessel [48], and a 4-DOF Container vessel [49]. The vessel models are included in Appendix A.

Chapter 3

Design and implementation

3.1 Line-of-Sight guidance

In order to gauge the performance of the new guidance method which we will outline later, we need to be able to compare to the guidance systems which are used today. The most commonly used guidance system is the Line-of-Sight (LOS) method, in the simplest case the LOS algorithm is used to control the heading of the vessel such that the vessel converges to the path. In order to do so it is necessary with a motion control system, which can control the rudder in order to track a desired heading. This gives a cascading control architecture as seen in Figure 17.

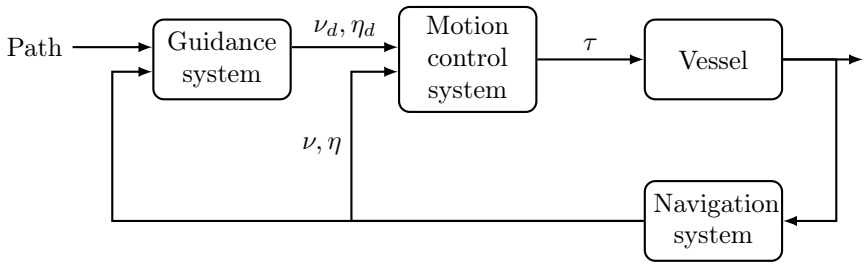


Figure 17: Illustration of the cascaded guidance architecture.

3.1.1 Motion control

For motion control systems such as heading and speed autopilots, a common approach is to model the simplified decoupled dynamics. A good approximation of the

decoupled yaw dynamics for a surface vessel, is the Nomoto model. The Nomoto model can be derived from the linearized maneuvering model, described in the handbook of Marine Craft Hydrodynamics and Motion Control [43]. For vessel heading the first order Nomoto model is given as:

$$\frac{r}{\delta}(s) = \frac{K}{1+Ts} \tag{3.1}$$

where r is the yaw rate, and δ is the rudder angle. In the time domain the yaw dynamics are given as:

$$\begin{aligned} T\dot{r} + r &= K\delta \\ \dot{\psi} &= r \end{aligned} \tag{3.2}$$

In order to identify the parameters of the first order Nomoto model we can use the step response of the vessel, to find the model parameters which best fit the vessel response. Given a constant rudder angle $\delta = \delta_0$ we can express the yaw rate $r(t)$, of the Nomoto model as:

$$r(t) = e^{-\frac{t}{T}} r(0) + [1 - e^{-\frac{t}{T}}] K\delta_0 \tag{3.3}$$

Using least squares curve fitting it is then possible to fit the observed response to the Nomoto model respones, by tuning the model parameters T and K. For a constant rudder angle $\delta_0 = 5°$, and the different vessel models traveling at typical cruising speed, we get the response seen in Figure 18 for the various models, with the following parameters:

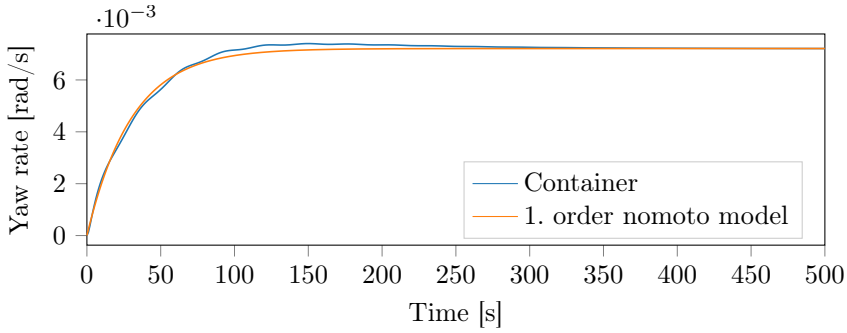
- Container ship: $K = 0.0826$, and $T = 30.4942$
- Tanker: $K : -0.0676$, and $T : 98.5742$
- Mariner: $K : 0.0846$, and $T : 19.0006$

Using the Nomoto models, we can design a simple state feedback linearizing yaw regulator. Using the yaw error $\tilde{\psi} = \psi - \psi_d$, and yaw rate error $\tilde{r} = r - r_d$ with ψ_d and r_d as the desired yaw and yaw rate respectively, we choose the following control law.

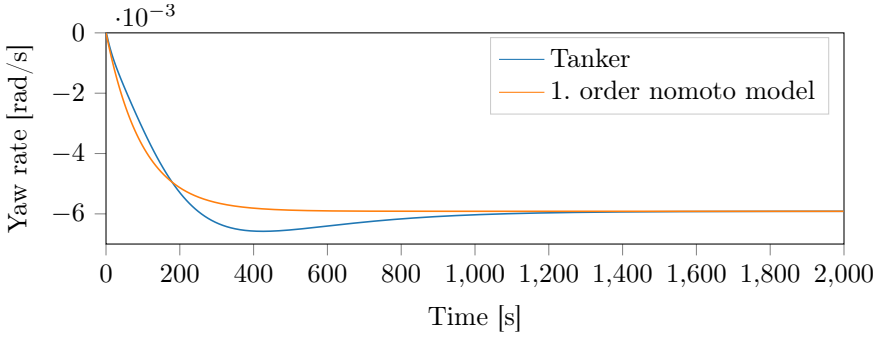
$$\delta_c = \frac{T}{K}[-K_p\tilde{\psi} - K_d\tilde{r}] + \frac{1}{K}r \tag{3.4}$$

Combining the control law with the dynamics of the Nomoto model we get the following error dynamics:

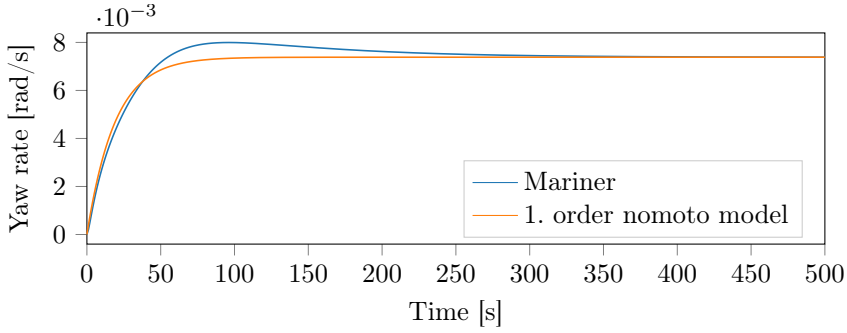
$$\begin{aligned} \dot{\tilde{\psi}} &= \tilde{r} \\ \dot{\tilde{r}} + K_d\tilde{r} + K_p\tilde{\psi} &= 0 \end{aligned} \tag{3.5}$$



(a) Container ship



(b) Tanker



(c) Mariner

Figure 18: Step response comparison for first order Nomoto model and vessels.

This is a second order linear system which will be globally exponentially stable at the origin if the eigenvalues are in the left half plane. By placing all poles at $-\lambda$ we get from the error dynamics in the frequency as:

$$s^2 + K_d s + K_p = (s + \lambda)^2 \tag{3.6}$$

which gives the following proportional and derivative gains.

$$K_d = 2\lambda, \quad K_p = \lambda^2 \tag{3.7}$$

Selecting $\lambda > 0$ will then ensure the error dynamics are globally exponentially stable. This approach also ensures that the dynamics are critically damped, as both the eigenvalues lie on top of each other on the real axis of the complex plane. In addition we now only have one tuning parameter, which makes tuning the controller quite simple. It should however be noted that the properties stated above only hold for the first order Nomoto model which is only an approximation of the actual vessel dynamics, however we assume the the behaviour of the vessel is similar to that of the first order Nomoto model, and hence the control law should perform satisfactory.

3.1.2 Guidance

For the guidance scheme, the Line-of-Sight guidance algorithm discussed in Section 2.3.2 was used. The desired course angle χ_d is given as:

$$\chi_d = \chi_p + \chi_r \tag{3.8}$$

The path tangential angle $\chi_p = \gamma_p$ is used in order to ensures the vessel has the same course as the path, and give course convergence. The relative path angle χ_r is used to steer the vessel towards the path in order to ensure path convergence. The path relative angle is chosen as a mapping form the cross-track error $y_e \in \mathbb{R}^1$ to an angle $\chi_r \in [-\pi/2, \pi/2]$ by the following function:

$$\chi_r = \text{atan}\left(\frac{-y_e}{\Delta_{y_e}}\right) \tag{3.9}$$

where the tuning parameter Δ_{y_e} represents the lookahead distance to the path. Based on heuristics, this parameter is usually chosen to be between 1.5 and 2.5 times the length L_{pp} of the vessel. For our implementation, a lokahead distance $\Delta_{y_e} = 3L_{pp}$ was chosen for all the vessels, as this seemed to give the best performance.

The desired heading that is given to the motion control system is finally calculated as:

$$\psi_d = \chi_d - \beta \tag{3.10}$$

where β is the sideslip angle. Subtracting the sideslip angle in this way ensures that the we align the course angle of the vessel with the course of the path, which helps compensate for vessel asymmetry, and external forces.

3.2 Guidance using deep reinforcement learning

For the straight-line path following problem the most common approach is to use a Line-of-Sight method. Using the Line-of-Sight algorithm is a simple approach which has proved to work quite well, there are however some drawbacks. One drawback of the Line-of-Sight method is the use of cascading systems, in where a motion control system is responsible for the control system, and having the Line-of-Sight algorithm setting the set-points for the motion control system. This creates abstractions in the control architecture, causing potentially useful information to be lost, as well as making the performance of the higher level control, dependant on the performance of lower level control systems. This usually leads to sub optimal global performance. In order to improve tracking performance, we propose using Deep Reinforcement Learning in order to perform the path following task. Using DRL has the benefit of being able to learn one global policy, from the vessel state, to the rudder command, hence eliminating the need for cascading control systems. Additionally DRL allows for optimizing a performance measure, and taking into account more of the system dynamics, then traditional control methods.

3.2.1 DRL algorithm

The main contribution of this thesis is looking at how DRL can be used for learning a control policy for guidance of marine vessels. For this problem, a vital part is the Deep Reinforcement Leaning algorithm, highlighted in Figure 19, which performs the learning, exploration, and eventual control of the marine vessel. For this, we used a Deep Deterministic policy gradient algorithm given in Algorithm 3. The main reason for choosing this algorithm is due to the fact that it is an off policy algorithm, this means that the algorithm is able to optimize a learned control policy, without having to follow the learned policy. This means that the algorithm is able to learn the task by observing someone else performing the task, which for a physical marine vessel would be necessary, as performing exploratory learning may be unfeasible due safety concerns. It should however be noted that performing

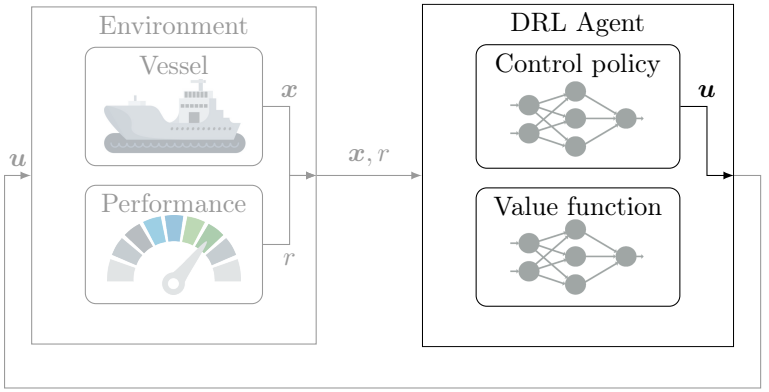


Figure 19: The control algorithm used to learn and perform the end to end guidance problem, highlighted above, is an actor-critic algorithm.

exploratory learning, where the learned policy is used to generate new training examples, is more efficient than learning from off policy examples.

The main components of the DDPG algorithm, are the policy $\pi(\boldsymbol{x})$, which is responsible for generating a control action, given a state, and action value function $Q(\boldsymbol{x}, \boldsymbol{u})$, which is an estimate of the expected cumulative discounted reward, when taking action $\boldsymbol{u}$ in state $\boldsymbol{x}$. In DDPG, both the policy, and value function are estimated using neural networks as function approximators. The policy network $\pi_{\boldsymbol{\theta}_\pi}(\boldsymbol{x})$, was implemented as fully-connected neural network, consisting of two hidden layers with 400 and 300 hidden units respectively. Between the layers the relu activation function,

$$\text{relu}(x) = \max(x, 0) \tag{3.11}$$

was used, in order to introduce nonlinearities into the function approximator. For the output layer of the policy, the output vector is reduced down to the number of outputs needed to perform the control task, and a hyperbolic tangent activation function is used in order to scale the value between -1 and 1. This output is then scaled by a linear transform such that it is between the saturating limits of the actuator it controls, giving the control vector $\boldsymbol{u}$ such that

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{\max}$$

As a function, where $h_i(x)$ represents the function of hidden layer i, and $\boldsymbol{x}$ is the input state, we get the following network:

$$\begin{aligned} h_1(\boldsymbol{x}) &= \text{relu}(\mathbf{W}_1\boldsymbol{x} + \boldsymbol{b}_1) \\ h_2(\boldsymbol{x}) &= \text{relu}(\mathbf{W}_2\boldsymbol{h_1}(\boldsymbol{x}) + \boldsymbol{b}_2) \\ \pi(\boldsymbol{x}) &= \tanh(\mathbf{W}_3\boldsymbol{h_2}(\boldsymbol{x}) + \boldsymbol{b}_3)\boldsymbol{u}_{\text{scale}} + \boldsymbol{u}_{\text{mean}} \end{aligned}$$

note that $\mathbf{W}_i$ and $\boldsymbol{b}_i$ are the weight matrices and bias vectors respectively, and are the trainable parameters which we wish to learn. For the action value function approximator $Q_{\boldsymbol{\theta}_Q}(\boldsymbol{x}, \boldsymbol{u})$, a similar architecture was used. The network consisted of two hidden layers with 400 and 300 units respectively, relu activation functions, and the a scalar value as output. For the action value function, both the state and action are given as inputs. In order to accommodate this, the state is passed though both layers, while the action is only passed through the last layer. As a function where $h_i(x)$ represents the function of hidden layer i, and $\boldsymbol{x}$ and $\boldsymbol{u}$ are the inputs, we get the following network:

$$\begin{aligned} h_1(\boldsymbol{x}) &= \text{relu}(\mathbf{W}_1\boldsymbol{x} + \boldsymbol{b}_1) \\ h_2(\boldsymbol{x}, \boldsymbol{u}) &= \text{relu}(\mathbf{W}_{2,\boldsymbol{x}}\boldsymbol{h_1}(\boldsymbol{x}) + \mathbf{W}_{2,\boldsymbol{u}}\boldsymbol{u} + \boldsymbol{b}_2) \\ Q(\boldsymbol{x}, \boldsymbol{u}) &= \mathbf{W}_3\boldsymbol{h_2}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{b}_3 \end{aligned}$$

where $\mathbf{W}_i$ and $\boldsymbol{b}_i$ are the trainable parameters.

For the training of the policy, and action value function networks, Algorithm 3 was used. The batch size for training was 64 transitions, which were sampled randomly form a buffer of a maximum of 10^6 transitions. For the policy, or actor learning rate a value of 10^{-4} was used, while for the critic or value function a learning rate of 10^{-3} was used. The discounting rate γ was 0.99, and the target network update rate τ was 10^{-3}.

3.2.2 Reward function

The reward function, or performance measure, is very important in reinforcement learning. The objective of any reinforcement learning agent is to maximize the reward. And hence the reward function is where the behaviour of the system is shaped. The performance measure or reward function, is not considered a part of the reinforcement learning agent, but rather a part of the environment, as illustrated in Figure 20. Having the performance measure as part of the environment,

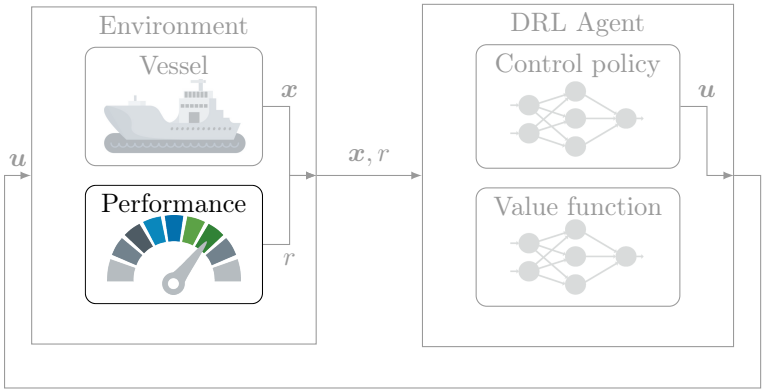


Figure 20: The performance measure, highlighted above, gives the learning algorithm a reward or penalty based on the performance. This is the actual measure we want the learning algorithm to optimize.

makes sense in many situations, as the reward may be a measurable quantity related to the environment, for which the underlying mapping from state and action to reward is unknown.

For the path following task, there are many options for designing a reward function, depending on the desired behaviour. Since the objective of a path following guidance system is to regulate the cross-track error to zero, a reward function which rewards the system for being close to the path, and penalizes it for being far away, is needed. For our path following approach we further wanted to maximize the time the vessel is on the path, or alternatively minimize the time the vessel is not on the path. One such reward function would be to give a reward of 1 when the vessel is on the path and a reward of 0 when it is not. Such a reward function would however be very difficult to use, as the the policy is learned through experience, and the probability of being exactly on the path, and hence getting experience, is zero. We therefore propose a similar boundary reward, which gives a reward of 1 if the vessel is sufficiently close to the path, and otherwise we would give a reward of 0. This reward function also has a minimum time interpretation, where we minimize the time until the vessel is within a certain bound of the path, while ensuring that it is possible to receive a reward while the RL algorithm is exploring the environment.

In addition to the path convergence, it is also important to follow the path in a certain direction, that is towards the next waypoint, which is also something that needs to be taken into account in the objective function. For achieving these tasks we give a reward of 1 if the vessel is sufficiently close to the path, and pointing in the correct direction, and 0 otherwise. This gives the following boundary reward function:

$$R(y_e, \tilde{\psi}) = \begin{cases} 1 \text{ if } |y_e| < b \text{ and } |\tilde{\psi}| < \frac{\pi}{2} \\ 0 \text{ oterwise} \end{cases} \tag{3.12}$$

where b is the width of the boundary. It should be noted that due to the shape of the boundary reward, we can not guarantee the vessel will converge to the path, as the reward for being anywhere within the boundary is the same as being on the actual path. In order to further improve path convergence we propose using a reward which will increase monotonically, peaking at a cross track error of 0. Many such functions are available, but in order to keep similar properties to the boundary function, we propose using a Gaussian reward function.

$$f(x) = ae^{-\frac{(x-\mu)^2}{2\sigma}} \tag{3.13}$$

This gives a bell curve with amplitude a, centered at μ with a standard deviation of σ. Using a standard deviation $\sigma = b$ and an amplitude $a = 1$ we get a function which has some the same properties to the boundary reward, but with a maximum reward with a cross-track error $y_e = 0$. In order to use this as a reward function we must still ensure that the vessel is traveling with the path and not against it, this can be done by making sure the yaw angle is within $\pm 90°$ of the angle of the path we are following. This gives the reward function:

$$R(y_e, \tilde{\psi}) = \begin{cases} e^{-\frac{(y_e-b)^2}{2\sigma}} & \text{if } |\tilde{\psi}| < \frac{\pi}{2} \\ 0 & \text{oterwise} \end{cases} \tag{3.14}$$

The Gaussian reward function has the benefit of promoting path convergence, as well as giving a larger area in which a reward is given. This manes a less sparse reward, and hence faster learning. The Gaussian reward, and boundary reward are illustrated in Figure 21. For the actual implementation, we used a reward boundary $b = 10m$ and a standard deviation $\sigma = 10m$, as these seemed to give fairly good convergence, while still giving reward often enough to have efficient learning. Decreasing the values should theoretically give better path convergence, however a decrease would also give a more sparse reward, which in practice gives less stable

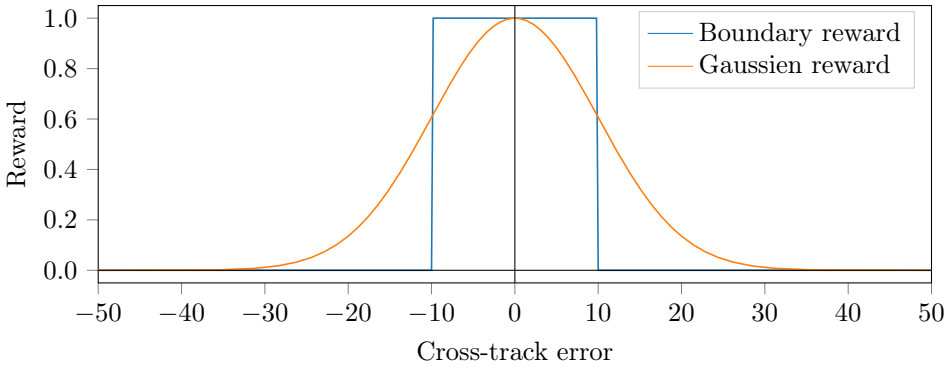


Figure 21: Comparison of Gaussian reward with $\sigma = 10$ and $\mu = 0$, and boundary reward with a boundary of 10, centered at 0

as well as slower learning.

Other considerations, when designing a reward function, is the magnitude of the reward. Given the highest possible reward magnitude r_{max}, we can compute an upper bound on the value function $V(\boldsymbol{x})$, given the discount rate $\gamma < 1$, as follows:

$$V(\boldsymbol{x}) \leq \sum_{n=0}^{\infty} r_{max}\gamma^n = \frac{r_{max}}{1-\gamma} \quad \forall \boldsymbol{x} \tag{3.15}$$

When using a reward with the largest possible magnitude of 1, and a discount rate $\gamma = 0.99$, the maximum value function value would be $\frac{1}{1-0.99} = 100$. If the reward becomes very large, it can be beneficial to reduce the maximum reward, this in order to increase numerical stability, and avoid large parameter values in the value function approximation.

In the reward function purposed above, it is possible to add additional rewards or penalties to achieve additional goals. One such goal we found useful was the addition of a penalty on aggressive control actions, as it would cause significant wear on the control surfaces, as well as passenger discomfort. In order to implement this type of penalty, we propose adding a small penalty term, based on the derivative of the rudder movement, to the reward function. This will favour slower and smoother control actions. For our implementation we decided to use the following quadratic

penalty term on the rudder derivative $-c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2$, where $c_{\dot{\delta}\dot{\delta}} > 0$ is the weighting factor of the penalty. Adding this to for example the Gaussian reward we get the following reward function:

$$R(y_e, \tilde{\psi}, \dot{\delta}) = -c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2 + \begin{cases} ae^{-\frac{y_e^2}{2\sigma}} & \text{if } |\tilde{\psi}| < \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

3.2.3 State augmentation

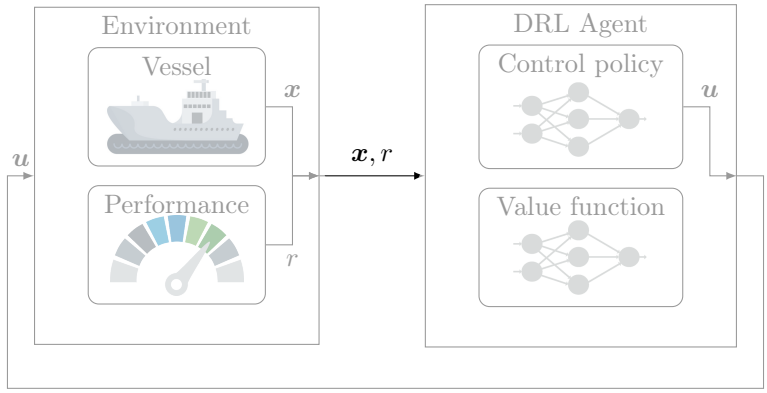


Figure 22: The state vector, highlighted above, can be augmented to give the DRL algorithm good features that are relevant for the assigned task.

For control using DRL, it is important to give the DRL algorithm useful information. This ensures it is able to quickly, and efficiently learn and optimize the policy. The state vector $\boldsymbol{x}$, highlighted in Figure 22, is the source of information that the DRL algorithm is given. The simplest state vector we could give the DRL algorithm is the vessel state vectors $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$, together with path information, such as waypoints. This is however not a good solution, as this means the state is not invariant to rotation or translation of the path. This type of state vector also requires the DRL algorithm to do a significant amount of transformation in the function approximators, in order to map the state to a control action and value function value. Even though the function approximators are able to learn nonlinear transforms of the state, augmenting the state vector by manually adding useful transformed states, reduces the amount of transforms needed in the function approximator, and hence reduces the architecture complexity needed. In general this translates to less

layers in the ANN, and hence a network which is easier to train, without adding to much additional cost, as only the layer size of the initial layer is changed in order to accommodate size of the state vector.

straight-line path following

For the straight-line path following task we propose the following state vector:

$$\boldsymbol{x} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \tilde{\psi} \\ \dot{\tilde{\psi}} \\ u \\ v \end{bmatrix} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \psi - \gamma_p \\ r \\ u \\ v \end{bmatrix} \tag{3.17}$$

Here we have chosen to include the cross-track error y_e and its derivative $\dot{y}_e$, as they give a measure of how close the vessel is to the path, and how fast it is approaching, while being invariant to the path position. The heading error $\tilde{\psi} = \psi - \gamma_p$, and heading error rate $\dot{\tilde{\psi}}$ are used to give a measure of the direction the vessel is traveling with respect to the path. We have also added the surge and sway velocities u and v which give information about the vessel dynamics at different velocities, and are invariant to the heading of the vessel.

In addition to the state vector proposed above, we also propose an extended state vector, which includes course information. Adding course angle error $\tilde{\chi} = \chi - \chi_d = \chi - \gamma_p$ and course angle error rate $\dot{\tilde{\chi}} = \dot{\chi} - \dot{\gamma_p}$ which is $\dot{\chi}$ for straight-line paths, we get the following state vector:

$$\boldsymbol{x} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \tilde{\chi} \\ \dot{\tilde{\chi}} \\ \tilde{\psi} \\ \dot{\tilde{\psi}} \\ u \\ v \end{bmatrix} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \chi - \gamma_p \\ \dot{\chi} \\ \psi - \gamma_p \\ r \\ u \\ v \end{bmatrix} \tag{3.18}$$

The benefit of using this state vector, is the additional information about about how the vessel velocity vector lines up with the path. This ensures that information about the sideslip due to model asymmetry, external forces, and model dynamics,

are included, and can be more easily compensated for in the policy.

It is worth noting that all the states in the proposed state vectors above can all be computed from the vessel state vectors $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$ when the path is known, as we showed in Section 2.3. This means that the proposed states are possible for a function approximator to compute, however it would increase the complexity needed for the function approximator, meaning slower and possibly less stable learning.

Curved path following

Training the algorithm for following curved line paths, we use the state vector:

$$\boldsymbol{x} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \tilde{\psi} \\ \dot{\tilde{\psi}} \\ u \\ v \\ r \end{bmatrix} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \psi - \gamma_p \\ r - \dot{\gamma}_p \\ u \\ v \\ r \end{bmatrix} \tag{3.19}$$

This state vector gives the same information, as in the curved path following task, as well as including information about path curvature. Similarly to the straight-line path following problem, it is possible to extend the vector with course information to get the following extended state vector

$$\boldsymbol{x} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \tilde{\chi} \\ \dot{\tilde{\chi}} \\ \tilde{\psi} \\ \dot{\tilde{\psi}} \\ u \\ v \\ r \end{bmatrix} = \begin{bmatrix} y_e \\ \dot{y}_e \\ \chi - \gamma_p \\ \dot{\chi} - \dot{\gamma}_p \\ \psi - \gamma_p \\ r - \dot{\gamma}_p \\ u \\ v \\ r \end{bmatrix} \tag{3.20}$$

For curved paths the additional course information is especially useful, as the turning induces sideslip, and hence the course angle becomes less reliable for path following, since the objective is to align the vessel course with path tangent when on the path.

3.2.4 Training

Training the policy $\pi_{\boldsymbol{\theta}_\pi}(\boldsymbol{x})$ and action value function $Q_{\boldsymbol{\theta}_Q}(\boldsymbol{x}, \boldsymbol{u})$ of the DDPG algorithm is in general performed as described in Algorithm 3. For the vessel guidance problem, the vessel was randomly placed on the map within 1000 meters of a path, and then the vessel was simulated for 1000 steps, while the action, state and performance of the vessel was recorded. For each time-step training was performed, by randomly sampling 64 transitions, and performing SGD in order to improve the action value function approximation, and furter optimize the policy.

Guided training

DDPG is an off policy algorithm, this means it is possible to train the algorithm off policy on transitions not generated by the learned policy. It is natural to assume that using transitions from a policy that is known to work, will improve training, as less exploration is required in order to find a good policy. This is however not the case, since training strictly on off policy training samples, generated by for example a Line-of-Sight method, does lead to quite poor control performance. The reason for this is that any deterministic policy, such as the Line-of-Sight guidance law, will in general tend to only observe a subset of the full state space. This means the training data is biased, and includes little exploration from which the action value function approximation can generalize, and the policy approximation can optimize performance.

In order to get the benefit of faster training by using known policies such as the Line-of-Sight algorithm, while avoiding the potential bias, and lack of exploration, we propose using guided training. Given a the policy we are learning $\pi_{\boldsymbol{\theta}_\pi}(\boldsymbol{x})$ and a known guide policy $\pi_{guide}(\boldsymbol{x})$, guided training can be performed by blending the two policies. The policies can be blended in the following way

$$\pi_{\text{blended}}(\boldsymbol{x}) = (1 - \omega)\pi_{\boldsymbol{\theta}_\pi}(\boldsymbol{x}) + \omega\pi_{\text{guide}}(\boldsymbol{x}) \tag{3.21}$$

where ω is the blending factor. Guided training may still bias the learned policy, by counteracting the actions of the learned policy. When using guided training, it is therefore beneficial to sample from the blended policy $\pi_{\text{blended}}(\boldsymbol{x})$ with a probability p and use the learned policy $\pi_{\boldsymbol{\theta}_\pi}(\boldsymbol{x})$, with a probability $1 - p$. This helps remove some of the bias introduced by the guide policy.

Transfer learning

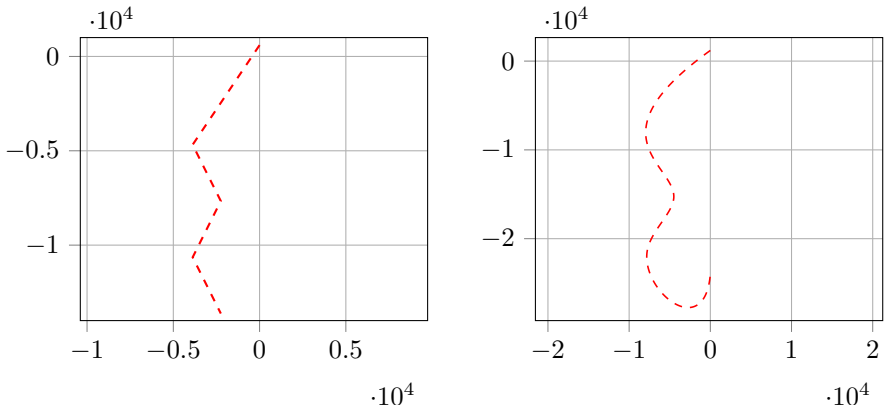
In addition to training the algorithm from scratch, we also utilized transfer learning, discussed in Section 2.1.3. Specifically we used transfer learning when training the control algorithm on path following of curved paths, by using the policy, and value function learned for doing straight-line path following. Since straight-line path following and curved path following are similar tasks, the transfer learning was able to use, and refine what it had learned in the straight-line path following task, in order to quickly learn the curved path following task.

Chapter 4

Simulations

4.1 Main Results

In the following section, we will discuss some of the main results that were achieved for both the curved path and straight-line path following tasks. The benchmark tests were performed on the paths seen in Figure 23. The straight-line path in Figure 23a consists of the waypoints in Table 2, and the curved path in Figure 23b was made by interpolating the third order splines for the points in Table 3.



(a) straight-line path (b) Curved path

Figure 23: Paths used for testing algorithm performance.

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

Table 2: Waypoints used for generating the straight-line path segments

Waypoint	1	2	3	4	5	6
Path parameter ω	0	1000	2000	3000	4000	5000
Position x	0	-3880	-2248	-3878	-2246	0
Position y	600	-4680	-7662	-10644	-13626	-12000

Table 3: Waypoints used for generating interpolated curved path

4.1.1 Training progress

Training the control law consisted of episodes in where the vessel is randomly placed on the map within 1000 meters of a the path, and simulating the vessel for 1000 steps, while recording the actions, performance and and state transitions of the vessel. Saved transitions are sampled in batches at each time step, and SGD is performed according to Algorithm 3, in order to iteratively optimize the value function and policy parameterization.

When training the control algorithm on the straight-line path following task for the three different vessels, we got the training progress seen in Figure 24. From the results we can see that when using normal training, a good policy is found after about 300 and 500 episodes for the Mariner and Container vessel respectively, while no policy is found for the Tanker. When using guided training, a good policy is found after 400, 800 and 300 for the Mariner, Tanker and Container vessel respectively. For normal training, the results are as expected, when considering the size and maneuverability of the different vessels, since the mariner is the smallest vessel we would expect to see faster path convergence and more reward, leading to faster training, while the larger container vessel, and even larger tanker vessel, we would expect to see slower convergence, less reward, and hence slower training.

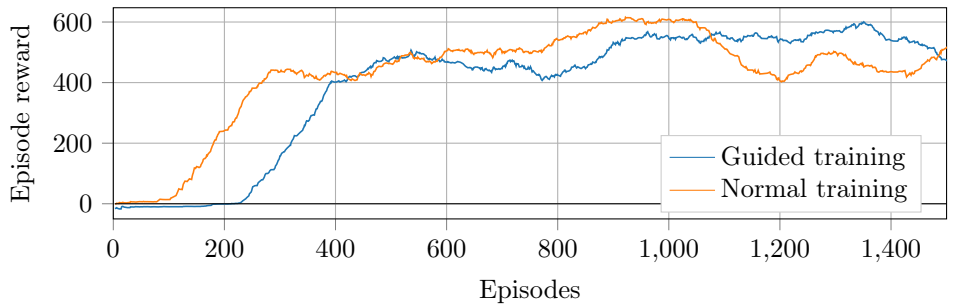
Using guided training, we see from the results in Figure 24, that there is a clear improvement in training for the tanker, as it now is able to learn a policy which ensures the vessel is able to follow the path. For the container vessel we also see an improvement in terms of a shorter exploration phase, however after the initial

peak, the reward for the guided policy has a significant dip, which may be due to interference from the bias introduced by the guide policy. For the mariner vessel we see the opposite effect of what was seen for the two other vessels, in that the initial learning phase takes longer when using guided training, this is also most likely due to bias introduced by the guide policy, which give training examples that are less useful, then what is achieved when using normal training.

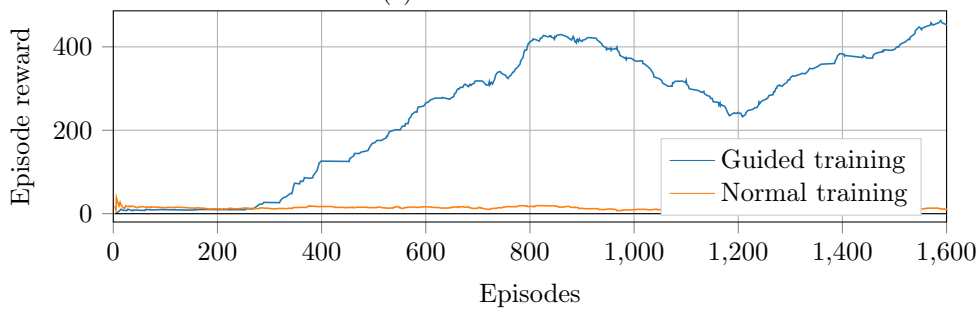
Using guided training seems to speed up training in the initial exploration phase in some cases, it should however be noted that tuning the blending factor ω, and probability p may give a different performance. It may also be beneficial to decrease the probability p of taking actions from the blended policy as time progresses, this will help speed up the initial training phase, as well as avoid biasing the policy with the a sub-optimal policy in the later training phase.

4.1.2 Counteracting rudder noise

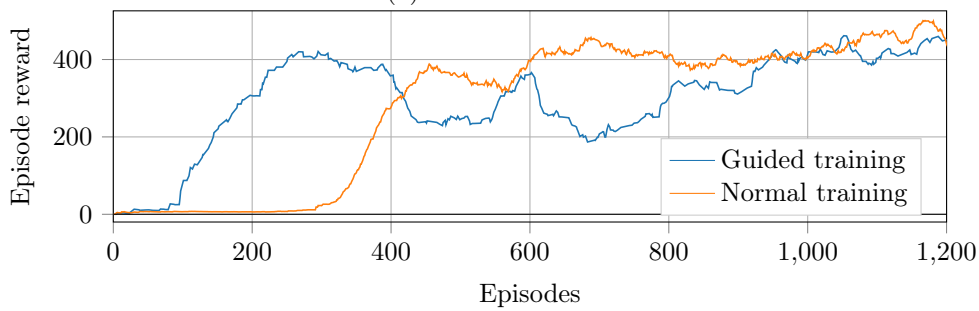
For the straight-line path following task, when following the path seen in Figure 23a, we see from the results in Figure 25b and Figure 26b, that some very aggressive control actions are being taken, the same behaviour was observed both when using the Gaussian reward and boundary reward. This behaviour is however not desirable, and in order to counteract it, the addition of a quadratic penalty on the rudder derivative $-c_{\dot{\delta}\dot{\delta}}\dot{\delta}^2$ was added. After tuning the the weighting factor, we found that $c_{\dot{\delta}\dot{\delta}} = 20$ gave good results, in were the control actions were significantly smoother, while still exhibiting good path following behaviour. After training the Mariner and Container vessel with the new reward function, we got the result given in Figure 25a, and Figure 26a when simulating the vessels for the same path. These are some very interesting results, as they demonstrate how the reward function can easily be shaped in order to accommodate any desired behaviour. Changing the reward however, means that the original minimum time interpretation of the the Gaussian and boundary reward no longer hold. The objective can now be interpreted as a weighting between reducing the rudder angular velocity, while still getting to the path in a reasonable amount of time.



(a) Mariner vessel

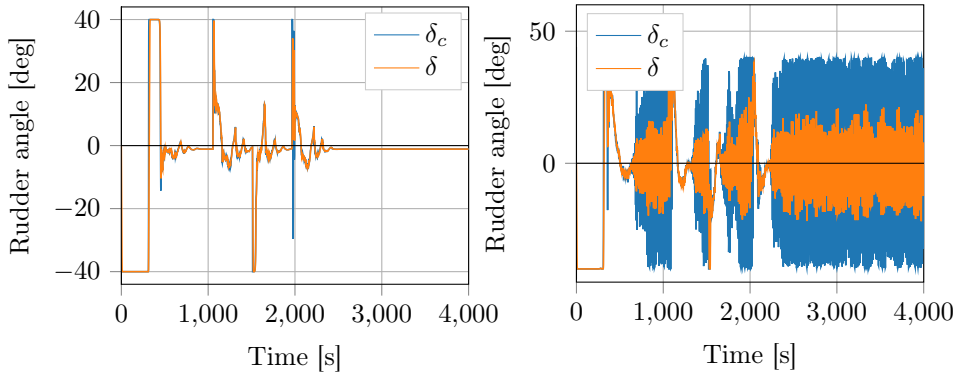


(b) Tanker vessel



(c) Container vessel

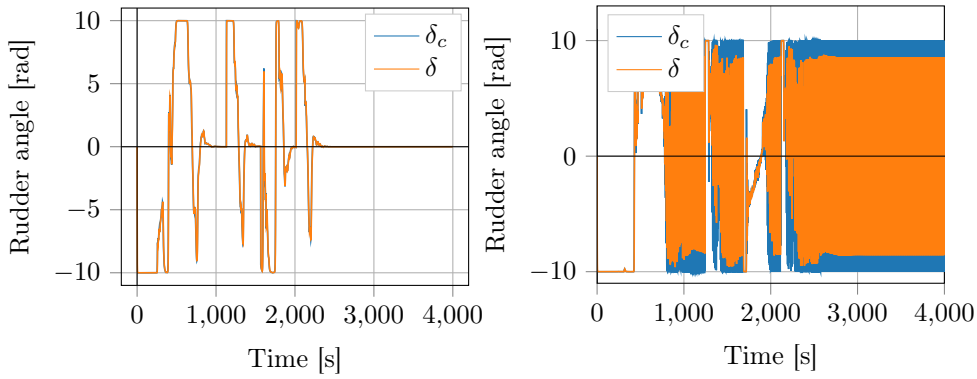
Figure 24: Comparison of training time when using guided and normal training, for the straight-line path following problem.



(a) With rudder derivative penalty

(b) Without rudder derivative penalty

Figure 25: Mariner vessel rudder angle δ and command rudder angle δ_c, with and without rudder derivative penalty.



(a) With rudder derivative penalty

(b) Without rudder derivative penalty

Figure 26: Container vessel rudder angle δ and command rudder angle δ_c, with and without rudder derivative penalty.

4.1.3 Path convergence

Path convergence is a desirable trait for a path following controller. If convergence can be shown, then we can ensure the vessel will continuously get closer to the path such that as time approaches infinity, the vessel approaches the path.

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

For the curved path following, path convergence is difficult to achieve, this is due to the vessel being underactuated, which causes certain vessel configurations to not be reachable. For a straight-line path however, path convergence is possible. Testing the learned control policies on a straight-line path we got the results in Figure 27. As we can see, a steady state error is present, and hence path convergence is not achieved with neither the Gaussian reward or the boundary reward directly. For the Boundary reward this is an expected result, since the vessel has no incentive to converge to the path as long as the vessel is within the boundary, and receives the same reward. For the Gaussian reward however, we would expect the vessel to converge, since the highest reward is found with cross-track error $y_e = 0$. This is not the case, however we can observe that the Gaussian reward has a lower steady sate error then the boundary reward. The reason for not achieving convergence for the Gaussian reward is most likely due to the nature of the deep reinforcement learning algorithm, which does not converge to the optimal solution where it learns how to counteract the steady state error, but rather finds a sub optimal solution. It may be possible to get better convergence by slowing down the learning rate, and training on larger batches, however this will slow down learning, and is still not guaranteed to work.

As discussed in Section 3.2.3, DRL is highly dependant on the quality of the information it receives. Augmenting the state vector in order to get more relevant information, and reduce the complexity of the function approximatiors, will usually give better results. Using the extended state vector, where the the additional course error $\tilde{\chi}$ and course error rate $\dot{\tilde{\chi}}$ is added, we give the control algorithm information about about how the vessel velocity vector lines up with the path. This ensures that information about the sideslip due to model asymmetry, external forces, and model dynamics, is included, and can be compensated for. From the results in Figure 27, we see a significant improvement over the performance when using the minimal state vector, however the vessel still has a steady state error.

One way of achieving convergence is estimate the steady state cross-track error $\hat{e}_{ss}$, and using this to compensate by offsetting the observed cross-track error by the steady state error before it is used in the control algorithm. Intuitively, this can be viewed as trying to follow a virtual path, which is offset from the actual path such that the steady state error causes the vessel to stay on the actual path. Estimating the steady state error can either be done offline, or alternatively it can be done adaptively online by integrating the cross-track error. This is given by the following equation

$$\hat{e}_{ss}(t) = k_i \int_0^t e(t)dt \tag{4.1}$$

where k_i describes the rate at which the error estimate changes. In discrete time this can be implemented in the following way

$$\hat{e}_{ss} \leftarrow \hat{e}_{ss} + \tau k_i e_t \tag{4.2}$$

where τ is the sample time, and e_t is the cross track error at time t. In order to implement the compensation, the augmented state vector

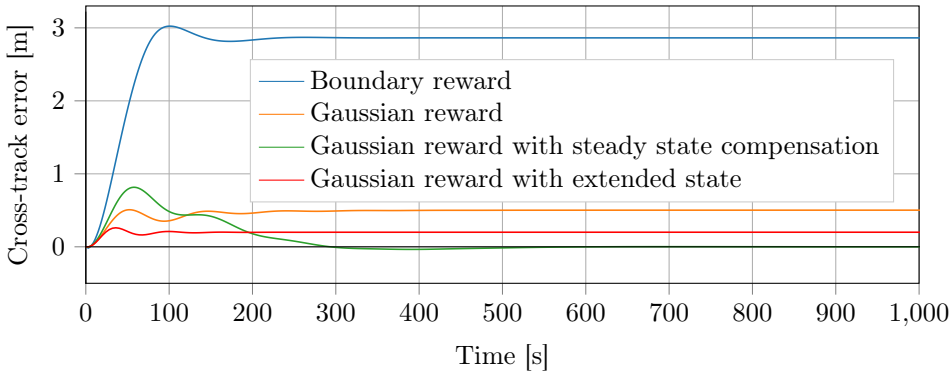
$$\boldsymbol{x} = \begin{bmatrix} y_e + \hat{y}_{e,ss} \\ \dot{y}_e \\ \vdots \end{bmatrix} \tag{4.3}$$

is used as input to the trained policy. It should be noted that adding the steady state compensation should only be done on a trained policy, and not performed during training, as doing so will cause interference and possibly divergence due to exploration noise. Additionally, an integration strategy with anti windup is used in order to reduces overshoot, and improve stability.

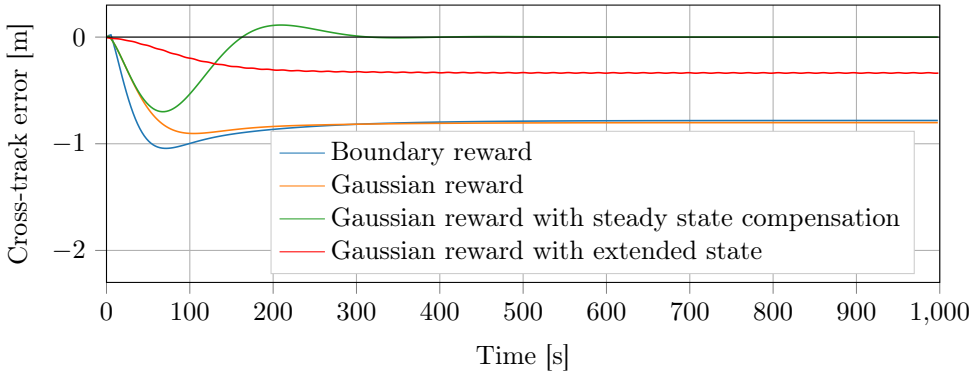
When adding the adaptive steady state error compensation proposed above, we see from the results in Figure 27, that we are able to compensate for the steady state errors such that the vessels are able to fully converge to the path.

4.1.4 Guidance for straight-line paths

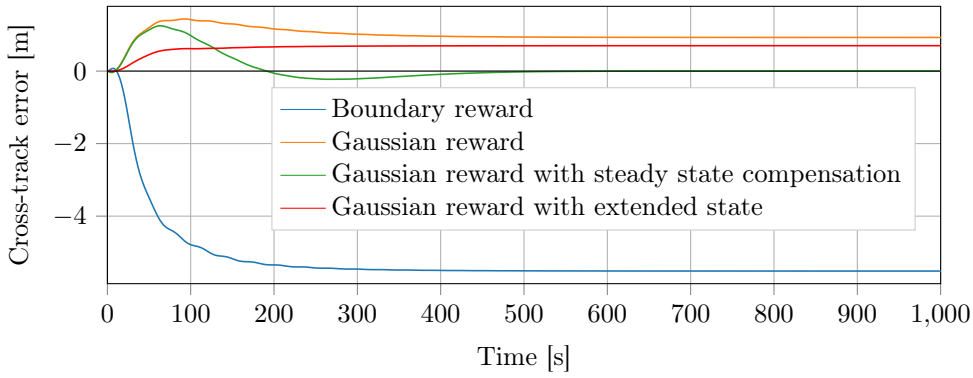
For bench marking the performance of the DRL learning algorithm, we look at the cumulative reward gathered, when performing path following on the straight-line path given in Figure 23a, giving the results found in Table 4. For the results we have chosen to focus on the performance of the Gaussian reward, as it performs



(a) Mariner vessel



(b) Tanker vessel



(c) Container vessel

Figure 27: Cross-track error comparison when using boundary reward, Gaussian reward, Gaussian reward with steady state error compensation and Gaussian reward with extended state space.

similarly to the boundary reward, but has better convergence properties, and faster as well as more stable learning. From the results we see that the learned policy when using both the minimal, and extended state vector give very similar results, and for both the Mariner and the Container vessel the DRL approach outperforms the Line-of-Sight guidance law. The fact that the learned policy is outperformed on the Tanker, reflects the struggle that was seen when training on the Tanker, where guided training was required to achieve a working policy, and even then, the training was unstable. Looking at the performance when using the minimal state vector inn comparison with using the extended state vector, we see that the minimal state vector seems to outperform the extended state vector in all but one of the cases. This shows that no significant additional information is added when extending the state vector. The extended state vector however gave more robust performance, as the minimal state vector more often would fail to follow the path, as seen for the container vessel when exposed to currents.

Without ocean currents

Simulating the different vessels with the policy found after training on the path following task without currents, we get the results seen in Figure 28, Figure 29 and Figure 30 for the Mariner, Container and Tanker respectively. It should be noted

	Mariner		Container		Tanker	
Current	$0m/s$	$0.9m/s$	$0m/s$	$0.9m/s$	$0m/s$	$0.9m/s$
LOS	1885	1137	1128	1138	<u>1597</u>	<u>1022</u>
DRL minimal	<u>2174</u>	<u>1783</u>	<u>1397</u>	157	1252	735
DRL extended	2136	1606	1281	<u>1291</u>	1056	685

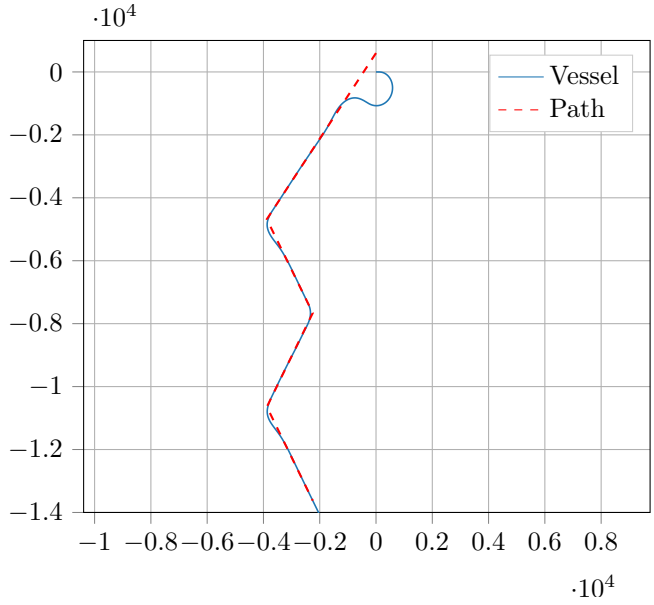
Table 4: Cumulative Gaussian reward for the straight-line path following problem

that these are the results when using the Gaussian reward, as it gave the best convergence properties, as well as more stable, and faster training. When following a path defined by the waypoints in Table 2, we observe some very different results for the various vessels. For the Mariner we see a very good performance, where the vessel is able to converge reasonably quickly to the path, with some minor spikes when switching from one line segment to another. The Container vessel gives a reasonably good performance, in comparison to the Mariner we see a significantly slower path following behaviour, and more overshoot, which is to be expected for a larger vessel. For the Tanker vessel we see a quite poor performance, with very slow path convergence. It should be noted that the poor observed performance is partly due to the slow path convergence, leading the vessel to not have converge before switching between path segments.

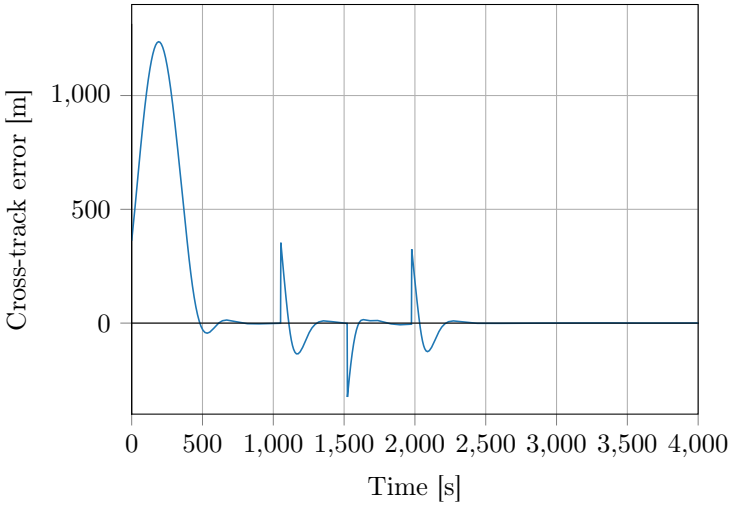
Comparing the behaviour of the different vessels in Figure 28, Figure 29 and Figure 30 it is interesting to note the difference in how aggressively the vessels head towards the path, with the smaller Mariner preferring sharper turns, while the larger Tanker preferring larger turns. This behaviour reflects the dynamics of the different vessels, since the Mariner is quick and easy to maneuver, performing sharp turns will get you more quickly on to the path. While the larger and slower Container and Mariner take longer to turn, and loose more speed in the process, such that larger turns, give faster convergence.

With ocean currents

With the addition of ocean currents to the straight-line path following problem, we introduce a new challenge, in where the vessel is affected differently depending on the heading of the vessel with respect to the current heading. In order to be able to compensate for currents the learning algorithm must learn to extract useful information about the current in the body frame of the vessel, which it can use to compensate.

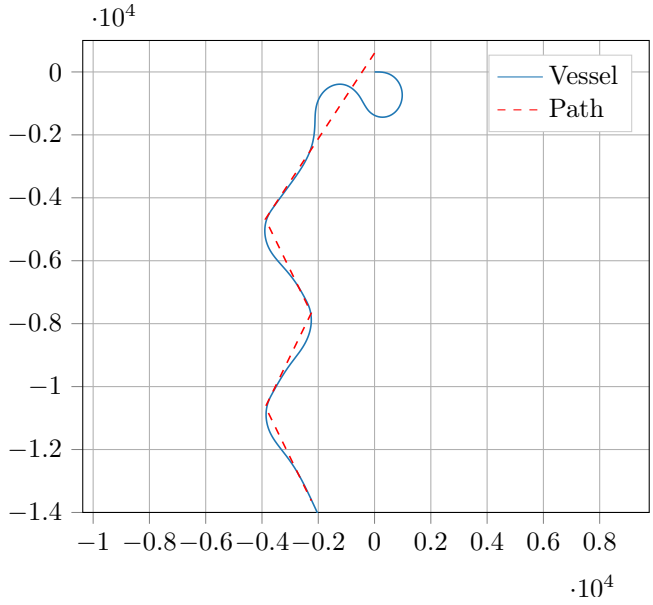


(a) Path following

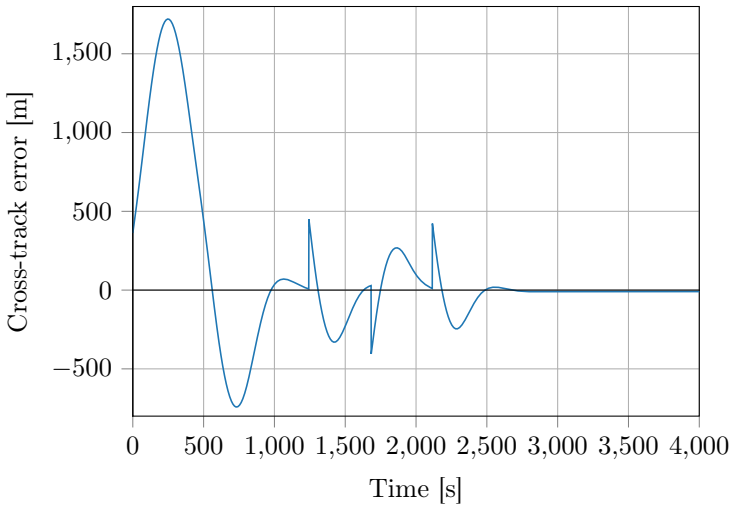


(b) Cross-track error

Figure 28: Mariner guidance simulation

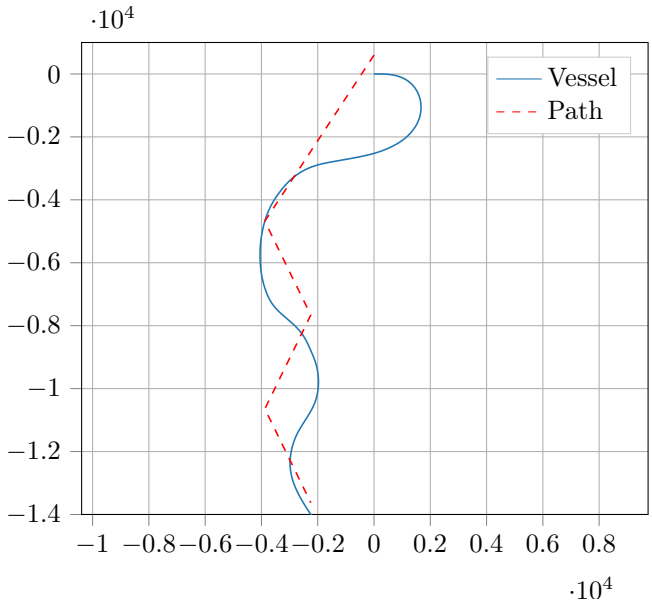


(a) Path following

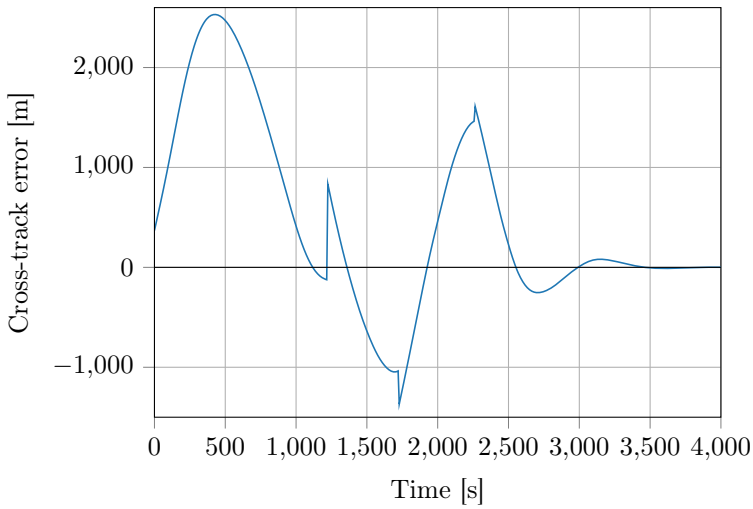


(b) Cross-track error

Figure 29: Container guidance simulation



(a) Path following



(b) Cross-track error

Figure 30: Tanker guidance simulation

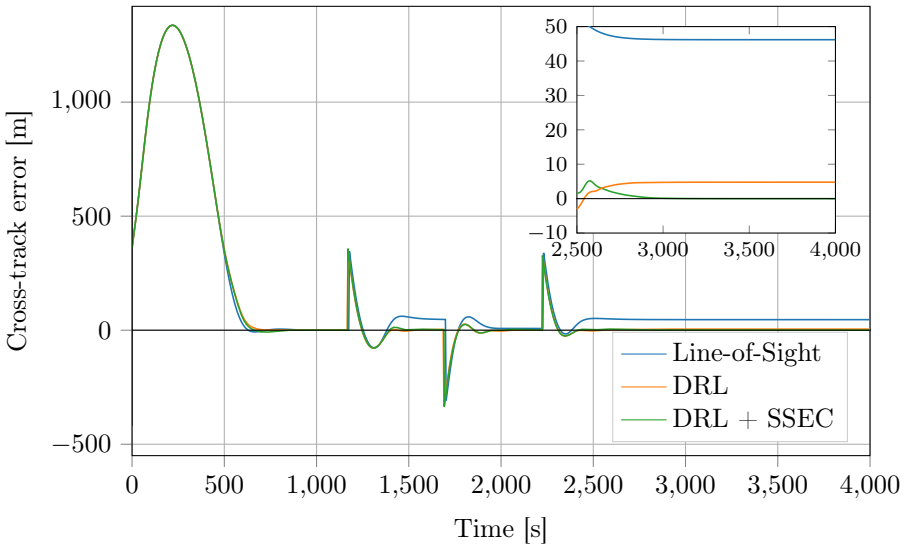


Figure 31: Cross-track error comparison for Mariner vessel with current, when using Line-of-Sight, DRL and DRL with steady state error compensation (SSEC).

In order to teach the learning algorithm to compensate for ocean currents, training was performed with the addition of a current with a uniformly distributed random magnitude and heading, for each training episode. This ensures that learning algorithm learns a robust policy which can work for a variety of current configurations.

Applying the learned policy to the path following problem in Figure 23a, with a current of $0.9m/s$ at $45°$, we get the behaviour seen in Figure 31, Figure 32 and Figure 33 for the Mariner Container and Tanker vessels respectively. From the results we clearly see a performance increase over the Line-of-Sight algorithm without any current compensation. For the Mariner and Container vessel, the learned policy is able to get the vessel within $5m$ and $10m$ of the path, while for the Tanker there is only a slight performance increase over the Line-of-Sight method with a cross track error of about $80m$ at the end of the simulation. Adding steady state compensation, where the path steady sate error is estimated and added to the state vector, the algorithm is able to achieve path convergence for all vessel models.

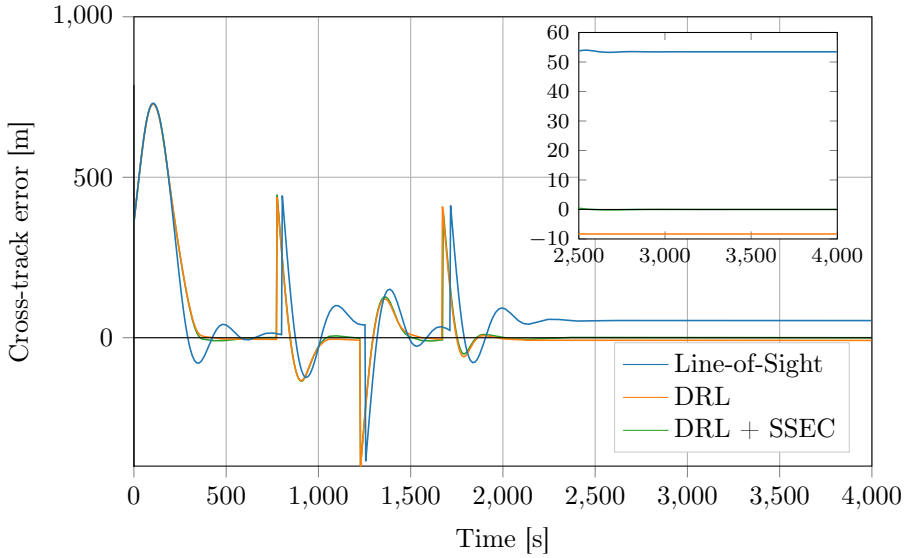


Figure 32: Cross-track error comparison for Container vessel with current, when using Line-of-Sight, DRL and DRL with steady state error compensation (SSEC).

4.1.5 Guidance for curved paths

For the curved path following problem, the objective is the same as for the straight line following problem, namely to converge to the path, the difference being that the path we wish to converge to is no longer a straight line, but a parametric curve. In order to train the DRL algorithm, a method known as transfer learning was used, where the models used for straight-line path following were used as initial models, and refined on the curved path following problem. Doing this meant that the learning algorithm did not have to learn a vessel policy from scratch. Rather it starts with the straight-line path following algorithm, and can augment the learned policy and value function in order to better suite the curved path following problem. In Figure 34, we can see that using transfer learning significantly reduced the training time for the Mariner vessel, when compared to training from scratch.

Similarly to the straight-line path following problem, we benchmarked the performance of the learned policy for the curved path following task by recording the cumulative reward achieved on the path in Figure 23b, to get the results in Table 5.

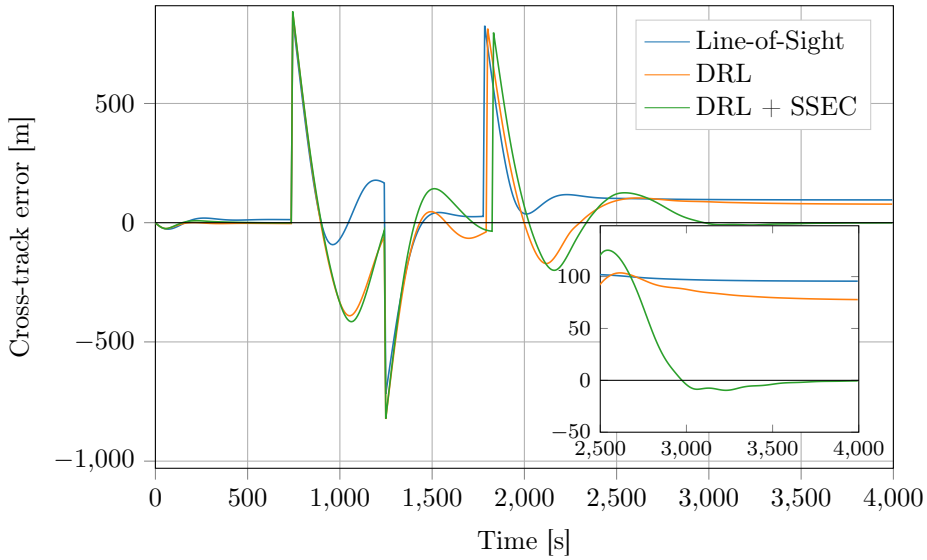


Figure 33: Cross-track error comparison for Tanker vessel with current, when using Line-of-Sight, DRL and DRL with steady state error compensation (SSEC).

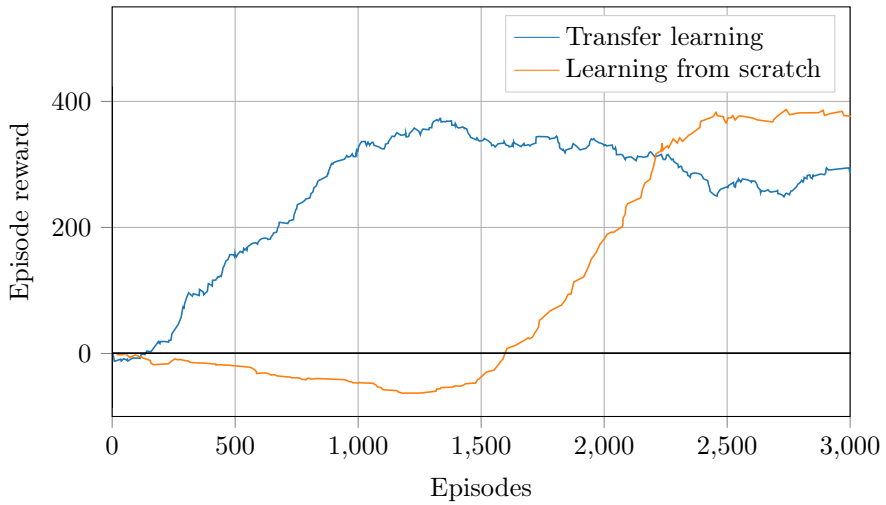


Figure 34: Training progress for transfer learning, and learning from scratch on mariner vessel.

	Mariner		Container		Tanker	
Current	$0m/s$	$0.9m/s$	$0m/s$	$0.9m/s$	$0m/s$	$0.9m/s$
LOS	161	499	807	829	65	62
DRL minimal	1250	1249	<u>2495</u>	1492	176	34
DRL extended	<u>3264</u>	<u>3134</u>	2430	<u>2763</u>	<u>1010</u>	<u>819</u>

Table 5: Cumulative Gaussian reward for the curved path following problem

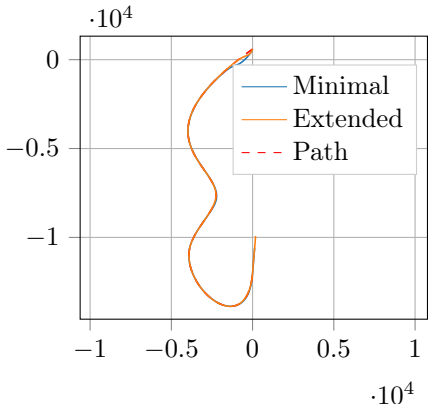
Comparing the performance of the Line-of-Sight guidance law with the learned control law when using both the minimal, and extended state representations, we see that the learned policies for both the state representations, significantly outperform the Line-of-Sight guidance law. Comparing the performance of the minimal and extended state representation, we also see that the extended state representation mostly outperforms the minimal representation. This indicates that the added course information is quite important when following curved paths. This is likely due to larger sideslip angles experienced when turning, causing the heading angle to be less reliable.

Without current

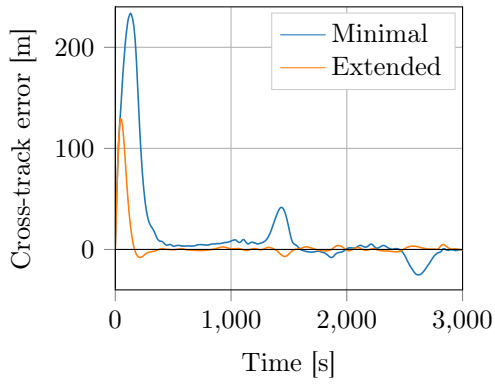
Training the different vessels on the curved path following problem, with the Gaussian reward function and rudder derivative penalty, we got the simulation seen in Figure 35, Figure 36 and Figure 37 for the Mariner, Container and Tanker vessel respectively. It should be noted that the waypoints used for the Tanker vessel were scaled by a factor of 2, due to limitations in the turning rate of the vessel, making the path difficult for the Tanker to follow. From the results we clearly see that the minimal state vector struggles to keep the vessel aligned with the path when performing sharper turns, in comparison the extended state vector is able to compensate and significantly reduce the cross-track error. This observation, confirms the results in found in Table 5, and indicates that adding course information is important when following curved paths, due to the additional sideslip.

With Current

Simulating the policies found when training for the curved path following problem with current, we got the result in Figure 38, Figure 39 and Figure 40, for the Mariner, Container and Tanker vessel respectively. From the results we again see the extended state representation gives significantly better performance then the minimal state representation. This was also apparent when training the control

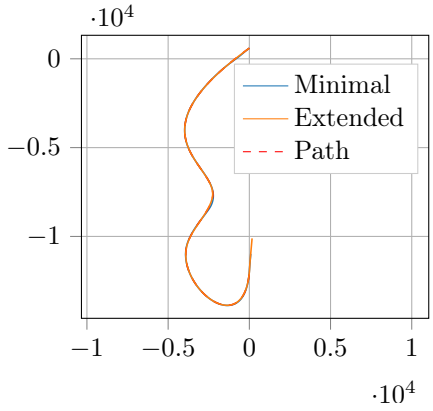


(a) Path.

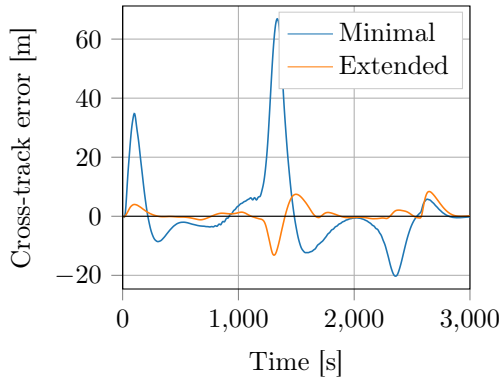


(b) Cross-track error.

Figure 35: Mariner curved path following when using minimal state and extended state representations.

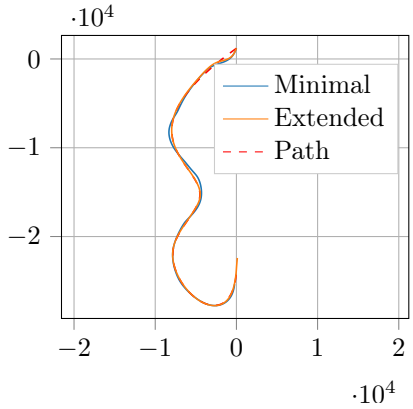


(a) Path.

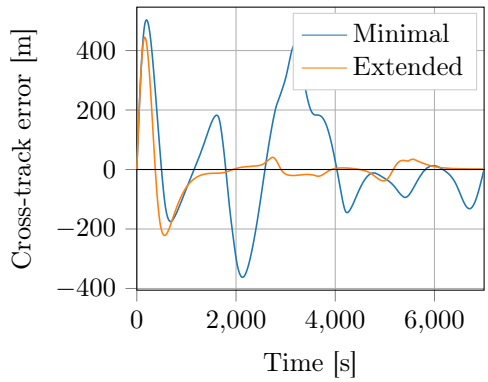


(b) Cross-track error.

Figure 36: Container curved path following when using minimal state and extended state representations.



(a) Path.



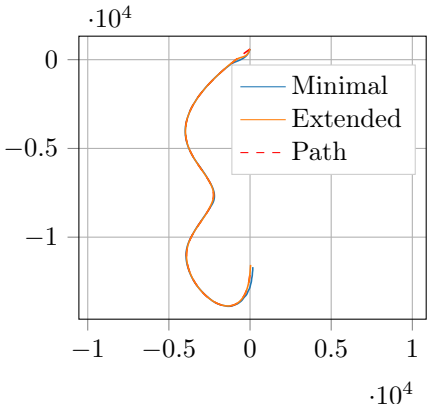
(b) Cross-track error.

Figure 37: Tanker curved path following when using minimal state and extended state representations.

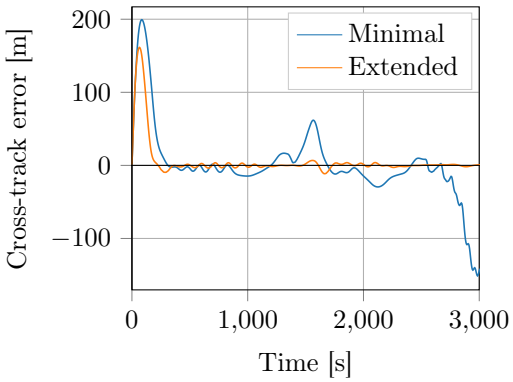
policy for the different state representations. While training with the extended representation was stable, training with the minimal representation was unstable, leading to the DRL algorithm to unlearn the policy.

4.2 Future work

From the observed behaveour, we see that we have achieved quite good results for using DRL on the path following problem. Our proposed method is able to both learn and optimize the path following behaviour, for all three vessel models, as well as outperforming the Line-of-Sight method in most cases. For future work, testing the methods for physical vessels would be quite interesting. Since the vessel dynamics for the models are quite similar to that of actual vessels, we would expect the method to work. However with the addition of measurement noise, and other environmental forces, the problem becomes more difficult, which may effect the performance of the DRL approach. For using the proposed method on physical vessels, an important consideration to take into account is the training, as letting the algorithm explore may be unfeasible do to safety concerns. One way of overcoming the problem is by performing training on simulations, and only applying a learned policy to a physical task after it performs sufficiently well on the simulation.

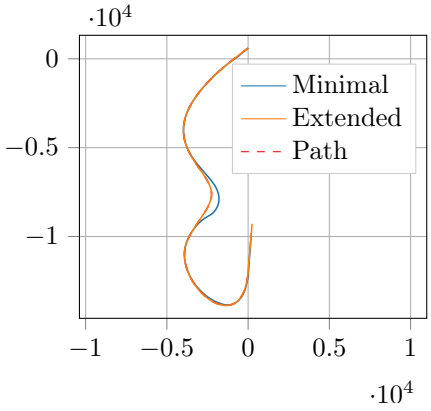


(a) Path.

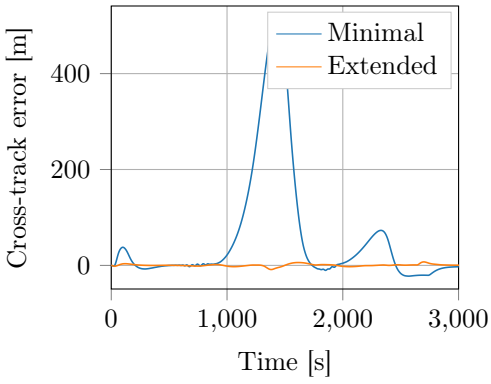


(b) Cross-track error.

Figure 38: Mariner curved path following when exposed to ocean currents, for minimal and extended state representations.

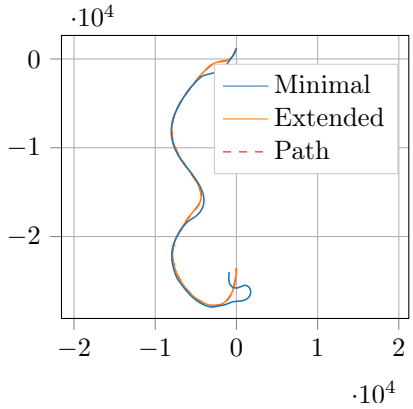


(a) Path.

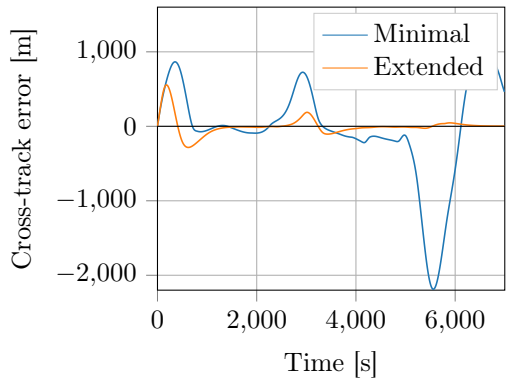


(b) Cross-track error.

Figure 39: Container curved path following when exposed to ocean currents, for minimal and extended state representations.



(a) Path.



(b) Cross-track error.

Figure 40: Tanker curved path following when exposed to ocean currents, for minimal and extended state representations.

In order to further improve the proposed method, other reward functions can be used, this may give more stable training, or different behavior which may be better suited for the path following task. Additional improvements may be found by investigating other state augmentations, and different sizes of the ANN used for the function approximators.

While the results for the DRL approach are fairly good, we do not have any guarantees that the vessel will converge to the track. While reinforcement learning inherently has stability build in, the fact that we are approximating the policy and value functions, which most likely will not converge to the true policy and value functions of the problem, we have no guarantee for stability. An other concern is robustness of the system, and how well the learned policy is able to handle noise. In general, the reinforcement learning framework is designed to handle stochastic systems, meaning the control algorithm can learn to still perform optimally under the influence of noise. This is however also dependant on the learned policy and value function converging to the true policy and value function, for which we do not have any guarantees when using function approximators. Showing stability would greatly improve the viability of the proposed method, and would therefore be an important area of research not only for our proposed method, but for DRL

in general.

DRL does not need to be confined to the path following problem. Other tasks such as path tracking, dynamic positioning and control allocation are other problems in the marine control community on which DRL can be applied. It may also be interesting to see how the proposed path following method works for problems in other domains, such as robotics.

Chapter 5

Conclusion

In this thesis we have presented a method for applying deep reinforcement learning to the problem of path following for marine vessels in uncertain environments. Unlike most existing methods, our proposed approach is a model-free optimal control method. This is achieved by letting the DRL algorithm explore the environment, and creating an internal representation in the form of an action value function, which is used in order to optimize the control policy. In order to achieve the path following task we have proposed two different reward functions, namely the boundary reward, and Gaussian reward, which both aim at minimizing the time until path convergence. While both functions have proven to work, the lower sparsity, and better convergence properties of the Gaussian reward function, makes it a better choice. In addition to the reward functions, we have also looked at two different state representations, namely the minimal and extended representation. Based on the results, the extended state representation gives significantly better results for the curved path following problem, while performing similarly to the minimal representation on the straight-line path following task.

In addition to presenting a method for applying DRL to the path following problem, we have also presented results from simulations on three different vessel models. Based on the results, the proposed method is quite promising. For the straight line following task, the DRL approach was able to outperform the Line-of-Sight guidance law for both the Mariner and Container vessel. For the curved path guidance

law, the DRL approach was able to outperform the Line-of-Sight approach for all three vessels, both with and without currents.

While the method has shown promising results, it does also have a few drawbacks that must be taken into consideration. One drawback is the limited expressive power of function approximators, which means that the policy and value function that are found are only approximations, and hence they will most likely never give the true optimal policy and value function. Increasing the depth and hence the complexity of the ANNs used as function approximators, increases the expressive power, however this makes training more difficult and computationally demanding. An additional concern with using DRL, is the stability of training. In some cases, training can be unstable, meaning that the function approximators are not converging to a good policy and value function. Unstable training is however a problem for most deep learning methods, and is an active area of research. An other concern for DRL in general is the lack of being able to prove stability, for the path following problem, this means we have no guarantees that the vessel will go to the path for all initial states. Despite these drawbacks, DRL is in general a very flexible framework, which has proven quite good at solving a wide variety of problems, including path following. Research on DRL has increased in the past few years, and new improvements will likely address some of the drawbacks, and make it more robust for a wide variety of problems, including path following.

Appendix A

Vessel models

A.1 3-DOF nonlinear manuvering model

The horizontal motion of a ship can be described by the motion in surge sway and yaw. For these systems the state vector $\boldsymbol{\nu} = [u, v, r]^\top$ and $\boldsymbol{\eta} = [x, y, \psi]$ are used. From Fossen (2011)[43] the system can then be described in vectorial form as:

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

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

where the mass and Coriolis matrices can be split into a rigid body and a added mass matrix, such that $\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A$ and $\mathbf{C} = \mathbf{C}_{RB} + \mathbf{C}_A$. And $\mathbf{R}(\psi) \in SO(3)$ is the rotation matix given by

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

For a three degree of freedom vessel model with an xz-plane symmetry, the rigid body kinematics computed on the craft centerline, reduce to the following

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & mx_g \\ 0 & mx_g & I_z \end{bmatrix} \tag{A.4}$$

$$\mathbf{C}_{RB} = \begin{bmatrix} 0 & 0 & -m(x_g r + v) \\ 0 & 0 & mu \\ m(x_g r + v) & -mu & 0 \end{bmatrix} \tag{A.5}$$

It is worth noticing that the surge is decoupled form sway and yaw in $\mathbf{M}_{RB}$, this is due to symmetry assumptions made above. For the added mass matrices computed in the same point as the rigid body dynamics, we get the following

$$\mathbf{M}_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix} \tag{A.6}$$

$$\mathbf{C}_A = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix} \tag{A.7}$$

For the dampening matrix $\mathbf{D}(\boldsymbol{\nu})$ it is difficult to find an analytical expression based on first principals. This is why a more pragmatic approach is used for curve fitting experimental data to a Taylor series describing the nonlinear terms $\boldsymbol{N}(\nu) = \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu}$, giving a nonlinear Abkowitz model [50]. Including the control surfaces $\boldsymbol{u}$, we get the following nonlinear model

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{A.8}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{N}(\boldsymbol{\nu}, \boldsymbol{u}) = 0 \tag{A.9}$$

A.1.1 Mariner model

The mariner model is a 3-DOF surface vessel based on the work of M.S. Chislett and J. Stroem-Tejsen [47], and released as a MATLAB model in the MSS Toolbox [2]. The vessel takes in the control rudder angle δ_c, and has the following vessel dynamics:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{A.10}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}\boldsymbol{N}(\boldsymbol{\nu}, \delta) \tag{A.11}$$

$$\dot{\delta} = \begin{cases} \delta_c - \delta \text{ If } \delta <= \delta_{\max} \\ 0 \text{ otherwise} \end{cases} \tag{A.12}$$

The mass matrix is given as

$$\mathbf{M} = \begin{bmatrix} \frac{L}{U^2} & 0 & 0 \\ 0 & \frac{L}{U^2} & 0 \\ 0 & 0 & \frac{L^2}{U^2} \end{bmatrix} \left(\begin{bmatrix} m & 0 & 0 \\ 0 & m & mx_g \\ 0 & mx_g & I_z \end{bmatrix} + \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix} \right) \tag{A.13}$$

where $U = \sqrt{u^2 + v^2}$ is the absolute velocity of the vessel. The nonlinear terms are based on the Abkovitz model described above, giving the following polynomial

vector:

$$\boldsymbol{N}(\boldsymbol{\nu},\delta) = \begin{bmatrix} X(\boldsymbol{\nu},\delta) \\ Y(\boldsymbol{\nu},\delta) \\ N(\boldsymbol{\nu},\delta) \end{bmatrix} \tag{A.14}$$

where the polynomials describing the nonlinear components, are given as:

$$\begin{aligned} X(\boldsymbol{\nu},\delta) =& X_u u + X_{uu}u_r^2 + X_{uuu}u_r^3 + X_{vv}v_r^2 + X_{rr}r^2 + X_{rv}rv_r \\ & X_{\delta\delta}\delta^2 + X_{u\delta\delta}u_r\delta^2 X_{v\delta}v_r\delta + X_{uv\delta}u_r v_r\delta \\ Y(\boldsymbol{\nu},\delta) =& Y_v v_r + Y_r r + Y_{vvv}v_r^3 + Y_{vvr}v_r^2 r + Y_{vu}v_r u_r + Y_{ru}ru_r + Y_\delta\delta + \\ & Y_{\delta\delta\delta}\delta^3 + Y_{u\delta}u_r\delta + Y_{uu\delta}u_r^2\delta + Y_{v\delta\delta}v\delta^2 + Y_{vv\delta}v^2\delta + \\ & (Y_0 + Y_{0u}u_r + Y0_{uu}u_r^2) \\ N(\boldsymbol{\nu},\delta) =& N_v v_r + N_r r + N_{vvv}v_r^3 + N_{vvr}v_r^2 r + N_{vu}v_r u_r + N_{ru}ru_r + \\ & N_\delta\delta + N_{\delta\delta\delta}\delta^3 + N_{u\delta}u_r\delta + N_{uu\delta}u_r^2\delta + N_{v\delta\delta}v_r\delta^2 + \\ & N_{vv\delta}v_r^2\delta + (N_0 + N_{0u}u_r + N_{0uu}u_r^2) \end{aligned}$$

For the mariner vessel, the parameters are given in Table 6, it should also be noted that the velocities $u_r = u/U$ and $v_r = v/U$ used in the nonlinear terms, are relative velocities, with respect to the absolute velocity.

A.1.2 Tanker model

The tanker model is a 3-DOF surface vessel based on the work of Van Berlekom, W.B. and Goddard, T.A. [48], and released as a MATLAB model in the MSS Toolbox [2]. The vessel takes in the control vector $\boldsymbol{u}$ consisting of the desired rudder angle δ_c, and desired shaft velocity n_c. The vessel dynamics of the tanker are given as:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{A.15}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}\boldsymbol{N}(\boldsymbol{\nu},\delta) \tag{A.16}$$

$$\dot{\delta} = \begin{cases} \delta_c - \delta \text{ If } \delta <= \delta_{\max} \\ 0 \text{ otherwise} \end{cases} \tag{A.17}$$

$$\dot{n} = \frac{60}{T_m}(n_c - n) \tag{A.18}$$

The mass matrix is given as

$$\mathbf{M} = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{33} \end{bmatrix} \tag{A.19}$$

Term	Value	Term	Value	Term	Value
$X_{\dot{u}}$	$-42e-5$	$Y_{\dot{v}}$	$-748e-5$	$N_{\dot{v}}$	$4.646e-5$
X_u	$-184e-5$	$Y_{\dot{r}}$	$-9.354e-5$	$N_{\dot{r}}$	$-43.8e-5$
X_{uu}	$-110e-5$	Y_v	$-1160e-5$	N_v	$-264e-5$
X_{uuu}	$-215e-5$	Y_r	$-499e-5$	N_r	$-166e-5$
X_{vv}	$-899e-5$	Y_{vvv}	$-8078e-5$	N_{vvv}	$1636e-5$
X_{rr}	$18e-5$	Y_{vvr}	$15356e-5$	N_{vvr}	$-5483e-5$
$X_{\delta\delta}$	$-95e-5$	Y_{vu}	$-1160e-5$	N_{vu}	$-264e-5$
$X_{u\delta\delta}$	$-190e-5$	Y_{ru}	$-499e-5$	N_{ru}	$-166e-5$
X_{rv}	$798e-5$	Y_δ	$278e-5$	N_δ	$-139e-5$
$X_{v\delta}$	$93e-5$	$Y_{\delta\delta\delta}$	$-90e-5$	$N_{\delta\delta\delta}$	$45e-5$
$X_{uv\delta}$	$93e-5$	$Y_{u\delta}$	$556e-5$	$N_{u\delta}$	$-278e-5$
		$Y_{uu\delta}$	$278e-5$	$N_{uu\delta}$	$-139e-5$
		$Y_{v\delta\delta}$	$-4e-5$	$N_{v\delta\delta}$	$13e-5$
L	160.93	$Y_{vv\delta}$	$1190e-5$	$N_{vv\delta}$	$-489e-5$
m	$798e-5$	Y_0	$-4e-5$	N_0	$3e-5$
I_z	$39.2e-5$	Y_{0u}	$-8e-5$	N_{0u}	$6e-5$
x_g	-0.023	Y_{0uu}	$-4e-5$	N_{0uu}	$3e-5$

Table 6: Mariner parameter value table

The nonlinear terms are based on the Abkovitz model described above, where the the nonlinear terms are given by the following polynomial vector:

$$\boldsymbol{N}(\boldsymbol{\nu},\delta) = \begin{bmatrix} X(\boldsymbol{\nu},\delta) \\ Y(\boldsymbol{\nu},\delta) \\ N(\boldsymbol{\nu},\delta) \end{bmatrix} \tag{A.20}$$

where the polynomials describing the nonlinear components, are given as:

$$\begin{aligned}
X(\boldsymbol{\nu}, \boldsymbol{u}) =& 1/L(X_{uu}u^2 + Ld_{11}vr + X_{vv}v^2 + X_{cc\delta\delta}|c|c\delta^2 \\
&+ X_{cc\beta\delta}|c|c\beta\delta + Lg_T(1-t) + X_{uuz}u^2 z \\
&+ LX_{vrz}vrz + X_{vvzz}v^2 z^2) \\
Y(\boldsymbol{\nu}, \boldsymbol{u}) =& 1/L(Y_{uv}uv + Y_{vv}|v|v + Y_{cc\delta}|c|c\delta + Ld_{22}ur \\
&+ Y_{cc\beta\beta\delta}|c|c\beta\beta\delta + Y_T g_T L \\
&+ LY_{urz}urz + Y_{uvz}uvz + Y_{vvz}|v|vz \\
&+ Y_{cc\beta\beta\delta z}|c|c|\beta|\beta|\delta|z) \\
N(\boldsymbol{\nu}, \boldsymbol{u}) =& N_{uv}uv + LN_{vr}|v|r + N_{cc\delta}|c|c\delta + Ld_{33}ur \\
&+ N_{cc\beta\beta\delta}|c|c|\beta|\beta|\delta| + LN_T g_T \\
&+ LN_{urz}urz + N_{uvz}uvz + LN_{vrz}|v|rz \\
&+ N_{cc\beta\beta\delta z}|c|c|\beta|\beta|\delta|z
\end{aligned}$$

where β, g_T and c are computed as the following

$$\begin{aligned}
\beta &= \frac{v}{u} \\
g_T &= 1/LT_{uu}u^2 + T_{un}un + LT_{nn}|n|n \\
c &= \sqrt{c_{un}un + c_{nn}n^2};
\end{aligned}$$

and the parameters for the tanker are given in Table 7.

A.2 4-DOF nonlinear manuvering model

Describing the horizontal motion of a surface vessal, a 3-DOF model is often enough. In certain situations however, including the roll dynamics is beneficial in order to get a more accurate vessel model, as the roll dynamics are often strongly coupled to the sway and yaw dynamics. Similarly to the 3-DOF case, the 4-DOF model can be written on vectorial form as:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\nu)\boldsymbol{\nu} \tag{A.21}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{A.22}$$

The major differences from the 3-DOF model is the addition of the pitch angle ϕ, giving the state vector:

$$\boldsymbol{\nu} = [u, v, \psi, \phi]^\top \tag{A.23}$$

Term	Value	Term	Value	Term	Value
d_{11}	2.020	m_{11}	1.050	T_{uu}	−0.00695
d_{22}	−0.752	m_{22}	2.020	T_{un}	−0.00063
d_{33}	−0.231	m_{33}	0.1232	T_{nn}	0.0000354
X_{uuz}	−0.0061	Y_T	0.04	N_T	−0.02
X_{uu}	−0.0377	Y_{vv}	−2.400	N_{vr}	−0.300
X_{vv}	0.3	Y_{uv}	−1.205	N_{uv}	−0.451
X_{udotz}	−0.05	Y_{vdotz}	−0.387	N_{rdotz}	−0.0045
X_{uuz}	−0.0061	Y_{urz}	0.182	N_{urz}	−0.047
X_{vrz}	0.387	Y_{vvz}	−1.5	N_{vrz}	−0.120
X_{ccdd}	−0.093	Y_{uvz}	0	N_{uvz}	−0.241
X_{ccbd}	0.152	Y_{ccd}	0.208	N_{ccd}	−0.098
X_{vvzz}	0.0125	Y_{ccbbd}	−2.16	N_{ccbbd}	0.688
		Y_{ccbbdz}	−0.191	N_{ccbbdz}	0.344
t	0.22	c_{un}	0.605	L	304.8
T_m	50	c_{nn}	38.2	g	9.8
T	18.46				

Table 7: Tanker parameter value table

and the addition of pitch in the rotation matrix.

$$\boldsymbol{J}(\boldsymbol{\eta}) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{A.24}$$

A.2.1 Container model

The Container model is a 4-DOF surface vessel based on the work of Son og Nomoto [49], and released as a MATLAB model in the MSS Toolbox [2]. The vessel model takes in the control vector $\boldsymbol{u}$ consisting of the desired rudder angle δ_c, and desired shaft velocity n_c, and returns the state derivatives $\dot{\boldsymbol{\nu}}$ and $\dot{\boldsymbol{\eta}}$. The vessel dynamics of the tanker are given as:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{A.25}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}\boldsymbol{N}(\boldsymbol{\nu}, \delta) \tag{A.26}$$

$$\dot{\delta} = \begin{cases} \delta_c - \delta \text{ If } \delta <= \delta_{\max} \\ 0 \text{ otherwise} \end{cases} \tag{A.27}$$

$$\dot{n} = \frac{60}{T_m}(n_c - n) \tag{A.28}$$

The mass matrix is given as

$$\mathbf{M} = \begin{bmatrix} m+m_x & 0 & 0 & 0 \\ 0 & m+m_y & -m_y l_y & m_y \alpha_y \\ 0 & -m_y l_y & I_x + J_x & 0 \\ 0 & m_y \alpha_y & 0 & I_z + J_z \end{bmatrix} \begin{bmatrix} \frac{L}{U^2} & 0 & 0 & 0 \\ 0 & \frac{L}{U^2} & 0 & 0 \\ 0 & 0 & \frac{L^2}{U^2} & 0 \\ 0 & 0 & 0 & \frac{L^2}{U^2} \end{bmatrix} \tag{A.29}$$

where $U = \sqrt{u^2 + v^2}$ is the absolute velocity. The nonlinear terms are based on the Abkovitz model described above, where the the nonlinear terms are given by the following polynomial vector:

$$\boldsymbol{N}(\boldsymbol{\nu}, \delta) = \begin{bmatrix} X(\boldsymbol{\nu}, \delta) \\ Y(\boldsymbol{\nu}, \delta) \\ K(\boldsymbol{\nu}, \delta) \\ N(\boldsymbol{\nu}, \delta) \end{bmatrix} \tag{A.30}$$

where the polynomials describing the nonlinear components, are given as:

$$\begin{aligned}
X(\boldsymbol{\nu}, \boldsymbol{u}) =& X_{uu}u^2 + (1-t)T + X_{vr}vr + X_{vv}v^2 + X_{rr}r^2 + X_{\phi\phi}\phi^2 + \\
& c_{RX}F_N \sin(\delta) + (m+m_y)vr \\
Y(\boldsymbol{\nu}, \boldsymbol{u}) =& Y_v v + Y_r r + Y_p p + Y_\phi \phi + Y_{vvv}v^3 + Y_{rrr}r^3 + Y_{vvr}v^2 r + \\
& Y_{vrr}vr^2 + Y_{vv\phi}v^2\phi + Y_{v\phi\phi}v\phi^2 + Y_{rr\phi}r^2\phi + \\
& Y_{r\phi\phi}r\phi^2 + (1+a_H)F_N \cos(\delta) - (m+m_x)ur \\
K(\boldsymbol{\nu}, \boldsymbol{u}) =& K_v v + K_r r + K_p p + K_\phi \phi + K_{vvv}v^3 + K_{rrr}r^3 + K_{vvr}v^2 r + \\
& K_{vrr}vr^2 + K_{vv\phi}v^2\phi + K_{v\phi\phi}v\phi^2 + K_{rrphi}r^2\phi + \\
& K_{r\phi\phi}rphi^2 - (1+a_H)z_R F_N \cos(\delta) + m_x l_x ur - WG_M\phi \\
N(\boldsymbol{\nu}, \boldsymbol{u}) =& N_v v + N_r r + N_p p + N_\phi \phi + N_{vvv}v^3 + N_{rrr}r^3 + N_{vvr}v^2 r + \\
& N_{vrr}vr^2 + N_{vv\phi}v^2\phi + N_{v\phi\phi}v\phi^2 + N_{rr\phi}r^2\phi + \\
& N_{r\phi\phi}r\phi^2 + (x_R + a_H x_H)F_N \cos(\delta)
\end{aligned}$$

utilizing the following calculations

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

where the parameters are given in Table 8.

Term	Value	Term	Value	Term	Value
m	0.00792	m_x	0.000238	m_y	0.007049
I_x	0.0000176	α_y	0.05	l_x	0.0313
l_y	0.0313	I_x	0.0000176	I_z	0.000456
J_x	0.0000034	J_z	0.000419	x_G	0
B	25.40	d_F	8.00	g	9.81
d_A	9.00	d	8.50	∇	21222
K_M	10.39	K_B	4.6154	A_R	33.0376
Δ	1.8219	D	6.533	G_M	$0.3/L$
ρ	1025	t	0.175	T	0.0005
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}	0.0	c_{pr}	0.0	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 8: Container parameter value table

Bibliography

[1] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971. URL: http://arxiv.org/abs/1509.02971, arXiv:1509.02971.

[2] Fossen, T. I. & Perez, T. 2004. Marine systems simulator (mss). URL: http://www.marinecontrol.org.

[3] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. 2016. *Deep learning*, volume 1. MIT press Cambridge.

[4] SNAME. 1950. Nomenclature for treating the motion of a submerged body through a fluid.

[5] Lapierre, L., Soetanto, D., & Pascoal, A. 2003. Nonlinear path following with applications to the control of autonomous underwater vehicles. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 2, 1256–1261. IEEE.

[6] Fredriksen, E. & Pettersen, K. Y. 2006. Global κ-exponential way-point maneuvering of ships: Theory and experiments. *Automatica*, 42(4), 677–687.

[7] Lekkas, A. M. & Fossen, T. I. 2014. Integral los path following for curved paths based on a monotone cubic hermite spline parametrization. *IEEE Transactions on Control Systems Technology*, 22(6), 2287–2301.

[8] Caharija, W., Pettersen, K. Y., Bibuli, M., Calado, P., Zereik, E., Braga, J., Gravdahl, J. T., Sørensen, A. J., Milovanović, M., & Bruzzone, G. 2016. Integral line-of-sight guidance and control of underactuated marine vehicles: Theory, simulations, and experiments. *IEEE Transactions on Control Systems Technology*, 24(5), 1623–1642.

[9] Aguiar, A. P. & Hespanha, J. P. 2007. Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty. *IEEE Transactions on Automatic Control*, 52(8), 1362–1379.

[10] Lekkas, A. M. & Fossen, T. I. 2014. Trajectory tracking and ocean current estimation for marine underactuated vehicles. In *Control Applications (CCA), 2014 IEEE Conference on*, 905–910. IEEE.

[11] Moe, S., Pettersen, K. Y., Fossen, T. I., & Gravdahl, J. T. 2016. Line-of-sight curved path following for underactuated usvs and auvs in the horizontal plane under the influence of ocean currents. In *Control and Automation (MED), 2016 24th Mediterranean Conference on*, 38–45. IEEE.

[12] Fossen, T. I., Breivik, M., & Skjetne, R. 2003. Line-of-sight path following of underactuated marine craft. *IFAC Proceedings Volumes*, 36(21), 211–216.

[13] Skjetne, R., Jørgensen, U., & Teel, A. R. 2011. Line-of-sight path-following along regularly parametrized curves solved as a generic maneuvering problem. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, 2467–2474. IEEE.

[14] Sutton, R. S. & Barto, A. G. 1998. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[15] Bertsekas, D. P. 2012. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4th edition.

[16] Bertsekas, D. P. & Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*. Athena Scientific.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. 2013. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602. URL: http://arxiv.org/abs/1312.5602, arXiv:1312.5602.

[18] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489. URL: https://www.nature.com/nature/journal/v529/n7587/pdf/nature16961.pdf.

[19] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783. URL: http://arxiv.org/abs/1602.01783, arXiv:1602.01783.

[20] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347. URL: http://arxiv.org/abs/1707.06347, arXiv:1707.06347.

[21] Martinsen, A. B. & Lekkas, A. M. 09 2018. Straight-path following for underactuated marine vessels using deep reinforcement learning.

[22] Frank, R. 1957. The perceptron a perceiving and recognizing automaton. *Cornell Aeronautical Laboratory, Buffalo, NY, USA, Tech. Rep*, 85–460.

[23] Cybenko, G. Dec 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303–314. URL: https://doi.org/10.1007/BF02551274, doi:10.1007/BF02551274.

[24] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 541–551.

[25] Werbos, P. 1975. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University.

[26] Ng, A. Y. 2004. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, 78. ACM.

[27] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580. URL: http://arxiv.org/abs/1207.0580, arXiv:1207.0580.

[28] Russell, S. & Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[29] Bellman, R. 1957. *Dynamic programming*. Princeton University Press, Princeton, N.J.

[30] van Hasselt, H. & Wiering, M. A. 2007. Convergence of model-based temporal difference learning for control. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, 60–67. IEEE. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.235&rep=rep1&type=pdf.

[31] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533. URL: https://www.nature.com/nature/journal/v518/n7540/pdf/nature14236.pdf.

[32] Levine, S., Finn, C., Darrell, T., & Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39), 1–40. URL: http://www.jmlr.org/papers/volume17/15-522/15-522.pdf.

[33] Peters, J. 2010. Policy gradient methods. *Scholarpedia*, 5(11), 3698. revision #137199. doi:10.4249/scholarpedia.3698.

[34] Williams, R. J. May 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3), 229–256. URL: https://doi.org/10.1007/BF00992696, doi:10.1007/BF00992696.

[35] Kakade, S. M. 2002. A natural policy gradient. In *Advances in neural information processing systems*, 1531–1538.

[36] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. 2015. Trust region policy optimization. *CoRR*, abs/1502.05477. URL: http://arxiv.org/abs/1502.05477, arXiv:1502.05477.

[37] Grondman, I., Busoniu, L., Lopes, G. A. D., & Babuska, R. Nov 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6), 1291–1307. doi:10.1109/TSMCC.2012.2218595.

[38] Li, Y. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

[39] Ratcliffe, D., Devlin, S., Kruschwitz, U., & Citi, L. 2017. Clyde: A deep reinforcement learning doom playing agent.

[40] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438. URL: http://arxiv.org/abs/1506.02438, arXiv:1506.02438.

[41] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783. URL: http://arxiv.org/abs/1602.01783, arXiv:1602.01783.

[42] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. 2013. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602. URL: http://arxiv.org/abs/1312.5602, arXiv:1312.5602.

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

[44] Shneydor, N. 1998. *Missile Guidance and Pursuit: Kinematics, Dynamics and Control*. Horwood engineering science series. Horwood Pub. URL: https://books.google.no/books?id=8X7CQgAACAAJ.

[45] Nocedal, J. & Wright, S. J. 2006. *Numerical Optimization*. Springer.

[46] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[47] Chislett, M., Strøm-Tejsen, J., & og earodynamisk laboratorium. Hydrodynamics Department, H. 1965. *Planar Motion Mechanism Tests and Full-scale Steering and Manoeuvring Predictions for a Mariner Class Vessel*. Report (Hydro- og earodynamisk laboratorium. Hydrodynamics Department). Hydro- and Aerodynamics Laboratory. URL: https://books.google.no/books?id=zCmsPQAACAAJ.

[48] Van Berlekom, W. B. & Goddard, T. A. 1972. Maneuvering of large tankers.

[49] Son, K.-H. & Nomoto, K. 1982. 5. on the coupled motion of steering and rolling of a high-speed container ship. *Naval Architecture and Ocean Engineering*, 20, 73–83.

[50] Abkowitz, M. A. Lectures on ship hydrodynamics–steering and manoeuvrability. Technical report, 1964.