Simen Sem Øvereng

Dynamic Positioning using Deep Reinforcement Learning

Master's thesis in Marine Cybernetics
Supervisor: Dong Trong Nguyen
June 2020

Simen Sem Øvereng

Dynamic Positioning using Deep Reinforcement Learning

Master's thesis in Marine Cybernetics
Supervisor: Dong Trong Nguyen
June 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MSC THESIS DESCRIPTION SHEET

Name of the candidate:	Simen Sem Øvereng
Field of study:	Marine Control Systems
Thesis title (Norwegian):	Dynamisk Posisjonering ved bruk av Dyp Forsterkende Læring
Thesis title (English):	Dynamic Positioning using Deep Reinforcement Learning

Background

Classic methods for Dynamic Positioning of surface vessels often include combining a motion controller with a thrust allocation method, using optimization or simplified methods for finding approximate solutions. For ships with various thruster setups, especially those including rotatable azimuth thrusters and thrusters with rudders, the optimization face challenges such as non-convex domains, and computationally demanding solvers must be used in order to find a solution. Therefore, simplified methods for thrust allocation have been used commercially in order to always find a solution which is feasible, of the cost of a potentially suboptimal solution. This thesis sets to investigate how advances in machine learning can be used to train a model for controlling a vessel in an accurate and energy efficient manner, without heavy computation on board during Dynamic Positioning operations. The work was initiated by a project thesis (TMR4510) during the fall of 2019; “Solving Thrust Allocation with Machine Learning”, which investigated how machine learning methods could be used in order to solve the thrust allocation problem specifically. Acceptable results emerged from using Supervised Learning when compared to classic methods, but it was concluded with that it was difficult to create a good dataset in order to formulate the problem properly, especially in terms of temporal constraints. It was therefore proposed that Reinforcement Learning seemed like a flexible methodology, which is the motivation in this master’s thesis.

Work description

1. In addition to the literature of classic methods for control of surface vessels in Dynamic Positioning operations, a background and literature review is to be performed to provide information and relevant references on Reinforcement Learning methods for large and continuous state- and action spaces.

2. Update theory section from Project Thesis by expanding and outlining important details within Reinforcement Learning which shall be used in the thesis’ implementations, especially focusing on state-of-the-art algorithms which uses Neural Networks as function approximators.

3. (Re)formulate the problem in a way that allows for implementation and testing of the algorithms from the literature studies. Select one Reinforcement Learning algorithm which shall be benchmarked against the classic motion control with quadratic programming and pseudoinverse thrust allocation, which was used in the Project Thesis.

4. Implement the algorithm to perform training in simulations, and testing in ROS (Robot Operation System, the architecture on which the control system of ReVolt is running on).

5. Verify and validate performance, and compare with the classic methods. Adjust, tune and retrain correspondingly.

6. Test performance on the real-life ReVolt model to compare results from simulator.

7. Create a draft of a publishable paper of the work done, given that promising results are obtained early. The completion of the master thesis itself should be the main focus.

Specifications

The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the various steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 60-80 A4 pages, from introduction to conclusion, unless otherwise agreed upon. It shall be written in English (preferably US) and contain the following elements: Title page, abstract, acknowledgements, thesis specification, list of symbols and acronyms, table of contents, introduction with objective, background, and scope and delimitations, main body with problem formulations, derivations/developments and results, conclusions with recommendations for further work, references, and optional appendices. All figures, tables, and equations shall be numerated. The original contribution of the candidate and material taken from other sources shall be clearly identified. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. *natbib* Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct. Such practice is taken very seriously by the university and will have consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed upon.

The thesis shall be submitted with a printed and electronic copy to the main supervisor, with the printed copy signed by the candidate. The final revised version of this thesis description must be included. The report must be submitted according to NTNU procedures. Computer code, pictures, videos, data series, and a PDF version of the report shall be included electronically with all submitted versions.

Start date: 15 January 2020 **Due date:** 10 June 2020

Supervisor: Dong Trong Nguyen
Co-advisor(s): Geir Hamre (DNVGL)

Trondheim, 19.06.2020

[illegible]

Dong Trong Nguyen, Supervisor

Preface

This thesis presents the work done during TMR4930 Marine Technology - Master's Thesis at NTNU (30 ECTS), and represents the final delivery for a Master of Science within Marine Cybernetics. The work builds on a project thesis from the Fall of 2019, and was done from January to June in 2020. The thesis was written in its entirety by Simen Sem Øvereng.

The work was motivated by looking for ways of improving motion control and thrust allocation for surface vessels during Dynamic Positioning. The approach was to utilize recent advances within Machine Learning and where the project thesis utilized Supervised Learning, the master's thesis improved performance by using Deep Reinforcement Learning. The goal was to create a controller which accomplished Dynamic Positioning in terms of positional accuracy, while being energy efficient, and low on wear and tear of the actuators.

The resulting control method achieved higher positional accuracy while being more energy efficient when compared to two classic control schemes in several testing scenarios in simulation, where the simulations also showed that the Deep Reinforcement Learning method was quite robust. It was also directly deployed on a real life model ship, with the result being good positional accuracy, though with some oscillatory behavior presumably coming from hardware issues of the ship model's bow propeller. The work concluded with that learning to efficiently control a vessel using Deep Reinforcement Learning is not only possible, but also quite effective.

The reader is expected to have knowledge of basic control theory, including the development and design of motion controllers and the principles of control allocation, especially for marine vessels. It is also beneficial to have basic knowledge of modern machine learning. However, the machine learning theory is explained in quite detail as it was not expected that all stakeholders in the academic and industrial societies within marine technology were familiar with it. A good intuition on gradients from Calculus is also beneficial.

Acknowledgments

I would like to thank my supervisor, Prof. Dong Trong Nguyen, for providing me with good references, relevant rules and regulations, and discussions on methodology and test scenarios. I would also like to thank my co-supervisor, Geir Hamre (DNVGL) for taking his time to discuss problem formulations and solutions, and for giving feedback on the thesis. A big thank you also goes to PhD candidate Andreas Bell Martinsen from the Department of Engineering Cybernetics at NTNU, for both theoretical aspects of Reinforcement Learning, and good conversations about implementation details. The same goes to Ella-Lovise Rørvik, Machine Learning Engineer at TrønderEnergi, who has helped with useful discussions upon reward shaping and implementational nuances.

The thesis has not been possible without the help from several people at DNVGL supporting me with practical issues. Especially, I would like to thank Jon Arne Glomsrud for help with the interface between Python and Java in order to communicate efficiently with the simulator, César Augusto Ramos de Carvalho for support with simulators for various use cases, and Tom Arne Pedersen for support and hardware debugging on the real ship model of ReVolt.

I would also like to thank my fellow students Magnus Oanes Knædal and Knut Turøy for great cooperation during the work with ReVolt, especially during the sea trials. And last but not least, a big thank you goes to friends and family, providing unconditional support and encouragement during my studies at NTNU.

[illegible]

SSØ
June 24, 2020

Abstract

Classic methods for Dynamic Positioning (DP) of surface vessels often consists of first calculating desired forces and moments to exert on the vessel by using a motion controller, followed by a thrust allocation scheme for translating these forces and moments into individual thruster commands. Common thrust allocation methods usually performs this translation by calculating a pseudoinverse matrix, or by using various formulations of an objective function for an optimization problem. In optimization, several solutions might exist that satisfies the desired forces/moments due to the vessels being overactuated, and additional objectives and/or constraints are usually added for finding a final solution at each time step. This can be challenging in terms of computationally demanding solvers due to the potentially complex configuration of actuators, including azimuth thrusters, variable pitch propellers, tunnel thrusters, rudders etc. This thesis was set to investigate how advances in machine learning could be used to train a machine learning model in simulation for the development a control scheme which was both positionally accurate and energy efficient, while avoiding heavy computations on board during DP operations. The thesis was written in collaboration with DNVGL, which gave access to technical advisory, in addition to both a simulation platform and a physical ship model on which testing of algorithms could be performed.

This thesis built on work from a project thesis from 2019 [1]. There it was found that using supervised learning for thrust allocation could give adequate results in terms of accurate translation of desired forces and moments. It was however found that it was difficult to formulate the the constraints and boundaries on the azimuth thrusters' force production and angular rates using supervised learning. In this thesis, another type of machine learning called Deep Reinforcement Learning (DRL) was used. It replaced both the motion controller and the thrust allocation, translating a desired vessel pose directly into individual thruster commands. This was obtained by by training the DRL model in a simulator while attempting to maximize return over time from a reward function. The neural networks which were used for representing the DRL model were trained using the digital twin of the ReVolt ship model, and the Proximal Policy Optimization (PPO) algorithm. The entire training regimen for the DRL method was created, and a (to the author's knowledge) novel reward function was used for positional accuracy, being a *multivariate, Gaussian* reward function.

The DRL method was tested against classic methods, using a Proportional-Integral-Derivative (PID) motion controller with feedforward, and two different thrust allocation methods. The first thrust allocation method was a proprietary method given by DNVGL based on calculating a pseudoinverse, while the other was a method based on Quadratic Programming (QP), implemented by the author. All methods were tested against each other in various test scenarios in the simulator for evaluation of accuracy, robustness and energy efficiency. In addition, a sea trial was performed on the physical ship model of ReVolt to evaluate the difference between performance in simulator and in real life.

The simulation results showed that the DRL method was robust to changes in the desired pose which was larger than what it had been trained on while performing DP, and performed better positional accuracy than the classic methods while doing so. Test scenarios with no environmental loads enabled showed that the DRL method was both better in terms of positional accuracy, energy usage, and wear and tear than the classic methods. After enabling a constant ocean current, the resulting positional accuracy was similar among the methods, while the energy efficiency was lowest when using the DRL method. A station-keeping test with large wind, wave and current loads showed that the DRL method was able to maintain acceptable positional accuracy while station-keeping in a larger sea state than what the method had been trained on. In the sea trial, the DRL method's positional accuracy was found good, having comparable performance to the similar test in simulation. Due to hardware issues related to the ship model's bow propeller, the vessel movement was more oscillatory, resulting in a substantial increase of the energy usage when compared to the simulation since the method had to compensate for the hardware issue.

The thesis concluded with that DRL's potential for solving tasks with continuous control signals which requires accuracy and energy efficiency is realizable both in terms of good performance in the simulated tests, as well as by performing well when transitioning from simulations to real life. Recommended future work was divided into two branches. The first aimed at improving real life performance by retraining the DRL model trained in simulation by using transfer learning on board the real ship model in order to learn details of the hardware which was not modeled in the digital twin. The second was to consider newer methods within the area of providing safety and stability guarantees during training of control policies in order to develop a notion of the *worst case* performance of DRL systems.

Sammendrag

Klassiske metoder for Dynamisk Posisjonering (DP) av overflatefartøy består ofte av en kombinasjon av en bevegelseskontroller som regner ut ønskede krefter og moment på fartøyet, etterfulgt av en algoritme for kontrollallokering som oversetter disse til individuelle kommandoer til hver aktuator. Dagens metoder for kontrollallokering baserer seg ofte enten på utregningen av en pseudoinvers, eller ulike formuleringer av en objektivfunksjon som skal optimeres. Avhengig av formulering så kan optimeringsproblemene ha flere løsninger som tilfredsstiller de ønskede kreftene/momentene siden fartøyet er overaktuert, og flere objektivfunksjoner og/eller begrensninger kan legges til for å velge mellom disse løsningene. Dette kan skape utfordringer på grunn av at de fartøyene som utfører DP kan ha et bredt spekter av oppsett av ulike typer aktuatorer, som for eksempel roterbare propeller, propeller med eller uten justerbar stigning, tunnellpropellere, og propeller med ror bak, noe som kan gjøre optimeringsprogrammene tidkrevende å løse. Denne masteroppgaven tar for seg å utforske hvordan nyere tids metoder innen maskinlæring kan brukes for å trene opp en modell i simulering for utvikling av et kontrollsystem som er både nøyaktig og energieffektivt, og som unngår tunge beregninger i sanntid ombord i fartøyet under oppgaver som krever DP. Oppgaven ble skrevet i samarbeid med DNVGL, noe som ga tilgang på rådgivning, en simulator med en digital tvilling av en skipsmodell, samt den ekte skipsmodellen for fysisk testing av ulike algoritmer.

Arbeidet i oppgaven bygger på en prosjektoppgave fra 2019 [1]. I den ble det konkludert at det å bruke Veiledet Læring til å løse kontrollallokering fungerte tilstrekkelig med tanke på avviket mellom ønskede og kommanderte krefter/moment. Det var derimot vanskelig å generere gode datasett som formulerte begrensninger på aktuatorene, som for eksempel å ivareta maksimal kraftproduksjon og rotasjonshastigheten av roterbare propeller. I denne oppgaven ble derimot Dyp Forsterkende Læring (DFL) brukt, ikke kun til kontrollallokering, men også som erstatning for bevegelseskontrolleren, noe som ble gjort gjennom at algoritmen trente seg selv i simulatoren ved å forsøke å maksimere gevinst over tid med hensyn til en ukjent belønningsfunksjon. Det betyr at DFL-modellen oversatte ønsket posisjon og retning på skipet direkte til individuelle aktuatorkommandoer. De nevrale nettverkene som representerte DFL-modellen ble trent på den digitale tvillingen av ReVolt ved å bruke en algoritme kalt Proximal Policy Optimization (PPO). Hele treningsregimet ble utviklet, og en (til forfatterens kjennskap) ny type belønningsfunksjon ble brukt for å belønne små posisjonsavvik: en multivariabel gauss-funksjon.

Det opptrente kontrollsystemet ble testet mot klassiske metoder metoder. Det besto av en eksisterende implementasjon av en PID-regulator (Proporsjonal Integrasjon Derivasjon) med foroverkopling for bevegelseskontroll, samt to klassiske metoder for kontrollallokering. Den første algoritmen var en metode levert av DNVGL som baserte seg på å finne en pseudoinvers, mens den andre baserte seg på optimering av en kvadratisk objektivfunksjon som ble implementert av forfatteren. Alle metodene ble først testet mot hverandre i simulatoren for å teste nøyaktighet, robusthet og energibruk. I tillegg ble det utført en test der DFL-modellen ble testet med krefter fra vind, bølger og strøm fra mange ulike retninger for å evaluere nøyaktighet og robusthet under vanskelige forhold. Til slutt ble DFL-modellen, som kun ble trent i simulator, testet i en sjøprøve om bord den fysiske skipsmodellen av ReVolt for å evaluere ytelse fra simulator til virkelighet.

Simuleringsresultatene viste at DFL-modellen var robust mot endringer i ønsket posisjon og kurs som var større enn det den hadde blitt trent med, og oppnådde større nøyaktighet i følging av referansesignalet enn de klassiske metodene. Testscenarioene uten ytre belastninger viste at DFL-modellen var mer nøyaktig i minimering av avvik fra referansesignalet, hadde lavere energibruk, samt lavere slitasje av aktuatorene sammenlignet med de klassiske metodene. Etter å aktivere ytre belastninger i form av en konstant havstrøm ble resultatene tilnærmet lik nøyaktighet i holding av posisjon og kurs, men DFL-modellen oppnådde lavere energiforbruk og mindre slitasje. Simuleringstester av posisjonsholding når både vind, bølger og havstrømmer var aktivert viste at DFL-modellen holdt referanseposisjon og -kurs med tilstrekkelig nøyaktighet, selv om modellen kun hadde blitt trent i stille vann uten ytre belastninger. Sjøprøven viste at DFL-modellen ga god nøyaktighet posisjonsmessig sammenlignet med en tilsvarende test i simulator. På grunn av problematikk i tilknytning til maskinvaren til baugpropellen var bevegelsene på skipsmodellen oscillerende, hvilket resulterte i høyere energiforbruk sammenlignet med i simulering ettersom modellen måtte kompensere for maskinvareproblemet.

Oppgaven konkluderte med at potensialet til DFL til kontrolloppgaver med kontinuerlige kontrollsignal som krever nøyaktig posisjonskontroll samt lavt energiforbruk er realiserbart, både i form av god ytelse i alle tester i simulering, samt i form av samsvar mellom simulering og fysiske tester. Anbefalt videre arbeid ble delt i to grener. Den første tok sikte på å forbedre ytelsen i fysiske teseter ved å ta i bruk den opptrente modellen fra simulator, og gjennomføre overførende læring slik at algoritmen får mulighet til å lære seg detaljer omkring maskinvaren på den virkelige modellen som ikke har blitt modellert i den digitale tvillingen. Den andre grenen omhandlet å ta i bruk nyere metoder for å sørge for stabilitet og sikkerhet underveis i treningen av kontrollalgoritmer for å utvikle et mål på den *verst mulige* ytelsen av DFL-systemer.

Table of contents

List of Figures

List of Tables

Abbreviations

AI	=	Artificial Intelligence
ANN	=	Artificial Neural Network
API	=	Application Programming Interface
CG	=	Center of Gravity
CPU	=	Central Processing Unit
DFL	=	Dyp Forsterkende Læring (Norwegian)
DL	=	Deep Learning
DOF	=	Degree(s) of Freedom
DP	=	Dynamic Positioning
DRL	=	Deep Reinforcement Learning
FFNN	=	Feedforward Neural Network
GPU	=	Graphics Processing Unit
GUI	=	Graphical User Interface
IPI	=	Iterative Pseudoinverse
MDP	=	Markov Decision Process
ML	=	Machine Learning
mp-QP	=	Multi-Parametric Quadratic Programming
MSE	=	Mean Squared Error
NED	=	North-East-Down
PD	=	Proportional-Derivative (controller)
PG	=	Policy Gradient
PID	=	Proportional-Integral-Derivative (controller)
PPO	=	Proximal Policy Optimization
QP	=	Quadratic Programming
RL	=	Reinforcement Learning
RMSE	=	Root Mean Squared Error
RPM	=	Revolutions per Minute
RPS	=	Revolutions per Second
SGD	=	Stochastic Gradient Descent
TA	=	Thrust Allocation

Chapter 1

Introduction

1.1 Autonomy At Sea

Automation has been increasing the production accuracy and capabilities of many industries during the last century. Currently, increased levels of automation, computational capabilities and inter-robotic communication is opening a landscape of *autonomy*. Autonomy is enabling robotic systems to not only replace traditional automated tasks, but also to perform the decision making during missions by themselves. The application of autonomy includes fields all the way from subsea to interplanetary. Shipping was given another dimension for Norway during the 60's with the discovery of oil on the Norwegian continental shelf, where Norway's shipping fleet today has become one of the top 10 fleets in the world(1). As the field of autonomous and remotely operated vessels is rapidly expanding [2], more stakeholders are showing interest in the developments.

1.2 Motivation

As DNVGL is a major stakeholder in the ship classification business, technological developments needs to be understood in order for the company to be able to verify the performance and security of new types of systems emerging within the shipping industry. Therefore, DVNGL developed the ReVolt concept vessel [3], which was brought to life by creating a 1:20 scaled ship model, which is displayed in Figure 1.1. Today, there exists a control system on-board the ReVolt ship model based on classical methods, implemented by previous graduate students. As developments within artificial intelligence (AI) are being used more and more in many academic and industrial areas, including control systems, it is expected that they will have to be verified by class societies such as DNVGL. This motivated the research carried out by this thesis, as most modern control techniques being used for vessel control today are based on theory developed before the explosive attention to AI emerged, and more knowledge on how to ensure performance of AI-based systems was needed.

(1)https://stats.unctad.org/handbook/MaritimeTransport/MerchantFleet.html

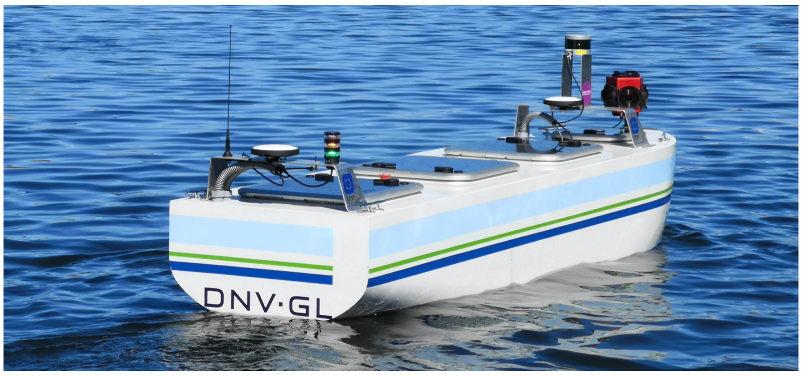


Figure 1.1: The ReVolt model ship in Dorabassenget, Trondheim, Norway.

In the recent years, applications of AI have become prevalent in a lot of industrial sectors. The current state of AI has been displayed, among other examples, by Google's DeepMind and OpenAI, where the latter recently achieved beating the world champion team in the complex computer game "Dota 2", using their model called OpenAI Five. This model was trained by playing itself, accumulating an estimated 250 years of simulated experience per day[(2)]. In guidance and control, Waymo, Optimus Ride and Tesla (among others) has been employing AI in various way for making cars that are currently capable of navigating through regular traffic by themselves[(3)]. However, at sea, there seems to be only a few stakeholders currently working on achieving results on the level and scale as is being displayed in computer games and on the roads. Another motivation of this thesis was therefore to inspire more work combining the fields of AI and marine/naval engineering.

A last point of motivation was the fact that the ReVolt vessel is planned to run only on electric power, and hence energy consumption becomes important as the capability of today's electric energy storage units on marine vessels does not have the capabilities of traditional, fossil fuel based energy storage. This underlined the motivation for finding energy efficient control methods.

1.3 Literature Review

As an introduction to the relevant work done on the area of motion control and thrust allocation (TA) previously (especially for surface vessels), the following provides a literature review on classic methods, in addition to relevant work within AI that is connected to the proposed approaches in this thesis.

1.3.1 Classic Thrust Allocation

Work on Dynamic Positioning (DP) control systems has been carried out since the 1960s, being accelerated in the development within multivariable control, and the introduction of the iterative optimal state estimator (Kalman filter) in the 1970's [4]. DP systems are traditionally used for low-speed applications, such as stationkeeping while controlling surge, sway and yaw motions, but current work is being done within *hybrid control*, which looks at developing one control system to be used in both low-speed and high-speed applications (e.g. work done by Brodtkorb [5]).

[(2)]https://openai.com/blog/how-to-train-your-openai-five/
[(3)]https://www.tesla.com/autopilot

- Early work from 1997 by Sørdalen [6] introduced the notion improving the ”extended thrust” formulation, in which force vectors from each rotatable thrusters are decomposed into surge and sway components, making it possible to calculate a pseudoinverse of the thruster effector model while adding weights on each thruster, thus finding a solution to an unconstrained optimization problem. This method formed the basis for a lot of approaches within TA in the years following due to its computational efficiency, but offers no guarantees on optimality in terms of minimum fuel usage or other explicit goals. It is however used commercially with some modifications [7]. Sørdalen proposed an improvement including filtering techniques for reducing extensive azimuth rotations, and singular value decomposition techniques for avoiding high thruster inputs close to singular configurations.

- Based on their development on an algorithm in 2003 called Multi-Parametric Quadratic Program (mp-QP) [8], Johansen, Fossen, and Tøndel [9] demonstrated in 2005 the performance and efficiency compared to other optimization techniques. Tests with a model vessel displayed a well performing allocation scheme, robust to several constraints incurring the optimization objective.

- In 2008, Leavitt [10] proposed work on the limitations of the linear optimization formulation $Ax \leq b$, as there is no way of handling forbidden sectors, rudders, non-zero minimum thrust etc., e.g. by proposing that rudder thrust regions can be split into several convex regions for Quadratic Programs (QP). Weaknesses in mp-QP formulations used until this point was summarized in the presentation as: (1) azimuthing thrusters will rotate excessively in low sea states, (2) spoil zones cannot effectively be handled (3) non-zero minimum thrust cannot effectively be handled for azimuthing thrusters, (4) explicit rate limits for azimuths are not possible, (5) power limiting can only be handled for simple cases, but with prohibitive offline computation times in a commissioning setting. An improved solution was proposed, including online computations of several mp-QP cases, enforcing constraints if the best mp-QP results consisted of a breach of the constraints imposed on the thrusters, as shown in Figure 1.2.

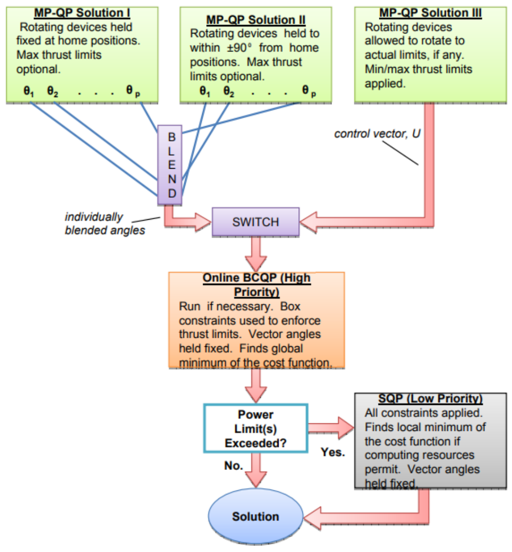

MP-QP Solution I
Rotating devices held fixed at home positions. Max thrust limits optional.
θ_1 θ_2 . . . . θ_p
MP-QP Solution II
Rotating devices held to within ±90° from home positions. Max thrust limits optional.
θ_1 θ_2 . . . . θ_p
MP-QP Solution III
Rotating devices allowed to rotate to actual limits, if any. Min/max thrust limits applied.
BLEND
control vector, U
individually blended angles
SWITCH
Online BCQP (High Priority)
Run if necessary. Box constraints used to enforce thrust limits. Vector angles held fixed. Finds global minimum of the cost function.
Power Limit(s) Exceeded?
Yes.
No.
SQP (Low Priority)
All constraints applied. Finds local minimum of the cost function if computing resources permit. Vector angles held fixed.
Solution

Figure 1.2: Proposed improvement from regular mp-QP for thrust allocation. Courtesy: Leavitt [10].

- Rindarøy and Johansen [11], 2013, proposed a constrained control allocation strategy with rudders, where the mp-QP approach is applied to four pre-determined convex zones as a work-around for the non-convex thruster regions that appears when thrusters are used in configuration with rudders. Figure 1.3 shows a flowchart of the intended structure of the suggested TA strategy. This method is also applicable to illegal zones for azimuths, where the illegal zones are removed from the thruster regions, and the remaining regions are pre-computed as independent convex regions, requiring explicitly formulating these regions beforehand. As stated in the conclusion of the publication: "*The non-convexity of the problem is handled by a mixed-integer-like convex reformulation that allows the solution to be pre-computed in a piecewise linear form using multi-parametric quadratic programming. Benefits compared to online numerical optimization are higher software reliability and less computational effort. A weakness of the approach is that the pre-computed explicit solution does not easily admit reconfiguration, unless several cases are pre-computed in which case computer memory requirements will be limiting the practical applicability of the approach*".

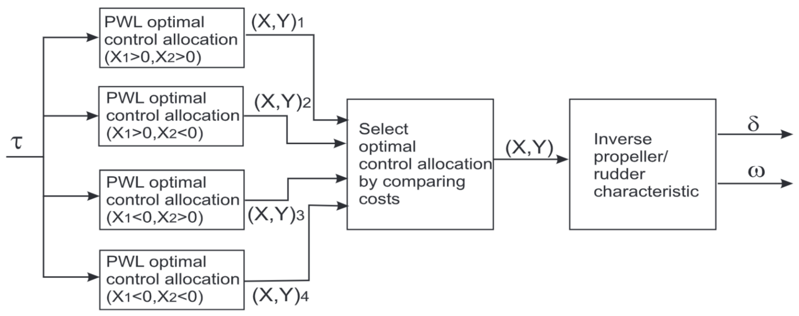

τ
PWL optimal control allocation (X1>0,X2>0)
(X,Y)1
PWL optimal control allocation (X1>0,X2<0)
(X,Y)2
PWL optimal control allocation (X1<0,X2>0)
(X,Y)3
PWL optimal control allocation (X1<0,X2<0)
(X,Y)4
Select optimal control allocation by comparing costs
(X,Y)
Inverse propeller/ rudder characteristic
δ
ω

Figure 1.3: Thrust allocation scheme using multiple mp-QP with different convex regions. Courtesy: Rindarøy and Johansen [11].

- A comprehensive survey of control allocation techniques for marine vessels was published in 2013 by Fossen and Johansen [12], where the authors lined out the options and challenges of TA in general. A walk-through of solutions for linear actuator models such as Linear and Quadratic Programming (LP and QP, respectively) were discussed. Of special interest for this thesis, it is stated in Chapter 2.2.3 that "*Optimal solutions of LPs are found at vertices of the feasible set. A consequence of this is that the LP-based method tends to favor the use of a smaller number of effectors, while methods based on a quadratic cost function and ∞-norm tends to use all effectors, but to a smaller degree, (Bodson 2002). This seems to be the main reason why error minimization approaches are based on quadratic programming in most cases.*". The survey's concluding notes were that control allocation is fairly well understood for linear actuator models, where optimization techniques provides several advantages, but having cons such as difficult numerical implementations and verification of performance. Fewer methods for non-linear models exist, but the paper summarizes how e.g. Lyapunov-based design has been used to propose some promising results.

- Model Predictive Control (MPC) has also been emerging as a viable alternative for control allocation due to the recent advances in both hardware and algorithms. MPC is a control method for both planning and control of dynamic systems while concurring with physical constraints, relying on a mathematical representation of the system dynamics. The planning aspect of the MPC is what sets it apart from simpler methods, since it can be used for optimizing the current time step's control input with respect to a finite time horizon into the future. Relevant work includes the likes of Vermillion, Sun, and Butts [13] which used MPC for control allocation of an overactuated dynamic system for translating desired forces and moments into actuator commands, Naderi, Khaki Sedigh, and Johansen [14] which used an extension of the pseudoinverse methodology for ensuring feasible control allocation by using MPC, and Veksler et al. [15] which performed control allocation of an overactuated marine vessel in a DP task by replacing the traditional setup of a separate motion control algorithm and a control allocation algorithm, using MPC to encapsulate these entities into one.

1.3.2 Learning Based Approaches

Not much literature exists within the use of AI towards marine applications, but within other domains such as robotics, a large amount of attention has been drawn to the potential of utilizing advances in AI for solving optimal control problems. It should be noted that AI is a very broad term, and could in some sense apply to the classic allocation methods in Section 1.3.1 also. However, the methods mentioned in the following all share a "learning" strategy, meaning that they are not explicitly programmed to solve a problem, but rather "learn" how to solve it. Learning methods are further explained in Chapter 3.

- Luman, Roh, and Hong [16] proposed in 2015 using genetic algorithms to solve power optimal TA. Genetic/evolutionary algorithms learn by simulating different populations (representing different sets of parameters), and learn by selecting an amount of the best sets as a baseline for the next evolution - the methods are inspired by natural selection [17, 18]. In their work, generation no. 300 of the evolutionary population was able to learn the proposedly optimal TA, where the only objective function used was to minimize the power consumption of the four thrusters on the vessel being used.

- In 2016, Wu, Ren, and Zhang [19] proposed an energy optimal TA method based on adaptive bee colony algorithm, which differs from traditional optimization methods by mimicking the behavior of bee swarms, each bee representing a subset of the decision variables used in the optimization, with the goal of the swarm converging to a set of decision variables which gives an optima to the problem to be solved.

- Supervised Learning as means of control has been explored in the context of control theory several times. Monga and Moehlis [20] gave a large overview of its applications, and showed Supervised Learning's ability to be used in various applications within optimal control of underactuated systems outputting bang-bang policies(4), and discusses the interpretability of such algorithms due to their "black-box" nature.

- In 2018, Skulstad et. al [21] implemented a static Neural Network (NN), trained with supervised learning, which transformed virtual force commands from a PID motion controller into individual thruster commands on a vessel, thus using the neural network as a TA scheme. The NN was implemented with a single layer, using 3 input units, 12 hidden sigmoid units, and 6 output units, the latter representing the number of actuators on the vessel in question. No azimuthing thrusters were used.

(4)"Bang-bang" policies refers to action selection yielding either 0, maximum, or minimum of the actuator limits: https://www.semanticscholar.org/topic/Bang-bang-control/1288573

The NN was trained on 7800 training samples, gathered from running simulations in the Offshore Simulator Centre[(5)]. The training samples consisted of manually labeled data where the thruster commands had been set manually.

- In the project thesis on which this master's thesis is built upon [1], Supervised Learning was also attempted employed as a means of translating the desired forces from a PID motion controller into thruster commands, using a neural network similarly to [21], but allowing for rotatable azimuth thrusters. The conclusion was that a regular feed-forward neural network trained with Supervised Learning showed limited use case in terms of dynamic control using optimality conditions such as minimizing fuel and/or wear and tear on the thrusters. The reason was that it was hard to formulate any time-dependent constraints, causing the trained agent to often violate thruster rates, and to have a very aggressive usage of the thrusters in general. The trained neural network for TA was able to perform the station-keeping task it was trained to do - it just did not do it very efficiently when compared to classic methods such as the pseudoinverse or quadratic programming. It was proposed that further work should either be put into using recurrent neural networks instead, so that temporal constraints could be added, or to shift focus from Supervised Learning to Reinforcement Learning (RL), arguing for that the latter approach seems more promising due to the results obtained in publications presented in the following.

- Since 2015, two main algorithms within Deep Reinforcement Learning (DRL) has become the baseline for applications to decision making in complex games, and for robotic manipulation. Being demonstrated on OpenAI's gym repository[(6)] which includes simulation environments for games and robotics, Deep Deterministic Policy Gradient (DDPG) [22] and Trust Region Policy Optimization (TRPO) [23] are the algorithms upon which most work on DRL is being compared to as of this date.

- In 2017, Hwangbo et al. [24] used DRL to learn to control a quadcopter by using a deterministic policy for the selection of the four rotor's thrust output. They compared their performance to known algorithms at the time, such as DDPG and TRPO, in which their algorithm was able to compete with TRPO in terms of final performance while beating TRPO in terms of learning rate, measured by performance versus number of learning steps made in the simulation environment.

- Around the same time as Hwangbo et al. published their work, OpenAI published a more sample-efficient version of TRPO which aimed at simplifying some of its key concepts, naming it Proximal Policy Optimization (PPO) [25]. The PPO algorithm was tested by Lopes et al. [26], which demonstrated the sample efficiency of the PPO algorithm, tested on a simulated quadrotor. The results showed that it was possible to train a DRL based agent with PPO to obtain a policy which was capable of controlling the aircraft by deciding the thrust output of all four rotors. As the PPO algorithm uses a stochastic policy in order to achieve sufficient exploration during training, the authors removed the action-selection noise during testing in order to reduce the variance of the action selection of the trained agent, which was deemed successful, and improved the quadcopters ability to reach setpoints from different extreme initial conditions, in addition to improving the ability of trajectory following.

[(5)] https://osc.no/
[(6)] https://gym.openai.com/

- Also Bohn et al. [27] implemented a successful PPO algorithm, showing that it was capable of stabilizing an inherently unstable system being a fixed-wing aircraft. They used an actor-critic setup with a unique actor network for each of the actuators, and showed that it was possible to use Convolutional Neural Networks as a means of improving temporal abilities of the DRL method.

- Looking towards marine applications, in 2018, Martinsen [28] used the DRL algorithm Deep Deterministic Policy Gradient (DDPG) for learning a one-dimensional policy, controlling the rudder angle of marine vessels for path-following while assuming the vessel to have constant total speed. The resulting controller was able to perform as well or better when compared to a standard in today's guidance methods, namely Line-of-Sight (LOS) guidance.

- Kjærnli [29] compared in 2018 the use of DDPG and PPO for learning the calculations of motion control signals for a Remotely Operated Vehicle (ROV). The work concluded that the PPO algorithm learned faster, and was the one learning stable policies. The agent was able to perform station keeping in 3 DOF under the assumption that the ROV could be assumed stable in roll and pitch, but included noisy action selections.

- Knudsen [30] further explored the possibility of creating a controller for an ROV with a deep reinforcement learning method in 2019. The algorithm of choice was DDPG which was considered as an alternative to extensive and precise modeling of the dynamics; learning from interactions with the environment instead. Knudsen showed that it was possible to use Deep Learning for translational motion control in conjunction with a PD controller for rotational motion control in order to develop a combined controller which was able to station keep in 6 DOF. The simulation results were promising, and the experiments showed that the controller was able to to control surge and sway of the ROV. The learned controller calculated the desired force vector only, before a separate, existing TA scheme was used to employ the physical forces and moments on the ROV.

- Vallestad [31] built on work by [28] in 2019, and used DDPG to develop a control algorithm for path following, including surge and heading control, while performing collision avoidance. In 2020, Rørvik [32] implemented both the DDPG and PPO algorithms for learning the thruster commands directly in order to solve the task of docking of an autonomous vessel; the learned controller directly translated a positional set-point into thruster commands instead of having a motion controller and a thrust allocation scheme divided into two entities. The conclusion was that both algorithms was able to learn the docking scenarios, with PPO displaying the most stable results, and having the lowest error on the performance metrics Rørvik selected. In addition, the noisy nature of the resulting control policy was discussed, as the output from the learned policies was quite fluctuating.

1.3.3 Previous Work on ReVolt

On the ReVolt model ship, there has been written master's degrees directly concerning the control system, leading to how it is currently set up. In addition, auxiliary work has been done, using ReVolt as demonstration platform.

- The first thesis that was written with ReVolt in focus was done by Alfheim and Muggerud in 2016 [33]. They developed both a guidance system, a navigation system, and a control system. The control system was made primarily as a DP system, using a PID controller with feedforward for calculating the desired forces that was to be put on the vessel, and a thrust algorithm for calculating the corresponding thruster commands in order to physically generate these forces. The thrust allocation algorithm was based on the pseudoinverse, provided by DNVGL. The code from DNVGL calculated the individual thruster's forces, which Alfheim and Muggerud translated into propeller speeds by using that the thruster force was modeled as $T = Kn|n|$, solving for n using $n = sign(T/K)\sqrt{|T/K|}$.

- In 2018, Havnegjerdet [34] expanded the control and guidance system, implementing a speed and heading controller, combining them to enable path following. A path following scheme was also implemented, using LOS guidance with lookahead-based steering.

- In 2020, Martinsen et al. [35] proposed a control scheme based on model-based Reinforcement Learning for motion control, demonstrated on the ReVolt platform, both in simulations and during real life sea trials. The scheme combined ideas from optimal control and RL in order to first estimate model-parameters used in a feedforward control law, followed by using RL to develop a feedback control law. The resulting controller calculated the desired forces to put on the vessel, and used the existing thrust allocation. The results was a control scheme which performed well in both DP and trajectory tracking, in addition to being versatile in terms of ease of applicability for a wide range of systems.

1.4 Objectives

The goals and research questions which this thesis was set to investigate was comprised into the following objectives:

- Review a wide variety of literature within the field of classic control theory, control allocation, and find where deep learning theory fits into this framework.

- Propose one or more machine learning methods for control of marine surface vessels. The control scenarios was to be restricted to low speed applications such as DP, focused on station-keeping with the possibility of reacting to minor setpoint changes of the vessel's desired pose.

- Verify performance by numerical simulations on a digital twin of the 1:20 scale ship model of DNVGL's concept ship ReVolt by using the Open Simulation Platform (OSP). It was also desirable to perform tests on the physical ship model to evaluate how transferable performance in the simulations was compared to real life performance.

- Compare the performance of the DRL method to other, classic methods which were implemented as a part of the control system on ReVolt by previous master's students. This was to be done both with and without environmental forces for evaluation of the DP capabilities of the different methods.
- Evaluate DRL as a methodology for control of marine vessels in general, both in terms of development and training, robustness, accuracy and energy efficiency.

1.5 Contributions

The main contributions of this thesis to the both the scientific, industrial and classification communities can be summarized as the following:

- An implementation of a control scheme which replaces the traditional structure of using a motion controller in combination with a thrust allocation scheme, combining them both into one entity by using DRL. By using a neural network, the selection of actuator commands is done in negligible time, enabling usage on systems within a large range of computational power.
- A discussion upon reward shaping, concluding in a reward function seemingly new to the field.
- Rigorous testing of the method in simulation, displaying good energy efficiency while being positionally accurate when compared to classic methods, in addition to displaying good results on a physical ship model during a sea trial.
- An open-source implementation of the learning algorithm, the trained model, the final system that was used for testing in the Robotic Operating System (ROS), in addition to documentation on how to use to code for further work. This can be found at https://github.com/simensov/ml4ca.
- A draft of an abstract to be submitted to the *Ocean Engineering* journal, added to Appendix G.

1.6 Outline

Chapter 1 gives the reader an introduction to the foundation this thesis builds on. **Chapter 2** gives a general, theoretical overview of the classic methods within control systems and thrust allocation. **Chapter 3** gives a brief overview of machine learning and how deep neural networks works, followed by a theoretical walk-through of the Deep Reinforcement Learning methods used in the thesis. **Chapter 4** outlines the simulation platform and the physical system of ReVolt, in addition to the software framework used for implementations. **Chapter 5** explains the thrust allocation method provided by DNVGL, the development of the Quadratic Programming thrust allocation, and the considerations taken when developing the Deep Reinforcement Learning system. **Chapter 6** presents the test scenarios used, the results from both simulations and real life testing, and discussions of these results. **Chapter 7** discusses the results on a more general basis and evaluates the characteristics of the Deep Reinforcement Learning system, while **Chapter 8** gives a conclusion of the work. Lastly, **Chapter 9** proposes points of interest for further work within DRL.

Chapter 2

Control Theory for Marine Vessels

2.1 Notation and Reference Frames

The notation used in this report follows conventions from the Society of Naval Architects and Marine Engineers (SNAME), summarized in Table 2.1 for the relevant Degrees of Freedom (DOF). Figure 2.1 shows a visualization of the conventions. For analyzing the motion of ReVolt, the geographical reference frame North-East-Down (NED) and the body-fixed reference frame was used. NED is chosen as a tangent plane to the surface of the earth, and positions within the frame is denoted (x_n, y_n, z_n), where the x_n-axis points towards true north, the y_n-axis points east and the z_n-axis points downwards. The body-fixed reference frame is denoted (x_b, y_b, z_b), where the x_b-axis points in the longitudinal direction of the vessel, the y_b-axis in the transverse direction, and z_b points in the direction normal to the x_b,y_b-plane. $(x_b, y_b, z_b) = (0, 0, 0)$ was defined to be located in the Center of Gravity (CG) of the vessel. In this thesis, y_b was positively defined in the starboard direction, hence z_b was positively defined downwards. In addition, the term *position* referred to a 2 or 3 dimensional coordinate, while *pose* was used for vectors consisting of both position and the orientation of the vessel relative to true North.

Table 2.1: Notation by SNAME for 3 DOF models.

DOF	Position in Euler coordinates	Linear and angular velocities	Forces and moments
Surge	x	u	X
Sway	y	v	Y
Yaw	ψ	r	N

To avoid confusion with a *reward*, which is used widely in Reinforcement Learning terminology, the yaw rate will be denoted as $\dot{\psi}$ instead of r. r_t will therefore denote a reward at time t, and $\dot{\psi}_t$ denote the yaw rate at time t.

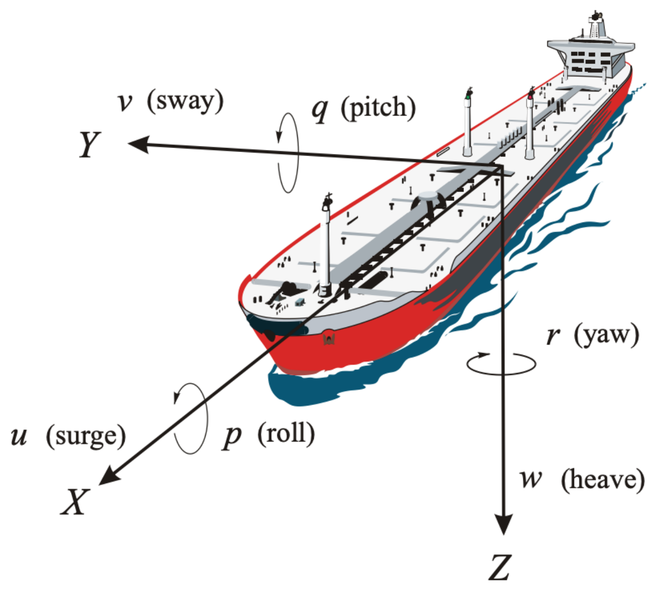


Figure 2.1: Definition of surge, sway, heave, roll, pitch and yaw modes of motion in body-fixed frame. Courtesy: Sørensen [4].

2.2 DP Control Plant Model for 3 DOF

A control plant model is a simplified mathematical model of the vessel in the relevant DOFs for the control system being designed. Its purpose is to describe the vessel dynamics, and it is used to design the controller. Control plant models for Dynamic Positioning (DP) assume low vessel speeds, and are valid for up to approximately 2 m/s, according to Fossen [36]. In the following, all roll, pitch and heave motions has been neglected, which results in a 3-DOF model, consisting of surge, sway and yaw. The non-linear equations of motion can be presented as in Equation (2.1), where $\boldsymbol{C}(\boldsymbol{\nu}) \in \mathbb{R}^{3x3}$ is the Coriolis and centripetal matrix, $\boldsymbol{D}(\boldsymbol{\nu}) \in \mathbb{R}^{3x3}$ is the damping matrix, $\boldsymbol{M} \in \mathbb{R}^{3x3}$ is the system inertia, consisting of the added mass and the rigid body inertia. $\boldsymbol{\tau}_i$ represents the different forces acting on the system, $\boldsymbol{\eta}$ is the NED-frame position, while $\boldsymbol{\nu}$ is the velocity of the vessel expressed in the body-frame. The numeric values of these matrices for ReVolt were found experimentally by Alfheim and Muggerud [33], and is added to Appendix A.

$$\begin{gathered}\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \\ \boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \sum_i \boldsymbol{\tau}_i\end{gathered} \tag{2.1}$$

The kinetic equation of motions is restricted to rotation about the z-axis due to the negligible roll and pitch motions during DP, such that the rotation matrix used to transform velocities from the body-frame to the NED-frame can be simplified to Equation (2.2).

$$\boldsymbol{R}(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

2.3 Control System

The control system on board a vessel is the system responsible for calculating and exerting the forces necessary to act on the vessel in order to keep a certain position when station-keeping, or following a certain trajectory when maneuvering. In addition, it needs to take into account thruster limitations with respect to acceleration limits, rotational limits etc. The control hierarchy can thus be divided into three levels (according to Sørensen [4]): the guidance system, the high-level plant control, and low-level thruster control. An overview of the control system can be seen in the block diagram presented in Figure 2.2.

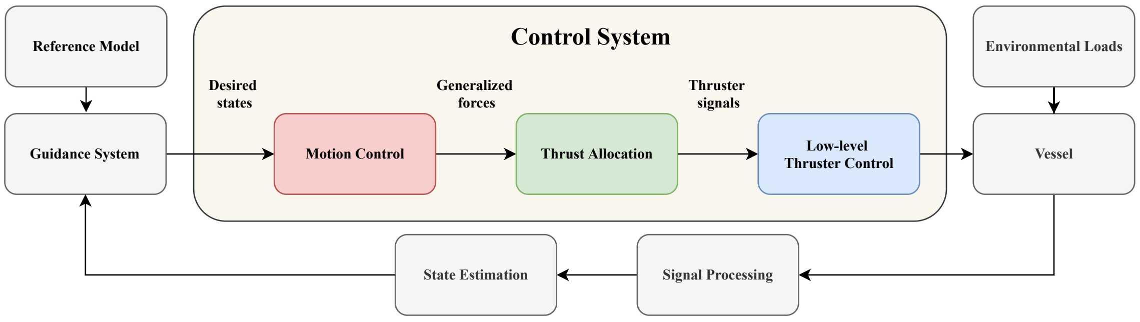


Figure 2.2: Block diagram of the components involved in control of a vessel, with the control system outlined.

2.3.1 Guidance system

The high-level control (being the motion control and the thrust allocation) receives a setpoint to act on with respect to the vessel's current state, and the guidance system is responsible for input to the controller. To do so, a combination of setpoint generation and a reference model is used.

Setpoint Generation

The desired states for the vessel can be generated in difference ways. For general maneuvering, the setpoints might be generated by a motion-planning algorithm, or they can be set manually by an on board or remote operator. For station-keeping situations, the setpoint is constant or slowly varying in low speed operations, but for path-following, the setpoints change more drastically over time. Path-following algorithms can be found in detail in Fossen [36], chapter 10. The generation of desired states needs a component that computes reference signals to the vessel's high-lever controller, which is where a reference model is used.

Reference Model

When the DP system of a ship attempts to control a vessel from one position to another, it might receive large changes between setpoints, potentially causing large accelerations of the actuators. This could induce large sudden movements of the vessel, as well as increased actuator wear and tear. In order for the control system to work with smooth trajectories, mechanisms can be implemented to generate references that incrementally move the setpoints away from the current position, resulting in a more predictable behavior of the positioning procedure. An example of such a mechanism is a reference model. Although the name includes ”model”, it can be seen as filter, filtering the setpoints and computing references for the control algorithm. The reference model thus feeds references into the controller that get incrementally closer and closer to the global setpoint given from the setpoint generation mentioned above. During the tuning of the reference model, there needs to be made a compromise between accuracy and performance. Tuning the model too aggressively might make the vessel's controller act very aggressively and quickly, but in turn, the accuracy of the reference following could be reduced. Equation (2.3) shows a mass-damper-spring system that is able to filter step changes in the reference position (in the NED frame) η_{ref}^n in order to achieve smooth trajectories for the desired vessel states. A more rigorous explanation of reference filters can be found in Fossen [36], Chapter 10.2.1. A reference model was already implemented on ReVolt [33], which is discussed in Section 5.1.2.

$$\begin{bmatrix} \dot{\eta}_d \\ \ddot{\eta}_d \\ \dddot{\eta}_d \end{bmatrix} = \begin{bmatrix} 0 & \mathrm{I} & 0 \\ 0 & 0 & \mathrm{I} \\ -\Omega^3 & -(2\Delta + \mathrm{I})\Omega^2 & -(2\Delta + \mathrm{I})\Omega \end{bmatrix} \begin{bmatrix} \eta_d \\ \dot{\eta}_d \\ \ddot{\eta}_d \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \Omega^3 \end{bmatrix} \eta_{\mathrm{ref}}^n \tag{2.3}$$

2.3.2 Motion Control

In order to move from the vessel's current state to the desired state proposed from the reference model, forces and moments has to act on the vessel in order for it to move. The size and direction of these forces is calculated in the motion controller. A typical implementation of such a controller for surface vessels for path- or trajectory following is to use a combination of a speed controller and a heading controller, where two different controllers work in synergy to maintain the wanted speed and attitude of the vessel. However, for DP systems, there is one controller unit responsible for maintaining position and attitude as the vessel's speed is low. The controller calculates the *generalized* forces that the vessel needs to be subject to in order to eliminate the error between the current and desired state. A typical implementation of a *feedback* controller is a proportional-integral-derivative (PID) controller, which consists of three terms which (1) aims to reduce the current deviation from a given setpoint (proportional term), (2) reduce the steady state deviation from a setpoint in case of a disturbance put on the system (integral term), and (3) the rate of change deviation (derivative term). The generalized forces are calculated in the body-frame, so the vessel's positional error $\tilde{\boldsymbol{\eta}}$ must be transformed from the NED-frame. The resulting control law becomes as shown in Equation (2.4),

$$\boldsymbol{\tau}_{PID}(t) = \begin{bmatrix} X_d \\ Y_d \\ N_d \end{bmatrix} = -\boldsymbol{K}_p\Big(\boldsymbol{R}^\top(\psi)\tilde{\boldsymbol{\eta}}(t)\Big) - \boldsymbol{K}_d\tilde{\boldsymbol{\nu}}(t) - \boldsymbol{K}_i\int_0^t \Big(\boldsymbol{R}^\top(\psi)\tilde{\boldsymbol{\eta}}(\tau)\Big)d\tau, \tag{2.4}$$

where $\boldsymbol{K}_p, \boldsymbol{K}_d$, and $\boldsymbol{K}_i$ represents the proportional-, derivative, and integral gains respectively, while $\boldsymbol{\tau}_{PID}$ represents the desired forces in surge, sway and yaw. The K-values are parameters that are tuned for proper stability and performance. Fossen [36], chapter 12, gives an example of useful tuning rules for stable PID controllers. On the ReVolt model ship, a PID controller was already developed and tuned [33]. In addition, a *feedforward* controller was implemented, using the mass-, damping-, and Coriolis matrices as explained combined with the desired vessel states to react directly to the desired states instead of only relying on the feedback control law from the PID to correct for errors between the desired and the vessel's states. In theory, if the modeling is done perfectly, a feedforward control law should be enough to control the vessel to the desired states *if* the external forces acting on it is known. However, due to modeling errors, non-linearities, and often unknown and varying external forces such as the sea state, the necessity for a correcting, feedback control law is needed. Assuming no known external forces on the vessel, the feedforward control law was formulated according to

$$\boldsymbol{\tau}_{FF} = \boldsymbol{M}\dot{\boldsymbol{\nu}}_{\boldsymbol{d}} + \boldsymbol{C}(\boldsymbol{\nu}_d)\boldsymbol{\nu}_d + \boldsymbol{D}(\boldsymbol{\nu}_d)\boldsymbol{\nu}_d. \tag{2.5}$$

The resulting control law $\boldsymbol{\tau}_c$ of the motion controller thus becomes the sum of the two: the feedforward control law for "predicting" the necessary control inputs for reaching the desired states, and the feedback control law for correcting for the resulting errors:

$$\boldsymbol{\tau}_c = \boldsymbol{\tau}_{FF} + \boldsymbol{\tau}_{PID}. \tag{2.6}$$

2.3.3 Thrust Allocation

In order to physically put forces on the vessel, the generalized forces has to be transcribed into individual commands to each of the vessels' thrusters, which might be fixed-pitch propellers, azimuth thrusters, water jets etc. This process is called Thrust Allocation (TA), and is classically implemented as an optimization problem. A mapping between the actual thrust forces $\boldsymbol{\tau}$ and the thruster forces $\boldsymbol{T}$ can be expressed several ways, but the most common one is the *linear actuator model*, as shown in Equation (2.7), where $\boldsymbol{x}_{thr}$ is the thruster states, and t denotes time.

$$\boldsymbol{\tau} = B(\boldsymbol{x}_{thr}, t)\boldsymbol{T} \tag{2.7}$$

B is called the *effectiveness matrix*, and contains the geometric setup which transforms the individual thrusters' forces, $\boldsymbol{T}$, into forces and moments acting on the ship, $\boldsymbol{\tau}$. Due to that the thrusters on ReVolt are all azimuth thrusters, B depends on the thruster rotations, and is therefore not constant unless the thruster angles are constrained to a fixed angle. The entire thrust model can therefore be modeled as in Equation (2.8). Note that $s\alpha_i$ and $c\alpha_i$ corresponds to the sine and cosine of α_i, respectively, and that the thrusters' distances from CG in the longitudinal and transverse directions are $l_{x,i}$ and $l_{y,i}$, which are shown in Figure 4.2 in Chapter 4.

$$\boldsymbol{\tau} = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} = \begin{bmatrix} c\alpha_1 & c\alpha_2 & c\alpha_3 \\ s\alpha_1 & s\alpha_2 & s\alpha_3 \\ l_{x,1}s\alpha_1 - l_{y,1}c\alpha_1 & l_{x,2}s\alpha_2 - l_{y,2}c\alpha_2 & l_{x,3}s\alpha_3 - l_{y,3}c\alpha_3 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{2.8}$$

Figure 2.3 gives a representation of the three elements of $\boldsymbol{\tau}$ in 3DOF. Going from Equation (2.8), the thrust allocation procedure is concerned with finding the thruster commands that produces $\boldsymbol{\tau} = \boldsymbol{\tau}_d$. Note that this is not necessarily achievable instantaneously at all time instances, so the thrust allocation can be formulated as an optimization problem, attempting to get $\boldsymbol{\tau}$ as close as possible to $\boldsymbol{\tau}_d$ at every time step. After the thruster angles and the individual forces has been found, they have to be translated into thruster commands, usually through a known model of the thruster forces as a function of thruster commands, as shown in Equation (2.9), where K_T is a coefficient found through empirical tests [37], ρ is the fluid density, and D is the propeller diameter. n_i represents the rotational velocity of the propeller, and is the signal which is commanded from the thrust allocation scheme.

$$T_i = K_T \rho D^4 n_i |n_i| \tag{2.9}$$

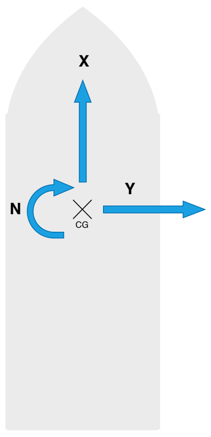


Figure 2.3: Definition of the thrust vector $\boldsymbol{\tau}$.

Extended Thrust Formulation using the Pseudoinverse

There are several formulations which takes different approaches the thrust allocation problem. A popular version is called the extended thrust formulation, where the $\boldsymbol{T}$-vector is extended to contain the x- and y-components of each thruster. The B-matrix is extended with one additional column per thruster such that the x-component in the extended thrust vector is connected to a column in the B-matrix that corresponds to the thruster angle being 0 degrees, and the y-component is connected to a column that correspond to the thruster angle being 90 degrees. This decomposing can be visualized as shown in Figure 2.4. The extended thrust formulation for ReVolt thus becomes as shown in Equation (2.10).

$$\boldsymbol{\tau} = B_{ext}\boldsymbol{T}_{ext} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ -l_{1,y} & l_{1,x} & -l_{2,y} & l_{2,x} & -l_{3,y} & l_{3,x} \end{bmatrix} \begin{bmatrix} T_{1,x} \\ T_{1,y} \\ T_{2,x} \\ T_{2,y} \\ T_{3,x} \\ T_{3,y} \end{bmatrix} \tag{2.10}$$

If the target is to minimize the size of $\boldsymbol{T}_{ext}$ with respect to that $\boldsymbol{\tau} = B_{ext}\boldsymbol{T}_{ext}$, solving the unconstrained linear control allocation without setting constraints of the size of the forces, nor on the rate of change of the forces or angles, can be formulated as shown in Equation (2.11a). W is a weighting matrix containing how much weighting each thruster force component should have in the minimization problem, and large weights corresponds to reducing the resulting thrust more than with small weights. The analytic solution is as shown in Equation (2.11b), using the pseudoinverse of B (the Moore-Penrose inverse [38]), assuming that B has full rank and that the desired force vector is $\boldsymbol{\tau}_d$.

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{T}_{ext}^{\top} W \boldsymbol{T}_{ext} \\ \text{subject to} \quad & \boldsymbol{\tau}_d = B_{ext}\boldsymbol{T}_{ext} \end{aligned} \tag{2.11a}$$

$$\implies \boldsymbol{T}_{ext} = \left(W^{-1} B_{ext}^{\top} (B_{ext} W^{-1} B_{ext}^{\top})^{-1} \right) \boldsymbol{\tau}_d \tag{2.11b}$$

The resulting forces are then found by calculating the magnitude as in Equation (2.12), having direction as in Equation (2.13), where `atan2(y,x)` is the four-quadrant inverse tangent. A visualization of the result can be seen in Figure 2.4[(1)].

$$T_i = \sqrt{T_{i,x}^2 + T_{i,y}^2} \tag{2.12}$$

$$\alpha_i = \text{atan2}\left(T_{i,y}, T_{i,x}\right) \tag{2.13}$$

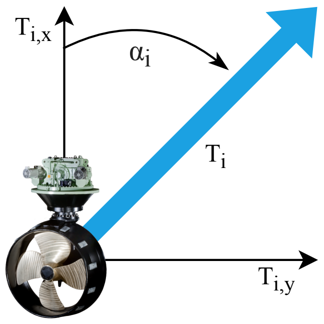


Figure 2.4: Thruster forces decomposed.

Using the extended thrust formulation as presented in in Equation (2.11a), no constraints regarding the rates of force production and angle of the azimuth thrusters is considered, but it is possible to iteratively search for solutions which reside within given constraints after a solution is found as shown in Equation (2.12) and Equation (2.13). More information about ways to accomplish this is given in Section 5.2.

Quadratic Programming Formulation

The extended thrust formulation, without any modifications, does not consider the previous thrust states nor thruster constraints from one time step to another. It is also not possible to include constraints such as forbidden azimuth angle zones, which is further discussed by Sørdalen [6] and Johansen and Fossen [12]. In order to handle this, a full quadratic programming approach could be taken to effectively formulate the constraints.

The constraints that could be enforced includes physical rate limitations on the force generation from each thruster between each time step, and rate limitations on how fast the azimuths are able to rotate. Due to the fact that many marine vessels are *overactuated* (having more control outputs than the DOFs that are being controlled), there might exist several solutions to an allocation problem that satisfies $\boldsymbol{\tau} = \boldsymbol{\tau}_d$. The constraints and extra goals of the optimization procedure are used to separate solutions from each other, typically by including a goal of minimizing power consumption in addition to reducing wear and tear on both the propeller shaft and the azimuths' rotational machinery.

The quadratic programming introduces a *slack variable* which represents the difference between the produced and the desired forces and moments such that $\boldsymbol{\tau} - \boldsymbol{\tau}_d = \boldsymbol{s}$. Therefore, minimizing the magnitude of $\boldsymbol{s}$ means minimizing the difference between the two force/moment vectors. Minimizing power consumption could be formulated as minimizing the total thruster forces, which means that $|\boldsymbol{T}|$ should be minimized. The variables to be minimized could have different priorities, assigning them different weights in the

(1) Thruster image from https://www.pngflow.com/en/free-transparent-png-cewun

objective function. This could be done in order to e.g. prioritize to minimize power consumption by allowing a larger deviation between $\boldsymbol{\tau}$ and $\boldsymbol{\tau}_d$ by assigning small weights to the slack variables, and higher weights to the thruster forces. Altogether, this could be formulated as minimizing a quadratic objective function such as $\boldsymbol{x}^T Q \boldsymbol{x}$, where Q is the weighting matrix, and $\boldsymbol{x}$ contains all *decision variables*, namely the variables that needs to be assigned values through the optimization.

The constraints on how large of a step the thrusters are able to make in terms of force per time step, and how much the azimuth thrusters can rotate per time step (according to their physical limitations), are formulated as inequality constraints, and the entire quadratic program could be formulated as in Equation (2.14), where the thruster forces and the slack variables are the parameters to minimize for each time step t, weighted differently by the weights q_i.

$$\begin{aligned} \min_{T,s} \quad & \sum_{i=1}^{3} q_{T_i} (T_i^t)^2 + \sum_{j=1}^{3} q_{s_j} (s_j^t)^2 \\ \text{subject to} \quad & B(\boldsymbol{\alpha}^t)\boldsymbol{T}^t - \boldsymbol{\tau}_d = s, \\ & \boldsymbol{T}_{min} \leq \boldsymbol{T}^t \leq \boldsymbol{T}_{max}, \\ & \delta_{T,min} \leq \boldsymbol{T}^t - \boldsymbol{T}^{t-1} \leq \delta_{T,max}, \\ & \delta_{\alpha,min} \leq \boldsymbol{\alpha}^t - \boldsymbol{\alpha}^{t-1} \leq \delta_{\alpha,max} \end{aligned} \tag{2.14}$$

Note that this formulation contains *non-linear* equality constraints due to the fact that the azimuth angles are occurring in trigonometric functions inside the B-matrix, and hence this becomes a *quadratic optimization problem with non-linear constraints.* The optimization problem could also be *nonconvex*, such that regular QP-solvers would not be guaranteed to find the global optimum. Methods for handling this has been proposed as mentioned by Johansen and Fossen [12], where the different azimuth angles are divided into separate regions, hence solving several convex QPs in parallel. This means that there needs to be done an extra evaluation afterwards to find the best solution coming from the different QPs based on an additional, predefined performance measure.

Chapter 3

Deep Reinforcement Learning

In this chapter, an explanation of Machine Learning generally, and Reinforcement Learning specifically, is given. The use of function approximators such as Deep Neural Networks to approximate value-functions and policies is discussed, leading up to Deep Reinforcement Learning. State-of-the-art algorithms used in the context of action selection processes like dynamic control will be given, where the algorithm of choice for this thesis is explained in detail. The chapter bases its derivations and notation on the "Deep Learning" book by Goodfellow, Bengio and Courville [39], the second edition of "Reinforcement Learning, An Introduction" by Sutton and Barto [40], OpenAI's "Spinning Up" website [41], and course material from "CS294: Deep Reinforcement Learning" at UC Berkeley [42].

3.1 Machine Learning - An Overview

Within the field of AI, Machine Learning (ML) is a sub-field which has gotten much attention over the last ten years, especially due to developments within Deep Learning (DL). In short, ML deals with developing algorithms that lets a computer learn from data without explicitly being programmed what to learn, but rather *how to learn*. Due to the improvement of different computer processing units in the 21st century, the computationally demanding processes involved in training Neural Networks have become much faster. The field of ML is typically divided into three sub-fields, namely Supervised Learning, Unsupervised Learning, and Reinforcement Learning. In the following, each of these sub-fields will be explained briefly. A more elaborate explanation of RL follows, in addition to an explanation of tools which modern *Deep* Reinforcement Learning builds on, and which has given it much attention the later years: Neural Networks. For a more in-depth explanation of the different applications of the various sub-fields, the reader is referred to chapter 5 of the Deep Learning textbook [39] which goes through the basics of ML.

3.1.1 Supervised Learning

Supervised Learning is used in situation where a dataset of input-output pairs are known, but how the pairs map to each other is not known. Therefore, supervised learning is used in classification tasks, where different classes is to be classified, e.g. through identification of objects in images. It is also used regression tasks, where one tries to predict the value of some output based the previous input-output data, such as predicting good control outputs from a vehicle's actuators based on the state of the vehicle and/or previous control outputs.

3.1.2 Unsupervised Learning

Unsupervised Learning differs from supervised learning through the fact that it is trained on a dataset that does not have *labels*. Hence, the learning algorithm is not explicitly told which predictions are right or wrong. This yields unsupervised learning helpful in applications where useful structural properties of the dataset can be discovered, which could be hard for humans to find. Due to the fact that these algorithms finds patterns which is not encoded by the humans applying them, a cost/reward function is not easy to implement. Therefore, the field of control has not been exploring Unsupervised Learning.

3.1.3 Reinforcement Learning

In Reinforcement Learning (RL), a trial-and-error based approach to the learning procedure is used. The basic setup of such a learning method involves an agent which performs actions within an environment while receiving rewards for each action in order to learn a *policy*. Unlike Supervised Learning, there are no existing ground truth dataset. Rather, the goal is to find a policy which selects actions that maximizes some expected reward when being in a certain state of the environment.

RL has become one of the most prominent machine learning areas due to its results in complex games consisting of both single and multiple agents(1). Previously, RL methods has struggled with systems with large state- and action-spaces since implementing them into computer programs could be limited by computational memory and time complexity. Therefore, instead of explicitly storing information about each combination of a state and an action which the system can consist of, *function approximators* has been applied to circumvent the problem of large state- and action spaces, a problem named "the curse of dimensionality". A much used approximator is the use of various architectures of *Neural Networks*. In fact, Neural Networks has become essential in enabling RL algorithms to be applied to real systems. Therefore, before going into the details of RL from Section 3.3 through Section 3.10, an introduction to Neural Networks follows as a mean of providing the foundation on which Deep Reinforcement Learning builds on.

3.2 Neural Networks and Deep Learning

An Artificial Neural Network (ANN) is a type of function approximator for functions that are hard to encode by hand, artificially mimicking the behavior of neurons in the human brain. This means that by itself, it has no relation with a particular sub-field of ML, but is used across various fields due to its expressive power. From a mathematical point of view, the ANN performs a mapping of one set of inputs in the input space in to another set of outputs in the output space. The nature of ML methods is to learn and represent connections that are hard to solve by humans either by analytical solutions, or by explicitly creating computer programs. Therefore, ANNs have been used extensively within ML as a means of approximation of various entities. In the eyes of control theory, the mapping could be done from a certain state of a system into control signals. In the following, a brief explanation of how a Neural Network works will be given.

(1)https://openai.com/five/

3.2.1 The Neuron

An artificial neuron in the ANN is a unit that works like a function which takes several arguments, and yields a single output. The arguments consists of differently weighted inputs and a bias term, which are summed up, passed through an activation function, with the result being the neuron's output as shown in Figure 3.1.

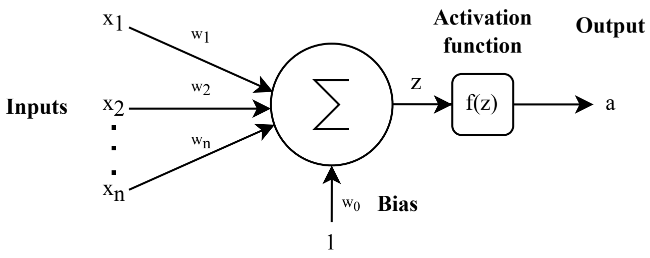


Figure 3.1: An artificial neuron's weights, bias and activation function.

The inputs might contain real numbers representing different features in the input data, and the weights are what expresses the relative importance of the different inputs. The bias term is not considered a part of the input set, but is a constant multiplied with a weight in order to shift the output with a constant. The activation function is what makes an ANN able to represent non-linear features, since if the neuron would only sum the inputs together, it becomes a *linear classifier*, only able to represent linear combinations of the input parameters. The formulation of the output can in general be written for a neuron with n inputs as

$$a = f(z) = f(\sum_{i=1}^{n} x_i w_i + w_0) \tag{3.1}$$

Popular activation functions include the hyperbolic tangent function (”tanh”, Equation (3.2)), the Sigmoid-function (Equation (3.3)), and the Rectified Linear Unit (”ReLU”, Equation (3.4)).

$$f_{tanh}(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{3.2}$$

$$f_{Sigmoid}(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

$$f_{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \tag{3.4}$$

3.2.2 Feedforward Neural Networks

There exists different ways to connect neurons together to form an ANN, with one of most regular ways resulting in a Feedforward Neural Network (FFNN). The neurons in an FFNN have weights going strictly from the input layer towards the output layer, using no feedback of information. It is characterized by having an input layer, one or more hidden layers, and one output layer. A hidden layer is the layer of neurons with activation functions. The neurons within a hidden layer are not connected to each other in an FFNN, so the only connections to the layer comes from the previous layer, and goes to the next layer. A simple FFNN is shown in Figure 3.2, where one hidden layer is drawn.

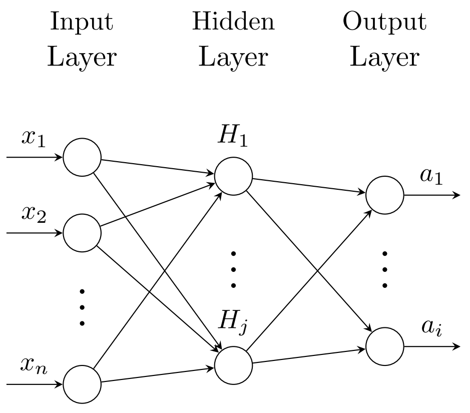


Figure 3.2: A Feedforward Neural Network with arrows representing weights between neurons.

The number of input neurons is decided by the number of features in the input data, and the output layer's size is determined by the task at hand. The number of neurons in the hidden layers, as well as the number of hidden layers, is subject to tuning, meaning that it comes down to trial and error (Heaton [43]). Generally, if there are more than one hidden layer, the learning process is called *Deep Learning.* As stated in chapter 6 of the Deep Learning textbook [39], there is no clear consensus on why deeper networks work better than shallow ones, but trends are showing that deeper networks seems to generalize better than shallow ones.

Going from the notation in Equation (3.1), it is possible to make a formulation for all parameters of an ANN. Let L denote the number of layers in the network so that l represents a specific layer number. Let k denote a neuron in layer l consisting of in total K_l neurons. The output a_k^l of neuron k in layer l depends on its input from the previous layer. The weighted connection between neuron $k^{'}$ in layer $l-1$ and neuron k in layer l can be written as $w_{k^{'},k}^{l}$. Thus the output a_k^l of a neuron k in layer l depends on the sum of all K_{l-1} weights and outputs from the previous layer, and can be written in component form as in Equation (3.5),

$$a_k^l = f(\sum_{k^{'}}^{K_{l-1}} w_{k^{'},k}^{l}\, a_{k^{'}}^{l-1} + w_{0,k}^{l}). \tag{3.5}$$

It is convenient to handle large networks by using vectors and matrices, so the entire output vector from layer l can be written in vector form as in Equation (3.6), where W^l is the weight matrix connecting weights in layer l to the previous layer's outputs and $\boldsymbol{w}_0^l$ is the bias vector for each of the neurons in layer l.

$$\boldsymbol{a}^l = f(W^l \boldsymbol{a}^{l-1} + \boldsymbol{w}_0^l). \tag{3.6}$$

Training a Neural Network

The process of which most ANN structures learn is by using *backpropagation.* During training, the output from each node is compared to a performance metric which is used to calculate "how much the output from a node contributes to the total performance", going from the output layer, through the nodes in the hidden layers, and to the input layer.

The goal of backpropagation in combination with an optimization function is to optimize an objective function J by adjusting the weights and biases of the network. An objective function is often referred to as a *loss* function when the goal is to minimize it. The objective function can be formulated in various ways, and for regression, Mean Squared Error is commonly used in order to formulate the error between outputs from the network and the desired ones. For measuring the performance of a network classifying different classes, outputting probabilities of each class between 0 and 1, the Log Loss can be used. Using an example of minimization, the total loss function J could be formulated as the average loss from each training sample m out of M training samples, such that

$$J = \frac{1}{M}\sum_{m} J_m. \tag{3.7}$$

Using Calculus, the rate of change of the loss function with respect to each layer's weights is found through the partial derivatives $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{W_0}$. In order to compute these derivatives, the loss depending on neuron k in the output layer (layer L) can be written as in Equation (3.8), stating how the error of the loss function changes with the input to neuron k the output layer, z_k^L.

$$\delta_k^L = \frac{\partial J}{\partial z_k^L} = \frac{\partial J}{\partial a_k^L} \cdot \frac{\partial a_k^L(z_k^L)}{\partial z_k^L} = \frac{\partial J}{\partial a_k^L} \cdot f'(z_k^L). \tag{3.8}$$

In vector form, this is formulated as shown in Equation (3.9),

$$\delta^L = \Delta_a J \odot f'(z^L), \tag{3.9}$$

where $\Delta_a J$ denotes the rate of change of the loss with respect of the activation function, $\odot$ represents the elementwise multiplication operator, and $f'(z^L)$ denotes the rate of change of the outputs of the output layer with respect to its inputs. Note that the shape of $\Delta_a J$ depends on the type of loss function, while $f'(z^L)$ depends on the type of activation function. Now that the output layer's error rate has been defined, the error rate of the other layers is calculated through *propagating* the output error rate backwards in the network, giving

$$\delta^l = W^{l+1}\delta^{l+1} \odot f'(z^l). \tag{3.10}$$

The effect on the error rate from the weight connecting neuron $k^{'}$ in layer $l-1$ and neuron k in layer l is formulated as in Equation (3.11), and the effect on the error rate from the bias of neuron k in layer l is found from Equation (3.12).

$$\frac{\partial J}{\partial w_{k',k}^l} = a_{k'}^{l-1}\delta_k^l \tag{3.11}$$

$$\frac{\partial J}{\partial w_{0,k}^l} = \delta_k^l \tag{3.12}$$

This defines the backpropagation, and the final step involves updating each of the weights based on the gradients found previously. For such updates, variants of gradient descent is commonly used. The reader is referred to Kiefer and Wolfowitz [44] and Du et al. [45] for gradient descent fundamentals. In all simplicity, the gradient descent update rule is defined as updating the weights according to the gradient which seems to minimize the output layer's loss, using a certain step size α. If the goal is to minimize the loss function J, the update rule for the weights of neuron k can be formulated as in Equation (3.13).

$$w_k \leftarrow w_k - \alpha \frac{\partial J}{\partial w_k} \tag{3.13}$$

Note that J in some cases refer to an objective function that is to be maximized instead of minimize, in which the sign in Equation (3.13) changes; this is the difference between gradient *ascent* and *decent*. Since the update rule has to be done for all weights and biases, for all data points in the data set when training, the learning process might become slow for increasing amounts of data. There exists several methods for increasing training speed by calculating the cost function gradient of each input for a small sample of randomly chosen training inputs, generalizing the idea of Stochastic Gradient Descent (SGD) [46], improved by Mini-Batch Gradient Descent [47], and optimized with clever selection of an adaptive step size $\eta(t)$ during training [48, 49, 50]. After training, the weights are set to be constant, i.e. no more backpropagation is carried out, and all inputs are only passed strictly forward - called a *forward pass*.

Deep Reinforcement Learning (DRL) describes the field that combines RL with Deep Neural Networks. In the following, the basics of RL will be described generally, leading up to the DRL algorithm of choice specifically, while explaining how the Neural Networks are being used to represent various entities.

3.3 Markov Decision Processes

In RL, it it usually assumed that the environment, and the behavior of making any actions within it, is described as a Markov Decision Process (MDP) [40]. The purpose of an MDP is to provide a general framework for finding optimal decisions for a decision maker in an environment which dictates the outcome of making actions. An MDP consists of the following:

- $s \in \mathcal{S}$: a set of states defining the state space of the MDP.
- $a \in \mathcal{A}$: a set of actions defining the action space of the MDP.
- $P \in \mathcal{P}(s'|s, a)$: a probability matrix relating the selection of an action a in a certain state s and ending up in a new state s'. This probability matrix can be viewed as the environment's dynamics.
- $r \in \mathcal{R}(s, a)$: a *scalar* reward function that describes how good a certain action was while being in state s.

Thus, an MDP can be, as a bare minimum, be defined in a 4-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$. Note that this tuple might vary in the literature: some include a discount factor γ in the tuple, while others might include an initial state distribution ρ_0. Regardless, an important property of an MDP is that the result of an action taken in one state is independent of previous states; transitions between states only depends on the most recent state-action pair. This property implies that all information about the sequence leading up to a certain state can be inferred from that state alone, and as stated in chapter 3.3 of Ibe [51]: "The Markov property implies that for all t, the process $\{X(t+s) - X(t), s \geq 0\}$ has the same distribution as the process $\{X(s), s \geq 0\}$ and is independent of $\{X(s), 0 \leq s \leq t\}$", where X describes a stochastic variable, being the state of the MDP in the context of this thesis.

The ultimate goal of methods for solving MDPs is to find the optimal action selection procedure. The action selection procedure is formulated through a *policy.* The following describes what an "RL agent" consists of, and how the rewards from the MDP can be used for updating the way the agent makes actions.

3.4 The Policy

The "RL agent" describes an entity which is able to interact with an environment like an MDP, through making observations and by selecting actions. While the policy describes how actions are decided upon, the actual performer of the actions would be elements like the actuators of a dynamic system. The two main ways policies are formulated is either being *deterministic*, being a fixed mapping directly from a state to an action, or it could be *stochastic*, following some probabilistic sampling of a distribution over the actions which are possible to make, given a certain state. A deterministic policy can be formulated as in Equation (3.14a), while stochastic policies are usually represented as probabilistic distributions as in Equation (3.14b).

$$a_t = \mu(s_t) \tag{3.14a}$$
$$a_t \sim \pi(s_t) \tag{3.14b}$$

Both ways of selecting actions are applicable to both types of *action spaces.* The action space describes the manner in which actions exist for the given system, and are divided into *discrete* and *continuous* spaces. Discrete action spaces are spaces where the number of all possible actions to take are finite. On the other hand, the continuous action spaces are not possible to enumerate. This includes many robotic systems, where actions can be a certain percentage of the maximum torque being applied to a joint. In reality, due to the finite accuracy of floating point numbers in computers, these continuous actions spaces are discrete, but they can become exponentially many, and thus unsuitable for enumeration. Due to parameterization, ANNs has given rise to Deep Reinforcement Learning (DRL) the last years, as Neural Networks have shown an ability to approximate complex functions, which can be used as the policy function for the RL agent. An example of a stochastic policy being represented as the probability of an action given a certain state *and* the parameters could be given as in Equation (3.15).

$$\pi_\theta = \pi(a|s, \boldsymbol{\theta}) = P(a|s, \boldsymbol{\theta}) \tag{3.15}$$

Various policy formulations combats the challenge of *exploration vs. exploitation* in different ways, explaining the challenge of knowing how to weight the exploration of the state space in the search of good policies, and when to utilize what the agent has learned this far in order to intensify the search for good policies in restricted areas of the state space. Too much exploration, and the learning becomes very slow and possibly never converges to a good policy. Too much exploitation, and the agent might converge to a local minimum due to lack of exploration. For stochastic policies in continuous action spaces, this problem is handled by probabilistic sampling. A typical formulation of such a policy is done by using Gaussian policies, consisting of a mean, μ and a standard deviation σ. For agents with more than one action output, multivariate Gaussian distributions can be used, in which the distribution is represented as a vector of means $\boldsymbol{\mu}$ and a covariance matrix Σ. The covariance matrix is normally diagonalized for simplicity in order to avoid policy outputs being dependent on each other. This gives a *diagonal, multivariate Gaussian distribution*. In this way it has become common for DRL applications that the policy is created by using a Neural Network as the parameterization of the means of this distribution, namely that the network outputs μ_θ, combined with the standard deviations σ. The policy π_θ outputs actions depending on a sampling procedure, and the sampling strategy using diagonal, multivariate Gaussian distribution is as shown in Equation (3.16) for a multidimensional action vector. $\odot$ represents the elementwise multiplication operator, and $\boldsymbol{z}$ is a noise vector sampled from a normal distribution with zero mean and unity variance ($z \sim \mathcal{N}(0, I)$).

$$\boldsymbol{a}_t \sim \pi_\theta(\boldsymbol{s_t}) = \boldsymbol{\mu}_\theta(\boldsymbol{s}_t) + \boldsymbol{\sigma}(\boldsymbol{s}_t) \odot \boldsymbol{z} \tag{3.16}$$

From here on out, all references to the policy will be about the stochastic policy, hence using π for the action selection procedure. This is due to that the setup of ReVolt's thrusters are in such a fashion that is advantageous to formulate the action space as continuous. In addition, the bold notation used for vectors will be used only when it is necessary to separate a vector from a scalar or a matrix.

3.5 Episodes and Rewards

Following the vocabulary from MDPs, it could be said that during learning, the agent observes states from the environment, and makes actions based on its policy. The outcome of the action is a new state, and a reward. This interaction can be visualized as in Figure 3.3.

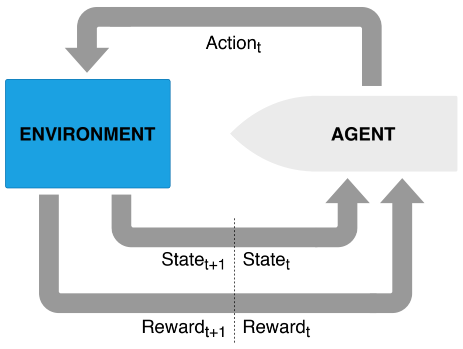


Figure 3.3: Agent-environment interaction in Reinforcement Learning.

An *episode* denotes one batch of state-action pairs which the agent generated while observing and acting in the environment. The length of an episode can be determined by several factors: either the episode is set to be of a particular maximum length, or the agent might encounter states which stops the episodes. Such states are called *terminal* states which are states where the agent cannot continue, either because the agent arrived in a certain goal state, or because the agent arrived in a state which is considered as "out of bounds" or a dangerous state. Episodes can, especially for dynamic systems, be viewed as *trajectories* of states and actions which describes how the agent got from the initial state, to any state in the trajectory by taking the actions leading up to that state. Such a trajectory, gathered under policy π_θ are usually denoted as $\tau \sim \pi_\theta$.

During these episodes, the agent's job is to somehow aggregate as much reward as it can over time in some way or another. Note that in this thesis, *reward* is used on a particular scalar value which the agent can receive at a given time, while *return* denotes a given aggregation of several rewards over time. A commonly used formulation for the return is the *infinite-horizon undiscounted return*, which represents the return G_t as the return of following a policy from time t until the end of the episode as shown in Equation (3.17). The reward r is an output of the reward function $\mathcal{R}$ introduced in Section 3.3.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+(k+1)}. \tag{3.17}$$

$0 \leq \gamma \leq 1$ is the discount rate, a weighting factor between rewards accumulated immediately, and reward accumulated in future time steps. If the discount factor is zero, the agent only maximizes the immediate reward and never considers future rewards. As the discount factor increases, more of the future reward will be considered. The practical consequence of the discount factor is discussed in Chapter 5. Equation (3.17) offers some attractive mathematical properties due to the lower and upper limit on the discount factor, since it converges to a finite value [40]. In the case of the agent finding the *optimal* reward r^*, and stays there from time t and through to infinity, the sum converges to the geometric series (and in this case the maximum possible discounted return),

$$G_t^* = r^* + \gamma r^* + \gamma^2 r^* + ... = \sum_{k=0}^{\infty} \gamma^k r^* = \frac{r^*}{1-\gamma}. \tag{3.18}$$

Since the RL agent does not have a perfect model of the world, it will try to maximize the return for whatever experience it aggregates. Therefore, its job is to maximize the *expected* return over the trajectories τ which are experienced by following the policy π_θ,

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [G(\tau)], \tag{3.19}$$

and by doing so, it tries to find the optimal policy, π^*. In the following, common ways of learning how to do this with RL will be shown.

3.6 Solving MDPs with Reinforcement Learning

An RL agent can interact with the MDP, gathering experiences of certain actions taken in certain states, leading to certain reward as shown in Algorithm 1. This experience is referred to as a trajectory, τ. In order to learn, different RL methods aims to use the trajectories in order to teach the agent how to behave, as shown in Algorithm 2. If the learning algorithm only uses trajectories which were sampled with the current policy for learning, the process is called "on-policy" RL. If the actions are taken without having information about the environment, it represents *model-free* RL.

Algorithm 1 On-policy RL trajectory generation

```
generate_trajectory = function(policy, M, initial_observation):
  for all M steps per episode do
      Select and action based on the current policy, given the current observation.
      Perform the action while observing the next state and the reward received.
      Store the observation, action and reward in trajectory.
      if Next state was a terminal state then
          return Trajectory.
      else
          Update current observation to be the observation of next state.
  return Trajectory
```

Algorithm 2 On-policy RL for solving MDPs

```
RL = function(agent, N, learning algorithm):
  Initialize all parameters of the agent.
  for all N episodes do
      Randomly initialize state in the environment, and observe the state.
      Run an episode as shown in Algorithm 1.
      Perform update on the agent's policy according to the learning algorithm.
return The agent's policy.
```

3.6.1 Value Functions

In almost all RL algorithms, a notion of the value of being in a certain state is used. The entity which represents the value of being in all the different states is called a *value function*. There are two main types of value functions used: one representing the value of being in a specific state (the state-value function), and the other representing the value of a certain combination of being in a certain state while taking a certain action (the action-value function). The exact value of these types of value functions are usually estimated due to imperfect information about the environment, and the values are therefore an expectation, represented with $\mathbb{E}[\cdot]$ in the equations to come. For the state-value function, the value of being in state s is denoted is

$$V^{\pi}(s) := \underset{\tau\sim\pi}{\mathbb{E}}[G(\tau)|s_0 = s], \tag{3.20}$$

where the notation means that $V^{\pi}(s)$ represents the *expected* return G of the trajectory τ generated by taking actions according to policy π, starting in state s. The *expected* return is often found through letting the agent interact with the environment while generating trajectories. For the action-value function, the formulation is very similar to the state-value function, except for the inclusion of the value of taking a certain action in state s as well, as shown in Equation (3.21). Note that it evaluates only action a in the *current* state, but the following actions are made by the followed policy.

$$Q^{\pi}(s, a) = \underset{\tau\sim\pi}{\mathbb{E}}[G(\tau)|s_0 = s, a_0 = a]. \tag{3.21}$$

It can be helpful to notice that the return in Equation (3.17) can be formulated as

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \\ &= r_{t+1} + \gamma\Big(r_{(t+1)+1} + \gamma r_{(t+1)+2} + ...\Big) \\ &= r_{t+1} + \gamma G_{t+1}, \end{aligned} \tag{3.22}$$

meaning that the return from time t to infinity is the next reward plus the discounted future rewards of the next time step's return. As the state-value function and the action-value function are both defined on the basis of the return, the same type of recursive property can be formulated, and forms the basis of the Bellman Equations presented in the next section. From here on out, the term value function will refer to the state-value function $V(s)$, and the action-value function $Q(s, a)$ will be referred to as the Q-function.

3.6.2 Bellman Equations and Dynamic Programming.

Richard E. Bellman (1920-1984) introduced the ideas of Dynamic Programming and the Bellman Equations in the 1950's. In the cases where the MDP is able to be fully formulated, *including* the transitional probabilities $\mathcal{P}$, dynamic programming serves as a collection of iterative methods for finding solutions to the optimal policy and/or value function in cases where the analytic solutions are complex to find. The field of Dynamic Programming is vast, and will only be explained in brevity here; chapter 3 and 4 of [40] gives a rigorous walk-through.

Bellman formulated equations which was proved to be necessary conditions for solutions of Dynamic Programs to be optimal[(2)]. The equations follows what was showed in Equation (3.22), and gives that the value of being in state s_t and following policy π, is the sum of the immediate reward and the discounted sum over the values of the next states, weighted by the probability of getting to state s_{t+1} when taking actions according to the policy π:

$$V^{\pi}(s_t) = r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, \pi(s_t)) V^{\pi}(s_{t+1}). \tag{3.23}$$

Note that since this includes the transitional probabilities, it represents an expectation, and by letting $s_{t+1} \sim P$ denote "going to state s_{t+1} from state s_t by taking action a_t, following the transitional probability $P(s_{t+1}|s_t, a_t)$", we get

$$V^{\pi}(s_t) = \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[r(s_t, \pi(s_t)) + \gamma \mathop{\mathbb{E}}_{s_{t+1}\sim P}[V^{\pi}(s_{t+1})]\Big]. \tag{3.24}$$

Bellman also formulated an *optimality* equation, which defined the optimal policy as the policy π^* that maximizes Equation (3.24),

$$\begin{aligned} V^*(s_t) &= \max_{a'}\Big\{r(s_t, a') + \gamma \sum_{s_{t+1}} P(_{t+1}|s_t, a') V^*(s_{t+1})\Big\} \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi^*}\Big[r(s_t, \pi^*(s_t)) + \gamma \mathop{\mathbb{E}}_{s_{t+1}\sim P}[V^*(s_{t+1})]\Big]. \end{aligned} \tag{3.25}$$

(2) These equations are formulated depending on the setup of how the agent receives rewards in the MDP, one being that rewards are given for transitions between states, and other being giving rewards given the presence in a state. This thesis uses the latter formulation.

It is worth noting that the same equations are valid for the Q-function, where $a' \sim \pi$ denotes taking next actions according to the policy *pi*, giving the Bellman equation in Equation (3.26).

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1} \sim P}\Big[r(s_t, a_t) + \gamma \mathop{\mathbb{E}}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]\Big] \tag{3.26}$$

The strategies of solving MDPs with Dynamic Programming are, as stated in [40], page 74, "*... obtained by turning Bellman equations such as these into assignments, that is, into update rules for improving approximations of the desired value functions*". In RL, the iterative fashion of Dynamic Programming is still pertained, but there are uncertainties in the MDP such as the transitional probabilities, which is typical for model-free RL. However, the Bellman Equations still stands as crucial tools.

3.6.3 Learning Through Estimation

Dynamic Programming and the Bellman equations serves as a tool for solving MDP's when the transitional probabilities are known. But when these probabilities are unknown, other tools must be applied, such as RL which learns through estimation.

Monte Carlo Methods

Monte-Carlo methods (MC) is a way of estimating value functions for usage in the Bellman equations letting the agent interact with the environment. Then, the return of being in the current state s_t is calculated the average of the returns the agent observed after moving on from the state. As more experience is gathered, the closer the average return gets to the true value. By keeping track of how many times s_t has been visited in a function $N(s_t)$ by sampling N trajectories, an estimate of the value in the state can be formulated as the iterative MC update in Equation (3.27), where G_t is the last available return from all trajectories $i = 1$ to N. The MC methods are shown to converge to the expected value, and are therefore said to have zero bias. The convergence can be slow due to that MC methods depends on sampling entire trajectories before doing any updates to the value function estimates; the episodes has to terminate before updating. They tend to have large variance since entire trajectories has to be sampled before the update rule can be applied, potentially making the updates noisy.

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t)) \tag{3.27}$$

Temporal Difference Methods

Another way of estimating the value function is to use the estimates of the value function of other states in order to update the value of the current state. This is what Temporal Difference (TD) methods are based on, and thus does not need to have entire trajectories in order to perform an update to the value estimates. The TD methods are closely related to the Bellman equation and uses an estimate for how well the value function is being approximated called the "temporal difference error" or "TD-error", δ_t. If we are looking only towards the next state we get the one-step TD-error as in Equation (3.28a). If the estimated value of the current state is too low compared to the immediate reward and the expectation of the future returns, the TD-error will be positive, and vice versa. This can be used to update the value estimates, using a step size α, as shown in Equation (3.28b).

$$\delta_t := r_t + \gamma V(s_{t+1}) - V(s_t), \tag{3.28a}$$
$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] = V(s_t) + \alpha\delta_t \tag{3.28b}$$

TD methods provides less variance in its estimation of the expected values since it can update much more frequently, and is said to be a bootstrapping method since it uses estimate values during the update of the same type of estimated values. However, it is sensitive to the initial values that the value function was set to have, and comes with a bias towards these values. This is known to potentially cause instabilities during learning in methods which are mainly focused on learning the optimal value function for finding optimal policies in some cases, as explained in chapter 11.3 in [40] about the deadly triad(3).

One idea of how to find the optimal policy is to learn the optimal values first, and then selecting the policy that always transitions between the states with the highest values. This idea is the basis of popular methods such as Q-learning, a method introduced in 1989 by Watkins [52], receiving large attention in 2013 with the demonstration of DQN [53] which combined Q-learning with Neural Networks to reach super-human level in Atari computer games. Another idea is to directly update the policy itself during learning, which forms the basis of methods within policy optimization which optimizes an objective function J through its gradient (named policy gradient methods) such as the REINFORCE algorithms [54]. In the following, the value function methods will be explained briefly, while policy gradient methods will be explained in detail. This is due to that the selected algorithm of implementation, PPO, relies on the theory of policy gradient methods, but it uses some of the principles of Q-learning which is beneficial to introduce beforehand.

3.7 Value Function Methods

The most known algorithm for learning the value function is the Q-learning algorithm. In fact, it learns the *Q-values* as presented in Section 3.6.1. The algorithm seeks to find the optimal policy by finding the Q-values of the MDP it is set to solve. It works by gathering experience through the before-mentioned interaction between the agent and the environment, and updates the Q-values according. δ_t is used to represent the TD-error as in Section 3.6.3, only using action-values instead state-values. The update rule uses the maximum of the future Q-values by iterating through the possible actions to take in s_t, $a' \in \mathrm{A}$, as shown in Equation (3.29) where α is a parameter deciding the size of the step to take in the direction of the TD-error. Note that a' represents the action at time $t+1$ which maximizes the Q-value of the next state s_{t+1}.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a' \in \mathrm{A}} Q(s_{t+1}, a') - Q(s_t, a_t)]. \tag{3.29}$$

The Q-values can be stored in tables if the states and actions can be discretized, but this gives problems when the state space and/or action space is too large, or is continuous.

(3)The deadly triad is Sutton and Barto's term on the potentially problematic combination of function approximation (e.g. Neural Networks), bootstrapping methods (e.g. TD methods) and off-policy training (training the value function and/or policy on data which was not produced under the current value function and/or policy).

Consequently, function approximation becomes relevant, and Neural Networks can be used to estimate the Q-values. The authors of [53] also introduced the idea of *experience replay*, in which previous experience can be reused update the Neural Networks, making Q-learning a sample efficient algorithm due to the use of the same experience several times. Training of the Neural Network is commonly done with the Mean Squared Error (MSE) objective function. Using a Neural Network with the parameters W to estimate the Q-values $Q_W(s,a)$, an objective function $J(W)$ as formulated as in [53] is done by using MSE, where the task of the optimization would be to minimize it. Note maximum operator calculates the maximum Q-values using the old network weights, W_{old}.

$$J(W) = \frac{1}{2}\Big(r(s_t,a_t) + \gamma \max_{a'} Q_{W_{old}}(s_{t+1},a') - Q_W(s_t,a_t)\Big)^2. \tag{3.30}$$

The optimization through gradient *decent* thus uses the gradient with respect to the Neural Network's weights as shown in Equation (3.30),

$$\nabla_W J(W) = -\Big(r(s_t,a_t) + \gamma \max_{a'} Q_{W_{old}}(s_{t+1},a') - Q_W(s_t,a_t)\Big)\nabla_w Q_W(s_t,a_t). \tag{3.31}$$

$\nabla_W Q_W(s_t,a_t)$ is a gradient related to the output of the Neural Network and depends on network architecture, as explained in Section 3.2. This gradient is then used in the update rule, which can be optimized with a gradient descent optimizer like SGD (see Section 3.2.2), an idea used in the PPO algorithm used in this thesis.

3.8 Policy Gradient Methods

In DRL, Neural Networks are used to parameterize π_θ. As the goal of the DRL agent is to find the policy that maximizes the expected return, a formula for the expected return as a function of the parameters θ lets us formulate an objective function $J(\pi_\theta)$ to *maximize*, as shown in Equation (3.19). This allows for a representation of the optimization procedure as a gradient step method, where to goal is to *maximize* the objective function (oppositely from gradient descent in Equation (3.30) where a loss function was to be minimized), and so we look for ways to carry out gradient *ascent*:

$$\theta_{new} \leftarrow \theta_{old} + \alpha \nabla_\theta J(\pi_{\theta_{old}}). \tag{3.32}$$

The gradient of $J(\pi_\theta)$ has to be estimated due to the uncertainties of the MDP, and methods focused on optimizing the parameters by estimating the policy gradient $\nabla_\theta J(\pi_\theta)$ are referred to as *policy gradient (PG) methods.*

3.8.1 Calculating the Gradient

The goal of PG methods is to find an expression for the gradient of the objective function in order to carry out gradient ascent, and in the following, an analytic approach to finding the gradient will be shown. By using the probability $P(\tau|\pi_\theta)$ of experiencing each trajectory, the expected return over all trajectories is the probability of a trajectory τ gathered with π_θ times the return of that trajectory, summed over all trajectories experienced by the agent in the environment:

$$J(\pi_\theta) = \int_\tau P(\tau|\pi_\theta) G(\pi_\theta), \tag{3.33}$$

The probability of experiencing τ is the probability $\rho_0(s_0)$ of starting in state s_0 times the transitional probability between states, $P(s_{t+1}|s_t, a_t)$, following π_θ:

$$P(\tau|\pi_\theta) = \rho_0\left(s_0\right) \prod_{t=0}^{T} P\left(s_{t+1}|s_t, a_t\right) \pi_\theta\left(a_t|s_t\right). \tag{3.34}$$

Using Equation (3.34) and taking the gradient with respect to the ANN's parameters, we obtain Equation (3.35). Since the integral represents the finite summation over all trajectories, the gradient of the sum is equal to the sum of the gradients.

$$\nabla_\theta J(\pi_\theta) = \int_\tau \nabla_\theta P(\tau|\pi_\theta) G(\tau). \tag{3.35}$$

$G(\tau)$ does not depend on the parameters θ and is treated as a constant in the differentiation. Therefore, we can look for the derivative of the probability distribution as a function of θ, $\nabla_\theta P(\tau|\pi_\theta)$. Here, a common identity in PG methods is used through the chain rule:

$$\nabla_\theta \log P(\tau|\pi_\theta) = \frac{1}{P(\tau|\pi_\theta)} \nabla_\theta P(\tau|\pi_\theta), \tag{3.36}$$

This leads to that $\nabla_\theta P(\tau|\pi_\theta)$ can be replaced with an expression including the natural logarithm log such that

$$\nabla_\theta J(\pi_\theta) = \int_\tau P(\tau|\pi_\theta) \nabla_\theta \log P(\tau|\pi_\theta) G(\tau). \tag{3.37}$$

Going from Equation (3.34), we obtain the expression for $\nabla_\theta \log P(\tau|\pi_\theta)$ through first calculating the expression for the *log-probability* $\log P(\tau|\theta)$ as

$$\begin{aligned}
\log P(\tau|\theta) &= \log \Big(\rho_0\left(s_0\right) \prod_{t=0}^{T} P\left(s_{t+1}|s_t, a_t\right) \pi_\theta\left(a_t|s_t\right) \Big) \\
&= \log \rho_0\left(s_0\right) + \log \prod_{t=0}^{T} P\left(s_{t+1}|s_t, a_t\right) \pi_\theta\left(a_t|s_t\right) \\
&= \log \rho_0\left(s_0\right) + \sum_{t=0}^{T} \Big(\log P\left(s_{t+1}|s_t, a_t\right) + \log \pi_\theta\left(a_t|s_t\right) \Big)
\end{aligned} \tag{3.38}$$

When taking the derivatives with respect to θ of the log-probability, the derivatives of the initial state distribution *and* the transitional probabilities evaluates to zero as they are not functions of the parameters of the policy, which results in

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right). \tag{3.39}$$

This is a quantity which only relies on the policy, and which we know how to compute, a property which was not present previously. The gradient of the objective function can therefore be written as in Equation (3.40).

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \int_\tau P(\tau|\pi_\theta) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta \left(a_t|s_t\right) G(\tau) \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} [\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta \left(a_t|s_t\right) G(\tau)].
\end{aligned} \tag{3.40}
$$

The consequence of this result is the ability to calculate the gradient for a gradient ascent optimizer. An expression of the gradient through estimation can be the mean gradient of the collected rewards over all M trajectories gathered, $\{\tau_1, ..., \tau_M\}$, namely

$$
\nabla_\theta J(\pi_\theta) \approx \hat{g} = \frac{1}{N} \sum_{\tau_i}^{\tau_M} \Big(\sum_{t=0}^{T_{\tau_i}} \nabla_\theta \log \pi_\theta \left(a_t|s_t\right) G(\tau) \Big) \tag{3.41}
$$

One of the earliest examples of an on-policy PG method is the REINFORCE algorithm, introduced by Williams [54], which samples from the environment by the Monte Carlo method (gathering full trajectories before updating the parameters), and can be set up like in Algorithm 3 using the infinite-horizon return from Equation (3.17). Note that the gradient calculation is done at each time step in each trajectory, thus representing the expression *inside* the rightmost sum in Equation (3.41).

Algorithm 3 Episodal REINFORCE with discounted return

```
REINFORCE = function(policy $\pi_\theta$, N):
  Initialize parameters $\theta$ of the policy network and the step size $\alpha$.
  for all N episodes do
      Generate trajectory $\tau$ of length T+1 following policy $\pi_\theta$ in Algorithm 1.
      for all t in T+1 steps do
          Calculate the discounted return from current time $t$, $G_t = \sum_{i=t}^{T} \gamma^i r_i$
          Calculate the estimated gradient $\hat{g}_t = \nabla_\theta \log \pi_\theta(a_t|s_t, \theta) G_t$
          Perform the parameter update $\theta \leftarrow \theta + \alpha\hat{g}$
  return Obtained policy $\pi_\theta$.
```

For clarity, the expression for the log-probability is an expression depending on the type of policy used. As an example of a stochastic policy by a multivariate, diagonal Gaussian distribution, the log-probability calculates to be as in Equation (3.42). This represents the "log-likelihood" of the k-dimensional action $\boldsymbol{a}_t$ given the state $\boldsymbol{s}_t$, and is differentiated for finding the so-called "grad-log probability". In practice, it can be computed with software packages supporting automatic differentiation such as Tensorflow[(4)]. The expression for $\nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ will vary depending on how the means $\boldsymbol{\mu}$ and the standard deviations $\boldsymbol{\sigma}$ depend on the parameters $\boldsymbol{\theta}$.

$$
\log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) = -\frac{1}{2} \sum_{i=1}^{k} \Big(\frac{(a_i - \mu_{\theta,i})^2}{\sigma_i^2} + 2\log\sigma_i \Big) - \frac{k}{2} log(2\pi) \tag{3.42}
$$

[(4)]https://tensorflow.org/

3.8.2 Reducing Variance in the Gradient Estimation

PG methods are, as mentioned by Schulman et al. [55], troubled by two aspects. The first is sample inefficiency, since once a trajectory has been used in the update rule, it cannot be used any more as it was gathered under an old policy. Thus the methods requires a large amount of interactions with the environment to show improvements. The other aspect involves difficulties with ensuring that the changes done to the policy leads to better performance due to the shift in the distribution of the data that is being gathered since the gathered data distribution changes as the policy changes). Schulman et al. proposed a solution to both, but their solution to the first aspect is what has been picked up by several similar PG methods in order to reduce variance. To understand their proposal of the *Generalized Advantage Estimator* (GAE), a baseline function must be introduced.

Baselines

In combination with an update rule, it can be viewed like the gradient in Equation (3.40) is used to increase the probability of good actions, while decreasing the probability for actions with bad outcomes. This is done while weighting the log-probabilities according to the trajectory's return. There are however other choices of "weighting functions" that, in expectation, gives the same gradient as in Equation (3.40), but lowers variance during training. As shown by Williams in 1992 [54], it is possible to introduce a *baseline* function b for reducing the variance of the estimations of the gradient,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right)\big(G(\tau) - b\big)\Big]. \tag{3.43}$$

In fact, due to Williams' proof, any function Ψ_t not depending on the policy parameters θ can replace $(G(\tau) - b)$ as it results in the same value of the gradient *in expectation*:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right)\Psi_t\Big]. \tag{3.44}$$

In practice, a trade-off between variance and bias has to be made when choosing Ψ_t. Good choices are discussed in chapter 3 of Peters and Schaal [56] and in the introduction of Schulman et al. [55], often including a notion of one of the value functions.

Generalized Advantage Estimation

Since the gradient should help improving the policy by increasing probabilities of taking good actions while decreasing it for bad ones, it would be preferable for Ψ_t to also help in the weighting of the importance of the good and bad actions. An estimate of how good an action is in a certain state compared to the average value of all actions in that state (from following the current policy) would therefore be helpful. This is called an Advantage Function, which is formulated as

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t). \tag{3.45}$$

It turns out that choosing $\Psi_t = A^{\pi_\theta}(s_t, a_t)$ is close to choosing the function minimizing the variance, but it could be hard to estimate. Schulman et al. proposed using the Advantage function which uses the infinite-horizon discounting as introduced in Equation (3.17), such that the discounted policy gradient becomes

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right) A^{\pi_\theta}\Big]. \tag{3.46}$$

Their estimator for the discounted advantage function builds on that the TD-error, as introduced in Section 3.6.1, can be seen as an estimate of the $A^{\pi_\theta}(s_t, a_t)$:

$$\begin{aligned} A^{\pi_\theta}(s_t, a_t) &= \mathop{\mathbb{E}}_{s_{t+1}}\Big[Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\Big] \\ &= \mathop{\mathbb{E}}_{s_{t+1}}\Big[r(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\Big] \\ &= \mathop{\mathbb{E}}_{s_{t+1}}\Big[\delta_t^{V^{\pi_\theta}}\Big]. \end{aligned} \tag{3.47}$$

By writing out the sums of k numbers of the TD-errors (where $k = 1$ represents a one-step look-ahead such as seen in Equation (3.47)) and denoting each sum $\hat{A}^{(k)}$, it can be shown that the infinite-step estimator simplifies to the infinity-horizon discounted return minus the value function,

$$\hat{A}^{(\infty),\pi_\theta} = \sum_{i=0}^{\infty} \gamma^i r_{t+i} - V^{\pi_\theta}(s_t). \tag{3.48}$$

The exponentially-weighted average of k estimators, using a decay factor λ to decay the importance of multi-step look-ahead estimators, gives the final GAE estimator,

$$\hat{A}_t = (1-\lambda)\Big(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + ... + \lambda^{k-1}\hat{A}_t^{(k)}\Big) = \sum_{i=0}^{k}(\gamma\lambda)^i \delta_{t+i}^{V^{\pi_\theta}}. \tag{3.49}$$

This has the advantage of low variance, while offering a potentially low bias. The computation of the TD-error, δ, can be done online, and the computation of $\hat{A}_t$ is efficient. However, the TD-errors are using the value function as used by the value based methods in Section 3.6.1, but not yet in the PG methods; until now, only the policy parameters is what has been stored during training. To be able to estimate these value functions, and update the estimates during training in order to guide the policy, hybrid methods called *actor-critic* methods are used.

3.9 Actor-Critic Methods

The actor-critic methods are methods where estimates of the value function are used as support in the evaluation of what actions are better actions than others for the policy. The methods therefore combine a parameterization of the policy, and another one of the value function. The former is called the *actor*, as it is the one making the actions. The latter is called the *critic*, since its estimations are used to critic the actions that the actor is making. The input to both entities could be the state, but the output of the actor is the action, while the output of the critic is a scalar value of the value function estimate. The way the two networks are trained directly follows from the explanations of value function methods in Section 3.6.1 and PG methods in Section 3.8. In the following, θ is used as the actor's parameters, and W is used as the critic's parameters - they are the weights in the Neural Networks. We can save the TD-errors for both generating good estimates of the Advantage function for the training of the actor (π_θ), and to create targets for the regression task of improving the accuracy of the value function (V_W). Since both the actor and the critic use the TD-errors, the RL agent architecture can be be imagined to look like in Figure 3.4.

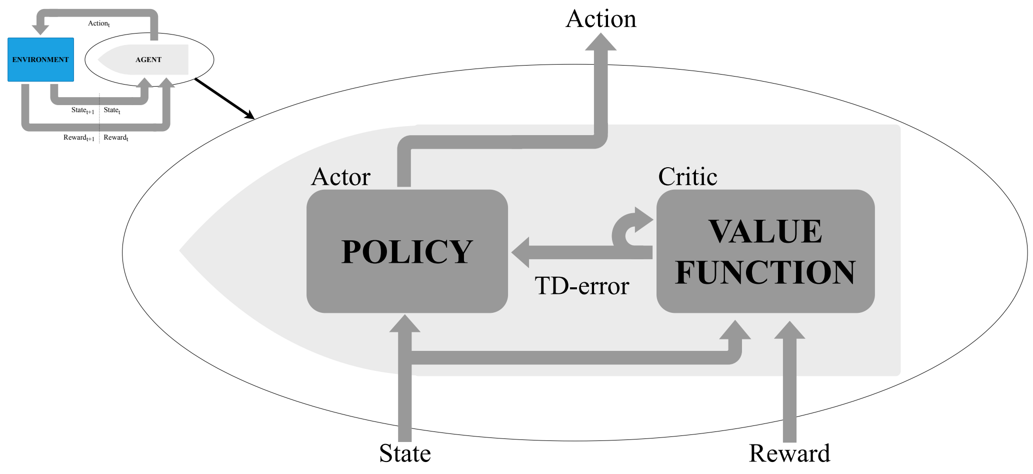


Figure 3.4: The actor-critic architecture.

3.10 Proximal Policy Optimization

This section introduces the DRL algorithm used in this thesis, named Proximal Policy Optimization (PPO), starting with the idea of trust regions for ensuring improvements to the PG methods' parameter updates, followed by explaining the objective function formulation which has made PPO successful in a large variety of DRL applications.

3.10.1 Trust Regions

An advantage of PG methods is that the optimization target is *directly* affecting what is searched for: the policy itself. However, the PG methods are not ensured of never doing an update which leads to instability, changing the policy to a bad one by making a too large changes during a parameter update. Newer PG methods aims at improving the stability convergence properties during training by avoiding large changes of the policy network for each update. Trust region methods for optimization has been used for a while [57, 58, 59], and a trust region in the context of Neural Network policies is, simply put, a region in policy parameter space (the space of the Neural Network weights) in which we are sure of a certain performance in the policy space (the resulting probability distribution).

As showed in Section 3.8.2 on GAE, the gradient used by PG methods so far can be formulated as in Equation (3.46) while using an estimator for the advantage function $\hat{A}_t$ at any state-action pair (s_t, a_t). This gives the objective function as proposed by Kakade and Langford [60],

$$J^{CPI}(\pi_\theta) = \mathbb{E}_t[\log \pi_\theta\,(a_t|s_t)\,\hat{A}_t]. \tag{3.50}$$

The superscript CPI denotes "conservative policy iteration", and $\mathbb{E}_t$ is used for the expectation at the current time step. To avoid performing optimization steps on this objective too far away from the old parameter space, it can be beneficial to consider the *probability ratio* between the policies of performing action a_t in state s_t, using the old and the updated network parameters. By using this ratio, it can be shown that formulating an objective function which has the same gradient as J^{CPI} enables the improvement of the original objective while actually optimizing an approximation of it, called a *surrogate* objective function:

$$J^{IS}(\pi_\theta) = \mathbb{E}_t\Big[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(s_t|a_t)}\hat{A}_t\Big], \tag{3.51}$$

where IS denotes "Importance Sampling", and as been shown to be very similar to the original PG-objective for reasonably small changes to the policy parameters [23, 60]. In fact, with the identity in Equation (3.36), we can see that the two objectives results in the same policy gradient. Letting θ be the parameters after a policy update, and θ_{old} be the parameters used when gathering trajectories of state-action pairs, we get the following by staying close to θ_{old}:

$$\nabla_\theta \log \pi_\theta(a_t|s_t)\Big|_{\theta_{old}} = \frac{\nabla_\theta \pi_\theta(a_t|s_t)\big|_{\theta_{old}}}{\pi_{\theta_{old}}(a_t|s_t)} = \nabla_\theta\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\Big)\Big|_{\theta_{old}} \tag{3.52}$$

$$\implies \nabla_\theta J^{CPI}(\pi_\theta) = \Big(\nabla_\theta J^{IS}(\pi_\theta)\Big)\Big|_{\theta_{old}}. \tag{3.53}$$

By using $J^{IS}(\pi_\theta)$, it becomes possible to formulate a notion used by trust region methods. Through determining how much we accept that the *probability distribution* changes from the current to the next policy, it becomes possible to limit the changes in the policy's probability space, and not only a bound on the policy's parameter space. The TRPO algorithm formulates the optimization of the objective function with respect to a limited change of the probability ratio by using a computational complex measurement for the ratio. PPO was introduced straight after TRPO by Schulman et al. [25], and aimed at making TRPO simpler while achieving the same performance. Therefore, PPO was the chosen algorithm in this thesis since, while being simpler in implementation and complexity than TRPO, it has shown great performance on a large variety of robotic tasks, having less hyperparameters to tune while also being less sensitive to the choice of hyperparameters [25, 61]. PPO exists with two variants: one with "adaptive KL-coefficient", and one with a "clipped surrogate objective". This section will only explain the latter since researchers from OpenAI experienced that the clipped surrogate objective outperformed the adaptive KL-coefficient [25].

Clipped Surrogate Objective

Schulman et al. restricted the size of each parameter update as in the following. Letting the probability ratio between the new and the old probability of taking action a_t given the state s_t be denoted as a function of θ,

$$\bar{r}(\pi_\theta)_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}. \tag{3.54}$$

Note that $\bar{r}(\pi_\theta) \geq 0 \; \forall \; \theta$ and $\bar{r}(\pi_{\theta_{old}})_t = 1.0$ since the probability in the numerator is the same as the one in the denominator of $\bar{r}$[5]. This way, the surrogate objective function can be rewritten as

$$J(\pi_\theta) = \hat{\mathbb{E}}_t\Big[\bar{r}(\pi_\theta)_t \hat{A}_t\Big], \tag{3.55}$$

which leads to the clipped PPO objective function,

$$J^{CLIP}(\pi_\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(\bar{r}_t(\pi_\theta)\hat{A}_t,\ \text{clip}\left(\bar{r}_t(\pi_\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\Big)\Big]. \tag{3.56}$$

The operator clip$(\bar{r}, (1-\epsilon), (1+\epsilon)$ represents a function which restricts the value of $\bar{r}$ to be within the lower bound $(1-\epsilon)$ and upper bound $(1+\epsilon)$, and ϵ is the clipping hyperparameter. To make it easier to understand this objective function for a single trajectory, it can be refactored as the sum of the expected objective function at each state-action pair for all the state-action pairs in the trajectory,

$$J^{CLIP}\left(\pi_\theta\right) = \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_{old}}}\Big[J^{CLIP}(\pi_\theta, s_t, a_t)\Big]. \tag{3.57}$$

Looking at each state-action pair, it can be shown that the objective function can be written as in Equation (3.58) [62], where $\hat{A}_t = \hat{A}_t^{\pi_{\theta_{old}}}$ refers to the estimated value of the Advantage function of the state-action pair (s_t, a_t) under the *old* policy used during the trajectory generation, $\pi_{\theta_{old}}$. For simplicity, the ratio $\bar{r}\left(\pi_\theta\right)_t$ is denoted as $\bar{r}_t$.

$$J^{CLIP}\left(\pi_\theta, s_t, a_t\right) = \begin{cases} (1+\epsilon)\hat{A}_t, & \hat{A}_t > 0 \text{ and } \bar{r}_t > (1+\epsilon) \\ (1-\epsilon)\hat{A}_t, & \hat{A}_t < 0 \text{ and } \bar{r}_t < (1-\epsilon) \\ \bar{r}_t\hat{A}_t, & \text{otherwise} \end{cases} \tag{3.58}$$

Note that $J^{CLIP}(\pi_\theta, s_t, a_t)$ is still a function of the new policy π_θ due to its dependency on $\bar{r}_t$, as it represents Equation (3.56), just for a single state-action pair. Equation (3.58) states that

1. If $\hat{A}_t$ is positive, we have estimated that making action a_t in state s_t is better than the average return from being in s_t. Therefore, we would like to *encourage* selecting a_t. However, the probability should not be increased too much, and so we bound the probability ratio to an upper bound $(1+\epsilon)$. Otherwise, if the ratio is not higher than the bound, then we use the conservative objective $\bar{r}_t\hat{A}_t$ from Equation (3.55).

2. If $\hat{A}_t$ is negative, the estimated value of choosing a_t in s_t is worse than average. Therefore, we want to *discourage* taking action a_t, still avoiding changing the policy *too* much. Therefore, the ratio is lower bounded to $(1-\epsilon)$. Otherwise, if the ratio is higher than the lower bound, then the minimum of $(1-\epsilon)\hat{A}_t$ and $\bar{r}_t\hat{A}_t$ is the latter since the advantage is negative and both the lower bound and the ratio themselves are positive numbers, and thus the conservative objective is used.

[5]Schulman et al. used r instead of $\bar{r}$, but here $\bar{r}$ has been chosen to avoid confusing the probability ratio with a reward, which has been denoted as r until now, and still will be.

This objective function makes PPO computationally efficient under the assumption that the it could be assumed linear within the trust region in policy space (bounding the probability ratio) which is defined by only *one* hyperparameter: ϵ. The exact implementation in this thesis used PPO with the clipped surrogate objective, an actor-critic structure with a single actor and critic, each represented by their own, separate Neural Network. The critic network outputted the value function estimates, while the actor outputted the mean of a Gaussian distribution using variable standard deviations like in [23, 61] for action selection. The choice of the critic's objective function, the optimizers, the selection of hyperparameters and more implementation details are explained in Section 5.4, while the pseudocode is given in Algorithm 4.

Algorithm 4 Clipped PPO with GAE

```
Input {agent network, critic network, N}:
Initialize actor parameters $\theta$ and optimizer.
Initialize critic parameters $W$ and optimizer.
for all N episodes do
    Collect M trajectories $\mathcal{D}_k = \{\tau_1, ..., \tau_M\}$ under the policy $\pi_{\theta_k}$.
    Calculate returns from all trajectories from t to T, $\hat{G}_t = \sum_{i=t}^{T} \gamma^i r_i$.
    Evaluate all advantages with GAE, $\hat{A}_t = \sum_{i=0}^{T} (\gamma\lambda)^i \delta_{t+i}$, using $\hat{V}_{W_k}$.
    Optimize $\theta$ wrt. objective averaged over all $MT$ samples for $I_a$ iterations:
        $$\theta_{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} \frac{1}{MT} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left( \bar{r}_t(\pi_{\theta_k}) \hat{A}_t,\ \operatorname{clip}\left(\bar{r}_t(\pi_{\theta_k}), 1-\epsilon, 1+\epsilon\right) \hat{A}_t \right) \quad (3.59)$$
    Fit the critic's value function network $\hat{V}_{W_k}$ to the returns for $I_c$ iterations:
        $$W_{k+1} \leftarrow \underset{W}{\operatorname{argmin}} \frac{1}{MT} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( \hat{V}_{W_k}(s_t) - \hat{G}_t \right)^2 \quad (3.60)$$
return Obtained policy $\pi_\theta$.
```

Chapter 4

System description

4.1 The ReVolt Model Ship

This chapter explains the physical system in brief. For more details, the reader is referred to the master's thesis of Alfheim and Muggerud [33], which gave an extensive walk through of the physical dimensions, sensor hardware, software, and more. In short, the ship model consisted of the hull, two 12V batteries, a Tank-720 computer running the Ubuntu operating system, sensor hardware including Inertial Measurement Units (IMU) used for acceleration and rotational velocity measurements, the Global Navigation Satellite System (GNSS) used for position measurements, among other on board sensors. All thrusters had servo motors which were controlled by Arduino microcontrollers. A brief overview of all the components and their placements on board is given in Figure 4.1. On board, an embedded computerized control was enabled by Alfheim and Muggerud, using the programming architecture Robot Operating System (ROS) to connect all hardware measurements and calculations with the software. The control system in ROS included an observer, a reference filter, a motion controller among many things, all being adjustable by a Graphical User Interface (GUI). The existing thrust allocation software used was provided by DNVGL. The mathematical modeling and calculations of the hydrodynamic coefficients had also been done (which can be seen in Appendix A), and the dimensions of the model ship can be seen in Figure 4.2. Since Alfheim and Muggerud wrote their thesis, additional sensors like a Lidar and a 360° camera has been mounted, used for improving the situational awareness and for object detection and tracking. Their placement can be seen in the introductory image in Figure 1.1 and Figure 4.1. In addition, new tests have been performed in a towing tank from which the thruster parameters was updated. The thrusters' specifications used in this thesis are shown in Table 4.1, and their coefficients will be explained in Section 5.1.1.

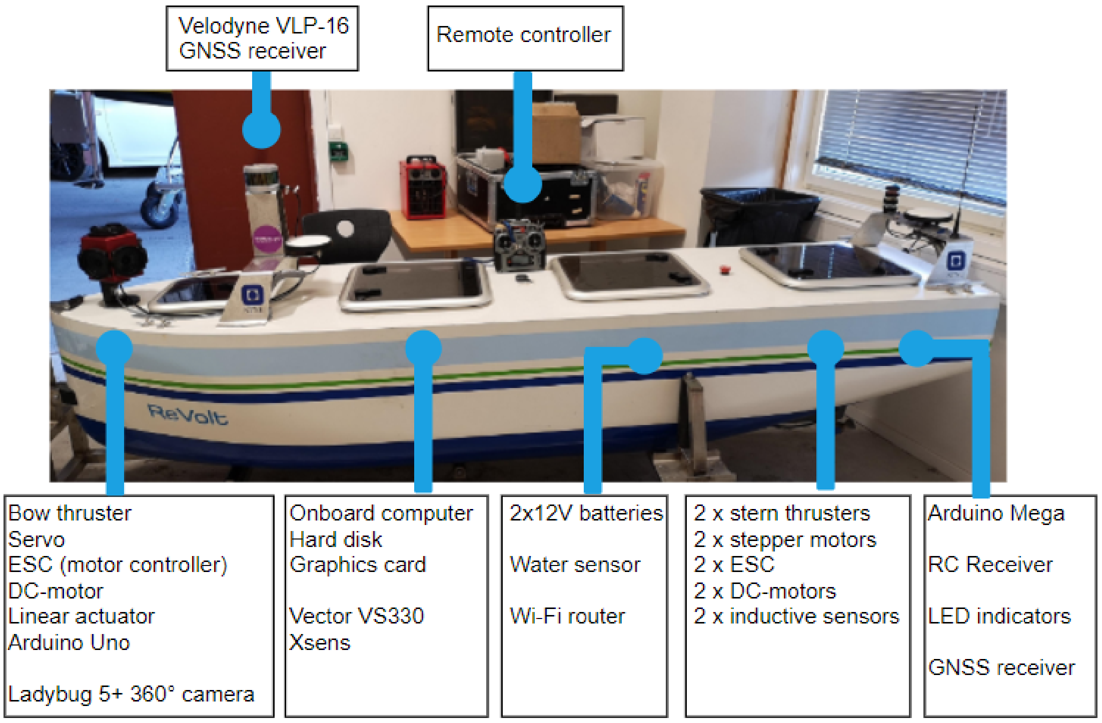


Figure 4.1: An outline of all components of the ReVolt ship model.

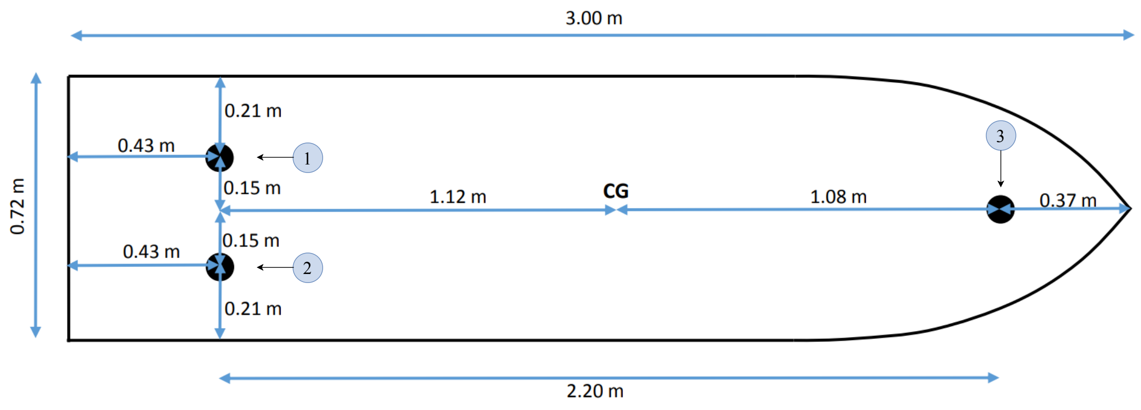


Figure 4.2: ReVolt ship model dimensions. Courtesy: Alfheim and Muggerud [33].

Table 4.1: ReVolt's thruster specifications.

Thruster	l_x [m]	l_y [m]	D [m]	Sectors
1	-1.12	-0.15	0.15	Fully rotatable
2	-1.12	0.15	0.15	Fully rotatable
3	1.08	0	0.06	$(-270°, 270°)$

The thrusters' angles were defined in their separate body-frames, where 0° represents the thrusters pointing parallel to the vessels' longitudinal direction, and were defined positive when rotating clockwise. The enumeration of the thrusters was: (1) the port thruster at the stern, (2) the starboard thruster at the stern, and (3) the bow thruster. Note that the definition of the direction of the *force vectors* from each thruster followed the definition of their angles.

4.2 Framework

4.2.1 Simulation Platform

DNVGL developed a simulation platform for testing control systems of different vessels, called Open Simulation Platform (OSP). In it, a digital twin of ReVolt was added in order to carry out testing of the software being developed, which was used in this thesis. The simulator contained a lot of useful functionality like modeling of surface, wind, and wave forces going from calm seas to extreme sea conditions. Figure 4.3 shows the simulator interface, where the ReVolt ship model is placed in Dorabassenget in Trondheim, Norway.

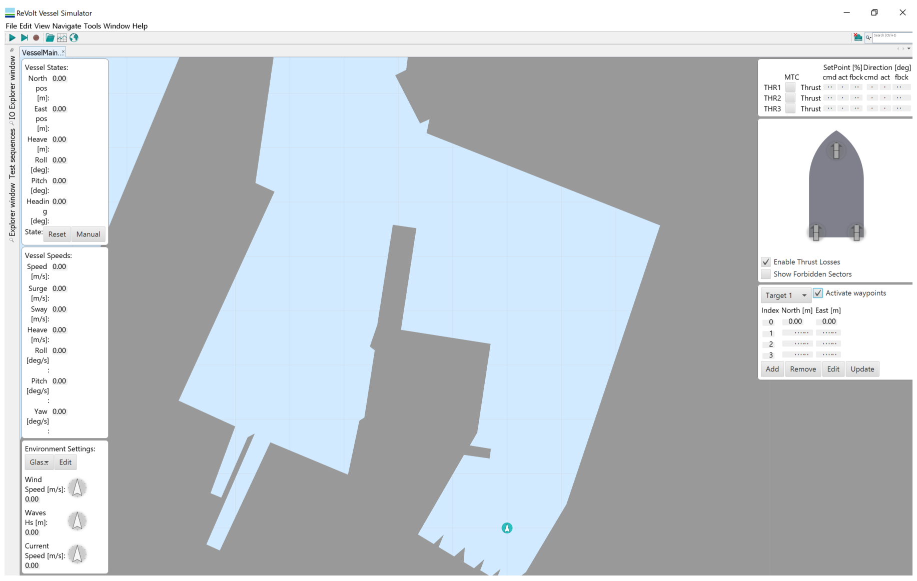


Figure 4.3: Simulator for the digital twin of ReVolt, showing Dorabassenget in Trondheim.

During simulations, the simulator ran on a computer with the Microsoft Windows operating system, and the digital twin was controlled by using the control system on a Linux computer. The two computers connected through an Ethernet cable such that the control system on the Linux computer could send control signals to the simulator. In sea trials however, all code was running directly on the Tank-720 computer on board, with the possibility of connecting to it through WiFi for making adjustments to the code.

4.2.2 Software

The code running the control system was mainly written in the two programming languages C++ and Python. The different scripts and configuration files were connected in the ROS[1] framework, which is an open-source software framework, enabling code written in different programming languages to interface with each other. Hence, ROS worked as the "core" of the entire software architecture, enabling communication between "nodes" (e.g. communication between the observer, the motion controller and the thrust allocation) that could be written in the programming language of choice.

[1]https://www.ros.org/

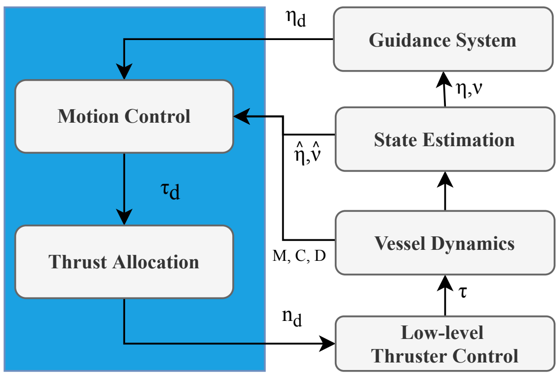
The specific computer used when training the Neural Networks, and for running the simulator, was a Huawei Matebook X Pro with an Intel i7-8550U CPU with 8 cores running at 1.8 GHz clock speed and 16 GB RAM. For simulating the training results, the model was loaded onto the control system of ReVolt a Dell Latitude E7470 with 16 GB RAM, equipped with an Intel Core i7-6600U CPU with four cores with 2.6 GHz clock speed. At the time, Ubuntu 16.04.6 LTS 64-bit was the operating system. For this thesis, Python was considered to be the most convenient language to program in due to the language being the most popular language for computer science in general. Especially, there exists a lot of optimization, statistics and machine learning relevant packages. The SciPy(2) package was used for implementing thrust allocation using Quadratic Programming, while Tensorflow(3) was used for handling the implementation and training of the Neural Networks. Since none of the computers had graphics cards with the necessary compatibility for GPU-versions of Tensorflow, the CPU-version was used.

(2)https://www.scipy.org/
(3)https://www.tensorflow.org/

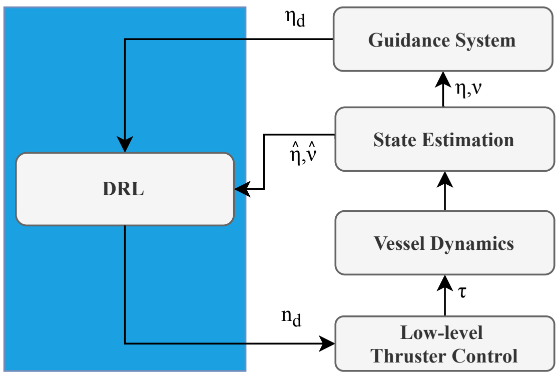
Chapter 5

Implementation

This section describes the details of the all of the methods used during tests, ranging from the classic methods used for comparison to the DRL method, and the DRL method itself. All the methods were implemented as individual ROS nodes. According to Sørensen [4], full scale thrust allocation systems should command signals to the actuators at a rate down to 1 second. By using Froude scaling, the scaling factor between the length of the full scale ReVolt and the model vessel was $\lambda = 3\ m/60\ m = 1/20 = 0.05$. As Froude scaling for time is done with a factor of $\lambda^{0.5}$, 1 second at full scale corresponds to $0.05^{0.5} \approx 0.22\ s$ at model scale. Rounding down to 0.2 s served as a lower bound, so the ROS nodes were set to operate at $(0.2\ s)^{-1} = 5\ Hz$. All methods also used the same reference filter, which has already been implemented by previous students. The classic thrust allocation methods performed their allocation after receiving the desired force vector from the existing implementation of a motion controller, while the DRL method performed thrust allocation directly from the reference filter signal; a difference which is illustrated in Figure 5.1. Following experiences done by Alfheim and Muggerud [33] and from the project thesis [1], the thrust allocation was performing in the best by fixing the bow thruster to 90°, making it act as a tunnel thruster. Accordingly, the actions being decided from the different control methods consisted of three revolutions per second (RPS) values (n) for the thrusters' propellers, and the two angles of the stern thrusters (α).



(a) Classic methods.



(b) Proposed method.

Figure 5.1: Flowcharts of the difference between classic vs. proposed method.

5.1 Existing Implementations

The implementation of this thesis builds on top of previous work as mentioned in Section 1.3.3. Therefore, some of the parts are explained in the following to give the reader an insight into the components that were used. The observer, responsible for state estimation, was recently upgraded during the summer of 2019, and was found to be performing well in the simulator. Other components was however tweaked in order to improve the performance of both the existing methods, but also the proposed DRL method. Section 5.1.1 explains the parameters which was used for the thrusters in the simulator, while Section 5.1.2 explains the parameters of the reference model and the motion controller.

5.1.1 Thruster Parameters

As mentioned by Alfheim and Muggerud [33], the thrusters made for ReVolt was custom made, hence many of the needed parameters were not known. In order to formulate a thruster force model for each thruster, they performed a Bollard pull test[(1)] in order to estimate coefficients for the three thrusters. The resulting non-linear model was found through measuring exerted force for the three different thrusters while going through the RPS-range of the thrusters from -100% to 100%. The resulting plots are shown in Figure 5.2. **Note** that u was used as their notation for "percentage of maximum RPS of the propeller". In the ROS implementation, the signals being sent to the thrusters were given as percentages of maximum. Therefore, the thruster coefficients had units of Newton, while the thrusters' RPS were given as percentage of max RPS. It was also found that the stern thrusters had the same force profile for rotating in both negative and positive directions, while the bow thruster (being unsymmetrical) yielded considerably less force when rotating in the negative direction versus the positive direction. The resulting thruster force formula used was as shown in Equation (5.1), giving the expression, when inserted into the linear actuator model in Equation (2.7), as shown in Equation (5.2).

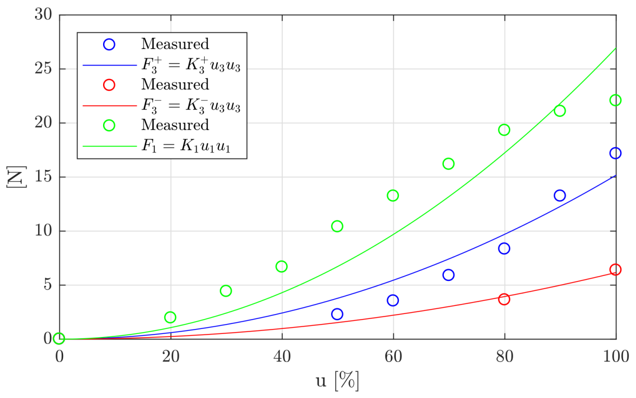


Figure 5.2: Results from bollard pull test. Courtesy: Alfheim and Muggerud [33].

$$T_i^{\pm} = K_i^{\pm} n_i |n_i| \tag{5.1}$$

$$\begin{bmatrix} X^{\pm} \\ Y^{\pm} \\ N^{\pm} \end{bmatrix} = \begin{bmatrix} c\alpha_1 & c\alpha_2 & c\alpha_3 \\ s\alpha_1 & s\alpha_2 & s\alpha_3 \\ l_{x,1} s\alpha_1 - l_{y,1} c\alpha_1 & l_{x,2} s\alpha_2 - l_{y,2} c\alpha_2 & l_{x,3} s\alpha_3 - l_{y,3} c\alpha_3 \end{bmatrix} \begin{bmatrix} K_1^{\pm} n_1 |n_1| \\ K_2^{\pm} n_2 |n_2| \\ K_3^{\pm} n_3 |n_3| \end{bmatrix} \tag{5.2}$$

(1) https://www.sciencedirect.com/topics/engineering/bollard-pull

The values found from the test above was used previously in the existing software in ROS for the motion controller and thrust allocation method. However, what was found during this thesis was that the parameters used as thruster coefficients in the simulator was not the same as the ones shown above. This was due to new tests of ReVolt performed in the towing tank at NTNU. The result of those tests gave rise to new thruster coefficients which was updated in the simulator but not in the software in ROS. The new maximum forces for each thruster was therefore updated in the ROS nodes according to the ones in the simulator, and is showed in Table 5.1, which includes the thrusters force production rates and and angular rates as well. Note that the bow thruster was now set to have equal values in all directions.

Table 5.1: New thruster parameters.

Coefficient	K_1 $[N]$	K_2 $[N]$	K_3 $[N]$
$K_i^{\pm}$	$2.05 \cdot 10^{-3}$	$2.05 \cdot 10^{-3}$	$0.9 \cdot 10^{-3}$
Max force	T_1 $[N]$	T_2 $[N]$	T_3 $[N]$
$T_{i,max}$	20.5	20.5	9.0
Force rate	ΔT_1 $[N/\Delta t]$	ΔT_2 $[N/\Delta t]$	ΔT_3 $[N/\Delta t]$
$\Delta T_{i,max}$	10.0	10.0	4.0
Angular rate	$\Delta\alpha_1$ $[rad/\Delta t]$	$\Delta\alpha_2$ $[rad/\Delta t]$	$\Delta\alpha_3$ $[rad/\Delta t]$
$\Delta\alpha_{i,max}$	$\pi/6$	$\pi/6$	0 (static)

5.1.2 Reference Filter and Motion Controller Specifications

The state-space equation for the reference filter used on ReVolt was as previously shown in Equation (2.3). The 3-dimensional diagonal matrices $\Delta > 0$ and $\Omega > 0$ contained the relative damping ratios and natural frequencies of the reference filter, respectively. The tuned values of these matrices were based on [33] and the existing implementation in ROS. Their coefficients are shown in Equation (5.3).

$$\Delta = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}, \ \Omega = \begin{bmatrix} 0.6 & 0.0 & 0.0 \\ 0.0 & 0.6 & 0.0 \\ 0.0 & 0.0 & 1.4 \end{bmatrix} \tag{5.3}$$

The reference filter's coordinates is in the order of 9 dimensions, outputting the desired pose (surge, sway and yaw) in both position, velocity and acceleration. All dimensions were used as inputs to the motion controller, while the DRL controller only used the pose (3 dimensions) since it was not trained with any reference filter in the simulator in Windows, where all the setpoints for velocity was set to zero. The motion controller, as explained in Section 2.3.2, was tuned in [33]. That work was done in 2016, and since, there has been several summer and master's students working on software. During this thesis, it was found that the outputs of the motion controller consisted of desired forces and yaw moment larger than what was feasible for the thrusters' capabilities. New calculations was done in order to find the maximum producible forces and moment with the thruster setup being as discussed earlier, setting the bow thruster fixed at 90°. These calculations can be seen in Appendix B, with the resulting bounds of the motion controller being

$$\boldsymbol{\tau}_{max} = [41\ N,\ 17.7\ N,\ 20.0\ Nm]^\top. \tag{5.4}$$

5.2 DNVGL's Pseudoinverse

The thrust allocation provided from DNVGL was as mentioned based on the pseudoinverse method. This was a replica of standard thrust allocation software, as they do not provide such software commercially. The method was set up according to the extended thrust formulation, in addition to taking the thrusters' limits into consideration. Initially, the method finds a pseudoinverse as explained in Section 2.3.3. If all thrusters are within their legal areas, then the method performs a check to constrain the outputs from breaching the rate limits, and publishes the results. However if some thrusters violates the rate limits, they are frozen to their maximum force production, and a new calculation of the desired forces is done by reducing the desired force vector by subtracting the force exerted from those thrusters who has been frozen to their maximums. From this, a new pseudoinverse is found in an iterative manner until an accepted solution has been found as shown in Algorithm 5. The implementation published the individual, commanded thruster angles and *forces*. As the thrusters' servo motors required commanded RPS signals, these forces was translated to RPS by using the thruster model in Equation (5.2), solving for $n = sgn(T/K)\sqrt{|T/K|}$. Due to giving a fair comparison to the other methods, no forbidden zones of the thrusters' angles were enabled. Du to the iterative nature of the method, it will be refered to as Iterative Pseudoinverse (IPI) hereafter.

Algorithm 5 Iterative Pseudoinverse (IPI) thrust allocation

```
Input {τ_d, previous thruster states, time limit}:
Define boundaries such as maximum force production and forbidden zones.
Define constraints such as thrusters' rates.
Initialize thruster configuration $B_{ext}$ and weights $W$.
Find thrust vector from the pseudoinverse, $\boldsymbol{T}_{ext} = W^{-1}B_{ext}^\top(B_{ext}W^{-1}B_{ext}^\top)^{-1}\boldsymbol{\tau}_d$.

while (Not all bounds are met) and (time limit is not breached) do
    Freeze the thrusters exceeding their limits and recalculate the desired forces.
    Recalculate the force vector using new pseudoinverse.

if (No solution within bounds are found within time limit) then
    Use thrust vector from solution with the smallest deviation between $\boldsymbol{\tau}_d$ and $\boldsymbol{\tau}$.

Calculate the resulting thrust and angles from Equation (2.12) and Equation (2.13).
Constrain the thrust and angles according to the thrusters' rates.
Translate thrust to thruster commands through $n_i = sgn(T_i/K_i)\sqrt{|T_i/K_i|}$.

return (n and α)
```

5.3 Quadratic Programming

The implementation of the quadratic programming (QP) approach was based on Section 2.3.3. As mentioned there, the fact that the vessel is overactuated means that there are several force/angle combinations from the thrusters that could yield the same resultant force vector. In order to find a particular solution from the multiple ones, it is possible to put a cost to both thruster forces and rotation rates in order to optimize for power consumption, and wear and tear, which was done as in the following.

Constraints used in the optimization was the maximum producible forces, the force production rates, and the angular rates of the azimuths. These were inequality constraints, which means that they were constraints which *bounds* the decision variables. Therefore, decision variables that are not expressed in the cost function are found as solutions based on their constraints. All the numerical values used were as shown in Table 5.1.

Due to the fact that decision variables are formulated as constraints containing trigonometric functions, the equality constraints are *non-linear*. This means that the solver chosen had to accept non-linear constraints. It was ensured that the solver was quick enough so that the ROS node was able to calculate solutions at 5 Hz. Various solvers in Python were tested, where none was slower than 20 Hz during the preliminary tests. The package `scipy.optimize` was selected due to its simplicity. The method within its optimize-package called `minimize` was chosen, using Sequential Least Square Quadratic Programming[(2)]. As an additional measure, since `scipy.optimize.minimize` required to set a certain iteration limit for the optimization, it was found that it failed to find a solution within the given limit, even though it reached the limit very fast. It was difficult to find a good iteration limit, and as a way of countering this, it was decided to increase the bounds of the slack variables if the iteration limit was reached, giving rise to an "Iterative Quadratic Program".

The decision variables was chosen to be all thruster forces and slack variables, as shown in Equation (5.5a). The decision variables were augmented as shown in Equation (5.5b) in order to include the thrusters' rates in the objective function, however the rates were not to be *decided*; they were only included for the means of *minimization*. Since the power usage of a propeller is proportional to the thrust force to the power of 1.5 ($P \propto |T|^{3/2}$), the augmented decision variable vector reflected this. Note that the minimum of $|T|$ and $|T|^{3/2}$ is the same, but the latter penalizes larger T-values more. This formulation ensured that both the thruster power usage was subject to minimization, and that the magnitude of the slack variables were small in order to get $\boldsymbol{\tau} - \boldsymbol{\tau}_d$ as close to zero as possible.

$$\boldsymbol{x}_t = \Big[(T_1)_t,\ (T_2)_t,\ (T_3)_t,\ (s_1)_t,\ (s_2)_t,\ (s_3)_t\Big]^\top \tag{5.5a}$$

$$\begin{aligned}\boldsymbol{y}_t = \Big[&(T_1^{3/2})_t,\ (T_2^{3/2})_t,\ (T_3^{3/2})_t,\ (s_1)_t,\ (s_2)_t,\ (s_3)_t,\ |(\alpha_1)_t - (\alpha_1)_{t-1}|, \\ &|(\alpha_2)_t - (\alpha_2)_{t-1}|,\ |(T_1)_t - (T_1)_{t-1}|,\ |(T_2)_t - (T_2)_{t-1}|,\ |(T_3)_t - (T_3)_{t-1}|\Big]^\top\end{aligned} \tag{5.5b}$$

[(2)] https://optimization.mccormick.northwestern.edu/index.php/Sequential_quadratic_programming

The solver had to be given initial values for the decision variables for which it starts its solving process at each time step. They were set to be the previous time step's thruster states stored in the ROS node. All constraints to the optimizer had to either be formulated as equality constrains including slack variables, or inequality constraints in order to bound the variables. All inequality constraints had to be set in the format of "greater than zero". Therefore, any constraint defined by upper and lower bounds, e.g. $\Delta T_{min} \leq \Delta T \leq \Delta T_{max}$ was formulated as $\Delta T_{max} - \Delta T \geq 0$ and $-\Delta T_{min} + \Delta T \geq 0$. The entire mathematical programming formulation is summarized, with relevant numerical values, in Equation (5.6). The constraints are enumerated in Equation (5.6) as the following: (***i***) limit the difference between desired and produced forces/moments, (***ii***) limit the largest producible force from each thruster, (***iii***) limit the largest negative producible force from each thruster, (***iv***) limit the largest thrust production rate, (***v***) limit the largest negative thrust production rate, (***vi***) limit the maximum positive angular rotation rate, and (***vii***) limit the maximum negative angular rotation rate.

$$
\begin{aligned}
\min_{\boldsymbol{y}_t \in \mathbb{R}^{11}} \quad & \boldsymbol{y}_t^\top \cdot \boldsymbol{Q}_{QP} \cdot \boldsymbol{y}_t \\
\text{subject to} \quad & B(\boldsymbol{\alpha}_t)\boldsymbol{T}_t - (\boldsymbol{\tau}_d)_t - \boldsymbol{s}_t = 0, & (\boldsymbol{i}) \\
& \begin{bmatrix} 20.5 \\ 20.5 \\ 9.0 \end{bmatrix} N - \boldsymbol{T}_t \geq 0, & (\boldsymbol{ii}) \\
- & \begin{bmatrix} -20.5 \\ -20.5 \\ -9.0 \end{bmatrix} N + \boldsymbol{T}_t \geq 0, & (\boldsymbol{iii}) \\
& \begin{bmatrix} 10 \\ 10 \\ 4 \end{bmatrix} {}^{N}\!/\!_{\Delta t} - \left(\frac{\boldsymbol{T}_t - \boldsymbol{T}_{t-1}}{\Delta t} \right) \geq 0, & (\boldsymbol{iv}) \\
- & \begin{bmatrix} -10.0 \\ -10.0 \\ -4.0 \end{bmatrix} {}^{N}\!/\!_{\Delta t} + \left(\frac{\boldsymbol{T}_t - \boldsymbol{T}_{t-1}}{\Delta t} \right) \geq 0, & (\boldsymbol{v}) \\
& \begin{bmatrix} \pi/6 \\ \pi/6 \end{bmatrix} {}^{rad}\!/\!_{\Delta t} - \left(\frac{\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}}{\Delta t} \right) \geq 0, & (\boldsymbol{vi}) \\
- & \begin{bmatrix} -\pi/6 \\ -\pi/6 \end{bmatrix} {}^{rad}\!/\!_{\Delta t} + \left(\frac{\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}}{\Delta t} \right) \geq 0. & (\boldsymbol{vii})
\end{aligned}
\tag{5.6}
$$

The bounds for the slack variables was set to be 1 N and 1 Nm initially, which set the magnitude of the allowed deviation between the desired and commanded force vectors. The increment factor, deciding how much the slack variables were allowed to increase if no solution was found initially due to the iteration limit, was set to 2 N and 2 Nm. The weight matrix used in the objective function was a diagonal 11x11 matrix $Q_{QP} = \text{diag}\left([1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 1.0,\ 0.25,\ 0.25,\ 0.5,\ 0.5,\ 0.5]\right)$, and the pseudocode of the implementation can be seen in Algorithm 6.

Algorithm 6 Iterative Quadratic Programming

```
Input {$\tau_d$, $x_{t-1}$, QP-solver, increment factor, time limit}:
 Set up objective function, using $y_t$ and $Q_{QP}$.
 Define boundaries of the decision variables.
 Initialize all constrains, including $\tau_d$ and $x_{t-1}$.
 Initialize decision variables with previous thruster states in $x_{t-1}$.
 Find thrust and angles through the QP-solver.

 while (No solution was found) and (time limit is not breached) do
     Increase bounds for slack variables with increment factor.
     Find the thrust and angles through the QP-solver.

 if No solution was found then
     Use solution from previous time step.

 Translate thrust to thruster commands through $n_i = sgn(T_i/K_i)\sqrt{|T_i/K_i|}$.

 return ($n$ and $\alpha$)
```

5.4 Deep Reinforcement Learning

For the implementation of the DRL agent, the work was divided into three phases: (1) building the framework in which PPO was to be used, (2) training the agent while iteratively tuning the reward function and network architectures, and (3) verifying/validating the performance afterwards. This section explains how the environment was created, how the state and action vectors of the agent was decided, how the network structure and features were decided upon, and how the reward function was created in order to achieve the desired behavior in DP-operations. The material presented is the resulting details of the final DRL model used for testing in Chapter 6, but should be read with the fact in mind that it was carried out as an iterative process, since the findings from one of the subsections might have led to changes in another.

5.4.1 Environment

The environment that the agent was to interact with was the aforementioned simulation platform. The simulator ran in the Windows operating system, so the learning algorithm was decided to be implemented in Windows, using the Python programming language. Several classes was made in Python in order to interact with the PPO algorithm while communicating with the simulator which was running on Java. The flow of information between these classes can be visualized in Figure 5.3.

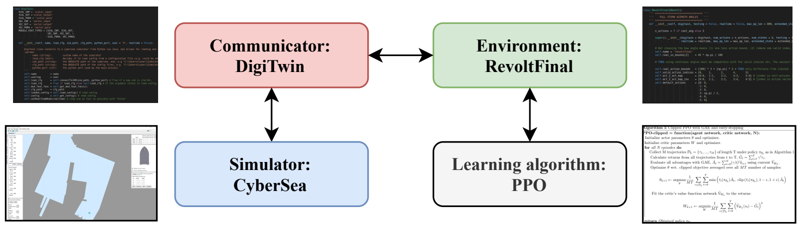


Figure 5.3: Flowchart of software implementations used during training.

For the pose, the state-space bounds were set to be within a reasonably large area around a setpoints, meaning that, relative to the vessel length, a maximum body-frame error was to be set in which the agent was not allowed to operate within. It was decided that the body-frame positional errors were allowed to be at a factor of 2.5 the vessel length, and that the yaw errors was not be allowed to be larger than 45°, giving $\boldsymbol{\eta}_{bound} = [8\ m,\ 8\ m,\ \pi/4\ rad]$. These were considered to be pretty large values at first, but after just a short amount of training, the DRL method was never close to coming near these boundaries. For the velocities, a test was run while activating thruster losses in the simulator. The velocity boundaries were set to $\boldsymbol{\nu}_{bound} = [\ |u_{max}|,\ |v_{max}|,\ |\dot{\psi}_{max}|\ ] = [1.4\ {}^{m}/{}_{s},\ 0.3\ {}^{m}/{}_{s}, 0.52\ {}^{rad}/{}_{s}]$. If the agent's states exceeded some of these values, it was treated as a *terminal* state. No states were treated as a *goal state*, which is explained in Section 5.4.8.

5.4.2 State Vector

The state vector for the actor and the critic was developed with focus on the agents' ability to reduce body-frame errors. The body-frame errors were added as the first components to the state vector, and it was believed that it could be beneficial for the agent to have access to the body-frame velocities u, v and $\dot{\psi}$, which was seen by previous work [28, 29, 30, 32]. To improve the agents ability to reduce the total thruster usage, it was decided to add the previous commanded thruster's RPS (as a fraction between -1 and 1, denoted $\bar{n}$). This led to the state vector of length 9 as shown in Equation (5.7a). The body-frame errors was calculated using the pose error in the NED-frame, $\tilde{\eta} = \eta - \eta_d = [\tilde{N}, \tilde{E}, \tilde{\psi}]^\top$ as shown in Equation (5.7b) and Equation (5.7c). $\tilde{\psi}$ was wrapped between $(-\pi, \pi]$ radians before being used in the state vector, while all other quantities was extracted straight from the simulator.

$$\boldsymbol{s}_t = [\tilde{x}_t,\ \tilde{y}_t,\ \tilde{\psi}_t,\ \tilde{u}_t,\ \tilde{v}_t,\ \tilde{\dot{\psi}}_t,\ \bar{n}_{bow,t-1},\ \bar{n}_{port,t-1},\ \bar{n}_{star,t-1}]^\top. \tag{5.7a}$$

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix}^\top \begin{bmatrix} \tilde{N} \\ \tilde{E} \end{bmatrix} \tag{5.7b}$$

$$\tilde{\psi} = \psi - \psi_d \tag{5.7c}$$

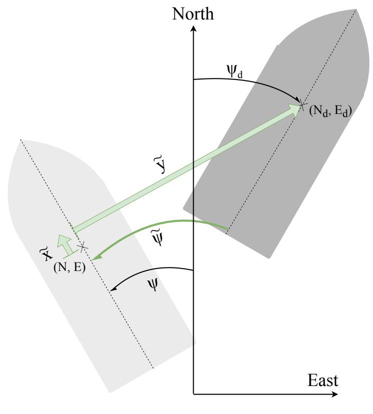


Figure 5.4: Example of body-frame errors (green arrows), showing desired pose in dark gray.

5.4.3 Action Vector

The action vector, being the output of the policy network, was set to consist of the RPS of each thruster, and the angles of the thrusters, all as components in the range [-1, 1]. The stochastic, Gaussian policy outputs values with a mean of 0, but the values might be smaller than -1 and larger than +1. Therefore, the RPS outputs from the actor was clipped to the range [-1, 1] before being scaled to [-100 %, 100 %] for the simulator. For the clipping and scaling of the angles, different approaches was tested to combat the issue of representing the continuity of angles, meaning that $-179°$ and $179°$ should be perceived as being $2°$ apart, rather than $358°$ apart. This challenge was attempted combated in two ways:

1. The actor's output was directly was wrapped to the region $(-\pi, \pi]$.

2. Instead of representing the angles as their raw values, their sine and cosine was to be outputs of the network. The advantage of this approach was that (1) the trigonometric functions already fits into the region [-1,1], and (2) the angle was represented as a circular parameter by selecting the angles through α `= arctan2(sin(), cos())`, where `arctan2()` represents the four-quadrant inverse tangent function. Appendix C demonstrates the output of `arctan2()`, showing that the action vector setup is able to represent all angles in addition to doing it in a circular manner in the region (-180°, 180°].

The action vector of the first approach was therefore as shown in Equation (5.8a), while the second approach resulted in an action vector as shown in Equation (5.8b) (the hat represents that it is the predicted sines and cosines from the network, and not the *actual* sines and cosines, as the `arctan2()`-function is responsible for calculating the *actual* angle).

$$\boldsymbol{a}_t = [n_{bow,t},\ n_{port,t},\ n_{star,t},\ \alpha_{port,t},\ \alpha_{star,t}]^\top \tag{5.8a}$$

$$\begin{aligned}\boldsymbol{a}_t = \ &[n_{bow,t},\ n_{port,t},\ n_{star,t},\\ &sin(\hat{\alpha}_{port,t}),\ cos(\hat{\alpha}_{port,t}),\ sin(\hat{\alpha}_{star,t}),\ cos(\hat{\alpha}_{star,t})]^\top\end{aligned} \tag{5.8b}$$

The results of the training sessions using the different action vectors has been added to Appendix D, from which it can be seen that the second action vector formulation displayed the fastest development in turns of average episodal return and convergence of the average value function estimates. The experiments was done with reward functions of varying complexity, and no matter the reward function, the second formulation resulted in faster convergence. In addition, it was being more consistent when validating after training (as explained in Section 5.4.11). Therefore, the action vector from Equation (5.8b) was used.

5.4.4 Reward Shaping

The way rewards are assigned is essential for how the agent is updated towards exhibiting a good policy. In order to achieve a model which learns a desired behavior, it is *crucial* to have a well formulated reward function. The reward function encapsulates everything that the agent is supposed to learn that leads to good and bad actions, and the agent learns what the reward function tells it to, and not what the designer *intended* it to do, so it must be shaped with caution. In order to guide the learning process for helping the agent towards the maximum reward, a process of *reward shaping* was commenced. Reward shaping stems from the idea of adding supplemental rewards in order to guide the learning process toward the natural reward of the problem (being zero pose deviation for DP operations) [63]. As an example, instead of only rewarding the agent for getting into a "goal state", the agent can receive reward for getting closer and closer to the goal. Several reward functions might fit this supplemental formulation, in which a few is discussed in the following.

The most intuitive way of rewarding the agent for reaching a setpoint could be to give a reward of 1 for getting into a region of accepted deviation from the desired pose, and 0 otherwise. This has been discussed by others previously [28, 31, 32]. A problem here arises due to "sparsity", meaning that it is hard for the agent to learn what to do when it is in an area of the state-space which yields zero rewards. As proposed by Martinsen [28], a Gaussian-shaped reward function could be used in order to combat sparsity. Such a function is showed in Equation (5.9) (being a *univariate* Gaussian function since it depends on *one* variable) where a is a constant deciding the amplitude, μ is the mean value around which the Gaussian is centered, and σ is the standard deviation.

$$f(x) = ae^{-\frac{1}{2}(\frac{(x-\mu)^2}{\sigma})} \tag{5.9}$$

If x represents the body-frame error for DP operations, then $\mu = 0$ is used, and a and σ becomes tunable parameters. This Gaussian was utilized initially, one with the euclidean distance $d = \sqrt{\tilde{x}^2 + \tilde{y}^2}$, and another with the body-frame error in yaw $\tilde{\psi}$. The parameter a was set to be 1 for each of the components, and the total reward function was the sum of the two Gaussian functions. In most DP operations, the priority of holding the desired yaw angle is prioritized before position [15, 64, 65], which was taken into account when testing for which values of σ that gave good results. Good behavior of the agent (following the validation process explained in Section 5.4.11) was found when $\sigma_d = 1.0\ m$ and $\sigma_{\tilde{\psi}} = 5.0°$.

As seen from Figure 5.5a, if the deviation of both position and yaw angle is large, this summed Gaussian reward function has many areas where there are large positive gradients in both the direction of lowering positional error and the yaw angle error. E.g, if $\tilde{\psi} = -20°$ and $d = 5\ m$, the fastest increase in rewards comes from reducing d first. This was also learned by the agent, which caused undesirable vessel motions when far away from the setpoint. In order to combat this behavior, a modification to the reward function was proposed, being a *multivariate* Gaussian. The formulation was motivated by the shape of the probability distribution of the policy itself, meaning that the multivariate Gaussian can be expressed as in Equation (5.10), where $\boldsymbol{x}$ is the vector containing multiple variables and Σ is the matrix containing the variances of each variable, called the *covariance matrix.*

$$\boldsymbol{f}(\boldsymbol{x}) = ae^{-\frac{1}{2}(\boldsymbol{x}^\top \Sigma^{-1}\boldsymbol{x})} \tag{5.10}$$

It was decided to use a diagonal covariance matrix for simplicity, meaning that the diagonals of the matrix contained the σ^2-values of each variable. Using the same values of σ as in the case of the sum of the univariate Gaussian reward functions gave the shape shown in Figure 5.5b. The resulting reward function for the pose was therefore formulated as R_{gauss}, shown in Equation (5.11b). A comparison of the summed univariate Gaussians and the multivariate Gaussian is shown in Figure 5.5.

$$\Sigma = \begin{bmatrix} \sigma_d^2 & 0.0 \\ 0.0 & \sigma_{\tilde{\psi}}^2 \end{bmatrix} \tag{5.11a}$$

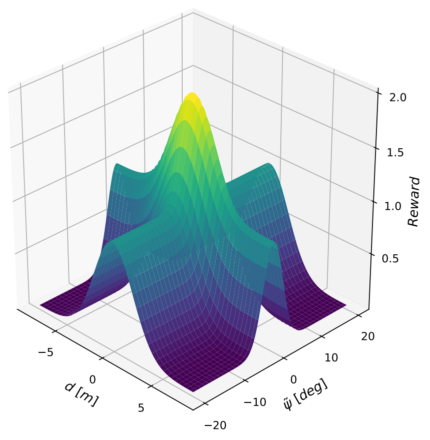
$$R_{gauss} = c_{gauss} \cdot e^{-\frac{1}{2}\left(\begin{bmatrix} d & \tilde{\psi} \end{bmatrix} \Sigma^{-1} \begin{bmatrix} d \\ \tilde{\psi} \end{bmatrix}\right)} \tag{5.11b}$$

A term was added as an "anti-sparsity" (AS) to decrease the sparsity of R_{gauss}, which increased linearly towards the goal of zero deviation. Since the function was multivariate, the linear function was shaped to look like a cone in the $(d, \tilde{\psi})$-space. A special measurement ϕ was used as a weighted distance metric in the $(d, \tilde{\psi})$-space as shown in Equation (5.12a), where $\tilde{\psi}$ was divided by 4 since the magnitude of the values of the yaw angles relative to the positional values were much larger for the range of values used during the learning task. It was given the shape of $(1 - c_{AS}\ \phi)$ having a maximum value of 1 at the origin in $(d, \tilde{\psi})$-space, and was lower bounded at 0.0 to avoid negative rewards as shown in Equation (5.12b). In order to ensure that it was not more valuable for the agent to output zero RPS when d and $\tilde{\psi}$ was large (meaning R_{gauss} was close to zero), a constant term was also added. The resulting reward function for the pose, R_η, was the sum of the multivariate Gaussian and the anti-sparsity term as shown in Equation (5.13), and is by the author's knowledge new to the field. All its components has been plotted separately in Appendix E, and the resulting shape is shown in Figure 5.6.

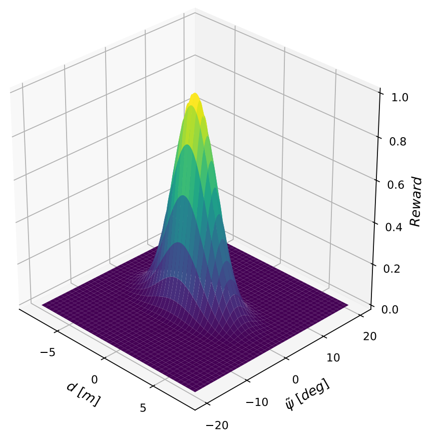
$$\phi = \sqrt{(d)^2 + (\tilde{\psi}/4)^2} \tag{5.12a}$$

$$R_{\text{AS}} = \max\left(0.0, (1 - c_{AS}\ \phi)\right) + c_{const} \tag{5.12b}$$

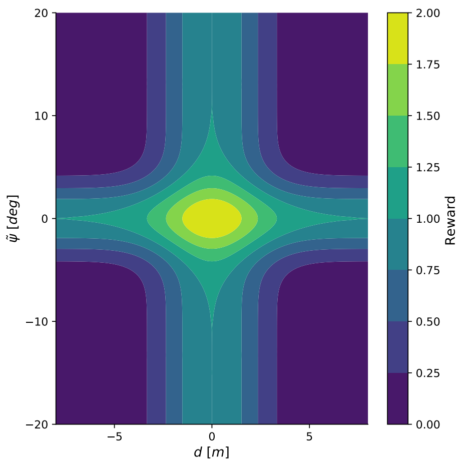
$$R_\eta = R_{gauss} + R_{\text{AS}} \tag{5.13}$$



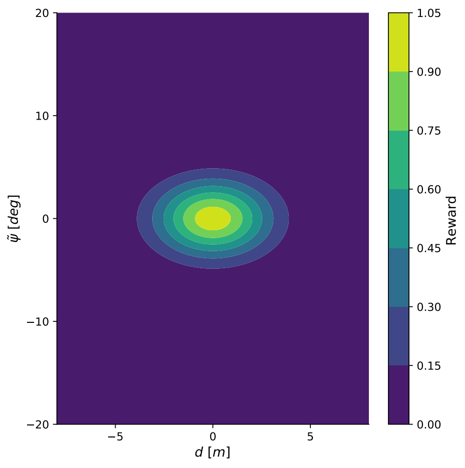
(a) Summed univariate Gaussians.



(b) Multivariate Gaussian.

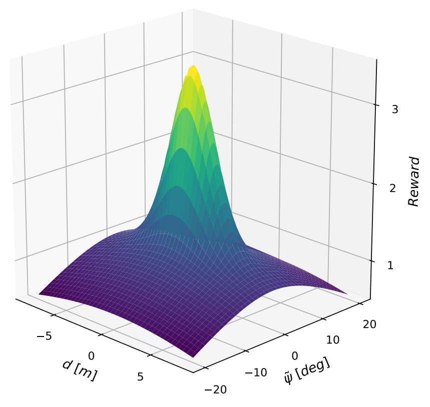


(c) Contour of summed Gaussians.

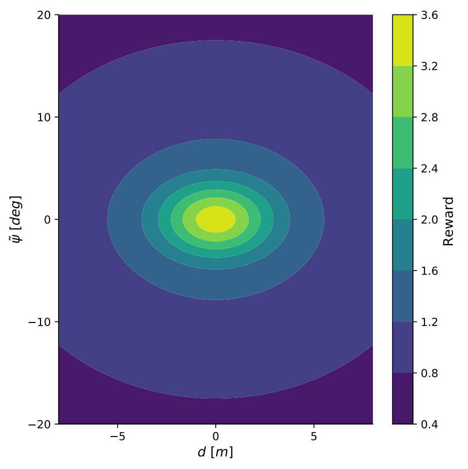


(d) Contour of multivariate Gaussian.

Figure 5.5: Difference between Gaussian functions, using $\sigma_d = 1.0$ and $\sigma_{\tilde{\psi}} = 5.0$.



(a) 3D plot.



(b) Contour plot.

Figure 5.6: Plots of R_η using $c_{gauss} = 2.0$, $\sigma_d = 1.0$, $\sigma_{\tilde{\psi}} = 5.0$, $c_{AS} = 0.1$ and $c_{const} = 0.5$.

Velocity, Energy, and Wear and Tear

Inspired by a classic PD-controller, it was decided to enforce small negative reward on the body-frame velocities to avoid aggressive maneuvers in addition to potential overshoots of the desired pose, as shown in Equation (5.14).

$$R_{vel} = -\sqrt{c_u(\tilde{u})^2 + c_v(\tilde{v})^2 + c_{\dot{\tilde{\psi}}}(\dot{\tilde{\psi}})^2}. \tag{5.14}$$

A negative reward on the actuator usage was added for two reasons, the first being that it was desired to train an energy efficient control system. Second, it was done in order to avoid the agent reducing the body-frame errors while staying at the setpoint and still using the thrusters extensively, causing unwanted wear and tear. Previous work has been adding negative rewards on the use of actuators [28, 27, 32], and as mentioned in [27], linear penalties (being linearly proportional to the actuator outputs) are better at eliminating small errors than e.g. a squared penalty. It was decided to add linearly increasing, negative rewards to the actuator output, as shown in Equation (5.15), where the rewards was calculated from the fractions of the magnitude of the actuator outputs with respect to their maximums.

$$R_{|n|} = -\left(c_{|n|,bow}\frac{|n_{bow}|}{100} + c_{|n|,port}\frac{|n_{port}|}{100} + c_{|n|,star}\frac{|n_{star}|}{100}\right) \tag{5.15}$$

The wear and tear of the actuators was addressed by using small negative rewards to the derivatives of the commanded thruster outputs and angles as shown in Equation (5.16) and Equation (5.17), respectively. The derivatives was normalized with a factor of 100 for the derivatives of the RPS values, and a factor of π for the azimuth angle derivatives. Note that only the port and starboard thrust angles' derivatives were penalized since the bow thruster was fixed at 90°.

$$R_{\Delta n} = -\sum_{i\in\{\text{bow, port, star}\}} \frac{c_{(\Delta n),i}}{100}\left(\frac{|n_i(t) - n_i(t-1)|}{0.2s}\right). \tag{5.16}$$

$$R_{\Delta \alpha} = -\sum_{i\in\{\text{port, star}\}} \frac{c_{(\Delta \alpha),i}}{\pi}\left(\frac{|\alpha_i(t) - \alpha_i(t-1)|}{0.2s}\right). \tag{5.17}$$

Final Reward Function and Coefficients

The final reward function was the sum of all the above mentioned reward components shown in Equation (5.18), and the coefficients used for the final agent are listed in Table 5.2.

$$R_{tot} = R_\eta + R_{vel} + R_{|n|} + R_{\Delta n} + R_{\Delta\alpha} \tag{5.18}$$

Table 5.2: Final reward function coefficients.

c_{gauss}	c_{AS}	c_{const}	σ_d	σ_ψ	c_u	c_v	$c_{\tilde{\psi}}$
2.0	0.1	0.5	1.0	5.0	0.5	0.5	1.0
$c_{\lvert n\rvert,bow}$	$c_{\lvert n\rvert,port}$	$c_{\lvert n\rvert,star}$	$c_{(\Delta n),bow}$	$c_{(\Delta n),port}$	$c_{(\Delta n),star}$	$c_{(\Delta \alpha),port}$	$c_{(\Delta \alpha),star}$
0.2	0.3	0.3	0.05	0.05	0.05	0.01	0.01

5.4.5 Activation Functions

As mentioned in Section 3.2, the activation functions are necessary for representing non-linear connections between the different layers, and the standard approaches in today's state-of-the art deep learning architectures for regression problems use some version of a hyperbolic tangent, a sigmoid, or a Rectified Linear Unit (ReLU). The vast majority of the previous work found on continuous control used a ReLU or a tanh activation function [23, 25, 26, 27, 66, 67]. However, inspired by findings in [68, 69, 70], in particular "leaky ReLUs" improved learning of various Neural Network setups, consistently performing well across several learning algorithms and environments. As such, it was chosen to use this leaky ReLU, which builds on the ReLU by introducing a "leakage-rate" $\alpha_{leak} \in (0, 1)$ for the negative inputs as displayed in Equation (5.19). Note that in the special case of $\alpha_{leak} = 0.0$ the activation function is the same as a regular ReLU, or if $\alpha_{leak} = 1.0$ it represents a linear activation function (just passing the input straight through without any non-linear modifications).

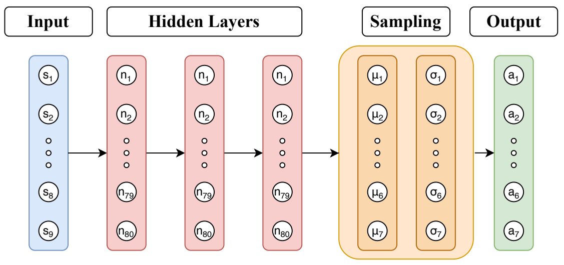
$$f_{\text{LeakyReLU}}(z, \alpha_{leak}) = \begin{cases} \alpha_{leak} z & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}. \tag{5.19}$$

Several training sessions with various layer activations were performed, in which it was found that the leaky ReLU with a fixed "leakage-rate" of $\alpha_{leak} = 0.2$ outperformed the other on both of the two network structures; both in terms of fast learning and stable convergence. The results from these training session can be seen in Appendix D. Therefore, the leaky ReLU activation function was chosen as the activation function, used across all hidden layers. The *output* layer was given a linear activation function, as the values the network was to predict included both positive and negative numbers.

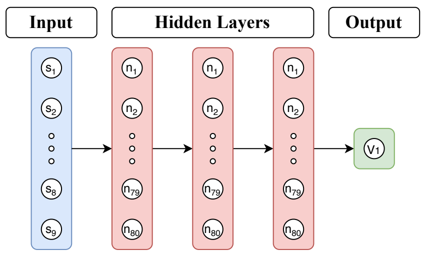
5.4.6 Actor and Critic Network Architecture

In the previous work, a variety of network widths and depths has been used, typically using between 2 and 3 hidden layers with between 5 to 400 nodes in each layer [25, 26, 27, 29, 30, 31, 32], with 2 layers of 64 nodes seemingly being a standard for continuous control problems [23, 24, 25, 26, 66]. Several training sessions were run using various network sizes, ranging from 1 to 3 hidden layers, using between 40 and 400 nodes per layer. The results of the training sessions can be found in Appendix D. It was found that using 1 layer with 40 nodes was too small: the learning did not converge fast enough. 1 layer with 400 nodes resulted in very fast learning, as did 3 layers with 80 nodes. Using the validation (Section 5.4.11) as the judge, it was found that, even though 1 layer with

400 nodes learned fast, the resulting behavior was quite oscillating. The larger networks tended to give smoother output signals, and thus a calmer behavior of the vessel. That was the reason for selecting 3 layers using 80 nodes. For visualization, the final network structures were as shown in Figure 5.7a and Figure 5.7b.



(a) Actor network architecture.



(b) Critic network architecture.

5.4.7 Optimizer and Learning Rates

In order to carry out gradient descent in the back propagation in order to calculate how to adjust the weights during the training process, using normal stochastic gradient descent could be quite slow, and it uses no information about the slope in order to dynamically adjust the learning rate. Using other optimizers have shown to improve performance, and the chosen one is one which is used a lot in the academic research, namely Adam, introduced by Kingma and Ba [50], which uses the notion of momentum to adjust the learning rate as the training is carried out. By selecting an optimizer which utilizes momentum to make the step size adaptive throughout training, the learning process becomes less sensitive to the initial selection of the learning rate. Kingma and Ba [50] also showed that Adam was applicable within a wide range of initial parameters. The default learning rates in the Spinning Up implementation was found to suffice in combination with the Adam optimizer from Tensorflow. These parameters are summarized in Table 5.4. Note that the critic and actor networks ended up using different initial learning rates, but both used the Adam optimizer.

5.4.8 PPO Algorithm

The implementation of the PPO algorithm was done in Python, and was based on the Spinning Up repository [41]. The repository was made by research scientists at OpenAI, lead by Joshua Achiam, serving as an educational tool for the methods used in their research within Deep Reinforcement Learning. Two implementations details can be noted from the pseudocode presented in Algorithm 4: an estimate of the KL-divergence (a metric for how similar two probability distributions are) was added for aiding the hindrance of too large steps on the network parameters, and the advantages in each batch were standardized (meaning that the advantages were standardized using the mean and standard deviation of each batch of data) which in practice leads to more stable training due to the neural networks' sensitivity to scale, and theoretically it contributes to less variance [55, 71]. The hyperparameters of the algorithm had to be tuned, but some of the default values presented in the Spinning Up repository were used. The estimate of the KL-divergence and the policy clipping ratio ϵ was found to be successful by using the defaults, as changes in the clipping ratio slowed down the training, while changing the KL-divergence bound was rather destabilizing for the training process.

Discount Factor and Decay Rate

The effect of a reward obtained at time step T steps ahead of a current time step is weighted γ^T, being its highest at time step 0 as long as $0 \leq \gamma < 1.0$. An overview of the effects are done in Table 5.3, where some values of γ^t has been shown in order to help choose the discount factor.

Table 5.3: Discount factor effect γ^T vs. number of time steps T.

γ \ T	**10 steps**	**50 steps**	**100 steps**	**500 steps**	**1000 steps**
0.900	0.35	0.01	0.00	0.00	0.00
0.950	0.60	0.08	0.01	0.00	0.00
0.975	0.78	0.28	0.08	0.00	0.00
0.990	0.90	0.61	0.37	0.07	0.00
0.995	0.95	0.78	0.61	0.08	0.01
0.999	0.99	0.95	0.90	0.61	0.37

In order to determine the range of the rewards to account for, a test was done by using the existing control system (PID + the pseudoinverse thrust allocation), reacting to a setpoint change of $[6\ m, 6\ m, -45°]$. This was a combination of setpoint elements that was expected to give a difficult DP-task, since all degrees of freedom had to be controlled simultaneously, and could act as an indicator for how much time the control system needs to reduce the body-frame errors. The results of the test can be found in Appendix F. It was seen that the time taken by the control system to minimize the positional error was approximately 60 seconds. To allow the agent to explore enough, 60 seconds was set to be a lower bound for the time step horizon in which the discount factor had to make rewards valuable. In order to reach a setpoint, but also to stay there (hence performing station-keeping), the time horizon was chosen to be 80 seconds to allow for accumulation of reward while being in the setpoint.

The number of time steps was determined based on this time horizon: as the control system was set to select control signal at a frequency of 5 Hz, the time step in the simulator was set to be 0.2 s. Hence, in order to simulate each episode having a maximum length of 80 seconds, 80 $s/(0.2\ s/\text{time step}) = 400$ time steps was chosen. In order for the discount factor to be in a valid range according to Table 5.3, it was iteratively tested with values within the range of (0.95, 0.995). The desired learning behavior was observed with various values close to 0.99, so 0.99 was therefore set to be the discount factor. The fact that 0.99 was also set to be the standard discount factor in the Spinning Up implementation further supported the selection of the discount factor in this thesis. As so, the decay rate used in the Generalized Advantage Estimation was also set to the default value, as iterative testing gave bad results of testing values below the default at 0.97.

From the fact that one episode could be no longer than 400 time steps, it was iteratively chosen that the minibatch size (being the number of time steps taken between each network update) should be at least 2000. This lead to that, if no trajectories during one epoch is terminal, the parameter updates were done with 5 trajectories of size 400 time steps. One epoch represents all time steps done between each update of the networks.

Hyperparameter Summary

Table 5.4[3] shows a summary of the chosen hyperparameters for the final agent used in the tests following in Chapter 6.

Table 5.4: PPO hyperparameter selection.

Parameter	**Symbol / abbreviation**	**Value**
Kullback-Leibler divergence limit	KL	0.015
Clipping ratio	ϵ	0.2
Discount factor	γ	0.99
Decay factor	λ	0.97
Actor learning rate	α_a	3e-4
Actor network iterations	I_a	80
Critic learning rate	α_c	1e-3
Critic network iterations	I_c	80
Adam initial momentum	$[\beta_1, \beta_2, \epsilon_A]$	[0.9, 0.999, 1e-07]
Number of epochs	N	1500
Max trajectory length	T	400 time steps / 80 s
Max time steps per epoch	MT	1600

5.4.9 Performing Agent

The critic only helps in evaluating states during training, and improves the stability of the PPO algorithm. However, when the training was done, the agent purely consisted of the policy network during testing. Its performance was validated in two steps: first, a simple test was performed on the Windows computer in order to see how the agent performed on a simple setpoint-change test with perfect information (using the states directly from the simulator). This is further explained in Section 5.4.11. Second, the network was loaded onto the Ubuntu computer, and used by the control system in ROS, where it was tested with "imperfect" state estimations coming from an observer.The latter representation was used when performing the final tests in Chapter 6.

Previous work has showed that removing the noise vector *after* training helps on stabilizing the performance [26]. A final adjustment in this thesis was therefore to remove the action noise from the actor network. This means that all components of $\boldsymbol{\sigma}$ from Equation (3.14b) was set to zero, which resulted in smooth control signals without any delay. As an addition, it was chosen to use an integral effect on the state vector of the DRL method during final testing (not validation). The integral effect was added as in [28], augmenting the body-frame error pose in the state vector according to $\hat{\boldsymbol{\eta}}_t = \boldsymbol{\eta}_t + \hat{\boldsymbol{\eta}}_{ss,t}$, where the discrete integration was implemented according to $\hat{\boldsymbol{\eta}}_{ss,t} = \hat{\boldsymbol{\eta}}_{ss,t-1} + \Delta t k \boldsymbol{\eta}_t$. Δt is the time step of 0.2 s for the ROS node, and k is a coefficient deciding the speed of the integral accumulation, set to 0.05 for *all* test scenarios, and a windup was added to prevent overshoots, set at $[\tilde{x}_{windup}, \tilde{y}_{windup}, \tilde{\psi}_{windup}] = [0.5\ m, 1.0\ m, {}^{\pi}/_{32}\ rad]$.

[3]Some values are given in scientific notation using the E-notation: $a \cdot 10^b = aeb$

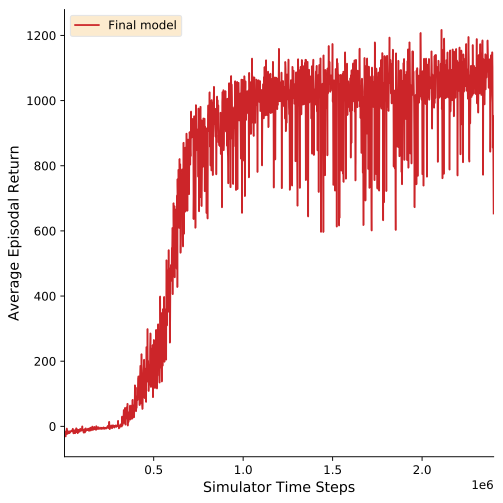
5.4.10 Training the Agent

The training of the agent was carried out in the following manner: First, the agent was reset to a random pose within the state-space bounds. The setpoint could be set to be the same point in the NED frame each episode, since the agent's body-frame error was randomly initialized. This made exploring the state space effective while using only one setpoint for all episodes. During training, the agent carried out observing and acting in the environment until it reached a terminal state, or the maximum episodal length was reached. The agent was reset every time this event occurred. The reset-method randomly selected values from the lower 80% of the maximum values for body-frame surge, sway and yaw errors, as this allowed for less terminal states early on during training, which empirically yielded faster learning. Since the purpose of this agent was to perform DP, low velocities was desirable, and thus the random initialization of velocities was from the lower 30% of their respective maximums.

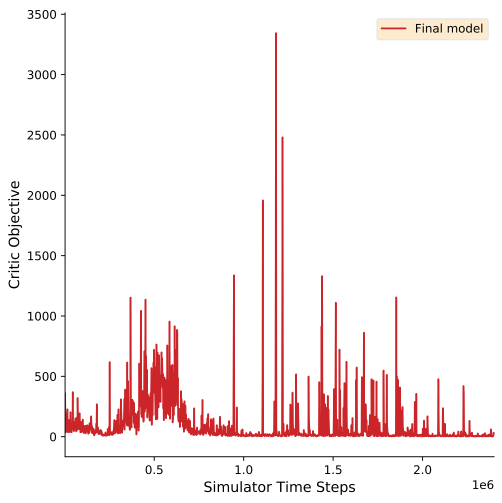
The results of the training of the final agent is shown in Figure 5.8. Note that all plots shows a running average of the 25 last epochs[(4)], and that the units of the x-axis is 10^6 time steps. Figure 5.8a shows the average return accumulated over each episode within each epoch. Since the largest instantaneous reward achievable from R_{tot} in Equation (5.18) was $r^* = 3.5$, the maximum episodal return was 1400 over the possible 400 time steps within each episode. Due to the pose being set to random values within the state space between each episode, 1400 was not achievable unless the agent was reset exactly at the setpoint, and stayed there throughout the entire episode. Therefore, this metric was expected to be noisy, but it was found that it converged to an area of good performance, validated as explained in Section 5.4.11.

The critic's objective function is shown in Figure 5.8b, which quickly converged towards low absolute values. The critic's average and maximum value function estimates within each epoch is shown in Figure 5.8c and Figure 5.8d, respectively. Note that the maximum value is $r^*/1-\gamma = 3.5/1-0.99 = 350$ (going from Equation (3.18)), which is what the maximum estimates converged towards in Figure 5.8d. The change of the actor's objective function per epoch is shown in Figure 5.8e, while an estimate of the policy's entropy (being the average value of $-\log \pi_\theta(\cdot)$ during one epoch) is shown in Figure 5.8f. These are not really expected to converge to a predetermined value, but an eye was kept on the improvement of the PG objective (thereby improving the policy's performance *on the PG-objective*), and that the entropy was going towards small numbers as large values of the entropy means a large spread in the action-selection from the policy.

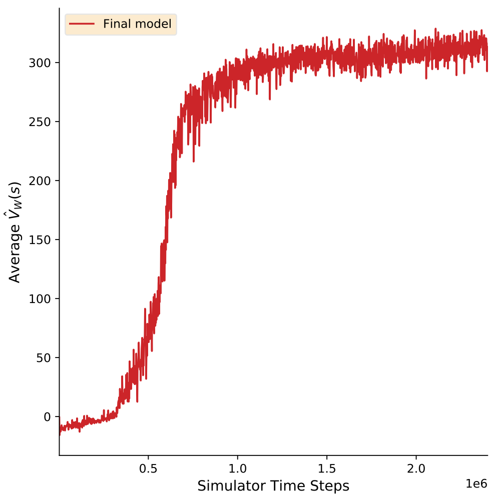
[(4)]This must not be confused with simulator time steps: one epoch consisted of MT time steps.



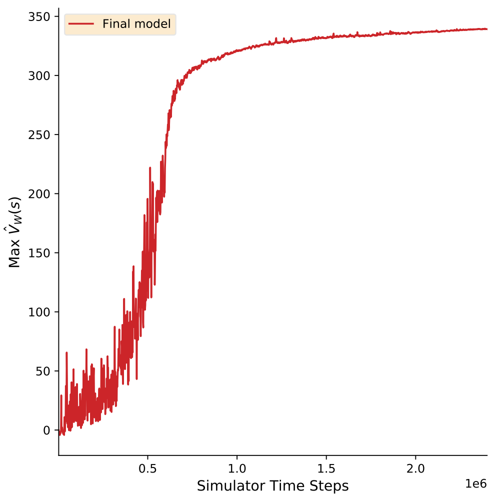
(a) Average return per episode / epoch.



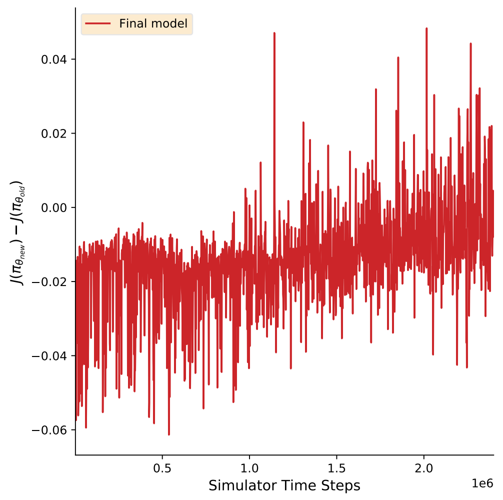
(b) Critic objective function.



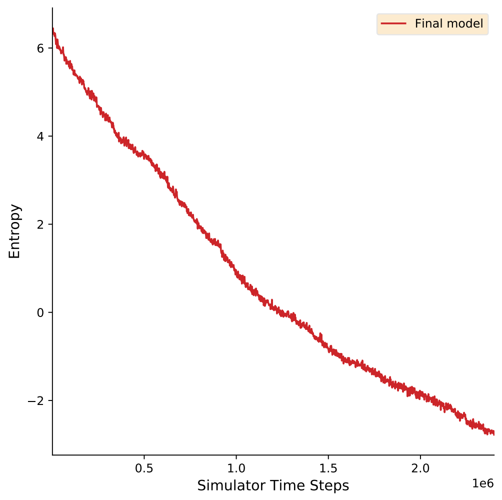
(c) Average estimated V-values.



(d) Maximum estimated V-values.



(e) Change in PG-objective of actor.



(f) Estimated entropy of actor.

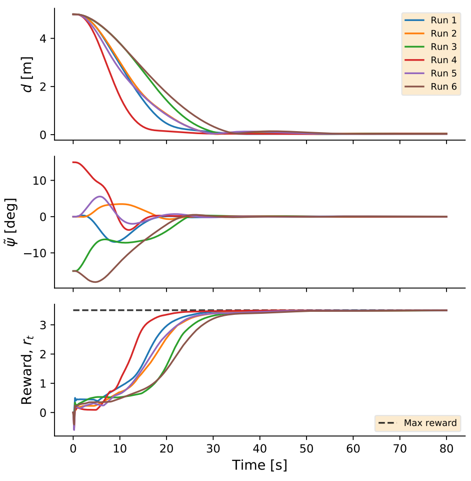
Figure 5.8: Running average of 25 epochs showing results of training the final DRL model.

5.4.11 Validating Performance

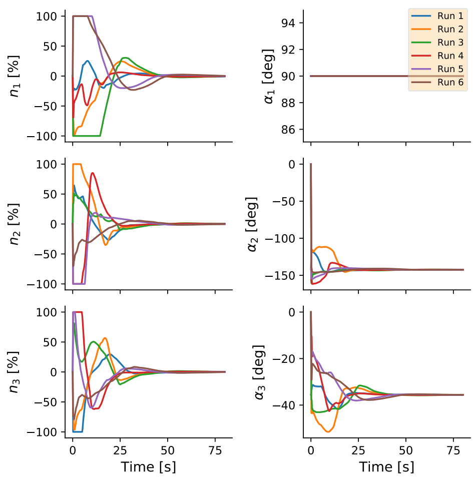
After a training session, a validation was done to test the agent's performance, starting in various initial poses. The agent was reset to 6 different poses, all with the target of reaching the setpoint at $[0\ m, 0\ m, 0°]$. Each of the sessions was set to last 80 seconds, and the coordinates are showed in Table 5.5. The validation sessions run with the final model is shown in Figure 5.9. The performance was found good in terms of quick convergence to low body-frame errors and high reward, in addition to the actuator outputs being smooth and relatively small.

Table 5.5: Validation test starting coordinates.

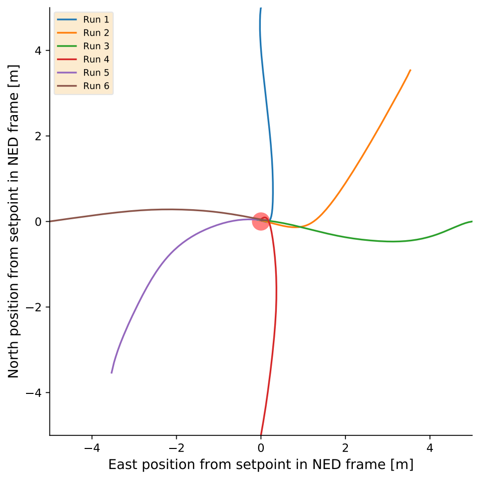
Run	**North** [m]	**East** [m]	**Yaw** [deg]
1	5.0	0.0	0.0
2	3.5	3.5	0.0
3	0.0	5.0	-15.0
4	-5.0	0.0	15.0
5	-3.5	-3.5	0.0
6	0.0	-5.0	-15.0



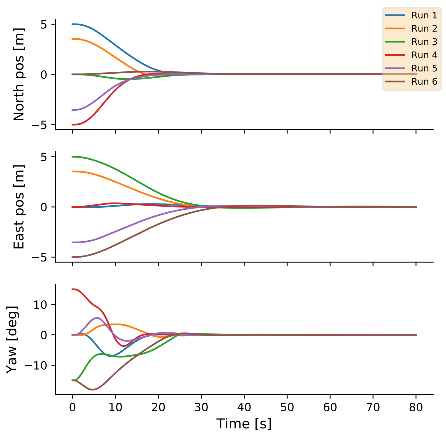
(a) d, $\tilde{\psi}$, and reward per time step.



(b) Actions.



(c) North-East plot.



(d) Pose in NED frame.

Figure 5.9: Final model validation plot.

Chapter 6

Results

This chapter explains and presents the results from various test scenarios. Positional plots are plotted in meters where the axes represents the distances away from the initial setpoint, and *yaw* is used instead of *heading* since the plots shows the orientation of the vessel relative to the initial heading. All angles are plotted in the region $(-180°, 180°]$. The plots containing scenarios with setpoint changes have alternating white and gray backgrounds to separate between the setpoints. The controllers tested was the existing motion controller + IPI thrust allocation (referred to as IPI in the plots), the existing motion controller + QP allocation (referred to as QP), DRL control (referred to as RL), and DRL control with integral effect as explained in Section 5.4.9 (referred to as RLI).

First, Section 6.1 explains the test scenarios the performance metrics used for quantitative analysis. Then, individual sections displays the results from each scenario, including a short comment on what can be seen in the plots presented. A more discussion of the results, in addition to an assessment of the performance and characteristics of the different methods is found in Chapter 7.

6.1 Test Scenarios

6.1.1 Large setpoint changes

In order to test the neural network's ability to perform with input data ranges which it has not been trained on, a test was made with setpoint changes as shown in Table 6.1. Recall that the body-frame error bounds were $[\tilde{x}, \tilde{y}, \tilde{\psi}] = [8\ m,\ 8\ m,\ 45°]$.

Table 6.1: Pose of large setpoint changes, relative to initial pose.

Setpoint no.	**Time span** [s]	**North** [m]	**East** [m]	**Yaw** [deg]
0	0 - 10	0	0	0
1	10 - 80	12	0	0
2	80 - 150	12	-12	0
3	150 - 190	12	-12	90
4	190 - 270	0	0	-45
5	270 - 350	0	0	0

6.1.2 Four Corner Test

The four corner test is a common way of testing the performance of a DP control system (e.g. [33, 35, 72]), where a selection of setpoints are to be followed in order to test the ability of the control system to control separate degrees of freedom individually, in addition to testing coupled motions. On ReVolt, a four corner program was written, which changes setpoints based on a time scale. An overview of the setpoints is shown in Table 6.2.

Table 6.2: Poses for regular four corner test, relative to initial pose.

Setpoint no.	**Time span** [s]	**North** [m]	**East** [m]	**Yaw** [deg]
0	0 - 10	0	0	0
1	10 - 60	5	0	0
2	60 - 110	5	-5	0
3	110 - 140	5	-5	-45
4	140 - 190	0	-5	-45
5	190 - 250	0	0	0

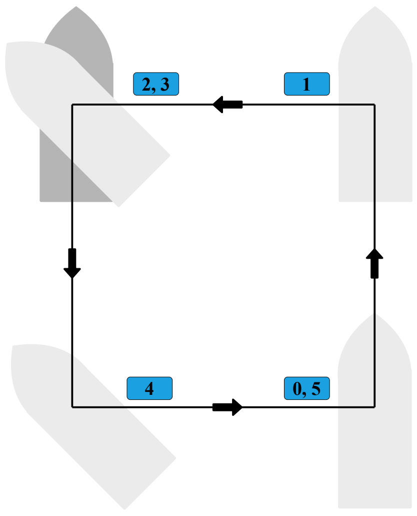


Figure 6.1: Four corner test illustration with the ReVolt model ship to scale.

6.1.3 Four Corner Test with Ocean Current

A four corner test was also performed while enabling current loads to observe the controllers' robustness to steady state deviations. The coordinates was the same as in the regular four corner test, but each positional translation was given 20 seconds extra due to the current loads acting as shown in Table 6.3. The current was irrotational and non-fluctuating (in order to keep the environmental loads and moments as constant as possible), having its velocity set to $\nu_c = 0.2\ m/s$, while the direction was set to $\beta_c = 135°$, meaning that it was traversing *from* North-West *towards* South-East. All tests were started when the vessel had been standing still for 30 seconds.

Table 6.3: Poses for four corner test with ocean current, relative to initial pose.

Setpoint no.	**Time span** [s]	**North** [m]	**East** [m]	**Yaw** [deg]
0	0 - 10	0	0	0
1	10 - 80	5	0	0
2	80 - 150	5	-5	0
3	150 - 190	5	-5	-45
4	190 - 270	0	-5	-45
5	270 - 350	0	0	0

6.1.4 Station-keeping Capability with Environmental Loads

A test was constructed according to DNVGL's standard "DNVGL-ST-0111 - Assessment of station keeping capability of dynamic positioning vessels" [73] for evaluating station-keeping capabilities while being subject to environmental loads from various angles. The test was run with an increment of 22.5 degrees, with the wind, current, and waves acting in the same direction at each increment. The environmental loads' parameters were set according to Table 6.4, including slowly varying wave drift forces and fluctuating wind and current speeds. Before each test was recorded, the new environmental loads (after changing their angles) was acting for 5 minutes in order for the sea state to properly build up, and the integral parts of the controllers to stabilize, before the station-keeping was to commence.

Table 6.4: Station-keeping capability test parameters.

Wind speed [m/s]	**Current speed** [m/s]	**Significant wave height** [m]	**Increments** [deg]
2.0 ± 0.3	0.1 ± 0.05	1.0	22.5

6.1.5 Performance Metrics

To evaluate the performance of the different methods, metrics were chosen in order to qualitatively evaluate the both positional accuracy, efficiency, and wear and tear.

Positional Accuracy

The Integral Absolute Error (IAE) measurement was used as a measurement of how accurate the DP system is to reduce the error between the current pose and the desired pose coming from the reference filter. The normalized measure used was $\bar{\eta}$ as shown in Equation (6.1), where the normalizing parameters were used for making the IAE a dimensionless number, set to $[x_b,\ y_b,\ \psi_b] = [5\ m,\ 5\ m,\ 25°]$, meaning that a 1 meter deviation was weighted equally as a 5 degree deviation from their desired values.

$$\bar{\eta} = \left[\frac{\tilde{x}}{x_b}, \frac{\tilde{y}}{y_b}, \frac{\tilde{\psi}}{\psi_b}\right]^\top . \tag{6.1}$$

$$IAE(t) = \int_{\tau=0}^{t} \sqrt{\Big(\bar{\boldsymbol{\eta}}(\tau) - \bar{\boldsymbol{\eta}}_{\boldsymbol{d}}(\tau)\Big)^\top \Big(\bar{\boldsymbol{\eta}}(\tau) - \bar{\boldsymbol{\eta}}_{\boldsymbol{d}}(\tau)\Big)}\ d\tau. \tag{6.2}$$

Energy Usage

The energy usage was calculated by integrating the power for each of the commanded RPS-signals. The power as a function of RPS, n, is given as shown under the integral in Equation (6.3). The water density ρ was set to 1025 kg/m^3 and the diameters D were as shown in Table 4.1. The thruster coefficients were found from the towing tank results, and were approximated to being constant over the entire tests due to the low advance speeds in DP-operations: $K_{Q,bow} = 0.02$ and $K_{Q,port} = K_{Q,star} = 0.036$. This approximation was done because the performance metric was only used for a comparison between methods, and not as an absolute measurement of energy usage. Therefore, the energy usage is presented as energy *equivalents*, being denoted W^*.

$$W^*(t) = \int_{\tau=0}^{t} \Big(2\pi\rho K_Q D^5 \operatorname{sgn}(n(\tau))n(\tau)^3\Big) d\tau. \tag{6.3}$$

Wear and Tear

The Integral of Absolute Differentiated Control (IADC) metric was adapted from Eriksen and Breivik [74], and assesses the wear and tear by integrating the magnitude of the change of commanded bow, port and starboard (b,p,s) outputs. The thrusters' RPS derivatives were calculated in the same way as the thruster outputs' derivatives in Equation (5.16) and the angles' derivatives as in Equation (5.17), normalized with a factor of 100 and π, respectively.

$$IADC(t) = \int_{\tau=0}^{t} \sum_{i\in\{b,p,s\}} \left(|\frac{\dot{n(\tau)}_i}{100}| + |\frac{\dot{\alpha(\tau)}_i}{\pi}| \right) \delta\tau. \tag{6.4}$$

6.1.6 Execution Notes

Some notes about how all tests were executed follows for clarity.

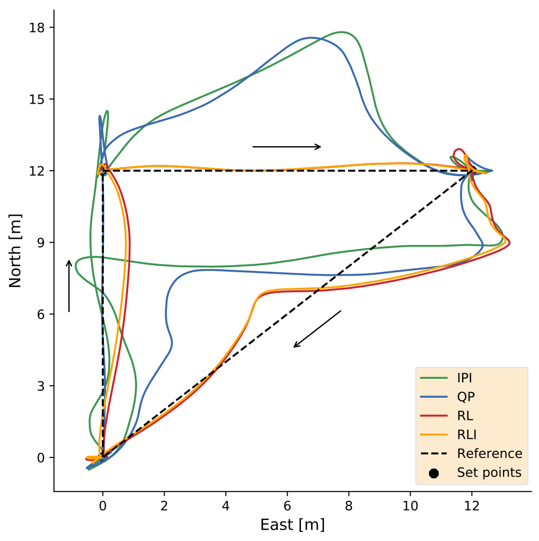
- All the values used for the energy usage and the IADC metrics were the *commanded* outputs since the physical model has no measurement / feedback of the actual propellers' RPS or angles, and the metrics were only used for comparisons between methods.

- All tests were carried out in the following sequence. Initially, the control system was set into "DP mode". On the software of ReVolt, this sets the initial setpoint in the NED frame which acts as the reference pose for the DP software. Then, the tests were started by a separate command, which was at a later point in time than the switch to "DP mode". This was done since the vessels' pose and the DP methods' integral effect had to stabilize before starting the tests. This meant that, for some of the tests where environmental loads were included, it was difficult to start the tests exactly in the pose which was set as the initial pose. Thus, some of the plots from tests including environmental loads might include minor deviations between the vessel's initial pose and the first setpoint's pose.

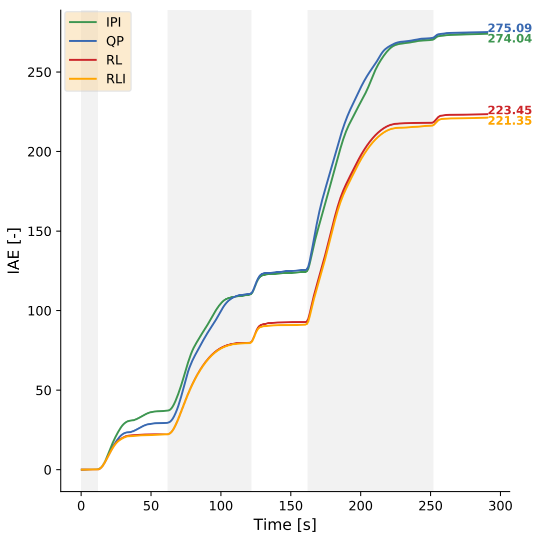
6.2 Large Setpoint Changes

6.2.1 Results

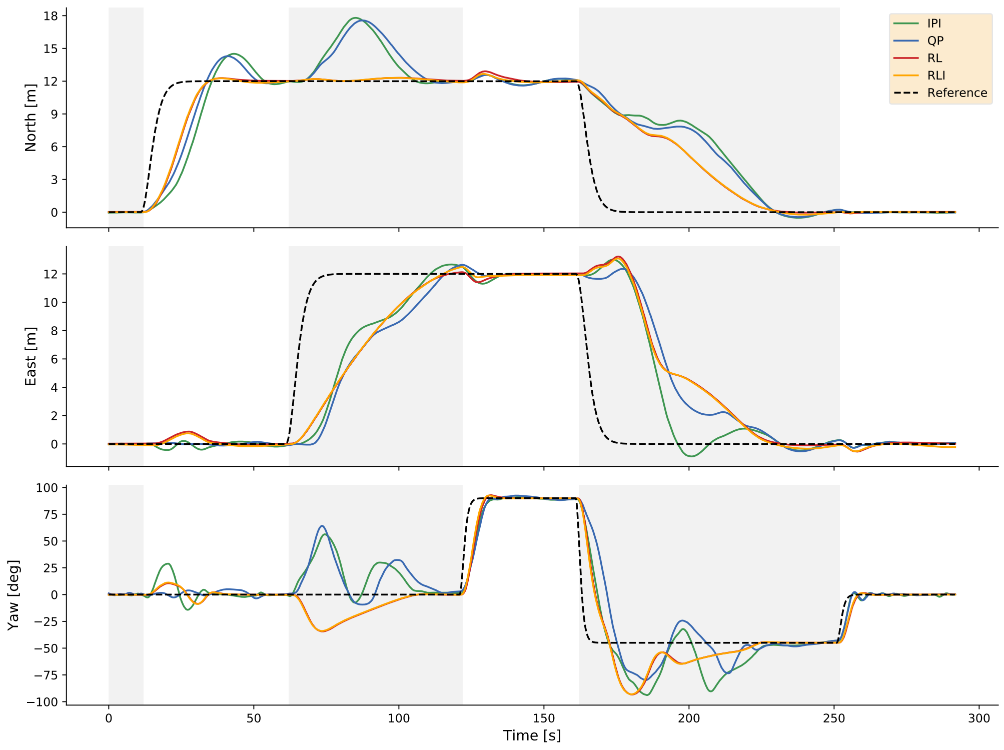
This section shows the resulting plots of the pose over time for the test using large changes of setpoints, where the position in the NED frame is shown in Figure 6.2a, the IAE over time is shown in Figure 6.2b, and the measured pose versus the reference pose is shown in Figure 6.2c.



(a) North-East plot.



(b) IAE.



(c) Pose vs. reference signal.

Figure 6.2: Resulting pose and IAE from large setpoint change test.

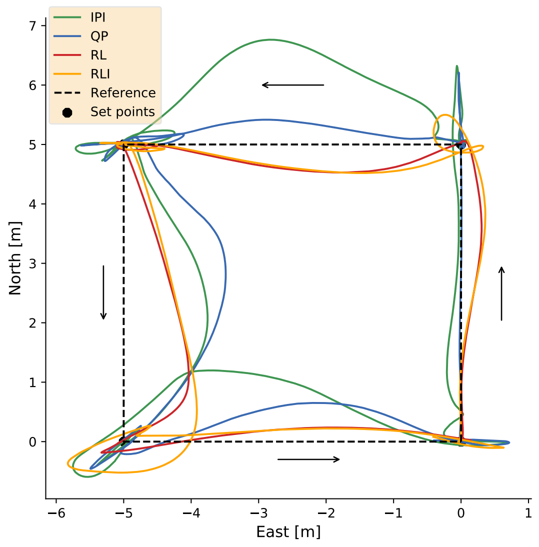
6.2.2 Comment

From the plots of the pose in Figure 6.2c, it was observed that the classic methods (being the motion controller with the IPI and QP thrust allocation methods) reached all setpoints, but with an overshoot in all respective degrees of freedom. The DRL methods both with and without integral effect reached all setpoints with no significant overshoot with the exception of the yaw at setpoint no. 4 (activated at 190 seconds). As can also be observed from the interval from setpoint no. 4 to 5 was that all methods were rather inefficient in reducing deviations in position (North and East), having a big impact on the IAE during the same interval. During the initial phases of all setpoint changes, all methods can be seen to deviate significantly in yaw, before reducing the yaw-deviation rather quickly. While the DRL-methods reduced deviations of the body-frame errors in a stable fashion, the existing methods seems to oscillate more, especially in yaw. Figure 6.2b showed that the DRL methods resulted in a mean IAE-value being 19% lower than the mean of the classic methods' IAE.

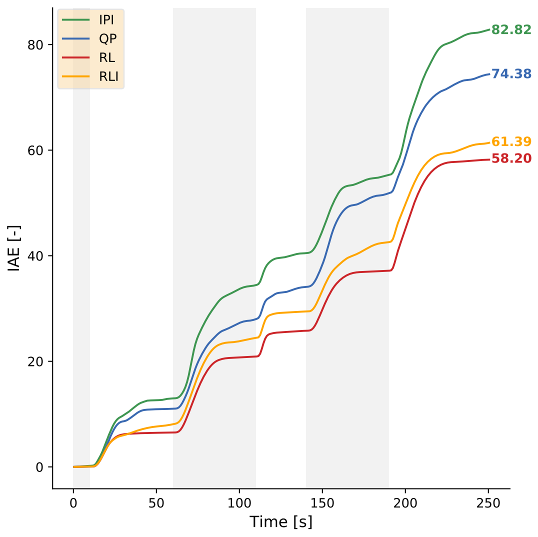
6.3 Four Corner Test

6.3.1 Results

The results of the four corner test with no environmental loads is shown in the following. The position in the NED frame and the IAE over time is shown in Figure 6.3, while the pose versus reference over time is shown in Figure 6.4. The resulting plots of commanded RPS and angles for the different methods are shown in Figure 6.5, while the wear and tear through the IADC metric, and the energy usage, is shown in Figure 6.6.



(a) North-East plot.



(b) IAE.

Figure 6.3: Resulting position and IAE from four corner test in still water.

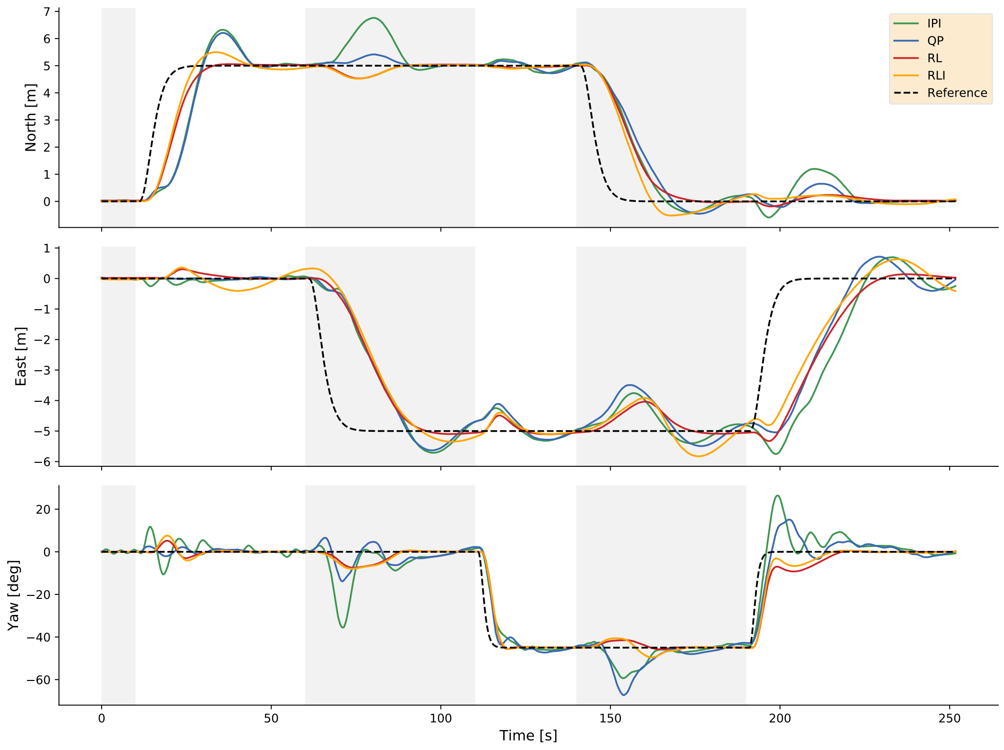
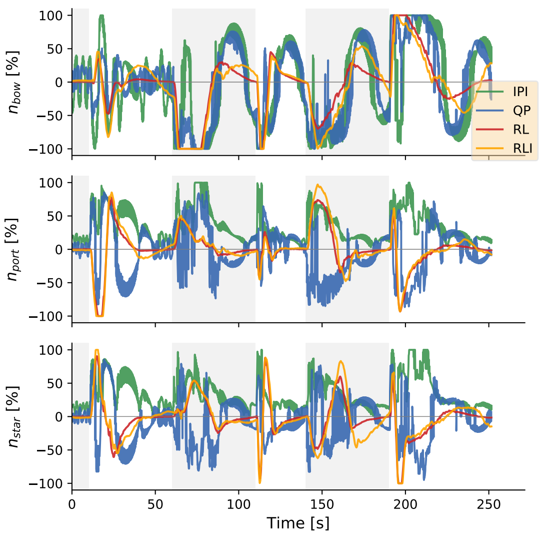
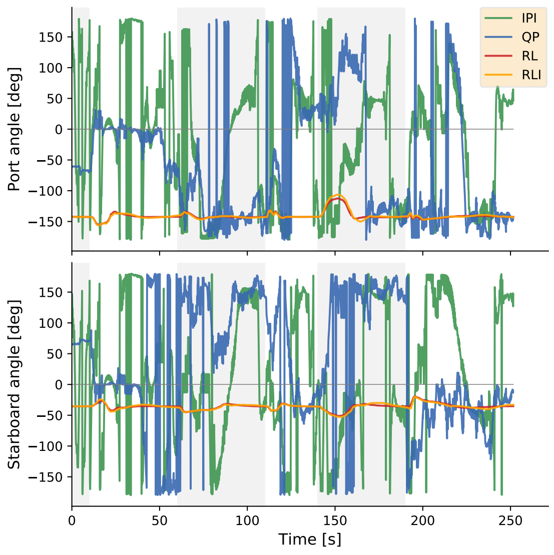


Figure 6.4: Pose vs. reference signal from four corner test in still water.

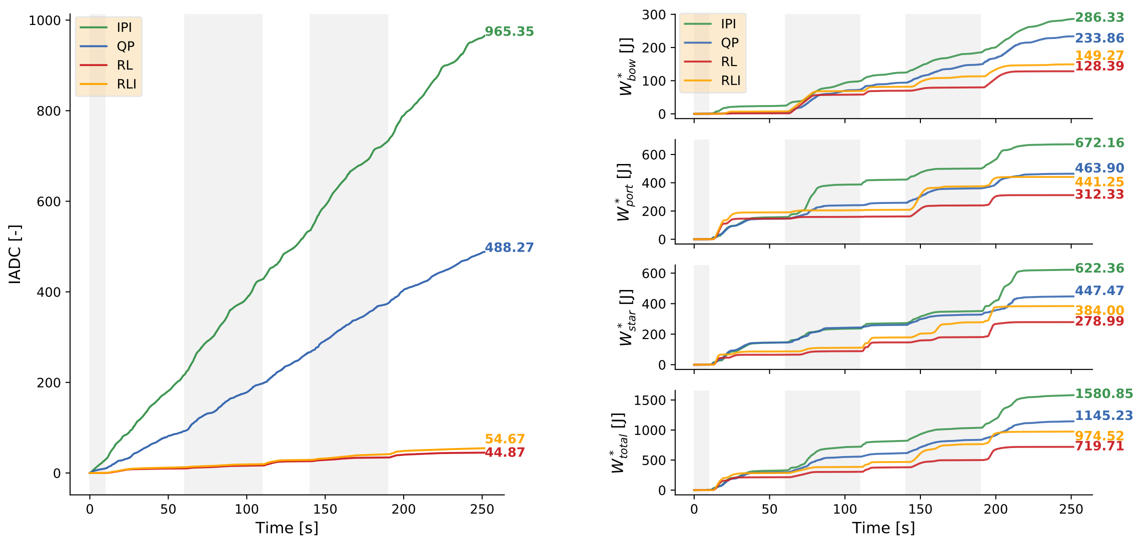


(a) Commanded RPS in [-100%, 100%].



(b) Commanded angles in $(-180^\circ, 180^\circ]$.

Figure 6.5: Resulting commanded RPS and angles from four corner test in still water.



(a) IADC. **(b)** Energy equivalents.

Figure 6.6: Resulting commanded IADC and energy usage from four corner test in still water.

6.3.2 Comments

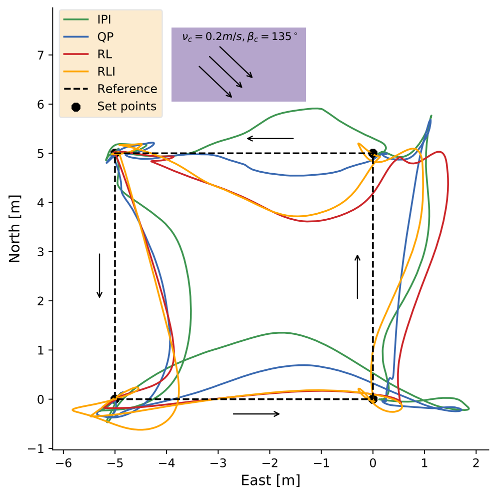
From the plots of position and pose in Figure 6.3 and Figure 6.4, it was observed that all methods were able to reach the setpoints without much steady state deviations. As for overshoots, the existing methods had an overshoot before reaching the setpoints. The DRL method with no integral effect had no significant overshoot in any of the setpoint changes, while the DRL method with integral effect did have small overshoots. In terms of oscillations, the existing methods oscillated (particularly in yaw) before settling on a setpoint, while this was not as noticeable for the DRL methods. The mean resulting IAE for the DRL methods was approximately 24% less compared to the existing methods' mean resulting IAE.

The DRL method with no integral effect commanded large RPS signals early during transitions between setpoints, before easing down on the commanded RPS pretty rapidly. The DRL method with integral effects looked to command quite similar RPS-signals, only with a slight delay when compared to the method without integral effect. Both DRL methods commanded smooth RPS and angles, while the commanded RPS-signals from the IPI and the QP allocation methods was seen to fluctuate between the rate-constraints, and looked quite similar to each other over certain time intervals. The same could be seen from the commanded angles, although the wrapping of the angles in $(-180°, 180°]$ might make it look like the commanded angles were fluctuating a lot,even though the angular rate constraints were being satisfied. The DRL methods kept the commanded angles at a fairly constant value of around $-145°$ for the port thruster and $-45°$ for the starboard thruster, only deviating immediately after each setpoints change occurred. The DRL methods having lower resulting IADC-value compared to IPI and QP. The DRL method without integral effect ended up with an energy usage approximately 37% lower than that of the QP allocation, and 55% less than that of the IPI allocation, while the DRL method *with* integral effect used approximately 15% and 38% less energy than the QP and IPI, respectively.

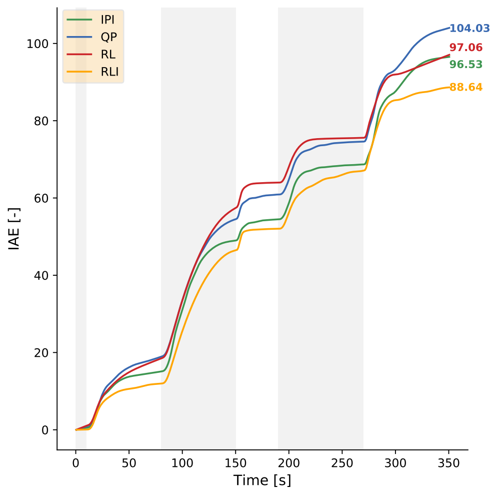
6.4 Four Corner Test with Ocean Current

6.4.1 Results

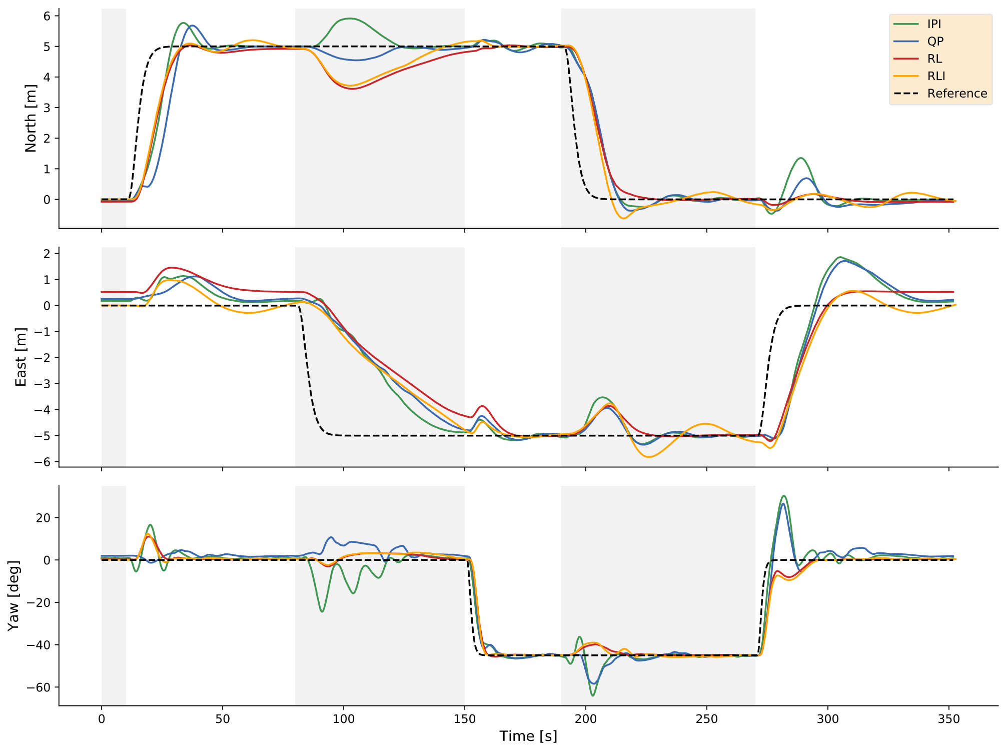
The results of the four corner test with a constant ocean current acting is shown in the following. Both the position in the NED frame, the IAE over time, and the pose vs. reference over time is shown in Figure 6.7. The resulting plots of commanded RPS, angles, IADC and energy usage for the different methods are shown in Figure 6.8.



(a) North-East plot.

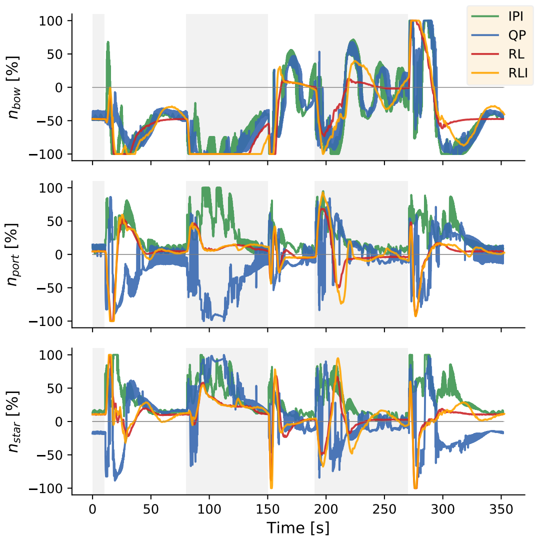


(b) IAE.

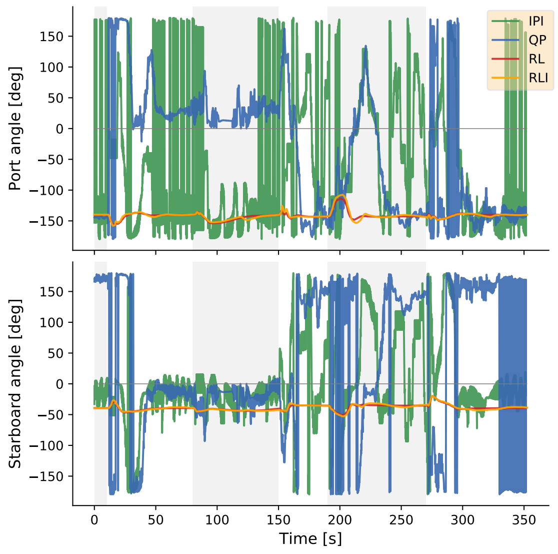


(c) Pose vs. reference signal.

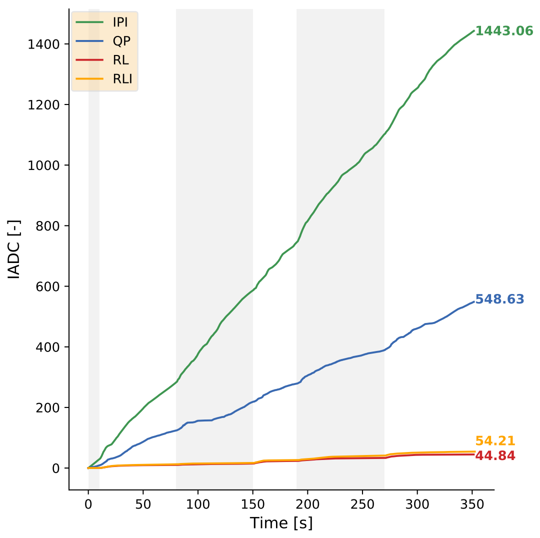
Figure 6.7: Resulting pose and IAE from four corner test with current loads.



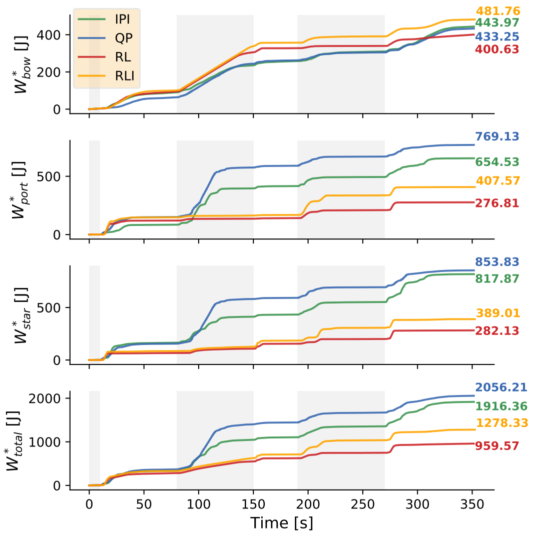
(a) Commanded RPS in [-100%, 100%].



(b) Commanded angles in $(-180°, 180°]$.



(c) IADC.



(d) Energy equivalents.

Figure 6.8: Resulting commanded RPS, commanded angles, IADC, and energy usage from four corner test with current loads.

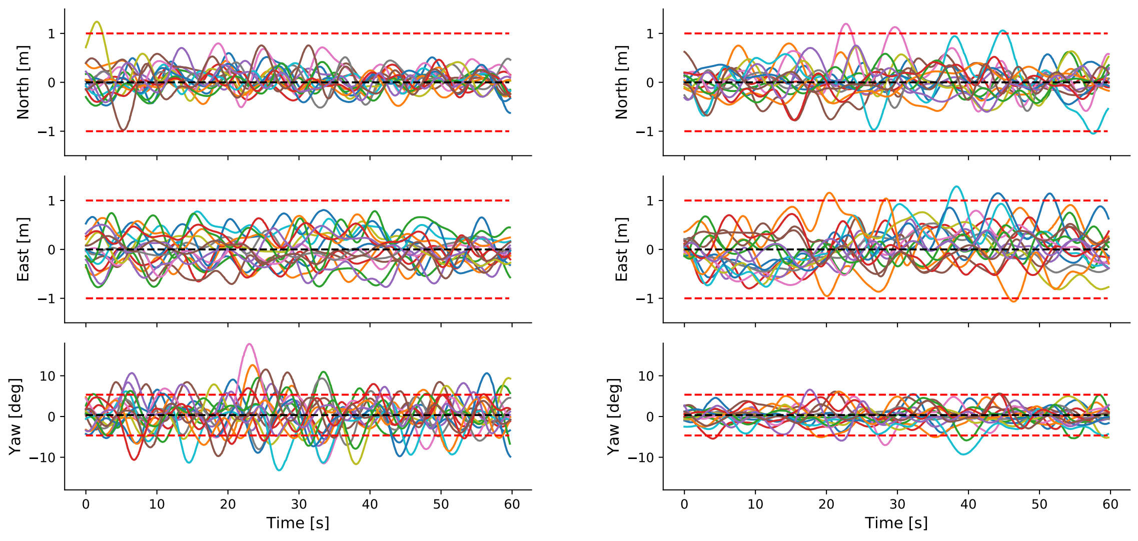
6.4.2 Comments

All methods displayed difficulties with keeping position when subject to the ocean current, which can be seen from e.g. that some methods started the test already with an offset relative to the initial setpoint. Only the DRL method with integral effect was able to totally remove the steady state body-frame error in all setpoints. It did display oscillations before managing to do so, in addition to some overshoots. The DRL method without integral effect was not able to remove steady state errors in the sway direction, making it unsuitable for comparisons on other metrics. The QP method resulted in the highest values both in terms of IAE and energy usage, while the DRL method with integral effect resulted in the lowest values for the IAE and the energy usage both. The latter resulted in 15% lower IAE and 38% less energy usage compared to the QP allocation, while having 8% lower IAE and 33% less energy usage compared to the IPI allocation.

6.5 Station-keeping Capability with Environmental Loads

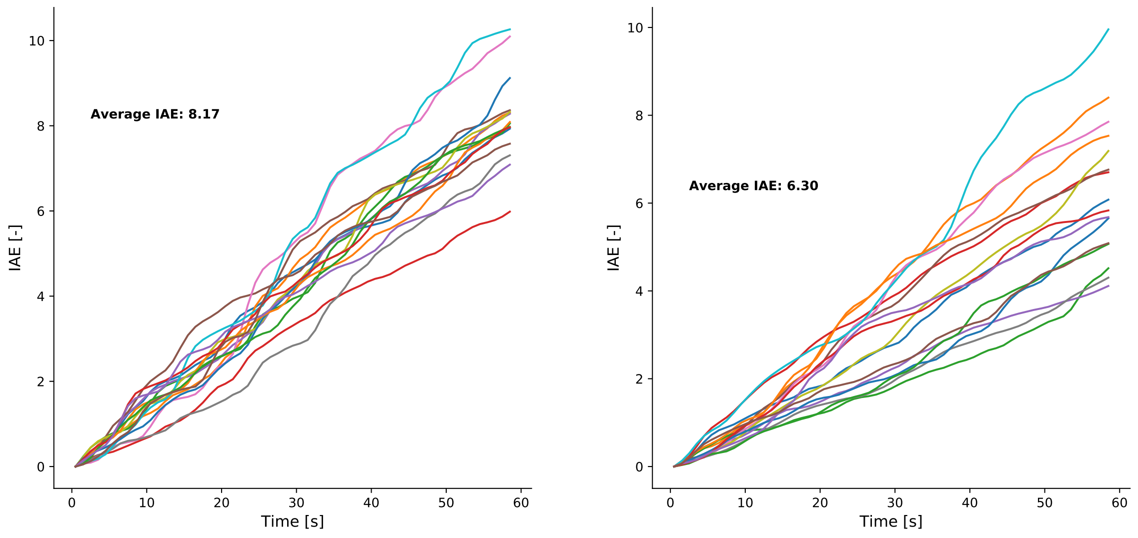
6.5.1 Results

This test was performed using the existing motion control law + the IPI allocation, and the DRL method using integral effect. Figure 6.9 shows the pose of the vessel during the test for both methods. The black lines shows the setpoint, while the red dashed lines shows the boundary of 1 m deviation in North and East, and 5° in yaw which were set inspired by the DNVGL standard [73]. Figure 6.10 shows the resulting IAE for each of the runs, including the average IAE.



(a) Motion control + IPI allocation. **(b)** DRL method with integral effect.

Figure 6.9: Pose from all environment angles during the station-keeping test.



(a) Motion control + IPI allocation. **(b)** DRL method with integral effect.

Figure 6.10: IAE from all runs during station-keeping test.

6.5.2 Comments

From the plots of the pose over time in Figure 6.9, it was observed that the boundaries were violated by the DRL method a few times during all 16 runs. The IPI allocation was for the most part well within the bounds in terms of translation, but violated the boundaries for yaw quite often, and with a larger magnitude than that of the DRL method. The average IAE was 30% larger for the motion controller + IPI allocation compared to the DRL method.

6.6 Sea Trial

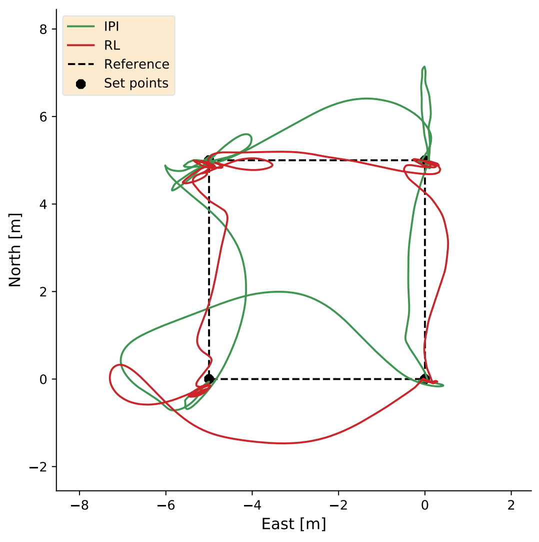
6.6.1 Notes

The sea trial was carried out with the physical model ship of ReVolt on June 23, 2020. First, a measurement was taken of the time it took the neural network to perform a feedforward pass on the computer on board the ship model. The maximum time recorded was 0.015 seconds, and the average computation time per forward pass was 0.011 seconds.

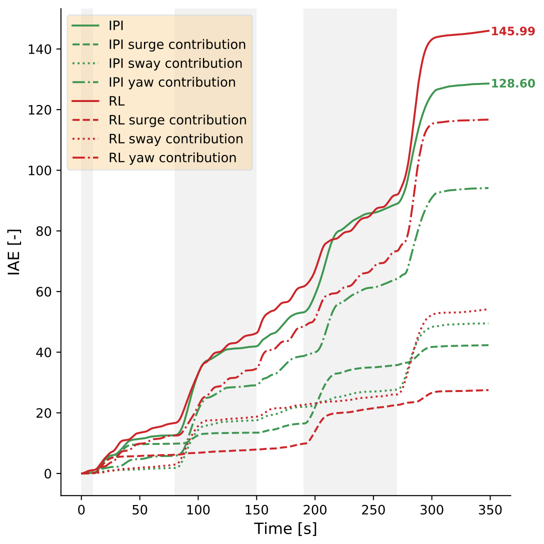
Initial tests were done with a remote controller to ensure that all hardware components was working as expected. It was found that the bow thruster's propeller only rotated when being commanded signals at 50% or above, seemingly due to resistance put on the propeller by the propeller housing. Commands lower than 50% yielded no response at all, but commands above 50% resulted in the propeller rotating at the commanded RPS value. Thus, no commands yielded a slow propeller rotation. A solution for this was not found. This meant the deviation in yaw had to become large before the DP algorithms finally outputted commanded signals above 50%. In order to carry out a test, it was decided to multiply the commanded bow thrust signals by 2.5 to get the signals large enough for the bow thruster to react more often than not. This came at the cost of an expected overshoot and/or oscillation in sway and yaw due to the bow thruster output being much larger than outputted by the DP methods. The results which are plotted in the following therefore shows 2.5 times the magnitude of the commanded bow RPS signal, only clipped at 100%.

Due to this issue, the sea trial turned out to be a test of robustness to dysfunctional hardware, a test which had not been carried out in the simulator. Unfortunately, due to the time consumed by debugging, time for testing was limited. Therefore it was chosen to test the DRL method in it's "purest" form, meaning without any integral effect on the state vector. For comparison, the motion controller in combination with the IPI thrust allocation was chosen. The test itself was carried out by using the same setpoints as the four corner test with current loads in simulation (Table 6.3). However, at the time of the four corner tests, the sea state consisted of negligible waves, but a slight breeze in combination with a small ocean current was acting towards South, which was not identical to the simulation with ocean current. In Figure 6.11, the position in the NED frame, the IAE over time, and the pose versus reference is shown. The resulting commanded RPS, angles, IADC, and energy usage is shown in Figure 6.12.

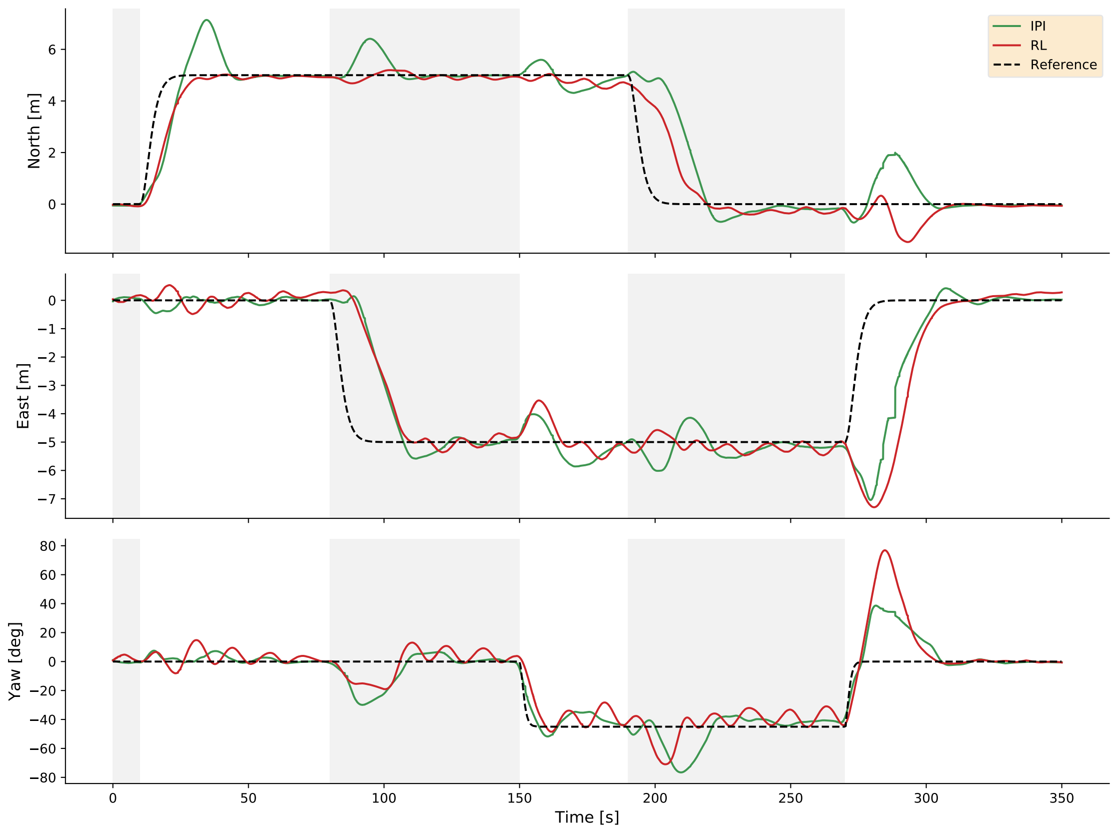
6.6.2 Results



(a) North-East plot.

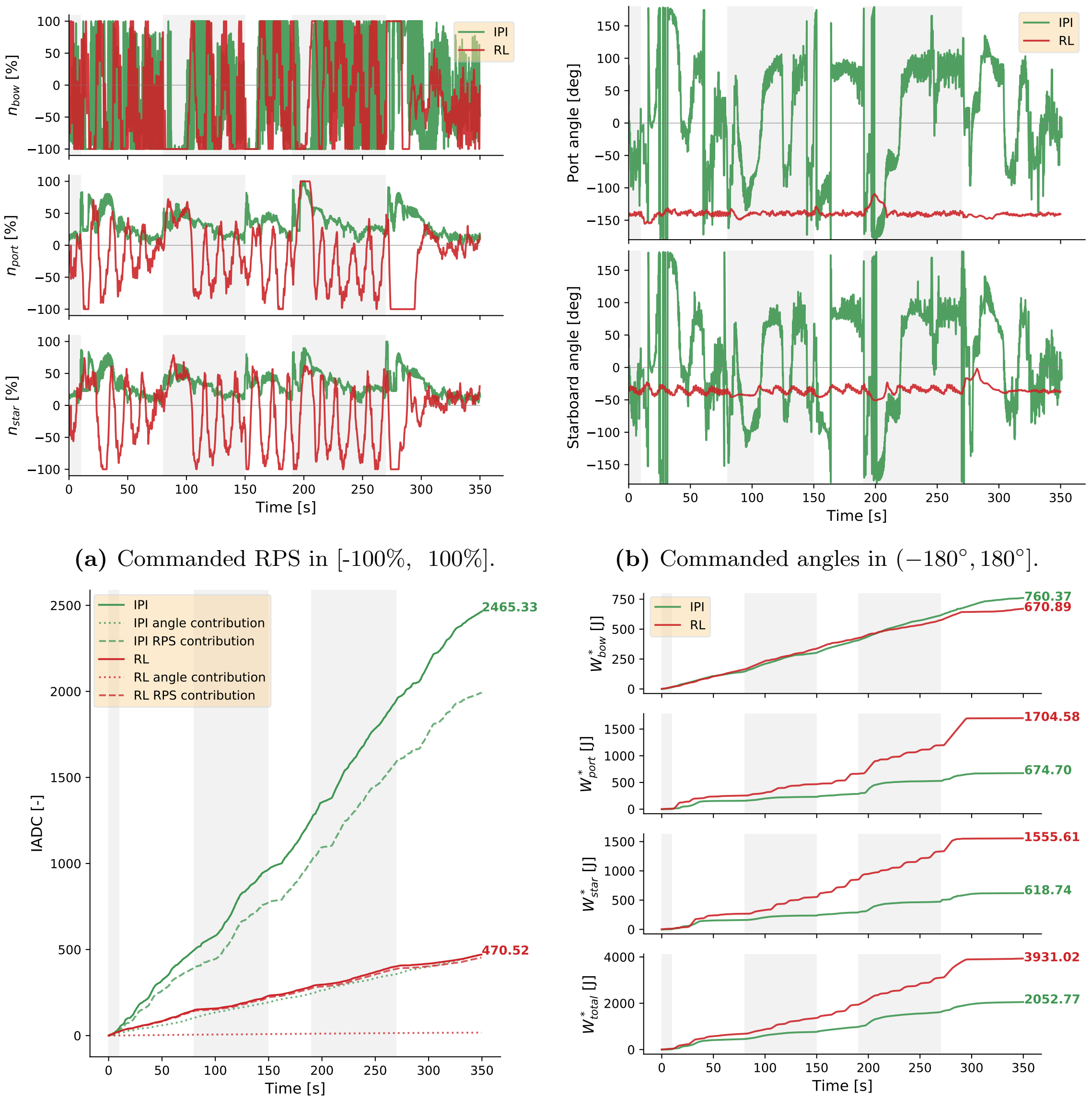


(b) IAE with contributions.



(c) Pose vs. reference signal.

Figure 6.11: Results of pose and IAE from long four corner test during sea trial.



(a) Commanded RPS in [-100%, 100%].

(b) Commanded angles in $(-180°, 180°]$.

(c) IADC with contributions.

(d) Energy equivalents.

Figure 6.12: Resulting commanded RPS, commanded angles, IADC, and energy usage from long four corner test during the sea trial. Note that the recorded bow signal is a factor of 2.5 larger in magnitude than the actual output from the different methods due to the way the hardware issue was attempted combated.

6.6.3 Comments

It was observed from Figure 6.11 that the vessel movements were more oscillatory around the reference signals than in simulation, with the DRL method being the most oscillatory. Both methods were able to get to all setpoints in approximately the same time as in the comparative simulation. The motion controller with IPI allocation resulted in an IAE 33% larger than for the same method in the comparable simulation, while the total energy usage was increased by 7%. For the DRL method, the resulting IAE was 49% larger compared to in simulation, increasing the energy usage by 310%. Compared to each other in the sea trial, the motion controller with IPI allocation resulted in 12% lower IAE and a 48% lower energy usage compared to the DRL method.

The commanded angles of the DRL method was similar to what was seen in results from simulations, but for the commanded RPS, the resulting outputs was oscillating and did not resemble the smooth output as displayed in e.g. Figure 6.8a. The thruster usage from the IPI allocation more closely resembled what had been observed during simulations, both for the angular commands and the RPS commands, but the bow thruster's commanded RPS reached saturation most of the time for both methods due to the ramped up signal used to combat the hardware issue. The stern thruster's commanded RPS was oscillating for the DRL method, but did not reach saturation as often, while the IPI allocation commanded more consistent RPS signals to the stern thrusters. For the motion controller with IPI allocation, the resulting IADC was 71% larger in the sea trial compared to in simulation, while the DRL method's IADC increased with 949%. The resulting IADC for the DRL method in the sea trial was however 424% lower than that of the IPI allocation.

Chapter 7

Discussion

This chapter discusses the results in Chapter 6 generally, and the development and performance of the DRL model specifically.

7.1 Fairness of Comparison

From all plots including results from the existing methods, it was deemed reasonable to believe that the existing implementations the motion control law (feedforward + PID feedback) could be better. This meant that the biggest drawback for the IPI and QP thrust allocation methods was that they converted signals from a motion control law that seemed to work rather poorly. Interestingly, the results from the four corner test without environmental loads did not match the results presented in the work of the master's thesis by Alfheim and Muggerud [33], nor from the best result presented in their publication a year later [75]. This indicated that later changes done to the software has reduced the performance of the motion control system, at least for DP tasks. Supporting the assumption of a poor motion controller was also the fact that the QP thrust allocation not only worked as a comparative method for the DRL method, but it was also compared against the IPI allocation. From several simulations, the trajectory of the vessel while using the motion controller and the two classic allocation methods was quite similar. This indicated that both allocation methods translated the desired forces from the motion controller similarly, but since the desired forces was calculated in a sub-optimal manner, the resulting DP capability using both classic thrust allocation methods suffered. It was first believed that one reason for this could be that the motion controller had been tuned on board the physical ship model and not the digital twin, but during the sea trial, the vessel movements with the existing motion controller and the IPI allocation was quite similar to those in simulations despite the hardware issues, supporting that the controller's DP ability had been reduced from previous work. Therefore, it can be discussed if the methods used as comparison to the DRL method is particularly useful. At least it forms a benchmark against methods which were in general able to reach setpoints, but they did not do so in a particularly accurate nor energy efficient manner in general.

It should also be noted that to use the four corner test with ocean current as a direct comparative scenario to the sea trial could be inaccurate. However, there was external loads present in the sea trial pushing the vessel southwards, so the simulation scenario still served as a useful benchmark to test performance between simulations and real life.

7.2 Test Results

7.2.1 Positional Accuracy

When discussing positional accuracy, it is referred to the controllers' ability to eliminate the body-frame errors. The NED plots can be deceiving since they do not inform about the time scale, but as the pose plots showed, the perceived errors in the NED frame pertained over a short amount of time, which was true for all methods. The short spikes of deviation from the reference signal could possibly have been reduced by using a slower reference filter. When evaluating using the IAE metric from the simulation results, it was seen from all IAE-plots that the DRL method (especially with integral effect) performed best, varying from outperforming the classic methods from $8 - 15\%$ when using environmental loads, to $19 - 55\%$ without environmental loads. It was also seen that it was doing so without much oscillations, but occasionally oscillating around the yaw angle reference, implying that the yaw angle priority could have been increased even more in the reward function. This could have been accomplished by reducing $\sigma_{\tilde{\psi}}$ in the multivariate Gaussian to reward a smaller range of yaw angle deviations, making the anti-sparsity distance metric ϕ steeper in the direction of $\tilde{\psi}$, or a combination of both.

From the sea trial, the positional accuracy of both the DRL method and the motion controller with IPI allocation was considerably reduced. This was expected due to both the hardware issues, and as can be seen in Figure 6.11b, especially the deviations in yaw contributed largely to the increase of the total IAE. What was also noticeable from the positional plots was that the DRL method resulted in a more oscillatory behavior in sway and yaw than the IPI method, and when inspecting the commanded bow thruster signals in Figure 6.12a, it was believed that this was mainly due to larger oscillations in commanded RPS, which again the DRL method was attempting to compensate for by commanding larger RPS values for the stern thrusters. This was observed physically during the sea trial as well since the small ship model is sensitive to actuator usage with respect to the roll motions, as the vessel rolled more during the test with the DRL method. This effect could also be observed from the positional plots when moving between setpoints, as the roll motions stopped when the vessels' velocity increased when moving between setpoints, also resulting in less oscillations in all degrees of freedom.

In addition to a more oscillatory behavior in general in sway and yaw, the bow thruster issue caused larger overshoots when switching between setpoints which included sway and yaw movements. However, this was more noticeable when the vessel was moving in positive sway (towards starboard) and yaw directions (clockwise). This brought attention to the unsymmetrical nature of the bow thruster since, as previously mentioned, it yielded larger forces when being commanded positive RPS signals, resulting in a larger force towards starboard than port when locking the bow thruster to $90°$ and a larger moment clockwise compared to counter clockwise. This was especially noticeable when moving between the last two setpoints as seen in Figure 6.11, as both methods lost a significant amount of accuracy in East and heading in addition to North since the surge, sway and yaw motions did not exactly align with the NED-frame as the vessel went from a heading of $45°$ towards $0°$.

The DRL method seemed to suffer the most from the bow thruster issue since it had taught itself to use the bow thruster more actively than the stern thrusters during simulations, and thus the ramped up bow thruster's RPS signal cause both more oscillations and larger overshoots than with the classic method. The classic method used also used the bow thruster extensively, however not with oscillatory and spiking commands since it included restrictions on the thruster's RPS rate. This could have caused the IPI allocation to calm down its bow command as the vessel had already started to close in on the setpoint, avoiding as large overshoots. Interestingly, the DRL method was more accurate in surge (as seen from the IAE plot in Figure 6.11b), which was not as influenced by the bow thruster usage, which indicates that its accuracy could have been considerably improved if solving the bow thruster issue.

7.2.2 Energy Usage

The DRL method consistently proved low energy usage in simulation. Looking at the commanded RPS signals from the two four corner tests (Figure 6.5a and Figure 6.8a), the DRL methods outputted instantaneous large RPS signals immediately after setpoint changes occurred, and reduced the magnitude of the commands rapidly after the vessel had started to move. The existing motion control law with the IPI and QP allocation yielded consistently larger RPS signals until the vessel was at rest at the setpoints. The DRL method with integral effect ended up using between 15%-38% less energy than the existing methods in the four corner test without environmental loads, and 33%-38% less energy than the existing methods in the four corner test *with* environmental loads. The DRL method without integral effect was even more energy efficient, but was not able to compensate for steady state errors while using environmental loads.

The results from the sea trial showed a large increase in the energy usage for both the DRL method and the motion controller with IPI allocation. Since the IPI allocation constrains the thrusters' force production rates, it was able to obtain an almost identical energy usage when compared to the comparative simulation scenario, where the increase mostly came from the increase in command bow thruster usage. The DRL method on the other hand increased its energy usage by a factor of 3.1, also being affected by the ramped up bow thruster's commanded RPS, but the stern thruster usage was the larger influence, as they had to compensate from the oscillations caused by the bow thruster. This indicated that the DRL method learned to achieve energy efficiency in simulations by using the bow thruster more extensively than the stern thrusters, which in fact resulted in a lower energy usage for the bow thruster itself when compared to the IPI allocation during the sea trial, but it suffered largely when compensating for the bow thruster hardware issue with the stern thrusters.

If inspecting the plots of the pose in Figure 6.11c and the commanded RPS in Figure 6.12a, it can be observed that the DRL methods' commanded RPS for all thrusters oscillates between positive and negative values with the same periodicity as the positional oscillations. Taking the interval of moving between the first setpoints as an example, a close inspection of the DRL method's RPS commands in Figure 6.12a indeed showed how the stern thrusters compensated for the errors caused by the bow thruster usage. When the bow thruster received large positive RPS commands, a force was put on the vessel in the starboard direction, causing a deviation in sway (closely related to the East position as the yaw deviations were small). Following this positive bow thruster command with a few seconds, the port side thruster was given positive commanded values, which exerted a force in the port side direction of the vessel since the port thruster was rotated at approximately $-140°$. At the same instance, the starboard thruster was given positive RPS commands as well, and since this thruster was rotated at $-40°$, it also exerted a force in the port side direction of the vessel. The same behavior was observed in the opposite direction as well, which displayed how both the DRL method used the stern thrusters to compensate for the extensive bow thruster usage, resulting in a large increase of the energy usage.

Since no limits were set to how large the changes in commanded RPS could be, the outputs from the DRL method were breaching the force production rates, which can be seen in Figure 6.12a when comparing the RPS commands for the stern thrusters against the commands from the IPI allocation which maintained the rate limitations. This indicated that in further work with the DRL method could benefit from adding more explicit limitations on the method's commanded force production rates in order to make it more energy efficient. In addition, the sea trial displayed a clear drawback of not having a feedback signal for the actual RPS values, because even though the commanded RPS signals were more rapidly changing for the DRL method, there were uncertainties connected to the actual RPS of the propellers which largely decides the actual energy usage.

7.2.3 Wear and Tear

Measuring the wear and tear by using the IADC metric displayed that the DRL method by far induced less stress on the actuators than the existing methods, resulting in an IADC value of approximately 10 times less than the QP allocation in both four corner tests in simulations. It is not clear why the IPI allocation ended up with between 2-3 times as much IADC as the QP allocation, but it was believed that this could have been due to different implementation details. As the IPI allocation was delivered though a pre-compiled Java code in ROS, it was not possible by the author to inspect the implementation details of it, while the QP allocation was directly implemented by the author himself. But all nodes was set to output commanded signals at maximum 5 Hz, which should have been enough for fair comparisons. An interesting note about the IADC values is that the DRL methods ended up with approximately the same IADC values in both the four corner test with and without environmental loads, even though the test with environmental loads was longer in duration. Both the existing methods increased quite noticeably, indicating that the DRL methods' wear and tear remains consistently low in various sea states.

What was not directly included in the IADC measure as a metric for wear and tear is whether or not the rate constraints were satisfied or not for the various methods. The IPI and QP allocation were both satisfying the rate constraints both for commanded RPS and angles by explicitly adding this to their respective implementations. Comparing the DRL method to these commanded signals, it was seen that the DRL method occasionally looked to be violating the rate constraints. Note that the only explicit restriction put on the RPS output from the DRL methods was to constrain the output in the range $n_i \in [-100\%, 100\%]$, so no formulation was done in terms of ensuring that the rate constraints were satisfied. For the angular rates, this was no issue for the DRL method due to its commanded angles being almost constant. In total, the rate constraints seemed to be respected for the vast majority of time steps in simulations.

The IADC increased significantly from simulations during the sea trials for both methods used as seen in Figure 6.12c. The commanded angles were more oscillating than in simulations but did not impact the IADC metric nearly as much as the commanded RPS changes. This might show another weakness of the IADC metric as the weighting done by dividing the RPS commands' derivatives with 100% and the commanded angles' derivatives with 180 degrees gave a benefit to the DRL method as it keeps the angle commands nearly constant. However, Figure 6.12c showed that the RPS contribution was still the largest by far, indicating that the IPI allocation commands a more oscillatory command on the low-level in between the rate constraints, while the DRL method's oscillating RPS commands did not have large derivatives when analyzing it over time. However, the largest increase of the IADC metric in the sea trials compared to simulations was by the DRL method. Due to the lack of feedback of the actual RPS of the thrusters' propellers as mentioned, the wear and tear could in reality be lower than what was measured by using the commanded values as in this thesis. On full-scale vessels, the resulting wear and tear becomes a result of the actual RPS output of a low-level propeller controller whose input would be the commanded signals, and thus the changes in the actual RPS values would most likely be smaller than what was observed when calculating the IADC metric with the commanded RPS values.

7.2.4 Robustness

Although robustness has no metric of its own, it is still a relevant parameter for DP systems. The robustness was tested in several ways, through large setpoint changes, station-keeping in a large sea state, and through deploying the DRL model trained in simulation to the real ship model.

The test with large setpoint changes showed that the DRL method, with and without integral effect, had a mean IAE of 19% less than the mean of the classic methods. In addition, the behavior of the vessel was more constant in terms of reducing body-frame errors in a stable fashion, only overshooting the largest setpoint change in yaw angle at 190 seconds, but without any oscillations in general. It was also seen from the pose plots in Figure 6.2c that the DRL method consistently and quickly prioritized to reduce errors in yaw.

The station-keeping test showed that the DRL method with integral effect was quite positionally accurate. This was not expected since the DRL method was trained in still water. As can be seen from both the plots of the pose in Figure 6.9 and the resulting IAE in Figure 6.10, the IPI allocation suffered a loss of positional accuracy, and especially the boundaries set for the yaw-range was violated both often, and with quite a margin. It could therefore be questioned whether or not the IPI allocation's result is valid for the test, since the main task was to station-keep within some set boundaries. Sacrificing positional accuracy for low energy usage must be a priority of the operator of the DP system, but in the case of DNVGL's standard in which the station-keeping test was built upon, these boundary violations was not satisfactory. As previously mentioned, this was believed to be an error related to the existing motion controller instead of the IPI allocation itself.

The entire sea trial itself demonstrated that the DRL method was rather robust for solving the four corner task with a model which was only trained in simulation, despite the hardware issues with the bow thruster. Even though the performance worsened in terms of both lower positional accuracy and larger actuator usage due to the hardware issue, the results showed that the DRL method was able to reach all setpoints, and that the largest increases in IAE when compared to simulations came from the overshoots in sway and particularly yaw. It was hypothesized that the DRL method, learning in simulations that yielding a certain RPS in the negative direction affected the vessel's motion as much as yielding positive RPS signals, could have been retrained in simulation with unsymmetrical bow thruster coefficients in order to properly compensate for this difference. This could have resulted in the DRL method learning to command larger RPS signals in the negative direction when needed, in order to control the sway and yaw motions more effectively on the physical model. Altogether, taking into account that the DRL method was only trained in a simulator with still water, the actual hardware issues, and the bow thruster's parameter mismatch, the method proved quite robust to the transition from simulator to real life application. However, robustness vulnerabilities of the DRL method were exposed when the differences between the actuators of the digital twin and the ship model was of the magnitude during these tests, as the performance of the motion controller with IPI allocation was closer to the simulation results which were performed with symmetrical coefficients for the bow thruster. To make the DRL method more robust, rate constraints could be explicitly be enforced, but this would also change the dynamic of the DRL method, requiring retraining the model. However, the exclusion of these constraints were on purpose since the thesis was to investigate the capability of employing DRL as directly as possible.

7.3 DRL Control Scheme Characteristics

From looking at the results, some main characteristics were possible to extract from the way that the DRL method completed its DP tasks. As seen in all the plots of commanded thruster angles, the DRL method taught itself to position the port thruster around $-140°$ and the starboard thruster at around $-40°$. This configuration is showed in Figure 7.1, where a screenshot was taken from the simulator during one of the simulations.

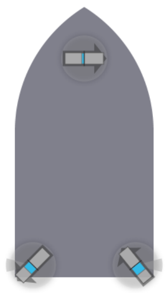


Figure 7.1: A screenshot of how the mean thruster angles were set by the DRL method.

In commercial systems, such configurations are usually referred to as "bias allocation", where the operator might activate a bias allocation mode in which the azimuth angles are not allowed to follow each other. This is done to prevent the azimuth angles from "hunting" the direction of the environmental loads, potentially pointing all thrusters in the same direction, and therefore the available force and moment production might be drastically reduced. Commercial systems solve this by adding additional objectives in the optimization formulation which favors certain thruster angles [76]. Interestingly, as can be seen from Figure 6.5b and Figure 6.8b, the classic methods also ended up on many occasions choosing similar thruster configurations with some oscillations *after* closing in on the setpoints, but between the setpoints, they rotated the thrusters a lot more. As mentioned in the previous work by Leavitt [10], optimization methods seemingly suffer from extensive thruster rotations, which was confirmed in the tests in this thesis as well. The QP allocation seemed to calculate quite similar angles for the two stern thrusters, while it was observed that the stern thrusters rotated more independently from each other for the IPI allocation.

By locking the stern thrusters' angles, the DRL method made an accuracy versus capability trade-off by prioritizing a thruster setup which made positional accuracy easier, but in turn not utilizing the full potential of the thruster setup. This was however to be expected by the way the model was trained, as the reward function only rewarded positional accuracy, and included no components related to capability. What was however interesting to note by this specific angle lock was that the method avoided at any point in time to angle the stern thrusters towards each other, indicating that it learned that such a thruster setup was inefficient both in terms of positional accuracy and energy efficiency. This supports the inclusion of forbidden zones in classic thrust allocation methods, where thrusters are given some areas of rotation in which they are not allowed to exert thrust within in order to avoid flow disturbances around neighboring thrusters.

In simulations, the DRL method proved energy efficient by acting slightly different than the motion controller + thrust allocation methods. When a setpoint change occurred, the DRL method was yielding commanding large RPS initially, followed by allowing the vessel to slowly drift towards the setpoint with minor RPS commands, slightly increasing the magnitude of the commands for small adjustments when closing in on the setpoint. This was believed to be the main reason as to why the DRL methods' performance was reduced between the simulation of the four corner test with and without ocean current enabled. As the DRL method was trained in still water, the strategy of letting the vessel drift towards setpoints while commanding small RPS values showed inefficient when the ocean current was enabled until the state integrator was added, but the integrator also caused the method to display some overshoots. The classic methods commanded more fluctuating and large RPS values over time, which resulted in that their energy usage was higher.

An advantage of the DRL method when compared to the classic methods was that it was able to learn directly from the dynamic system, and not from a parameterized, explicit formulation of the hull dynamics, the thrusters' configuration and their parameters. A clear example of such an advantage can be discussed from looking at the Bollard test from Alfheim and Muggerud in Figure 5.2. The thruster force from a thruster was assumed to follow a quadratic shape according to $T_i \propto K_i n_i |n_i|$, whereas the results from the actual Bollard test showed that this quadratic relation did not hold for the entire range of the commanded $|n_i|$-values, especially for the stern thrusters. Where the classic methods depends on simplifications such as the quadratic relationship between a thruster's force and RPS by using constant K_i-values, the DRL method might learn the actual relationship by itself. In reality, the K_i-values depends on the *advance number* $J = {}^{V_a}/_{nD}$, which again depends on the inflow velocity V_a to the propeller, the RPS value n and the propeller diameter D. In the classic methods, such a dependency could be formulated by storing a table of the K_i-values and adjusting them according to the advance number in the optimization problems, but the DRL method might learn this by itself. Therefore, the DRL methods' ability to learn more accurate thruster models including more complex, non-linear interactions between the vessels' velocity and orientation, the commanded RPS, and the resulting thrust force might explain why the DRL method turned out to be energy efficient in simulations, while at the same time being quite accurate.

A disadvantage of using the DRL method as it was implemented in this thesis is that if a change to the thrusters was to occur, e.g. by moving some of the thruster's locations, changing their characteristics, or a thruster malfunction as shown during the sea trials in this thesis, there would be a need of training a new model. This would take time both in terms of the training process itself, but also in terms of time consumed by validating and testing the new performance. For allocation methods such as those based on the pseudoinverse or optimization, only a few lines of code would suffice in order to adapt the software to these thruster configuration changes, making them more flexible and easy to handle for not only changes to the current thruster configuration, but to applications on completely new ones as well. The classic method is likely to need tuning as well, but it could be performed straight away, whereas the DRL method would need time to first be retrained, followed by validation of performance. If the performance is not found satisfactory, then another training session would consume even more time. This disadvantage could however become a smaller issue if retraining is used efficiently, where the trained network is directly applied as the initial network used during training as apposed to training a Neural Network from scratch.

Another disadvantage of the DRL method is that the rate constraints could be hard to enforce, and by using the outputs from the Neural Network directly, this thesis showed that rate constraints can easily be violated, causing more wear and tear on the actuators. In addition, a disadvantage in general is that it could be hard to perform stability analysis and to give other performance guarantees apart from performing various tests. Classic methods has been developed and tested for a long time, and mathematical stability guarantees could be given by e.g. Lyapunov Analysis. This is discussed more in the recommendations for further work in Chapter 9.

Chapter 8

Conclusion

In this thesis, the Deep Reinforcement Learning algorithm Proximal Policy Optimization was used in order to develop a control scheme for Dynamic Positioning of a marine surface vessel using Neural Networks to calculate various thruster commands. Various network architectures and state- and action vector representations was tested. Larger networks tended to give more stable learning results, and the inclusion of the previous time steps' commanded signals in the state vector made the agent learn the Dynamic Positioning tasks well in addition to reducing the actuator usage. Especially the inclusion of predicting sines and cosines of the thruster angles in the action vector helped the learning algorithm to yield more stable angle commands. A considerable amount of experimentation with different reward functions for positional accuracy was also done, where an analysis was carried out considering the performance of the Deep Reinforcement Learning agent between using a multivariate, Gaussian reward function instead of the sum of multiple, univariate Gaussian reward functions. The multivariate Gaussian reward function was constructed in a way which lowered sparsity, and aided the learning process in a way that to the author's knowledge is new to the field. By creating this smooth reward function which was to be optimized by the Deep Reinforcement Learning agent, the entire learning process itself became quick and stable, yielding a model which was able to perform well both in terms of positional accuracy, energy efficiency, robustness, and wear and tear of the actuators.

In addition to presenting the work of applying Deep Reinforcement Learning to solve Dynamic Positioning tasks, several test scenarios was constructed and presented, including comparisons to existing methods for Dynamic Positioning. The resulting Deep Reinforcement Learning control scheme was able to perform the Dynamic Positioning tasks well when tested on the large variety of test scenarios in simulations, in addition to proving capable during sea trials. In simulations, the Deep Reinforcement Learning method yielded better positional accuracy while being considerably more energy efficient than the methods it was tested against. These methods consisted of a motion controller consisting of both a feedforward control law and a Proportional-Integral-Derivative feedback control law, using two different thrust allocation methods for translating the desired forces into thruster commands. A station-keeping task displayed that the station-keeping ability of the Deep Reinforcement Learning method was quite good in rough sea states, even though the Deep Reinforcement Learning method had only been trained in still water. Even though the quality of the comparisons was discussed due to the seemingly sub-optimal implementation of the existing motion control law, it was proposed that the Deep Reinforcement Learning method was a strong contender when looking for new methods to perform station-keeping and low speed maneuvering in a way that encapsulates both the motion control and thrust allocation tasks, instead of separating them into separate entities like existing methods does today. The Neural Network which calculated the thruster commands, which was trained only in a simulator, was employed to a physical ship model. It was found to perform positionally accurate Dynamic Positioning, hence proving quite robust despite issues related to the bow thruster's hardware. In addition, due to the controller consisting of a Neural Network, the computational time used by the control system was negligible, combating the problem of computational complexity which most existing optimization based methods are facing today.

The thesis therefore concludes with that Deep Reinforcement Learning's potential for accurate and energy efficient control is not only theoretically possible. It was shown to perform well when compared to classic methods in simulations, in addition to performing well in a real life sea trial by directly applying the model which was trained in simulations. However, as with all Deep Reinforcement Learning applications, it is hard to give stability and performance guarantees, although this was tested to a certain degree by exposing the Deep Reinforcement Learning method to setpoint changes and sea states which it had not been previously trained to handle. In addition, it proved less robust to actuator malfunctions when compared to a classic method. This is an area which is especially important for industrial applications and crucial for the classification societies. Therefore, the recommendations for further work includes pointers to relevant work done within the field of *safe learning*, in addition to how Deep Reinforcement Learning agents can be trained to adapt to differences between the dynamic system in simulation and in real life.

Chapter 9

Further Work

This chapter focuses on recommendations for further work on the DRL topic for control of marine vessels. It should be emphasized that, as discussed, the methods used for comparison most likely suffers from a sub-optimal motion control law, so further work could go into improving these for more fair comparisons. However, this chapter only discusses the way forward using the DRL method.

9.1 Increasingly Complex Thruster Configurations

Further work which directly builds on top of this thesis is to test the PPO algorithm on vessels which contains a lot more complex thruster configurations that ReVolt does with its 3 thrusters. This could include setups with rudders behind the stern propellers, multiple tunnel thrusters, etc. Such elements would make the learning task more difficult, but would also give rise to a more complex situation for optimization based thrust allocation schemes, creating interesting cases for comparative studies.

9.2 Continuous Learning

It is possible to load the Neural Networks for enabling retraining of the DRL models, meaning that the training can *continue* from where it left of in the simulator and thus be a big time saver when training control policies for real life applications. Since the state-vector was formulated in terms of error states, the training could be performed efficiently by determining a region to sample new setpoints from after each trajectory in order to emulate resetting the environment in the simulator. This could enable the learning algorithm to tweak the Neural Networks' weights and biases to details that differ between the digital twin and the real model that are difficult to mathematically model, such as experienced during the real life experiments in this thesis where the thrusters was not complying to the commanded signals properly.

Recent work on this area has shown big effect: work published in April 2020 by Julian et al. [77] tried to answer their own formulated question "*Can we efficiently adapt previously learned behaviors to new environments, objects and percepts in the real world?*". The work demonstrated an efficient method with real life results on continuous learning, using a pre-trained model of a robotic gripper arm whose task was to grasp and manipulate various objects. They showed that, by including changes to the real life environment such as changes in background, the shape and appearance of the objects, various lighting conditions, and changes to the robot morphology, their method efficiently re-trained the model to adapt in a short period of time, making it flexible to a large variety of changes in the environment compared to what it was initially trained on. Work like this becomes highly relevant for marine applications for several reasons, probably the most important one being that the mathematical modeling of the vessel and thruster dynamics includes errors which could be re-learned on board the real vessel. As this could be time consuming, it is also important to utilize methods that considerably lower the necessity of the amount of data, which this work achieved with a stated necessity of only 0.2% of data for reaching the end performance compared with training a model from scratch to solve the same tasks on the various augmentations of environment.

9.3 Safe Learning

Work by Achiam et al. [78] on "Constrained Policy Optimization" sought to find safe ways of performing exploration for an RL agent, and combined reward shaping with *constraint* shaping in order to ensure a low bound of the worst performance. The performance of an RL agent might look good at first glance, but that is given that the environment it is trained in is a good replica of the environment it is going to be used in. For the agent trained in this thesis, if the changes in setpoints were large, the vessel *seemed* to be controlled in a reasonable manner by looking at the results from Figure 6.2, but it is hard to evaluate if it would behave similarly for *any* combination of setpoint changes that are larger than what the model was trained on. Therefore, further investigations into how to constrain the agent to be more safe is recommended, as this is crucial for the applications of DRL methods into real life industrial usage, in addition to being crucial for enabling continuous learning as explained above. It is also recommended to investigate if Constrained Policy Optimization could be used in order to enforce rate constraints of the thrusters, as using the output of the Neural Network directly as in this thesis might cause the thruster commands to violate the thrusters' physical limitations.

Ways of providing a sense of safety and/or stability was also investigated by Richards et al. [79], providing a framework for performing Lyapunov Analysis on Neural Networks as a way of ensuring safe performance on safety-critical systems learning the safest region of the state-space. Also Berkenkamp et al. [80] provided a method for model-based RL with stability guarantees by considering safety in terms of stability guarantees *during* training, combining Lyapunov Analysis and statistical models of the dynamics to obtain well performing policies with provable stability certificates. All of the above mentioned work may be of interest of exploring, especially for the classification society, on the path to providing performance guarantees of DRL methods.

References

[1] Simen Sem Øvereng. "Solving Thrust Allocation with Machine Learning". Unpublished. Project Thesis. NTNU, 2019.

[2] DNVGL. Autonomous and remotely-operated ships. https://www.dnvgl.com/maritime/autonomous-remotely-operated-ships/index.html (visited on 2019/11/14).

[3] DNVGL. *"The ReVolt - A new inspirational ship concept"*. https://www.dnvgl.com/technology-innovation/revolt/index.html (visited on 2019/10/01).

[4] Asgeir Johan Sørensen. *Marine Cybernetics. Towards Autonomous Marine Operations and Systems.* Unpublished. Department of Marine Technology, NTNU, 2018.

[5] Astrid H. Brodtkorb. "Hybrid Control of Marine Vessels - Dynamic Positioning in Varying Conditions". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2482881. PhD dissertation. NTNU, 2017.

[6] Ole Jakob Sørdalen. "Optimal Thrust Allocation for Marine Vessels". In: *Control Eng. Practice, Vol. 5. No. 9* (1997), pp. 1223–1231. DOI: 10.1016/S0967-0661(97)84361-4.

[7] Nils A. Jenssen and Bjørnar Realfsen. *"Power Optimal Thruster Allocation "*. Kongsberg Maritime AS, https://pdfs.semanticscholar.org/064e/1b38204f5d855266a4eeb4af7b705a9bb2b9.pdf (visited on 2019/11/18). 2006.

[8] P. Tøndel, T.A. Johansen, and A. Bemporad. "An Algorithm for Multi-Parametric Quadratic Programming and Explicit MPC Solutions". In: *Automatica, Volume 39, Issue 3.* (2003), pp. 489–497. DOI: 10.1016/S0005-1098(02)00250-9.

[9] T. A. Johansen, T. I. Fossen, and P. Tøndel. "Efficient Optimal Constrained Control Allocation via Multi-Parametric Programming". In: *AIAA J. Guidance, Control and Dynamics* 28 (2005), pp. 506–515. DOI: 10.2514/1.10780.

[10] John Leavitt. *"Optimal Thrust Allocation in DP Systems"*. Dynamic Positioning Committee, https://dynamic-positioning.com/proceedings/dp2008/thrusters_leavitt_pp.pdf (visited on 2019/09/17). 2008.

[11] Martin Rindarøy and Tor Arne Johansen. "Fuel Optimal Thrust Allocation in Dynamic Positioning". In: *IFAC Proceedings Volumes, Volume 46, Issue 33.* (2013), pp. 43–48. DOI: 10.3182/20130918-4-JP-3022.00032.

[12] Tor. A Johansen and Thor I. Fossen. "Control Allocation - A Survey". In: *Automatica* 46 (2013), pp. 1087–1103. DOI: 10.1016/j.automatica.2013.01.035.

[13] Chris Vermillion, Jing Sun, and Ken Butts. "Model predictive control allocation for overactuated systems - Stability and performance". In: *IEEE Conference on Decision and Control.* Vol. 47. 2007, pp. 1251–1256. DOI: 10.1109/CDC.2007.4434722.

[14] M. Naderi, A. Khaki Sedigh, and T.A. Johansen. “Guaranteed feasible control allocation using model predictive control”. In: *Control Theory Technol* 17 (2019), pp. 252–264. DOI: https://doi.org/10.1007/s11768-019-7231-90.

[15] A. Veksler et al. “Dynamic Positioning With Model Predictive Control”. In: *IEEE Transactions on Control Systems Technology* 24 (2016), pp. 1340–1353. ISSN: 1558-0865. DOI: 10.1109/TCST.2015.2497280.

[16] Zhao Luman, Myung-Il Roh, and Jeong-Woo Hong. “Optimal Thrust Allocation for Dynamic Positioning of Deep-sea Working Vessel”. In: *Journal of Advanced Research in Ocean Engineering* 1 (2015), pp. 94–105. DOI: 10.5574/JAROE.2015.1.2.094.

[17] García-Martínez C., Rodriguez F.J., and Lozano M. *Genetic Algorithms*. Springer, Cham, 2018. ISBN: 9783319071244. DOI: 10.1007/978-3-319-07124-4_28.

[18] Charles Darwin. *On The Origin of Species by Means of Natural Selection, or Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1859. DOI: 10.1038/005318a0.

[19] Defeng Wu, Fengkun Ren, and Weidong Zhang. “An energy optimal thrust allocation method for the marine dynamic positioning system based on adaptive hybrid artificial bee colony algorithm”. In: *Ocean Engineering* 118 (2016), pp. 216–226. DOI: 10.1016/j.oceaneng.2016.04.004.

[20] Bharat Monga and Jeff Moehlis. *Supervised Learning Algorithms for Controlling Underactuated Dynamical Systems*. 2019. arXiv: 1909.11119 [math.OC].

[21] Robert Skulstad et al. “A Neural Network Approach to Control Allocation of Ships for Dynamic Positioning”. In: *IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS* 51 (2018), pp. 128–133. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.09.481.

[22] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: 1509.02971 [cs.LG].

[23] John Schulman et al. *Trust Region Policy Optimization*. 2015. arXiv: 1502.05477 [cs.LG].

[24] Jemin Hwangbo et al. “Control of a Quadrotor With Reinforcement Learning”. In: *IEEE Robotics and Automation Letters* 2 (2017), pp. 2096–2103. ISSN: 2377-3774. DOI: 10.1109/lra.2017.2720851. URL: http://dx.doi.org/10.1109/LRA.2017.2720851.

[25] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[26] Guilherme Lopes et al. “Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning”. In: Latin American Robotic Symposium. 2018, pp. 503–508. DOI: 10.1109/LARS/SBR/WRE.2018.00094.

[27] Eivind Bohn et al. “Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy optimization”. In: *International Conference on Unmanned Aircraft Systems (ICUAS)* (2019). DOI: 10.1109/icuas.2019.8798254. URL: http://dx.doi.org/10.1109/ICUAS.2019.8798254.

[28] Andreas Bell Martinsen. “End-to-End Training for Path Following and Control of Marine Vehicles”. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2559484. Master’s Thesis. NTNU, 2018.

[29] Eirik F. Kjærnli. "Deep Reinforcement Learning Based Controllers In Underwater Robotics". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2615067. Master's Thesis. NTNU, 2018.

[30] Kristoffer Borgen Knudsen. "Deep Learning for Station Keeping of AUVs". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2622920. Master's Thesis. NTNU, 2019.

[31] Ingunn Johanne Vallestad. "Path Following and Collision Avoidance for Marine Vessels with Deep Reinforcement Learning". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2625707. Master's Thesis. NTNU, 2019.

[32] Ella-Lovise Hammervold Rørvik. "Automatic Docking of an Autonomous Surface Vessel". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2656724. Master's Thesis. NTNU, 2020.

[33] Henrik Alfheim and Kjetil Muggerud. "Dynamic Positioning of the ReVolt Model-Scale Ship". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2595418. Master's Thesis. NTNU, 2016.

[34] Albert Havnegjerde. "Remote control and path following for the revolt model ship". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2560382. Master's Thesis. NTNU, 2018.

[35] Andreas B. Martinsen et al. "Reinforcement Learning-Based Tracking Control of USVs in Varying Operational Conditions". In: *Frontiers in Robotics and AI* 7 (2020), p. 32. ISSN: 2296-9144. DOI: 10.3389/frobt.2020.00032.

[36] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control.* John Wiley & Sons, 2011. ISBN: 9781119994138. DOI: 10.1002/9781119994138.

[37] Øyvind Notland Smogeli. "Control of Marine Propellers: from Normal to Extreme Conditions". https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/237615. PhD dissertation. NTNU, 2006.

[38] Adi Ben-Israel and Thomas N.E. Greville. *Generalized inverses: Theory and applications (2nd ed.)* Springer,New York, NY, USA, 2003. ISBN: 9780387002934. DOI: 10.1007/b97366.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, 2nd edition.* http://incompleteideas.net/book/the-book.html. MIT Press, 2017.

[41] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning.* https://spinningup.openai.com/ (visited on 2020/04/12). 2018.

[42] Sergey Levine. *CS 294: Deep Reinforcement Learning, Fall 2017.* http://rail.eecs.berkeley.edu/deeprlcourse-fa17/index.html#lectures (visited on 2020/03/20). 2017.

[43] Jeff Heaton. *Artificial Intelligence for Humans, Vol 3: Neural Networks and Deep Learning.* https://www.amazon.com/dp/1505714346. Heaton Research Inc., 2015. ISBN: 1505714346.

[44] J. Kiefer and J. Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function". In: *Ann. Math. Statist.* 23 (1952), pp. 462–466. DOI: 10.1214/aoms/1177729392.

[45] Simon S. Du et al. *Gradient Descent Finds Global Minima of Deep Neural Networks*. 2018. arXiv: 1811.03804 [cs.LG].

[46] Moritz Hardt, Ben Recht, and Yoram Singer. "Train faster, generalize better: Stability of stochastic gradient descent". In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. 2016, pp. 1225–1234. URL: http://proceedings.mlr.press/v48/hardt16.html.

[47] Michael P. Perrone et al. *Optimal Mini-Batch Size Selection for Fast Gradient Descent*. 2019. arXiv: 1911.06459 [cs.LG].

[48] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Macine Learning Research* 12 (2011), pp. 2121–2159. ISSN: 1532-4435.

[49] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: 1212.5701 [cs.LG].

[50] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[51] Oliver C. Ibe. *Markov Processes for Stochastic Modeling, 2nd edition*. https://www.sciencedirect.com/book/9780124077959/markov-processes-for-stochastic-modeling. Elsevier Inc., 2013. ISBN: 9780124077959.

[52] Christoffer John Cornish Hellaby Watkins. "Learning from Delayed Rewards". http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf. PhD dissertation. King's College, 1989.

[53] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[54] R.J. Williams. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. https://doi.org/10.1007/BF00992696. 1992.

[55] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. arXiv: 1506.02438 [cs.LG].

[56] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21 (2008), pp. 682–697. DOI: 10.1016/j.neunet.2008.02.003.

[57] Ya-xiang Yuan. "Trust region algorithms for nonlinear programming". In: (1994). DOI: 10.1090/conm/163/01559.

[58] Ya-xiang Yuan. "A Review of Trust Region Algorithms for Optimization". In: *ICM99: Proceedings of the Fourth International Congress on Industrial and Applied Mathematics* (1999). https://www.researchgate.net/publication/2304086_A_Review_of_Trust_Region_Algorithms_for_Optimization.

[59] A.R. Conn, N.I.M. Gould, and P.L. Toint. *Trust Region Methods*. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2000. ISBN: 9780898714609. URL: https://books.google.no/books?id=5kNC4fqssYQC.

[60] Sham Kakade and John Langford. "Approximately Optimal Approximate Reinforcement Learning". In: *Proceedings of the Nineteenth International Conference on Machine Learning*. 2002, pp. 267–274. ISBN: 1558608737.

[61] A. Rupam Mahmood et al. *Benchmarking Reinforcement Learning Algorithms on Real-World Robots*. 2018. arXiv: 1809.07731 [cs.LG].

[62] Gang Chen, Yiming Peng, and Mengjie Zhang. *An Adaptive Clipping Approach for Proximal Policy Optimization*. 2018. arXiv: 1804.06461 [cs.LG].

[63] Eric Wiewiora. “Reward Shaping”. In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 863–865. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_731. URL: https://doi.org/10.1007/978-0-387-30164-8_731.

[64] MTS Dynamic Positioning Committee. *Guidelines on Testing of DP Systems*. https://dynamic-positioning.com/wp-content/uploads/2015/06/Guidelines_on_testing_dp_systems.pdf (visited on 2020/04/20).

[65] Kongsberg. *K-Pos Operator Manual - Dynamic Positioning Operators Manual*. https://www.scribd.com/doc/149472780/K-Pos-Operator-Manual (visited on 2020/04/20).

[66] Wil Koch et al. “Reinforcement Learning for UAV Attitude Control”. In: *ACM Transactions on Cyber-Physical Systems* 3 (2018). DOI: 10.1145/3301273.

[67] Chigozie Nwankpa et al. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *CoRR* abs/1811.03378 (2018). http://arxiv.org/abs/1811.03378. arXiv: 1811.03378.

[68] Chigozie Nwankpa et al. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. arXiv: 1811.03378 [cs.LG].

[69] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *Proceedings of the 30 th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013*. Computer Science Department, Stanford University, 2013.

[70] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015. arXiv: 1505.00853 [cs.LG].

[71] Andrej Karpathy. *Deep Reinforcement Learning: Pong from Pixels*. http://karpathy.github.io/2016/05/31/rl/ (visited on 2020/05/17).

[72] Roger Skjetne, Øyvind Smogeli, and Thor I.Fossen. “Modeling, identification, and adaptive maneuvering of CyberShip II: A complete design with experiments”. In: *IFAC Proceedings Volumes*. Vol. 37. 2004, pp. 203–208.

[73] DNVGL. *DNVGL-ST-0111; Assessment of station keeping capability of dynamic positioning vessels*. https://rules.dnvgl.com/docs/pdf/DNVGL/ST/2016-07/DNVGL-ST-0111.pdf (visited on 2020/05/20).

[74] Bjørn-Olav Eriksen and Morten Breivik. “Modeling, Identification and Control of High-Speed ASVs: Theory and Experiments”. In: *Sensing and Control for Autonomous Vehicles: Applications to Land, Water and Air Vehicles*. 2017, pp. 407–431. ISBN: 978-3-319-55372-6. DOI: 10.1007/978-3-319-55372-6_19.

[75] Henrik Lemcke Alfheim et al. “Development of a Dynamic Positioning System for the ReVolt Model Ship”. In: *11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018* 51 (2018), pp. 116–121. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.09.479.

[76] Karl Kaasen. “Efficient Thrust Allocation for DP Systems Using a Biased Least Squares Criterion”. In: 2015. DOI: 10.1115/OMAE2015-41948.

[77] Ryan Julian et al. *Efficient Adaptation for End-to-End Vision-Based Robotic Manipulation*. 2020. arXiv: 2004.10190 [cs.LG].

[78] Joshua Achiam et al. *Constrained Policy Optimization*. 2017. arXiv: 1705.10528 [cs.LG].

[79] Spencer M. Richards et al. *The Lyapunov Neural Network: Adaptive Stability Certification for Safe Learning of Dynamical Systems*. 2018. arXiv: 1808.00924 [cs.SY].

[80] Felix Berkenkamp et al. *Safe Model-based Reinforcement Learning with Stability Guarantees*. 2017. arXiv: 1705.08551 [stat.ML].

Appendix A

Hydrodynamic Coefficients

This appendix gives the hydrodynamic coefficients of the ReVolt ship model which was found during the work from Alfheim and Muggerud [33].

The system inertia matrix M in 3DOF is the sum of the rigid body inertia,

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & mx_g \\ 0 & mx_g & Iz \end{bmatrix} = \begin{bmatrix} 257 & 0 & 0 \\ 0 & 257 & 0 \\ 0 & 0 & 298 \end{bmatrix}, \tag{A.1}$$

and the added mass:

$$\boldsymbol{M}_{A} = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{v}} \end{bmatrix} = \begin{bmatrix} 6.930 & 0 & 0 \\ 0 & 49.440 & 7.007 \\ 0 & 7.028 & 24.556 \end{bmatrix}. \tag{A.2}$$

Their respective sum then becomes

$$\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_{A} = \begin{bmatrix} 263.93 & 0 & 0 \\ 0 & 306.44 & 7.00 \\ 0 & 7.03 & 322.15 \end{bmatrix}. \tag{A.3}$$

The Coriolis and centripetal matrix is also separated into rigid body and added mass elements:

$$\begin{aligned} \boldsymbol{C}(\nu) = \boldsymbol{C}_{RB}(\nu) + \boldsymbol{C}_{A}(\nu) = & \begin{bmatrix} 0 & 0 & -m\left(x_g r + v\right) \\ 0 & 0 & mu \\ m\left(x_g r + v\right) & -mu & 0 \end{bmatrix} \\ & + \begin{bmatrix} 0 & 0 & Y_i v + Y_i r \\ 0 & 0 & -X_i u \\ -Y_{ij} v - Y_i r & X_i u & 0 \end{bmatrix} \end{aligned} \tag{A.4}$$

The numerical values found was the following:

$$\boldsymbol{C}(\nu) = \begin{bmatrix} 0 & 0 & -207.56v + 7.00r \\ 0 & 0 & 250.07u \\ 207.56v - 7.00r & -250.07u & 0 \end{bmatrix}. \tag{A.5}$$

Lastly, the damping matrix found (being linear) was

$$\boldsymbol{D} = \begin{bmatrix} -X_u & 0 & 0 \\ 0 & -Y_v & -Y_r \\ 0 & -N_v & -N_r \end{bmatrix} = \begin{bmatrix} 50.66 & 0 & 0 \\ 0 & 601.45 & 83.05 \\ 0 & 83.10 & 268.17 \end{bmatrix}. \tag{A.6}$$

Appendix B

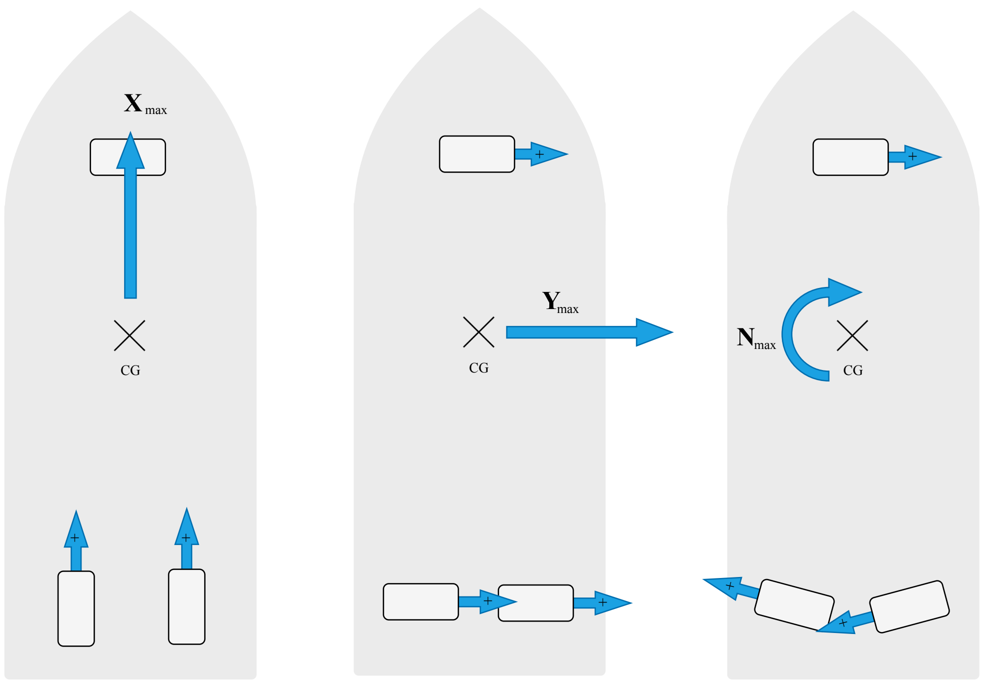
New Motion Controller Bounds

The various thruster configurations used for the calculations are shown in Figure B.1. The calculations was done by finding the maximum producible force in one dimension, while still reaching equilibrium in the two others. For the maximum surge force, the bow thruster produced zero force, while the two stern thrusters produced their maximum force at $0°$, giving that the sum of forces in sway and sum of moments in yaw were zero. For the maximum sway force, the bow thruster produced its maximum force, while the two stern thrusters each produced enough force to give equilibrium in yaw while staying at $90°$, meaning that no force was applied in surge. This force was found through $(2 \cdot f \cdot 1.12\ m - 9\ N \cdot 1.08\ m = 0) \implies f = 4.34\ N$. See Figure 4.2 for dimensions used. For the yaw moment, the bow thruster yielded maximum thrust, while the two stern thrusters were placed at two angles mirrored along the sway axis of the vessel in order for the sum of forces in surge to be zero. The angle was chosen from using Pythagoras' theorem for them to be placed at a $90°$ intercept where they are able to produce the largest yaw moment (being $\alpha' = \pm \arctan(0.15\ \mathrm{m}/1.12\ \mathrm{m}) = \pm 7.6°$). For their sum of forces in sway to cancel the bow thruster's force, the force they had to be producing was found through $2 \cdot f\cos(\alpha') - 9\ N = 0 \implies f = 4.54\ N$. The calculations of the values are shown in in Equation (B.1), resulting in the maximum force/moment vector $\boldsymbol{\tau}_{max}$ shown in Equation (5.4).

$$X_{max} = 0\ N + 2 \cdot 20.5\ N = 41.0\ N \tag{B.1a}$$

$$Y_{max} = 9\ N + 2 \cdot 4.34\ N \approx 17.7\ N \tag{B.1b}$$

$$N_{max} = 9\ N \cdot 1.08\ m + 2 \cdot 4.54\ N \cdot \sqrt{(0.15\ m)^2 + (1.12\ m)^2} \approx 20.0\ Nm \tag{B.1c}$$



(a) Surge. **(b)** Sway. **(c)** Yaw.

Figure B.1: Thruster configurations for calculation of motion controller output boundaries.

Appendix C

Arctan2 Angle Continuity

This table shows that the chosen action vector, calculating the sines and cosines of the thruster angles instead of the raw angles as shown in Equation (5.8b), is able to represent angles as continuous, or rather circular, values.

Table C.1: $\alpha = \text{arctan2}(\sin(\hat{\alpha}), \cos(\hat{\alpha}))$ for various sine and cosine values, in degrees.

$\sin(\hat{\alpha})$ \ $\cos(\hat{\alpha})$	-1.0	-0.67	-0.33	0.0	0.33	0.67	1.0
-1.0	-135	-123.7	-108.4	-90.0	-71.6	-56.3	-45
-0.67	-146.3	-135.0	-116.6	-90.0	-63.4	-45	-33.7
-0.33	-161.6	-153.4	-135.0	-90.0	-45	-26.6	-18.4
0.0	180.0	180.0	180.0	0.0	0.0	0.0	0.0
0.33	161.6	153.4	135.0	90.0	45.0	-26.6	18.4
0.67	146.3	135.0	116.7	90.0	63.4	45	33.7
1.0	135	123.7	108.4	90.0	71.6	56.31	45

Appendix D

Neural Network Training Plots

This appendix shows the training progress of different DRL agents using varying network sizes, activation functions and actor vector representations. For ease of inspection, they have been given their separate pages in the following. The results could briefly be summarized as:

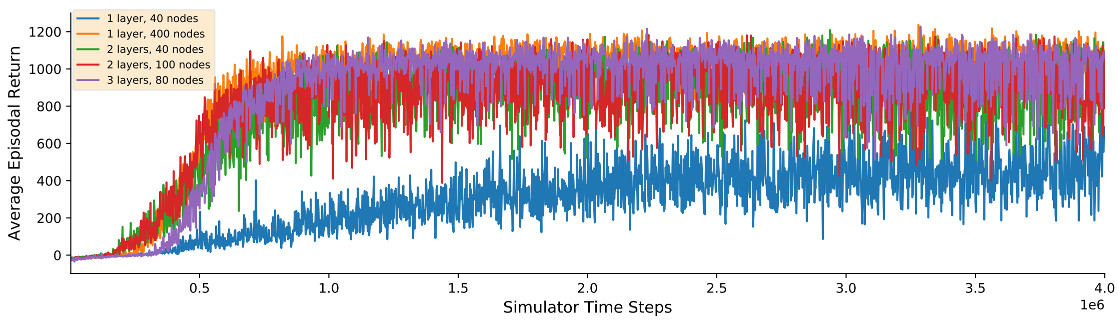
- **Network size**: Both the architectures with 1 layer with 400 nodes, 2 layers with 100 nodes, and 3 layers with 80 nodes converged to stable average episodal returns and average value function estimations rather quickly, but the latter architecture displayed less variance. It also displayed better results during validation (according to tests as explained in Section 5.4.11), giving the reasoning for it being selected.

- **Activation function**: For all tests using 2 layers with 80 nodes, the leaky ReLU activation function converged the fastest to stable average episodal returns as well as value function estimates. The same was observed for the tests using 3 layers with 80 nodes; thus, the leaky ReLU activation function was chosen.

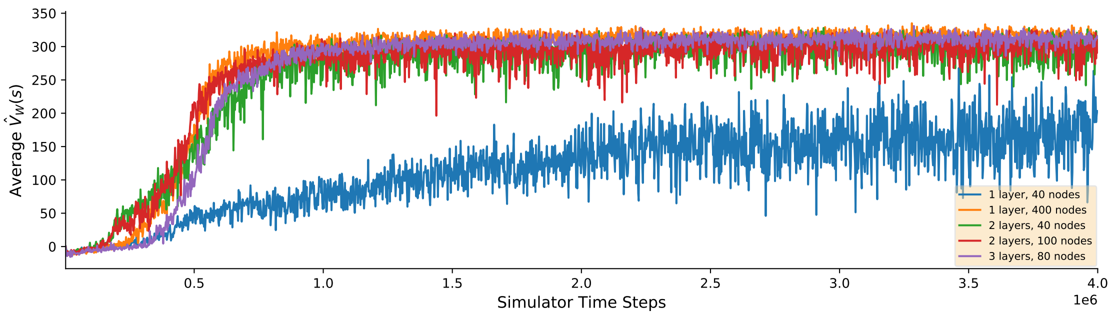
- **Action vector**: For both tests using the "continuous angle" representation in the action vector, the convergence time for both stable episodal returns and value function estimates was significantly faster compared to using networks which directly predicted the "raw" azimuth angles. The latter did not even seem to converge to particularly stable values over time, thus giving a simple choice for which actor vector representation to choose.

Network Size

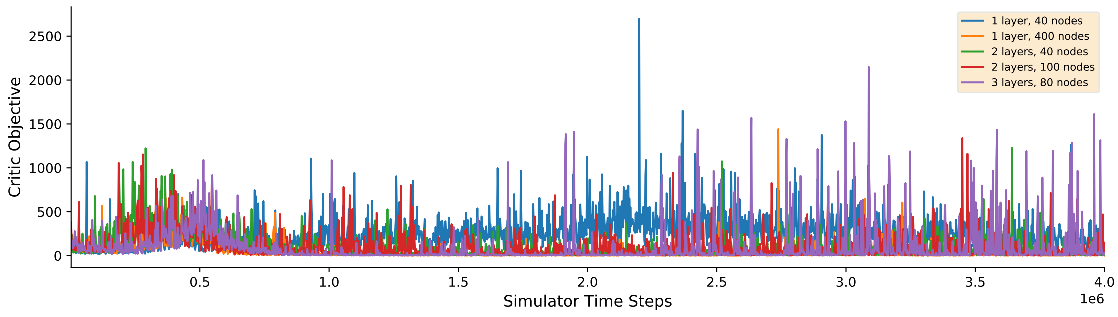
In the following, the various training sessions with different network architectures are shown.



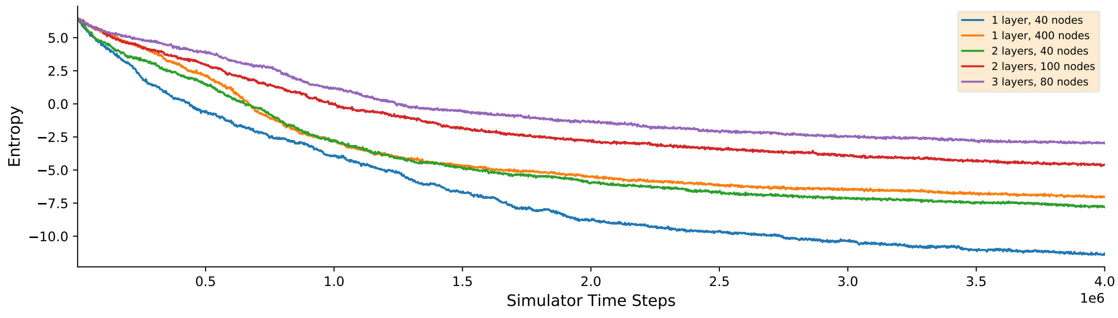
(a) Average return per epoch.



(b) Average V-values.



(c) Critic objective.

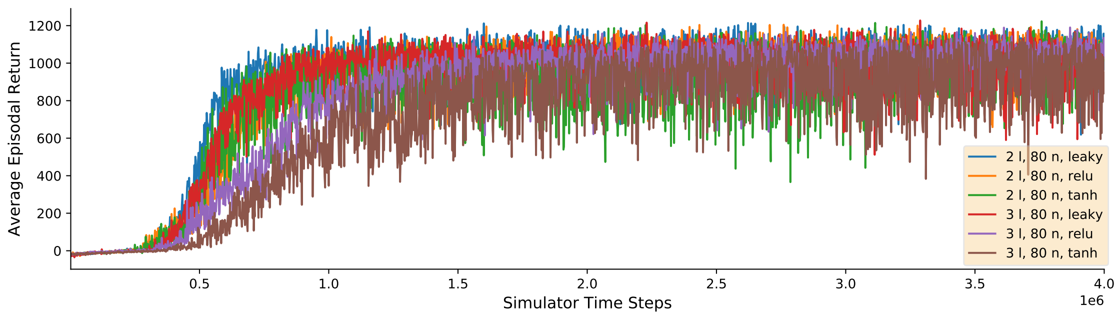


(d) Entropy.

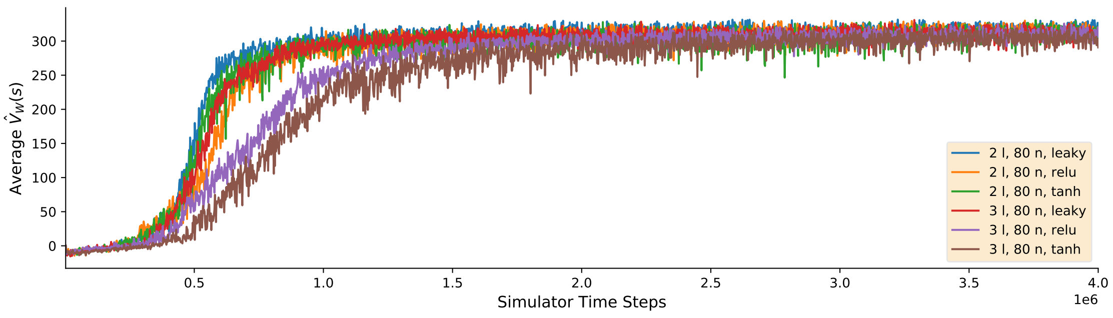
Figure D.1: Training results from various neural network architectures.

Activation Functions

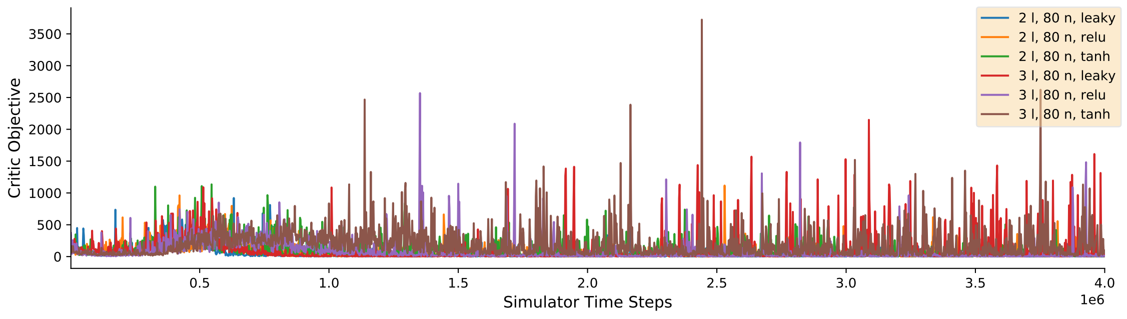
In the plots below, "l" refers to *layers*, "n" refers to *nodes*, and "leaky", "relu" and "tanh" refers to activation functions "leaky ReLU", "ReLU" and "hyperbolic tangent".



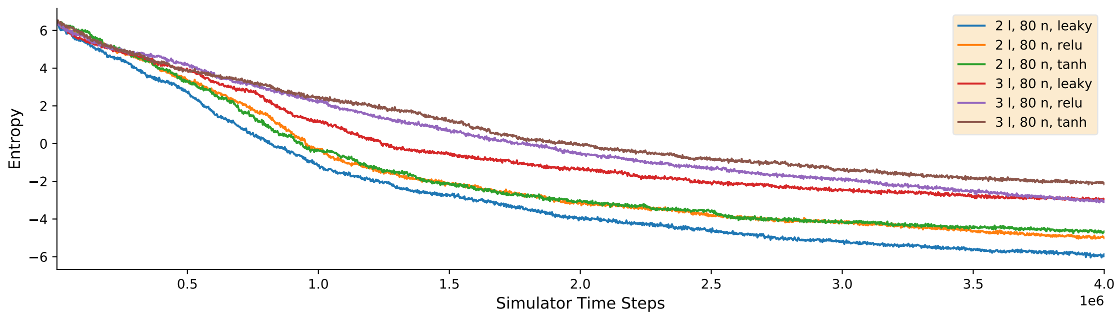
(a) Average return per epoch.



(b) Average V-values.



(c) Critic objective.

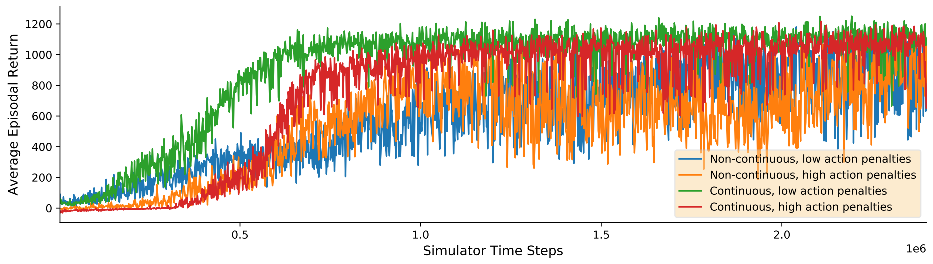


(d) Entropy.

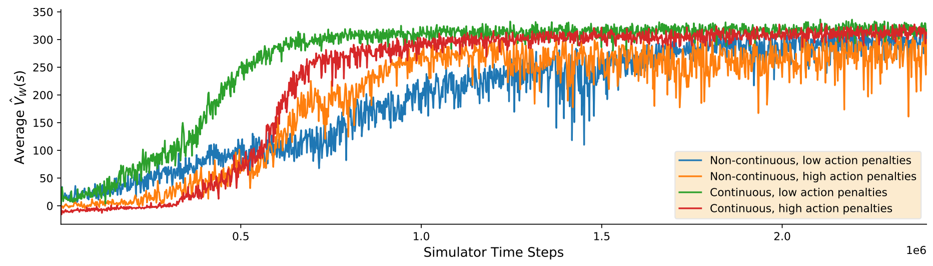
Figure D.2: Training results from various activation functions.

Action Vector Shape

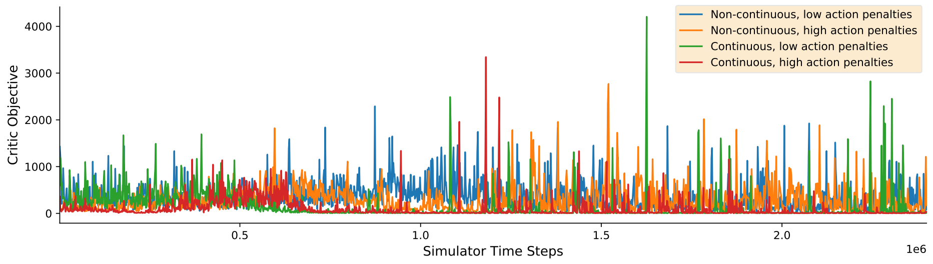
These plots shows the training results from varying the actor vector representations between Equation (5.8a) (represented as "non-continuous", and Equation (5.8b) (represented as "continuous"). The action penalty coefficients in the final reward function as presented in Section 5.4.4 was tweaked to slightly higher and lower values than used during the training of the final DRL model just to observe if that would have an impact on the convergence of the different action vectors, which it did not (at least on the objective, which was to evaluate the two different action vector representations against each other; clearly, it affected the training progress of each action vector independently).



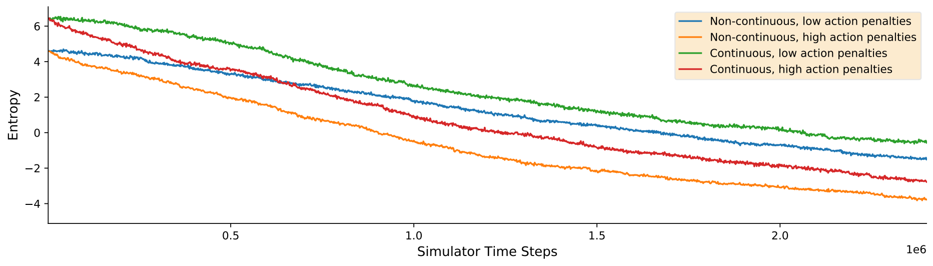
(a) Average return per epoch.



(b) Average V-values.



(c) Critic objective.



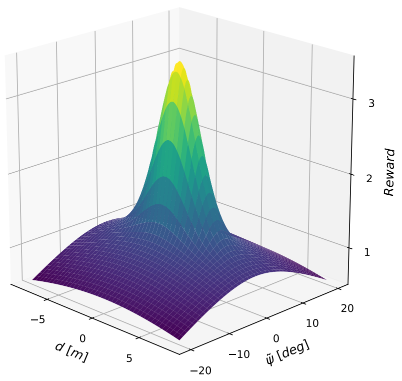
(d) Entropy.

Figure D.3: Training results from the different action vector proposals.

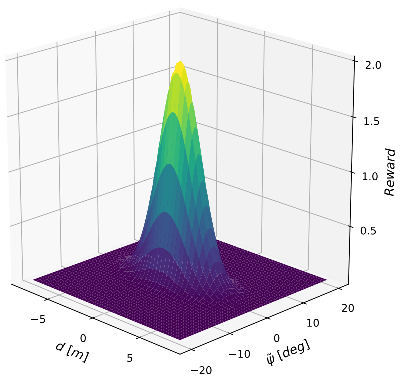
Appendix E

Reward Function Components

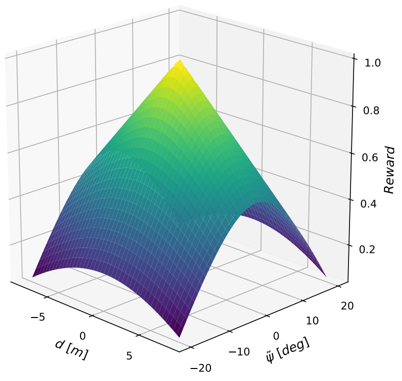
In the following, all components of the reward function for R_η is shown as explained in Section 5.4, using $c_{gauss} = 2.0$, $\sigma_d = 1.0$, $\sigma_{\tilde{\psi}} = 5.0$, $c_{AS} = 0.1$ and $c_{const} = 0.5$.



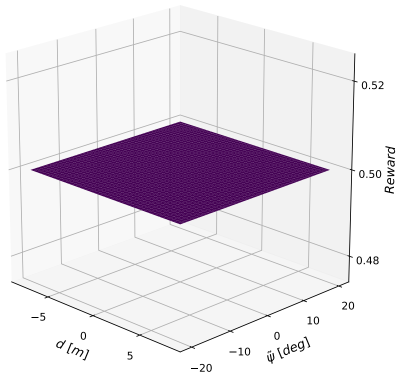
(a) The final reward function.



(b) The single multivariate Gaussian.



(c) Anti-sparsity element.



(d) Constant addition.

Figure E.1: Components of final reward function for pose.

Appendix F

Setpoint Test for Time Horizon

This plot shows the test of the existing control system, using the existing PID and IPI thrust allocation on a challenging DP-task, used as baseline for choosing episode lengths for the training of the DRL model. The set point was set to $[6\ m, 6\ m, -45°]$ away from initial pose, and was set back to $[0\ m, 0\ m, 0°]$ after 90 seconds. It was observed that it took approximately 60 seconds before the deviation from the set point change was eliminated, setting a lower bound for the DRL method's time horizon.

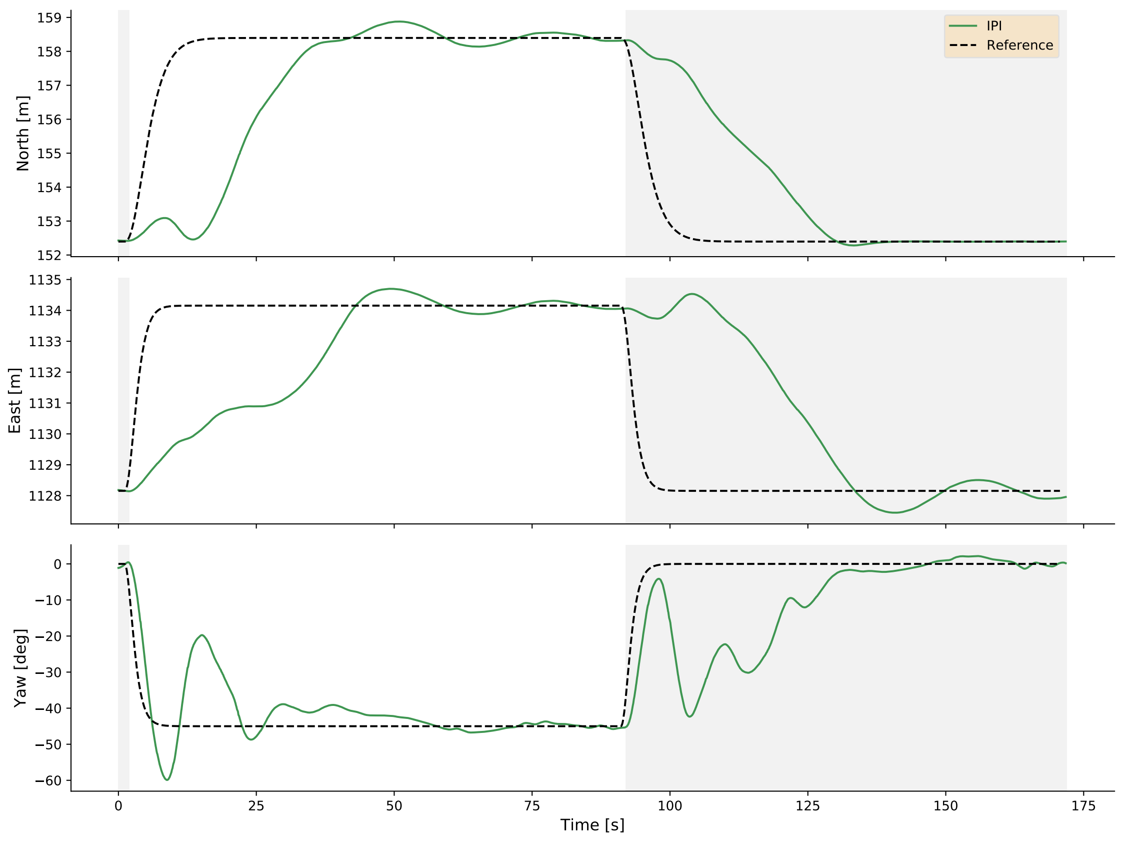


Figure F.1: Pose of setpoint test for DRL time horizon, using the setpoint [+6, +6, -45]

Appendix G

Draft to Ocean Engineering

This appendix presents the publication planned for submission to the International Journal of research and development, Ocean Engineering. Due to the COVID-19 pandemic, conferences was difficult to attend. The date of submission was not yet decided upon the delivery of this thesis.

Dynamic Positioning using Deep Reinforcement Learning

Simen Øvereng[1], Dong Trong Nguyen[1,2], and Geir Hamre[2].

[1] Department of Marine Technology, NTNU. 7491 Trondheim, Norway.
[2] DNVGL. Sluppenvegen 19, 7037 Trondheim, Norway.

Abstract - This paper demonstrates the implementation and performance testing of a Deep Reinforcement Learning based control scheme used for Dynamic Positioning of a marine surface vessel. The control scheme was trained on a digital twin in simulation without any prior knowledge of the system dynamics, using the Proximal Policy Optimization learning algorithm. By using a multivariate Gaussian reward function for rewarding small body-frame errors between the vessel and the setpoint, in addition to encourage small actuator outputs, the system displayed good positional accuracy while being energy efficient when compared to classic methods in simulation. The system was also directly employed to a physical ship model, and despite some hardware issues, the Dynamic Positioning task was solved efficiently.

1 Introduction

Today's methods for solving Dynamic Positioning (DP) often consist of a control system which includes a motion control law, and a thrust allocation (TA) method. The motion controller calculates the desired forces and moments $\boldsymbol{\tau}_d$ to put on the vessel in order to move it towards a setpoint, while the TA translates these forces and moments into thruster commands, resulting in the actual forces $\boldsymbol{\tau}$ on the vessel. In commercial systems, the motion controller usually contains the combination of a feedforward and a feedback control law [1]. The feedforward uses the states from a reference filter in combination with the modeled system dynamics in order to calculate the desired forces, while the feedback control law corrects for any deviations. The goal of the TA method is to output actuator forces which are as close to the desired forces as possible, that is minimizing $\boldsymbol{\tau} - \boldsymbol{\tau}_d$. Therefore, TA methods are typically based on optimization [2], which for complex actuator setups and objective functions can be computationally demanding to solve online.

For TA, the application of an extended thrust formulation for calculating a pseudoinverse of the linear actuator model has become fairly standard in the industry [3; 4], while a variety of Quadratic Programming (QP) formulations has been displayed [2; 5]. Also Model Predictive Control (MPC) has become a viable option for the allocation of actuator commands [6; 7], while also being applicable for combining the motion controller and the TA into one entity [8].

Among newer methods, Reinforcement Learning (RL) algorithms has been used with promising results in various path-following and stationkeeping tasks, for dynamic systems ranging from aerial vehicles [9; 10] to underwater vehicles [11; 12] and surface vessels [13; 14; 15]. These methods were all results of combining RL with Deep Neural Networks (DNN), giving rise to Deep Reinforcement Learning (DRL).

This paper presents work on utilizing the Proximal Policy Optimization (PPO) [16] algorithm for training a DRL-based model for solving DP of a marine surface vessel. It also presents results from simulations and a sea trial to demonstrate its performance. Section 2 gives a description of the DP task, the vessel model, and DRL, while Section 3 explains the development of the DRL framework. Section 4 displays results from both simulations and the sea trial, while Section 5 concludes the work.

2 System Description

2.1 Dynamic Positioning

Dynamic Positioning is concerned with maintaining a vessel's position and heading while using a computer program to control the vessel's actuators. To analyze the vessel's motions, the geographical reference frame North-East-Down (NED) and the body-fixed reference frame was used. NED is chosen as a tangent plane to the surface of the earth, and positions within the frame is denoted (x_n, y_n, z_n), where the x_n-axis points towards true North, the y_n-axis points East and the z_n-axis points downwards. The body-fixed reference frame is denoted (x_b, y_b, z_b), and as defined in this paper, the positive direction was defined for the x_b-axis to point in the forward direction of the vessel, the y_b-axis towards starboard, the z_b-axis downwards, with the origin placed in the Center of Gravity (CG) of the vessel.

A control plant model (CPM) is a simplified mathematical model of the vessel in the relevant DOFs for the control system being designed. When neglecting roll, pitch and heave motions for low-speed DP applications, the CPM consists of surge, sway and yaw. The motion can be described in the body-frame through the vessels' pose $\boldsymbol{\eta}^b = [x, y, \psi]$ and velocity $\boldsymbol{\nu}^b = [u, v, \dot{\psi}]$[1]. The non-linear equations of motion can be presented as in Equation (1), where $\boldsymbol{C}(\boldsymbol{\nu}) \in \mathbb{R}^{3x3}$ is the Coriolis and centripetal matrix, $\boldsymbol{D}(\boldsymbol{\nu}) \in \mathbb{R}^{3x3}$ is the damping matrix, $\boldsymbol{M} \in \mathbb{R}^{3x3}$ is the system inertia, consisting of the added mass and the rigid body inertia. $\boldsymbol{\tau}_i$ represents the different forces acting on the system, $\boldsymbol{\eta}^n$ is the NED-frame position, while $\boldsymbol{\nu}^b$ is the vessels' body-frame velocity. The numeric values of these matrices for ReVolt were found experimentally in [17]. Due to negligible roll and pitch motions during DP, the kinetic equation of motions is restricted to rotation about the z-axis such that the rotation matrix used to transform velocities from the body-frame ($\boldsymbol{\nu}^b$) to the NED-frame ($\dot{\boldsymbol{\eta}}^n$) becomes as in Equation (2).

$$\begin{aligned} \dot{\boldsymbol{\eta}}^n &= \boldsymbol{R}(\psi)\boldsymbol{\nu}^b, \\ \boldsymbol{M}\dot{\boldsymbol{\nu}}^b + \boldsymbol{C}(\boldsymbol{\nu}^b)\boldsymbol{\nu}^b + \boldsymbol{D}(\boldsymbol{\nu}^b)\boldsymbol{\nu}^b &= \sum_i \boldsymbol{\tau}_i. \end{aligned} \tag{1}$$

$$\boldsymbol{R}(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

The body-frame errors represents the deviation between the vessel's current pose and the desired pose, and a DP system should work to eliminate these deviations. By taking the errors in the NED-frame, $\tilde{\boldsymbol{\eta}}^n = \boldsymbol{\eta}^n - \boldsymbol{\eta}^n_d = [\tilde{N}, \tilde{E}, \tilde{\psi}]^\top$, the body-frame errors are calculated as shown in Equation (3a) and Equation (3b). Figure 1 gives a visual representation of the errors.

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix}^\top \begin{bmatrix} \tilde{N} \\ \tilde{E} \end{bmatrix}. \tag{3a}$$

$$\tilde{\psi} = \psi - \psi_d. \tag{3b}$$

[1]To avoid confusion with a *reward* which is used widely in Reinforcement Learning terminology, the yaw rate is denoted $\dot{\psi}$ instead of r in this work.

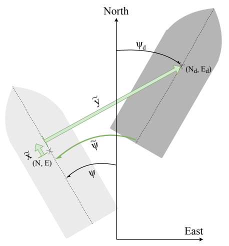


Figure 1: A display of body-frame error (green arrows), showing desired pose in dark gray.

2.2 The ReVolt Ship Model

The ReVolt ship model was used as demonstration platform. It is a 3 meter long ship model, representing a 1:20 scale model of the 60 meter long ReVolt ship concept [18], for which a digital twin has been developed for running simulations in DNVGL's CyberSea simulation environment. The physical model runs a Tank-720 computer on board powered by two 12V batteries, using the Robot Operating System (ROS) for communication between the sensors, software, and the thruster hardware, being three azimuth thrusters, positioned as shown in Figure 2.

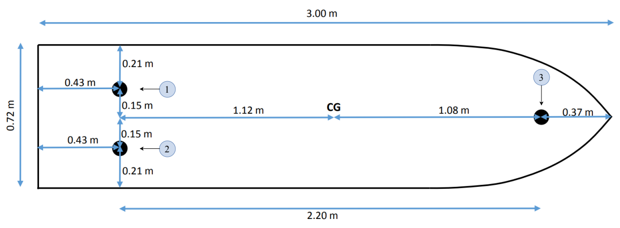


Figure 2: ReVolt model ship illustration.

2.3 Reinforcement Learning

In RL, it it usually assumed that the environment and the interactions with it can be described as a Markov Decision Process (MDP) [19]. An MDP consists of a set of states defining the state space; $s \in \mathcal{S}$, a set of actions defining the action space; $a \in \mathcal{A}$, a probability matrix relating the selection of an action a in a certain state s and ending up in a new state s'; $P \in \mathcal{P}(s'|s, a)$, and a *scalar* reward function that weights the reward of taking action a in state s; $r \in \mathcal{R}(s, a)$. The goal of the RL algorithm (or the *agent*) is to find an optimal policy π^* which maximizes the rewards over time. Letting the return of a certain time period following time t be denoted G_t, a common formulation is the *infinite-horizon undiscounted return*:

$$G_t := r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+(k+1)}, \tag{4}$$

where $0 \leq \gamma \leq 1$ is the discount rate, a weighting factor between rewards accumulated immediately, and reward accumulated in future time steps. Since the RL agent does not have a perfect model of the world, the task is to maximize the *expected* return. An objective function $J(\pi)$ to be maximized can be formulated as in Equation (5). It expresses the expected return over the trajectories τ of state-action pairs, following the action-selection of the agent's policy π, interacting with the environment by observing a state, taking an action, and getting a reward.

$$J(\pi) = \mathop{\mathbb{E}}_{\tau \sim \pi} \Big[G(\tau) \Big] \tag{5}$$

Commonly used in RL algorithms is the notion of a value function, describing the estimated value of the return by starting in state s_t and following the actions taken by the policy π into the future,

$$V(s_t) := \mathop{\mathbb{E}}_{\tau \sim \pi} [G(\tau) | s_0 = s_t]. \tag{6}$$

Various RL algorithms combines the policy and value function in different ways in order to approximate the optimal policy. By the inclusion of DNNs for approximation, RL algorithms' expressive powers has increased when solving MDPs with continuous state- and action spaces. Where actor-only algorithms only approximate the policy by using a DNN, actor-critic algorithms uses DNNs for approximating both the policy, called the actor, *and* the value function estimator, called the critic. In this work, the actor's output, being an action when given a state s_t, is denoted $\pi_{\theta_a}(s_t)$, while the value function estimate from the critic is denoted $\hat{V}_{\theta_c}(s_t)$.

The algorithm used in this paper is called Proximal Policy Optimization (PPO) by Schulman et al. [16] and uses the actor-critic structure to learn the policy with the help of a value function. Through PPO, Schulman et al. presented an objective function for training the actor's DNN in a way that was data efficient, robust with respect to hyperparameter changes, and easy to implement. It was based on maximizing the expected return while limiting the magnitude of the updates to the actor's DNN by using the idea of trust regions. PPO was built from the conservative estimate of the expected return of a state-action pair (s_t, a_t) as shown in Equation (7).

$$\mathbb{E}_t \Big[G_t \Big] \approx \mathbb{E}_t \Big[\bar{r}(\theta_a)_t \hat{A}_t \Big]. \tag{7}$$

Here, $\bar{r}(\theta_a)_t$ is the probability ratio between taking an action in a current state with the old network parameters, $\theta_{a,old}$, and the new ones, θ_a. Note that $\bar{r}(\theta_a) \geq 0 \;\forall\; \theta$ and $\bar{r}(\theta_{a,old})_t = 1.0$.

$$\bar{r}(\theta_a)_t = \frac{\pi_{\theta_a}(a_t|s_t)}{\pi_{\theta_{a,old}}(a_t|s_t)}. \tag{8}$$

$\hat{A}_t$ denotes the estimate of the *advantage function*, which represents how much better it was to select action a_t in state s_t, compared to the average return of all actions available in state s_t. Generalized Advantage Estimation [20] was used for estimating the advantage as shown in Equation (9a), where the critic was used through calculating $\delta_t^{\hat{V}_{\theta_c}}$.

$$\hat{A}_t = \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i}^{\hat{V}_{\theta_c}}, \tag{9a}$$

$$\delta_t^{\hat{V}_{\theta_c}} = r_t + \gamma \hat{V}_{\theta_c}(s_{t+1}) - \hat{V}_{\theta_c}(s_t). \tag{9b}$$

By limiting the magnitude of $\bar{r}(\theta_a)_t$ per parameter update, we can perform improvements on the objective function while being careful of not making too large parameter updates in θ_a-space which might reduce the performance in the policy's output space, π_{θ_a}. The proposed objective function from Schulman et al. included clipping the size of $\bar{r}(\theta_a)_t$ in order to enforce a limitation of the probability ratio so that $\bar{r}(\theta_a)_t \in [1 - \epsilon, 1 + \epsilon]$, giving their *clipped surrogate objective* function as shown in Equation (10).

$$J(\theta_a)_t = \min \Big(\bar{r}(\theta_a)_t \hat{A}_t,\ \text{clip}\left(\bar{r}(\theta_a)_t, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \Big). \tag{10}$$

The critic network performed parameter updates by attempting to minimize the deviations between its value function predictions vs. experienced values of being in a state by using the Mean Squared Error function. Thus the update rule was as showed in Equation (11) for the critic. The actor attempted to maximize the expected reward, and therefore used the update rule as in Equation (12).

$$\theta_{c,k+1} \leftarrow \underset{\theta_c}{\text{argmin}} \left\{ \frac{1}{T} \sum_{t=1}^{T} \Big(\hat{V}_{\theta_{c,k}}(s_t) - G_t \Big)^2 \right\}, \tag{11}$$

$$\theta_{a,k+1} \leftarrow \underset{\theta_a}{\text{argmax}} \left\{ \frac{1}{T} \sum_{t=1}^{T} J(\theta_a)_t \right\}. \tag{12}$$

3 Implementation

The state vector used as input to the actor and the critic networks was developed with the goal of DP in mind, thus adding the body-frame errors as the first components. Additionally, it was beneficial for the agent to have access to the body-frame velocities u, v and $\dot{\psi}$, which was added next. In addition, the previous time step's thruster commands (as a fraction between -1 and 1) was added to the state vector in order to increase the agent's ability to minimize the penalty put on the size of the output from the network's thrust-components. The quantities in the state vector was extracted straight from the simulator during training, and from the existing observer and reference filter in ROS during testing. The state vector is shown in Equation (13).

$$\boldsymbol{s}_t = [\tilde{x}_t,\ \tilde{y}_t,\ \tilde{\psi}_t,\ \tilde{u}_t,\ \tilde{v}_t,\ \tilde{\dot{\psi}}_t, \overline{n}_{b,t-1},\ \overline{n}_{p,t-1},\ \overline{n}_{s,t-1}]^\top. \tag{13}$$

The action vector formulation was found to increase the rate of learning if letting the policy predict the sines ($s(\cdot)$) and cosines ($c(\cdot)$) of the angles of the port (p) and starboard (s) thrusters instead of the raw angles directly. Empirically, the DP ability of the vessel improved by setting the bow thruster to a fixed angle of 90° [17; 21], so the bow thruster's angle was not included in the action vector. The resulting action vector was as shown in Equation (14).

$$\boldsymbol{a}_t = [n_{b,t},\ n_{p,t},\ n_{s,t}, s(\hat{\alpha}_{p,t}),\ c(\hat{\alpha}_{p,t}),\ s(\hat{\alpha}_{s,t}),\ c(\hat{\alpha}_{s,t})]^\top. \tag{14}$$

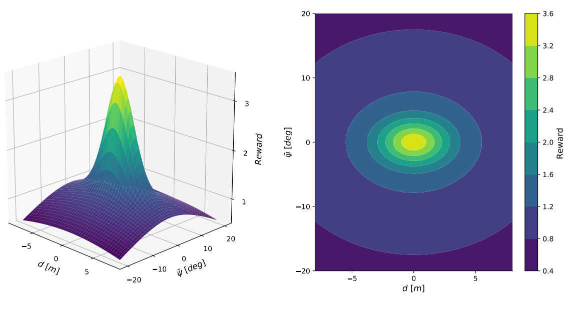
The shape of the reward function for rewarding small deviations from the setpoint in terms of euclidean distance, $d = \sqrt{\tilde{x}^2 + \tilde{y}^2}$, and heading deviation, $\tilde{\psi}$, was inspired by the shape out the output layer of the policy network itself, being a multivariate Gaussian function with covariance matrix Σ:

$$\Sigma = \operatorname{diag}([\sigma_d^2, \sigma_{\tilde{\psi}}^2]). \tag{15a}$$

$$R_{gauss} = c_{gauss} \exp\left(-\frac{1}{2}\begin{bmatrix} d & \tilde{\psi}\end{bmatrix} \Sigma^{-1} \begin{bmatrix} d \\ \tilde{\psi}\end{bmatrix}\right). \tag{15b}$$

To reduce the sparsity of the function since little reward was given when far away from the setpoint, a function was added by using a distance measurement ϕ in the $(d, \tilde{\psi})$-space for guiding the agent's learning process, in addition to a constant as shown in Equation (16). The resulting reward function for pose was therefore as shown in Figure 3.

$$R_{\text{AS}} = \max\left(0.0, (1 - c_{AS}\ \phi)\right) + c_{const}. \tag{16}$$



(a) 3D plot. **(b)** Contour plot.

Figure 3: Plots of $R_{gauss} + R_{AS}$ using $c_{gauss} = 2.0$, $\sigma_d = 1.0$, $\sigma_{\tilde{\psi}} = 5.0$, $c_{AS} = 0.1$ and $c_{const} = 0.5$.

To avoid the agent making overshoots of the setpoints, penalties on the velocities was added as

$$R_{vel} = -\sqrt{c_u(\tilde{u})^2 + c_v(\tilde{v})^2 + c_{\tilde{\dot{\psi}}}(\tilde{\dot{\psi}})^2}. \tag{17}$$

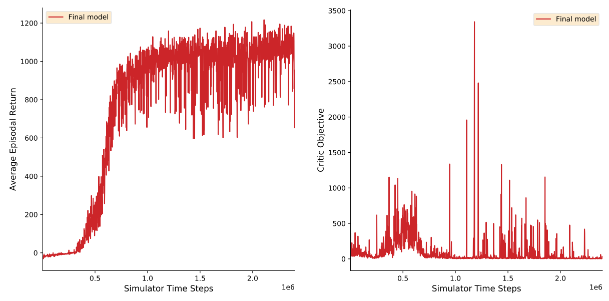
In order to achieve energy efficiency in addition to low wear and tear on the actuators, small penalties were put on the magnitude of the commanded thrust, in addition to the derivatives of the commanded actuator outputs as shown in Equation (18),

$$R_{act} = -\sum_{i}^{3} c_{|n|,i}|n_i| + c_{\dot{n},i}\dot{n_i} + c_{\dot{\alpha},i}\dot{\alpha_i}, \tag{18}$$

giving the total reward function as

$$R_{tot} = R_{gauss} + R_{AS} + R_{vel} + R_{act}. \tag{19}$$

The training was done by randomly initializing the pose of the vessel around a setpoint, with the agent's goal being to get to the setpoint by eliminating all body-frame errors according to the reward function. Parameter updates to the actor and critic networks were performed with state-action pairs from 2.4M time steps in total, using an implementation of the PPO algorithm based on OpenAI's repository[2]. The actor-critic architecture was set up using independent DNNs for the actor and the critic, both using $\boldsymbol{s}_t \in \mathbb{R}^9$ as input. The networks used 3 fully connected hidden layers with 80 neurons, and the output of the actor at time step t was $\boldsymbol{a}_t \in \mathbb{R}^7$, while the output from the critic was $V_{\theta_c}(s_t) \in \mathbb{R}^1$. Both networks used the leaky ReLU activation functions for non-linearities, and the ADAM optimizer for parameter updates. The average return obtained of the actor per episode (having an optimal value of 1400) and the value of the objective function of the critic (having an optimal value of 0) is shown in Figure 4. Note that the average episodal return fluctuated a lot due to the random initialization of the vessel's state between each trajectory within each episode, but steadily improved on average.



(a) Average return per episode. **(b)** Critic objective function.

Figure 4: Resulting training plots using the PPO algorithm.

4 Results and Discussion

A four corner test was performed in simulation while enabling current forces to observe the controllers' robustness external disturbances. The current was irrotational and non-fluctuating with velocity $\nu_c = 0.2\ m/s$ and $\beta_c = 135°$. None of the methods were given any information about the environmental loads, and the tests were started when the vessel had been standing still for 30 seconds. During simulations, an integral effect was added to the state vector of the DRL method, augmenting the body-frame error pose in the state vector according to $\hat{\boldsymbol{\eta}}_t = \boldsymbol{\eta}_t + \hat{\boldsymbol{\eta}}_{ss,t}$, where the discrete integration was calculated according to $\hat{\boldsymbol{\eta}}_{ss,t} = \hat{\boldsymbol{\eta}}_{ss,t-1} + \Delta t k \boldsymbol{\eta}_t$. Δt is the time step, and k is a coefficient deciding the speed of the integral accumulation. A windup guard was also added to prevent overshoots. During the sea trial, the same four corner test coordinates were used.

The simulation was performed using the DRL method with and without integral effect (denoted RLI and RL in the plots, respectively). It was also benchmarked against two other methods, the first being a motion control law consisting of feedforward + a PID feedback controller developed in [17], combined with a pseudoinverse-based TA (denoted IPI in the plots) and a Quadratic Programming TA (denoted as QP). Due to time limitations from hardware debugging, only the RL and the IPI methods were tested during the sea trial.

[2]`https://spinningup.openai.com/`

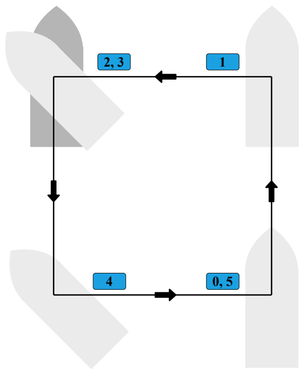


Figure 5: Four corner pose.

Time [s]	N [m]	E [m]	ψ [deg]
0-10	0	0	0
10-80	5	0	0
80-150	5	-5	0
150-190	5	-5	-45
190-270	0	-5	-45
270-350	0	0	0

Table 1: Coordinates.

To evaluate performance, the Integral of Absolute Error (IAE) metric was used as a measurement of how accurate the DP system was in terms of reducing the body-frame error between the current pose and the desired pose. The measurement $\tilde{\bar{\eta}}$ was used, where the deviation in North, East and yaw was normalized for making the IAE a dimensionless number, dividing the respective deviations with $[5\ m,\ 5\ m,\ 25°]$, meaning that a 1 meter deviation was weighted equally as a 5 degree deviation.

$$IAE(t) = \int_{\tau=0}^{t} \sqrt{\tilde{\bar{\eta}}^\top \tilde{\bar{\eta}}}\ d\tau. \tag{20}$$

The energy usage was calculated by integrating the power for each of the commanded RPS-signals. The power as a function of a propeller's revolution per second (RPS), n, is given as shown under the integral in Equation (21), where ρ is the sea water density, D is the propeller diameters, and K_Q is the propeller torque coefficient.

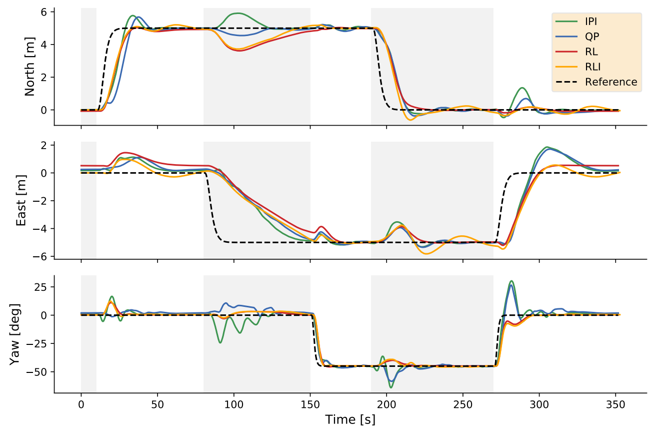
$$W(t) = \int_{\tau=0}^{t} 2\pi\rho K_Q D^5 \operatorname{sgn}(n(\tau))n(\tau)^3 d\tau. \tag{21}$$

4.1 Simulation

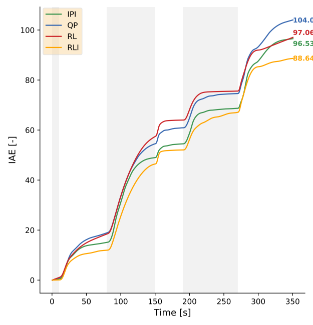
The simulation results are displayed in Figure 6. It was observed that only the DRL method with integral effect was able to totally remove the steady state body-frame error in *all* setpoints, while the classic methods displayed a slight steady state deviation in some setpoints. The DRL method without integral effect struggled with removing steady-state errors in sway. The classic methods and the DRL method with integral effect showed some overshoots of the setpoints, but only the classic methods showed oscillations (particularly in yaw) before settling on a setpoint. This lead to questioning the quality of the existing implementation of the motion control law on ReVolt, which both the IPI and the QP allocation depended on. It was believed that this came from the fact that the motion controller was probably tuned on board the physical model of ReVolt in previous work, thus resulting in sub-optimal performance when using the digital twin. The QP allocation resulted in having the highest values of IAE and energy usage. The DRL method with integral effect resulted in the lowest values of both metrics, as it resulted in 15% lower IAE and 38% less energy usage compared to the QP allocation, while also having 8% lower IAE and 33% less energy usage compared to the IPI allocation.

4.2 Sea Trial

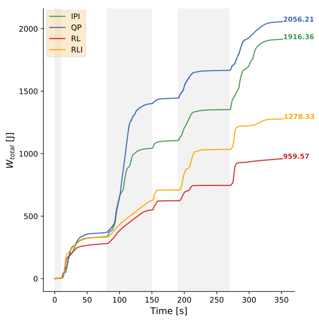
During the sea trial, a measurement was taken by the DNN to perform a feedforward pass on the computer on board the ship model. The maximum time recorded was 0.015 seconds, indicating that control using DNN based approaches is not likely to suffer from computation time.



(a) Pose.



(b) IAE.

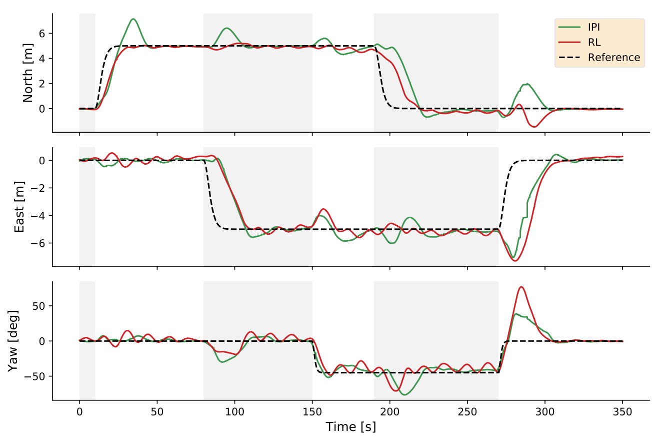


(c) Energy usage.

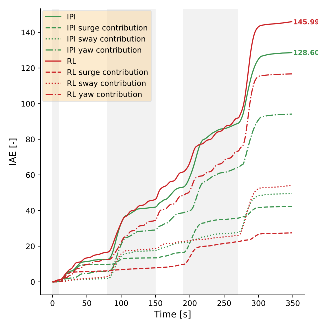
Figure 6: Results of four corner test during simulations.

On the ship model, it was found that the bow thruster's propeller only rotated when being commanded signals at 50% or above, without any solutions being found at the spot. In order to get any results, it was decided to multiply the commanded bow thrust signals with 2.5 in order to get the signals large enough for the bow thruster to react more often than not. This came at the cost of an expected overshoot and/or oscillation in sway and yaw due to the bow thruster output being much larger than outputted by the DP algorithms. It was also found that the bow thruster's propeller was unsymmetrical, yielding a maximum force when rotating positively (exerting forces towards starboard when locked to 90°) of 2.4 times the size of the maximum when rotating the other way. The DRL method had however been trained with symmetrical bow thruster parameters. Due to time constraints following the debugging, it was only time to perform the four corner test with some of the previously mentioned methods, in which it was chosen to test the DRL method in it's "purest" form, meaning using no integral effect on the state vector, and the motion controller + the pseudoinverse based thrust allocation method as a baseline.

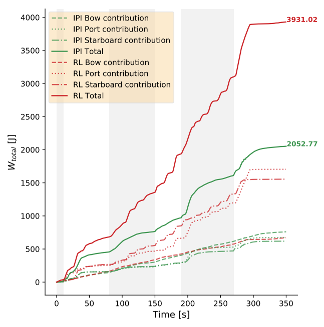
The results from the four corner test in Figure 7 showed that the DRL method was able to reach all setpoints in approximately the same time in the sea trial as in the simulation, and reached the setpoints without much overshoot except when the vessel was traveling with sway motions (at 80 and 270 seconds), which was also the case for the baseline method. This was due to the bow thruster issue, being more severe for the DRL method which had learned to use the bow thruster more extensively than classic methods in simulation in order to be energy efficient. The oscillatory movements was both coming from the bow thruster issue, in addition to that the stern thrusters had to compensate for in order to follow the reference signal. Due to this issue, the resulting IAE value was now larger when compared to the baseline controller. The increased use of all thrusters also made the DRL method significantly less energy effective, mainly due to the compensating stern thrusters, both compared to in simulation and to the baseline controller in the sea trial.



(a) Pose.



(b) IAE.



(c) Energy usage.

Figure 7: Results of four corner test during sea trial.

5 Conclusion

This paper presented the implementation of the PPO algorithm for developing a DRL control scheme for applications to low speed tasks such as DP. The learning process benefited greatly from including prediction of sines and cosines of the stern thrusters' angles, and by using a multivariate, Gaussian reward function with an additional element to combat sparsity.

Tests were done through a four corner test with an ocean current in simulation, where the DRL method with integral effect was able to obtain better positional accuracy when compared to two classic methods for motion control and TA. Especially, the DRL method proved more energy efficient. Even though the quality of the comparisons can be discussed due to the seemingly sub-optimal implementation of the existing motion control law, it could be proposed that the DRL method is a strong contender when looking for new methods for station-keeping and low speed maneuvering in a way that encapsulates both the motion control and TA tasks. The DRL method also showed good positional accuracy when employing it to a physical model in a sea trial, albeit being more oscillatory due to a bow thruster issue which resulted in increased stern thruster usage to compensate, also reducing the energy efficiency. The result was however deemed excellent considering that the DRL method was trained in simulation with perfect information about the vessel's state, zero environmental forces, and no thruster issues, which was not the case during the sea trial. In addition, due to the use of a DNN, the computational time for the control system was negligible, combating the problem of computational complexity which classic optimization based methods might face.

The DRL method's advantage is that it is able to learn details from the vessel's dynamics through trial and error, while classic methods depends on mathematical formulations including estimations of the vessel's dynamics which can be inaccurate. The downside of the method is that it could be less robust to changes to the dynamic system, and it is hard to give performance guarantees of the DNNs, requiring rigorous empirical tests. In addition, the training process itself might be time consuming, whilst classic methods could be reconfigured with only a few lines of code and some tuning.

The work concludes that DRL's potential for accurate and energy efficient control is realizable, and suggests that further work should be focused on providing stability guarantees of DRL systems, and to explore *transfer learning* as a method for retraining a DRL model from simulations to the real life model in order to adapt to the differences in the system dynamics and actuator characteristics.

6 Acknowledgements

This paper summarizes the work of a master's thesis [21], which was written in collaboration with DNVGL. The work was not funded, and the author had no conflicts of interest. All implementations can be found at `https://github.com/simensov/ml4ca`.

References

[1] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control.* John Wiley & Sons, 2011.

[2] T. A. Johansen and T. I. Fossen, "Control allocation - a survey," *Automatica*, vol. 46, pp. 1087–1103, 2013.

[3] O. J. Sørdalen, "Optimal thrust allocation for marine vessels," *Control Eng. Practice, Vol. 5. No. 9*, pp. 1223–1231, 1997.

[4] N. A. Jenssen and B. Realfsen, ""power optimal thruster allocation "," 2006. Kongsberg Maritime AS, `https://pdfs.semanticscholar.org/064e/1b38204f5d855266a4eeb4af7b705a9bb2b9.pdf` (visited on 2019/11/18).

[5] J. Leavitt, ""optimal thrust allocation in dp systems"," 2008. Dynamic Positioning Committee, `https://dynamic-positioning.com/proceedings/dp2008/thrusters_leavitt_pp.pdf` (visited on 2019/09/17).

[6] C. Vermillion, J. Sun, and K. Butts, "Model predictive control allocation for overactuated systems - stability and performance," in *IEEE Conference on Decision and Control*, vol. 47, pp. 1251 – 1256, 2007.

[7] M. Naderi, A. Khaki Sedigh, and T. Johansen, "Guaranteed feasible control allocation using model predictive control," *Control Theory Technol*, vol. 17, p. 252–264, 2019.

[8] A. Veksler, T. A. Johansen, F. Borrelli, and B. Realfsen, "Dynamic positioning with model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 24, pp. 1340–1353, 2016.

[9] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, 2018.

[10] E. Bohn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization," *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019.

[11] E. F. Kjærnli, "Deep reinforcement learning based controllers in underwater robotics," master's thesis, NTNU, 2018. `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2615067`.

[12] K. B. Knudsen, "Deep learning for station keeping of auvs," master's thesis, NTNU, 2019. `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2622920`.

[13] A. B. Martinsen and A. Lekkas, "Straight-path following for underactuated marine vessels using deep reinforcement learning," vol. 51, pp. 329–334, 09 2018.

[14] R. Skulstad, G. Li, H. Zhang, and T. I. Fossen, "A neural network approach to control allocation of ships for dynamic positioning," *IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS*, vol. 51, pp. 128 – 133, 2018.

[15] E.-L. H. Rørvik, "Automatic docking of an autonomous surface vessel," master's thesis, NTNU, 2020. `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2656724`.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[17] H. Alfheim and K. Muggerud, "Dynamic positioning of the revolt model-scale ship," master's thesis, NTNU, 2016. `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2595418`.

[18] DNVGL, ""the revolt - a new inspirational ship concept"." `https://www.dnvgl.com/technology-innovation/revolt/index.html` (visited on 2019/10/01).

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, 2nd edition.* MIT Press, 2017. `http://incompleteideas.net/book/the-book.html`.

[20] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015.

[21] S. S. Øvereng, "Energy efficient dynamic positioning using deep reinforcement learning," master's thesis, NTNU, 2020. Manuscript submitted for publication.