Ella-Lovise Hammervold Rørvik

Automatic Docking of an Autonomous Surface Vessel

Developed using Deep Reinforcement Learning and analysed with Explainable AI

Master's thesis in Cybernetics and Robotics

Supervisor: Anastasios Lekkas

February 2020

Master's thesis

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

NTNU
Norwegian University of Science and Technology

Ella-Lovise Hammervold Rørvik

Automatic Docking of an Autonomous Surface Vessel

Developed using Deep Reinforcement Learning and analysed with Explainable AI

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
February 2020

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

NTNU
Norwegian University of
Science and Technology

Abstract

Docking is considered a complex, high-risk process where a vessel must follow the rules of the harbour, avoid both static and dynamic obstacles, reach the desired docking point, and hold its position while awaiting fastening to the dock. Autonomous docking is a vital part of achieving ship autonomy, and has been researched since the 1990s. It has proven to be a difficult task, due to significantly reduced manoeuvrability during docking and nonlinearities, to mention some of the more essential challenges. Techniques such as optimal control theory and fuzzy control logic have been proposed to solve the task of autonomous docking. These methods have produced noteworthy results but also have some drawbacks. One prominent example is the need for reliable and good mathematical models, coping with inherent nonlinearities and varying conditions (including speed, weather etc.). Other limitations are dependencies on lower-level controllers, and for some methods high computational requirements during operations.

In recent years, a data-based field of study called deep reinforcement learning (DRL) has successfully been applied to some continuous control problems of marine vessels in simulations. Deep reinforcement learning optimises decision-making problems through exploring actions in an environment and receiving feedback on performance. Recent developments in DRL have led to successful solutions of previously unsolved tasks by otherwise promising data-based approaches. Deep RL is, for instance, able to utilise sensor information to create functional control laws and end-to-end solutions. Using DRL to create docking models helps towards avoiding several of the drawbacks of previous methods. A DRL-based docking model can handle uncertainties in the models of the marine vessel and harbour and utilise direct sensor information. Additionally, Deep RL-based models may benefit from having access to manoeuvring data from the ship master during learning or retraining, but the method does not need it.

The main objective of this thesis is to explore the possibility of using deep reinforcement learning (DRL) to create an end-to-end harbour docking system for a 3 degrees-of-freedom (3-DOF) fully-actuated autonomous surface vessel, and analyse its performance and explainability. The DRL-based docking model is created through a progressive methodology, first

solving tasks such as berthing and target tracking, before combining these solutions into an end-to-end docking model. The docking model can control a vessel efficiently from just outside the harbour to a berth, and hold its position once at the berth.

The end-to-end DRL-controller uses information about the vessel's position relative to the harbour, to avoid collisions and can (up to a certain extent) handle unforeseen ocean currents. The DRL agent solves all these tasks by controlling both thruster angles and forces. The DRL-based control law is, therefore, able to both replace thruster allocation, the traditional controllers and guidance systems.

The DRL-based model was analysed using Shapley additive explanation (SHAP), a technique from the field of explainable AI (XAI), to get insight and understanding of the model. Shapley additive explanation was used to find the states' relative contributions to the agent's selection of thrust, and thereby provide insight into certain aspects of the DRL-agent's reasoning. The reasoning was analysed both from general point of views, and for given events at specific moments. It was demonstrated that such insight, provided by SHAP, could be used to improve the DRL-agent.

Two different DRL-algorithms were explored, namely proximal policy optimization (PPO) and deep deterministic policy gradient (DDPG). It was found that PPO was easier to adapt to the docking phases, where PPO was equally or more successful on all encountered aspects of docking.

The result of this thesis shows that DRL can be useful to solve several aspects and the entire docking problem, creating models with high accuracy and efficient trajectories. The proposed use of SHAP for analysing the behaviour of DRL-based controllers shows promising results of gaining better insight. It consequently makes it easier improve solutions and increases the trust of DRL-based models. Even though DRL-based controllers were found using a simplified simulator, the methodology can be extended to real systems.

Sammendrag

Automatisk dokking er viktig for å realisere autonom skipsfart, og har blitt forsket på siden 1990-tallet. Dokking ses på som en kompleks høyrisikoprosess, der et fartøy trenger å følge havnens regler, unngå både statiske og dynamiske hindringer, nå det ønskede dokkingspunktet og holde sin posisjon mens det venter på å bli festet på en trygg måte. Dette har vist seg å være en vanskelig oppgave, blant annet på grunn av betydelig redusert manøvrerbarhet under dokking og faren for kollisjoner og andre uhell.

Teknikker som optimal kontrollteori og fuzzy-logikk (eng. fuzzy logic) har blitt foreslått for å løse oppgaven med automatisk dokking. Disse metodene har noen ulemper. Et viktig eksempel er behovet for pålitelige og gode matematiske modeller. Utfordringer med å lage gode matematiske modeller ligger i å håndtere iboende ulinearitеter og varierende forhold under dokking av et fartøy, slik som vekslende hastigheter og værforhold. Andre utfordringer med tradisjonelle metoder inkluderer blant annet avhengigheter av lavnivå kontrollsystemer og meget høye beregningskrav i kontrollsystemene, om bord på skipene.

De siste årene har et felt innen databaserte tilnærminger, kalt dyp forsterkende læring (eng. deep reinforcement learning, DRL), blitt brukt med gode resultater på endel tilfeller av kontinuerlig styring av fartøyer i simuleringer. Dyp forsterkende læring (DRL) optimaliserer løsninger på beslutningsmessige (eng. decision-making) problemer. Dette skjer ved at læringssystemet selv utforsker handlinger i et miljø, og mottar tilbakemeldinger på det som er oppnådd. Nyere utvikling innen DRL har ført til vellykkede løsninger på tidligere uløste oppgaver sammenlignet med andre lovende databaserte tilnærminger.

Ved å bruke DRL for å styre et fartøy til kai (kalt dokking, fra engelsk «docking») unngås flere av ulempene med tidligere brukte metoder. En DRL-basert modell for dokking kan håndtere usikkerhetene i modellene til både fartøy og havn, og kan koples direkte til styringsorganene (aktuatorene), i en ressurseffektiv ende-til-ende-løsning. I tillegg kan DRL-baserte modeller dra fordel av tilgang til manøvreringsdata fra fartøy under læringen, men metoden krever det ikke. Dyp forsterkende læring (DRL) er også i stand til å bruke sensorinformasjon direkte for å lage funksjonelle styringsregler (eng. functional control laws).

Hovedmålet med denne masteroppgaven er å utforske muligheten for å bruke dyp forsterkende læring (DRL) for å lage et ende-til-ende dokkingssystem for et 3-frihetsgraders (3-DOF) fullstyrt autonomt overflatefartøy, og analysere ytelsen og forklarbarheten til kontrollreglene. Den DRL-baserte dokkingsmodellen er opprettet gjennom en progressiv metodikk, som først løser oppgaver som å legge til kai og målsporing (eng. target tracking), før disse kombineres i en ende-til-ende dokkingmodell. Ende-til-ende dokkingsystemet leder fartøyet på en effektiv måte fra like utenfor havnen helt fram til kai, og holder fartøyet ved den angitte plasseringen ved kaia.

Ende-til-ende DRL-kontrolleren bruker informasjon om fartøyets posisjon i forhold til havna for å unngå kollisjoner, og kan til en viss grad håndtere uforutsette havstrømmer. DRL-agenten løser alle disse oppgavene ved å kontrollere både thrustervinkler og krefter. Den DRL-baserte modellen er derfor i stand til både å erstatte thrusterallokeringen, de tradisjonelle kontrollerne og føringssystemet. Den DRL-baserte modellen ble analysert ved bruk av en tilpassning av Shapley additiv forklaring (eng. Shapley additive explanation, SHAP). Dette er tilpasning av en teknikk fra feltet for forklarbar AI (XAI), og har som formål å skaffe innsikt i og forståelse av DRL-baserte dokkingmodellen. Teknikken presenterer mål på tilstandenes relative bidrag til agentens valg av thrustere, og gir dermed innsikt i enkelte aspekter av DRL-modellenes resonnement. Resonnementet ble analysert både fra generelle synspunkt og for gitte hendelser på bestemte øyeblikk. Det ble vist at slik innsikt fra SHAP kunne brukes til å forbedre DRL-dokkingmodellen.

To forskjellige DRL-algoritmer ble utforsket, proksimal politikkoptimalisering (eng. proximal policy optimisation, PPO) og dyp deterministisk politikkgradient (eng. deterministic policy gradient , DDPG). Det ble vist at PPO fungerte like bra eller bedre enn DDPG for alle læringsaspektene i denne oppgaven rundt dokking.

Dette prosjektet viser at DRL kan være nyttig for å løse dokkingsproblemer, og lage modeller med høy nøyaktighet og effektive baner. Den foreslåtte bruken av SHAP for å analysere atferden til DRL-baserte modeller viser lovende resultater med tanke på å skaffe seg bedre innsikt i resonnementet. Dette gjør det følgelig lettere å forbedre løsningene, og kan øke tilliten til DRL-baserte modeller. Selv om de DRL-baserte kontrollerne ble funnet ved hjelp av en forenklet simulator, kan metodikken utvides til reelle systemer.

Preface

This thesis is a result of my work in the fall of 2019 at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology, as part of the course TTK4900. It was performed under the supervision of Associate Professor Anastasios Lekkas.

The main contribution of this thesis is a progressive methodology to solve the docking problem for a marine surface vehicle, using deep reinforcement learning (DRL), and how to analyse the reasoning of the model, using explainable AI (XAI). The thesis is a continuation of my specialisation project, where path following for a fully-actuated vessel using DRL was studied. In the specialisation project, the deep deterministic policy gradient (DDPG) from Lillicrap et al. [1] was adapted based on an implementation by OpenAI [2]. The dynamics of a container vessel model was also implemented in the specialisation project, based on vessel model presented in Martinsen et al. [3]. The DDPG implementation and vessel model from the specialisation project are reused here but adapted to the new scenario of docking. The main contributions come, however, from the DRL-algorithm proximal policy optimization (PPO), which needed to be added due to poor performance of the DDPG agent in several of the docking problems.

For continuation, relevant information from the specialisation project is included here, with some modifications. These chapters include background and theory material on DRL, artificial neural networks (ANNs), dynamics of surface vessels and implementation of DDPG and the vessel model. A complete list of the material included (with minor modifications) from the specialisation project is Chapter 1 (some paragraphs in Section 1.1), Chapter 2 (Section 2.2-2.3.3.3), Chapter 3 (Sections 3.2.1 and 3.4.1) and Appendix A.1.

This assignment is inspired by Martinsen et al. [3–5]. Martinsen et al. [3] developed an algorithm of autonomous docking using optimal control theory, where their representation of the harbour and vessel as a convex set is reused in this thesis. Martinsen et al. [4, 5] was the inspiration to use Gaussian-shaped reward functions, for some purposes.

Deep RL can solve several complex continuous problems. It does however require extensive computations, and several libraries are therefore used to save computational time. The

methods are implemented in the open source programming language Python, using libraries such as Tensorflow [6], Numpy [7], Matplotlib [8], SHAP [9] and Pandas [10], all available as open source code. The DRL algorithms PPO and DDPG, were both based on an open source implementation by the organisation Spinning Up [2]. They were modified in this thesis, to better correlate with the original algorithm and fit the docking problem better. This includes adding normalisation of state vector, tuning of parameters, adding batch normalisation, adding transferring of weights between DRL agents and changing handling of the action space by scaling and saturation. The report is written in Latex, and most visualisations are produced using the library Matplotlib and the webpage Draw Io [11].

The software was run on a Lenovo Yoga Pro 2 and a Dell workstation provided by NTNU, neither with GPU, using the Linux operating system. The learning processes of the DRL-based controllers needed high computational capacity, witnessed through extensive learning runs, ranging from several hours to several days. In an effort to be able to run quicker, a considerable amount of time was used to adapt the solution to an Intel-based university internal high-performance computing facility. This did, unfortunately not improve the speed of the process, due to difficulties utilising parallelisation successfully in the environment.

I wish to thank supervisor Anastasios Lekkas for invaluable discussions, with deep insight and suggestions of how to develop docking DRL-algorithms, and how they should be tested. I also wish to thank Andreas Bell Martinsen for discussing his docking article [3], and for advice on how to build on his work in developing the proposed autonomous docking system design. In addition I wish to thank deck officer Bjørn Erik Mellomsether and harbour captain Svein Olav Fagerdal for clarifications about harbours in general and specifications of Trondheim harbour.

Acknowledgment

A wholehearted thank you to my supervisor, associate professor Anastasios Lekkas. You provided new insights and creative solutions to any and all challenges during the project in the autumn and winter of 2019-2020. To my fellow students, thank you for caring for me during this work. Last, but not least, a warm hug to my familiy for sticking out with me during long hours of machine learning and writing.

Trondheim, 12. February 2020

Ella-Lovise Hammervold Rørvik

Nomenclature

Acronyms

AI	Artificial intelligence
ANN	Artificial neural network
COLREGS	The international regulation for preventing collision at sea
DDPG	Deep deterministic policy gradient
DL	Deep learning
DOF	Degree of freedom
DP	Dynamic positioning
DRL	Deep reinforcement learning
MDP	Markov decision process
ML	Machine learning
MPC	Model predictive control
NED	North-East-Down frame
POMPDP	Partially observable Markov decision process
ReLU	Rectified linear unit
RL	Reinforcement learning
SGD	Stochastic gradient descent
TD	Temporal difference

Symbols

$\boldsymbol{\eta}$	Position vector for marine vessel
$\boldsymbol{\nu}$	Velocity vector for marine vessel
α	Learning rate
ψ	Heading, yaw
$\mathbf{w}$	Weight vector
$\mathbf{b}$	Bias vector
$J(\boldsymbol{\theta})$	Loss/objective function of parameter vector $\boldsymbol{\theta}$
$Q(s, a)$	Action-value function
$V(s)$	State-value function
$R(s, a)$	Reward-function given by state s and action a
S	State-space
A	Action-space
R	Reward-space
$\tilde{x}, \tilde{y}$	The cartesian distances from origin of vessel to a target in body coordinates.
$\tilde{\psi}$	The relative difference between heading of vessel and a target
l	A binary variable describing whether the vessel is on land.
h	A binary variable describing whether the vessel is inside a harbour
u, v, r:	Linear and rotational velocities in body frame of the vessel, called surge (forward velocity), sway and yaw rate (SNAME convention)
$d_{obs}, \tilde{\psi}_{obs}$	Describes the relative distance and angle from the vessel to an obstacle, given in body.
f_a	The fraction of a vessel area inside a target rectangle
$\dot{f}_a$	The derivative of the fraction of vessel inside a berth-rectangle

Contents

Chapter 1

Introduction

1.1 Background and previous research

The use of autonomous systems is increasing, ranging from robotic vacuum cleaners, lawnmowers and unmanned busses, to name a few. Autonomous marine vessels are an example a of a still largely emerging but very prominent field within autonomous systems. There is increased interest in several fields of autonomy at sea, such as transportation, environmental monitoring, seafloor mapping, oceanography, and military use. There have been developed full-scale prototypes of autonomous marine surface vessels, such as Falco [12] which represent the world's first fully-autonomous ferry. Even more are being developed, for instance Yara Birkeland [13], which is the world's first autonomous zero-emission bulk freight ship, respectively. According to Rolls-Royce Marine (now part of Kongsberg Maritime), autonomous ships will be a common sight and play an important part in the oceans by 2030 [14, 15].

There are several potential advantages of autonomous marine systems, such as the ability to go into places and situations where humans cannot, elimination of human error, reduction of crew costs, increase in the safety of life, more efficient use of fuel as well as decrease of greenhouse gas emissions [16–19]. Over 90 % of the global trade is carried by sea and studies indicate that 60 % of accidents on merchant ships are due to human error [20]. Autonomous systems are therefore believed to be a key element in achieving a competitive and sustainable shipping industry in the future, changing transportation systems, operations and business models [18, 21, 22]. The MUNIN-project (Maritime Unmanned Navigation through Intelligence in Network) has for instance predicted savings of over 7 million USD over 25 years per autonomous vessel in fuel consumption and crew costs [23].

Despite recent theoretical and practical achievements within the field of autonomous marine vessels, several challenges remain. These challenges must be addressed before the potential of safe, efficient, accurate, and reliable operation in a harsh marine environment can be realised in a viable manner. One of these challenges is docking.

Docking refers traditionally to the process of reaching a berth in a controlled manner [24–26]. A berth is a fixed location with a structure facilitating fastening of a vessel along a quay, jetty or similar, in order to get off and on the vessel, safekeeping or maintenance. Docking is usually divided into several steps [27–29]:

- The first step consist of moving from open seas into confined waters, where the vessel needs to adapt to the local speed regulations, and start heading towards the desired berth.
- The second step is berthing, which starts when the vessel is in the proximity of the desired berth .Berthing is the specific process of positioning a vessel at the berth and may be likened with parking a vessel in a designated area.
- The last step is mooring, fastening the vessel to the dock. How the docking is performed depends on several factors, such as the design of the ship and quay, weather conditions, available personnel and whether help from smaller vessels such as tugboats (tugs) is involved.

During docking, the shipmaster can be forced to maintain a specific position, for instance immediately before mooring when the vessel is close to the quay. This operation has close ties to dynamic positioning. Dynamic positioning (DP) can be defined as the procedure of maintaining a vessel's position and heading (fixed location and pre-determined track), exclusively utilising thrusters [30]. The position is maintained by producing thrust in multiple directions, obtained typically by applying either tunnel thrusters and main thrusters or only azimuth thrusters. DP has proven to be highly efficient and useful in a multitude of settings such as deepwater operations.

Docking of a vessel, is considered a complex and high-risk operation, needing high precision and gentle control of the shipmaster to ensure safe operations [27, 29, 31]. The procedure depends highly on the shipmaster's experience/expertise and involves at times intensive control operations, where several tasks need to be carried out concurrently. To ensure safe operations, the shipmaster needs to know the ship's position precisely and be able to predict the movement toward the berth. During docking, the vessel dynamics are highly nonlinear, and predicting the behaviour is difficult, needing to consider numerous factors such as actuator characteristics, wind effects, wave and current disturbances. The vessel has significantly reduced manoeuvrability during docking as well, due to factors such as operating

in a confide area with reduced speed, which reduces the utility of the main propulsion/rudder. [27, 28, 31, 32]. Reduction in speed also means that disturbances such as tides, wind, and ocean currents have more effect, making manoeuvering even harder. Due to the significant reduction in manoeuvrability of larger vessels, auxiliary devices are often used to make the approach safer and faster, such as tugboats [33].

The shipmaster is usually able to dock the vessel successfully. However, 70% of the insurance claims from harbours are due to inappropriate ship handling, such as too rapid approaches, tug errors and other pilot errors [31]. A majority of these errors are caused by simple mistakes made by individuals. Autonomous vessels can potentially be able to reduce the occurrence of at least some of these mistakes. Enabling new information and communication technologies for autonomous vessels can on the other hand lead to new challenges and associated risks that must be mitigated efficiently.

Autonomous docking has been performed on full-scale vessels in real life. Wärtsilä [34] tested their autodocking technology on an 83-meter long ferry owned by the Norwegian operator Norled in 2018. The testing started 2000 meters from the berth in transit speed, and the autonomous system gradually slowed down the speed and manoeuvred the ship to the berth. Rolls-Royce (now part of Kongsberg Maritime) and a Finnish state-owned ferry operator Finferries [12] successfully demonstrated the world's first fully autonomous ferry Falco later the same year, conducting not only docking but also collision avoidance.

A limited number of studies focusing on autonomous docking of marine vessels has been published. Examples of proposed techniques are optimal control theory, expert systems, fuzzy logic controllers and artificial neural networks (ANNs) [27, 33]. One key distinguishing factor has been to what degree the solutions rely on a well-defined mathematical model of the dynamics of the vessel and its surroundings. Example of methods using a well-defined model of the vessel are often found in optimal control theory, with examples such as target tracking [35], model predictive control [3], optimal preview sliding mode controller with adaptation mechanisms [36] and quasi-real-time optimal control scheme [37]. Several of these systems solutions often involve a cascade of controllers, constructed of three independent systems, called navigation, guidance and control [38]. The control system consist of lower-level controllers which satisfies a specific control objective, such as speed control and course control. Navigation estimates the state of the vessel, and guidance system performs the higher-level tasks of planning the trajectory of the vessel and provide references to the control system.

A significant challenge of docking is its highly nonlinear nature, making it hard to establish reliable mathematical models of ship manoeuvring at all speeds and under all circumstances, used in traditional control-systems. Trying to overcome these challenges relating to the

lack of well-defined mathematical models, previous research have focused on fuzzy control [39, 40] and supervised learning using artificial neural networks (ANNs) [28, 33, 41, 42].

Fuzzy logic control is a heuristic approach that tries to embed the knowledge and critical elements of human thinking into the design of nonlinear controllers [43]. A difficulty with fuzzy rules is defining rules for ship docking in all situations, including weather influences and other disturbances that may arise.

Artificial neural networks (ANNs) are a set of algorithms designed to recognise patterns, loosely modelled after the human brain [44, 45]. ANNs have been used to learn how to dock based upon manoeuvring data [33], which is a technique within supervised learning. One limitation of these methods is that the they can only be as good as the explicit domain knowledge and experiences provided, using collected or simulated training data [42]. Supervised learning methods will in essence copy the behaviour with all inherent limitations and any bad practices. Deep Reinforcement Learning (DRL) is one way of avoiding depending on learning all the necessary behaviours directly from humans. Deep reinforcement learning (DRL) is a technique were the properties of the ANNs are exploited, but with not need of explicit need of manoeuvring data. This means DRL-methods can be capable of finding an acceptable, equally good or sometimes even better way of solving a problem than humans. There are already published a few articles successfully demonstrating the use of DRL for aspects of theoretical autonomous docking.

Reinforcement learning (RL) is also known as neuro-dynamic programming or approximate dynamic programming. It is a theory developed by the artificial intelligence (AI) community for obtaining an optimal performance of a system, accounting for uncertainties of the environment [46]. The main idea in RL is that an agent learns, by interacting with the environment, to find a behavioural policy optimising some objective given in the form of cumulative rewards. The environment may be stochastic, and the agent may even only perceive partial information about the current state. RL algorithms have demonstrated to be capable of tackling a wide range of control problems, ranging from robotics and healthcare to self-driving cars and finance. All these applications consist of finding a sequence of actions to be performed in an uncertain environment to achieving predefined goals. RL proposes a formal framework for solving such tasks.

Reinforcement learning has close ties to optimal control theory, both trying to maximise an objective function over time [45]. A behaviour policy is the equivalent of a control law in traditional control theory and solves the goal/objective in the best way possible [47]. The main distinguishing factor is that RL gives evaluative feedback, rather than instructive feedback when creating these control laws. The evaluative feedback in RL is typically given through an engineered reward function, using a scalar depending on whether an

action improves on the given objective or not in a specific state. The RL algorithm uses this evaluative feedback to find a mapping from state to action, called policy. This means that RL evaluates the actions taken rather than instructing by giving correct actions [46].

The interest in RL was boosted when RL's potential to deal with the curse of dimensionality and modelling was discovered [47]. The curse of dimensionality concerns the explosion of computational cost as the number of states increases, while the curse of modelling concerns how to model the reality accurately. Being able to handle these challenges represented a breakthrough in the practical application of RL to complex problems. Some of the achievements were realised by combining RL with deep learning techniques, called deep RL (DRL). Deep learning is a field within artificial intelligence, concerned with finding levels of abstraction within raw data [44, 48, 49].

Deep reinforcement learning is most advantageous in problems with high dimensional state-spaces and helps in tasks with lower prior knowledge because of its ability to implicitly learn different levels of abstractions from data [50]. Through the development in DRL, machines have attained a "super-human" level in playing Atari video game based on pixel information [51, 52] and mastering the game of Go [53]. Through impressive results on sequential decision-making problems in computer games, researchers started using DRL for real-world applications such as robotics [54], self-driving cars [55] and finance [56]. DRL has increased the applicability of RL for continuous control. Even so, challenges remain for efficient exploration of the environment, generalizing good behaviour in a similar but slightly different context, etc. To improve this, several algorithms have been proposed.

DRL is proved successful in controlling marine vessels in several tasks for both guidance and control [4, 5, 57]. A few articles using DRL to solve select parts of the docking problem of a marine vessel using DRL have recently been published, where few are related to unmanned surface vehicles (USVs). These published works consist of solving problems such as docking of autonomous underwater vehicles (AUVs), navigation in restricted waters of USVs, and obstacle avoidance of USVs. DRL has also been used to solve tasks similar to marine dockings, such as parking of ground vehicles [58, 59], and landing of aircraft [60, 61].

Anderlini et al. [62] used DRL to solve docking of an AUV onto a moving platform in 3 degrees of freedom, by control of continuous thruster force and rudder angle. The agent was given information about the continuous distance to the target and their derivatives and rewarded if converging towards the desired pose, and penalised if crashing. The agent is trained on only one scenario and close to the dock. The agent is however not given information about the distance to potential obstacles in the environment, making this solution less capable to generalise its knowledge to other docking scenarios.

Amendola et al. [63] solved the problem of keeping a ship at the centre of a straight channel, aligned with a guidance line. In this case the DRL-agent controls discretised rudder angles and receives continuous information about the length to desired guidance line, distance to the shore (channel sides), turning rate and velocity. The agent was rewarded when close to the guidance line and with a low rotational element. Amendola et al. [63] gives the agent information about the distance to obstacles, meaning thea agent might have some generalisation abilities, but we are not presented with more test cases to verify this hypothesis. A drawback of both these methods is the use of discretised control inputs, which can lead to suboptimal control.

A successful DRL-agent finding a collision-free path through an area of static obstacles is presented by Cheng et al. [64]. It uses continuous observations of obstacles and basic measurements of the vessel's operational states and controls discretised forces, and the agent is rewarded based on vessels distance to obstacles, velocity and distance to the desired position. Shen et al. [65] presented a DRL-agent performing automatic collision avoidance of multiple vessels in restricted waters and incorporates environmental elements such as the vessel's manoeuvrability, human experience, and navigation rules. The algorithm uses discretised course changes and is rewarded based on the distance to obstacles. This demonstrates a hybrid approach, using heuristics to incorporate existing knowledge, and switches between different controller modes, leading to a high number of design parameters.

Summarized the reasons for using DRL approaching the docking problem are:

- DRL has started to prove successful in controlling marine vessels in several tasks for both guidance and control [4, 5, 57].
- DRL's data-based approaches might help towards creating a more general approach with less time spent on engineering a solution for each scenario, compared to more traditional methods.
- DRL is well suited due to the possibility of using multiple sources of measurements, especially important in restricted waters, where several sensors often are involved to detect potential obstacles.
- Not necessitating previous knowledge of the ship dynamics or harbour area to learn how to handle guidance and control. This means better handling of scenarios with high uncertainty in the modelling and changing dynamics, such as in a harbour.
- DRL includes the possibility of creating end-to-end control laws, eliminating cascaded control systems, where the resulting accuracy depends highly on the performance of lower-level systems. This is especially important in the docking scenario, due to

complexity of modelling, as discussed by Bitar et al. [66]. They experienced that the PID-controller was unable to keep up with their Model Predictive Controller (MPC).

- In a harbour area, there is a need for swift decisions due to high-risk operations in restricted waters. While it takes time to learn a control law with DRL, the computational time is low during operations, compared to that of optimal control theory.

One of the most discussed issues using deep reinforcement learning is how to ensure safety and reliability, due to the "black-box" nature of artificial neural networks. "Black-box" is a term used for describing a system that "hides" its internal logic to the user. The lack of explainability and transparency constitutes both a practical and ethical issue to users and creators [67, 68]. Explainable artificial intelligence (XAI) has become an area of interest in the research community. It tries to improve trust, by analysing, for instance, the degree of unbiasedness (fairness), reliability, safety, explanatory justifiably, privacy and usability. To increase trust, XAI can be used to create abstract explanations to find useful properties and generating hypotheses about data-generating processes, such as causal relationships. It can therefore be interesting to apply XAI-techniques to the RL created control law, to obtain some comprehension into the logic of the control law.

Few works are conducted within reinforcement learning and explainable AI. Early research shows that the decision-making processes of RL agents can be translated into human readable descriptions [69, 70]. They are "post-hoc" interpretation approaches, meaning interpretations made on a trained model, and provide some insights into RL agents decision -making process. Lee [71] created a solution which derive a more comprehensible agent from a trained DRL agent [71]. This DRL-agent is more comprehensible due to simpler and more easily human-readable models, in addition to having comparable accuracy to that of the original RL-agent.

1.2 Goal and research questions

The goal of this thesis is to explore how to use deep reinforcement learning (DRL) to create an end-to-end harbour docking system for fully-actuated autonomous surface vessel and how to analyse the reasoning of the DRL-model using explainable AI (XAI). Docking consists of several stages. In this thesis, we study the phases from when the vessel starts to slow down its speed before entering the harbour, to when it reaches the desired berth. Mooring is not considered in this thesis since it is not a motion control problem of the vessel. The docking scenario of this thesis, consists of a simple convex harbour, without any additional obstacles, and the only external force on the vessel is a constant current, which is a common approach in research works dealing with ship control and guidance systems.

A DRL-agent performing docking will need to meet several objectives, changing over different phases of the docking process. The agent will need to handle both obstacle avoidance, speed regulation, and convergence to the desired docking position. Additionally, the agent's behaviour should be interpretable since it is performing actions within a challenging area.

Based on the goal of the thesis and these problems, the following research questions were formulated for this thesis:

- Is it possible to create an end-to-end model solving the complex multi-layer problem of docking successfully, handling model uncertainties, the influence of unknown ocean current and static obstacles?
- Is it possible to understand the reasoning behind the DRL-controller, using Explainable AI (XAI) algorithms?

1.3 Contributions

This thesis has the following main contributions:

- A progressive methodology for creating an end-to-end docking controller, solving the task of controlling a vessel from just outside a harbour to the desired berth, using deep reinforcement learning (DRL). The performances of the resulting controllers from the progressive methodology are analysed, such as accuracy when operating under external disturbance.
- Comparison of the training process and performance of the proximal policy optimisation (PPO) and deep deterministic policy gradient (DDPG) in the initial phases of the progressive methodology, showing consistent and good results using PPO.
- Application of Shapley additive explanations (SHAP), an explainable AI (XAI) techniques, to gain insight into the reasoning of DRL-based controllers.

Compared to previous work within DRL, the proposed docking model represents a more complete docking operation, in a more thorough simulation of a fully-actuated vessel in a harbour. The DRL-based controller uses state information such as the vessels velocities and distance to the target and the closest obstacle, to avoid static obstacles, perform target tracking and obey speed regulations.

The docking problem is solved progressively, given the complexity and challenges of the problem. By designing the solution progressively, the individual phases can be confirmed sep-

arately, before combining them. The suggested progressive methodology has the advantages of being model-free and optimising a specific control objective.

The progressive methodology consists of solving the following learning phases:

- Controlling the vessel from proximity of a target to the target, and keeping it there, in essence performing dynamic positioning.
- Controlling the vessel from the proximity of the berth into the berth, and keeping it there, without getting in physical contact with the quay.
- Controlling the vessel from just outside the harbour to the proximity of the berth, and keeping it.
- Performing the three previous phases into one integrated operation, controlling the vessel from the start of the harbour to the berth, and holding the vessel in the desired pose without physical contact with the berth.
- Performing the previous phase, but also obeying the velocity limits inside the harbour.

The two DRL-algorithms, proximal policy optimization (PPO) and deep deterministic policy gradient (DDPG), were applied in this thesis. Initially, DDPG was applied; inherited from the specialisation project. The performance was however not satisfactory, and PPO was tested and found to be a good alternative.

It was investigated how to use Shapley additive explanations (SHAP) on the DRL-based controller, to provide an intuition of the DRL-agents reasoning, and thereby increase the insight and understanding of the control law found during learning. Shapley additive explanations are used to explain supervised learning models, and is a technique from explainable AI. There is little previous work on how to use explainable AI in deep reinforcement learning. This is, therefore, an initial study of how to use SHAP to analyse the relative contribution of a state to the agent's decision making, with little previous work on the use of explainable AI in DRL.

The thesis contribution is the result of the following steps:

- Creating a simulator:
  - Implementing a simulator based on the dynamics of the fully-actuated cargo vessel with a pentagon-shape, a static convex harbour and a constant ocean current, based on a part of Trondheim harbour. The vessel model is reused from Martinsen et al. [3].
  - Creating states to improve the learning of the DRL-based controllers, such as distance to berth given in body-frame, distance to obstacles, area of the vessel

inside a berth-rectangle and binary variables for when a vessel is inside the docking area and harbour.

- Creating DRL-based controllers solving the docking problem:
  - Formulating the learning phase as a Markov decision processes (MDPs), which involves designing action spaces, state spaces and reward functions.
  - Adapting the DRL implementations by SpinningUp [2], to better fit with the docking problem.
  - Tuning and learning the end-to-end DRL-based models for each of the learning phases of the progressive methodology.
- Analysing the performance and reasoning of DRL-based controllers
  - Performing general analysis of the performance for all the phases, using numerical values, illustrations and videos. The robustness of some of the phases is analysed by subjecting the vessel to a constant ocean current.
  - Apply SHAP to the DRL-based controller, and analyse the SHAP-values to gain insight into the reasoning of the DRL-based controllers.
  - Comparing the performance of the PPO- and DDPG-based controllers, from several aspects of docking problem.

1.4 Outline of the thesis

The thesis consists of six chapters, which consists of:

- Chapter 1 introduces the motivation behind the thesis, providing an overview of related work, goal, research questions and contributions.
- Chapter 2 presents a theoretical background, explaining main concepts of docking, kinetics and kinematics of marine vessels, before moving on to reinforcement learning and explainable artificial intelligence (XAI).
- Chapter 3 presents the design and implementation details of this thesis. This includes presenting the progressive methodology used to solve the docking scenario using DRL algorithms. It is also explains how the DRL-based controllers can be studied and improved with the application of a XAI technique.
- Chapter 4 presents the simulation results of the implemented DRL-based controller of the progressive methodology, and analysis of the DRL-based controllers using the chosen XAI technique.
- Chapter 5 discusses suggestions for future work.
- Chapter 6 gives a brief conclusion of the thesis.
- Appendix A shows parameter values relevant to the behaviour of the container vessel and harbour.
- Appendix B shows formulas for calculating are of polygon from vertexes.
- Appendix C includes extra plots of learning phases.
- Appendix D shows a few extra SHAP plots.

Chapter 2

Theory

In this thesis a controller is created using deep reinforcement learning (DRL). The DRL algorithm learns control laws through interacting with an environment, in this case the vessel in a docking scenario. The DRL-based control laws were analysed using a technique from explainable AI. This chapter will give an introduction to :

- Section 2: Aspects of docking.
- Section 2.2: Kinetics and dynamics of a marine vessel.
- Section 2.3: Reinforcement learning.
- Section 2.4: Explainable artificial intelligence.

2.1 Aspects of docking

The Bureau Veritas S. A. is an international certification agency and has created guidelines for autonomous vessels [19]. Following the guidelines, any autonomous vessel should have at least the same degree of safety, security, and protection of the environment as provided by a conventional vessel having the same purpose of design. It is therefore crucial that the autonomous vessels are designed to be safe in themselves, to the other vessels, maritime infrastructures and the marine environment. The crew or remote operators should always be able to regain control of an autonomous vessel in case of emergency.

In general, any vessel covered by the Guidance note should be capable of:

- Complying with all relevant international and local regulations

- Managing, updating and navigating according to a predefined voyage plan while avoiding collisions with any obstacles
- Maintaining a sufficient level of manoeuvrability and stability under various sea conditions.
- Withstanding unauthorised physical or virtual trespassing

In this thesis, withstanding unauthorised physical or virtual trespassing is not part of the scope. In these next subsections, voyage plan, obstacle avoidance and manoeuvrability are discussed related to the docking scenario.

2.1.1 International and local regulations

All autonomous vessels must comply with the same set of rules and regulations as nonautonomous vessels, as noted in the previous section. Currently, there exist few (if any) regulations targeting autonomous vessels directly. Due to increased activity and development within the field of autonomous vessels, the topic has received more attention. In 2015, the fiord "Trondheimsfjorden" in Norway was established as the first test area for autonomous vessels. In addition to testing new technology, this area is also used to research regulations, operations, and infrastructure needed for autonomous vessels [18, 72]. The IMO has already started to address safe, secure, and environmentally sound operations of Maritime Autonomous Surface Ships (MASS). The IMO and Norwegian Maritime Authority ("Sjøfartsdirektoratet") are expected to publish regulations for Autonomous shipping operations by 2023 [18].

Generally, there are several regulations related to activities at ports. Some of the principal regulations in Norway and internationally are the rules of the road at sea (“Sjøveisreglene”) [73], the maritime code (“Sjøloven”) [74], the harbor act (“Havne- og farvannsloven”) [75], the International convention of life at sea (SOLAS), and convention on the International Regulations for Preventing Collisions at Sea (COLREG) [76]. These regulations cover topics such as collision avoidance, placement, and operation of lanterns and signals, responsibilities to ensure safe navigation, shipbuilding and salvaging.

Of these rules and regulations, the harbour act, COLREGs, and SOLAS are of particular interest in this thesis, and will be discussed in the following sections.

2.1.2 SOLAS

A detailed plan of the voyage from berth to berth, including areas necessitating the presence of a pilot [77], is required for all vessels by the SOLAS. In addition, SOLAS specifies a minimum standard for the construction, equipment, and operation of ships, compatible with their safety requirements [78].

A voyage follow the IMO Guidelines for Voyage planning [79] must:

- Take into account any relevant ship routeing systems
- Ensure sufficient sea room for the safe passage of the ship throughout the voyage
- Anticipate all known navigational hazards and adverse weather conditions
- Take into account marine environmental protection measures that apply

Factors that need to be considered are, for instance, the condition and state of the vessel, relevant charts, current, tidal atlases, existing ships' routing.

A somewhat more specialised plan is the port passage plan, which will connect the various port sections the vessel is navigating through from the open sea to the berth [26]. This plan will include the berth position.

The berth is allocated based on planned arrival, size of the ship, cargo etc. In the local harbour of Trondheim a berth is given a number or name, and if necessary the navigational coordinates [80]. The vessels will manoeuvre into its designated berth, between two marks on the quay, until finally, it lies beside the fenders. Fenders are bumpers designed to absorb the kinetic energy of a vessel berthing against a jetty, quay or another vessel [71]. They are used to prevent damages to vessel and berthing structures. At Trondheim Harbour the fenders diameter range from approximately 60 cm to 120 cm [80].

2.1.2.1 The harbour act

The harbour act was agreed upon by the IMO (international maritime organization) and EU (European Union), to facilitate efficient and safe port operations and sea transport [75]. The Norwegian coastal administration is upholding these regulations within the confines of Norwegian harbours.

The harbour act can supplement, clarify, or derogate from the general rules that seafarers must follow, e.g., the rules of the road at sea ("sjøveisreglene"). The harbour act includes traffic

regulations, e.g., sailing rules (including rules of speed), traffic management and injunctions towards specific types of vessels regarding the path. The general speed recommendation is:

(In my own translation): Vessels shall exercise caution and adjust the speed to the size, construction, manoeuvrability and water conditions of the vessel so that no damage or danger of injury to persons, including bathers, other vessels, shorelines, quays, aquaculture facilities or any surrounding area, may occur.

In addition to the mentioned general speed regulations, there often exist local regulations established by the local city council. The maximum speed in the harbour is often set to 5 knots (2.57 m/s), at least for smaller vessels.

2.1.2.2 COLREGs convention

The Convention on the International Regulations for Preventing Collisions at Sea (COLREGs) is published by the IMO and are rules to be followed by vessels to prevent collisions between vessels at sea [76]. In the guidance notes for autonomous vessels [19], they have analysed two tasks that should be handled by any ship covered by the Guidance note:

- The lookout: To ensure that the ship is always monitored by using appropriate information to have a full appraisal of the situation and the risk of collision.
- The operational decisions: Obligation for a ship to take avoidance decisions.

Although this thesis scope does not include other vessels, there are however three rules which affect this thesis from COLREGs Part B regarding steering and sailing section 1 (Conduct of vessels in any conditions of visibility):

- 6. Safe speed: *Every vessel shall at all times proceed at a safe speed so that she can take proper and effective action to avoid collision and be stopped within a distance appropriate to the prevailing circumstances and conditions.*
- 7.Risk of collision: *Vessels must use all available means to determine the risk of a collision, including the use of radar (if available) to get early warning of the risk of collision by radar plotting or equivalent systematic observation of detected objects. (e.g. ARPA, AIS).*
- 9: Narrow channels:
  - *A vessel proceeding along a narrow channel must keep to starboard.*
  - *Small vessels or sailing vessels must not impede (larger) vessels which can navigate only within a narrow channel.*

– *Ships must not cross a channel if to do so would impede another vessel which can navigate only within that channel.*

2.1.3 Navigation system

In the guidelines for autonomous vessels [19], the minimum level of functionality of the navigation automation system (NAS) is as follows:

The goal of the Navigating Automation System (NAS) is to be able to navigate a ship safely and efficiently along a predefined voyage plan taking into account of traffic and weather conditions.

The NAS should be able to handle all matters related to navigation, including voyage planning, docking and undocking, mooring and unmooring, navigation, anchoring and assistance in distress situations. It is emphasised that during docking and undocking monitoring sensors should be used (e.g. pressure sensors, radar) to confirm that there are no obstacles. Another functional requirement is that a device must be able to stop the sequence of docking or undocking at any time in case the system has not detected a hazardous situation.

2.1.4 Manoeuvrability

The manoeuvrability of a vessel is an important factor in planning the docking process and includes factors such as:

- The ships dynamics, e.g., ships inertia, response to strong currents, wind gusts and propulsion, and hydrodynamic effects such as thrusters working near harbor structs
- The available actuators and control system

2.1.4.1 Ship dynamics

A "masters guide to berthing" [31] defined a set of "golden rules of berthing." These include the importance of planning the voyage to the berth, teamwork, checking the equipment, slow speed, and controlled approach towards the dock. Slow speed and controlled approach are needed due to the conditions of a harbour, containing regions of small under-keel clearance, narrow channels, sailing proximity of other ships, and collaboration with tugs.

Generally, many berthing accidents occur due to overly high approach speed. The speed should always be such that the ship's stopping distance and general manoeuvring characteristics are within critical range. When close to a dock, speed should be the minimum necessary

to maintain control. It can be challenging to reduce speed and maintain control, and therefore it is essential to reduce speed in good time. At lower speeds, the current and wind have a more significant effect on manoeuvrability, and characteristics of the thrusters change. The captain needs to fully understand the ship's speed and manoeuvring characteristics to successfully dock.

2.1.4.2 Actuators

Historically, docking of large vessels has been performed by support vessels such as tug boats [3, 81]. This was mainly due to limitations in manoeuvrability and problems with accuracy. The manoeuvrability of vessels has been improved drastically, by adding for instance thrusters such as:

- Transverse tunnel thrusters: Used by ships to provide low-speed lateral manoeuvrability when docking and high thrust while at a standstill. Tunnel thrusters are mainly used during the berthing of ships, especially for huge ships, in heavy wind and tide conditions [82].
- Azimuth thrusters: Gives ships better manoeuvrability than a fixed propeller and rudder system, and are even used at high speeds.

During docking, it is important to use thrusters in the right sequence and combination, to achieve accurate and efficient docking. For a traditional or fully actuated vessel, the steps can be:

- Approach the quay at an angle, applying astern thrust in order to turn the ship and bring it parallel to the quay.
- Once stopped, the vessel can be manoeuvred into the right position using transverse thrust (if available) or by applying small kicks [of astern thrust, ed.] with an appropriate angle using a rudder.

2.2 Kinetics and kinematics of a marine vessel

A marine vessel can maximally have 6 degrees of freedom (DOF), meaning the vessel can move and rotate along all the (x,y and z) coordinate axis [38]. "The Society of Naval Architects and Marine Engineers" (SNAME) notation of 6-DOF are presented in Table 2.1, and are the notation used in this thesis. These six coordinates can be used to determine the position and orientation of the marine vessel, and can be grouped as the following:

- Coordinates describing position and translation motion: The Cartesian coordinates describing position (x, y, z) and linear velocities (u, v, ω).
- Coordinates describing orientation and rotational motion: The Euler angles (ϕ, θ, ψ) and angular velocities (p, q, r).

Table 2.1: The notation of SNAME (1950) for marine vessels , [38, p. 16].

DOF	**Linear and angular velocite**	**Position and euler angle**	**Force and moment**
Motions in the x direction (surge)	u	x	X
Motions in the y direction (sway)	v	y	Y
Motions in the z direction (heave)	ω	z	Z
Rotation about the x axis (roll)	p	ϕ	K
Rotation about the y axis (pitch)	q	θ	M
Rotation about the z axis (yaw)	r	ψ	N

These coordinates can be given in several geographic reference frames. Two common reference frames are:

- **The north-east-down (NED) coordinates system**: $\{n\} = \{x_n, y_n, z_n\}$, is usually defined in the tangent plane on the surface of the earth moving with the craft. The x-axis points towards north, the y-axis points towards east, and the z-axis points downwards normal to the earths surface.
- **The body-fixed (BODY) reference frame**: $b = \{x_b, y_b, z_b\}$ is a moving coordinate frame fixed to the craft. The body axes of a marine craft are chosen to coincide with the principal axes of inertia, giving x_b directed from aft to the fore, y_b directed to starboard, and z_b directed from top to bottom.

The dynamics of a system consists of kinematics and kinetics. Kinematics concerns the geometrical aspects of motion, while kinetics is concerned with motion caused by forces. The marine craft equations of motion give a vectorial form of a rigid body dynamics, notation from [38], and are:

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_{\Theta}(\boldsymbol{\eta})\boldsymbol{\nu}, \tag{2.1a}$$

$$\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\nu = \boldsymbol{\tau}_{RB}, \tag{2.1b}$$

$$\boldsymbol{\tau}_{RB} = \boldsymbol{\tau}_{hyd} + \boldsymbol{\tau}_{hs} + \boldsymbol{\tau}_{wind} + \boldsymbol{\tau}_{wave} + \boldsymbol{\tau}_{control}, \tag{2.1c}$$

where the rigid-body forces are represented in (2.1c). The matrices consist of inertia matrix

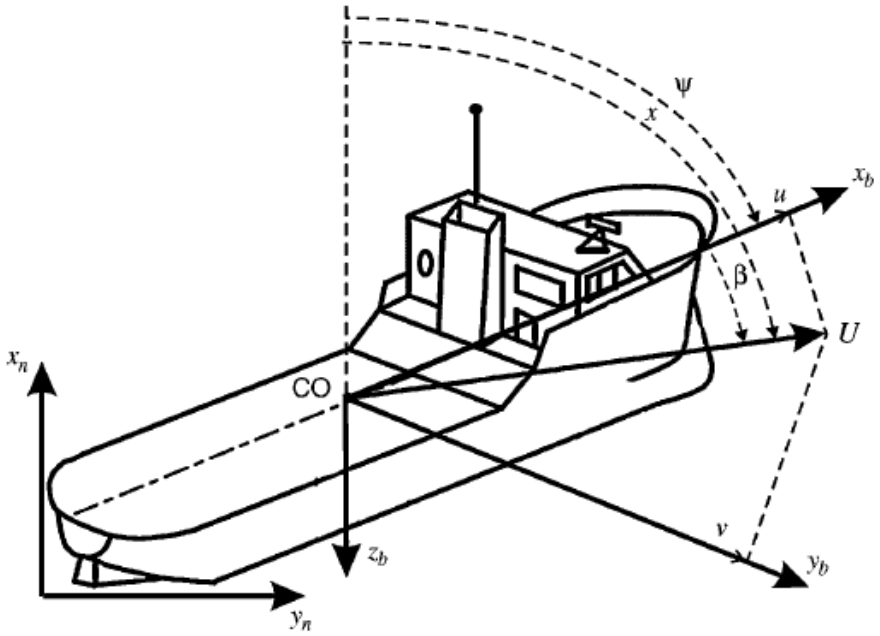


Figure 2.1: An illustration of a 3-DOF surface vehicles, with illustration of surge u, sway v, course χ, heading angle ψ and sideslip angle β, with body-fixed reference frame $\{b\} = \{x_b, y_b, z_b\}$ and Earth-fixed reference frame $\{n\} = \{x_n, y_n, z_n\}$. Illustration from [38, p. 40].

$\mathbb{M} \in \mathbb{R}^{6x6}$ and Coriolis matrix $\mathbb{C}(\nu) \in \mathbb{R}^{6x6}$. $\boldsymbol{\nu} = [u, v, w, p, q, r]^T$ and $\boldsymbol{\eta} = [x_n, y_n, z_n, \phi, \theta, \psi]$, consisting of:

- (x, y, z): The distance from NED to BODY expressed in NED coordinates.
- (ϕ, θ, ψ): The Euler angles, representing angles between n and b.
- (u, v, w, p, q, r): The linear and angular velocities in body-fixed reference frame.

2.2.1 Manoeuvring models for surface vessel

A frequently used simplification of a surface vessel is to only us 3-DOF. The three remaining degrees of freedom are surge,sway and yaw, and are illustrated in Figure 2.1.

The 3-DOF horizontal plane models for manoeuvring are based on Equation (2.1b). The assumptions for using 3-DOF model is that the hydrostatic forces $\boldsymbol{\tau}_{hs} = 0$ and with small angular velocity ω and Euler angles ϕ and θ. These simplifications provide the basis for a sufficiently good approximation for most conventional ships, and yields the following equations of motion:

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_\Theta(\boldsymbol{\eta})\boldsymbol{\nu}, \tag{2.2a}$$

$$\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\nu = \boldsymbol{\tau}_{wind} + \boldsymbol{\tau}_{wave} + \boldsymbol{\tau}_{control}, \tag{2.2b}$$

where $\boldsymbol{\nu} = [u, v, r]^T$ and $\eta^n = [N, E, \psi]$. The rotation matrix $\mathbf{R}(\psi)$ is:

$$\mathbf{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & -\cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

If an ocean current is present, the hydrodynamic forces need to be considered. The hydrodynamic forces can be modelled for a 3-DOF marine vessel as:

$$\boldsymbol{\tau}_{hyd} = \mathbf{M}_A\dot{\boldsymbol{\nu}} + \mathbf{C}_A(\boldsymbol{\nu_r})\nu_r - \mathbf{D}(\boldsymbol{\nu_r})\nu_r, \tag{2.4}$$

where $\boldsymbol{\nu}_r$ is the relative velocity $\boldsymbol{\nu} - \boldsymbol{\nu}_c$, and $\boldsymbol{\nu}_c$ is the velocity of the ocean current.

If the ocean currents are constant and irrotational in n, and use the model of hydrodynamic forces from (2.4), the equations of motion be can reorganised to:

$$\mathbf{M}\dot{\boldsymbol{\nu}}_r + \mathbf{C}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r = \boldsymbol{\tau}_{\text{wind}} + \boldsymbol{\tau}_{\text{wave}} + \boldsymbol{\tau}_{\text{control}} \tag{2.5a}$$

$$\mathbf{M} = \mathbf{M}_{\text{RB}} + \mathbf{M}_{\text{A}}, \tag{2.5b}$$

$$\mathbf{C}(\boldsymbol{\nu}_r) = \mathbf{C}_{RB}(\boldsymbol{\nu}_r) + \mathbf{C}_A(\boldsymbol{\nu}_r), \tag{2.5c}$$

with inertia matrix $\mathbb{M} \in \mathbb{R}^{3x3}$, Coriolis matrix $\mathbf{C}(\nu) \in \mathbb{R}^{3x3}$ and dampening matrix $\mathbf{M} \in \mathbb{R}^{3x3}$.

2.2.2 Ocean current forces and moments

One simple model of an ocean current for a 3-DOF marine vessel model is a 2-D irrotational ocean current model [38]. The model only simulates motion in the horizontal plane, with:

$$\boldsymbol{\nu}_c^n = \begin{bmatrix} V_c\cos(\beta_c) \\ V_c\sin(\beta_c) \\ 0 \end{bmatrix}, \tag{2.6}$$

where β_c is the angle of the current and V_c the absolute value of the ocean current velocities.

The ocean current can be transformed to reference-frame BODY $\{b\}$ by using the rotation matrix $\mathbf{R}(\psi)$ from (2.3), leading to:

$$\boldsymbol{\nu}_c^b = R(\psi)^\top \boldsymbol{\nu}_c. \tag{2.7}$$

2.2.3 Control allocation

The generalised control forces $\boldsymbol{\tau}_{\text{control}} \in \mathbb{R}^n$ to the actuators, need to be expressed in terms of control inputs $\mathbf{u} \in \mathbb{R}^r$ [38]. When $r >= n$ it is called an fully-actuated control problem, and if $r < n$ it is referred to as an underactuated control problem.

The relationship between actuator forces, moments and control forces can be specified by the thrust configuration matrix $\mathbf{T}(\boldsymbol{\alpha}) \in \mathbb{R}^3$. For a 3-DOF model it maps the thrust force f and body-frame angles α from each thruster into the surge, sway and yaw forces and moments in the body frame. The control force is calculated as follows:

$$\boldsymbol{\tau} = \mathbf{T}(\boldsymbol{\alpha})\mathbf{f}, \tag{2.8}$$

where each column in $\mathbf{T}(\boldsymbol{\alpha})$ gives the thruster configuration of thruster i, called $T_i(\alpha_i)$. Thruster configuration of thruster i is:

$$T_i(\alpha_i)f_i = \begin{bmatrix} F_x \\ F_y \\ F_y l_x - F_x l_y \end{bmatrix} = \begin{bmatrix} f_i \cos(\alpha_i) \\ f_i \sin(\alpha_i) \\ f_i(l_x \sin(\alpha_i) - l_y \cos(\alpha_i)) \end{bmatrix}, \tag{2.9}$$

where l_x and l_y are the moment arms.

The thruster allocation problem is selecting thruster angles $\boldsymbol{\alpha}$ and forces $\mathbf{f}$ to achieve the desired force $\boldsymbol{\tau}$. It can be solved in numerous ways for a fully-actuated vessel. The problem can also include for instance limitations of input amplitude or rate saturation.

The thruster allocation problem depends on the characteristic of the thruster, and two examples of common actuators are:

- **Tunnel thrusters** produces force F_y in the y-direction. It is only effective in low speeds, and are therefore typically used in low-speed manoeuvring and stationkeeping.

- **Azimuth thrusters** produces two force components F_x, F_y in the horizontal plane, controlled by rotating with an angle α about the z-axis and force F.

2.3 Reinforcement learning

Artificial intelligence (AI) has no generally accepted formal definition, but there is however consensus that it is concerned with a *thought process*, *reasoning* and *behaviour* [45]. An important field within AI is Machine Learning (ML), which addresses the question of how to build computer programs that improve their performance of some task through experience [83]. ML algorithms have successfully been applied on several problems, e.g. playing chess at master level and driving autonomously in a crowded street [45].

Commonly ML algorithms are divided into four categories, based on their purpose and data [45]. The four main categories are:

- **Supervised learning**: The task of supervised learning approximates the relationship between given input- and output-data, learned by a training set of examples of input-output pairs.
- **Unsupervised learning**: In unsupervised learning, the program learns patterns/rules based on only the input data, and is not supplied any explicit feedback, meaning input-output pairs.
- **Semi-supervised learning**: In semi-supervised learning, the program learns patterns/rules based on a given mixture of examples of input-output pairs and input without output examples.
- **Reinforcement learning**: In reinforcement learning (RL) the program learns by receiving a series of reinforcements, called rewards, in addition to inputs. The program needs to analyse and learn how to act to to receive the highest cumulative reward.

Deep reinforcement learning (DRL) is a subfield within RL, using deep learning to approximate functions, and thereby approximating an optimal solution to the RL-problem. Deep learning opens the possibility of using RL efficiently in continuous control input and state space.

The subjects of the next sections give an overview of fields related to deep reinforcement learning, and provides an introduction to specific areas and methods used in this project. Introductions provided are:

- Section 2.3.1: Artificial neural networks (ANNs), used for function approximation.

- Section 2.3.2: Reinforcement learning fundamentals.
- Section 2.3.3: Deep reinforcement learning (DRL).

2.3.1 Neural networks for function approximation

Function approximation is an instance of supervised learning, where the mathematical definition is [45]:

Given a training set of n example input-output pairs $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$ *where each* y_j *was generated by an unknown function* $y = f(x)$ *discover a function* $\hat{f}$ *that approximates the true function.*

The set $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$ is called a training set.

The function approximation $\hat{f}$ is found through searching the space of possible function approximations, for one who performs well on the training set. A hypothesis performs well if it correctly predicts the value of y for a sample not present in the training set. If so, this means the algorithm is applicable to other relevant datasets of the domain in question and generalises well.

Different techniques exist, based on the type of input-output data (continuous, discrete,etc.) and complexity of function (linear, nonlinear, etc.). In recent years deep neural networks (DNNs) have improved the learning from high-dimensional data such as time series, images and videos [50]. DNNs is the combination of artificial neural networks (DNNs) and deep learning (DL). DNNs are considered well suited when working on continuous input-output pairs, and with high function complexity, and has shown great promise in Deep Reinforcement Learning (DRL) [51–53]

To give a short introduction into this filed, this section will briefly present:

- Section 2.3.1.1: Artificial neural networks (ANNs).
- Section 2.3.1.2: Deep neural networks (DNNs).
- Section 2.3.1.3: Training of neural networks.

2.3.1.1 Artificial neural networks

Artificial neural networks (ANNs) is data-driven computing, inspired by the human brain [46, 84]. The internal function structure of ANNs is patterned after the interconnections between

neurons found in biological systems and is capable of performing advanced computing with large amounts of data. The ANN is well suited for finding relations in problems with huge amounts of data, without prior explicit knowledge of the underlying relationship between the measured inputs and the observed outputs.

The objective of an ANN is to map an input into a desired output, analogous to a mathematical function, and can be used for nonlinear function approximation. It uses training data (X, Y) to learn the function $f : X \leftarrow Y$, giving approximation $f(x; \mathbf{w}, \mathbf{b})$, parameterized with $\mathbf{w}$ and $\mathbf{b}$. The parameters of the neural network $(\mathbf{w}, \mathbf{b})$ are adjusted during training, such that the networks predictions $\hat{y}(x)$ matches the target output $y(x)$ provided by the training data.

An ANN is a network of interconnected units, inspired by the neurons in the brain [44–46]. A unit consists of inputs and one output. The output z is found by first calculating a weighted sum of the input signal $\mathbf{x}$, meaning $z = \sum x_i w_i + b$, with weight vector $\mathbf{w}$ and bias b. The output a is computed by applying the weighted sum to a activation function $f(z)$, giving output $a = f(\sum_i x_i w_i + b)$. Figure 2.2 shows the relationship between the input and output, of one neuron, where each link to the neuron has its individual weight.

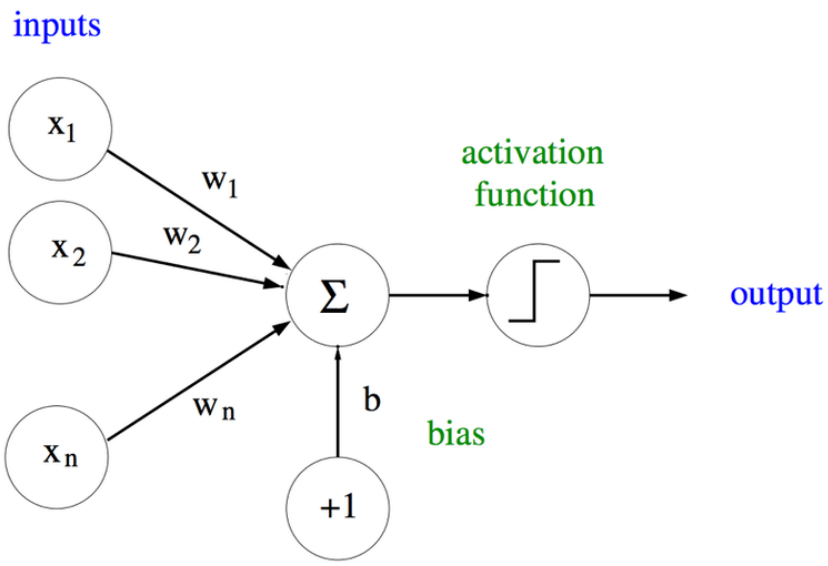


Figure 2.2: Model of an artificial neuron. Illustration from [85].

The individual weights multiplied with the respective inputs gives the strength of the signal into the unit. The units are connected through those links, where a link from unit i to j propagate the activation a_i from unit i to j, with a numeric weight $w_{i,j}$, visualised in Figure 2.3.

The activation function is typically a nonlinear function. Some of the more commonly used activation functions are sigmoid (2.10a), rectifier nonlinearity (ReLU) (2.10b), hyperbolic

tangent (tanh) (2.10c), or no activation at all (2.10d):

$$f(z) = \frac{1}{1 + e^{-z}}, \tag{2.10a}$$

$$f(z) = \max(0, z), \tag{2.10b}$$

$$f(z) = \tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{2.10c}$$

$$f(z) = z. \tag{2.10d}$$

The units of an NN can be connected in different manners, using different topologies. There are two fundamentally distinct ways of connecting units: *Feed-forward networks* and *recurrent networks*. Feed-forward networks only have connections in one direction, making a directed acyclic graph. A recurrent network, on the other hand, sends at least some of the outputs back into its inputs. By feeding back previous outputs, it is possible to gain short-term memory.

Feed-forward networks are usually arranged in layers, with each unit in one layer receiving input only from units from within the immediately preceding layer. There are different types of layers, where the three main categories are:

- Input layer: The main assignment of the input layer is to feed the input to the first hidden layer.
- Hidden layer: A hidden layer is neither directly connected to the input nor the output. The hidden layers are all the layers between the input and output layer.
- Output layer: The output layer presents the output from the neural network.

An example of a generic feedforward ANN is shown in figure 2.3. If all the units in subsequent layers have a connection, it is called a fully-connected layer, meaning all the units in the previous layer influences all the subsequent units.

An entire layer's parameters can be represented by a weight matrix $\mathbf{W} \in R^(nxm)$ and bias vector $\mathbf{b} \in R^(n)$, associated with the nodes in the layer. The output of a layer can be given when as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.11}$$

with input vector $\mathbf{x} \in R^(m)$, output vector $\mathbf{y} \in R^(n)$ and activation function f. The activation function is the same for the entire layer.

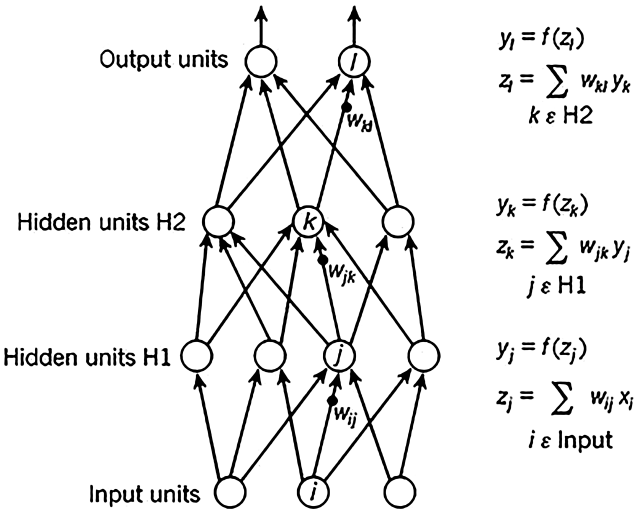


Figure 2.3: A generic feedforward ANN with four input units, two output units, and two hidden layers. Note that bias is not included in this particular figure. Illustration from [44].

The number of layers represents the depth of a network, and an ANN can consist of as little as no hidden layers. A network consisting of no hidden layers can represent only a small fraction of the possible input-output functions. An ANN with one single hidden layer can approximate any continuous function to any degree of accuracy, as long as it contains a large enough finite number of sigmoid units. This theorem is called the universal approximation theorem. It is also valid with other nonlinear activation functions under certain conditions.

The ANNs properties are decided by both how the units are connected (topology) and the properties of the units. The topology, weights and activation function must be adjusted to approximate a desired function.

2.3.1.2 Deep neural networks

Experience and theory show that the task of approximating complex functions are made easier through the use of several hidden layers, in so-called deep neural networks (DNNs) [44–46]. Deep neural networks are characterised by a succession of multiple processing layers, where the sequence of these transformations lead to learning different levels of abstraction. An example of a simple deep feed forward neural network is shown in Figure 2.3.

Deep learning methods are representation learning methods with multiple levels of representation [44]. This means each layer consists of a distinct representation of the input, and for each hidden layer, a new more abstract representation of the input is found. Very complex functions can be approximated using a sufficiently large number of hidden layers.

An image can serve as an example, where the first layer might transform an array of pixel values into a representation of presence or absence of edges for certain locations and orientations. The second layer may be representing patterns of edges, regardless of small variations in the edge position. Learning from data using a "general-purpose learning procedure", without human engineers, is one of the key features of deep learning.

2.3.1.3 Training of neural networks

Training of neural networks (NN) is the process of adjusting weights, to improve the networks overall performance as measured by an objective function $J(\boldsymbol{w})$, with weight vector $\boldsymbol{w}$ [44–46]. The objective function $J(\boldsymbol{\theta})$, also called loss-function, measures the error between the output from the NN and the correct output in the training set (called a label). The objective depends on the desired properties of the system.

A popular method to adjust the weights using the gradient of the objective is stochastic gradient descent (SGD). SGD-methods are called "gradient descent" methods since it adjusts the weights with a step proportional to the negative gradient of the objective function $J(\boldsymbol{\theta})$. The partial derivative of the objective function with respect to the weights $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ indicates how the error from the network would increase or decrease if the weights were increased by an infinitely small amount, given current values of all the network's weights.

Gradient descent methods are called stochastic when the update is performed, on a single stochastically selected example. SGDs try to minimize errors on a single example by adjusting the weight vectors after each example, by a small amount in the direction that would give the highest error reduction on that sample:

$$\mathbf{w} = \mathbf{w} + \alpha \nabla_{\mathbf{w}} J(\mathbf{w}), \tag{2.12}$$

with $\alpha > 0$ as step-size parameter, also called learning rate

The gradient $\nabla J(\boldsymbol{\theta})$ is simpler to calculate in networks with no hidden layers, as the training data provides an easy way to calculate the error, and only one layer of weights causing the error. Updating several layers of weights, with only performance measure available in one layer, the task is more challenging. The error can be back-propagated from the output layer to the hidden layers to solve this problem. Backpropagation enables DNNs to change the weights to differentiate the representation in each layer from the representation of the previous layer.

The backpropagation algorithm consists of alternating forward and backward passes through

the network, calculating the partial derivatives of the error with respect to the weights for each layer. The idea is that hidden node j is "responsible" for some fraction of the error of the output nodes connected to it. It is, therefore, necessary to divide the error between the different hidden nodes. This is done according to the strength of the connection between the hidden nodes and the output nodes, and propagated back to the specific hidden layer.

To propagate the error back, the algorithm utilizes that the gradient of the objective function with respect to the weights of a DNN can be found by applying the chain rule for derivatives. This is illustrated in Figure 2.4, showing a computational graph from input x to output y. The partial derivative of the performance measure between neuron i and j in subsequent layers with weight $w_{i,j}$ is:

$$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{i,j}}. \tag{2.13a}$$

$$y_i = f(z_j) \tag{2.13b}$$

$$z_j = \Sigma_{k=1}^{n} w_{kj} y_k + b_j, \tag{2.13c}$$

with output y_i of neuron i , the weighted input z_j to neuron j, activation function f_j to neuron j, bias b_j to neuron j and weight $w_{kj} \forall k \in (1, n)$, where n is the size of the input vector to unit j.

The forward passes compute the activation of each unit using the current activation of the network's input units. After each forward pass, a backward pass efficiently computes a partial derivative for each weight, as an estimate of the true gradient, using Equation (2.13). Through backpropagation, the partial derivative of the error with respect to the input is calculated, by simply passing gradients back through the network in reverse order, and multiplying it with the gradients of the previous layers.

The deeper the neural network is, the harder it can be to train. The backpropagation algorithm can produce good results for networks with few layers, typically with 1 to 2 hidden layers. There are several reasons for this. Deeper networks have more weights, and it is harder to generalise correctly (called overfitting). Another problem is that the partial derivatives computed by backward passes either decay rapidly toward the input side of the network, or grow rapidly towards the input side of the network. This makes it harder to learn. To solve problems learning deep neural networks, several techniques have been developed, such as dropout [86], batch normalisation [87] and deep residual learning [88].

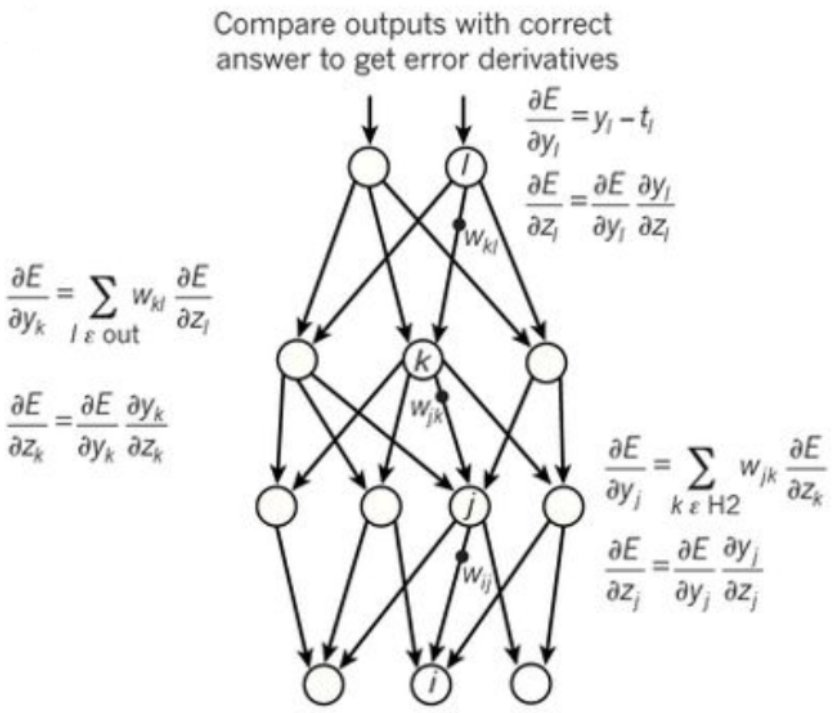


Figure 2.4: Illustration of backpropagation [44]

2.3.2 Reinforcement learning fundamentals

Reinforcement learning (RL) is an area of machine learning that deals with sequential decision-making [89]. AN RL agent learns how to behave in an environment in order to optimise a given objective. The agent may have some information about the environment (a priori), or it needs to gain this through interaction with the environment, learning through trial-and-error, and collecting this information.

The agent is expected to operate autonomously, being able to perceive an environment, adapt to changes and pursue goals [46]. The agent acts according to a policy, changed primarily according to the feedback of the agent's behaviour, given as a scalar reward by the environment. The reward signals whether the action performed in a given state was beneficial to achieve the desired objective or not, and reflects the goal of the RL-problem. It signals to the agent what to achieve, but not how. The goal of the agent is to maximise the

expected cumulative reward. The reward signal is a way of formalising the purpose or goal of the agent, and is a distinctive features of RL.

The following introduction consists of:

- A problem formulation in RL called Markov decision process (MDP) in Section 2.3.2.1.
- A discussion of the role of a policy in Section 2.3.2.2.
- The principles of how to calculate the expected cumulative reward in Section 2.3.2.3.
- The principles of how to evaluate the value of being in a certain state s in Section 2.3.2.4.
- The principles of optimal behaviour and value in Section 2.3.2.5.
- Learning methods for estimation of value functions in Section 2.3.2.6.
- Classes of RL algorithms in section 2.3.2.7.

2.3.2.1 Markov decision process

The framework called Markov Decision Processes (MDPs) is used to formalise the reinforcement learning problem. Markov decision processes is a class of sequential decision problems [45]. Sequential decision problems incorporate utilities (performance), uncertainty and perception. The utility of a system depends on a sequence of actions performed in an uncertain environment, and may be positive or negative but must be bound. RL solves MDPs by finding a policy which maximises the expected cumulative reward.

In a MDP an agent interacts with an environment in discrete time steps [45, 46]. The environment is modelled as a set of states s, where an agent performs actions a to control/influence the environment based on its perception of the environment. After performing an action, the environment gives the agent a new state s' and the agent receives feedback through a scalar reward signal r. The reward signal r is the immediate and intrinsic desirability of environmental states. The interaction between the environment and the agent is illustrated in Figure 2.5.

A MDP is defined as a 5 tuple $< S, A, R, P, \gamma >$, where:

- S is is the set of all valid states, called observation/state space. A MDP needs to have a set of fully observable states, with an initial state s_0. Fully observable means the observations are the same as the states of the environment.

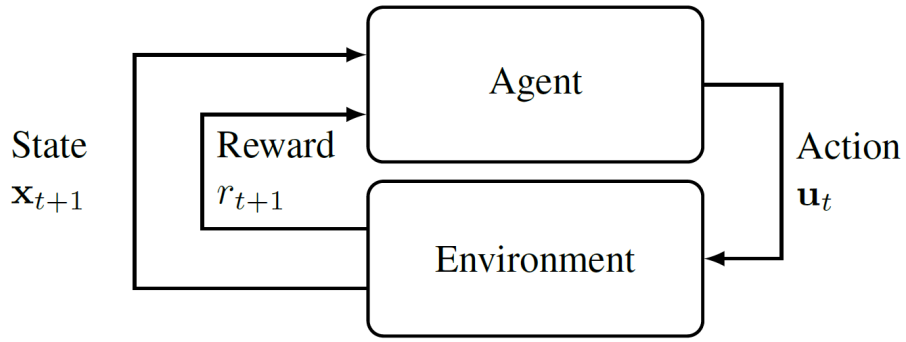


Figure 2.5: Visualisation of the interaction between the environment and the agent. Illustration from [5].

- A is the action space. $A(s)$ is the set of all valid actions for state s. Action space can be discrete or continuous.
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1}) \in R$. r_t is the expected immediate reward at t when performing action a_t in state s_t. It can also only depend on the current state of the world, the action just taken, the next state of the world, or any combination of the three.
- $P : S \times A \times S \rightarrow [0, 1]$ is the Markovian transition probability function $P(s'|s, a)$. The stochastic transition model $P(s'|s, a)$ denotes the probability of reaching state s' given action a is performed in state s.
- $\gamma \in [0, 1)$ is the discount factor.

For a Markovian transition model, the probability of reaching s' from s depends only on s and not on the history of earlier states. This is called the Markov property and means that the future of the process only depends on the current observation, and the agent has no interest in looking at the full history.

The design of the reward signal is critically important, as it is the primary basis for altering the policy. It is crucial that the reward function only should aid the agent to achieve the desired objective, and not how this should be achieved [46]. The agent should be free to solve the assignment within the constraints, hopefully providing new and smart (or at least creative) ways of solving the task.

2.3.2.2 Policy

A policy is a mapping from state s to action a and is a solution to the MDP [46]. The policy is learned through the agent interacting with the environment (i.e., the MDP), gaining knowl-

edge about how to optimise its behaviour through the rewards received after performing an action in a specific state. If the agent has a complete policy, the agent will always know what to do next from any state at any given time.

A policy can either be deterministic or stochastic:

- A stochastic policy is denoted $\pi(s, a)$, with $a \sim \pi(\cdot|s_t)$, and gives the probability of selecting each possible action for a given state s.
- A deterministic policy is denoted $\mu(s)$, with $a = \pi(s_t)$, and gives the action a for a given state s

The quality of a policy is measured by the expected utility of the possible environment. An optimal policy yields the highest expected utility and is further explained in Section 2.3.2.5. RL methods specify how its experience should change the agent's policy in order to achieve a higher expected cumulative reward.

2.3.2.3 Return

The goal of the agent is to maximise the expected cumulative reward. The term "return" is used to denote some notion of the cumulative reward, and thereby the goal of the agent. The return is denoted G_t and is defined as some specific function of the reward sequence. Two ways of defining the return is finite-horizon undiscounted return, (2.14a), and infinite-horizon discounted return, (2.14b):

$$G_t = \Sigma_{k=0}^{T} r_{t+k+1}, \tag{2.14a}$$

$$G_t := \Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.14b}$$

where the sequence of rewards received after time step t is denoted $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$.

Finite-horizon undiscounted return is the sum of rewards obtained in a fixed number of steps. Examples of where finite-horizon undiscounted return might be more appealing are in episodic tasks. In an episodic task, there is a natural notion of the final time step, meaning the agent-environment interaction breaks naturally into subsequences, called episodes. The end state of an episode is called the terminal state, happening at time step T. The terminal state may have different outcomes (rewards). An example of how the same terminal state of a system can give different outcomes is when playing games. The player either win or lose, when in a terminal state, but by how much may vary.

Infinite-horizon discounted return is more desirable to use in tasks when $T \rightarrow \infty$. In these tasks, the finite-horizon undiscounted return can be rendered infinitely, and this is not desirable as the agent maximises the return. These scenarios are called continuing tasks and are tasks that cannot naturally be broken into natural subsequences. Examples of such processes are on-going process-control tasks or a robot performing an assignment with a long life span.

The infinite-horizon discounted return is the sum of all rewards received k time steps in the future discounted by γ^{k-1}. γ is the discount rate, where $0 \leq \gamma \leq 1$, and determines the present value of future rewards. The closer γ gets to one the agent becomes more farsighted, meaning it takes the future rewards more strongly into account. Similarly, if $\gamma = 0$, the agent is only concerned with maximising immediate rewards.

Both finite-horizon undiscounted and infinite-horizon discounted return meets the consistency condition. This means that the returns at successive time steps are related, giving:

$$G_t := r_{t+1} + \gamma G_{t+1}. \tag{2.15}$$

This condition makes it easier to calculate returns from reward sequences, and is used in developing algorithms in RL. It works for all time steps t < T, even if termination occurs at t+1, as long as the termination reward $G_T = 0$.

Moving forward, unless otherwise state, the discounted infinite-horizon return will be used.

2.3.2.4 Value functions

A value function seeks to estimate the expected return [46]. It "indicates the long-term desirability of states after taking into account the states that are likely to follow based on the policy π, and the rewards available in those states". The value function can be used to select the actions giving the highest expected amount of reward in the long run, and thereby optimising the agent goal/objective.

There are two main types of value-functions:

- **State-value function** gives the expected return when the agent follows policy π after timestep t, starting in state s. It is denoted $v_\pi(s)$.
- **Action-value function** gives the expected return for taking action a in state s under policy π. It is denoted $q_\pi(s, a)$.

The state-value function $v_\pi(s)$ with the agent following policy π after timestep t, starting in state s is:

$$v_\pi(s) := \mathbb{E}_\pi[G_t|S_t = S] = \mathbb{E}_\pi[\Sigma_{k=0}^{\infty}\gamma^k R_{t+1+1}|S_t = s], \text{for all } s \in S, \tag{2.16}$$

where $\mathbb{E}_\pi(\cdot)$ denotes the expected value of the return given state s.

The action-value function $q_\pi(s, a)$ starting from state s, taking the action a under policy π is:

$$q_\pi(s, a) := \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}[\Sigma_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a]. \tag{2.17}$$

In addition a useful function is the advantage function:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s, a) \tag{2.18}$$

,where $A_\pi(s, a)$ describes the advantage/disadvantage of performing action a in state s compared to randomly selecting an action according to $\pi(\cdot|s)$, assuming you act according to π forever after.

The advantage function gives the advantage of selecting a certain action from a certain state.

The state-value and action-value function can be estimated based on experiences received over the agent's lifetime. How this estimation is performed is an important part of several reinforcement algorithms.

The consistency condition of the return is transferred to the value-function. This means for any policy π and state s a consistency condition holding between the values of successive states s and s':

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ &= \Sigma_a \pi(a|s)\Sigma_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \end{aligned} \tag{2.19a}$$

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \\ &= \Sigma_{s',r} p(s', r|s, a)[r + \gamma \Sigma_{a'} \pi(s', a') q_\pi(s', a'), \end{aligned} \tag{2.19b}$$

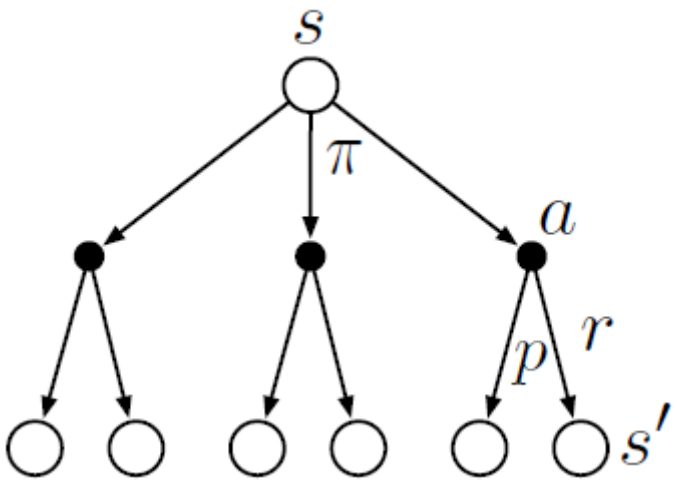


Figure 2.6: Backup diagram illustrating different paths from initial state s (root node), based on performed action a based on policy π and possible next states s' based on transition model p. The agent receives reward r for performing action a in state s [46]

for all $s, s' \in S$, $r \in R$ and $a \in A(s)$. These equations are the Bellman equations for v_π and q_π, acting on-policy. On-policy means that the agent is always acting according to the current policy π.

The Bellman equation for v_π gives the sum overall values of the three variables a, s' and r. It is illustrated in Figure 2.6. The open circles represent a state, and the solid circles represent a state-action pair. The root node at the top represents the starting state s. The agent can take any actions available for a certain state, based on its policy π. Performing a certain action will lead to a new state s', along with a received reward r, depending on the dynamics given by probability p. The Bellman equation averages over all possibilities, weighting each by its probability of occurring.

2.3.2.5 Optimal policies and optimal value functions

The goal in RL is to find a policy that achieves the highest expected return, where the optimal policy performs better or equal to all other policies [46]. A policy π is defined better than or equal to policy π' if and only if $v_\pi(s) \leq v_{pi'}(s)$ for all $s \in S$, denoted $\pi \leq \pi'$. The optimal policy is denoted π_*. It may be more than one optimal policy, but they all share the same optimal value-function. The optimal action-value is denoted q_* and optimal state-values is

denoted v_*. These are defined by the Bellman optimality equations:

$$\begin{aligned} v_* &= \max_\pi v_\pi(s) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}|S_t = s, A_t = a] \\ &, \text{ for all} s \in S, \end{aligned} \tag{2.20a}$$

$$\begin{aligned} q_*(s,a) &= \max_\pi q_\pi(s,a) \\ &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \\ &, \text{ for all} s \in S \text{ and } a \in A(s). \end{aligned} \tag{2.20b}$$

The Bellman optimality equation gives that the value of a state following an optimal policy must equal the expected return for the best action from that state. The Bellman optimality Equation (2.20a) is a system of n equations, one for each state, with n unknowns. For finite MDPs (finite state-, reward- and action-space) a unique solution to the optimal state-value function v_* exists. In cases where the dynamics of the environment p are known, v_* and q_* may be found by applying traditional methods for solving nonlinear equations. Dynamic programming (DP) is algorithms that use a perfect model of the environment to compute optimal policies.

After calculating v_*, the optimal actions can easily be found by selecting the action leading to the neighbouring state with the highest value. If calculating the optimal action-value q_* instead, the optimal policy is found immediately by looking up the actions for a specific state s, which gives the maximum value. The optimal action-value and state-value function give optimal actions without knowledge of the dynamics of the environment.

2.3.2.6 Learning methods for estimating value functions

Different techniques can be used to find the optimal policy. When the dynamics of the environment is known, dynamic programming (DP) may be used, as described in the previous section. When the environment is unknown, estimates of value function are typically needed [46]. This may be done using either:

- Monte Carlo methods.
- Temporal-difference learning.

In the following sections, the techniques will be explained through the estimation of state-value function $V(S_t)$, but these methods can also be used to approximate other functions.

Monte Carlo methods

In RL, Monte Carlo (MC) methods estimate the return, by simply averaging the returns observed after visits to that state. Monte Carlo methods must, therefore, wait until the return of a certain state s is known, and thereafter use the return as a target for state-value function updates. The average converges towards the expected value, as more returns are observed.

MC methods thus do not need explicit knowledge of the environment dynamics but depends rather on experience. Using only experience, the method is theoretically, able to attain optimal behaviour.

There are different methods to update the average return. One technique is called the every-visit MC method, which estimates $v_\pi(s)$ as the average following all visits to state s. A simple every-visit Monte Carlo method is:

$$V(S_{t+1}) = V(S_t) + \alpha[G_t - V(S_t)], \tag{2.21}$$

where G_t is the actual return following time t, and α is the step-size parameter.

MC-methods gives an unbiased but noisy estimator of the expected return.

Temporal-difference learning

Temporal-difference (TD) learning combines ideas from Monte Carlo and dynamic programming (DP). TD uses experience, in the same manner as Monte Carlo methods, and has no direct knowledge of the dynamics of the environment. The difference is that TD methods update estimates based partly on other learned estimates, without waiting for a final outcome (bootstrapping). This means TD will only have to wait until the next time step, whereas Monte Carlo methods have to wait until the end of an episode to update the estimate (receiving G_t). The simplest TD method at time $t + 1$ updates value-function estimation based on reward R_{t+1} and the estimated $V(S_{t+1})$, giving:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{2.22}$$

with step size α, deciding how much the estimate $V(S_t)$ should move towards the target $R_{t+1} + \gamma V(S_{t+1})$. This is called TD(0), or one step TD, looking ahead one step at a time.

TD error is defined in (2.23), and is the difference between the estimated value $V(S_t)$ and the

better estimate $R_{t+1} + \gamma V(S_{t+1})$. γ_t is the error in $V(S_t)$, available at time $t + 1$:

$$\delta_t := R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{2.23}$$

2.3.2.7 Classes of RL algorithms

There are different components used to learn an agent how to maximise the expected reward [89]. An RL agent will include one or more of the following components:

- A representation of a value function.
- A representation of the policy π.
- Model of the environment, with the estimated transition- and/or reward function.

When the method uses a model of the environment, the method is called model-based RL, otherwise the method is called model-free. The model-free methods are typically split into value-based, policy-based and actor-critic methods [89–91], based on which components the agent consists of:

- Value-based: Methods based on value functions. The value-based methods aim at learning a value function that estimates the expected return of being in a given state, which subsequently is used to define a policy [89]. The optimal policy is found by selecting the action that maximises the learned value function. Examples of value-based methods are Q-learning [92] and deep Q-network [52].
- Policy-based: Policy-based methods, or policy search methods, learn directly a parameterised policy [90]. The methods search directly for an optimal policy $\pi*$, but without learning the value function. A parameterised policy π_θ is typically used, with parameters updated to maximise the expected return $\mathbb{E}[R|\theta]$, using either gradient-based or gradient-free optimisation. Most methods are using gradient-based training, due to being more sample-efficient for policies with a large number of parameters. An example of a policy gradient method is REINFORCE [93].
- Actor-critic methods:Actor-critic methods learn both a value function and an explicit representation of the policy, merging the advantages of value-based and policy search methods [90]. Actor-critic methods tend to converge much faster than policy search methods but have a higher bias. The methods consist of an actor and a critic, where the actor, the policy, learns by using feedback from the critic, the value function. The goal of these methods is to learn an actor that maximises the critic.Examples of actor-

critic methods are actor-critic [94, 95], natural actor-critic [96], trust-region policy optimization [97], proximal policy optimization algorithms [98] and deep deterministic policy gradient [1].

2.3.3 Deep reinforcement learning

Explicitly solving the Bellman optimality Equation (2.20) is one way of finding an optimal policy, and thereby solving the reinforcement learning problem [46]. However, this is a method seldom used for real-world problems. There are three main challenges: The need to know the dynamics of the environment accurately, the high computational requirements needed to find the solution and the Markov property. The computational resources and memory constraints arise in problems with high state- and action-space, leading to the need for solving several optimal Bellman equations and requiring sufficient space to store these solutions.

To avoid the need for a value-function with an explicit value for each state or state-action pair, the idea is to approximate a good general policy or value-function, which can be used in the entire state-space. The approximated functions use some sort of compact parameterized representation. The online nature of RL makes it possible to put more effort into learning to make good action selections for frequently encountered states and putting less effort into infrequently encountered states. The approximated functions are still able to make decisions in states which it has not experienced and is distinguishing feature of RL, compared to other approaches approximately solving MDPs.

In theory function approximation in RL can be performed using all methods from supervised learning and/or deep learning, but not all methods are equally suitable. Methods that are more suitable for RL are those with the following properties:

- The learning can be performed online, meaning the function approximation are performed while the agent interacts with its environment or with a model of it. This requires the method to learn efficiently from incrementally acquired data.
- The method is able to handle target function or values changing over time, called non-stationary target functions and values. The first might come from approximating the value function q_π while π are still changing, while the second might come from approximated return from bootstrapping methods like TD learning.

Neural networks are for instance well suited for dealing with high-dimensional sensory inputs. Often function approximation are used on value-functions $Q(s_t, a_t)$ or $V(s_t)$, or policy $\pi(a_t|s_t)$, with approximations $Q(a_t|s_t, \mathbf{w})$ and $V(s_t, \mathbf{w})$, with parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{d'}$

and$\gtrapprox \in R^d$. The parameterized policy $\pi(a|s, \boldsymbol{\theta})$ gives the probability of taking action a at time t given the environment state s with parameter $\boldsymbol{\theta}$, giving $\pi(a|s, \boldsymbol{\theta}) = P(A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$. The parameterized state-value function gives the expected return for state s_t at time t with parameter $\mathbf{w}$, while action-value function gives the expected return given state s_t and action a_t at time t with parameter $\mathbf{w}$.

For continuous control, the problem of finding an action is not trivial, since choosing an action requires solving a maximisation problem. Extensions are made for value-based methods to handle continuous controls, but they are often restricted [99, 100]. Therefore in the next sections, the policy-gradient methods and actor-critic are further discussed with respect to DRL.

2.3.3.1 Policy gradient methods for DRL

Policy gradient methods is a class of methods which uses stochastic gradient ascent with respect to the policy parameters to optimize the expected cumulative reward to find a good policy[46, 89]. They belong to the broader class of policy-based methods, since it directly learns a parameterized policy.

The policy is parameterized by parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. The parameterized policy $\pi(a|s, \boldsymbol{\theta})$ gives the probability of taking action a at time t given the environment state s with parameter $\boldsymbol{\theta}$, giving $\pi(a|s, \boldsymbol{\theta}) = P(A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$. Neural networks are often used to approximate the policy, but it is not important in policy gradient methods which approximation methods is used, as long as the method gives a policy which is differentiable with respect to its parameters.

The parameter vector is updated in order to estimate the optimal policy $\pi_{\boldsymbol{\theta}} = \pi_*$. This is done by updating the policy to maximise the performance measure $J(\boldsymbol{\theta})$, which is the expected cumulative reward. For an episodic case the performance measure is the value of the start state s_0 for the episode, meaning:

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0) = \mathbb{E}_\pi[G_0|S_0 = s_o] \tag{2.24}$$

$v_{\pi_{\boldsymbol{\theta}}}$ is the true value function for the policy $\pi_{\boldsymbol{\theta}}$ determined by $\boldsymbol{\theta}$.

The policy parameter $\boldsymbol{\theta}$ is updated using stochastic gradient ascent. Stochastic gradient ascent is to stochastic gradient descent, but it maximises the objective, and takes a step in the positive direction of the gradient. Updating $\boldsymbol{\theta}$ in order to maximise the performance measure $J(\boldsymbol{\theta})$ gives:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \hat{J}(\theta_t), \tag{2.25}$$

where $\nabla \hat{J}(\theta_t) \in \mathbb{R}^{d'}$ is an approximation of the gradient of J with respect to the policy parameter $\boldsymbol{\theta}_t$. It can be approximated using the policy gradient theorem.

The policy gradient theorem provides an analytical expression for the gradient of the performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameter $\boldsymbol{\theta}$ [46]. The gradient of the performance measure can be estimated in different ways, but using the policy gradient theorem gives a representation which is independent of the action selection and state distribution. State distribution is the distribution of how often the states occur following the policy. Independence of state distribution is a good feature, especially when the environment is unknown. Lacking this feature would make it difficult to estimate the effect of a policy update on the state distribution.

The policy gradient theorem represents the gradient of J as a column vector of partial derivatives with respect to the components of $\boldsymbol{\theta}$. The policy gradient theorem for an episodic case is:

$$\nabla_\theta J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\theta), \tag{2.26}$$

where μ is the on-policy distribution under π, meaning the distribution of how often the states occur following policy π.

The policy gradient theorem may be rewritten when both state and action distribution follow the policy $\pi_{\boldsymbol{\theta}}$ (on-policy), meaning replacing a and s by sample A_t π and S_t μ. To avoid summation over all the actions, and only update the action performed at time t, we want to achieve an expectation under π. This can be performed only if each term is weighted by $\pi(a|S_t, \boldsymbol{\theta})$. Therefore the gradient is multiplied and divided by $\pi(a1S_t, \boldsymbol{\theta})$. Using these tricks, the theorem for on-policy algorithms can be rewritten as:

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}] \\ &\propto \mathbb{E}_\pi[G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})] \end{aligned} \tag{2.27}$$

with $q(S_t, A_t) = \mathbb{E}_\pi[G_t, S_t, A_t]$ and the eligibility vector $\frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} = \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$.

The gradient vector gives the direction in parameter space that most likely increases the

probability of repeating the action A_t on future visits to state S_t. Using this gradient in the stochastic gradient ascent algorithm, helps moving in the parameter vector $\boldsymbol{\theta}$ in the direction that favour actions that gives the highest return.

The edition of the policy gradient theorem presented in Equation (2.27) gives an expression which quantity can be sampled on each time step. The update of the parameter vector $\boldsymbol{\theta}$ using (2.27), yields the REINFORCE update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}), \tag{2.28}$$

with learning rate α, policy $\pi(a|s, \boldsymbol{\theta}$ and action-value function $q(a, s)$.

The policy gradient theorem lays the theoretical foundation for several policy gradient algorithms, with an unbiased estimate of $\nabla J(\boldsymbol{\theta})$. The algorithm called REINFORCE uses the update of the same name, sporting good theoretical convergence properties. This happens at the cost of high variance using a Monte Carlo method to approximate the policy gradient (2.27). A high variance might lead to slower convergence, while high bias can lead to converging to a poor solution or being unable to reach a solution. Therefore it is desirable to both have low bias and variance.

One way of reducing the variance is by including a comparison of the action value to an arbitrary baseline $b(s)$, giving the approximated policy gradient:

$$\nabla J(\boldsymbol{\theta}) \propto \Sigma_s \mu(s) \Sigma_a (q_\pi(s, a) - b(s)) \nabla(a|s, \boldsymbol{\theta}) \tag{2.29}$$

The baseline can be any arbitrary function, as long as it does not depend on action a, such that Equation (2.29) remains valid. When b(s) is independent of a, the subtracted quantity is zero:

$$\Sigma_a b(s) \nabla \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla \Sigma_a \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0 \tag{2.30}$$

The baseline in general improves the variance, while not affecting the expected value of the update.

The policy gradient theorem with baseline is used to give a version of REINFORCE where the parameter vector is updated according to:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta} + \alpha (G_t - b(S_t)) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}, \tag{2.31}$$

This is a strict generalisation of REINFORCE, since the baseline could be uniformly zero. One intuitive choice for baseline is an estimate of the state value, $b(s) = \hat{v}(S_t, \mathbf{w})$, where $\mathbf{w} \in R^d$ is a weight vector in the approximated state-value function. Using the same assumptions as when deriving the REINFORCE update, and $J(\boldsymbol{\theta}) = \frac{1}{2}(G_t - \hat{v}(S_t, \mathbf{w}))^2$, the approximated policy gradient is:

$$\nabla J(\boldsymbol{\theta}) = (G_t - b(S_t))\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \tag{2.32}$$

The REINFORECE algorithm with baseline, is represented in Algorithm 1. The baseline here is an approximated state-value function, $b(s) = \hat{v}(S_t, \mathbf{w})$, where state-value weights $\mathbf{w}$ may be learned using Monte Carlo methods. This algorithm has two step sizes, denoted $\alpha^{\boldsymbol{\theta}}$ and $\alpha^{\mathbf{w}}$.

Algorithm 1 REINFORCE: with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} = \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$ and state-value function parameterisation $\hat{v}(s, \mathbf{w})$
Algorithm parameter: step sizes $\alpha^{\boldsymbol{\theta}} > 0$ and $\alpha^{\mathbf{w}} > 0$.
Initalize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and $\mathbf{w} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)
for each episode **do**
  Generate an episode $S_0, A_0, R_1, \ldots S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode t = 0,1,..., T-1
  $G \leftarrow \Sigma_{k=t+1}^{T} \gamma^{k-t-1} R_k$
  $\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t \delta \nabla ln \pi(A_t|S_t, \boldsymbol{\theta})$
end for

2.3.3.2 Actor-critic

Methods that learn approximations of both policy and value functions are called actor-critic methods and are a hybrid of value-based and policy-based methods [101]. These methods consist of an actor and a critic. The actor (policy) selects the action to be performed in a certain state. The critic (value function) continuously evaluates the performance of the actor by providing the value of being in a certain state when acting according to the policy. Feedback from the critic is used to improve both the policy of the actor and value-function of the critic. The interaction between actor and critic is illustrated in Figure 2.7.

In the previous section, the algorithm REINFORCE with baseline was presented, see algorithm 1, using a learned value-function and policy, but this algorithm is not an actor-critic method [46]. This is because the estimation of the value function is not updated based on estimated

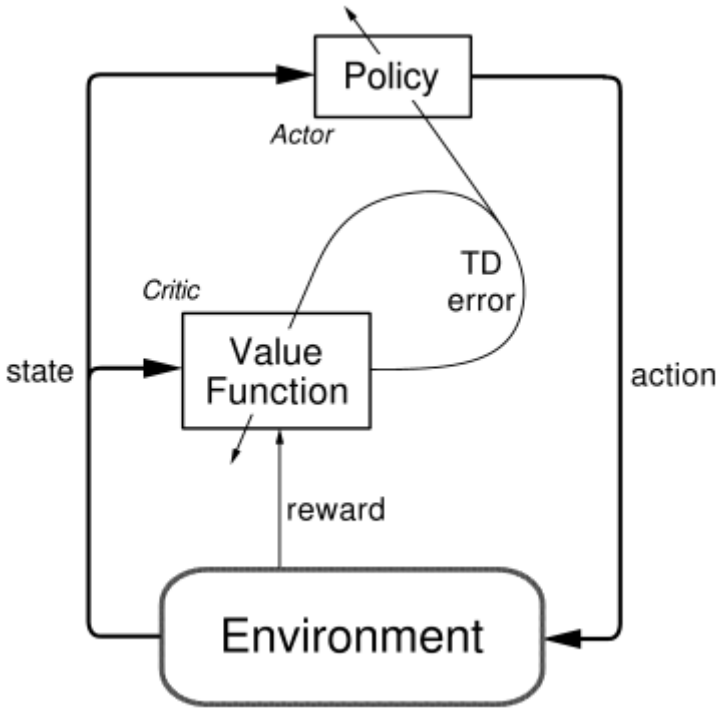


Figure 2.7: Figure showing the interaction between actor and critic in the actor-critic model. The critic gives the value of being in a certain state when acting according to the policy of the actor, while the actor gives the action to be performed in a certain state [46]

values of subsequent states (bootstrapping), and the value-function is only used as a baseline for the actor. Therefore the value-function in REINFORCE does not criticise directly the progress of the development of policy function, and is not acting as a critic.

Actor-critic methods approximates the policy $\pi(a|s, \boldsymbol{\theta})$ with parameter vector $\boldsymbol{\theta}$ and approximated state-value function $\hat{v}(S_t, \mathbf{w})$ by weight $\mathbf{w}$. The actor-critic methods can learn the value-function through TD(0)-learning. The REINFORCE algorithm uses Monte Carlo methods to estimate the value function and tends to learn slowly due to high variance and is inconvenient to implement online or for continuing problems, due to the need to wait for the return after an episode. Using TD methods, these inconveniences can be eliminated, giving lower variance and an online-algorithm. By using TD(0), the algorithm is fully online and incremental. This means the algorithm is constantly learning with every step, collecting labels and training itself, while also executing as a working agent.

The actor applies an action to the environment, the critic use the TD error, (2.23), to evaluate whether if the return is higher or lower than expected. If the TD error is positive, the critic suggests that the tendency to select this action should be strengthened for the future, and weaker if the TD error is negative. This interaction is shown in Figure 2.7. The desired objective is to get TD error towards zero. This means that the value-function estimator is a

good approximation of the true value function V_π, meaning gaining an optimal policy. The TD-error is minimised using stochastic gradient ascent for updating the weight $\mathbf{w}$ of the critic, giving the update equation:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_w \delta_t \nabla_{\mathbf{w}} V_{\mathbf{w}}(s), \tag{2.33}$$

with performance measure $J(\boldsymbol{w}) = \frac{1}{2}(\delta_t)^2 = \frac{1}{2}(R_{t+1} + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))^2$, meaning the squared TD-error.

The actor updates its policy based on the approximated policy gradient, using TD error given by the critic. The actor is updated in a similar manner as in REINFORCE, but the full-return is replaced with the one-step return ($R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ giving:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \delta_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}). \tag{2.34}$$

An entire one-step actor-critic algorithm for estimating $\pi_{\boldsymbol{\theta}} = \pi^*$, is presented in algorithm 2.

Algorithm 2 One-step Actor-Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} = \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$ and state-value function parameterisation $\hat{v}(s, \mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$ and $\alpha^{\mathbf{w}} > 0$.
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)
for each episode **do**
  Initialize S (first state of episode)
  $\mathbf{I} \leftarrow 1$
  for each step of the episode t= 0,1...,T-1 **do**
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S',R
    $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \mathbf{I} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
    $\mathbf{I} \leftarrow \gamma \mathbf{I}$
    $S \leftarrow S'$
  end for
end for

2.3.3.3 Deep deterministic policy gradient

Deep deterministic policy gradient (DDPG) [1] is an off-policy model-free algorithm, able to operate in continuous state and action space. It is actor-critic and has a structure similar to the one-step episodic actor-critic Algorithm 2.

The policy and value-function are approximated using DNNs. DDPG consists of a deterministic policy, $\mu : S \rightarrow A$, and an action-value function $Q^{\pi}(s_t, a_t)$. This is a difference from the actor-critic Algorithm 2, where a stochastic policy and state-value function was used.

The action-value function describes the expected return after taking an action a_t in state s_t, and thereafter following policy μ:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t], \tag{2.35}$$

with infinite-discounted future return R_t,and discounting factor γ.

Since DDPG uses the action-value, the algorithm does not depend on explicitly knowing the dynamics of the environment, and thus the algorithm can learn Q^{μ} off-policy. This means the algorithm can use transitions generated from a different stochastic behaviour policy β during learning.

The desired objective for the approximation of the action-value function is to minimise the squared TD-error for action-value, the same as for value-function in algorithm 2, giving the loss function parameterized by parameter θ^{Q}:

$$L(\boldsymbol{\theta}^{Q}) = \mathbb{E}[(Q(s_t, a_t | \boldsymbol{\theta}^{Q}) - y_t)^2], \tag{2.36a}$$

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) | \boldsymbol{\theta}^{Q}). \tag{2.36b}$$

The Q-function is updated by taking one step of gradient descent using $\nabla_{\boldsymbol{\theta}^{Q}} L(\boldsymbol{\theta}^{Q})$:

$$\boldsymbol{\theta}^{Q} \leftarrow \boldsymbol{\theta}^{Q} + \alpha^{\boldsymbol{\theta}^{Q}} \nabla_{\boldsymbol{\theta}^{Q}} L(\boldsymbol{\theta}^{Q}). \tag{2.37}$$

The deterministic policy $\mu(s, \boldsymbol{\theta}^{\mu})$ is parameterized with parameter vector $\boldsymbol{\theta}^{\mu}$, giving a deterministic mapping from states to a specific action. To find a good policy it is necessary to explore the solution space. Since DDPG uses a deterministic policy, itself won't lead to sufficient exploration of the environment. It is therefore needed to follow an exploratory stochastic behaviour policy during training, while learning a deterministic policy. This is only possible since the algorithm is off-policy, meaning the exploration can be treated independently from the learning algorithm. The stochastic behaviour policy β is computed by adding noise samples from noise process N to the actor policy during training:

$$\mu'(s_t) = \mu(s_t | \boldsymbol{\theta}_t^{\mu}) + N. \tag{2.38}$$

Using a deterministic policy, the policy gradient can be estimated much more efficiently than with usual stochastic policy gradient [102]. Results of using deterministic policy gradients demonstrates significant performance advantages compared to stochastic policy gradients, especially for high dimensional tasks [102].

The actors parameter $\boldsymbol{\theta}^{\mu}$ is updated by calculating the gradient of the loss function $J(\boldsymbol{\theta}^{\mu})$, with respect to the actor parameters. The goal for the policy is to learn a policy which maximises the expected return from the start distribution giving optimisation of the performance measure $J(\boldsymbol{\theta}^{\mu}) = \mathbb{E}[G_0|S_0 = s]$. With a deterministic policy, the gradient is:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}^{\mu}} J &\propto \mathbb{E}_{s_t \sim \rho^{\beta}}[\nabla_{\boldsymbol{\theta}^{Q}} Q(s, a|\boldsymbol{\theta}^{\mu})|_{s=s_t, a=\mu(s_t|\boldsymbol{\theta}^{\mu})}]) \\ &= \mathbb{E}_{s_t \sim \rho^{\beta}}[\nabla_a Q(s, a|\boldsymbol{\theta}^{Q})|_{s=s_t, a=\mu(s_t|\boldsymbol{\theta}^{\mu})} \nabla_{\boldsymbol{\theta}^{\mu}} \mu(s|\boldsymbol{\theta}^{\mu})|_{s=s_t}], \end{aligned} \tag{2.39}$$

where ρ^{β} is state distribution over behaviour policy $\rho^{\beta}(s)$, where $\mathbb{E}_{s\ \rho^{\beta}}(\cdot)$ is the expected value with respect to the behaviour policy state distribution. The chain rule $\frac{\partial Q}{\partial \theta_i^{\mu}} = \frac{\partial Q}{\partial \mu(s_t|\theta_i^{\mu})} \frac{\partial \mu(s_t|\theta_i^{\mu})}{\partial \theta_i^{\mu}}$ was used to find the policy gradient..

The policy is updated by using one step of gradient ascent, giving:

$$\boldsymbol{\theta}^{\mu} \leftarrow \boldsymbol{\theta}^{\mu} + \alpha^{\boldsymbol{\theta}^{\mu}} \nabla_{\boldsymbol{\theta}^{\mu}} L(\boldsymbol{\theta}^{Q}). \tag{2.40}$$

Theoretical convergence is not guaranteed when introducing non-linear function approximators, as DNNs. These approximators are essential to be able to generalize when having large state and action spaces. DDPG solves this problem by using replay buffer, and a separate target network for calculating y_t.

The replay buffer is a finite buffer storing transitions (s_t, a_t, r_t, s_{t+1}) during exploration. The actor and critic are updated at each time step by a sampling a small set of transitions (minibatch) uniformly from the buffer. When the buffer is sufficiently large, DDPG is able to learn from a set of uncorrelated transitions. It therefore fulfils the assumptions (of many optimization algorithms) that the samples are independently and identically distributed. The replay buffer can be used due to DDPG being off-policy.

Soft target updates consists of creating a copy of the actor and critic networks, called target critic $Q'(s, a|\boldsymbol{\theta}^{Q'})$ and target actor $\mu'(s|\boldsymbol{\theta}^{\mu'})$. These networks are used for calculating the target values, where the target policy is the optimal policy giving $\mu(s) = \text{argmax}_a Q*(s, a)$ and target Q-function $Q*(s, a)$. The reason for using soft target update is that $Q(s, a|\boldsymbol{\theta}^{Q})$ is used in the loss function. This means the Q-update is prone to divergence, due to the

loss function depending on the parameters being trained. The target-networks are therefore updated by factor $\tau << 1$, slowly tracking the learned networks:

$$\boldsymbol{\theta}^{Q'} \leftarrow \tau\boldsymbol{\theta}^{Q} + (1-\tau)\boldsymbol{\theta}^{Q'}, \tag{2.41a}$$

$$\boldsymbol{\theta}^{\mu'} \leftarrow \tau\boldsymbol{\theta}^{\mu} + (1-\tau)\boldsymbol{\mu'}. \tag{2.41b}$$

This update improves the stability of learning, but might lead to slow learning. In practice the improved stability outweighs slower learning.

Another problem is how to handle features with different scales, as this can lead to problems with too small or too big gradients during backpropogation. This is solved by using batch normalization [87], previously discussed in 2.3.1.

A pseudocode of the DDPG-algorithm is in Algorithm 3.

Algorithm 3 DDPG algorithm, for estimating $\pi_{\boldsymbol{\theta}} = \pi_*$

Input: a differentiable policy parameterization $\mu(s|\boldsymbol{\theta}^{\mu})$ and action-value function parameterisation $Q(s, a|\boldsymbol{\theta}^{Q})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}^{Q}} > 0$ and $\alpha^{\boldsymbol{\theta}^{\mu}} > 0$.
Randomly initialize critic network $Q(s, a|\boldsymbol{\theta}^{Q})$ and actor $\mu(s|\boldsymbol{\theta}^{\mu})$, with parameter $\boldsymbol{\theta}^{Q}$ and $\boldsymbol{\theta}^{\mu}$.
Initialize target network $Q(s, \boldsymbol{\theta}^{Q'})$ and $\mu(a|s, \boldsymbol{\theta}^{\mu'})$, with weights $\boldsymbol{\theta}^{Q'} = \boldsymbol{\theta}^{Q}$ and $\boldsymbol{\theta}^{\mu'} = \boldsymbol{\theta}^{\mu}$.
Initialize replay buffer
for each episode **do**
  Initialize a random process N for action exploration
  Initialize S (first state of episode)
  Loop for each step of the episode t = 0,1,..., T-1
  $A \sim \mu(S|\boldsymbol{\theta}^{\mu}) + N_t$
  Take action A, observe S',R
  Store transition (S,A,R,S') in replay buffer
  Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from replay buffer
  Set $y_i = r_i + \gamma Q(s, \mu'(S|\boldsymbol{\theta}^{\mu'}|\boldsymbol{\theta}^{Q'})$ for $i \in 1...N$
  Update critic by minimizing the loss: $\frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\boldsymbol{\theta}^{Q}))^2$
  Update policy with $\frac{1}{N}\sum_i \nabla_a Q(s_i, \mu(s_i|\boldsymbol{\theta}^{\mu}|\boldsymbol{\theta}^{Q})\nabla_{\boldsymbol{\theta}^{\mu}}\mu(s_i|\boldsymbol{\theta}^{\mu}$
  Update critic target network: $\boldsymbol{\theta}^{Q'} \leftarrow \tau\boldsymbol{\theta}^{Q} + (1-\tau)\boldsymbol{\theta}^{Q'}$
  Update actor target network: $\boldsymbol{\theta}^{\mu'} \leftarrow \tau\boldsymbol{\theta}^{\mu} + (1-\tau)\boldsymbol{\mu'}$
end for

2.3.3.4 Trust region policy optimisation

In general, policy gradient methods adjust the parameters of the policy by taking a step in the direction which have the steepest ascent in expected reward [103]. A dilemma during the update is deciding the size of this step, as a large step can decrease training time, but too

large might delay or prevent convergence. The nonlinear nature of the policy implies that a too large step might lead to missing the optima and delay the training, or not being able to converge at all. Another factor is that a bad update of the policy might not be recovered by subsequent updates, but produce bad experience samples that will be used on subsequent training steps, and further damage the policy. It is therefore desirable to avoid making too large updates. One might use a small learning rate for the SGD, to have smaller steps, but this might significantly slow down the convergence.

Several methods have been proposed to try to create the largest possible steps, with guaranteed improvement. Two of these methods are trust region optimisation (TRPO) and proximal policy optimisation (PPO).

Trust region policy optimization (TRPO) [97] is a policy gradient method. It has an actor-critic architecture, similar to DDPG. The most important difference from DDPG is how the policy parameters of the actor are updated. The policy is updated by taking the largest possible step while limiting the parameter changes with a constraint on the difference between new and the old policy. Trust region policy optimization tends to give monotonic improvements, requiring little tuning of the hyperparameters. It is effective for optimising large nonlinear policies such as neural networks, and experiments havn shown robust performance on a wide variety of tasks, such as learning simulated robotic swimming, hopping, and walking gaits etc.

In order to take the largest possible step while trying to guarantee monotonic improvement, TRPO is using minorisation-maximization (MM) algorithm, trust region, importance sampling, and conjugate gradient algorithm.

The optimization objective

The policy is updated in order to maximise expected discounted reward $\eta(\pi)$:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}[\sum_{t=0}^{\infty} \gamma^t r(s_t)], \tag{2.42}$$

where s_0 is the initial state sampled from the initial state distribution ρ_0, s_{t+1} is the next state from the transition probability s_{t+1} $P(s_{t+1}|s_t, a_t)$ and a is the action from the stochastic policy $\pi : SxA \rightarrow [0, 1]$.

The expected return of another policy $\tilde{\pi}$ can be expressed by the advantage over π, accumulated over timesteps:

$$\begin{aligned}\eta(\tilde{\pi}) &= \eta(\pi) + \mathbb{E}_{s_0,a_0,\ldots\,\tilde{\pi}}[\sum_{t=0}^{\infty}\gamma^t A(s_t,a_t)],\\ &= (\eta) + \sum_{t=0}^{\infty}\sum_{s} P(s_t = s|\tilde{\pi})\sum_{a}\tilde{\pi}(a|s)\gamma^t A_\pi(s,a),\end{aligned} \tag{2.43}$$

where the action a is sampled from the policy $\tilde{\pi}$ and $A_\pi(s,a)$ is the advantage function from Equation (2.18). This identity was given by Kakade & Langford [104].

The unnormalized discounted visitation frequency $\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + ..$ where $s_0 \in \rho_0$, is the frequency of which states are visited under the policy π. By using this unnormalized discounted visitation frequency the equation for $\eta(\pi)$, Equation (2.43), may be expressed as:

$$\eta(\tilde{\pi}) = (\eta) + \sum_{s}\sum_{s}\rho_{\tilde{\pi}}(s)\sum_{a}\tilde{\pi}(a|s)\gamma^t A_\pi(s,a) \tag{2.44}$$

This equation implies that the policy update $\pi \rightarrow \tilde{\pi}$ is guaranteed to increase the policy performance η, or leave it constant, as long as the expected advantage at every state s is nonnegative, i.e. $\sum_a \tilde{\pi}(a|s)\gamma^t A_\pi(s,a) \geq 0$

The discounted visitation frequency $\rho_(\tilde{\psi})$ is strongly dependent on $\tilde{\pi}$, and this makes it difficult to optimise Equation (2.44) directly. Instead a local approximation to η is introduced:

$$L_\pi(\tilde{\pi}) = (\eta) + \sum_{s}\sum_{s}\rho_\pi(s)\sum_{a}\tilde{\pi}(a|s)\gamma^t A_\pi(s,a), \tag{2.45}$$

where L_π uses the visitation frequency from the old policy ρ_π, and thereby ignoring changes in state visitation frequency $\rho_{\tilde{\pi}}$ due to changes in the policy.

The approximation L_π matches η to first order, if a parameterized policy π_θ is differentiable by the parameter vector θ, shown by Kakade & Langford [104]. This implies that improvements on $L_{\tilde{\psi}}$ will also improve η, as long sufficiently small steps are used when updating the policy. The equation does not, however, give guidance on how big or small the step should be.

Monotonic improvement guarantee for general stochastic policies

To address the issue of how small the step should be, an explicit lower bound on the improve-

ment of η is generated. The following bound holds for the expected discounted reward:

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{kl}^{\max}(\pi, \tilde{\pi}) \tag{2.46a}$$

$$C = \frac{4\sigma\gamma}{(1-\gamma)^2}, \tag{2.46b}$$

$$sigma = \max_{s,a} |A_{\pi}(s,a)|, \tag{2.46c}$$

$$D_{KL}^{\max}(\pi, \tilde{\pi}) = \max_{s} D_{KL}(\pi(\cdot|s)||\cdot\tilde{|}s)) \tag{2.46d}$$

where $D_{KL}^{\max}$ is the maximum KL-divergence. The KL-divergence is a sort of measurement of the distance between the two policies π and $\tilde{\pi}$.

By maximizing the lower bound $M = L_{\pi}(\tilde{\pi}) - CD_{KL}^{\max}(\pi, \tilde{\pi})$ iteratively, one can guarantee that the true objective η is non-decreasing, meaning that the training will cause monotonical improvements. This is therefore a minorization-maximization (MM) algorithm [105].

Optimization of parameterized policies

The policy $\pi_{\theta}(a, s)$ is parameterized with parameter vector θ. For the paramterised policy, the relation $\eta(\theta) \geq L_{old}(\theta) - CD_{KL}^{\max}(\theta_{old}, \theta)$ still holds true, from Equation (2.46). This yields the following maximisation, which guarantees to improve the true objective η:

$$\max_{\theta}[L_{\theta_{old}}(\theta) - CD_{KL}^{\max}(\theta_o ld, \theta)] \tag{2.47}$$

where θ_{old} denotes the previous policy paramters, and $\eta(\theta) := \eta(\pi_{theta})$, $L_{\theta}(\tilde{\theta}) := L_{\pi_{\theta}}(\pi_{\tilde{\theta}})$ and $D_{KL}(\theta||\tilde{\theta}) := D_{KL}(\pi_{\theta}||\pi_{\tilde{\theta}})$.

If the penalty coefficient C, from Equation (2.47), is used as recommended, the step size would be very small. In order to take larger steps in a robust way, a trust region constraint is used. The constraint specifies how close the new and old policies are allowed to be, and is expressed in terms of KL-divergence between the new and old policy. This means that the objective is:

$$\begin{aligned} &\max_{\theta} L_{\theta_{old}}(\theta) \\ &\text{s.t. } D_{KL}^{\max}(\theta_o ld, \theta) \leq \delta \end{aligned} \tag{2.48}$$

Since the trust region constraint is imposed at every point in the state space, it is impractical to solve due to the large number of constraints. Instead, Schulman [106] uses a heuristic approximation which considers the average KL divergence:

$$\bar{D}_{KL}^{\rho} := \mathbb{E}_{s\ \rho}[D_{KL}(\pi(\cdot|s)||\cdot\tilde{|}s))]], \tag{2.49}$$

where ρ is the state distribution.

This leads to the following optimisation problem to generate a policy update, also called the TRPO update:

$$\begin{aligned} &\theta_{k+1} = argmax_{\theta} L_{\theta_k}(\theta) \\ &\text{s.t. } \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_k, \theta) \leq \delta, \end{aligned} \tag{2.50}$$

for iteration k, which represents the old policy.

The objective and constraints in Equation (2.48), can be written in terms of expectations. To do this several tricks are used, as explained by Schulman [106]. One of them is to use an importance sampling estimator, which lets TRPO choose samples from an old policy, instead of only using samples from the current policy, leading to better sample efficiency. The optimisation problem written in terms of expectation is:

$$\begin{aligned} &\mathbf{E}_{s\ \rho_{\theta_{old}}, a\ q}\left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)A_{\theta_{old}(s,a)}}\right] \\ &\text{s.t. } \mathbf{E}_{s\ \rho_{\theta_{old}}}[\bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta)] \leq \delta \end{aligned} \tag{2.51}$$

where q does the sampling distribution.

Exploration vs Exploitation

TRPO is an on-policy algorithm and uses a stochastic policy. This means that it explores by sampling actions according to the current stochastic policy, and the amount of randomness in action selection depends on initial conditions and training procedures. The update rule encourages to exploit rewards that has already been found increasingly, and helps therefore the policy to become less random over the course of the training. The downside is that this may also cause the policy to get trapped in local optima [107].

2.3.3.5 Proximal policy optimization

Proximal policy optimization (PPO) was made by Sculman et al. [98], and is a policy gradient method which works in discrete and continuous environments. It is motivated by the same questions as TRPO, of how to take as large a step as possible without performance collapse, using the current available data. The algorithm attains the efficiency and reliable performance of TRPO, but is simpler to implement, more general and have better sample complexity (empirically).

There exist two primary variants of PPO:

- PPO-penalty
- PPO-clip

In this thesis only PPO-clip is presented, since Schulman generally finds PPO-clip to perform best.

PPO-clip

The objective TRPO maximizes, in Equation (2.51), can be rewritten as:

$$L^{CPI}(\theta) = \hat{\mathbf{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] = \hat{\mathbf{E}}_t[r_t(\theta)\hat{A}_t], \tag{2.52}$$

where $r_t(\theta)$ denotes the probability ration $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. The probability ration $r_t(\theta) = 1$, when the new policy is the same as the old, meaning $\theta = \theta_{old}$.

In TRPO a the KL-divergence constraint was added so that the maximization of L^{CPI} would not lead to excessively large policy updates. PPO-clip solves the problem by modifying the objective. It penalises changes to the policy that moves $r_t(\theta)$ away from 1. The main objective is:

$$L^{CLIP}(\theta) = \hat{\mathbf{E}}_t[\min([r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.53}$$

where ϵ is a hyperparameter and clip is linear a saturation function, with lower limit $1-\epsilon$ and upper limit $1+\epsilon$).

The first term inside the min is L^{CPI}, meaning $L^{CPI} = L^{CLIP}$ in first order around $\theta = \theta_{old}$. The second term in the min function modifies the surrogate objective by clipping the probability ratio. This removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$. By taking the minimum of the unclipped and clipped objective, the final objective is a lower

bound (i.e. pessimistic bound) on the unclipped objective. This means that the changes in the probability ratio is ignored when it makes the objective improve, and is handled by an upper or lower limit when it makes the objective worse.

The expression for L^{CLIP} is quite complex, and the intuition behind it can be easier explained through the simplified version [107]:

$$L^{CLIP}(\theta) = \hat{\mathbf{E}}_t[\min([\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t, g(\epsilon, \hat{A}_t))] \tag{2.54a}$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & , A \geq 0 \\ (1-\epsilon)A & , A < 0 \end{cases}, \tag{2.54b}$$

The intuition behind the simplified objective is:

- If the advantage is positive, the objective will increase if an action becomes more likely, meaning $\pi_\theta(a|s)$ should be increased. However, the min term limits how much the objective can increase, with a upper limit of $(1+\epsilon)A$. This means the new policy is not benefiting for going too far away from the old policy.
- If the advantage is negative, the objective will increase if the action becomes less likely, meaning $\pi(a|s)$ should be decreased. However, the min term limit how much the objective can decrease, with a lower limit of $(1-\epsilon)A$. This means the new policy will not benefit for going too far away from the old policy.

The policy is updated through:

$$\theta_{k+1} = argmax_\theta \mathbf{E}_{s,a^\pi_{\theta_k}}[L^{clip}(s, a, \theta_k, \theta)], \tag{2.55}$$

where the objective L^{clip} is optimised using stochastic gradient ascent. In order to calculate the policy update, it is necessary to have an estimation of the advantage $A^{\pi,\gamma}(s_t, a_t)$. It is desired that the advantage estimation has low variance. One popular technique for advantage estimation is generalised advantage estimation (GAE) [108].

Advantage-function estimation

The generalised advantage estimation GAE(γ, λ) [108] is an variance reduced estimator of the advantage, which tries to keep the bias below a tolerable level. The advantage is estimated as the discounted sum of TD errors:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\inf} (\gamma\lambda)^l \delta_{t+1}^V, \tag{2.56}$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$, i.e. the TD error of V with discount γ. δ_t^V can be considered an estimate of the advantage of the action a_t.

The generalised estimator for $0\lambda < 1$ makes a compromise between bias and variance, controlled by λ. γ and λ both contribute to the bias-variance trade-off when using an approximate value function.

One popular style of the GAE that can be easily applied to PPO (or any policy-gradient-like algorithm) is [98]:

$$A_t(s_t, a_t) = r_t + \gamma r_{t+1} + \ldots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) - V(s_t), \tag{2.57}$$

where T denotes the maximum length of a trajectory but not the terminal time step of a complete task, and γ is a discounted factor. If the episode terminates, $V(s_T)$ can be set to zero, meaning $A_t = G_t - V(s_t)$ where G_t is the discounted return following time t.

In order to calculate the GAE, it is necessary with a learned value function V_{ϕ_k}. A variety of different methods can be used to estimate the value function [108]. Similarly to DDPG, the value function can be estimated using the Monte Carlo (or TD(1)) approach [46]. The difference is that for PPO a state-value function is estimated, while an action-value function is estimated in DDPG.

The Monte-Carlo estimation of the value-function consists of solving the nonlinear regression problem::

$$\min_\theta \sum_{n=1}^{N} ||V_\phi(s_n) - \hat{V}_n||^2, \tag{2.58}$$

where $\hat{V}_t = \sum_{l=0}^{\inf} (\gamma^l r_{t+1})$ is the discounted sum of rewards, and n indexes over all timesteps in a batch trajectory. This is the simplest approach when using a nonlinear function approximator (such as ANNs) to represent the value function.

Pseudocode of PPO-Clip

The PPO algorithm consists of:

- For each iteration the actor collects T timesteps of data.

- Compute advantage estimates, $\hat{A}_t$, using current estimate of value function
- Use the data to optimize the constructed surrogate loss with minibatch SGD and fit value function by regression on mean squared error typically via some SGD-algorithm.

A pseudocode for Actor-Critic PPO-Clip that uses fixed-length trajectory segments, using Monte Carlo for estimation of value-function and general advantage estimation is found in Algorithm 4, from [107].

Algorithm 4 PPO-Clip algorithm, for estimating $\pi_{\boldsymbol{\theta}} = \pi_*$

Input: initial policy parameteters θ_0, initial value function parameters ϕ_0
for each epoch **do**
  Initialise S (first state of episode)
  Sample set of trajectories $D_k = \tau_k$, sampling for t = 0,1,..., T-1
  $A = \pi_k = \pi_(\theta_k)$
  Take action A, observe S',R
  Store transition (S,A,R,S') in replay buffer
  Compute $\hat{V}_t$
  Compute advantage estimates, $\hat{(A)}_t$, based on current value function V_{ϕ_k}
  Update the policy for maximising the PPO-clip objective
  Fit value function by Monte-Carlo method
end for

Exploration vs Exploitation

Similarly to TRPO, PPO is an on-policy algorithm, which trains a stochastic policy by exploring samples from the latest version of the stochastic policy.

Both PPO, TRPO and DDPG have random exploration. This makes it more unlikely to successfully solve a problem with a sparse reward function [109]. One of the methods that has been developed to help with exploration is reward shaping. Reward shaping is a technique where supplemental rewards are provided to make a problem easier to learn [110, 111]. The goal of reward shaping is to shape the original delayed reward (such as reaching berth) to reward appropriate intermediate actions as they happen.

2.4 Explainable AI

Being able to understand and trust a certain prediction can be as crucial as the accuracy of the prediction for many applications [112, 113]. Being able to interpret a prediction model's output engenders appropriate user trust, provides insight into how a model may be improved, and supports understanding of the process being modelled. Often complex models are used

to achieve high accuracy, but even experts can struggle to interpret these model. This creates a dilemma between accuracy and interpretability.

Deep reinforcement learning uses artificial neural networks, which is a black box method, for function approximation. Despite the efficiency and versatility of NN-based DRL agents, their decisions often remain incomprehensible, reducing their utility. Trust is fundamental if DRL is going to be accepted for use in the real world, such as with autonomous vessels. It is therefore imperative to develop explainable DRL agents to increase trust.

There exist several perspectives of how one can trust a given model [113]. Two examples are:

- **Trusting a prediction**: Can an individual prediction be trusted to such a degree that it can be acted upon? For instance: A prediction of a model cannot be acted upon in blind faith within the field of medical diagnosis or terrorism, as the consequences might be catastrophic.

- **Trusting a model**: Can the user trust a model to behave in a reasonable way if deployed? The users need to be confident that the model will perform well on real-world data, as there often is no time to verify the trustability of each prediction. One example is embedded systems, with no time to verify the trustability of all the predictions.

The examples also remain valid within DRL. For DRL models, the question can be: Is it possible to trust a specific action (prediction) or trust the DRL-agent (model)?

Both the aspects above are directly affected by the user's understanding of a model's behaviour, and not merely as a black-box which gives accurate actions. Another reason for inspecting the model's behaviour is to see if the model serves the objectives it is designed to do, which might not always be seen from the evaluation metric. For instance, when training a model to predict the presence of a polar bear, there is often ice present around the bear. The prediction model might find the distinction in the training set; it is a polar-bear if it is ice in the image, and a brown-bear if not. This might yield high accuracy, but the model does not fulfil the objective of the designer fully if it was to detect polar bears under any circumstance.

To help users interpret the predictions of complex models, several methods have been proposed, such as the unified framework Shapley Additive exPlanations (SHAP) [112]. The method has shown improved computational performance and/or better consistency with human intuition than other previous approaches.

2.4.1 Shapley additive explanations

SHAP is a post-hoc interpretability method that helps users to interpret the output of a model, by Lundberg et al. [112]. A post-hoc method tries to interpret an already trained model [114]. The trained model is denoted f, and has an input x, and an output y. Lundberg et al. consider the input to consist of different features (variables), and the output is called a prediction.

SHAP creates values to describe the importance, or contribution, of each feature to a prediction by the model [115]. These SHAP values are a type of Shapley values, from game theory [112]. Shapley values give a "fair" way of distributing the importance a feature has to a prediction. The method is "fair" due to its properties, which are described later. By approximating Shapley values, SHAP values of a feature represent the importance of that feature for a given prediction. In order to generate these SHAP values, SHAP creates an explanation model.

An explanation model is any interpretable approximation of the origin trained model f. For a simple model, the best explanation model is the model itself, since it perfectly represents itself and is easy to understand. A complex model, such as a DNN, is not easy to understand and consequently not its own best explanation. Therefore it is a need to create a simpler explanation model. The explanation model is denoted g.

SHAP is an additive feature attribution method and a local method. It is called a local method since it explains a prediction y based on a single input x. The goal of SHAP is to explain the prediction y of an instance x by computing the importance of each feature of x [116]. All these importance's should add up to the prediction y, and SHAP is therefore called an additive feature attribution method [112]. This is further described in the following section.

2.4.1.1 Additive feature attribution methods

The explanation model in SHAP is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i, \tag{2.59}$$

with binary variable $z' \in 0, 1^M$, M is the number of simplified input features, and effect $\phi_i \in \mathbb{R}$. The binary variable is a simplified input z', where the original input z can be found by using the mapping function $z = h_z(z')$. A local method tries to ensure $g(z') \approx f(h_x(z'))$ whenever $z' = x'$.

Additive feature attribution methods have an explanation model that attributes an effect ϕ_i to each feature, and summing up all this effects should approximate the output $f(x)$.

Additive feature attribution methods have several desirable properties, which include:

1. **Local accuracy**: Explanation model g must be equal to the original model f, when $x = h_x(x')$.
2. **Missingess property**: If a simplified input is missing in the original input, it should have zero impact on the explanation model. This means that if $x'_i = 0 \rightarrow \phi_i = 0$.
3. **Consistency states**: If a model changes so that some simplified input's contribution increases or stays the same regardless of the other inputs, that input's effect should not decrease.

There is only one way to calculate the effect ϕ_i such that the explanation model g is an additive feature attribution method and satisfies the three properties. The effect ϕ_i is therefore:

$$\phi_i(f, x) = \sum_{z' \in x'} \frac{|z'|!(M - |z'| - 1)!}{M!}[f_x(z') - f_x(z'\ i)] \tag{2.60}$$

where $|z'|$ is the number of non-zero entries in z',and $z' \in x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x'.

Here ϕ_i are the Shapley values, from cooperative game theory [117]. Young [117] discovered that Shapley-values are the only set of values that satisfies the three properties mentioned above. For a given simplified input mapping h_x, there is only one possible additive feature attribution method which fulfils the three properties. This implies that methods not based on Shapley-values violate local accuracy and consistency.

2.4.1.2 SHAP values

SHAP uses the mapping function $f_x(z') = f(h_x(z')) = E[f(z)|z_S]$, where S is the set of non-zero indices in z' and $E[f(z)|z_S]$ is the expected value of the function conditioned on a subset S of the input features. SHAP values combine these conditional expectations with the classic Shapley values from game theory to attribute ϕ_i to each feature, using Equation (2.60). The SHAP value of a feature is therefore the average expected marginal contribution of a feature to a given prediction after all possible combinations of states have been considered. SHAP values are therefore proposed as a unified measure of a feature importance to a prediction.

A more intuitive explanation of this is that SHAP values explain the output of a function f

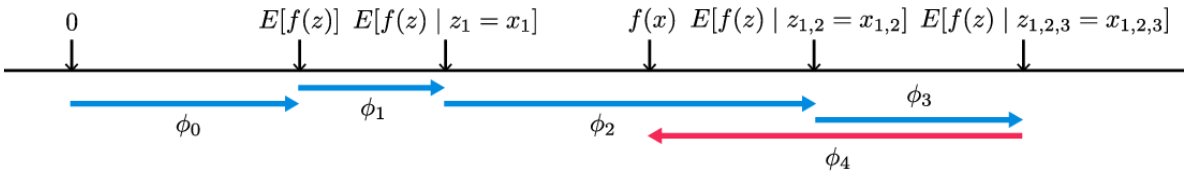


Figure 2.8: An example of how SHAP values attribute to each feature the change in the expected model prediction when conditioning on that feature. Source [112].

as a sum of the effects ϕ_i of each feature being introduced into a conditional expectation, as illustrated in Figure 2.8. They can be thought of as explaining how to get from the base value $\mathbb{E}[f(x)]$ to the prediction $f(x)$, where $\mathbb{E}[f(x)]$ is what would be predicted without prior knowledge of any features to the current output $f(x)$. Figure 2.8 shows a single ordering. Unfortunately, the ordering matters when the model is non-linear or the input features are not independent. SHAP values are therefore calculated as the average ϕ_i across all possible orderings, for all possible combinations of states. The exact computation of SHAP values is therefore challenging and are, therefore, often approximated. Three approximation methods of SHAP values are Kernel SHAP, Tree SHAP and Deep SHAP. In the following section, only Kernel SHAP will be introduced, due to its versatility.

2.4.1.3 Kernel SHAP

Kernel SHAP interprets individual model predictions based on locally approximating the model around a given prediction. It is a model-agnostic estimation of SHAP values, meaning the method does not depend on any special form of the prediction model f.

To find ϕ Kernel SHAP minimizes the objective function:

$$argmin_{g \in G} L(f, g, \pi_{x'}) \tag{2.61}$$

In order to recover the Shapley values, the loss function L and weighting kernel $\pi_{x'}$ is defined as:

$$\pi_{x'}(z') = \frac{(M-1)}{(\binom{M}{|z'|})|z'|(M-|z'|)} \tag{2.62a}$$

$$L(f, g, \pi_{x'} = \sum_{z \in Z} [f(h_x^{-1}(z') - g(z')]^2 \pi_{x'}(z')], \tag{2.62b}$$

where $|z'|$ is the number of non-zero elements in z'. When $|z'| \in 0, M$ to leads to $\pi_{x'} = \infty$. In practice these scenarios are therefore usually eliminated.

The SHAP values can be found using regression, assuming independent features and a linear model. This leads to better efficiency than direct use of the classical Shapley equations.

The SHAP values are calculated through 5 steps [118]:

- Sample simplified input vector z'_k, $k \in 1, ..., K$, where 1 means the feature is present and 0 the feature is absent. K is the number of samples.
- Get a prediction for each z'_k by first transforming back to input vector x' and then applying to prediction model f: $f(h_x(z'_k))$.
- Compute the weight for each z'_k
- Fit weighted linear model by optimizing (2.61).
- Return approximations of Shapley values ϕ_k, which are the coefficients from the linear model.

The impact of a feature is determined by setting a feature to missing, and observing the change in the model output [9]. The SHAP values are calculated as the average marginal contributions across all the permutations. However, most models cannot handle arbitrary patterns of missing input values, and therefore "missing" is simulated by replacing features with values taken from a background dataset.

2.4.1.4 Interpretation of SHAP values

SHAP values can be used for both global and local importance (contributions) [116, 119]. The local importance value is the SHAP-value for a single prediction. The global importance can be found by calculating the SHAP values for an entire dataset. It represents the overall importance of a feature in the model, and gives an interpretation of the models overall behaviour. The global feature importance (contribution) can be calculated as the mean absolute SHAP per feature across several instances. Global interpretations are consistent with local interpretations, since the Shapley values are the "atomic unit" of the global interpretation.

Chapter 3

Design and implementation

Docking is considered a challenging and complex task, where several objectives need to be considered simultaneously. This includes, among others, obstacle avoidance, target tracking, and obeying speed limits. The main contribution of this thesis is to investigate how to use deep reinforcement learning (DRL) to develop an automatic docking solution, resulting in a progressive methodology creating an end-to-end docking model. To get insight into the reasoning of the DRL-based controllers, the influences of states upon the selection of control inputs were analysed, using the SHAP-algorithm from the field of explainable AI (XAI).

A DRL agent learned a docking policy by interacting with a simulation of a fully-actuated cargo vessel in a docking area. It interacted with the vessel by controlling thruster forces and angles and received feedback through a reward and the new state of the vessel. Through exploring these interactions, the DRL agent learned an end-to-end docking policy. This policy solved the entire suite of tasks, such as thruster allocation, speed control, obstacle avoidance and target tracking. One major advantage of a DRL-based docking policy is that there is no need for prior knowledge of the harbour or the dynamics of the marine vessel. Another advantage is that it does not depend on the performance of subsystems, such as separate motion control and thruster allocation systems.

The new state of the vessel and the reward was calculated based on the vessel dynamics, ocean currents and information about the harbour. It was in addition created transformed states, which were correlated to the objectives of docking, to help the agent learn faster.The system calculating the new states of vessel and rewards is called the environment, where the interaction between the DRL agent and environment is illustrated in Figure 3.1. The design of a fitting reward function, state vector and action vector was part of formulating the docking problem as a Markov decision process. The DRL agent tried to solve the Markov decision process by learning a policy optimising the docking objective.

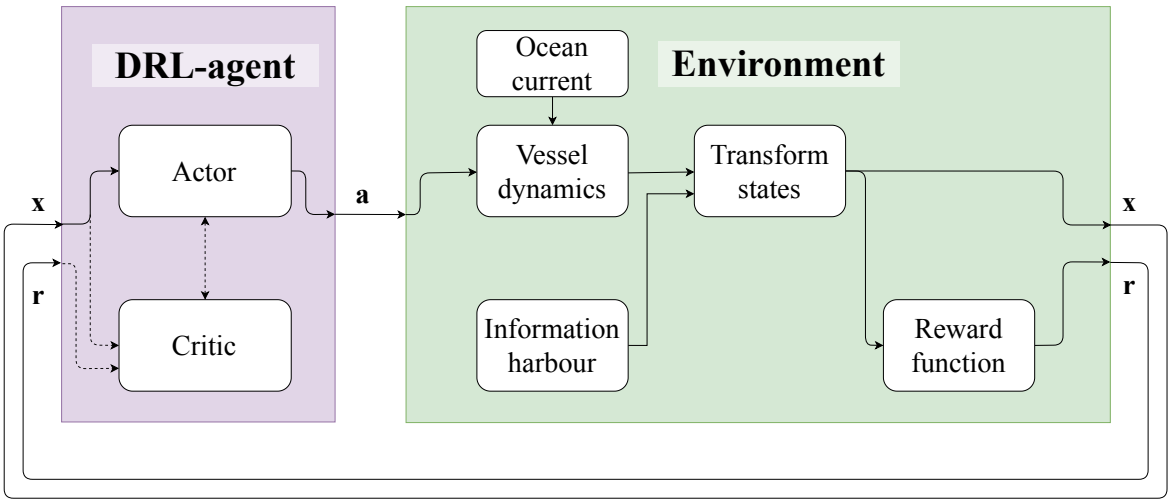


Figure 3.1: Illustration of interaction between DRL agent and the environment. The DRL agent consist of an actor and a critic, where the actor gives action a based on state x. The environment calculates its new state x and reward r based on previous state and given action a. The dotted lines illustrate links that are only active during training, where the critic and reward r are only used to learn a policy.

In this thesis, two state-of-the-art DRL algorithms were explored: Deep deterministic policy gradient (DDPG) and proximal policy gradient (PPO). The selected DRL algorithms have policies approximated using neural networks. Neural networks are considered as black-box methods and are not easy to interpret or to judge with respect to reliable and predictable behaviour. In order to gain some trust of the black-box policy, the policy was analysed using an explainable AI (XAI) method called Shapley additive explanations (SHAP). This method is used to assess each state's contribution to the agent's decisions. It engenders user trust by adding more interpretability of the policy and provides insight into how a model may be improved.

In the following sections, the design choices and corresponding implementations are presented and discussed. This is an overview of the following sections:

- Section 3.1 formulates the docking scenario of this thesis.
- Section 3.2 presents the simulation of the vessel's dynamics, representation of harbour and geometry of the docking problem.
- Section 3.3 presents how docking can be solved using deep reinforcement learning.
- Section 3.4 presents the implementation details of the selected DRL algorithms: PPO and DDPG.
- Section 3.5 presents the setup of SHAP.

- Section 3.6 presents the programming language and libraries used in this thesis.

3.1 Formulating the docking scenario

Even though docking is old craftsmanship, it is lacking an established and precise definition of which tasks the docking process entails. In this thesis, the docking scenario was considered to consists of the following phases:

1. **Approach**:
    - Controlling the vessel from just outside the harbour to a point close to the berth.
    - Slowing down the speed before entering the harbour, obeying the harbour speed limits.
2. **Berthing**:
    - Controlling the vessel from the proximity of the berth, to the actual berth and holding the desired berth pose using small alternations of thruster forces and angles.
    - Obeying the speed limits.
3. **Mooring**: Securing the vessel to the quay.

The third phase, mooring, was not considered a part of the scope, being a purely mechanical operation. Only the first two docking phases, illustrated in Figure 3.2, were explored.

In this thesis, the harbour was considered to be the area containing the berth and having speed regulations. The docking area was defined as the area within the harbour, where the vessel was performing the docking phases.

The docking scenario was based on the layout of Trondheim harbour, where the coastal liner "Hurtigruten" normally docks. This area is illustrated in Figure 3.3 and Figure 3.2. In Figure 3.3, the docking area was defined by the orange polygon, and the blue polygon defined the harbour. The harbour was scaled up with 55 % compared to the original size, to provide the agent with more exploration space. This increased the agent's opportunity to understand the dynamics of the vessel and control objectives during training.

It is common to let two perpendicular markers along the quay delimit the extent of a berth where the vessel will be parked, close to the fenders. The fenders in Trondheim extend approximately 60-120 cm from the quay [80].

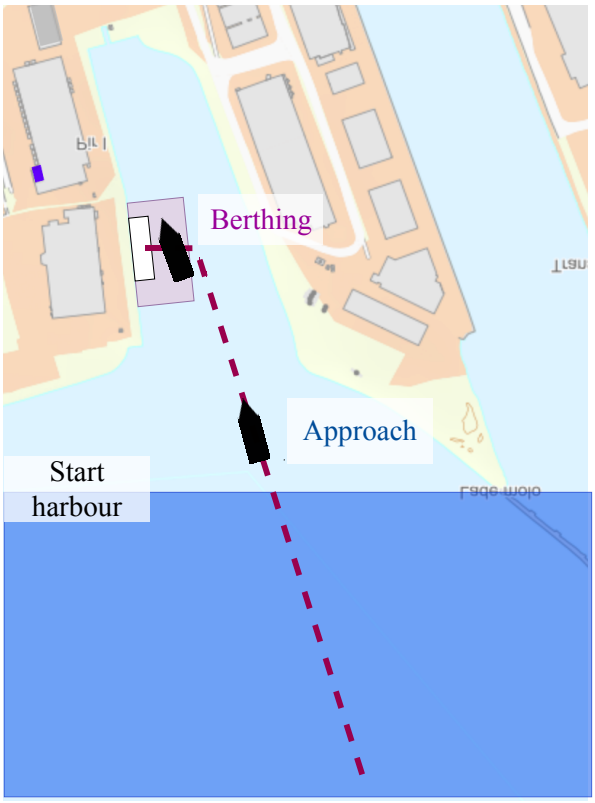


Figure 3.2: Illustration of the two phases, approach and berthing. The area outside the harbour is marked with a darker blue colour, while the harbour is the light blue and the purple area. The harbour has a speed limit of 2.5 m/s. The approach phase concerned both the darker blue and lighter blue areas, while berthing was performed in the purple area.

During docking, the shipmaster can either berth based on the markers on the quay, or by coordinates. Hence berthing can be considered either:

- **Controlling a vessel inside a berth rectangle with equal width and length as the vessel.** A rectangle was placed so that the edge of the vessel was 130 cm from the quay and was called a berth rectangle.
- **Controlling a vessel to reach a berth pose.** A berth pose was placed so that the vessel was parked 130 cm from the quay, equivalent to the objective of the berth rectangle.

In this thesis, the harbour has the following specifications, as illustrated in Figure 3.3:

- The harbour was defined as the blue polygon in Figure 3.3. Inside the harbour, the speed limit was set to 2.5 m/s as this seems to be the most common speed limit in harbours in Norway. There were no other speed limits in the simulated environment.
- The centre of the berth is set at $(x, y, \psi) = (800, 517.8, -\pi/27)$, leading to a minimum distance of approximately 130 cm from the quay when the vessel was parked.

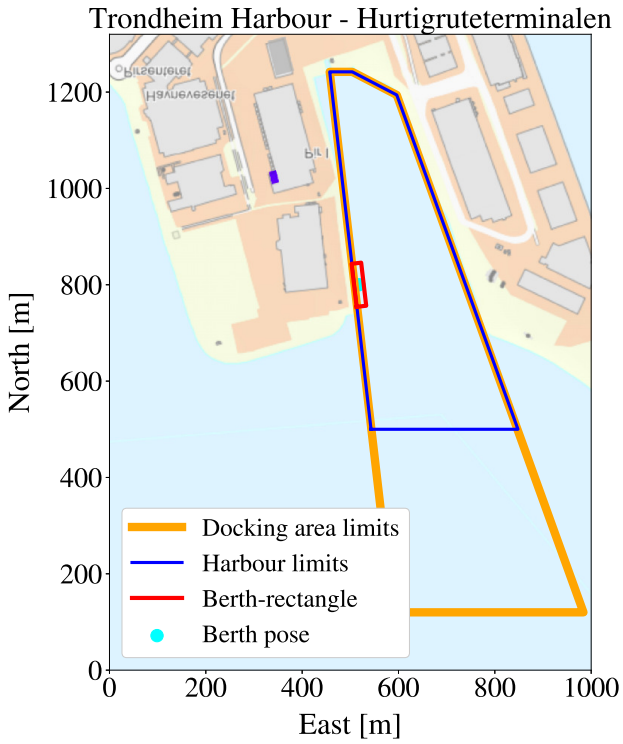


Figure 3.3: Illustration of Trondheim harbour, focused on the area around Trondheim "Hurtigrute kaia". The Docking area is marked orange, the harbour is marked blue, berth rectangle is marked red, and berth pose is the cyan dot.

- Berthing was considered to start one to two boat lengths from the berth, inspired by Nguyen et al. [27].
- There are no other vessels in the harbour. This means that the agent does not have to pay attention to regulations as in COLREGS.
- The only external force in the environment was a constant ocean current in north-east-down (NED) reference frame. This excluded forces such as waves, wind and any shaping effects of external forces due to properties of the harbour itself. Trondheim harbour did not have any measurements of ocean currents. The forecasts of ocean currents were therefore analysed from the Norwegian weather forecast service yr.no [120]. In the time frame November to December 2019, the ocean currents were observed in the range of 1-25 cm/s. For this reason, the vessel was tested for a worst-case of 0.25 m/s, and sea current heading of 45 degrees in NED-frame.
- Outside the designated docking area, everything was considered out of bounds and modelled as land. This means that the sea area outside the docking area polygon in Figure 3.3 was considered land.

3.2 Simulation of the marine vessel and harbour

Two of the environment's tasks are to generate rewards and states based on the agent's actions and previous states. In the next sections, the calculations needed to express the objectives of docking and states of the environment are presented. These include:

- Section 3.2.1: Dynamics and shape of the marine vessel.
- Section 3.2.2: Representations of the harbour.
- Section 3.2.3: Geometry of the docking problem.

The dynamics of the marine vessel were used to calculate the vessel's new position and velocity, after applying a control action. The information about the vessels pose and velocity, together with information about the harbour, was then used to calculate states that complied with the control objectives of the docking problem. This logic is illustrated by the "vessel dynamics"-,"information harbour"- and "transform states"-blocks in Figure 3.1.

3.2.1 Dynamics and shape of the marine vessel

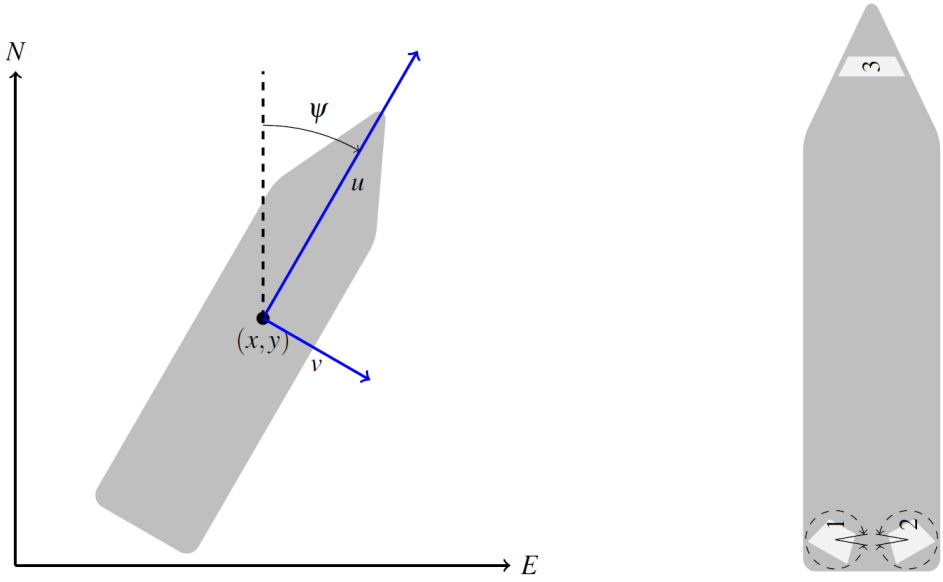
The 3-DOF container ship model applied in this thesis was 76.2 meters long and has a dead weight of 6000 tonnes, from Martinsen et al. [3]. The degrees of freedom consisted of the position vector $\boldsymbol{\eta} = [x, y, \psi]^\top \in \mathbb{R}^2$, and the velocity vector $\boldsymbol{\nu} = [u, v, r]^\top \in \mathbb{R}^2$, with Cartesian coordinates (x, y), yaw angle ψ, linear velocities (u,v), and yaw rate r. The ship was modelled with a rigid-body mass matrix $\mathbf{M}$ and constant damping matrix $\mathbf{D}$, and no external forces, giving the equations of motion:

$$\dot{\boldsymbol{\eta}} = J_\theta \boldsymbol{\nu}, \tag{3.1a}$$

$$\mathbf{M}\dot{\boldsymbol{\nu}}_r + \mathbf{D}\boldsymbol{\nu}_r = \boldsymbol{\tau}_{\text{control}}. \tag{3.1b}$$

The vessel was propelled by two azimuth thrusters in the aft, and one tunnel thruster in the bow, as shown in Figure 3.4b. This meant the vessel had five thruster inputs consisting of the three forces $\mathbf{f} = [f_1, f_2, f_3]$ and two angles $\boldsymbol{\alpha} = [\alpha_1, \alpha_2]$. The agent directly controlled these thruster angles and forces, and a mapping to control forces and moments $\boldsymbol{\tau}_{\text{control}}$ was therefore needed. The mapping is given by the thruster allocation Equation (2.8).

A small current was added to assess the robustness of the DRL agent. The current was modelled as a 2-D irrotational ocean current model (2.6), with $V_c = 0.25$ m/s and $\beta_c = 45°$.



(a) A 3-DOF marine vessel in a NED-reference frame, centred at (x,y), with heading ψ, surge u and sway v. Illustration from [3]

(b) The marine vessel consisted of 3 thrusters. Number 1 and 2 were azimuth thrusters, and 3 was a tunnel thruster. Illustration from [3]

Figure 3.4: Illustrations of the cargo vessel

The agent interacted with the environment by setting thruster input $\mathbf{a} = [f_1, f_2, f_3, \alpha_1, \alpha_2]^\top$, based on received measurements of the state. The new states were calculated based on the thruster input given by the agent, and thereby solving the 3-DOF equation of motion, using Euler's method:

$$\boldsymbol{\eta}_{t+1} = \mathbf{R}(\psi_t)\boldsymbol{\nu}_t h + \boldsymbol{\eta}_t, \tag{3.2a}$$

$$\boldsymbol{\nu}_{r,t+1} = (\mathbf{M}^{-1}(\boldsymbol{\tau}_t(\boldsymbol{\alpha}_t, \mathbf{f}_t) - \mathbf{D}\boldsymbol{\nu}_{r,t}))h + \boldsymbol{\nu}_{r,t}, \tag{3.2b}$$

with step size $h = 1$ s. With a step size of 1s and the agent applying one action per time step, the control rate was 1 Hz.

Additional states can easily be calculated, such as distance to berth and quay, then knowing the new states $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$.

For docking purposes, the ship's shape was approximated by a pentagon, with the bow thrusters placed at the fore and azimuth thrusters in the respective corners of the aft. The shape is visualised in Figure 3.5, with the following spatial constraints S_b:

$$S_b \in \{0 \geq \mathbf{A}_b \mathbf{p}^b - \mathbf{b}_b\}, \tag{3.3}$$

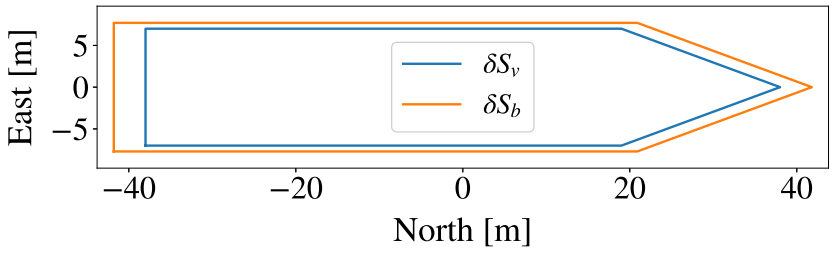


Figure 3.5: Illustration of the edge of the vessel set ∂S_v and vessel set with safety margin ∂S_b.

where $\mathbf{p}^b = [x^b, y^b]$ are Cartesian coordinates in body and matrix $\mathbf{A}_b$ and vector $\mathbf{b}_b$ are constants, with values presented in Appendix A.2.

Inspired by Martinsen et al. [3], a safety margin was added to the shape of the vessel, by dilating the vessel set S_v by 10%. The safety boundary $S_b \subset S_v$ is illustrated in figure 3.5, and wasused to ensure that the vessel did not crash into the quay when docking. This extended vessel shape was used to calculate the control objectives, instead of the original shape. This meant that when parking the vessel 130 cm from the quay, the distance to the edge of the vessel from the quay was in reality 137.5 cm, including the additional 7.5 cm in safety margin of the vessel. The fenders would then be 17.5 cm from the vessels edge, instead of the intended 10 cm.

3.2.2 Representation of the harbour

The docking area was represented as a convex set S_s, inspired by Martinsen et al. [3], meaning:

$$S_s \in \{0 \geq \mathbf{A}_s \mathbf{p}^n - \mathbf{b}_s\}, \tag{3.4}$$

where $\mathbf{p}^n = [x^n, y^n]$ are Cartesian coordinates in NED-frame and matrix $\mathbf{A}_s$ and vector $\mathbf{b}_s$ are constants, see appendix A.3. The convex set is illustrated as the orange polygon in Figure 3.3.

The vessel shape and the docking area are convex sets. This implies that the vessel (including the safety margin as presented in 3.2.1) will not intersect with the land as long as all the vertexes of the vessel are inside the docking area. Martinsen et al. [3] expressed that the

vessel is within the spatial constraints as long as all the vertices of the vessel boundary follow the linear inequality representing the spatial constraints, meaning:

$$\mathbf{A}_s\mathbf{p} - \mathbf{b}_s < \mathbf{0}, \forall \mathbf{p} \in \text{Vertex}(S_b) \tag{3.5}$$

A binary variable l was defined based on the inequality above. The binary variable is true if the entire vessel (including the safety margin) is in the docking area set S_s:

$$l = \begin{cases} 1 & , \mathbf{A}_s\mathbf{p} - \mathbf{b}_s < \mathbf{0}, \forall \mathbf{p} \in \text{Vertex}(S_b) \leq 0 \\ 0 & , \text{otherwise} \end{cases} \tag{3.6}$$

The harbour was expressed as the convex set S_h:

$$S_h \in \{0 \geq \mathbf{A}_h\mathbf{p}^n - \mathbf{b}_h\}, \tag{3.7}$$

where the matrix $\mathbf{A}_h$ and the vector $\mathbf{b}_h$ are constants, see appendix A.4, and $\mathbf{p}^n = [x^n, y^n]$ is the Cartesian position in NED-frame.

The vessel is within the spatial constraints of the harbour as long as all the vertices of the vessel boundary follow the linear inequality representing the spatial constraints because the set is convex. This means that the vessel is inside the harbour if:

$$\mathbf{A}_h\mathbf{p} - \mathbf{b}_h < \mathbf{0}, \forall \mathbf{p} \in \text{Vertex}(S_v) \tag{3.8}$$

The binary harbour variable h informs if the vessel is in the harbour set S_h:

$$h = \begin{cases} 1 & , \mathbf{A}_h\mathbf{p} - \mathbf{b}_h < \mathbf{0}, \forall \mathbf{p} \in \text{Vertex}(S_h) \leq 0 \\ 0 & , \text{otherwise} \end{cases} \tag{3.9}$$

3.2.3 Geometry of the docking problem

Two of the control objectives of the docking problem consisted of reaching the desired berth and avoiding obstacles. The following states were proposed to reflect these objectives:

- Section 3.2.3.1: Area of the vessel inside a berth rectangle.
- Section 3.2.3.2: Distance to the closest obstacle.

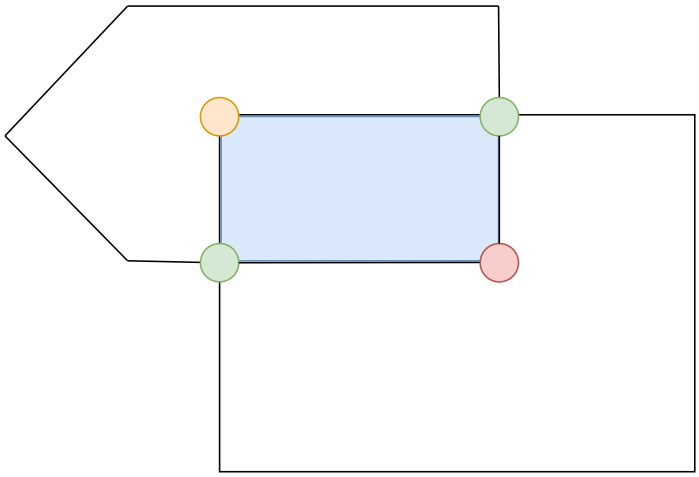


Figure 3.6: Illustration of corners of the overlapping area between berth rectangle set S_{be} and vessel set S_b.The blue area represents the overlapping areas of the two sets. The green dots represent intersections between the boundaries of the two sets, whereas the red and orange dots are vertexes of the two sets inside the overlapping area.

- Section 3.2.3.3: Difference between the vessel's pose and the berth's pose.

3.2.3.1 Area of the vessel inside the berth rectangle

The berth rectangle can be described as a convex set:

$$S_{be} \in \{\mathbf{A}_{be}\mathbf{p}^n - \mathbf{b}_{be} \leq 0\}, \tag{3.10}$$

where $\mathbf{A}_{be}$ and $\mathbf{b}_{be}$ are constants, and $\mathbf{p}^n$ is Cartesian position in NED-frame.

One way of calculating the overlapping area between a rectangle and a pentagon is by finding the corners of the overlapping area. The corners consist of intersections between the boundaries of berth rectangle S_{be} and vessel S_b, and the vertexes of sets S_{be} and S_b in the overlapping area. This is illustrated with an example in Figure 3.6.

The intersection between boundaries of the berth-rectangle set S_{be} and the vessel set S_b can be found by equation:

$$\mathbf{A}_{b_i}p + \mathbf{b}_{b_i} = \mathbf{A}_{be_j}p + b_{be_j}, \forall j \in \mathbf{I}_{be} \text{ and } \forall i\mathbf{I}_b \tag{3.11a}$$

$$\mathbf{A}_{be}p\mathbf{b}_{be} \leq 0, \tag{3.11b}$$

where $\mathbf{p}$ is Cartesian position in NED-frame, and $\mathbf{I}_{be}$ and $\mathbf{I}_b$ are the sets of inequalities of spatial constraints of the berth set S_{be} and the vessel set S_b. Equation 3.11a gives the intersections and the solution is only valid when inequality 3.11b is met.

The vertexes of the vessel set S_b and berth-rectangle set S_{be} in the overlapping area is found through the following inequalities:

$$\mathbf{A}_b p + \mathbf{b}_b \leq 0, \forall p \in Vertex(S_{be}), \tag{3.12a}$$

$$\mathbf{A}_{be} p + \mathbf{b}_{be} \leq 0, \forall p \in Vertex(S_b), \tag{3.12b}$$

where $\mathbf{p}^n$ are Cartesian coordinates in NED-frame, and all valid solutions of $\mathbf{p}$ represent vertexes of sets S_b and S_{be} which constitutes vertexes of the overlapping area.

By combining the solutions from Equation (3.11) and (3.12), all the vertexes of the overlapping areas can be found. The vertexes were further organised, before using the Shoelace formula to find the area a_v [121]. The pseudo-code of the Shoelace formula can be found in appendix B.

The fraction of the vessel area inside the berth f_a can be found using the following equation:

$$f_a = \frac{a_v}{a_b}, \tag{3.13}$$

where a_v is area of the vessel inside the berth rectangle and a_b is the total area of berth rectangle.

3.2.3.2 Distance to the closest obstacle

In this thesis, the only obstacle considered was land, including the quay. Traditional ways of informing a system about locations of the harbour are for instance:

- Indirectly by providing a map of the harbour and relating to the vessel's position within this area
- Directly by using ranging sensors such as LIDAR or RADAR.

There exist, for instance, RL solutions for obstacle avoidance for cars using LIDAR [122]. Simulations of LIDAR or RADAR was considered out of the scope for this thesis. In this thesis, the agent was given information about the position to the closest obstacle to simplify

the learning problem. This was inspired by the principle of "closest point of approach" (CPA), used in obstacle avoidance [123].

The closest obstacle is the obstacle nearest to any of the edges of the vessel. In this context, this can be simplified as the shortest distance between any point on the harbour boundary ∂S_s to one of the vertexes of the vessel set S_v, since both of the sets are convex. This is only true as long as the vessel is inside the docking area. An example of the closest obstacle from the harbour boundary ∂S_s to a vertex of the vessel is illustrated in Figure 3.7, marked in green. The figure also illustrates all the closest points from a specific vessel's vertex to all the edges of the harbour set S_s, marked in purple.

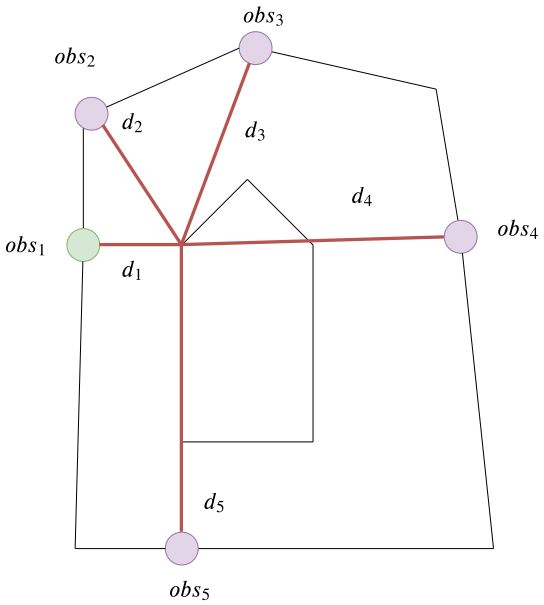


Figure 3.7: Illustration of closest obstacle from one vertex of the vessel to each edge of the harbour. The green dot represents the closest obstacle between the harbour boundary ∂S_s to vessel vertex, while the purple dots represents the potential closest obstacles from the other harbour edges.

The difference in position from the agent to the closest obstacle can be represented in several ways. In this thesis the difference was given in relative polar coordinates, using the angle relative to the heading of the vessel and distance to the closest obstacle from the edge, as illustrated in Figure 3.8. The reason for selecting this representation is discussed in Section 3.3.1.1.

The closest obstacle was found by first finding the shortest distance from all the vertexes of the vessel set S_v to the spatial constraints of the docking area set S_s, as illustrated for one vertex of the vessel in Figure 3.7. The equation for calculating the shortest distance is:

$$d_{b,obs} = \min d_{i,p}, \forall p \in \text{Vertex}(S_b), i \in I_s \tag{3.14a}$$

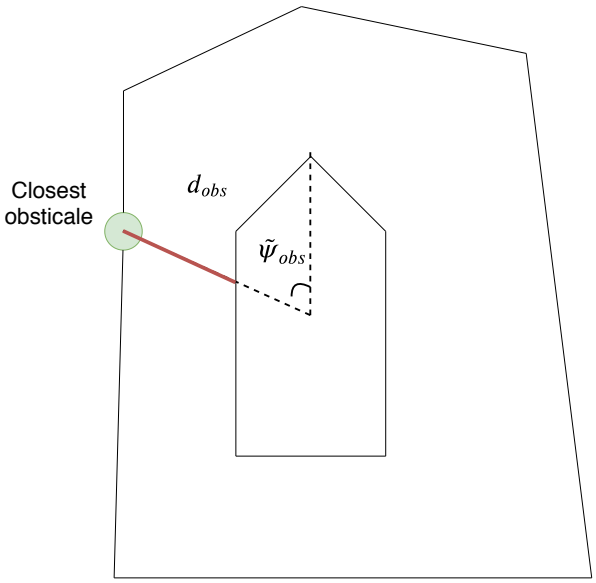


Figure 3.8: Illustration of the distance to an obstacle relative to the vessel edge in the direction of the obstacle and the angle to the obstacle relative to the vessels heading.

$$d_i = \begin{cases} \frac{|\mathbf{a}_i \mathbf{p} + bi|}{|\mathbf{a}_i|} & , A_s p + b \leq 0 \\ 0 & , \text{otherwise}, \end{cases} \tag{3.14b}$$

where $\mathbf{I}_s$ is the set of inequalities of the spatial constraints of the docking area set S_s. This means that $\mathbf{a}_i, b_i$ are constants of one of the inequalities in S_s. This equation gives the minimum distance from a point to a line [124]. These calculations were used to find the harbour constraint of S_s which were closest to vessel, with constants a_c and b_c, and closest vessel vertex to the harbour constraints p_c.

The coordinates of the closest obstacle were found by using the vessel vertex p_c and constraint with constants a_c and b_c. The coordinates of the closest obstacle (x^n_{obs}, y^n_{obs}) are:

$$x^n_{obs} = \frac{a_2(a_2 x - a_1 y) - a_1 b}{a_1^2 + a_2^2}, y^n_{obs} = \frac{a_1(-a_2 x + a_1 y) - a_2 b}{a_1^2 + a_2^2}, \tag{3.15}$$

where $a_c = [a_1, a_2]$ and $b_c = b$ and $p_c = [x_1, x_2]$.

The distance and relative angle from obstacle position (x^n_{obs}, y^n_{obs}) to origin of the vessel (x^n, y^n) can be calculated by:

$$d_{o,obs} = \sqrt{(x^n_{obs} - x^n)^2 + (y^n_{obs} - y^n)^2}, \tag{3.16a}$$

$$\tilde{\psi}_{obs} = \psi - \arctan2((y^n_{obs} - y^n), (x^n_{obs} - x^n)), \tag{3.16b}$$

where $d_{o,obs}$ is the distance from the origin of the vessel to the closest obstacle and $\tilde{\psi}_{obs}$ is the relative angle. This will be called vector $\mathbf{p}_{o,obs}$

The desired objective was to find the distance relative to the vessel edge in the direction of vector $\mathbf{p}_{o,obs}$. To calculate this a line was defined as going through the obstacle (x^n_{obs}, y^n_{obs}) and the vessel's origin x^n, y^n. This line was called $A_{\mathbf{obs}}x + b_{\mathbf{obs}} = 0$, which also goes through vector $\mathbf{p}_{o,obs}$, as illustrated in Figure 3.9. The distance from the vessel's origin to its edge in direction vector $\mathbf{p}_{o,obs}$, can be calculated by finding the closest valid intersection between the line from origin of the vessel to obstacle $A_{\mathbf{obs}}x + b_{\mathbf{obs}} = 0$ and boundaries of vessel set S_v. The valid intersections $p_{\mathbf{int}}$ between vessel boundaries ∂S_v and the line $\mathbf{A}_{obs}x + \mathbf{b}_{obs} = 0$ are illustrated in Figure 3.9. The intersections $p_{\mathbf{int}}$ were found using the same method as for calculations of area in Equation (3.11). The shortest distance from origin to the intersections is given by the equation:

$$d_{obs} = \min d_p, \forall p \in p_{int} \tag{3.17a}$$

$$d_p = \begin{cases} |p - p_{obs}| & , A_v p^b + b_v = 0 \\ \infty & , \text{otherwise} \end{cases} \tag{3.17b}$$

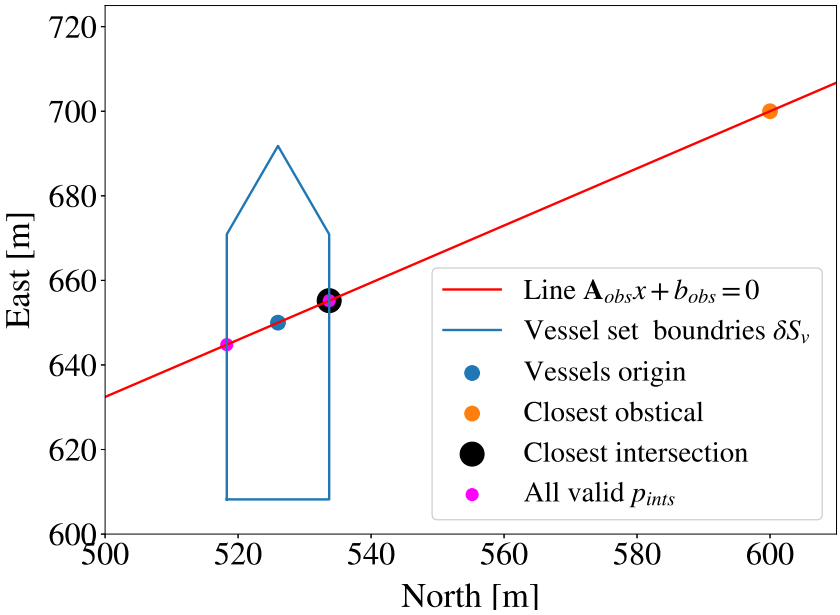


Figure 3.9: Illustrating how to find the relative distance d_{obs}. $A_{\mathbf{obs}}x + b_{\mathbf{obs}} = 0$ is the line from obstacle (x^n_{obs}, y^n_{obs}) to vessel origin x^n, y^n and $p_{\mathbf{int}}$ are the intersections between the line $A_{\mathbf{obs}}x + b_{\mathbf{obs}} = 0$ and vessel boundary set ∂S_s.

3.2.3.3 Difference in pose of the vessel and the desired berth

The distance from the vessel's origin to the berth's origin can be calculated in several ways. In this thesis, the distance was calculated as the distance from the origin of the berth (x_b, y_b, ψ_b) to the origin of the vessel (x, y, ψ). The distance was calculated as the Cartesian distance in body frame, leading to:

$$\tilde{\mathbf{p}}^b = \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \mathbf{R}^T(\psi)(\mathbf{p}_b^n - \mathbf{p}^n), \tag{3.18}$$

where $\mathbf{R}(\psi)$ is the rotation matrix from Equation (2.3), $\mathbf{p}_b$ is the position of the berth and $\mathbf{p}$ is the position of the berth.

The difference in the heading is defined as:

$$\tilde{\psi} = \psi - \psi_b \tag{3.19}$$

3.3 Docking using deep reinforcement learning

The objectives of docking were considered to be:

- **Avoiding obstacles**: Making sure distance from vessel edge to obstacles is higher than zero.
- **Target tracking**: Reaching the berth pose (x_b, y_b, ψ_b) from just outside the harbour.
- **Obeying speed regulations**: Assuring the vessel obeys the speed limit of 2.5 m/s inside the harbour.

The DRL agent performing docking needed to learn how to optimise all these objectives simultaneously, with no preinstalled controllers or thruster allocation to rely on. This is a complex and challenging task, and it was, therefore, decided to divide the docking problem into several phases. The suggested progressive methodology consisted of five learning phases, where the final learning phase was an end-to-end docking solution. The later learning phases were most often based on solutions from the earlier learning phases. The relation between the learning phases is presented in Figure 3.10. By creating several learning phases, the individual phases could be confirmed and optimised separately, before continuing to the next

learning phase. The order of the learning phases were inside out, first focusing on solutions close to the berth before ending just outside the perimeter of the harbour.

Learning phases one and two were used to solve the berthing phase of docking. The first learning phase consisted of creating a DRL agent capable of controlling the vessel to the desired parking pose in the middle of the harbour from relatively close range. Also as a part of learning phase one, the agent needed to keep the vessel in the desired parking pose, performing dynamic positioning (DP). Learning phase one is similar to berthing except for the lack of immediate proximity of the quay, and was therefore considered a precursor of learning phase two. Learning phase one was performed to see if the agent was capable of reaching and holding a certain pose, without the need to deal with the requirements involved in avoiding physical contact with the quay. Based on the results of learning phase one, actual berthing close to the quay was solved in learning phase two, taking into account the collision risk.

Learning phase three solved the task of controlling the vessel from right outside the harbour to a point close to the berth, learning the target tracking part of the approach phase of docking. Learning phases two and three were then combined to solve learning phase four. Learning phase four consisted of creating one agent controlling the vessel from right outside the harbour to the berth.

Learning phase five is the last phase, where the agent controls a vessel which obeys the speed limits, in addition to controlling it from right outside the harbour to the berth. This meant that the vessel had higher forward velocities outside the harbour and was supposed to reduce it before coming into the harbour. Learning phase five was only based on learning phase four, where the aspect of speed regulations was added directly. The simplicity of this objective made it possible to add this directly without solving this in a separate learning phase.

The agent trained in learning phase X is called "LPX: Y agent", where Y is a short abbreviation of the control task. The desired outcome of each learning phase were:

1. **LP1:Dynamic positioning**: An agent controlling the vessel to a desired pose from close range, and tries to stay in this pose, performing dynamic positioning (DP). The desired pose is in the middle of the harbour. It was tested with two different tactics: one based on the area of a vessel inside a target-rectangle, and one based on the difference between the vessel pose $\boldsymbol{\eta}$, and the desired target pose $\boldsymbol{\eta}_d$. The solution considering the area of the vessel inside a rectangle is called LP1-a:DP, while the other solution is called LP1-η:DP.

2. **LP2:Berthing**: An agent performing berthing, controlling the vessel from the proxim-

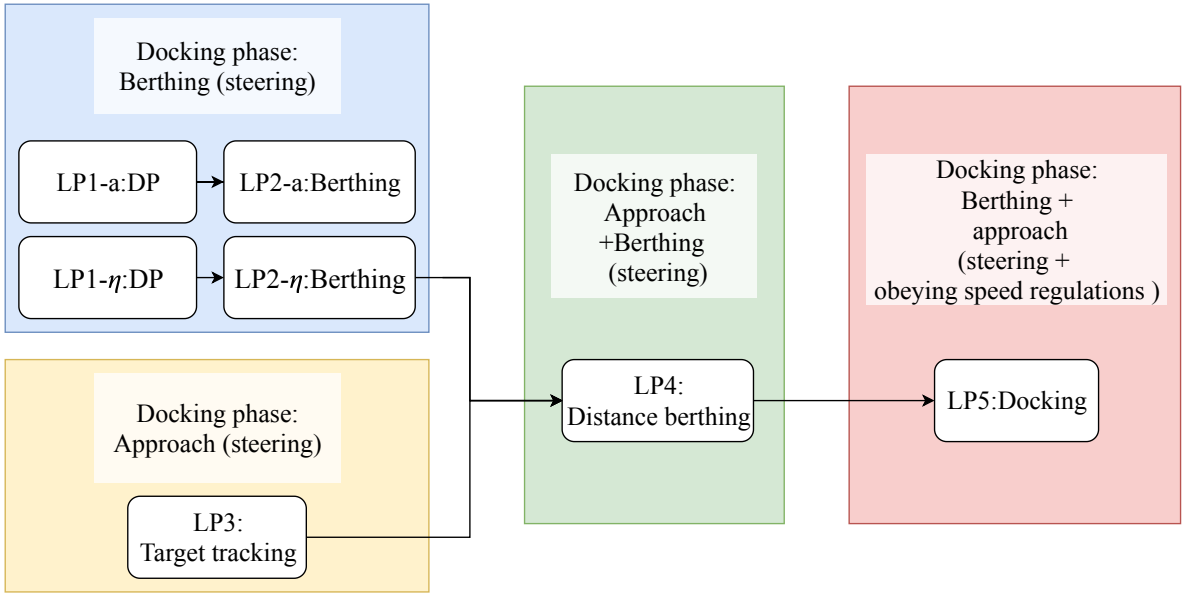


Figure 3.10: The relation between docking phases and learning phases (LP).

ity of the berth to the berth and keeping it there. This learning phase directly solved the berthing phase of docking. Two different approaches were also tested here, considering the area of the vessel inside a berth rectangle, called LP2-a:Berthing, and the difference between vessels pose and berth pose, called LP2-η:Berthing.

3. **LP3:Target tracking**: An agent controlling the vessel from outside of the harbour to a point in the vicinity of berth and staying there, performing dynamic positioning. This learning phase directly solved the approach phase of docking, with the addition of keeping the position once there.

4. **LP4:Distance berthing**: An agent performing berthing from larger distances, and keeping the vessel there. This is equivalent to performing the berthing and approach-phases of docking, without defining a hard limit between them. This learning phase is quite similar to LP2-η:Berthing, but in this learning phase, the berthing was performed from larger distances.

5. **LP5:Docking** : An agent performing docking, meaning an agent following the speed regulations and reaching the berth from just outside the harbour. This is equivalent to solving all the phases of the docking definition.

Well defined Markov decision processes (MDPs) were necessary to achieve agents solving these multi-objective problems. This entails designing action-vector, state-vector, reward functions and training scenarios. Experience often shows that designing a good MDP is more important than tuning parameters of the DRL algorithms. The following sections will present and discuss the design of the MDPs for each learning phase.

3.3.1 State vector

3.3.1.1 Design considerations

The DRL agent used observed states and rewards to optimise an objective, in this case, docking objectives. To achieve this, the agent needed to be provided with sufficient information to understand essential aspects of docking. In in the scope of this thesis, this entailed:

- The forward velocity of the vessel needed to be kept below a certain speed limit in the harbour, but sufficiently high to maintain manoeuvrability. The safe speed was decided by the vessels size, construction, and water conditions such as sea current.
- The agent needed to use information about the berth and obstacles, to select an action which creates a safe and efficient voyage.

To achieve this, the agent needed to understand both the dynamics of the vessel and the vessels pose in the harbour. This included distances to potential static collision objects at the waterfront and differences between vessel's and berth's pose.

The basic information of pose $\boldsymbol{\eta}$ and velocities $\boldsymbol{\nu}$ of the vessel was sufficient to understand the dynamics of the vessel. A state vector using only these states will depend on the rotation and translation of the vessel. The neural networks would, therefore, need to perform several mathematical transformations, before creating states which informs more directly about the docking objectives in a general manner. The non-linear properties of the neural network could, in principle, be able to learn transformations to handle these difficulties. However, practice shows that this typically means longer and less stable training to achieve the desired objective. It can also require a larger neural network to achieve the necessary transformations. To circumvent these issues, several new states relating the vessels pose to the docking objectives were created, avoiding these challenges as explained below, and given directly to the agent.

The five states created specifically to reflect the objectives of the docking task were:

- Representations of vessels pose related to obstacles:
  - 1) $(d_{obs}, \tilde{\psi}_{obs})$: The distance to the closest obstacle in body frame.
  - 2) l: A binary variable informing of any contact (crash) with obstacle.
  - 3) h: A binary variable informing when the vessel is inside the harbour and hence affected by the speed limit of 2.5 m/s.
- Representations of vessels pose related to berth:

- 4) $(\tilde{x}, \tilde{y}, \tilde{\psi})$: Cartesian and angular distance to berth in body frame.
- 5) f_a: The fraction of area of vessel inside the berth rectangle.

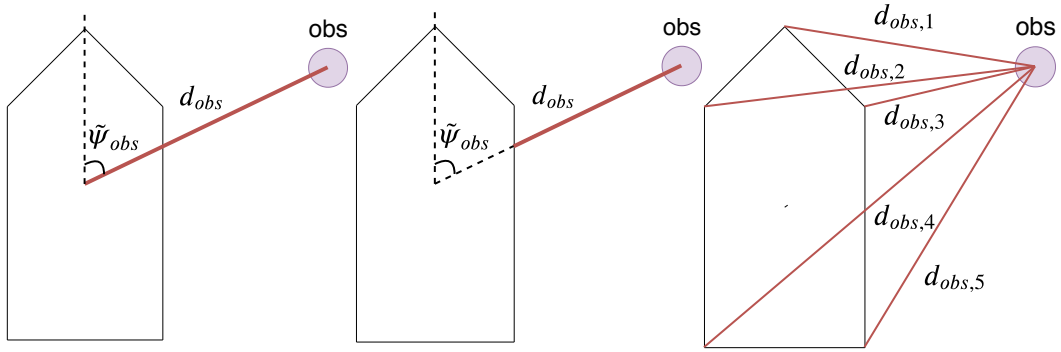
These five states are discussed in the following paragraphs, first state 1-3 and thereafter state 4-5.

Representation of vessel's pose related to closest obstacle

The DRL agent had no prior knowledge of the vessels shape. This means the agent would have to learn how to act in order to avoid crashing the pentagon-shaped vessel into the dock. It was desirable that the agent learned a general understanding of the harbour scenario, generalising the policy such that it could be used in other harbours. An alternative was that the agent created an explicit internal representation of the harbour, by for instance learning the position of all obstacle by trial-and-error, needing to retrain the solution for new harbours.

Several alternatives to achieve information about the position of the closest static obstacle, such as land, independent of the harbour layout were considered:

1. Alternative 1: Angle relative to heading and distance from vessels origin to closest obstacle, shown in Figure 3.11a.
2. Alternative 2: Angle relative to heading and the corresponding distance from edge of vessel to closest obstacle, illustrated in Figure 3.11b.
3. Alternative 3: Distances to closest obstacle measured from all the vertexes, visualised in Figure 3.11c .



(a) Alternative 1 (b) Alternative 2 (c) Alternative 3

Figure 3.11: Alternative ways of gaining insight about the closest obstacle, such as land

Alternative 2 was selected since initial testing showed better performance of the DRL agent using this alternative. This alternative represents the closest obstacle by the distance d_{obs},

Equation (3.17), and relative angle $\tilde{\psi}_{obs}$, Equation (3.16b). The reasons for this behaviour might be that:

- Alternative 2 gives a distance equal to zero when colliding with an obstacle, while alternative 1 will not indicate zero when colliding.
- Alternative 2 consists of fewer variables compared to alternative 3, which appears to require less learning to assess collision risk.

The binary land state l signalled if the vessel had crashed, see Equation (3.5). This state was employed to help the agent to learn the difference between land and sea more quickly.

The binary harbour state h was used, to inform if the vessel was in the harbour, see Equation (3.9). This variable was activated if the vessel was in a speed regulation area of 2.5 m/s, and was therefore important to let the agent know when to avoid speeding.

Representation of vessels pose related to berth

Two representations were tested to help the agent learn how to reach the berth without colliding with obstacles, but without learning an internal representation of the geographical properties of the harbour. These two are:

- $\boldsymbol{\eta}$-representation: This representation gives the difference between the vessels pose and desired pose. The states $(\tilde{x}, \tilde{y}, \tilde{\psi})$, Equation (3.18) and (3.19), gives the difference between the pose of vessels origins and the pose of berth origins, in body frame. Transforming the states from north-east-down (NED) frame to body-frame achieves a distance invariant to the position of vessel and berth.
- **a**-representation: This representation was used when considering parking a vessel inside a berth rectangle. The agent in this scenario needs to know how much of the vessel's shape was inside the berth rectangle. The state f_a gave the fraction of the vessels area inside a rectangle, equations (3.13). This state is invariant to the position of the berth and vessel.

3.3.1.2 State vectors of the learning phase

One could theoretically use all the states in all the learning phases. Experience, from this thesis, shows that neural networks often learn faster when only given the states needed for a given learning situation. The transformed states above were used in different combinations to satisfy the needs of the respective learning phases. The need depends on the objectives of learning phases.

The linear and rotational velocities (u, v, r) were always part of the state vector, giving a foundation for the agent's understanding of the dynamics of the vessel. These velocities are invariant to the position of the vehicle. Knowledge of the velocities is also important in a harbour area due to speed limits, and to of maintain a safe speed and reasonable manoeuvrability.

Table 3.1 summarises all the states used to create state vectors in this thesis. These states are combined into three different state vectors described in the next sub-sections.

Table 3.1: Summary of the state variables

State	Description
$\tilde{x}, \tilde{y}$	The Cartesian distances from origin of the vessel to the target position, in body frame, described by Equation (3.18).
$\tilde{\psi}$	The relative difference between heading of vessel and heading of the desired target, Equation (3.19)
l	A binary variable describing whether the vessel is in contact with land, Equation (3.6).
h	A binary variable describing whether the vessel is within the harbour, Equation (3.9).
u, v, r:	Linear and rotational velocities of the vessel, in body frame.
$d_{obs}, \tilde{\psi}_{obs}$	The distance from the vessel's edge to the closest obstacle and the relative heading from the vessel to the closest obstacle, Equation (3.17) and (3.16b).
f_a	Fraction of vessel area inside a berth rectangle, Equation (3.13)
$\dot{f}_a$	Derivative of the fraction of vessel inside a berth rectangle

State vector 1

The learning phases LP1-η:DP, LP2-η:Berthing,LP3:Target tracking and LP4:Distance berthing, all shared the same state vector due to similarities of the objectives. They all tried to reach a target pose and avoid obstacles, leading to the objectives:

- Avoid obstacles, meaning $d_{obs} > 0$
- Steer to the desired pose, meaning $\eta \rightarrow \eta_d$

The pose of the target (at sea, close to the berth, berth) and the distance from where the scenario starts, distinguishes the learning phases. The target pose for LP1-η was in the middle of the harbour, for LP3:Target tracking the target was close to the berth and for LP2:berthing and LP4:Distance berthing the target pose was the berth. For LP3:Target tracking $\tilde{\psi} = \psi$, due

to $\psi_d = 0$ since the target was only a position, and not a pose.

The state vector consisted of the difference between vessel pose and target pose $(\tilde{x}, \tilde{y}, \tilde{\psi})$, the body velocities (u, v, r), distance to closest obstacle $(d_{obs}, \tilde{\psi}_{obs})$ and the binary l indicating contact with obstacle. This gives the following state vector:

$$\mathbf{x} = \left[\tilde{x}, \tilde{y}, \tilde{\psi}, u, v, r, l, d_{obs}, \psi_{obs}\right], \tag{3.20}$$

State vector 2

The two learning phases LP1-a:DP and LP2-a:Berthing also share the same state vector, due to their common objectives:

- Avoid obstacles, meaning $d_{obs} > 0$
- Park the vessel inside a rectangle, meaning $f_a \rightarrow 0$

The pose of the target and the distance from where the scenario starts distinguishes the learning phases. The target pose is in the middle of the harbour for LP1-a:DP and at the berth for LP2-a:Berthing.

This state vector is the largest state vector in this thesis. It differs from state vector 1 by adding the states informing of the area f_a and its derivative $\dot{f}_a$.

This gives the following state vector:

$$\mathbf{x} = \left[\tilde{x}, \tilde{y}, \tilde{\psi}, u, v, r, l, d_{obs}, \tilde{\psi}_{obs}, f_a, \dot{f}_a\right], \tag{3.21}$$

The state vector needed to include both the distance and angle to the target, in addition to the fraction of vessel area in the target-rectangle. Through f_a, the agent received information on how much the vessel and target-rectangle were intersecting. By adding the distances to the target $(\tilde{x}, \tilde{y})$, the agent always had spatial information of the berth making the design more robust for both not quite so near phases of DP and berthing. The derivative of f_a was provided to let the agent better understand the relation between the vessel's dynamics and the berth. $\dot{f}_a$ was found using finite-difference.

State vector 3

Learning phase LP5:Docking was the only learning phase using state vector 3. This state vector is similar to the state vector 1, due to the similarity of objectives. The difference is that LP5:Docking also needed to know when speed regulations was active, leading to the

objectives:

- Avoid obstacles, meaning $d_{obs} > 0$
- Reach desire pose, meaning $\eta \rightarrow \eta_d$
- Obey speed limits, meaning $u \leq 2.5$ if $h = 1$

Thus, the state vector also included the binary state variable h to inform the agent if the vessel was in a limited speed area. This lead to an extension of state vector 1, yielding the following state vector:

$$\mathbf{x} = \left[\tilde{x}, \tilde{y}, \tilde{\psi}, u, v, r, l, d_{obs}, \tilde{\psi}_{obs}, s\right], \tag{3.22}$$

3.3.2 Action vector

The model of the container ship used three actuators, exerting forces $\mathbf{f} = [f_1, f_2, f_3]^\top$ and angles $\boldsymbol{\alpha} = [\alpha_1, \alpha_2]^\top$. The vessel had azimuth thrusters at the stern, controlled by $[f_1, f_2, \alpha_1, \alpha_2]^\top$, and tunnel thruster at the bow, controlled by f_3.

The action space should reflect the need of manoeuvrability in the harbour, meaning that the vessel would have to be capable of moving backwards, sideways and forwards. Two action vectors were defined, one for LP3:Target tracking and another for LP1:DP, LP2:Berthing, LP4:Distance berthing and LP5:Docking. The reason for a separate action vector for LP3:Target tracking was that during transit from the start of the harbour to proximity of berth, the tunnel thrusters are not traditionally used.

The following action vectors and spaces were proposed:

- LP3:Target tracking: $[\alpha_1, \alpha_2] \in [-90, 90]$ degrees and $[f_1, f_2] \in [-70, 100]$ kN.
- Other learning phases: $[\alpha_1, \alpha_2] \in [-90, 90]$ degrees, $[f_1, f_2] \in [-70, 100]$ kN and $[f_3] \in [-50, 50]$ kN.

3.3.3 Reward function

The reward function gives feedback on the performance of the agent and is essential in reinforcement learning. The reward function formalises the goal of the agent and needs to be designed to reflect the objectives of docking.

Both PPO and DDPG solve a problem using random exploration. This means they are less likely to successfully solve a problem with a sparse reward function. Only rewarding the agent when solving the ultimate objective of docking is therefore not sufficient. The reward-function was therefore shaped, giving supplemental rewards to make the problem easier to learn, as discussed in Section 2.3.3.5. In Martinsen et al. [4] an agent is given rewards when the vessel is getting closer to the desired path and thus producing a less sparse reward function. The proposed reward function of Martinsen et al. was a Gaussian function of the distance to a guidance line.

There are many ways to design reward functions. Several reward functions can potentially achieve similar results. Most important is giving the DRL agent sufficient feedback, so that desired behaviour can be achieved. If the reward function is unfortunately designed, the agent might opt for local optima. This can result in a policy failing to provide the desired behaviour.

Several of the learning phases shared the same state and action vector. Though several of the learning phases operate with the same state and action vectors they do not share the same reward functions. The reasons for structuring and shaping the reward functions in a particular way is explained in the following sections, and the similarities and differences between the reward functions are pointed out

3.3.3.1 LP1-η:DP

The objectives of this learning phase were:

- Reach desired pose from short distances and hold this, meaning $\eta \rightarrow \eta_d$ or $\tilde{\eta} \rightarrow 0$.
- Avoid obstacles, meaning $d_{obs} > 0$.

The reward function needed to reflect that all the objectives must be solved simultaneously. It was for instance not sufficient to assure that the vessel converged to the desired position, while ignoring the heading, or crashing into an obstacle to achieve the perfect pose. A reward function was proposed, reflecting a hierarchy of all the control objectives in this learning phase. It was designed to guide the agent to opt for solving all the objectives simultaneously and sufficiently.

The reward function is a sum of several reward components, where the priority is reflected by the weight of the reward components and how they depend on each other. The reward components are Gaussian functions inspired by Martinsen et al. [4]. The proposed reward

function is:

$$r(d_d, \tilde{\psi}, l, d_{obs}) = r_{d_d} + r_{\tilde{\psi}} + r_{obs}, \tag{3.23a}$$

$$r_{d_d} = \begin{cases} C_{d_d} e^{\frac{-(d_d^2)^2}{2\sigma_{d_d}^2}} & , l = 0 \text{ and } |\tilde{\psi}| < \pi/2 \\ 0 & , \text{otherwise} \end{cases}, \tag{3.23b}$$

$$r_{\tilde{\psi}} = \begin{cases} C_{\tilde{\psi}} e^{\frac{-\tilde{\psi}^2}{2\sigma_{\tilde{\psi}}^2}} & , l = 0 \text{ and } r_{d_d} \leq C_{dock}/2 \\ 0 & , \text{otherwise} \end{cases}, \tag{3.23c}$$

$$r_{obs} = \begin{cases} C_{obs,T} & , \text{otherwise} \\ C_{obs} e^{\frac{-d_{obs}^2}{2\sigma_{d_{obs}}^2}} & , l = 0, \end{cases}, \tag{3.23d}$$

where $d_d = \sqrt{\tilde{x}^2 + \tilde{y}^2}$ is the distance from the origin of the vessel to the desired target, $C_{obs,T} < 0$, $C_{obs} < 0$, $C_{\tilde{\psi}} > 0$, $C_{d_d} > 0$, $C_{d_d} > 0$ are weights, and $\sigma_{\tilde{\psi}} > 0$, $\sigma_{d_d} > 0$, $\sigma_{d_{obs}} > 0$ are standard deviations. All the standard deviations and weights are constants.

The top priority in docking is to avoid obstacles. The reward component r_{obs} gives reward based on the distance to the closest obstacle d_{obs}. A massive penalty is given to the agent if the vessel crashes, which means that $l = 1$ and $d_{obs} = 0$. The agent receives a smaller penalty if the vessel comes too close to an obstacle. The reward component r_{obs} is illustrated in Figure 3.12 for $l = 0$ and $d_{obs} > 0$. The other reward components, $r_{\tilde{\psi}}$ and r_{d_d}, are set to zero if the agent crashes. This was to ensure that the agent would always be given minus points when crashing, making it clear that crashing was not a good thing.

The next most important objective was reaching the target position, reflected by reward component r_{d_d}. The reward component gives points as the vessel is closing in on the desired position. It will however not award points if the vessel is moving towards the target pose with a heading error larger than 90 degrees, $|\tilde{\psi}| > \pi/2$.This shaping was added to help the agent to avoid taking unnecessary turns and assuring that the agent does not optimise getting to the berth position and disregarding the desired heading. The reward component r_{d_d} is illustrated in Figure 3.13.

The reward component $r_{\tilde{\psi}}$ assigns credit to the agent when the vessel is close to the target position with a desirable heading. This reward would only be given when the vessel was close to the berth, thus depending on reward component r_{d_d}. This meant that if the agent achieved the correct angle far away from the berth, it would not be rewarded.

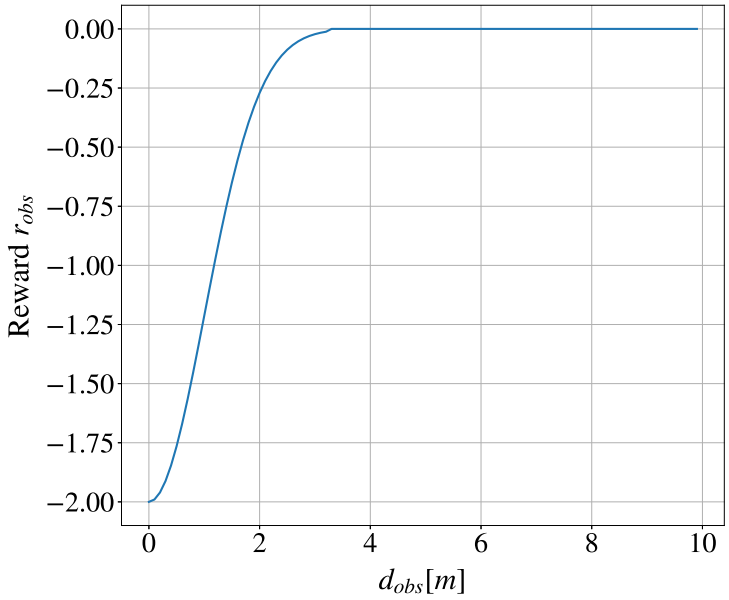


Figure 3.12: Illustration of reward component r_{obs}, when $l = 0$, with $\sigma_{obs} = 1$ m and $C_{obs} = -2$

The agent was not explicitly rewarded for reducing speed to zero when reaching the desired target, but would instead be given full credit when in the correct pose at the target. This was designed to stimulate the agent to try to stay at the berth for as long as possible.

When designing the reward function, it was found that the most challenging issue was to achieve convergence to the desired heading and position simultaneously. First, it was experimented with a reward function summarising Gaussian reward functions of the difference between vessel pose and target pose $\tilde{x}$, $\tilde{y}$ and $\tilde{\psi}$, as presented by Anderlini et al. [62]. This reward function did not, however, work in this scenario, since that agent was found to prefer to solve only one objective at a time with this reward function.

Another alternative tested was a multivariate Gaussian reward function of $\tilde{x}, \tilde{y}, \tilde{\psi}$, to guide the agent to solve all the objectives simultaneously. After some initial experimentation, it appeared to be a too sparse reward function, due to the agent apparently lacked understanding of the problem. Instead, the proposed reward function reflects a hierarchy of objectives. This was constructed to give a balance between not giving rewards to quickly, and not too few. The proposed reward function ensures that the agent can't achieve full reward without solving all the objectives simultaneously.

3.3.3.2 LP1-a:DP

The objectives in this learning phase were:

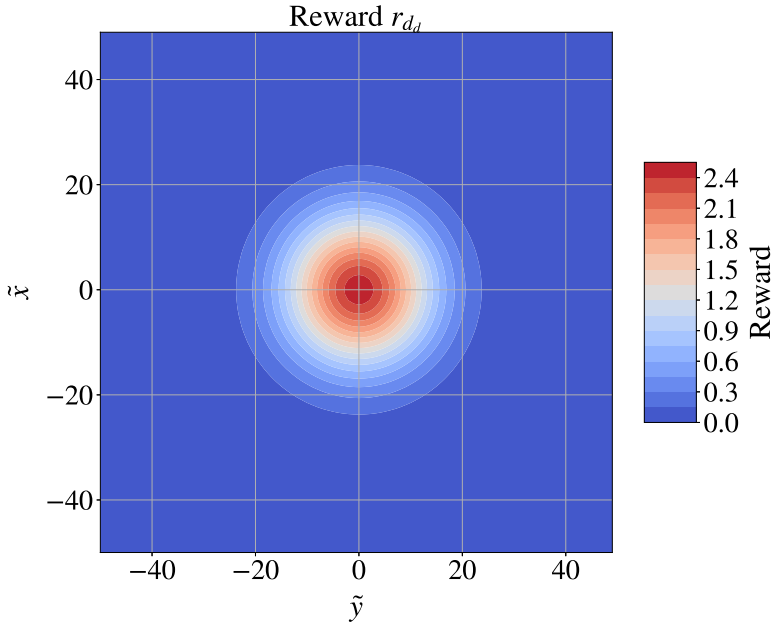


Figure 3.13: Reward component r_{d_d}, when $l = 0$ and $|\tilde{\psi}| < \pi/2$, with $\sigma_{d_d} = 10$ and $C_{d_d} = 2.5$

- Park the vessel inside a rectangle and stay there, which means $f_a \rightarrow 1$.
- Avoid obstacles, which means $d_{obs} > 0$

The proposed reward function is the sum of the two reward components r_{f_a} and r_{obs}. The reward component r_{f_a} concerns parking the vessel inside a rectangle and r_{obs} concerns avoiding obstacles. The objective of LP1-a:DP and LP1-η:DP are in principle equivalent. The reward function therefore consists of the same reward component r_{obs}, while the reward components for reaching the desired pose $r_{\tilde{\psi}} + r_{d_d}$ is replaced with the reward component for parking inside a rectangle r_{f_a}. The proposed reward function for LP1-a:DP is:

$$r(f_a, \tilde{\psi}, l, d_{obs}) = r_{f_a} + r_{obs}, \tag{3.24a}$$

$$r_{f_a} = \begin{cases} C_{f_a} e^{\frac{-(1-f_a)^2}{2\sigma_a^2}} & , l = 0 \text{ and } |\tilde{\psi}| < \pi/2 \\ 0 & , \text{otherwise,} \end{cases} \tag{3.24b}$$

where $C_{f_a} > 0, \sigma_a > 0$ are constants.

The component r_{f_a} gives rewards if the vessel is partly inside the target-rectangle, and facing in the desired direction, as given by the condition of $|\tilde{\psi}| < \pi/2$. This reward component is a Gaussian function, and is illustrated in Figure 3.15 for $l = 0$ and $d_{obs} > 0$.

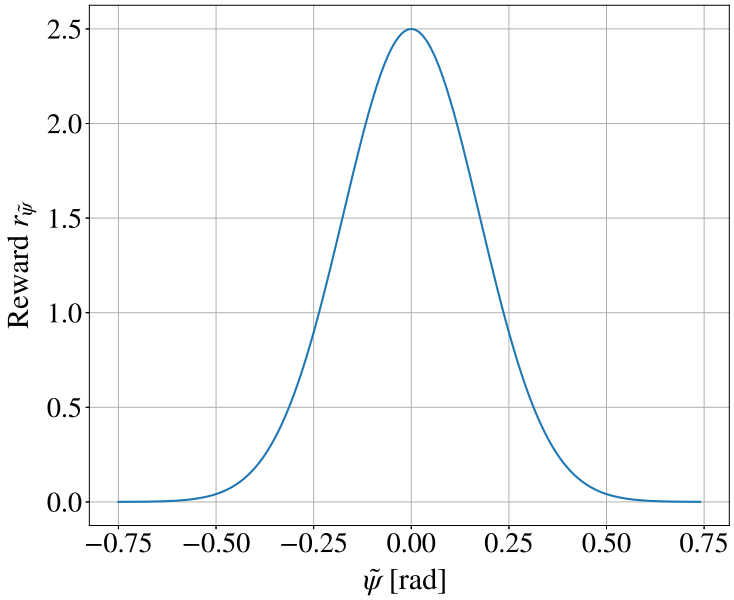


Figure 3.14: Reward component $r_{\tilde{\psi}}$, when $r_{d_d} > C_{d_d}/2$ and $l = 0$, with $\sigma_{\tilde{\psi}} = 10$ degrees and $C_{\tilde{\psi}} = 2.5$

In the same manner as for LP1-η:DP, the task of highest priority was avoiding obstacles, and then to reach the desired target-rectangle. The agent is only rewarded for being inside the berth rectangle when not in contact with land, as defined by the condition $l = 0$.

3.3.3.3 LP2-η:Berthing

Parking the vessel inside a target pose and avoiding obstacles was the objective in this learning phase, in the same manner as in LP1-η:DP. The most significant difference is that the target is a berth, which is close to obstacles, in LP2-η:Berthing. Using the reward function of LP1-η:DP for LP2-η:Berthing was tested, but this did not work. Experimentation showed the agent was hesitant to move towards the berth, most likely due to the massive penalties received if the vessel crashed into the quay. The reward function was shaped to "lure" the agent towards the berth. The resulting reward function is:

$$r(d_d, l, \tilde{\psi}, d_{obs}, \dot{d}_d) = r_{d_d} + r_{\tilde{\psi}} + r_{obs} + r_{d_{dot}}, \tag{3.25a}$$

$$r_{d_{dot}} = \begin{cases} C_{d_{dot}} \dot{d}_d & , otherwise \\ 0 & , \dot{d}_d > 0 \text{ and } |\tilde{\psi}| > \pi/2 \\ C_{d_{dot}} & , \dot{d}_d < -1, \end{cases} \tag{3.25b}$$

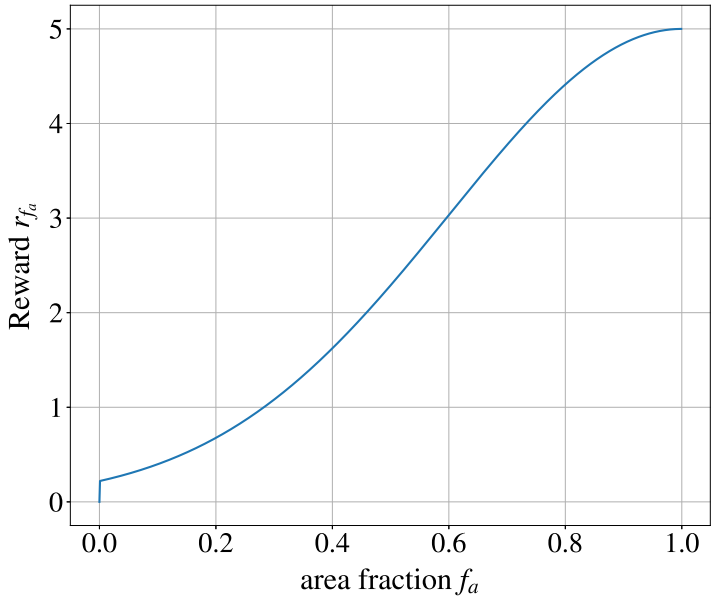


Figure 3.15: Illustration of reward component r_{f_a}, when $l = 0$, with $\sigma_{f_a} = 0.4$ m and $C_{f_a} = 5$

where $C_{d_{dot}} > 0$ is a constant and $\dot{d}$ is the derivative of the distance from berth to the vessel.

The reward components r_{d_d},$r_{\tilde{\psi}}$ and r_{obs} are the same as in LP1-η:DP. A new reward component $r_{d_{dot}}$ was added to reward the vessel for moving towards the berth, which means $\dot{d}_d < 0$. The reward component $r_{d_{dot}}$ is a function increasing linearly with $\dot{d}_d$ until a given saturation limit, as illustrated in Figure 3.16. The reason for making this a saturated reward function was to limit the maximum receivable reward, to avoid encouraging high velocities inside the harbour.

The reward component $r_{d_{dot}}$ is considered of lower priority, than r_{f_a} and r_{obs}. It should therefore be a relatively low value of the weight $C_{\dot{d}_d}$.

An alternative way to make the reward function less sparse, without adding the reward component $r_{d_{dot}}$, was to increase the values of σ_{d_d}. This implies increasing the area around the berth where the component r_{d_d} gives rewards. When increasing the values of σ_{d_d}, the rewards given in positions closest to the berth would be almost the same as for reaching the berth. The agent would, therefore, not be significantly stimulated to reach the desired berth. It would either use extended time to get there, ramble around or stay somewhere nearby. This is an inherent property of a Gaussian reward function in this kind of application. The reward component $r_{d_{dot}}$ was instead added to the reward function.

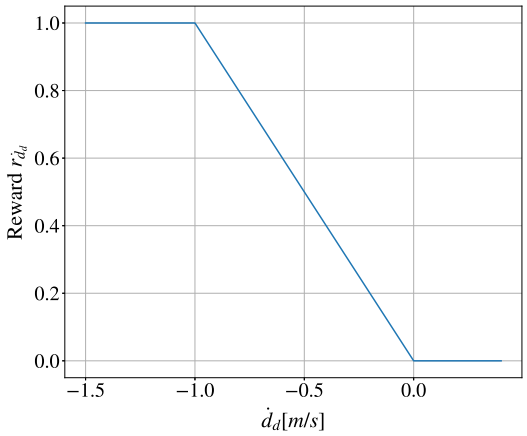


Figure 3.16: Reward component $r_{d_{dot}}$, with $C_{\dot{d}_d} = 1$.

3.3.3.4 LP2-a:Berthing

The objective here was to avoid obstacles and park the vessel inside a target-rectangle, the same objective as for LP1-a:DP. The biggest difference is that the berth is close to an obstacle, and only award points for entering the rectangle will not suffice for LP2-a:Berthing. The reward component $r_{d_{dot}}$ was added to "lure" the agent towards the berth, and the quay with its potential massive penalties, in the same fashion as for LP2-η:Berthing. The resulting reward function was:

$$r(f_a, \dot{d}_d, \tilde{\psi}, l, d_{obs}) = r_{f_a} + r_{obs} + r_{d_{dot}}, \tag{3.26a}$$

Here r_{f_a} and r_{obs} are the same reward components as for LP1-a:DP, giving rewards when the vessel is inside the berth rectangle and penalties if the vessel is too close to an obstacle.

3.3.3.5 LP3:Target tracking

The objectives of target tracking were:

- Reach desired target position and hold it, which means $\tilde{\mathbf{p}} \to 0$ or $\mathbf{p} \to \mathbf{p}_d$, where $\mathbf{p}$ is the position of the vessel and $\mathbf{p}_d$ is the target position.
- Avoid obstacles, which means $d_{obs} > 0$

These objectives are relatively similar to that of LP1-η: DP, with reaching a target pose and avoiding obstacles. The main difference is that LP3:Target tracking is performed at more

substantial distances, and the desired target is a position $\mathbf{p}_d$ instead of a pose $\boldsymbol{\eta}_d$.

It was initially proposed to use a similar reward function in LP3:Target tracking as in LP1-η:DP, consisting of reward components r_{obs}, r_{d_d} and $r_{\tilde{\psi}}$. This did, however, not work, most likely because the reward function for the problem became too sparse. A reward component for moving towards the target $r_{\dot{d}_d}$ was therefore added. This component decreases the sparsity of the problem and "lures" the agent towards the target. This reward component was also used in LP2:Berthing, to stimulate the agent to explore the berth, even though it was close to an obstacle.

The proposed reward function for LP3:Target tracking was therefore:

$$r(d_d, \tilde{\psi}, l, d_{obs}) = r_{d_d} + r_{obs} + r_{d_{dot}} \tag{3.27}$$

This function gives rewards when the vessel is close to the desired position through reward component r_{d_d}, and for moving towards the berth through reward component $r_{d_{dot}}$. In the same fashion as previous reward function, it also penalises the agent when the vessel gets to close to an obstacle through reward component r_{obs}.

3.3.3.6 LP4:Distance berthing

The learning phase LP4:Distance berthing involved creating an agent performing berthing from more substantial distances. This phase can be considered as a combination of LP3:Target tracking and LP2:Berthing. From previous learning phases, it was discovered that parking a vessel at a pose is preferred over parking a vessel inside a rectangle, as discussed in Section 4.4. Therefore the objective of LP3:Target tracking was the same as of LP1-η:DP and LP2-η:Berthing, giving the objectives:

- Reach desired target pose and hold it, which means $\tilde{\eta} \rightarrow 0$ or $\eta \rightarrow \eta_d$
- Avoid obstacles, which means $d_{obs} > 0$

In addition to the reward function needed to reflect these objectives, an additional reward component was added to trick the agent to move from more substantial distances towards the berth. These problems were solved in LP2:Berthing and LP3:Target tracking by adding the reward component $r_{d_{dot}}$, giving points when the vessel moved towards the berth. The reward function proposed for the learning phases was, therefore, the same as in LP2-$\boldsymbol{\eta}$:Berthing, giving:

$$r(\tilde{x}_d, \tilde{y}_d, l, d_{obs}) = r_{d_d} + r_{\tilde{\psi}} + r_{obs} + r_{d_{dot}}, \tag{3.28}$$

The total reward function consists of:

- Penalising the agent if the vessel is to close to an obstacle, through reward component r_{obs}
- Rewarding the agent if controlling the vessel towards the berth, through reward component $r_{d_{dot}}$
- Rewarding the agent if the vessel is close to the desired berth, through reward component r_{d_d} and $r_{\tilde{\psi}}$

3.3.3.7 LP5:Docking

The objectives of docking were:

- Reach desired target pose and hold it, which means $\tilde{\boldsymbol{\eta}} \rightarrow 0$ or $\boldsymbol{\eta} \rightarrow \boldsymbol{\eta}_d$.
- Avoid obstacles, which means $d_{obs} > 0$.
- Obey speed regulations in the harbour, which means $u \leq 2.5$ m/s.

These objectives are quite similar to LP4:Distance berthing, reaching a berth from long distance and avoiding obstacles. The difference is the additional objective of speed regulation inside the berth. This leads to the following reward function:

$$r(\tilde{x}_d, \tilde{y}_d, l, d_{obs}) = r_{d_d} + r_{\tilde{\psi}} + r_{obs} + r_{d_{dot}} + r_u, \tag{3.29a}$$

$$r_u = \begin{cases} C_u, s = 1 & u > 2.5 \\ 0 & , otherwise \end{cases}, \tag{3.29b}$$

where $C_u < 0$ is an constant.

The total reward function, has an additional speed reward component r_u, compared to the reward function for LP4:Distance berthing. The reward component r_u represents a constant penalty if the forward velocity is on or above 2.5 m/s in the harbour. A constant penalty of r_u was chosen, rather than a Gaussian or linear function, to simplify learning.

The reward components for avoiding obstacles and reaching the desired berth were given

higher weight than the speed reward component, to ensure higher priority. Even though the penalty of having too high speed was set low, as discussed later, the agent often selected lower velocities in the harbour, most likely due to lower risk of crashing and lower penalties thus received.

The reward function in this learning phase consists of five reward components, yielding one scalar reward signal. Prioritising the components in the compound reward was found challenging, and meeting all criteria simultaneously is a classical multi-objective optimisation challenge. Balancing the rewards so that the speed was kept below the speed limit, and at the same time avoiding crashing with the quay and reaching the berth, made the construction of the reward function of this learning phase a challenging task.

3.3.3.8 Reducing tear and wear of actuators

The DRL agent can find an optimal solution of a control problem, but might exhibit aggressive use of the actuators. In the real world, this is not desirable due to tear and wear. A DRL agent can also find a solution where the actuators are used more reasonably. The agent will however in itself see little difference between solutions with intense and less intense use of the actuators, as the behaviour of the vessel might appear to be the same from the agent's point of view. With a too high penalty on the use of actuators, the agent might be discouraged from exploring essential states. A small penalty of the use of the actuators can, however, help the agent towards less intensive actuator use, and still provide satisfactory results.

The proposed reward component to penalise frequent and extensive changes in the use of the actuators was:

$$r_{\dot{\alpha},\dot{f}} = \sum_{n=0}^{3} C_{\dot{f},i}|\dot{f}_n| + \sum_{n=0}^{2} C_{\dot{\alpha},i}|\dot{\alpha}_n|, \tag{3.30}$$

where $\mathbf{C}_{\dot{f}} < 0$ and $\mathbf{C}_{\dot{\alpha}} <$ are constants.

The reward component $r_{\dot{\alpha},\dot{f}}$ is a linear reward function with regard to the rate of change of the actuators $\dot{\mathbf{f}}$ and $\dot{\boldsymbol{\alpha}}$. The penalty is the weighted sum of the derivatives. The weight was selected so that the derivatives of the actuator forces and angles are equally penalised.

The state vector can also be extended to include the action vector $\mathbf{a}$ and its derivative $\dot{\mathbf{a}}$. This can help the agent to calculate similar actions at consecutive time steps.

3.3.4 Summary of reward functions, state vectors and action vectors

A summary of reward functions, state vectors and action vectors is presented in table 3.2. The parameters used in the reward function are summarised in Table 3.3 for the proximal policy optimization (PPO) agents and in Table 3.3 for the deep deterministic policy graident (DDPG) agents.

Table 3.2: A summary of reward function, action vector and state vector for the different learning phases

LP	**Reward**	**State vector**	**Action vector**
LP1-a:DP	$r_{f_a} + r_{obs}$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \tilde{\psi}, f_a, \dot{f}_a$	f_1, f_2, f_3, α_1, α_2
LP2-a:Berthing	$r_{f_a} + r_{obs} + r_{d_{dot}}$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \tilde{\psi}, f_a, \dot{f}_a$	f_1, f_2, f_3, α_1, α_2
LP1-η:DP	$r_{d_d} + r_{\tilde{\psi}} + r_{obs}$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \tilde{\psi}$	f_1, f_2, f_3, α_1, α_2
LP2-η:Berthing	$r_{d_d} + r_{\tilde{\psi}} + r_{obs} + r_{d_{dot}}$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \tilde{\psi}$	f_1, f_2, f_3, α_1, α_2
LP3:Target tracking	$r_{d_d} + r_{obs} + r_{d_{dot}}$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \psi$	f_1, f_2 α_1, α_2
LP4:Distance berthing	$r_{d_d} + r_{\tilde{\psi}} + r_{obs} + r_{d_{dot}}$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \tilde{\psi}$	f_1, f_2, f_3, α_1, α_2
LP5:Docking	$r_{d_d} + r_{\tilde{\psi}} + r_{obs} + r_{d_{dot}} + r_u$	$u, v, r, l, d_{obs}, \tilde{\psi}_{obs}$, $\tilde{x}, \tilde{y}, \tilde{\psi}, s$	f_1, f_2, f_3, α_1, α_2

Table 3.3: Parameters used in reward functions for PPO, for each learning phase.

	$C_{obs,T}$	C_{obs}	C_{f_a}	C_{d_d}	$C_{\tilde{\psi}}$	$C_{\dot{d}_d}$	C_u	σ_{obs}	σ_{f_a}	σ_{d_d}	$\sigma_{\tilde{\psi}}$
LP1-a:DP	-600	-2.5	5					1	0.4		
LP2-a:Berthing	-600	-2.5	5			1		1	0.4		
LP1-η:DP	-600	-2.5		2.5	2.5			1		10	0.17
LP2-η:Berthing	-600	-2.5		2.5	2.5	1		1		10	0.17
LP3:Target tracking	-600	-2.5		2.5		1		1		10	
LP4:Distance berthing	-600	-2.5		2.5	2.5	1		1		10	0.17
LP5:Docking	-600	-2.5		2.5	2.5	1	-5	1		10	0.17

Table 3.4: Parameters used in reward functions for DDPG, for each learning phase.

	$C_{obs,T}$	C_{obs}	C_{f_a}	C_{d_d}	$C_{\tilde{\psi}}$	$C_{\dot{d}_d}$	C_u	σ_{obs}	σ_{f_a}	σ_{d_d}	$\sigma_{\tilde{\psi}}$
LP1-a:DP	-600	-2.5	5					1	0.4		
LP2-a:Berthing	-600	-2.5	5			1		1	0.4		
LP1-η:DP	-600	-2.5		2.5	2.5			1		30	0.17
LP2-η:Berthing	-600	-2.5		2.5	2.5	1		1		30	0.17
LP3:Target tracking	-600	-2.5		2.5		1		1		30	

3.3.5 Training scenario

During training, the agent was subjected to various training scenarios. The different learning phases required different learning scenarios to give the agent the opportunity to explore the dynamics of the environment.

The training scenarios used for the different learning phases were:

3.3.5.1 Training scenario 1

The learning phases LP1:DP, LP2:Berthing, LP3:Target tracking and LP4:Distance berthing, were given almost identical training scenarios. They differed only by the scaling of the training area and placement of target. For instance, LP4:Distance berthing had the same target as LP2:Berthing, but LP4:Distance berthing performed berthing on larger distances.

The training episodes were initialised randomly from a set of valid states. The valid set of states for each learning problems are presented in Table 3.5. From this table, we can see that the vessel was given a random position within a predefined training area inside the docking area, according to properties of the learning phase. If a random pose placed the vessel on land, a new random pose was generated.

The agent was initialised with a heading in the range of $(-45, 45)$ degrees from the desired heading and forward velocities of $(-0.5, 0.5)$ m/s. A constraint was added to the initialisation to ensure that the vessel had an initial forward velocity towards the desired position, to avoid excessive learning.

The training scenario for LP4:Distance berthing was conducted with a varying desired heading ψ_d of ± 180 degrees. This was done to make sure that the agent learned how to approach the berth on the port, but also on the starboard side.

3.3.5.2 Training scenario 2

The training scenarios for LP5:Docking was the same as for LP4:Berthing with regard to (x, y, ψ, v, r). The difference is the range of possible forward velocities and the placement of the berth.

The initial forward velocities were randomly selected within the range of (2.0,3.5) m/s outside the harbour, so that the agent had to slow down to avoid penalties when reaching the harbour. Within the harbour, the velocities were initialised as in LP4:Distance berthing. Excluding illegally high initial velocities inside the harbour was not only necessary from a legal point

Table 3.5: Table of initialisation sets in learning phases 1 - 4

Variable	**LP1: DP**	**LP2: Berthing**	**LP3: Target A**	**LP4: D Berthing**
x_d	700	700	700	700
y_d	640	529.5	640	529.5
ψ_d	$-\pi/27$	$-\pi/27$	0	$-\pi/27$
x	$(x_d - 60, x_d + 60)$	$(x_d - 120, x_d + 120)$	$(x_d - 400, x_d + 400)$	$(x_d - 400, x_d + 400)$
y	$(y_d - 40, y_d + 40)$	$(y_d - 80, y_d + 80)$	$(y_d - 400, y_d + 400)$	$(y_d - 400, y_d + 400)$
ψ	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$
u	(-0.5,0.5)	(-0.5,0.5)	(-0.5,0.5)	(-0.5,0.5)
v	0	0	0	0
r	0	0	0	0

of view. It was also necessary because higher velocities made it harder for the agent to avoid obstacles in the restricted area, due to the dynamics of the vessel.

Even when enforcing the speed limit approaching the harbour area, the trained agent experienced some collisions with the docks when η_d was set to $(700, 529.5, -\pi/27)$. High velocities appeared to be challenging with the relatively short distances from the start of harbour to berth. . Increasing the distance to improve learning was investigated, but this limited the number of possible training scenarios with higher initial velocities, but did however show that the agent became more capable of making a controlled reduction of its speed.

3.3.6 Steady state error compensation

The nature of the Gaussian reward function can lead to steady-state error (SSE), as discussed in Martinsen et al [4]. In the same paper it was proposed to estimate the SSE error through integral action of the error, and then compensate for it by augmenting the state accordingly. The SSE e_{ss} can be estimated as:

$$e_{ss,t+1} \leftarrow e_{ss,t} + k_i h e_t, \tag{3.31}$$

where $e_{ss,0} = 0$ and k_i describes the integration rate. The factor k_i is a constant and needs to be tuned so that the estimator is slower than the system to avoid instability, meaning $k_i \in (0, 1]$. The steady-state compensation was not used during training, as e_{ss} is dependent on previous states (not only $e_{ss,t-1}$), making it a nonstatic Markov process.

To handle SSE, the state was augmented so that:

$$x = x + e_{ss}. \tag{3.32}$$

Anti-windup on the steady state compensation was also implemented to avoid overshoots. The anti-windup was implemented using saturation as follow:

$$e_{ss,t} = \begin{cases} e_{ss,\max} & , e_{ss,t} > e_{ss,\max} \\ e_{ss,\min} & , e_{ss,t} < e_{ss,\min} \\ e_{ss,t} & , \text{otherwise} \end{cases}. \tag{3.33}$$

In this thesis the general SSE was relatively low, except when the LP4:Distant Berthing agent was subjected to a constant current. Steady state error augmentation was therefore performed on $\tilde{x}$ and $\tilde{y}$ in the state vector, and parameters used are presented in Table 3.6.

Table 3.6: Parameters used for state augmentation.

State	k_i	**Anti-windup limits**
$\tilde{X}$	0.0001	10 m
$\tilde{y}$	0.00001	50 m

3.4 DRL algorithms

The two DRL algorithms, proximal policy optimization (PPO) and deep deterministic policy gradient (DDPG), were applied in this thesis. Both PPO and DDPG have gained popularity due to their successes in various simulated robotic tasks [125]. In addition, both of these DRL algorithms have been applied to control of marine vessels [4, 5, 126].

These algorithms share some properties such as applicability to continuous state and action vector, being model-free and online algorithms. They differ in that PPO is on-policy while DDPG is off-policy. On-policy means the agent only learns based on experiences from the current policy, while an off-policy agent can learn from experiences of other policies. Off-policy methods tend to converge slower towards the solution than on-policy methods. A distinctive favourable property of off-policy methods is that when applying DRL to real-life marine vessels, the agent might not be able to explore directly due to safety concerns. It can then be useful to use experiences from a simulator, or by delving into previous real-life experiences. It is therefore interesting to compare these two methods.

In the following sections, the implementation details of these algorithms are presented.

3.4.1 Deep deterministic policy gradient

The DDPG algorithm used in this project was implemented based on the original implementation of Lillicrap et al., based on the same set of hyperparameters and architecture, as presented in the paper [1]. Pseudocode is presented in Algorithm 3 in Section 3.4.1.

The policy $\pi(\mathbf{x})$ was responsible for generating a control action, generating $\mathbf{a} = [f_1, f_2, f_3, \alpha_1, \alpha_2]^\top$ or $\mathbf{a} = [f_1, f_2, \alpha_1, \alpha_2]^\top$. The value function $Q(x, u)$ estimated the cumulative discounted reward for taking action $\mathbf{a}$ in state $\mathbf{x}$.

The policy and value-function were approximated using fully-connected neural networks. Both policy and value-function had two hidden layers, consisting of 400 and 300 hidden units, respectively. The activation function between the hidden layers was ReLU, Equation (2.10b), with batch normalisation between each layer. The states were also normalised, using running average of mean and standard deviation for normalisation. This ensured that the states sent to the neural network were in the same range of values, and thereby increasing learning performance [127].

The policy network was given the state $\mathbf{x}$ as input, and the output was action $\mathbf{a}$. The activation function of the output layer of the policy network, was a hyperbolic tangent-function, (2.10c), with values in the range $(-1, 1)$. This value needed to be scaled before being applied to the marine vessel. This led to the following logic of the neural network:

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b_1}), \\ \mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b_2}), \\ \pi(\mathbf{x}) &= \tanh(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b_3})\mathbf{u}_{\text{scale}} + \mathbf{u}_{\text{mean}}, \end{aligned} \tag{3.34}$$

with maximum control action u_{max}, minimum control action u_{min}, mean control action u_{mean}, maximum difference from the mean control action u_{scale} and $\mathbf{h}_i$ representing the output from hidden layer i, the trainable parameters weight matrix $\mathbf{W}_i$ and bias vector $\mathbf{b}_i$.

The value-function has both $\mathbf{x}$ and action $\mathbf{a}$ as inputs, and output is a scalar. In Lillicraps implementation, the action was added as an input to the second hidden layer. During experimentation, it did not seem to matter whether it was added at the input layer or the second hidden layer. The output-layer of the value-function has no activation, due to the

nature of approximating a scalar value. This leads to the following logic of the neural network:

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1(\mathbf{x} + \mathbf{a}) + \mathbf{b}_1), \\ \mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2), \\ Q(\mathbf{x}, \mathbf{a}) &= \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3, \end{aligned} \tag{3.35}$$

with $\mathbf{h}_i$ representing the output from hidden layer i, with the trainable parameters weight matrix $\mathbf{W}_i$ and bias vector $\mathbf{b}_i$.

Compared to the approach by Lillicrap et al. [1], the networks were trained using ADAM optimiser [128], instead of SGD that Lillicrap et al. were using. This was because ADAM is often a preferred optimiser. The relevant parameters used in the training of all the learning phases are:

- Batch size: 64
- Replay buffer size: 10^6.
- Actor learning rate: 10^{-4}.
- Critic learning rate: 10^{-3}.
- Discount rate $\gamma = 0.99$.
- Target network update rate $\tau = 10^{-3}$.

The docking environment had one terminal case when the agent crashes, but carries on "indefinitely" if not crashing. DDPG assumes an episodic problem. The DDPG-algorithm implemented from Spinning Up solved this by using a maximum episode length. In the docking scenario, the maximum length of one episode was set to 1200 seconds, for all the learning phases.

3.4.2 Proximal policy optimization

The PPO algorithm used in this project was based on the original PPO-clip actor-critic implementation of Schulman et al. [48], where the pseudocode is illustrated in Algorithm 4 in Section 2.3.3.5. The value function was estimated using TD(1)-estimatation, and the advantage was estimated using GAE-lambda.

The PPO-clip algorithm uses a clipped surrogate objective to ensure reasonable policy updates. However, it is still possible to end up with a new policy which is too far from the previous policy. In the SpinningUp implementation "early stopping" was employed to decrease the probability of this happening. Early stopping was performed by stopping to perform gradients steps if the mean KL-divergence of the new policy compared to the previous policy, grew beyond a certain threshold. The threshold used in this thesis was 0.015, based on the recommendations from Spinning up.

The PPO algorithm consisted of a policy $\pi(\mathbf{x})$ responsible for generating a control action $\mathbf{a}$, and a value function $V(\mathbf{x})$, estimating the cumulative discounted reward of being in a state $\mathbf{x}$. The value function in PPO is a state-value function whereas in DDPG, it is an action-value function.

The policy and value-functions were approximated using two fully-connected neural networks, where the networks do not share variables. These networks had the same architecture of two hidden layers, with each hidden layer consisting of 400 hidden units. The activation function between the hidden layers was ReLU. The states were normalised before being applied to the network. This was performed using running mean and standard deviation, the same as for DDPG.

PPO trains a stochastic policy, meaning $\pi(\mathbf{x}) = \mu(\mathbf{x}) + \sigma$, where μ is the deterministic mean policy and σ is the standard deviation of the stochastic policy. The deterministic policy component is approximated using a neural network, with input state vector $\mathbf{x}$, and output is the action vector $\mathbf{a}$. The standard deviation is calculated during the training process. The policy $\mu(\mathbf{x}) + \sigma$ can give actions higher than 1. It was, therefore, necessary to saturate this value before being applied to a physical system, for instance, as performed by OpenAI in their Baseline implementation of PPO [129]. The saturated value was thereafter scaled before being applied to the cargo vessel. This gives the following logic of the stochastic policy:

$$
\begin{aligned}
\mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \\
\mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2), \\
\mu(\mathbf{x}) &= \tanh(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3), \\
\pi(\mathbf{x}) &= (\mu(\mathbf{x}) + \sigma)\mathbf{u}_{\text{scale}} + \mathbf{u}_{\text{mean}}, \\
\pi_{sat}(\mathbf{x}) &= \begin{cases} u_{max} & , \pi(\mathbf{x}) > u_{max} \\ u_{min} & , \pi(\mathbf{x}) < u_{min} \\ \pi(\mathbf{x}) & , \text{otherwise}, \end{cases}
\end{aligned}
\tag{3.36}
$$

with maximum control action u_{max}, minimum control action u_{min}, mean control action u_{mean}, maximum difference from the mean control action u_{scale} and $\mathbf{h}_i$ representing the output from hidden layer i, the trainable parameters weight matrix $\mathbf{W}_i$ and bias vector $\mathbf{b}_i$.

The value network has state vector $\mathbf{x}$ as input, and the output is a scalar. The network does not have any output-activation due to the nature of approximating a scalar value. This leads to the following mathematics of the neural network:

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \\ \mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2), \\ V(\mathbf{x}) &= \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3, \end{aligned} \tag{3.37}$$

with $\mathbf{h}_i$ representing the output, $\mathbf{W}_i$ is the trainable parameters weight matrix at and $\mathbf{b}_i$ is the bias vector from hidden layer i.

The optimiser used during training was ADAM, as recommended by Schulman. The parameters used relevant to the performance of the training was:

- Mini batch size: 20 000
- Replay buffer size: 10^6.
- Discount rate $\gamma = 0.99$.
- Actor learning rate: $3 * 10^{-4}$.
- Critic learning rate: 10^{-3}.
- Number of epoch updates with minibatch (K): max 80
- GAE parameter λ: 0.97
- Clipping range: 0.2

These parameters were based on recommendations from Spinning Up, and adjusted by means of trial and error.

Similarly to DDPG, the PPO implementation from SpinningUp also assumed an episodic problem. The PPO algorithm implemented from Spinning Up solved this by setting a maximum length of an episode. In the docking scenario, the maximum length of one episode was set to 1200 seconds, for all the learning phases.

3.5 Explainable AI

Only a few published works were found within the area of explainable AI in RL. One example is translating the behaviour of RL agents into human-readable descriptions, using post-hoc methods [69, 70]. These methods are not easily applied to a complex setting such as the docking situation, due to the need of human descriptions of all the different states involved. Doing this to the full extent would require a massive effort and go way beyond the scope of this thesis. Techniques making a more transparent RL-agent based on a previously trained RL-agent have also been proposed [71]. This technique would, in principle, have been of interest in our case. However, it seemed to be tested only on simple environments, without any examples demonstrating contributions to explainability, and thus more a proof of concept. A more widely recognised XAI-technique was, therefore, opted for, although not primarily applied to DRL earlier, called shapley additive explanations (SHAP).

Shapley additive explanations [112] is a post-hoc method normally used on supervised learning problems. Shapley additive explanations is a unified framework for interpreting predictions, and has been shown to lead to intuitive explanations and with relatively low computational demands. SHAP was employed to get information about the states impact on the DRL agent's selections of actions. SHAP was selected due to its possibilities of both exploring these contributions from a general point of view, and for a specific time instances.

The SHAP framework creates SHAP values, which signifies an input's importance to a models output. The SHAP value can also be described as an input's average expected marginal contribution to a given output. In this thesis the input considered is the state vector $\mathbf{x}$, the model is the policy $\pi(\boldsymbol{x})$ and the output is an action vector $\mathbf{a}$. From this the SHAP values can be considered as a states contribution to an action. Through out this thesis SHAP values will therefore called contributions, instead of importances.

There exist several open-source implementations of SHAP. Lundberg, co-creator of SHAP, made an open-source implementation [9] including Kernel SHAP and Deep SHAP. Deep SHAP could potentially have been used, giving a specialised Shapley-value estimate for deep neural networks. Due to incompatibilities between the SHAP-library and the neural-network library applied, the model agnostic Kernel SHAP method was used. The opensource Kernel Shap implementation [9] is true to the original SHAP-article described in Section 2.4.1.3.

3.5.1 Kernel SHAP

Kernel SHAP was used to create SHAP values giving the contribution of states to the thruster forces and angles, also called actions. These SHAP values estimate the average expected

marginal contribution of a state to a given action after all possible combinations of states have been considered. This means that at any given time instance t, the action $a_{t,i}$ can be estimated as the sum of all the states SHAP values $\phi_{t,j}, j \in [1, m]$, giving:

$$a_{t,i} \approx \sum_{j=1}^{m} \phi_{t,j} + \phi_{t,0}, \tag{3.38}$$

where m is number of states, action $a_{t,i}$ is from action vector $\mathbf{a}$ given by the policy $\pi(\mathbf{x})$ and $\phi_{t,0}$ is the expected value of action a_i over the background set, $\phi_{t,0} = \mathbf{E}(a_i)$. The SHAP values can be viewed as the fairly estimated contribution of a state to the selection of a given action.

Kernel SHAP finds the SHAP values of a test dataset, using the neural network of the DRL policy $\pi(x)$ and a background dataset. The background dataset consists of random states from the environment and is used for integrating out states. The SHAP values of the state is determined by deliberately and systematically replacing the states from the test dataset with values of states from the background dataset and observe the change in action a from policy $\pi(\mathbf{x})$. If a state only has one value in the background dataset, the SHAP value will be calculated as zero, due to no variation. A K-means function is recommended for bigger background dataset, to summarises the training set, as k clusters, each weighted by the number of points they represent.

It was of interest to analyse the impact of states on the action selection, both in general and during each time step of a docking episode. In Section 2.4.1.4 different ways of interpreting the SHAP values, both globally and locally were presented. In this thesis the following interpretations were made:

- **General contribution of states**: This is the the mean absolute of the SHAP values for each feature for each action, and indicates the global effects/contributions of the states when selecting actions.
- **Contribution of states in an episode**: This is the SHAP values for a specific time instance during a docking episode. This gives an estimation of the contribution of each state changes during a episode.

The exact value of the contribution of a state to a given action might not be that interesting when analysing the development of the state's contribution to an action. This is because the SHAP values are dependent on the value of the action. A relative contribution of a state to a given action can be easier to analyse across several timesteps with differing values of the actions. A relative measure of the contribution is the percentage of a state's contribution to a given action, given the total sum all contributions. An example of this is that surge u

Table 3.7: Table of valid sets used to collect random samples from the environment for learning phases 2,4,5

	LP2:Berthing	**LP4:Distance berthing**	**LP5:Docking**
x_d [m]	700	700	700
y_d [m]	529.5	529.5	529.5
ψ_d [rad]	$-\pi/27$	$-\pi/27$	$-\pi/27$
x [m]	$(x_d - 120, x_d + 120)$	$(x_d - 400, x_d + 400)$	$(x_d - 400, x_d + 400)$
y [m]	$(y_d - 80, y_d + 80)$	$(y_d - 400, y_d + 400)$	$(y_d - 400, y_d + 400)$
ψ [rad]	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$
u [m/s]	(-0.5,0.5)	(-0.5,0.5)	(-0.5,0.5)
v [m/s]	(-0.05,0.05)	(-0.05,0.05)	(-0.05,0.05)
r [rad/s]	(-0.005,0.005)	(-0.005,0.005)	(-0.005,0.005)

contributes 20 % and sway contributes 5 % to why the thruster force f_1 is set to 50 kN.

The relative contribution $rc_{t,k}$ of state s_k, with SHAP values $\phi t, i, k$ at time t, to action a_i can be calculated as:

$$rc_{t,i,k} = \frac{|\phi_{t,i,k}|}{\sum_j^m |\phi_{t,i,j}|} \tag{3.39}$$

There were used 2000 random samples collected from the environment, within the range given by valid sets presented in Table 3.7, to provide the background data and test data used to calculate global explainability. This is almost the same valid sets as used to generate the training scenarios, in Table 3.5. It was also added random states of the sway v and yaw rate r in the range $v \in (-0.05, 0.05)$ m/s and $u \in (-0.005, 0.005) m/s$. These ranges were added based on the observed values through an episode.

The 2000 random samples were divided in two: The background data with 80 %, and test data with 20 %, based on a general rule of thumb from applications within supervised learning. The local explainability was analysed during one docking episode, studying the change in relative contribution. The test data during the docking episode consisted of 1200 test samples.

3.6 Tools

The DRL algorithm, XAI-algorithm and environment used the Python programming language, which has a rich collection of machine learning libraries available.

The DDPG and PPO implementation in this project were based on an implementation from Spinning Up [2]. Spinning Up is an educational resource made by the organization OpenAI, with the mission to simplify learning of DRL. Their code philosophy is to produce code that is as simple as possible, highly consistent with other associated algorithms and exposing fundamental similarities between them. Due to this philosophy, the following changes were made to Spinning UP's implementation of DDPG, either by providing strong support to the original implementation of DDPG by Lillicrap et al. .[1] or by doing alterations in select areas:

- Including batch normalisation, handling larger differences in the magnitude of observations. This change adopts the Spinning Up implementation to conform with the original implementation by Lillicrap.
- Changing the transformation of action vector, to handle action-space with non-zero means. This was implemented to handle the action-space of the marine vessel.
- Update the network at each step, and not at the end of the episode, as presented in the original implementation by Lillicrap et al. .[1].
- Including running mean normalisation of states before entering them into the replay-buffer. This was implemented to avoid an overly large range of values in the state vector, and thereby improve learning.

The PPO implementation was also altered in select areas:

- Changing transformation of action vector, to handle action-space with non-zero means. This was implemented to handle the action-space of the marine vessel.
- Adding saturation of the action vector, such that the vessel always received control inputs in the valid range.
- Including running mean normalization of states before entering them into the buffer. This was performed to avoid an overly large range of values in the state vector, and thereby improve learning.

Both DDPG and PPO-implementation used the library Tensorflow [6], to implement function approximators of policy and value functions. Tensorflow is an open source platform for machine learning, providing a library that eases the implementation of backpropagation, batch normalization etc.

The Explainable AI method Kernel-SHAP was tested using the libary SHAP [9]. Other libraries used were numerical libraries Pandas [10], Numpy [7], and visualization library Matplotlib [8].

Chapter 4

Results and discussion

In this chapter the main results of the DRL-agents are presented and discussed for each learning phase. This chapter consists of the following sections:

- Section 4.1: Presents and discusses the results of the LP1-a:DP and LP1-η:DP agents. It was tested using both PPO and DDPG to train the agents.
- Section 4.2: Presents and discusses the results of the LP2-a:Berthing and LP2-η:Berthing. It was tested using both PPO and DDPG to train the agents.
- Section 4.3: Presents and discusses the results of the LP3:Target tracking agent. It was tested using both PPO and DDPG to train the agents.
- Section 4.4: Discusses the results of training the LP1-LP2, with regard to DDPG vs. PPO and the differences in having an objective of either placing a vessel inside a target-rectangle (LP1-a:DP and LP2-a:Berthing) or at a target pose (LP1-η:DP and LP2-η:Berthing).
- Section 4.5: Presents and discusses the results of the LP4:Distance berthing agents. It was used only PPO to train the agent.
- Section 4.6: Presents and discusses the results of the LP5:Docking agent, It was used only PPO to train the agent.
- Section 4.7: Through the LP1-LP5 it was experienced aggressive use of the thrusters. This sections presents the results of trying to reduce the aggressive actions of the LP4:Distance berthing agent. The LP4:Distance berthing agent was selected due to being the best performing end-to-end docking solution, and sharing all the properties of LP5:Docking, except the speed limit added.

Learning phases LP1-LP3 were trained using both proximal policy gradient (PPO) and deep deterministic policy gradient (DDPG). Leaning phases LP4-LP5 were trained using only PPO. This was primarily because the previous learning phases (LP1-LP3) experienced better performance by using PPO, and DDPG required vastly extended learning periods exceeding available computational capacity, ruling out training both DDPG and PPO. For each of these, the learning processes were t in order of some hours at the best to several days at the worst, for each run.

For each learning phase the training process and performance of the trained agent is presented and analysed. The training process is analysed using the finite-horizon undiscounted return as a function of the number of interactions between the agent and the environment during training. The finite undiscounted return is the sum of all rewards received by the agent during one episode, and will be called return through the rest of this chapter.

The performance of the agent is analysed for different episodes. The episodes are generated with random initial states of pose $\boldsymbol{\eta}$ and velocities $\boldsymbol{\nu}$, described in Section 3.3.5. Some of these episodes are presented using numerical findings. The numerical findings consist of:

- **Mean absolute distance to target d_d and heading error $\tilde{\psi}$**: This is the mean absolute value over an episode, and gives an indication of how fast the vessel converges towards the target.
- **Minimum absolute distance to target d_d and heading error $\tilde{\psi}$**: This is the minimum absolute value over an episode, indicating the best achieved accuracy of vessel reaching the target.
- **Mean absolute normalised change of control inputs $\dot{f}$ and $\dot{\alpha}$**: This is the mean absolute value over an episode and indicates the aggressiveness of the thrusters.
- **Return**: This is the cumulative reward received by the agent when executing the episode.

For each learning phase, one episode is illustrated using:

- **Time series plots: These gives states, actions, and rewards at a given time:**
  - State plots: Plot of relevant states of the agent.
  - Reward plots: Plot of rewards given to the agent, and relevant components if several sub-objectives were involved.
  - Thruster plots, including the relevant thruster angles and forces the agent used.

- **Trajectory plot**: Plots of the trajectory of the vessel in the docking area, in NED reference frame.
- **Video**: Videos are available showing the vessel's trajectory, offering a better comprehension of the vessels movement. These are appended as hyperlinks in the respective sections.

In addition to analysing the agents based on performance, selected learning phase agents were analysed using shapley additive explanations (SHAP). The following results are presented::

- LP2:Berthing: SHAP-values were used to improve the performance of the LP2:Berthing agent by analysing the state's general contribution to the control inputs.
- LP4:Distance berthing and LP5:Docking: SHAP-values were used to analyse the behaviour of the agent, through both analysing the states general contribution to control inputs and states contribution to control inputs at given timesteps of a selected episode.

4.1 LP1:Dynamic positioning

The agents performing dynamic positioning (LP1:DP) had the objective of controlling the vessel to a desired pose and staying in that pose once there. The vessel starts the episode with a low velocity (between -0.5 to 0.5 m/s), at a random pose within 60 meters from the desired pose. The agents control the thruster forces and angles $[\alpha_1, \alpha_2, f_1, f_2, f_3]$.

There were two types of dynamic positioning agents. The LP1-a:DP agent tried to get the vessel inside a target-rectangle, and having an objective of $f_a \rightarrow 1$, meaning getting the fraction of vessel area inside target-rectangle f_a to one. The LP2-η:DP agent had the objective of controlling the vessel to a desired pose η_d (LP2-η:DP). Both of these methods were trained with both DDPG and PPO, using the training episodes described in Section 3.3.5. The parameters of the reward function are expressed in Table 3.3 for the PPO agents and in Table 3.3 for the DDPG agents.

The reward function for LP1-a:DP, Equation (3.24), has two reward components: One reward component for avoiding obstacles r_{obs} and one for parking the vessel inside a rectangle r_{f_a}. All the reward components in this thesis have a maximum value of one if the corresponding weight is set to one. The penalty for the vessel coming in contact with an obstacle was $C_{obs,T} = -500$, while the weight for coming closer to an obstacle $C_{obs} = -2.5$. The weight for r_{f_a} was $C_{f_a} = 5$. These were found using trial and error, where it was they were adjusted so that the agent found it more encouraging to explore the environment than to terminate the episode through crashing into an obstacle.

The same line of thinking was also used for the LP1-η:DP reward function, Equation (3.23), with three reward components: One reward component for avoiding obstacles r_{obs} and two for reaching a desired pose $r_{d_d} + r_{\tilde{\psi}}$. The weights of r_{obs} was selected to be the same as LP1-a:DP, while the reward components reaching a desired pose was $C_{\tilde{\psi}} = 2.5$ and $C_{d_d} = 2.5$. They were chosen to make it easier to compare the results of the two methods, in addition to giving similar results as for LP1-a:DP.

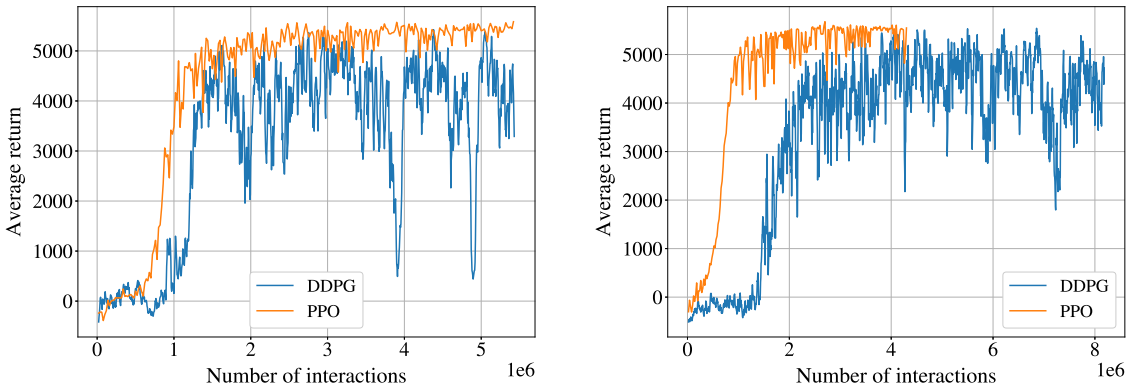
The variances of the reward components were shaped to encouraged the agent to explore, but trying to avoid making the agent too satisfied before reaching the target either. The variance of the Gaussian reward component for being close to obstacles $\sigma_{obs} = 1$, was selected to ensure that the agent also explored areas close to the quay. The variance of the Gaussian reward function LP1-a:DP σ_{d_d} and σ_{f_a} was kept as low as possible, in order to have agents with high accuracy of the reaching the target. The LP1-a:DP PPO and DDPG agents used identical variance $\sigma_{f_a} = 0.4$. The LP1-η:DP DDPG agent needed relatively high values of σ_{d_d}, in order to encourage the agent to converge towards higher average returns during training. This, unfortunately, also lead to a loss of accuracy. The lowest variances giving good convergence for LP1-η:DP was therefore $\sigma_{\tilde{\psi}} = 10$ degrees for both PPO and DDPG, and $\sigma_{d_d} = 10$m for PPO and $\sigma_{d_d} = 30$m for DDPG.

4.1.1 Training progress

Figure 4.1 presents the development of the average return during training as a function of the number of interactions between the environment and the agent. The DDPG agents converged slower to a lower average return than the PPO agents, for both learning phases LP1-a:DP and LP1-η:DP.

The two approaches LP1-a:DP and LP1-η:DP produced similar training progress for the two PPO agents, with convergence towards approximately 5100 in average return after approximately two millions of interactions between the environment and the agent. The maximum achievable return was 6000, only achievable if the vessel started at target, and stayed there the entire episode.

The DDPG agents exhibited larger differences in the training progress of LP1-η:DP and LP1-a:DP. The LP1-a:DP agent converged somewhat slower than LP1-η:DP. This is most likely due to less sparse reward function of LP1-η:DP, compared to LP1-η:DP DDPG agent. This is further discussed in Section 4.4.



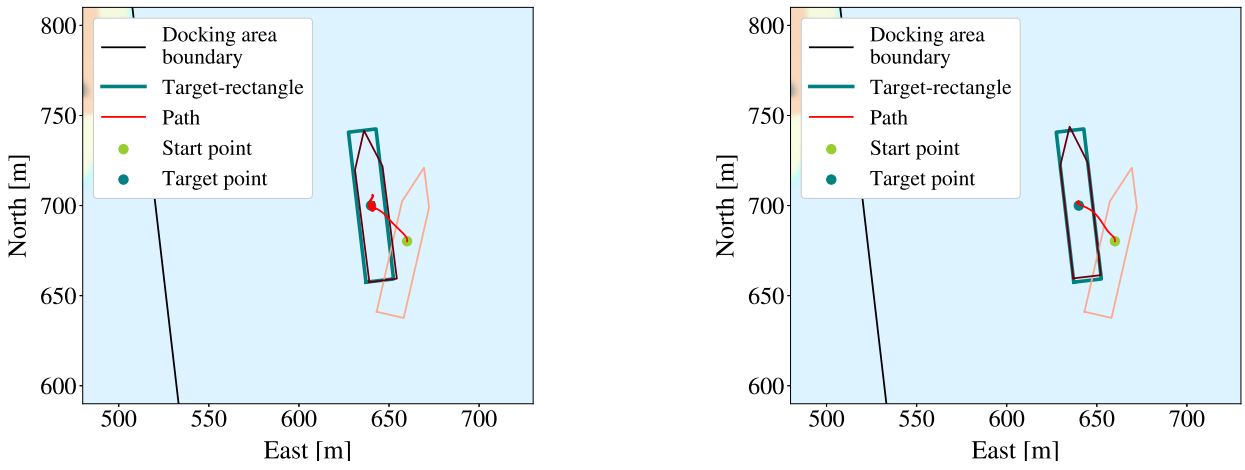
(a) LP1-η:DP

(b) LP1-a:DP

Figure 4.1: The average return during training as a function of number of interactions between agent and environment, for the LP1:DP agents.

4.1.2 Performance of LP1-a:DP

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s]. Both the DDPG and PPO agent were able to solve the task of placing the vessel inside the target-rectangle and holdings its there. The trajectory plots are presented in Figure 4.2, with corresponding video LP1-a-DP-PPO and LP1-a:DP-DDPG. In the illustrated test episode, the agents controlled the vessel nicely towards the target-rectangle, with best achieved accuracy of 0.553 m and 0.190 m from the target, respectively. This is arguably a relatively good result for a vessel of length 76 m.



(a) DDPG

(b) PPO

Figure 4.2: Trajectory plots for the LP1-a:DP agents. Test episodes start with initial states $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].

The state plots of the test episode are presented in Figure 4.3. Plots of heading error $\tilde{\psi}$ and distances to target $\tilde{x}$ and $\tilde{y}$, demonstrate that both the PPO and DDPG agents reached the desired target efficiently, but exhibited tiny fluctuations around it. The reward plot, in Figure 4.5a, shows noticeable fluctuations in the rewards given for the area of the vessel inside the target-rectangle r_a of the DDPG agent. The PPO agent, on the other hand, exhibited less noticeable fluctuations in the reward function, and achieved a higher return than the DDPG agent, as illustrated in Figure 4.5b. This was expected after analysing the training progress, where the DDPG agent had a lower average return, indicating that it was less able to maximise the return and thereby the objective. From analysing several examples, the DDPG agent often exhibited higher fluctuations than the PPO when performing DP.

The PPO agent appears to achieve close to the maximum achievable reward keeping the vessel inside the target-rectangle, shown in Figure 4.5b. The DDPG agent appears to get a somewhat smaller reward in the same period, with some fluctuations, illustrated in Figure 4.5a. When analysing the trajectory and state plot, the DDPG agent appears to have a small part of the tip (bow) outside the target-rectangle, as shown in Figure 4.2a, and exhibiting a small negative distance to target in the x-direction, as shown in Figure 4.3a. This can come from the fact that we are placing a pentagon inside a rectangle, where the small tip of the triangle at the front of the vessel does not account for much of the episode's return. This can discourage the agent from taking notice of any tip extending outside the rectangle.

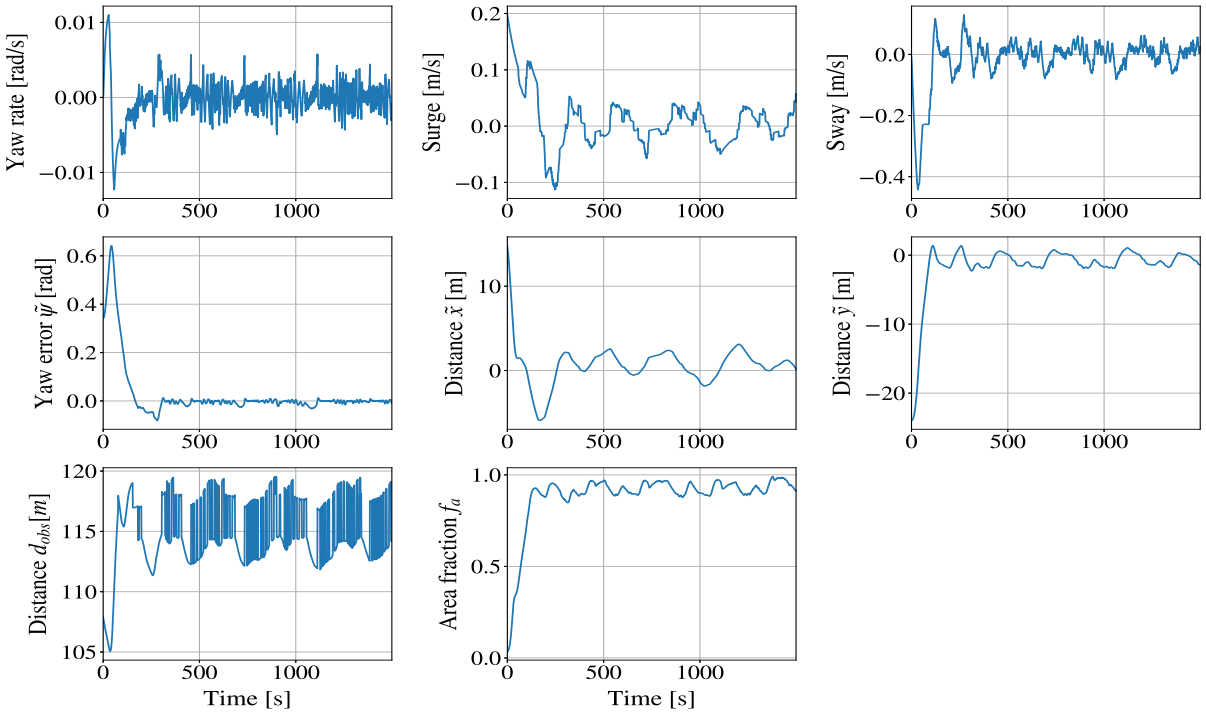
The plots of the velocities of the PPO and DDPG agent are illustrated in Figure 4.3. These show that the velocities are close to zero when the vessel has reached the desired target rectangle. The velocities, however, exhibit small fluctuations, which can be caused by the agent actively trying to keep the vessel inside the rectangle, as usual in a DP-solution, in addition to the reasons mentioned above. The thruster plots, in Figure 4.7, confirms that the agents do not turn off the engines when reaching the target. One reason that the agent uses the control input aggressively can be that it does not receive any penalty for doing it. If using the thrusters aggressively gives a high return, the agent does not have a sufficient reason for searching for solutions with lower usage of the thrusters. How to reduce aggressive use of the thrusters is discussed in Section 4.7.

The test example above illustrated the characteristic behaviour of the two models, with similar efficiency and accurate behaviour. Results of the PPO and DDPG agents behaviour with multiple initial states are presented in Table 4.1. The DDPG and PPO agents demonstrated similar high values of the return in each episode, with the maximum achievable return of 7500, and only achievable if the vessel starts at the target and is kept there the entire episode. This indicates that the agents have understood the objectives fairly well. The mean absolute distances to target d_d and heading error $\tilde{\psi}$ of the test episode are fairly low, indicating efficient

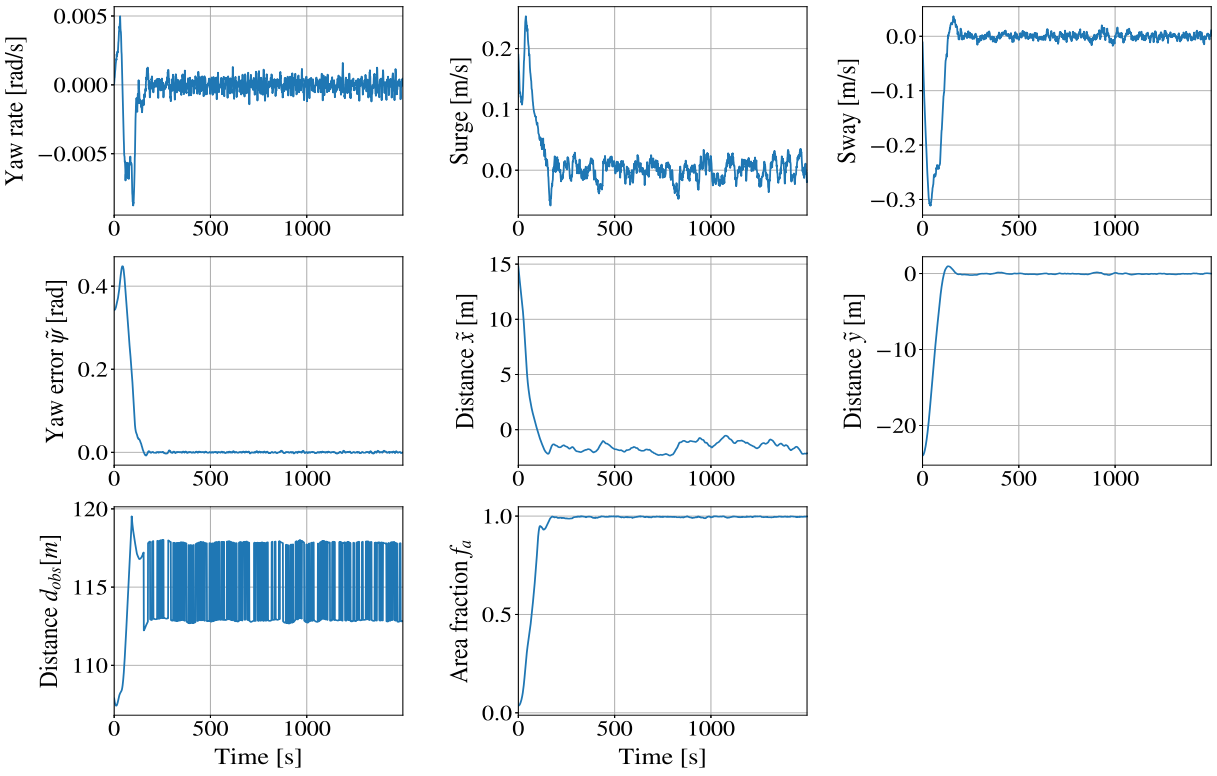
Table 4.1: The results from the LP1-*a*:DP on several test episodes. The test episodes are run for 1500 seconds.

DRL-algorithm	**Initial position** $[x, y, \psi, u, v, r]$ **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Mean absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Return**
PPO	680, 660, 0.23,0.2, 0, 0	0.553, 0.000	2.463, 0.025	7162.95
DDPG		0.190, 0.000	2.722, 0.043	7017.76
PPO	735, 655, 0.56,-0.155, 0, 0	0.438, 0.000	4.632, 0.034	7149.21
DDPG		0.633, 0.000	4.386, 0.048	7001.27
PPO	757,615,-0.25,-0.446,0,0	0.479, 0.000	7.579, 0.024	6745.55
DDPG		0.049, 0.000	7.386, 0.018	6861.15
PPO	657, 620, -0.64,0.350, 0, 0	0.536, 0.000	4.592, 0.030	6776.74
DDPG		0.032, 0.000	4.597, 0.060	6691.67
PPO	710, 613, 0.13,-0.029,0,0	0.569, 0.000	3.974, 0.015	7080.40
DDPG		0.023, 0.000	3.809, 0.018	7043.60

control of the vessel. The minimum absolute value of the distance to target d_d indicates that the vessel has a best achieved accuracy of lower than 0.5 meters from the target.

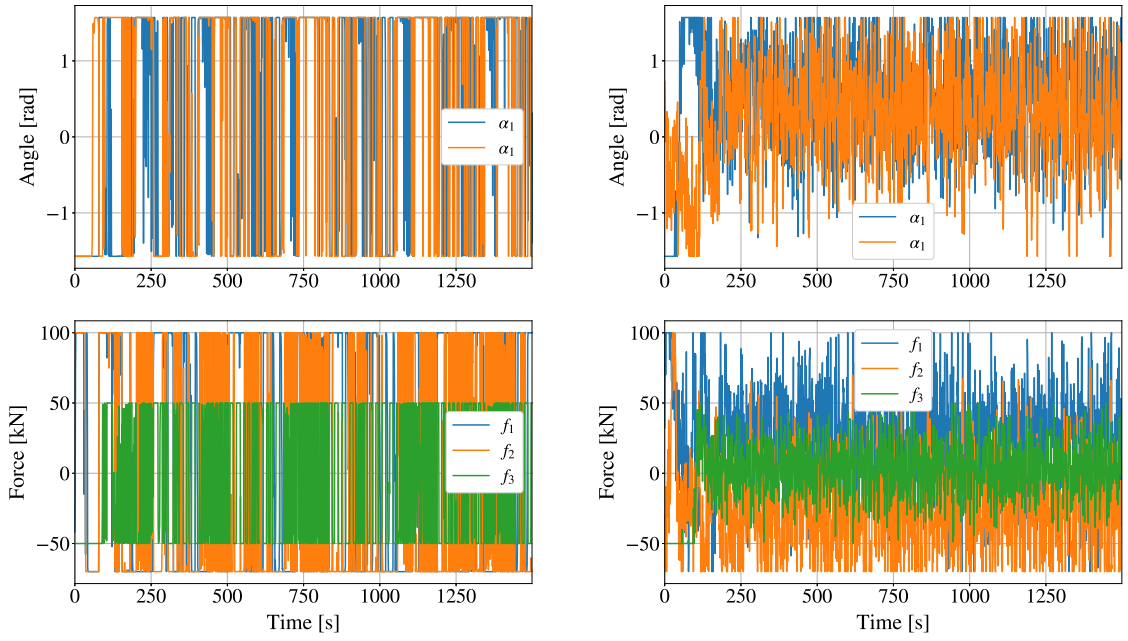


(a) DDPG



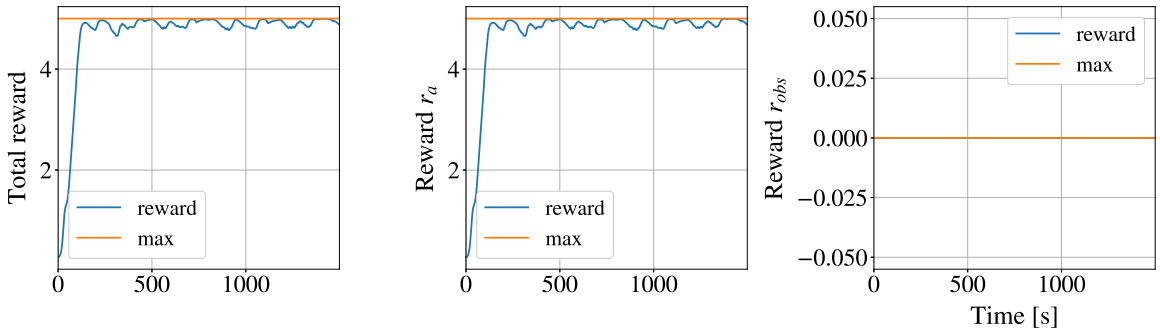
(b) PPO

Figure 4.3: State plots for the LP1-a:DP agents. Test episodes start with initial states $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].

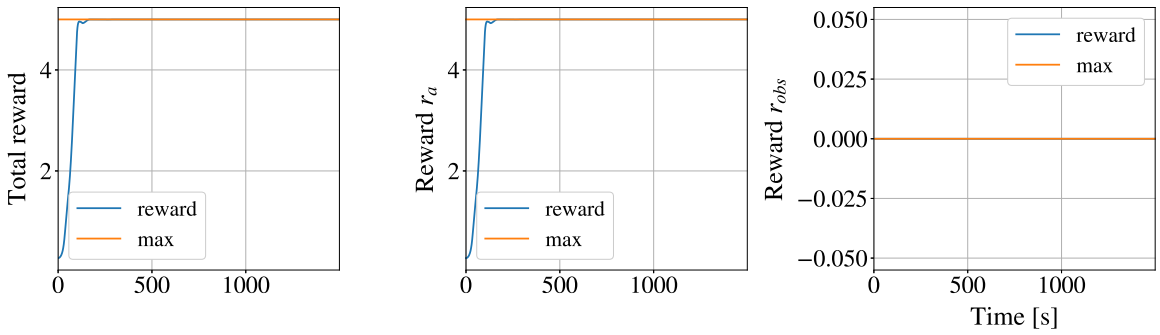


(a) DDPG (b) PPO

Figure 4.4: Thruster plots for the LP1-a:DP agents. Test episodes start with initial states $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].



(a) DDPG



(b) PPO

Figure 4.5: Reward plots for the LP1:DP agents. Test episodes start with initial states $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].

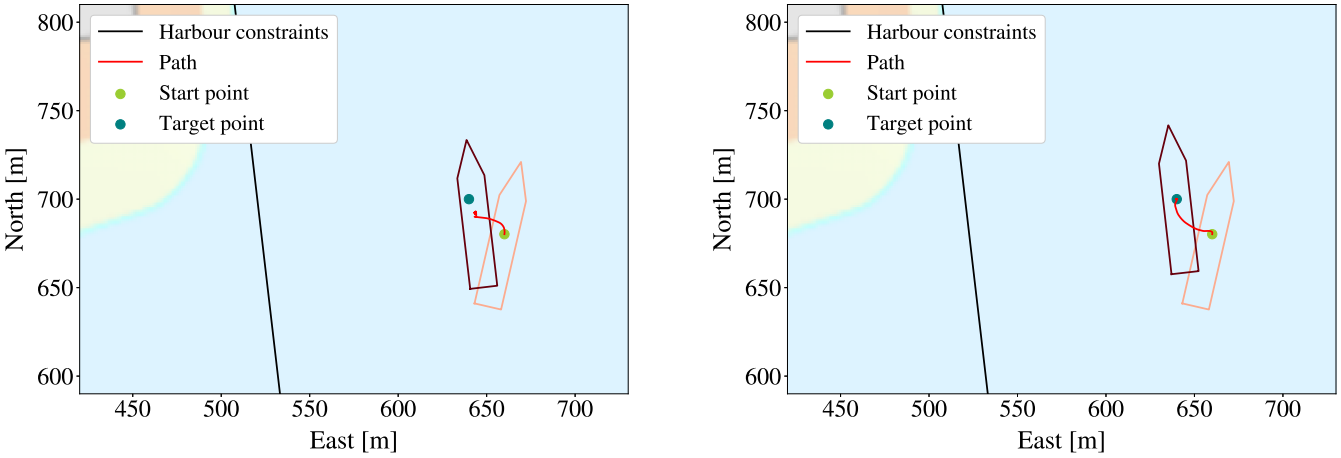
4.1.3 Performance for LP1-η:DP

The test episode illustrated in this section had the same initial conditions as for LP1-a:DP, with initial vessel state $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s, 0 m/s, 0 m/s]. The PPO agent was able to control the vessel to the desired target-rectangle and keep it, with an best achieved accuracy of 1.814 m from the target. The trajectory plot of the PPO agent is illustrated in Figure 4.6b, with a corresponding video LP1-eta-DP-PPO. This illustrates that the PPO agent is capable of reaching the desired pose nicely. The reward plots are visualised in Figure 4.7. These shows that the PPO agent achieved high rewards quite fast, solving the given objectives. The result of the LP1-η:DP PPO agent is similar to the results of the LP1-a:DP agents, but they do not share the same trajectory. This means that they all have found somewhat different policies, where all are efficient, with similar travel time to the target and accuracy of reaching the target.

The LP1-η:DP DDPG agent was able to control the vessel towards the berth in the illustrated test episode, but with a noticeably lower best achieved accuracy of 9.5 m from the target. The trajectory of the vessel controlled by the DDPG agent is visualised in Figure 4.6a, with the corresponding video LP1-eta-DP-DDPG. The figure illustrates that the vessel stopped close to the target. The states of the test episode are visualised in Figure 4.8. The distances to the target $\tilde{x}$ and $\tilde{y}$ converged quite fast towards zero for the PPO agent, while the DDPG agent exhibited a steady-state error. The steady-state error is most likely due to the increased variance σ_{d_d} of the Gaussian reward component r_{d_d} for being in proximity of the target pose. It was mentioned earlier that it was necessary to increase the variance to achieve convergence of the DDPG-agent within reasonable training time, where the downside was reduced ability to converge fully to the target. This notion further confirmed by the plots of the reward functions illustrated in Figure 4.7. The DDPG agent achieves high rewards from poses to far away from the target for the desired accuracy. It is, therefore, understandable why the DDPG agent does not want to work towards reaching the desired target position. This illustrates that shaping rewards is a complex operation, and it needs to be closely correlated with the objective of the task.

One interesting observation is that the LP1-η:PPO agent exhibited fewer fluctuations than the LP1-a:PPO agent. It is shown in in Section 4.4 that the reward function for reaching the target of LP1-a:PPO exhibited a larger intersecting area, closer to the optimum than LP1-η:PPO. This suggests that with "sharper" reward functions around the target, the agent might exhibits fewer fluctuations. By sharper reward function we mean differentiating more closer to the objective.

The plot of the velocities, in Figure 4.8, shows that the DDPG agent exhibits higher fluc-



(a) DDPG (b) PPO

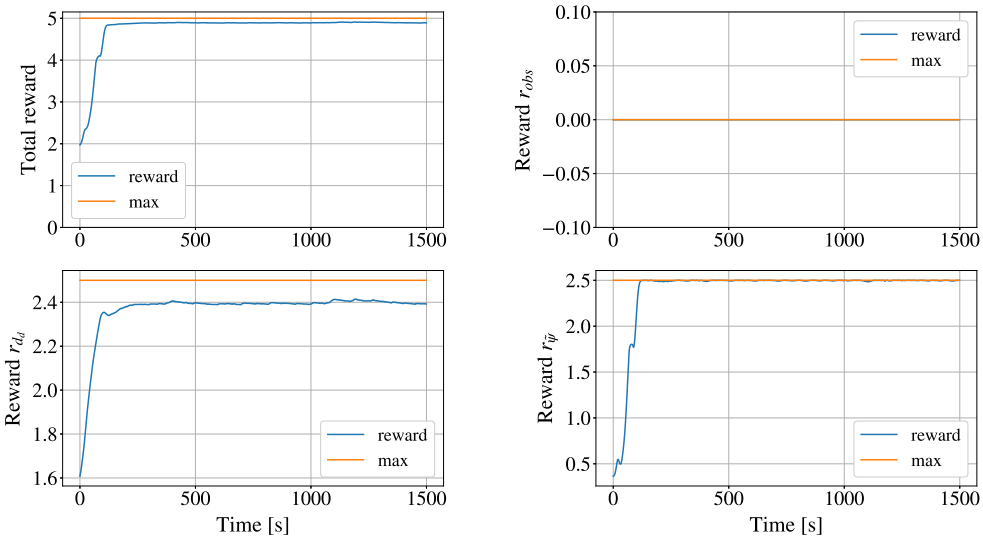
Figure 4.6: Trajectory plots for LP1-η:DP agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].

tuations than the PPO agent in the velocities surge u, sway v and yaw r. This is probably caused by the fact that the DDPG agent does not have to reach the target at the same level of accuracy as the PPO agent to achieve high rewards. Even though the PPO agent achieves a better accuracy, both agents use the thruster quite aggressively to both reach and hold the target, as observed in Figure 4.9.

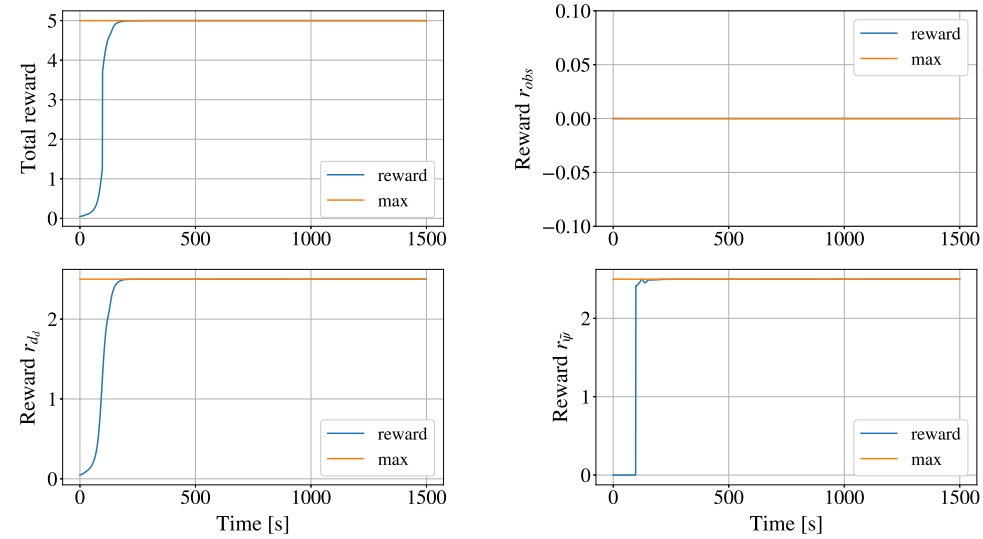
The test example illustrated above shows the generally observed behaviour of the two models. Results from several episodes with different initial states are presented in table 4.2. The PPO agent generally solved the task efficiently, with relatively high accuracy to the target pose, while the DDPG agent achieved lower accuracy with more fluctuations around the target position. The mean absolute distances to target d_d and heading error $\tilde{\psi}$ over an episode were fairly low for the PPO agent. The DDPG agent, on the other hand, exhibited higher values, an unfortunate side effect of the shaping of the reward function for the DDPG agent to ensure sufficient tracking towards the target. The shaping gave both the PPO and DDPG agents high return in most episodes, with the maximum theoretical value of 7500. The DDPG agent did however on some occasions collide the vessel into an obstacle, yielding a low return in those episodes. These findings indicate that the PPO agent was better at solving the objective of LP1-η:DP.

Table 4.2: he results from the LP1-η:DP on several test episodes. The test episodes are run for 1500 seconds.

DRL-algorithm	**Initial position** $[x, y, \psi, u, v, r]$ **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Mean absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Episoe return**
PPO	680, 660, 0.23,0.2, 0, 0	0.002, 0.000	1.814, 0.015	7008
DDPG		7.929, 0.000	9.490, 0.023	7153
PPO	735, 655, 0.56,-0.155, 0, 0	0.002, 0.000	2.049, 0.037	7020
DDPG		1.608, 0.000	10.625, 0.039	6903
PPO	757,615,-0.25,-0.446,0,0	0.002, 0.000	2.174, 0.019	7093
DDPG		1.523, 0.000	10.377, 0.008	6929
PPO	657, 620, -0.64,0.350, 0, 0	0.002, 0.000	3.450, 0.030	6758
DDPG		0.000, 0.000	39.267, 1.226	5
PPO	710, 613, 0.13,-0.029,0,0	0.002	1.822, 0.038	6955
DDPG		0.890, 0.000	7.123, 0.009	7043.60

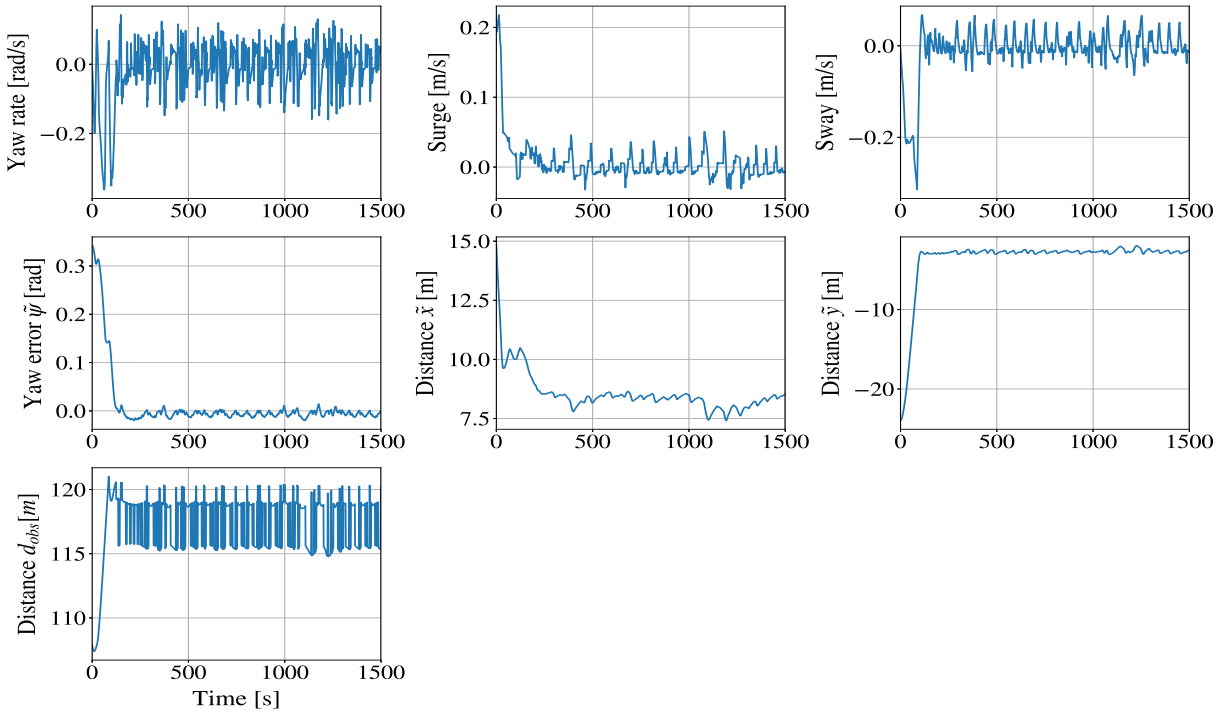


(a) DDPG

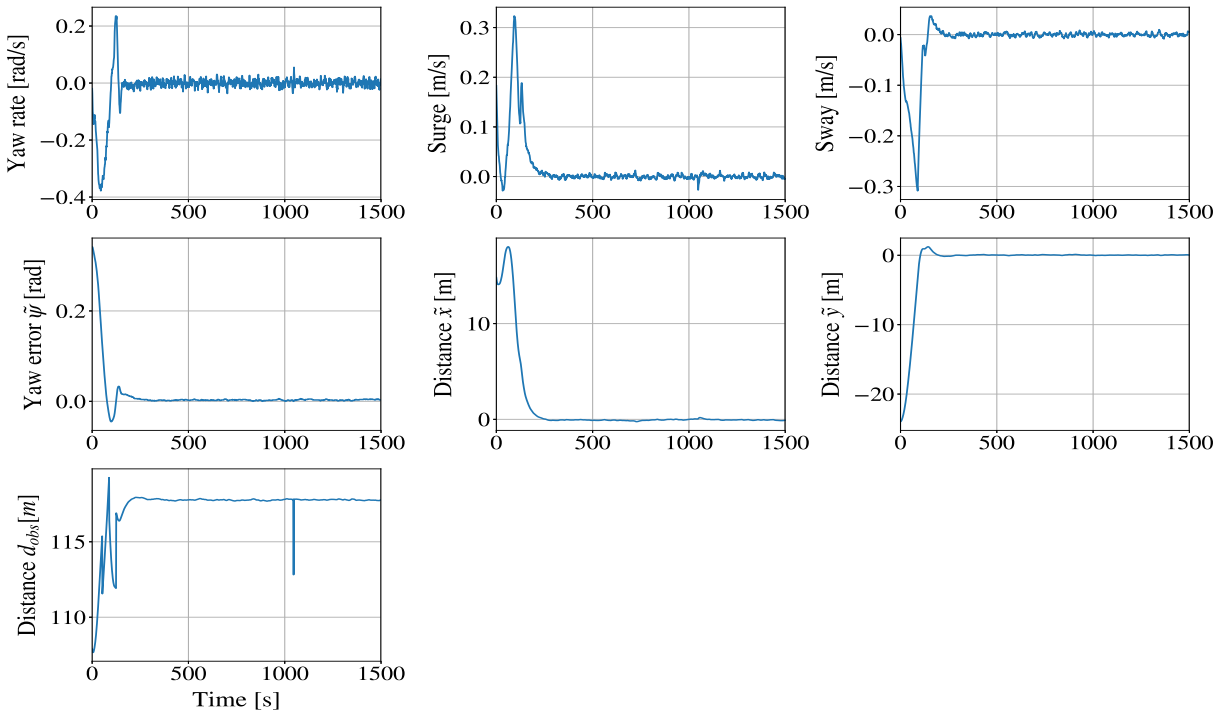


(b) PPO

Figure 4.7: Reward plots for LP1-η:DP agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].

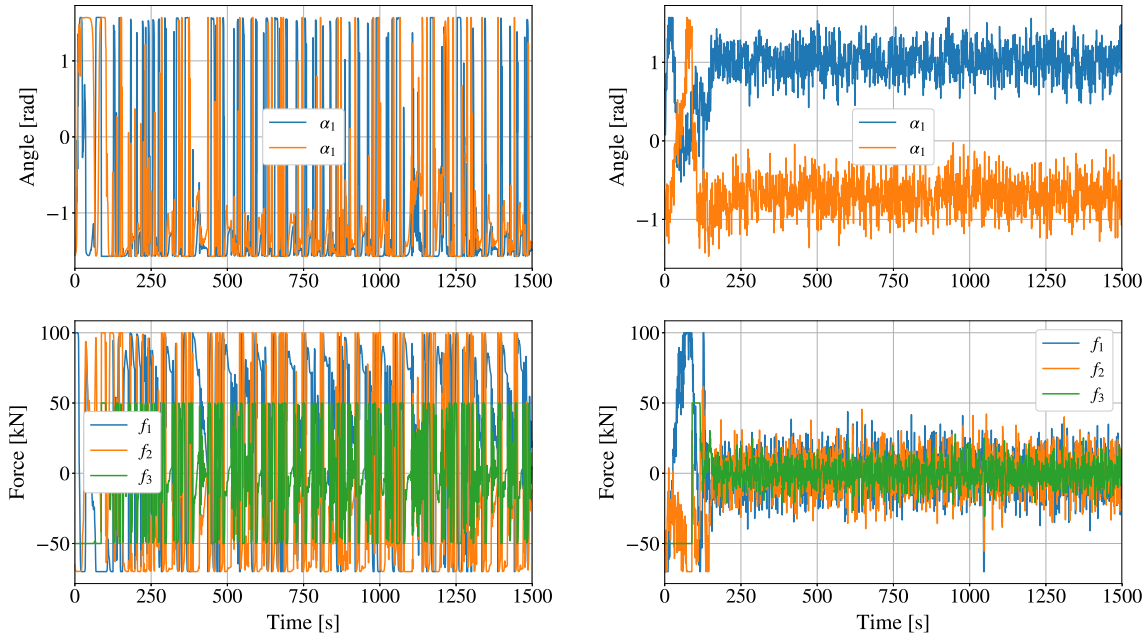


(a) DDPG



(b) PPO

Figure 4.8: State plots for LP1-η:DP agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].



(a) DDPG (b) PPO

Figure 4.9: Thruster plots for LP1-η:DP agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [680 m, 660 m, 0.23 rad] and $\boldsymbol{\nu}$ = [0.2 m/s,0 m/s,0 rad/s].

4.2 LP2:Berthing

The agent performing berthing (LP2:Berthing) had the objective of controlling the vessel to the berth, with a start position within 120 meters from the berth and a start velocity in the range of -0.5 to 0.5 m/s. Once inside the berth, the vessel should stay there. The agent controlled the vessel through thruster forces $[f_1, f_2, f_3]$ and angles $[\alpha_1, \alpha_2]$.

Two different berthing objectives were tested. The LP2-a:Berthing agent should get (and keep) the area of the vessel inside a berth-rectangle (LP2-a:Berthing), meaning $f_a \rightarrow 1$. The LP2-η:Berthing should get the vessel to a desired berth pose η_d, and keep it there. Both of these methods were trained with both DDPG and PPO, using the training episodes described in Section 3.3.5.

The reward function of LP2-a:Berthing, equation (3.24), and LP2-η:Berthing,equation (3.23), are the same as LP1-a:DP and LP1-η:DP, but with an additional reward component $r_{d_{dot}}$ for moving towards the target. The reward function parameters of reward components similar between LP1-a:DP and LP2-a:Berthing and LP1-η:DP and LP2-η:Berthing are the same. The additional reward component $r_{d_{dot}}$ for moving towards the target, had the weight $C_{d_{dot}} = 1$. This was the lowest weight which achieved the objective, as lower seemed not to encourage the agent enough. The reward function parameters are expressed in Table 3.3 for PPO and Table 3.3 for DDPG .

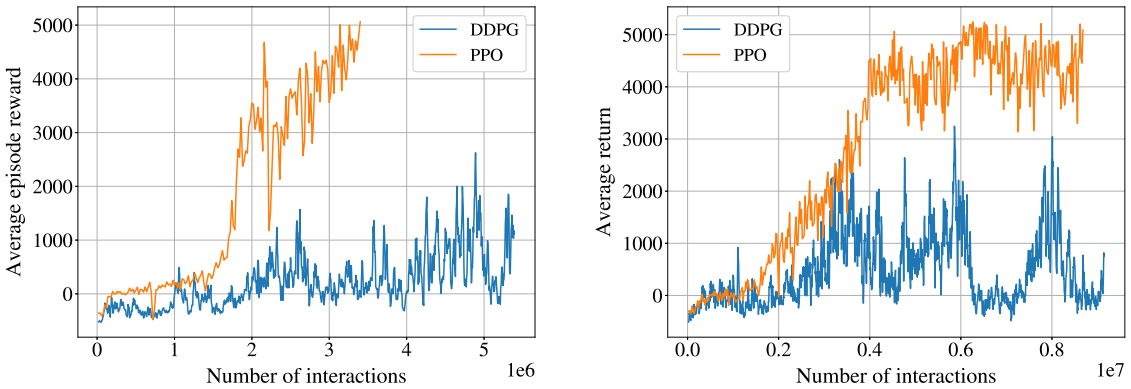
4.2.1 Training progress

The training progress of the DDPG and PPO agents of LP2-a:Berthing and LP2-η:Berthing is visualised in Figure 4.10. The average return of the PPO agent of LP2-η:Berthing and LP2-a:Berthing, converged to a similar value of 4800, with a theoretical maximum return of 6000 if the vessel starts inside the berth and staying there the entire episode. The LP2-η:Berthing agent converged at around 4 million interactions between the agent and the environment, while LP2-η:Berthing agent converges a bit slower, at around 6 million interactions.

The PPO agents were able to converge to higher average returns during the training process than the DDPG agents. A considerable effort was undertaken to find workable settings of hyperparameters of the DRL algorithms. The shaping of the reward functions was also investigated, such as decreasing the sparsity by increasing the variance of reward components for parking a vessel inside a berth-rectangle r_{f_a} or for reaching the desired pose r_{d_d}. This shaping could improve the first part of the travel towards the berth, but decreased the accuracy, in the same manner as for LP1:DP. Shaping rewards and selecting hyperparameters

are significant operations of great importance to the outcome of the learning, as well as time-consuming.

Another possible way to improve the performance of the DDPG agents could be to train the DDPG agents longer. Some runs extending over five days were tried, but proved futile, never near to give the same high average returns as PPO. Consequently trying longer training periods was not prioritised, as PPO agents were capable of solving the task withing a more reasonable time. The computational capacity demands for further testing of the DDPG agents could be exceedingly high given the resources at hand and was therefore not tested.



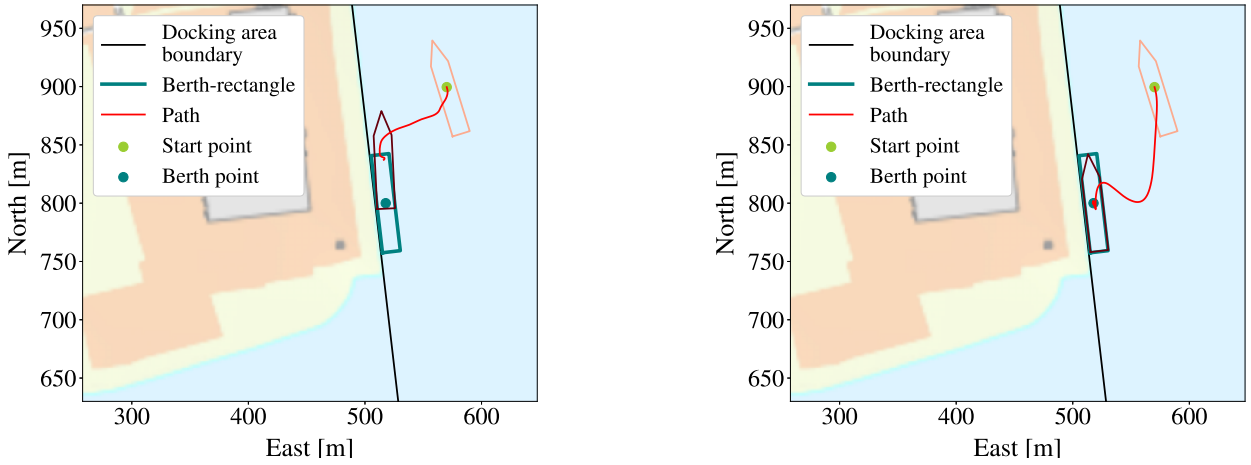
(a) LP2-η:Berthing (b) LP2-a:Berthing

Figure 4.10: The average return during training as a function of number of interactions between agent and environment, for the LP2:Berthing agents

4.2.2 Performance of LP2-a:Berthing

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s], which is approximately 120 m from the berth. The LP2-a:Berthing PPO agent was able to control the vessel nicely inside the berth-rectangle and hold it there, with a best achieved accuracy of 0.017m from the desired berth pose. This trajectory is illustrated in Figure 4.11 and corresponding video LP2-a:Berthing-PPO. The PPO agent received relatively fast close to optimal rewards, as seen in the reward plots in Figure 4.14. It also holds a sufficient length away from the quay, as it only received small penalties for being close to a obstacle (r_{obs}). This tendency of high accuracy and efficient trajectories was also observed in several other examples, as presented in Table 4.3. The PPO agent was able to achieve high returns generally, in the range of 6500-7300. The maximum theoretical return is 7500, only achievable if a vessel started inside the berth and stayed there the entire episode. This shows that the PPO agent was able to generalise quite well the knowledge of how to berth in several episodes.

On the other hand, the DDPG agent was not able to solve the illustrated test episode as well as the PPO agent. In the test episode, the DDPG agent moves the vessel towards the berth-rectangle, but does not park the vessel inside it, as illustrated in Figure 4.14 and corresponding video LP2-a:Berthing-DDPG. This means that the agent achieves only part of the objectives, which is also reflected on received rewards as seen in the plot of the rewards in Figure 4.14. Generally, the LP2-a:Berthing DDPG agent tried to control the vessel towards the berth, but stopped before the vessel got there. The results from several episodes are presented in Table 4.3, and reveals significant lower rewards in similar scenarios as the PPO agent. In some of the episodes, the DDPG agent crashed when trying to reach the berth, as witnessed by the negative reward.



(a) LP2-*a*:DP using DDPG

(b) LP2-*a*::DP using PPO

Figure 4.11: Trajectory plots for LP2-*a*:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].

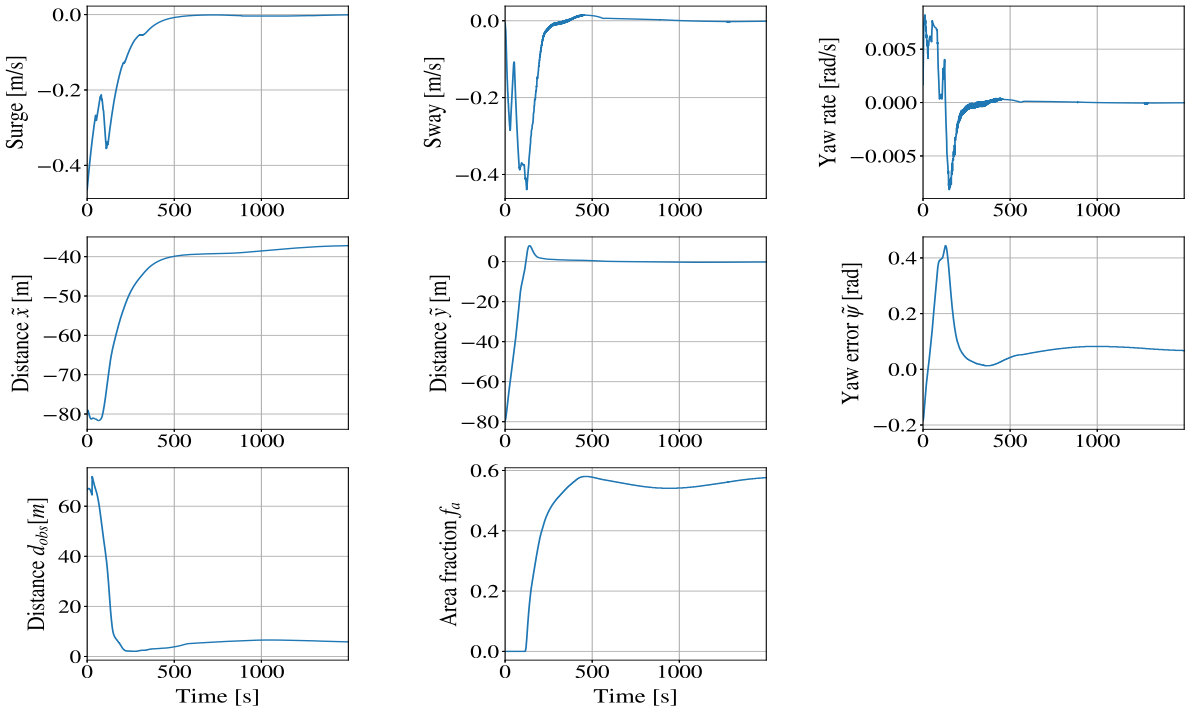
The state plot, in Figure 4.12b, demonstrates that the PPO agent approached the berth efficiently, with the distances to the target $\tilde{x}$ and $\tilde{y}$ and the heading error $\tilde{\psi}$ converging quite fast towards zero and hence the berth. The LP2-a:Berthing PPO agent achieves approximately zero velocity when at berth, but exhibited some small fluctuations, which was the same behaviour as seen for LP1:DP PPO agents. The fluctuations were so small that the vessel did not touch the quay, observed by analysing the distance to the closest obstacle d_{obs}. For the LP2:Berthing agents it should be more important to not deviate from the berth compared to LP1:DP agents, since the LP2:Berthing agents receives a big penalty if touching the quay as compared to LP1:DP. The reason for the fluctuations in the distance to closest obstacle d_{obs}, might be due to how the measurement was made from the closest vertex, which changes rapidly close to land. The PPO agent was though using the thruster forces and angles quite aggressively to reach the desired berth-rectangle, as seen in Figure 4.13, similarly as the LP1:DP agents.

Table 4.3: he results from the LP2-*a*:Berthing on several test episodes. The test episodes are run for 1500 seconds.

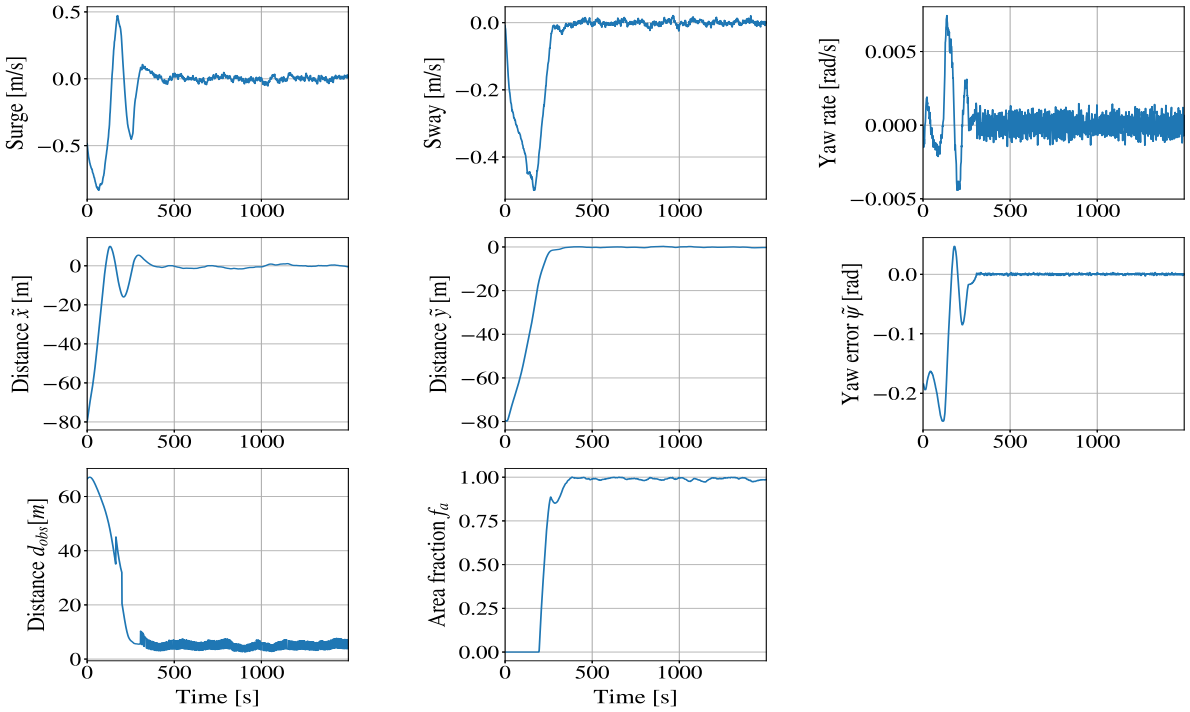
DRL -algorithm	**Initial position** $[x, y, \psi, u, v, r]$ **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Mean absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Episode return**
PPO	900, 570, -0.3,-0.47, 0, 0	0.017, 0.000	9.326, 0.024	6467
DDPG		37.204, 0.001	45.078, 0.086	3565
PPO	861, 546, 0.638,-0.1, 0, 0	0.085, 0.000	6.710, 0.046	6499
DDPG		37.174, 0.012	42.315, 0.106	3630
PPO	805, 584, 0,0, 0, 0	0.049, 0.000	6.328, 0.010	6620
DDPG		11.472, 0.013	23.410, 0.099	5524
PPO	742, 579, 0.01,0.05, 0, 0	0.002, 0.000	6.661, 0.012	6756
DDPG		51.504, 0.000	70.176, 0.648	-592
PPO	782, 529, -0.06,0.40, 0, 0	0.021, 0.000	2.071, 0.003	7255
DDPG		1.066, 0.000	14.722, 0.007	7168

The state plot, in Figure 4.12, of the DDPG agent, shows that the agent stopped the vessel approximately 40 meters away from the berth, and do not converge slowly towards the berth. The plots of thruster forces and angles during the episode are visualised in Figure 4.13. When at berth, the agent seems to be used the azimuth thrusters $[f_1, f_2, \alpha_1, \alpha_2]$ such that they neutralise each other. The force of the tunnel thruster is approximately 0.

Compared to LP1:DP, the reward function for LP2:DP has the additional reward component $r_{d_{dot}}$ for moving the vessel towards the berth. The reward component r_{obs} penalises the agent if going too close to an obstacle, including the quay. This induced a dilemma to the LP2:Berthing agents, as they were told to stay away from the quay (considered an obstacle), while they should reach the berth which is close to the quay. The reward component r_{d_d} was crucial to break this deadlock. This helped the agents to overcome the aversion of the prospective penalty if crashing into the quay. From the reward plot 4.14, one can see that the agent tried to maximise r_{d_d}, while avoiding punishment through reward component for obstacles r_{obs}. This indicates that the intended purpose of luring the agent towards the berth through r_{d_d} works quite well.

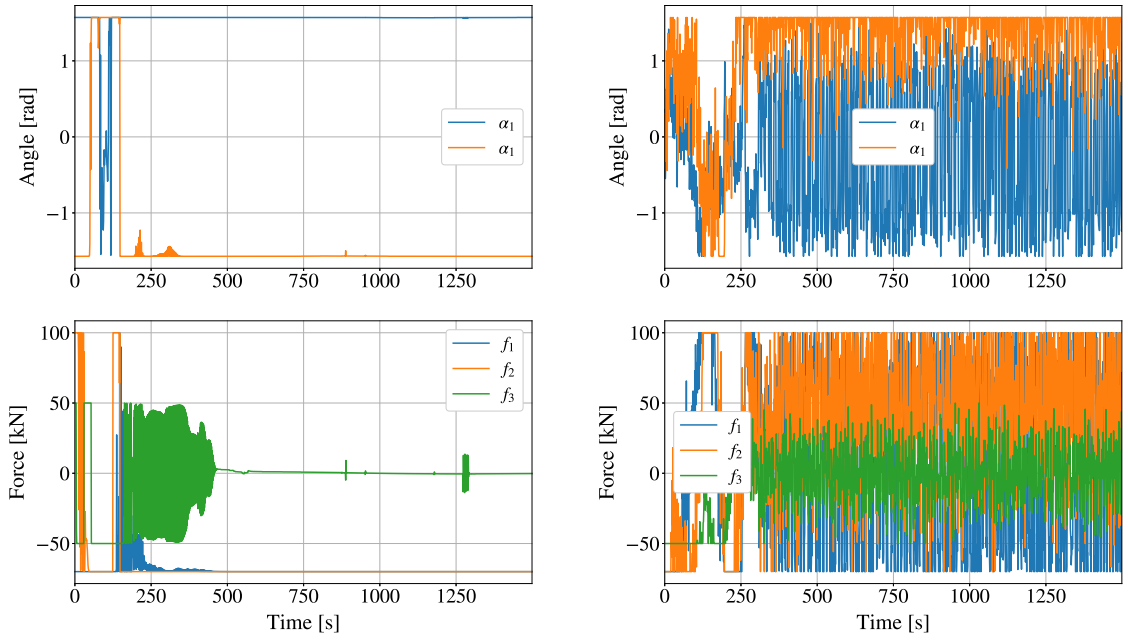


(a) DDPG



(b) PPO

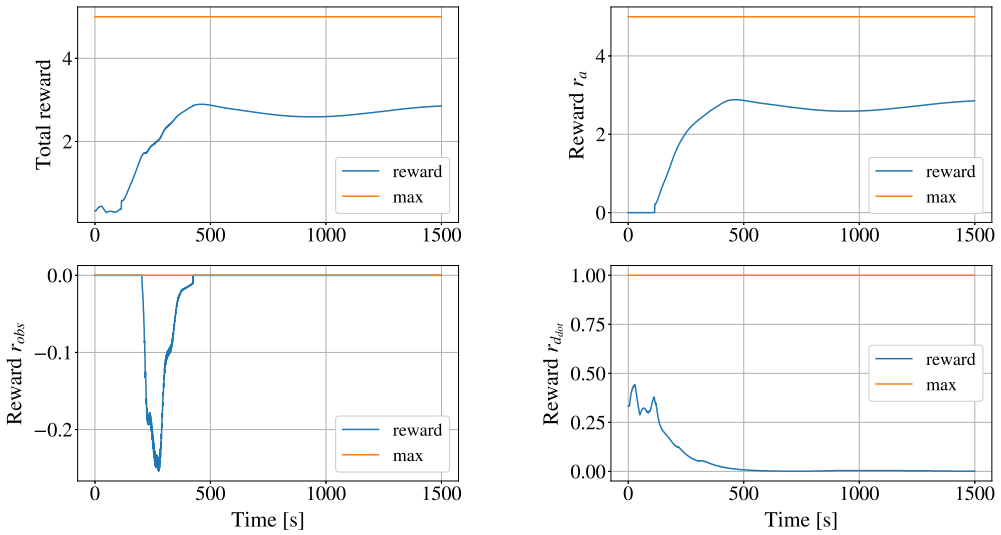
Figure 4.12: State plots for LP2-*a*:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].



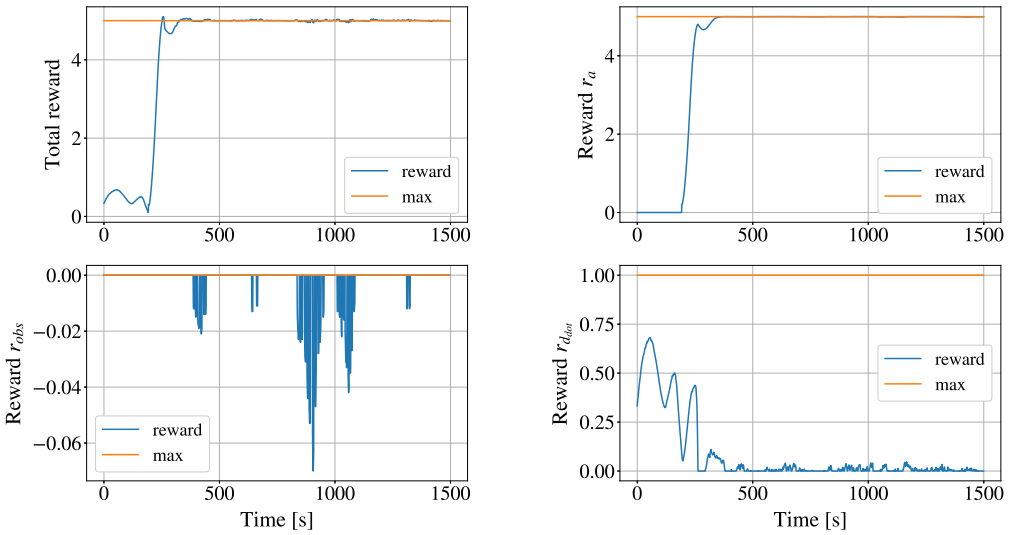
(a) DDPG

(b) PPO

Figure 4.13: Thruster plots for LP2-*a*:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].



(a) DDPG



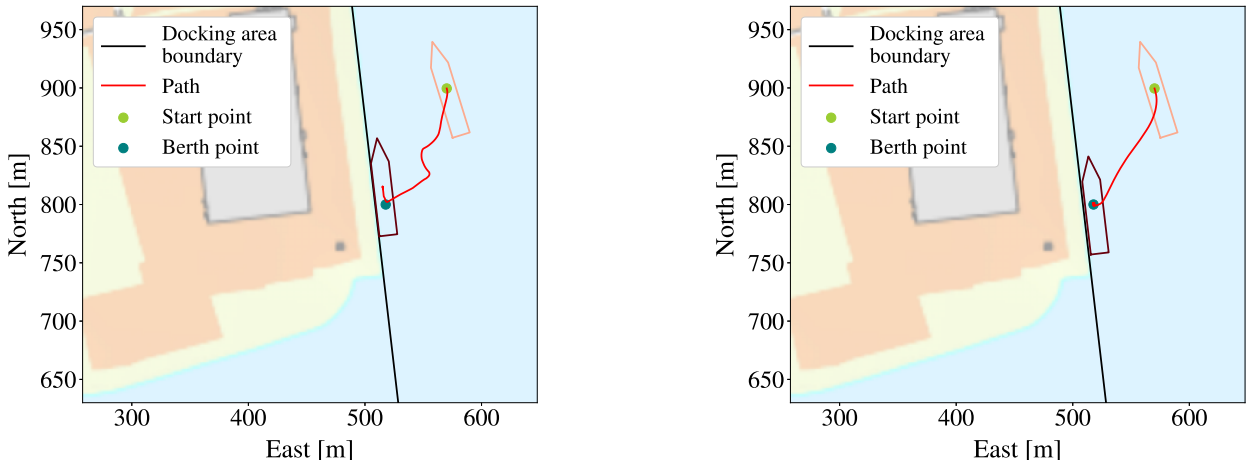
(b) PPO

Figure 4.14: Reward plots for LP2-*a*:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].

4.2.3 Performance of LP2-η:Berthing

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta} = [900m, 570m, -0.3rad]$ and $\boldsymbol{\nu} = [-0.47m/s, 0m/s, 0rad/s]$, the same as for LP2-$a$:Berthing. The PPO agent for LP-η:Berthing was able to control the vessel nicely to the berth and keeping it there, as visualised in the trajectory plot in Figure 4.15b and corresponding video LP2-eta-Berthing-PPO . The PPO agent achieved a best accuracy of 0.014m from the berth, in the illustrated test episode. This is also illustrated in the state plot, in Figure 4.16b, where the heading error $\tilde{\psi}$ and distances to target $\tilde{x}$ and $\tilde{y}$ all converged to zero.

The PPO agent exhibited some small fluctuations in several of the vessels states when trying to keep the vessel at the berth, visualised in Figure 4.16. This seemed to be a tendency for all agents which tried to achieve high accuracy when performing dynamic positioning. It is observed in the plot of the rewards, in Figure 4.18, that the fluctuations do not significantly affect the total achieved reward from the episode. From analysing several cases, it seems that the PPO agent, in general, was able to converge efficiently towards the berth with high accuracy. Numerical examples of the performance of the PPO agent are presented in Table 4.4. The results of several episodes indicate that the PPO agent is able to generalise the knowledge of how to reach the berth quite well.



(a) DDPG (b) PPO

Figure 4.15: Trajectory plots for LP2-η:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].

The LP2-η:Berthing DDPG agent was not able to solve the entire illustrated test episode, a similar result to that of the LP2-a:Berthing DDPG agent. The trajectory is illustrated in Figure 4.15a and corresponding video LP2-eta-Berthing-DDPG. These shows that the vessel appeared to travel towards the berth, but it stopped before reaching it, similarly as for the

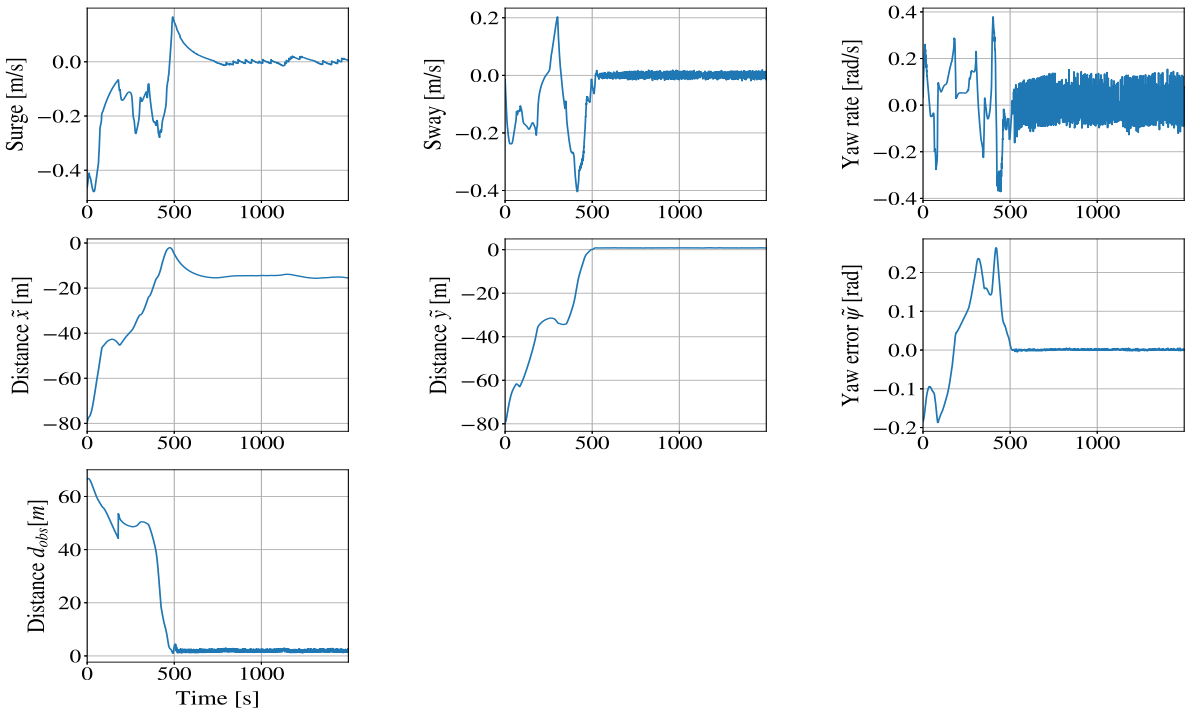
Table 4.4: he results from the LP2-η:Berthing on several test episodes. The test episodes are run for 1500 seconds.

DRL -algorithm	**Initial position** $[x, y, \psi, u, v, r]$ **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Mean absolute** $(d_d, \tilde{\psi})$ **[m,rad]**	**Episode return**
PPO	900, 570, -0.3,-0.47, 0, 0	0.014, 0.000	9.117, 0.015	6637
DDPG		2.423, 0.000	26.702, 0.044	5052
PPO	861, 546, 0.638,-0.1, 0, 0	0.017, 0.000	5.047, 0.034	6617
DDPG		0.839, 0.000	106.179, 0.887	1222
PPO	805, 584, 0,0, 0, 0	0.061, 0.000	7.750, 0.008	6367
DDPG		1.025, 0.000	19.291, 0.033	5931
PPO	742, 579, 0.01,0.05, 0, 0	0.066, 0.000)	8.121, 0.012	6300
DDPG		25.843, 0.077	0.638, 0.000	5495
PPO	782, 529, -0.06,0.40, 0, 0	0.066, 0.000	1.373, 0.004	7342
DDPG		3.776, 0.000	24.646, 0.009	6042

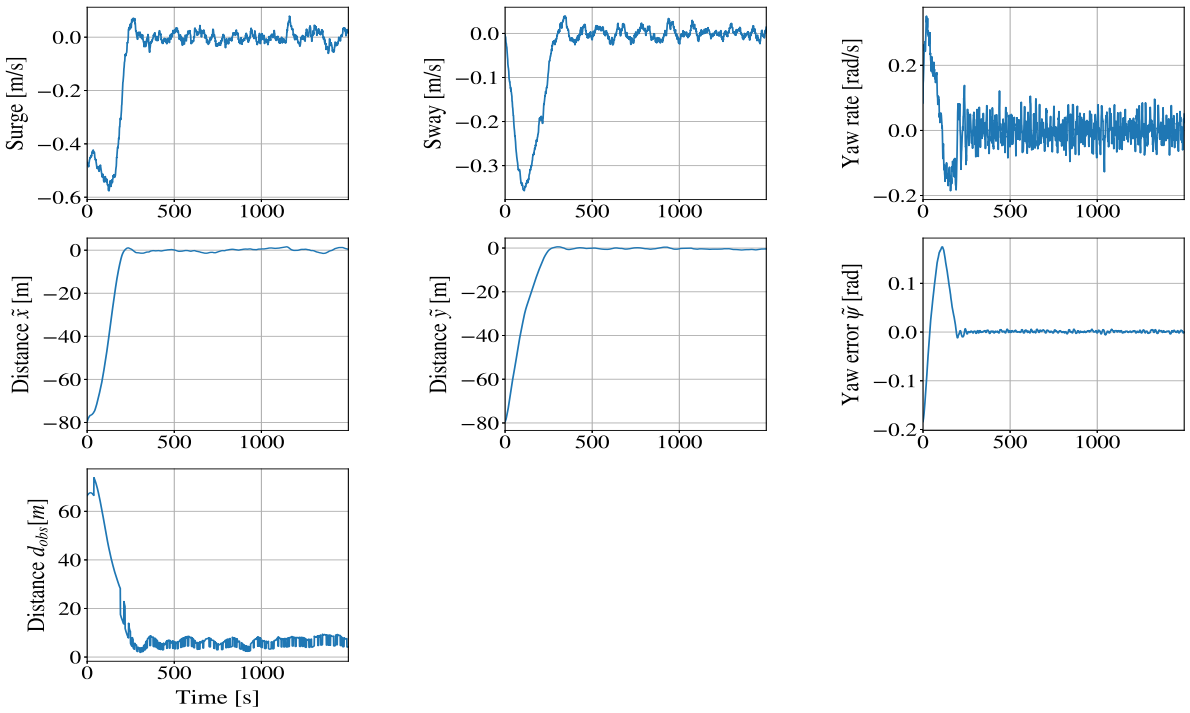
LP2-a:Berthing agent. The state plot, in Figure 4.16a, shows that the vessel have a steady-state error in the distance to the target $\tilde{x}$, as illustrated in Figure 4.16a. The thruster plot shows quite aggressive use, where the agent seems to be alternating between maximums of the thrust, as seen in Figure 4.13a. The resulting force appears to keep the vessel in place, as seen for several of the other agents.

It can be observed that that the LP2-η:Berthing DDPG agent achieved high rewards in the final position despite the apparent distance to the berth, in Figure 4.18. This is the same result as for the LP1-η:DP DDPG agent, where it was also seen that increasing σ_{d_d} of the reward component r_{d_d} was correlated with the agent failing to close in on berth with the desired accuracy. The reward component r_{d_d} is the reward component which encourages the agent to place the vessel to the quay.

In general, it was seen that the DDPG agent performed worse than the PPO agent, as seen in Table 4.4. The DDPG agent did not crash in any of the episodes, but the mean absolute value of distance to the target and heading error are quite high for the DDPG agent. This means the vessel is on average parked quite far from the berth. The PPO agent generally achieved high returns (with a maximum of 7000), and high accuracy from the target.

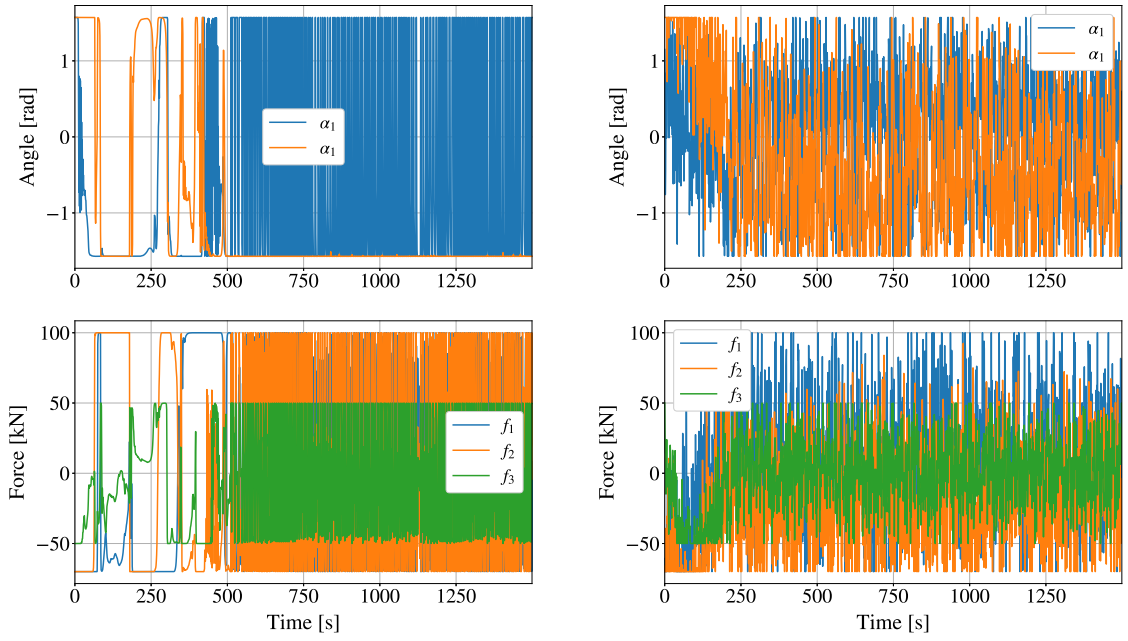


(a) DDPG



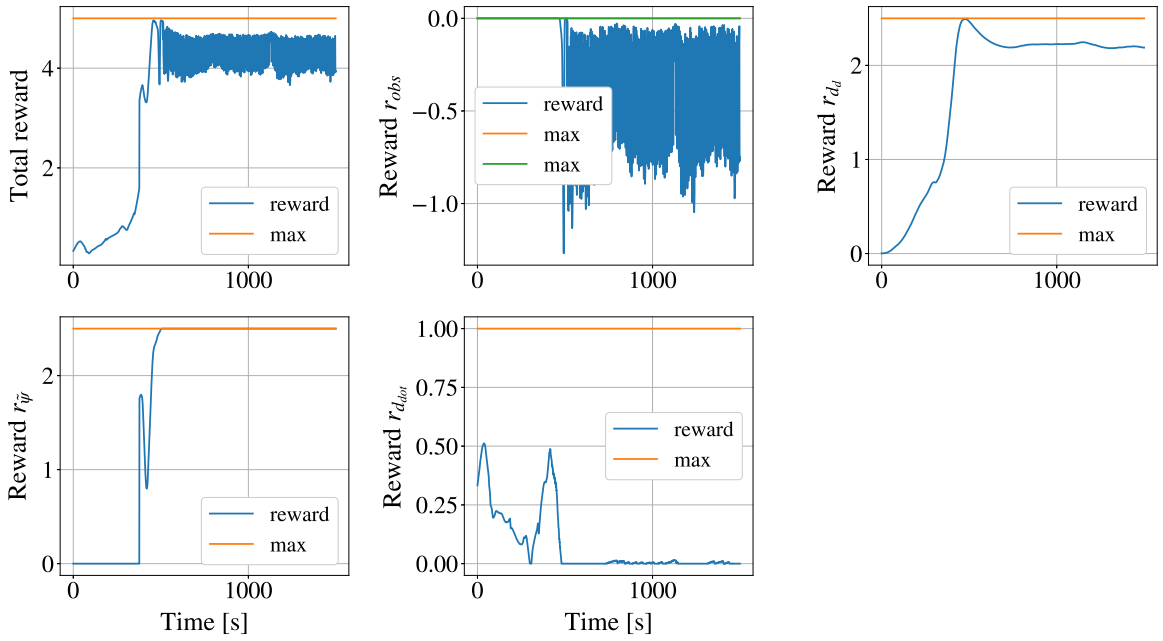
(b) PPO

Figure 4.16: State plots for LP2-η:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].

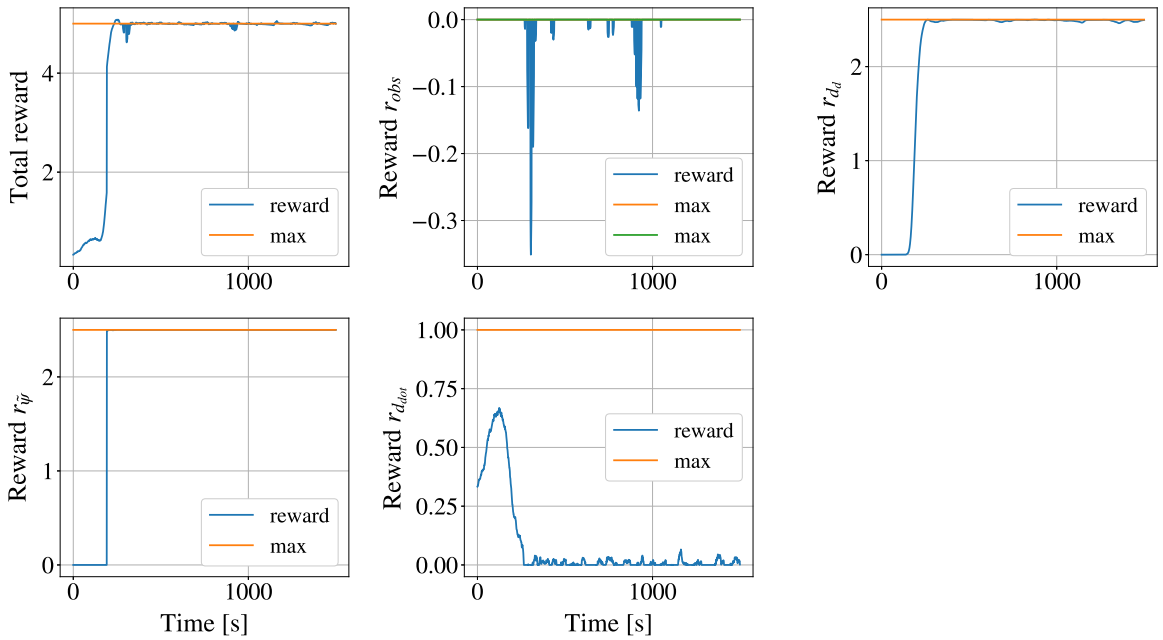


(a) DDPG (b) PPO

Figure 4.17: Thruster plots for LP2-η:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].



(a) DDPG



(b) PPO

Figure 4.18: Reward plots for LP2-η:Berthing agents. Test episodes starts with initial state $\boldsymbol{\eta}$ = [900 m, 570 m, -0.3 rad] and $\boldsymbol{\nu}$ = [-0.47 m/s, 0 m/s, 0 rad/s].

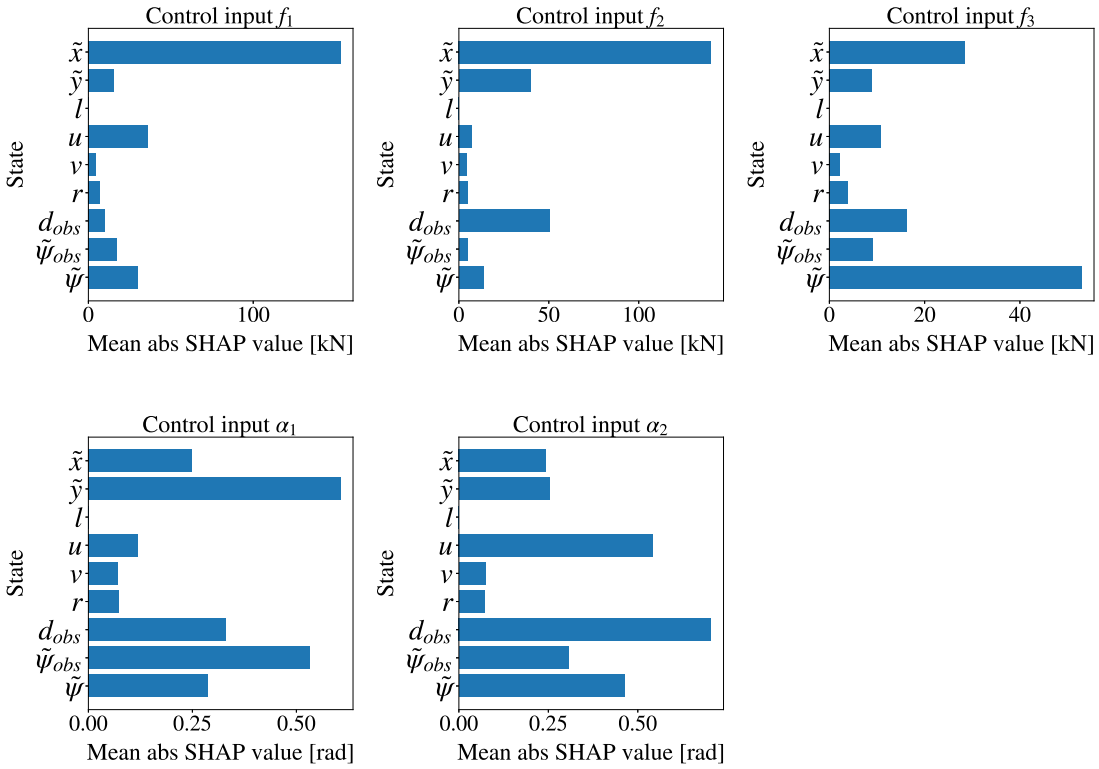
4.2.4 Using SHAP to correct DRL-agent

One of the concerns when adding the quay to the problem was to make sure that the agent could be able to berth both on port and starboard side. Initially, training of the berthing agent was performed on the port side only. Tests of LP2-η:Berthing PPO agent showed that the agent was not able to successfully berth on starboard when trained on port. The question is, why did the agent not work as well on the opposite side?

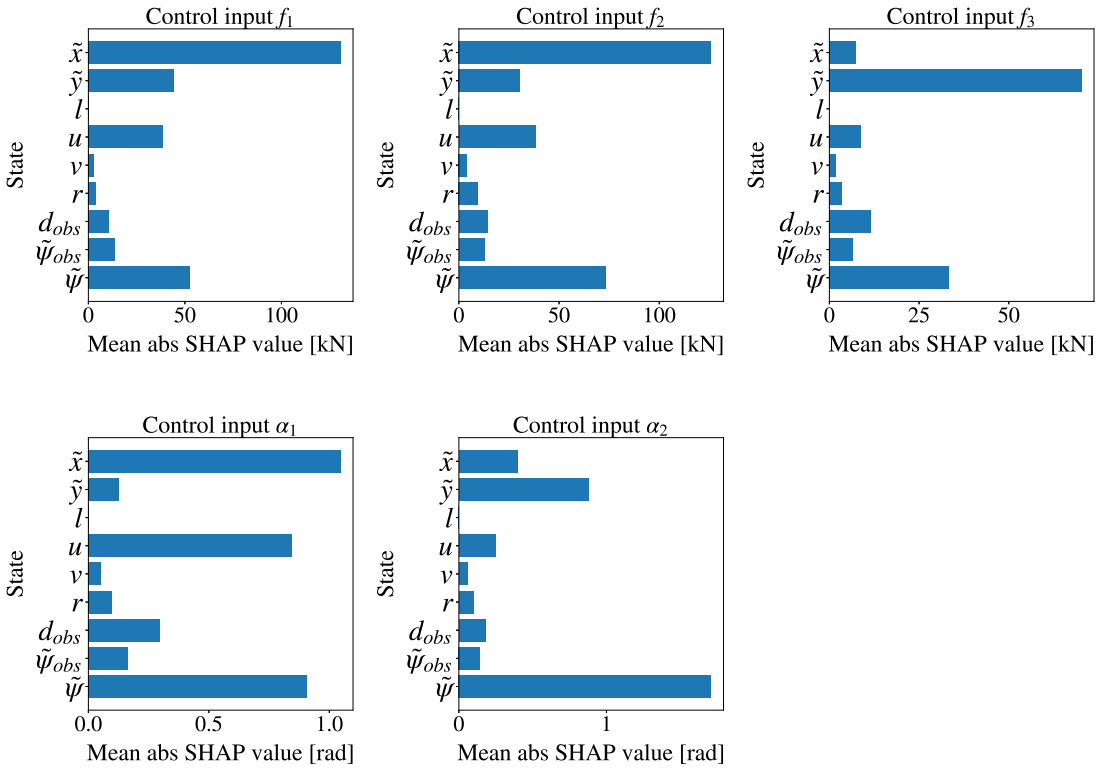
To analyse this question, SHAP values were used. The general contribution of states to the control input (thruster angles and forces), was analysed through the mean absolute SHAP-values of 400 random samples from the berthing environment. For each sample, the SHAP values of the states to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$ was calculated. The accuracy of the explanation model to capture the DRL-agents policy is visualised in Appendix D.1 and Figure D.1 for training using only one side, and for training on two sides in Appendix D.2 and Figure D.2. The valid sets of states for selecting random samples from the environment are listed in Table 3.7.

The mean absolute SHAP values show that the contribution of distance to the obstacle d_{obs} relative to the other states, varied quite a lot between the different thruster forces and angles, in Figure 4.19a. The distance to the obstacle had a relatively higher contribution for the azimuth thrusters (f_1, α_1), which is the thruster closest to the quay when berthing in port side. This indicates that one azimuth thruster was more specialised towards handling the obstacles than the other.

Training the agent to berth on both port and starboard side was therefore tested. The resulting mean absolute SHAP values of the new agent are presented in Figure 4.19b. It shows that the contribution of the distance to the obstacles d_{obs} exhibited similar relative contribution for both the azimuth thruster forces and angles $(f_1, f_2, \alpha_1, \alpha_2)$. This indicates that both the azimuth thruster was more trained to handle obstacles than the other. The new agent demonstrated similar accuracy and efficiency of reaching the berth pose, as the agent trained on only one side.



(a) Training only vessel on the right side of the dock



(b) Training on both sides

Figure 4.19: Mean absolute SHAP values of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$

4.3 LP3: Target tracking

The agent performing target tracking (LP3:Target tracking) has the objective of controlling the vessel to a target point and keep it there. The solution is expected to work within 400 meters from the target point, and with a low start forward velocity in the range of -0.5 to 0.5 m/s. The DRL agent controls the thruster forces and angles $[\alpha_1, \alpha_2, f_1, f_2]$. The target tracking agent was learned through both DDPG and PPO, using the training episodes described in Section 3.3.5.

The reward function in this learning phase is almost the same as for LP2-η:Berthing, minus the reward component $r_{\tilde{\psi}}$ for achieving a desired heading at the berth. The same reward function parameters (minus parameters for $r_{\tilde{\psi}}$) were therefore used , providing good results for the LP3:Target tracking PPO agent. The reward function parameters are expressed in Table 3.3 for PPO and 3.4 for DDPG.

4.3.1 Training progress

The training progress of the DDPG and PPO agents are illustrated in Figure 4.20. The figure shows that the PPO agent converged after approximately 4 million interactions between the environment and the agent, to a steady-state value of approximately 4300 in average return. The maximum receivable reward for one episode is 6000 in return, given in the sole case of when the agent starts at the target position and stays there.

The average return of the DDPG agent did not converge towards sufficiently high value, as the PPO agent, after training with over double as many interactions as the PPO agent. Alternative values of the hyperparameters of the DDPG agent and reward function parameters were tested, but this produced insignificant improvements. Shaping rewards for the DDPG agent using a higher variance of σ_{d_d} of reward component r_{d_d}, which gives points for being close to the target, helped to aid the vessel towards the berth. It did however not help as much as for LP1-η:DP and LP2-η:Berthing, most likely due to larger distances. It could be that the DDPG agent could improve its training progress by trying even more combinations of different hyperparameters, reward parameters, and training longer.

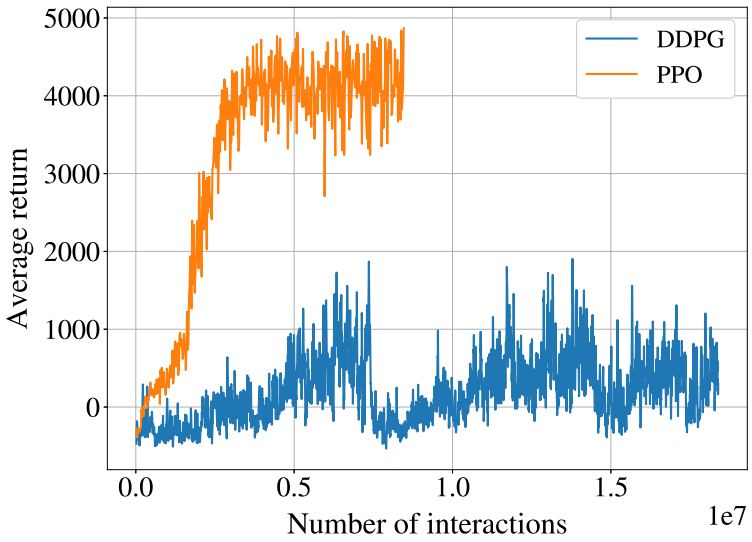


Figure 4.20: The average return during training as a function of number of interactions between agent and environment, for the LP3:Target tracking agents.

4.3.2 Performance

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad]. The trajectory of the PPO agent is illustrated in Figure 4.21b and corresponding video LP3-Target-tracking-PPO. They illustrate that the PPO agent appears to be capable to control the vessel nicely towards the desired target and hold its pose once there. It was able to reach the target with a best achieved accuracy of 0.073 meters.

Figure 4.22 illustrates the state plots of the illustrated test episode for the LP3:Target tracking PPO agent. It shows the same tendency as for the LP1:DP PPO and LP2:Berthing PPO agents, with the vessel efficiently travelling towards the desired target, judging by the gradual descent of distance to target variables $\tilde{x}$ and $\tilde{y}$. From the development of forward velocity, one can see that the agent increased the forward velocity u of the vessel in the beginning, to reach the target quicker. When the vessel came closer to the target, the agent decreased the forward velocity, before stopping totally and holding the position, performing DP. The reward plots 4.24 shows that acceleration of the vessel led the agent receiving a lot of points from reward component $r_{d_{dot}}$, for converging towards the target. When the vessel came sufficiently close to the berth, the agent optimised the reward component r_{d_d}, for getting close to the target position.

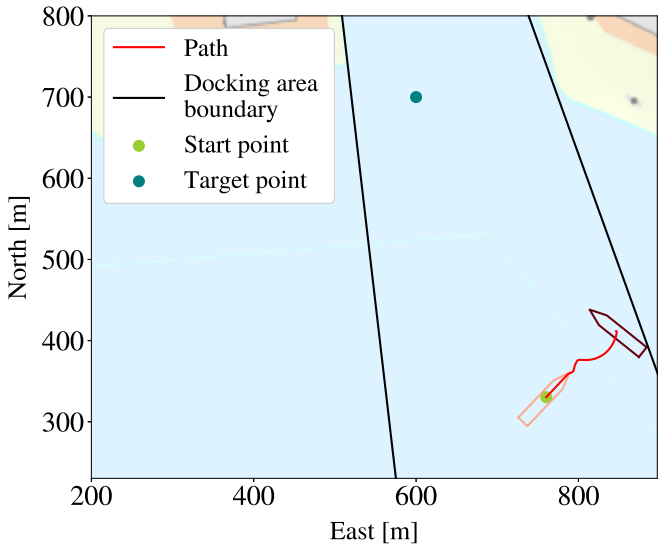
There was small fluctuations in the velocities when the agent tried to hold the vessel at the desired target in the test episode, visualised in Figure 4.22. This was also observed in LP1:DP

and LP2:Berthing. The thruster plots, Figure 4.23, illustrates that the agent was using the thrusters quite aggressively when performing DP. The use of the thruster when the vessel is further away was, however, not as aggressive. This is consistent with the hypothesis that the reason for the very aggressive use of thrusters at short range is the difficulties of holding the vessel at the target and not being penalised for the use of the thrusters.

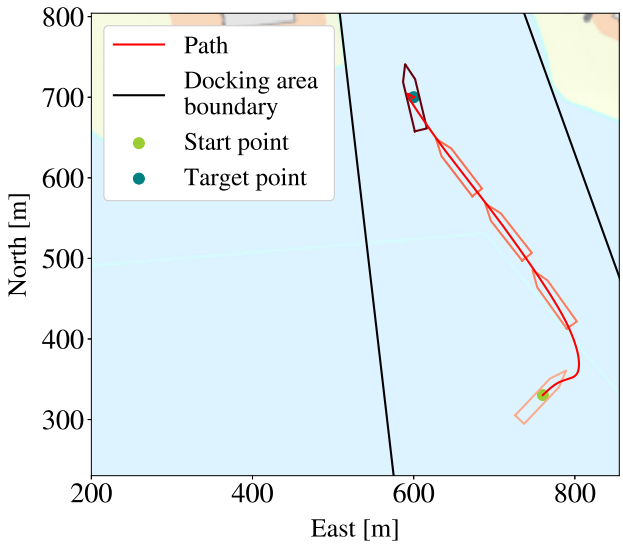
The PPO agent solved the task efficiently in all the episodes tested, as observed in Table 4.5, with a maximum achievable return of 7000. The DDPG agent, however, was not capable of solving the task of approaching the target in the illustrated test episode nor any other episode. The trajectory of the DDPG agent in the test example is illustrated in Figure 4.21a. In all the episodes the DDPG agent seemed to be giving up and appeared to have no idea of how to reach the berth. This is a similar result to that of the LP2:Berthing DDPG agents, with giving up to reach the objective at a certain point. The training progress of the LP3:Target tracking DDPG agent also indicated that the agent did not understand the objective, by the agent not receiving a high average return. The state plots, thruster plots and reward plots of the DDPG agent is shown in the Appendix C.1 in respective Figure C.1, Figure C.3, and Figure C.2.

Table 4.5: he results from the LP3:Target tracking on several test episodes. The test episodes are run for 1500 seconds.s

DRL-algorithm	**Initial position** x, y, ψ, u, v, r **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** d_d, **[m]**	**Mean absolute** d_d **[m]**	**Return**
PPO	[330,760,0.759, 0.5,0,0]	0.073	84.899	5546
DDPG	[330,760,0.759, 0.5,0,0]	382.104	379.093	-86
PPO	[639,604,-0.61,0.413,0,0]	0.134	6.187	6322
DDPG	[639,604,-0.61,0.413,0,0]	17.347	20.392	5975
PPO	[274, 638, 0.33,0.21, 0, 0]	0.072	64.265	5921
DDPG	[274, 638, 0.33,0.21, 0, 0]	383.862	388.292	44
PPO	[517, 700, 0.16,0.19, 0, 0]	0.086	24.166	6393
DDPG	[517, 700, 0.16,0.19, 0, 0]	195.3687	196.368	13
PPO	[950, 605, -0.202,-0.24, 0, 0]	0.015	22.828	6556
DDPG	[950, 605, -0.202,-0.24, 0, 0]	233.728	246.646	16



(a) DDPG



(b) PPO

Figure 4.21: Trajectory plots for LP3:Target tracking agents. Test episodes starts with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad].

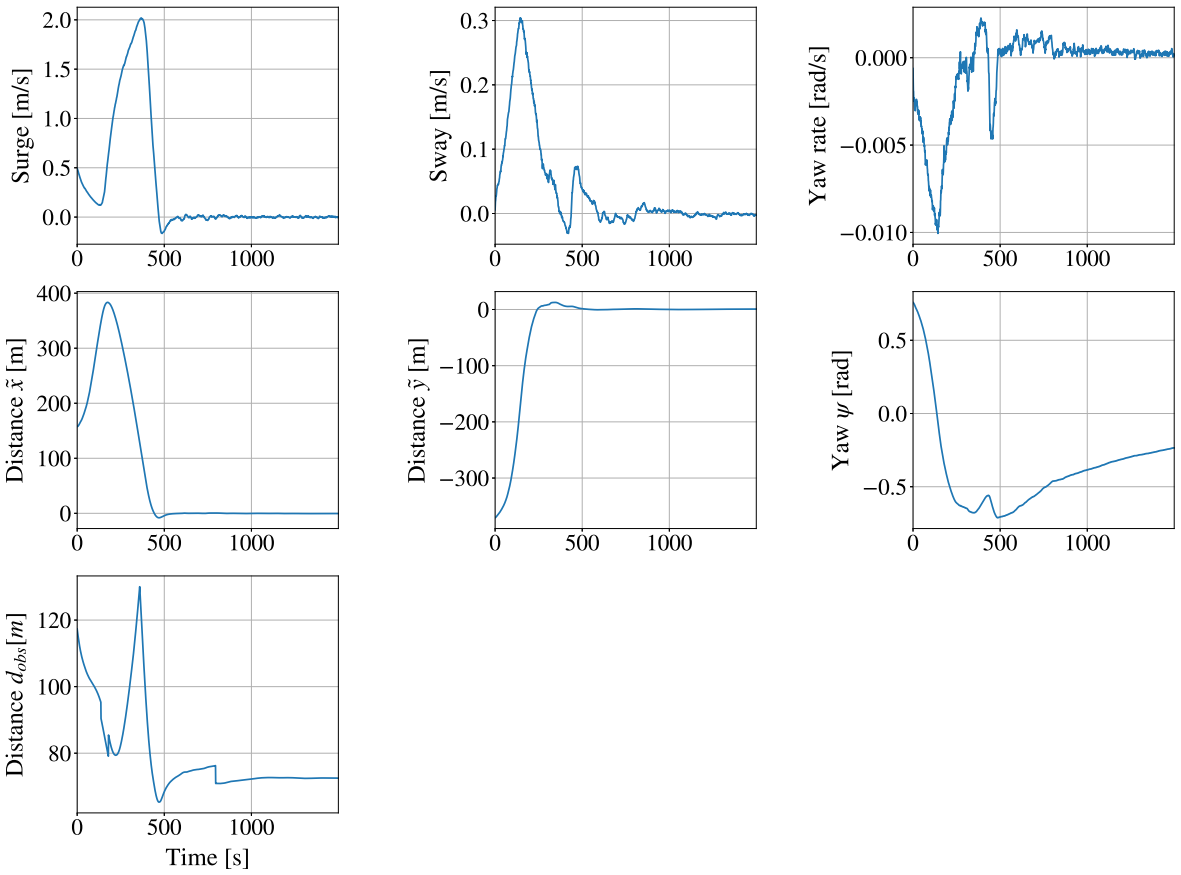


Figure 4.22: State Trajectory plots for LP3:Target tracking PPO agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad].

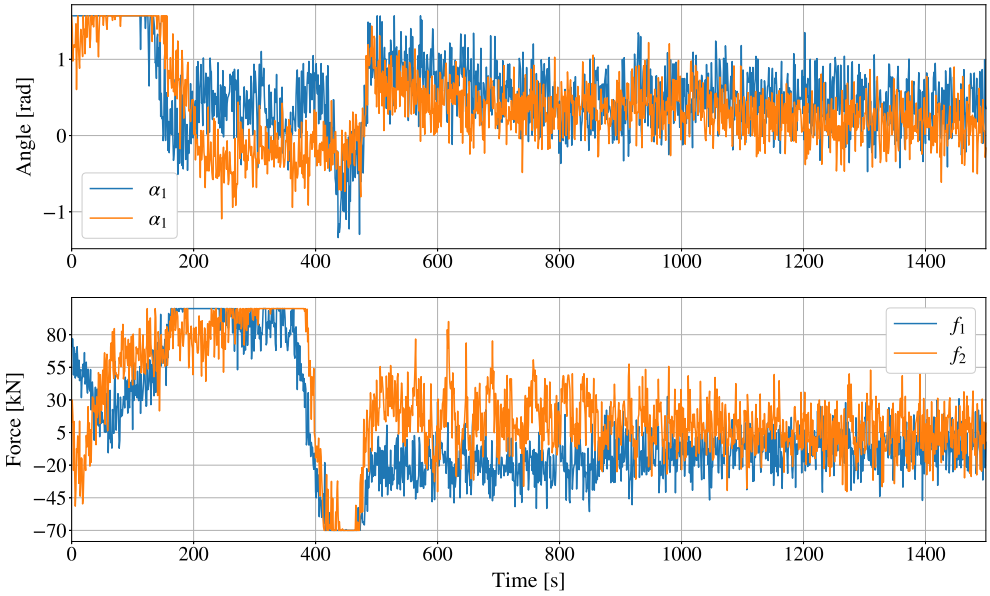


Figure 4.23: Thruster plots for LP3:Target tracking PPO agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad].

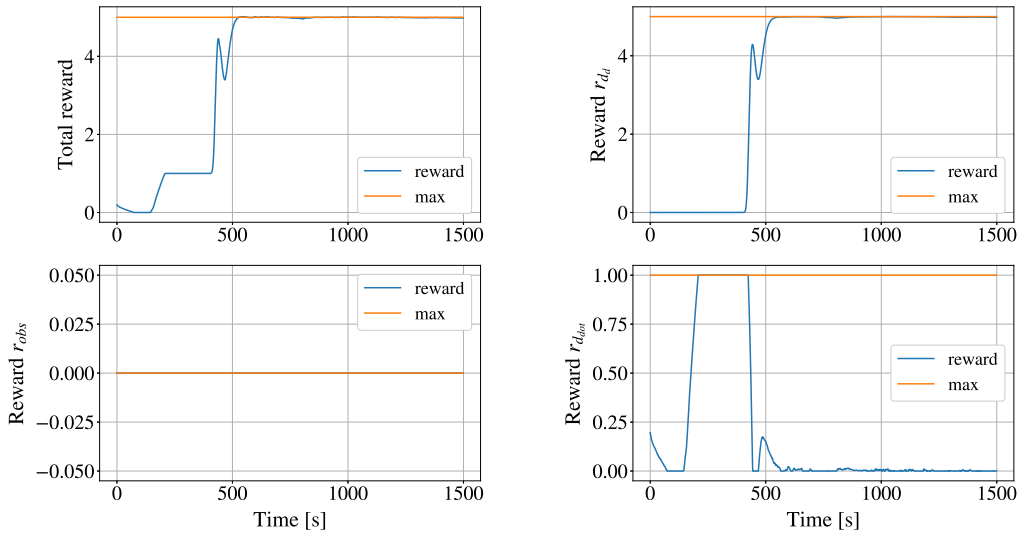


Figure 4.24: Reward plots for LP3:Target tracking PPO agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad]

.

4.4 Comparison of approaches and DRL algorithms in LP1-LP3

In this section the performance of PPO and DDPG in LP1-LP3 will be discussed, and whether the objective of parking the vessel inside a rectangle or at a certain pose worked best.

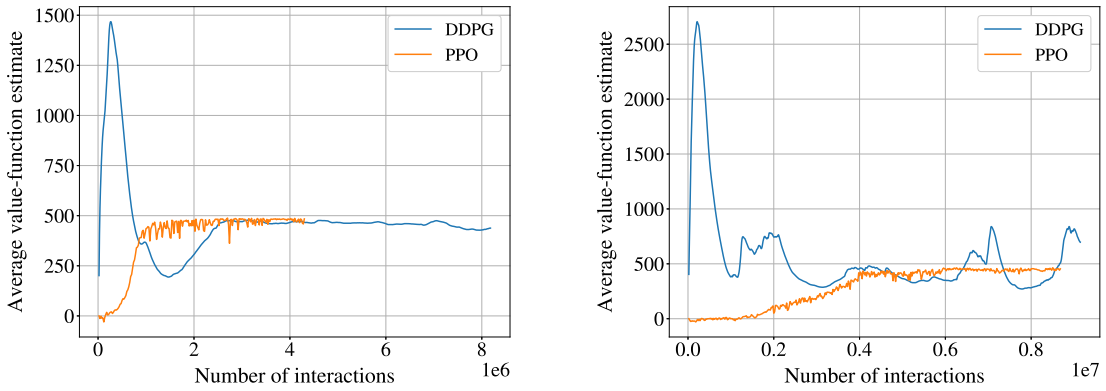
4.4.1 PPO versus DDPG

The results from training of LP1:DP, LP2:Berthing and LP3:Target tracking showed that the PPO agents achieved better than or as good performance as the DDPG agents, with less number of interactions between the agent and the environment. There can be several reasons as to why PPO performed better in these learning phases. There are many differences between PPO and DDPG, such as the exploration policies, value functions, off-policy vs on-policy and sample efficiency. Several studies compare the performance of PPO and DDPG [130–132]. They show that there exist scenarios where one is easier to use than the other and vice versa.

The DDPG algorithm was one of the first DRL-algorithms to deal with complex continuous problems and is shown to perform well in several tasks of controlling marine vessels [4, 5, 62]. One of the common failure modes of DDPG, is an overestimation of the value function [133, 134]. This means that the approximated Q-function overestimates the value of taking a specific action. This estimate is further used to update the estimate of a subsequent state, which means that the error is accumulated for each update. The accumulated error can cause arbitrarily bad states to be estimated as high values, resulting in suboptimal policy updates and divergent behaviour. This is solved in the twin delayed deep deterministic policy gradient algorithm (TD3) [134], which is trying to improve upon DDPG. Overestimation of value estimate is not a problem in PPO.

From the learning phases of LP1-LP3, it appears that the DDPG agents all reached lower average returns than the PPO agents during training, visualised for LP1:DP in Figure 4.1, LP2:Berthing in Figure 4.10, and LP3:Target in Figure 4.20. It was observed that the estimation of value function was growing extremely fast at the beginning for all the DDPG-learning phases, with an example of LP1-a:DP and LP2-a:Berthing in Figure 4.25. From Figure 4.25 it seems like the LP1-a:DP agent was able to an okay value function estimate after the overshoot, while LP2-a:Berthing was not able to do it. The overshoot is the most probable cause to why all of them DDPG agents was not able to reach as good solutions as the PPO agents.

There are several potential reasons for the overestimation. It has for instance been demon-



(a) LP1-a:DP (b) LP1-η:DP

Figure 4.25: The average value-function estimate during training as a function of number of interactions between agent and environment, for LP1-a:DP PPO agent and LP2-a:Berthing agent.

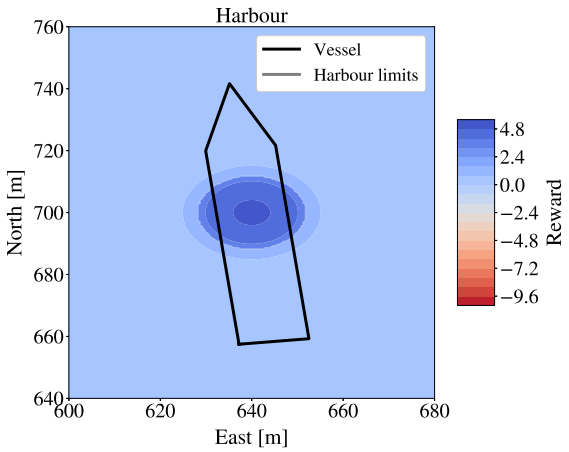
strated that the DDPG algorithm is brittle and sensitive to hyperparameters [130–133, 135]. Studies comparing the performance of PPO and DDPG demonstrates that there are certain scenarios where it is easier to tune hyperparameters of one method over another, to achieve good results [130–132]. It was therefore experimented with different settings of hyperparameters in this thesis, such as learning rate, the number of neurons in the hidden layer, batch size. It was, however, easier finding good parameters for PPO than for DDPG, that worked for all the learning phases.

The performance of the DDPG-algorithm is also sensitive to reward shaping and scaling [1, 136]. During training, it was observed that by increasing the variance from $\sigma_{d_d} = 10$ to $\sigma_{d_d} = 30$, of reward component r_{d_d} for being close to the target, the LP1:DP DDPG and LP2:Berthing DDPG agents learned faster, but with less accuracy from the target due to less differentiation in reward between close to target and at the target. When analysing the reward distributed over a map, Figure 4.26, it is observed a huge difference between the approaches considering the area of the vessel inside a berth-rectangle and the pose of the vessel related to a berth pose, with $\sigma_{d_d} = 10$. The LP1-a:DP is less sparse than LP1-η:DP with $\sigma_{d_d} = 10$ and LP2-η:Berthing with $\sigma_{d_d} = 10$ are more sparse than LP2-a:Berthing. The PPO agent was trained $\sigma_{d_d} = 10$, and the reason why PPO agent learned with more sparse reward functions, is most likely due to being more sample efficient than DDPG.

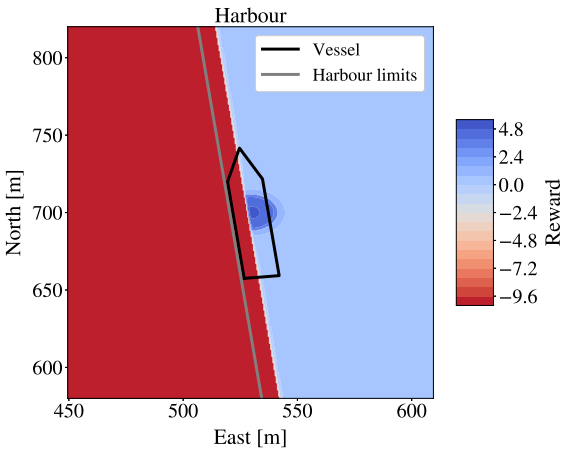
Peterson et al. [131] showed that the performance of the DDPG algorithm also depends on different properties of the environment. They showed, for instance, that in more unstable environments, the DDPG algorithm ended up with results where the solution only tried to stay alive, where failures were punished hard. They believe this is because DDPG is an off-policy method, where random exploration noise can cause sudden failures in unstable

environments. Parallels can be drawn to some elements of the docking environment, where the agent risks receiving a huge penalty within a relatively small constrained area if crashing into the quay/obstacles.

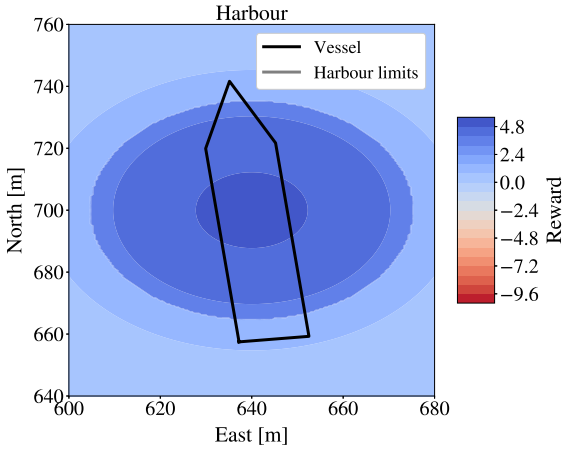
The conclusion is that with further tuning of the DDPG agents and longer training time (due to PPO being more sample efficient), it could improve the learning of the DDPG agents in LP1-Lp3. The PPO agent was experienced to be simpler to tune and achieved good performance faster, and therefore was the DRL-algorithm considered in the following learning phases.



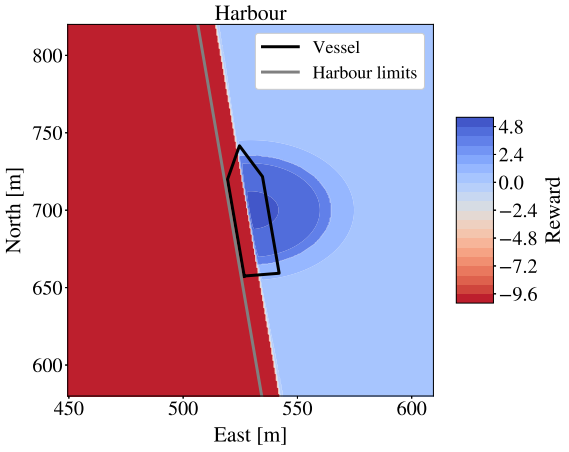
(a) LP1-η:DP: $r_{d_d} + r_{\tilde{\psi}}$, with $\sigma_{d_d} = 10$



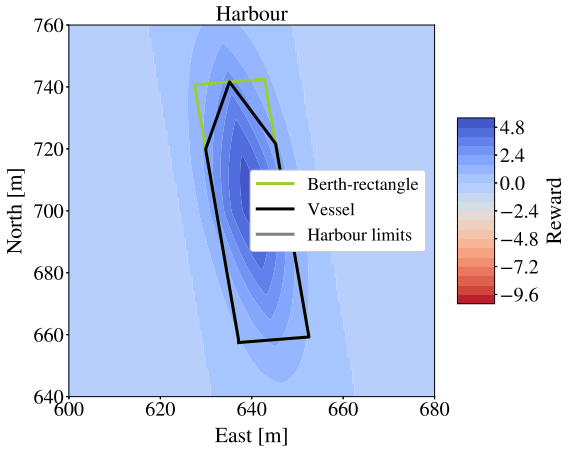
(b) LP2-η:Berthing: $r_{d_d} + r_{\tilde{\psi}}$, with $\sigma_{d_d} = 10$



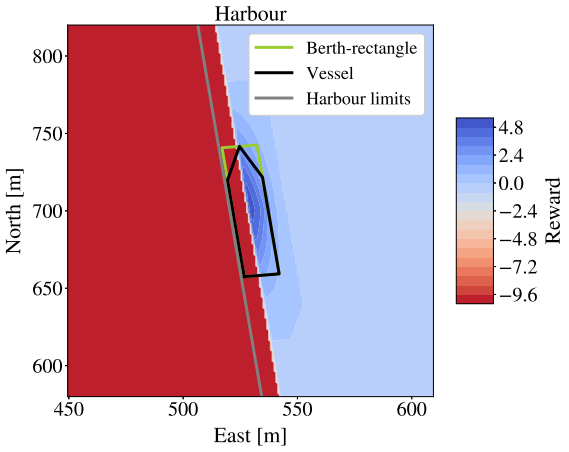
(c) LP1-η:DP: $r_{d_d} + r_{\tilde{\psi}}$, with $\sigma_{d_d} = 30$



(d) LP2-η:Berthing: $r_{d_d} + r_{\tilde{\psi}}$, with $\sigma_{d_d} = 10$



(e) LP1-a:DP: r_{f_a}, with $\sigma_{f_a} = 0.4$



(f) LP2-a:Berthing: r_{f_a}, with $\sigma_{f_a} = 0.4$

Figure 4.26: Reward components for reaching target pose or target-rectangle, for LP1:DP and LP2:Berthing.

4.4.2 Parking a vessel inside a rectangle versus at a pose

Two different objectives were considered In LP1:DP and LP2:Berthing, either parking a vessel inside a target-rectangle or parking the vessel at a target pose. These objectives are functionally speaking similar since the target rectangle is placed tightly around the vessel at berth pose. The objectives should result in the same goal if solved optimally. The properties were investigated and discussed using the PPO-agent, which proved to converge to stable solutions in all the learning phases.

The LP1-a:DP, LP1-η:DP,LP2-a:Berthing and LP2-η:Berthing PPO agents was able to control the vessel quite efficiently and with similar best achieved accuracy from target. This can be observed for LP1-a:DP in Table 4.1, for LP1-η:DP in Table 4.2, for LP2-a:Berthing in Table 4.3 and for LP2-η:Berthing in Table 4.4.

One thing to note is that with the berth-rectangle solution, a small tip of the rectangle of the vessel shape could end outside the berth rectangle. This was as discussed earlier, due to the small contribution of the triangle shape to the entire area. If the objective is to reach the desired pose accurately using the area of the vessel as a measurement, one may opt for a design using a berth slot with the same shape as the vessel. This should make the agent less biased towards having the tip outside. Another possibility would be to use another reward function, which is more harsh towards having the tip of the vessel outside the berth rectangle.

The approach of placing a vessel inside a rectangle is perhaps more appropriate if the objective is not to park at a specific pose but within an area larger than the shape of the vessel itself. For instance, when parking a car at a specific parking slot, the driver would most likely only care to get his car somewhere inside the designated parking space, and not exactly at the centre. In this case, the agent could select the most fitting placement within the rectangle based on some specific criteria.

A challenging property of the berth-rectangle approach is the additional computational costs incurred. This is due to the additional mathematics needed to calculate the area of the vessel inside a rectangle, for each interaction between the environment an agent. This adds up to a considerable amount of computations during training, with 6-8 million interactions.

The objective of this thesis was to park the vessel at a specific pose. There was little difference between placing the vessel inside a berth rectangle or at a specific pose in terms of best achieved accuracy and achieving efficient trajectories. It was, however, experienced a bias when considering the berth-rectangle, of having the tip outside the rectangle. It also increased the computational costs, when considering the target to be a berth-rectangle. It was hence

concluded that in this thesis, the approach used in the following steps will be placing the vessel at a specific pose.

4.5 LP4:Distance berthing

The agent performing berthing from longer distances (LP4:Distance berthing) had the objective of controlling a vessel to a berth, within a range of 400 meters from the target, with a start forward velocity in the range of -0.5 to 0.5 m/s. The agent performed this by controlling the thruster forces and angles $[\alpha_1, \alpha_2, f_1, f_2, f_3]$. The LP4:Distance berthing agent was trained with PPO, using the training episodes described in Section 3.3.5. The reward function parameters are shown in Table 3.3, and are the same parameters as used for the LP2:Berthing PPO agent.

4.5.1 Training progress

The training progress of the LP4:Distance berthing agent is illustrated in Figure 4.27. The figure shows that the average return of the agent converged after approximately 6 million interactions between the environment and the agent, to a value of approximately 4000. The maximum received reward for one episode was 6000, and only achievable if the vessel started at the berth pose and stayed there.

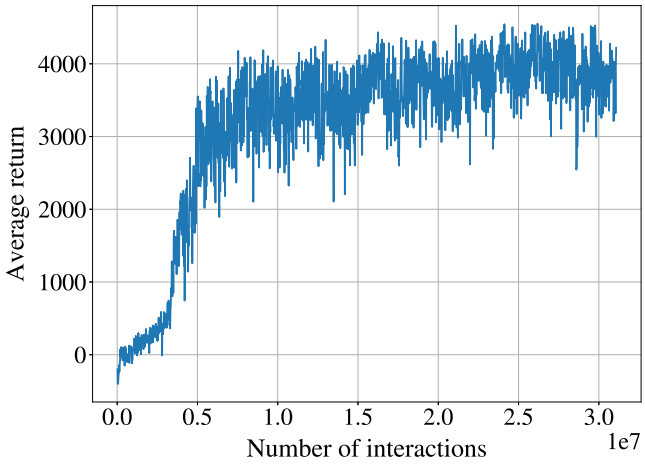
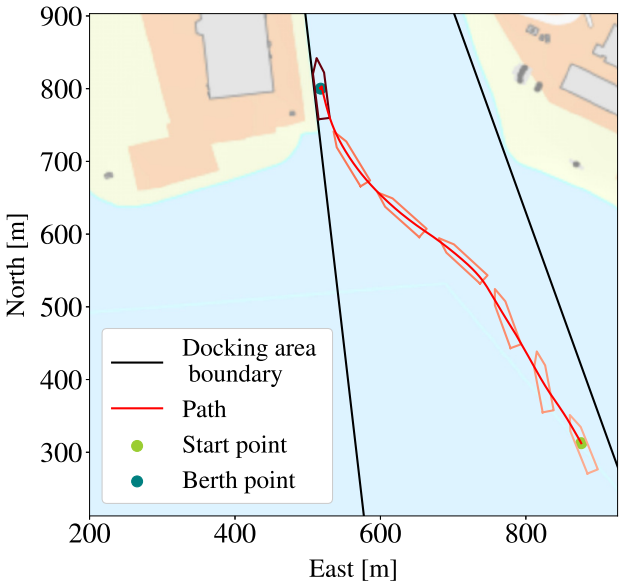


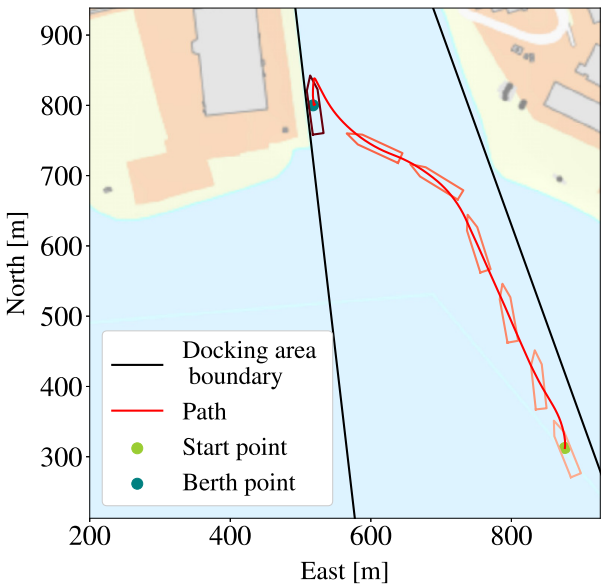
Figure 4.27: The average return during training as a function of number of interactions between agent and environment, for the LP4:Distance berthing agents

4.5.2 Performance

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta}$ =[312.183 m, 876.256 m, -0.384 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s]. The trajectory of the vessel without a constant ocean current is visualised in Figure 4.28a and corresponding video LP4-Distance-berthing-PPO. It shows that the PPO agent controlled the vessel nicely towards the berth and kept it there, with a best achieved accuracy of 0.25 meters from the berth.



(a) No ocean current



(b) Ocean current

Figure 4.28: Trajectory plots for LP4:Distance berthing agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[312.183 m, 876.256 m, -0.384 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].g

The agent was also tested with a constant ocean current, to check the robustness of the method. With a constant ocean current, the achieved accuracy to the target was in some scenarios lower than without the current. A steady-state error compensation for the distances to the target $\tilde{x}$ and $\tilde{y}$ was therefore added, with parameters described in Table 3.6, found through trial and error. In the illustrated test example, the agent was able to control the vessel towards the berth, with a best achieved accuracy of 1.178 m from the target, with steady-state error compensation. Without the steady-state error compensation, the agent was also able to control the vessel towards the berth, but with a best achieved accuracy of only 5.75 m from the target.

The trajectory of the vessel with current and steady-state error compensation is visualised in Figure 4.28, with a corresponding video LP4-Distance-berthing-PPO-current. With the added ocean current, the agent had a different trajectory than with no current, especially when close to the berth. When close to the berth the vessel seemed to be drifting away a bit, before converging towards the target pose. The drifting is better visualised in the development of the states, such as the velocities, in Figure 4.29. The state plots show, for instance, that during transit, both with and without the current, the agent increases the vessel's forward velocity to reach the target faster. When in the proximity of the berth, the agent slows the vessels forward velocity. Being trained without the ocean current, the agent appears to miss the influence of current on the vessel both in the forward and sideways direction, and passes the berth at first, before driving the vessel back into the berth correctly.

The state plots is illustrated in in Figure 4.29. It shows that the distances to the target $\tilde{x}$ and $\tilde{y}$ and heading error $\tilde{\psi}$ converged quite effectively to near zero, both with and without current. The vessel also maintained a decent distance from the berth, both with and without current. These achievements were also reflected by the received rewards, which are visualised in figure 4.30. The received rewards indicated that without the current, the agent was quite successful in achieving maximum rewards. With a current, it is observed that the agent does not achieve as good rewards, but are fairly close due to the steady-state error compensation.

The agents use of thrusters are visualised in Figure 4.40. The agent appears to have used the thruster forces and angles more aggressively when controlling the vessel without the ocean current than with a current close to the berth. The reduction of the use of thrusters is most likely due to the additional steady-state error compensation when testing with the current. The steady-state error compensation makes the agent observe an offset from the vessels actual distances to the target. It, therefore, appears that the agent, tried harder to hold the target pose when it thought it was closer to the berth than further away.

Similarly to LP3:Target approach, it was observed that the agent exhibited less aggressive use of the thrusters when approaching the berth than when the vessel was close to the berth.

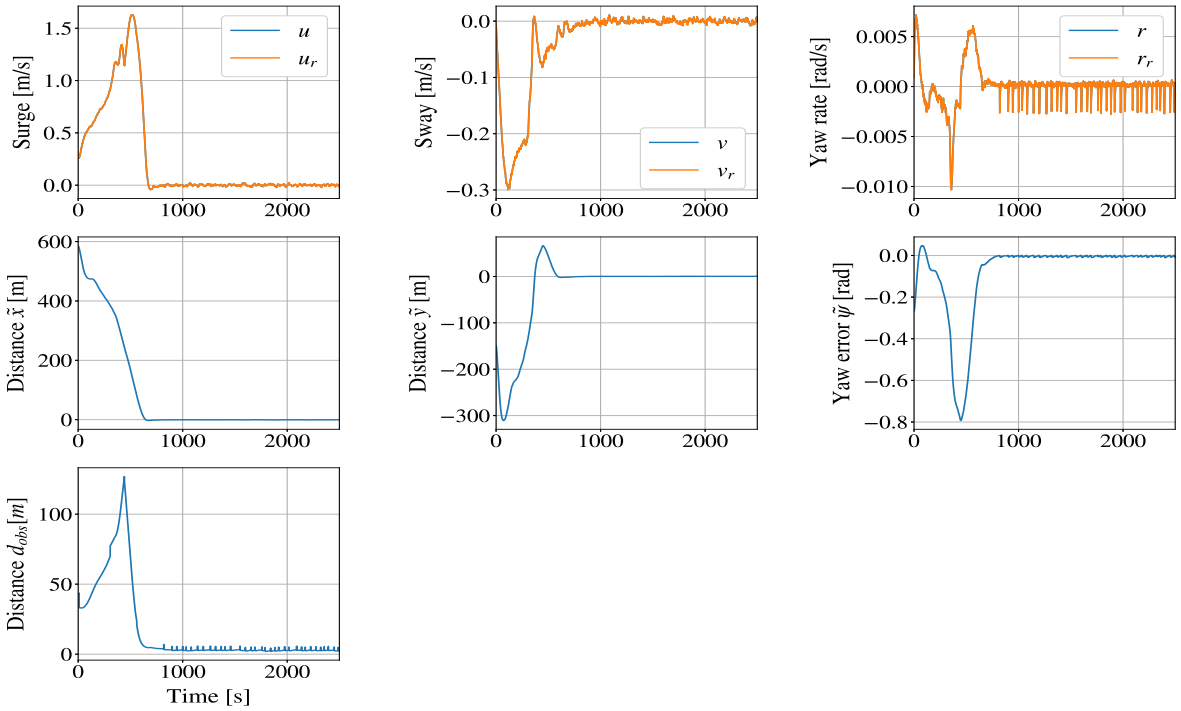
Table 4.6: he results from the LP4:Distance berthing on several test episodes. The test episodes are run for 1500 seconds.

Current	**Initial position** x, y, ψ, u, v, r **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** d_d, **[m]**	**Mean absolute** d_d **[m]**	**Return**
Yes	312.183, 876.256, -0.384	1.178, 0.000	118.771, 0.127	8198
No		0.252, 0.000	(91.606, 0.080	9786
Yes	1141, 487, 0.101,-0.280,0, 0	1.958, 0.000	36.696, 0.057	9892
No		(0.162, 0.000)	(24.978, 0.015	10955
Yes	632,688, 2.40, 0.05, 0, 0	2.618, 0.000	27.832, 0.156	9182
No		0.795, 0.000	17.252, 0.081	11239
yes	966 542 -0.583,-0.35,0, 0	1.783, 0.000	14.216, 0.049	10450
No		(0.228, 0.000)	(13.081, 0.083)	10291
yes	881, 611, -2.340,-0.33, 0.,0	5.956, 0.000	71.696, 0.202	582
No		0.715, 0.000	20.876, 0.093	10215

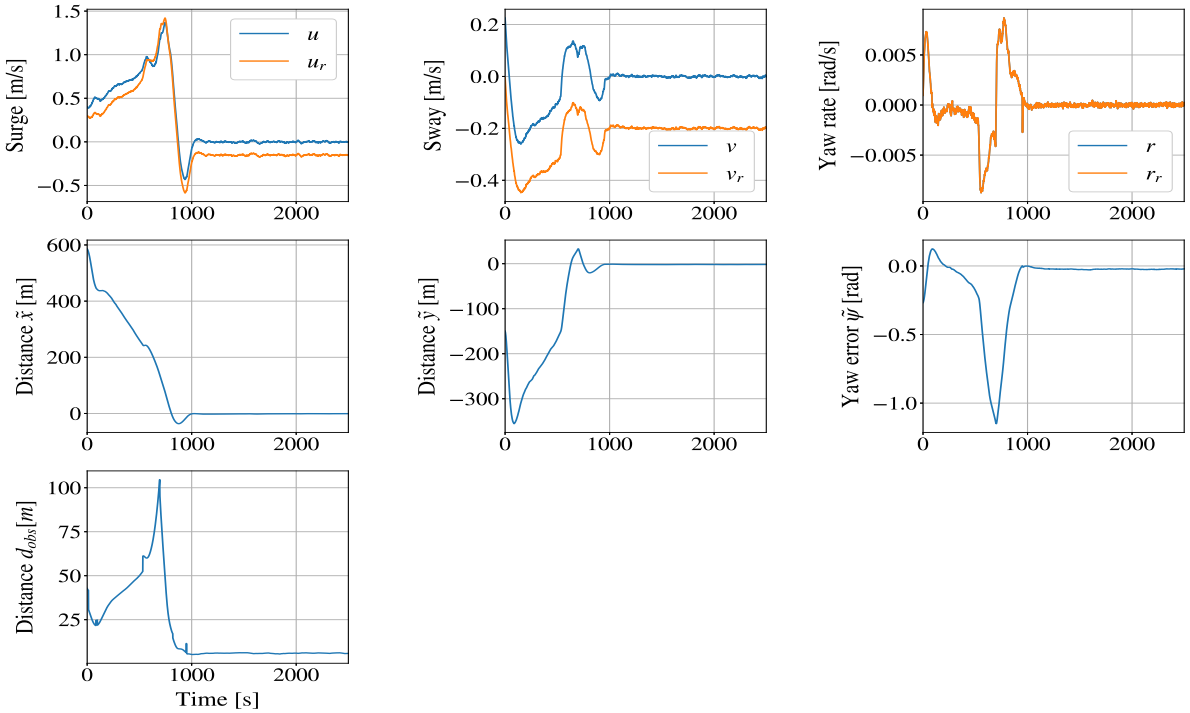
This is consistent with less varied thruster usage needed when only controlling the vessel in open water at higher speeds.

The results from testing the agent in several episodes, with and without current, are found in Table 4.6. The results show that the agent tested without ocean current was able to solve the task in all the tested episodes, with high accuracy in reaching the target. When adding the current, the vessel had some problems in some test episodes with making contact with the quay at slow speeds, even with steady-state error compensation engaged. The agent experienced problems with current, when started close to the berth, or at the final stages of berthing. The first scenario is not easily solved since at low start velocities and with a current, controlling a vessel away from the quay is challenging. An example of the second scenario is illustrated in Figure 4.32. It seems as if the agent at first makes an successful approach towards the berth, but when suddenly reconsiders the situation as too dangerous (aka risk of negative return), and turns around, before crashing. Possible solutions of how to avoid this could be allowing the agent for instance be not penalising the agent if combing in contact with the quay at low velocities. This is further discussed in Section 5.

One thing to consider is that the constant ocean current was set at 0.25 m/s, which is the worst-case scenario at Trondheim harbour (generally between 1-12 cm/s). When testing with lower currents (such as 0.1 m/s), the agent did not make contact with obstacles in any of the tested episodes.

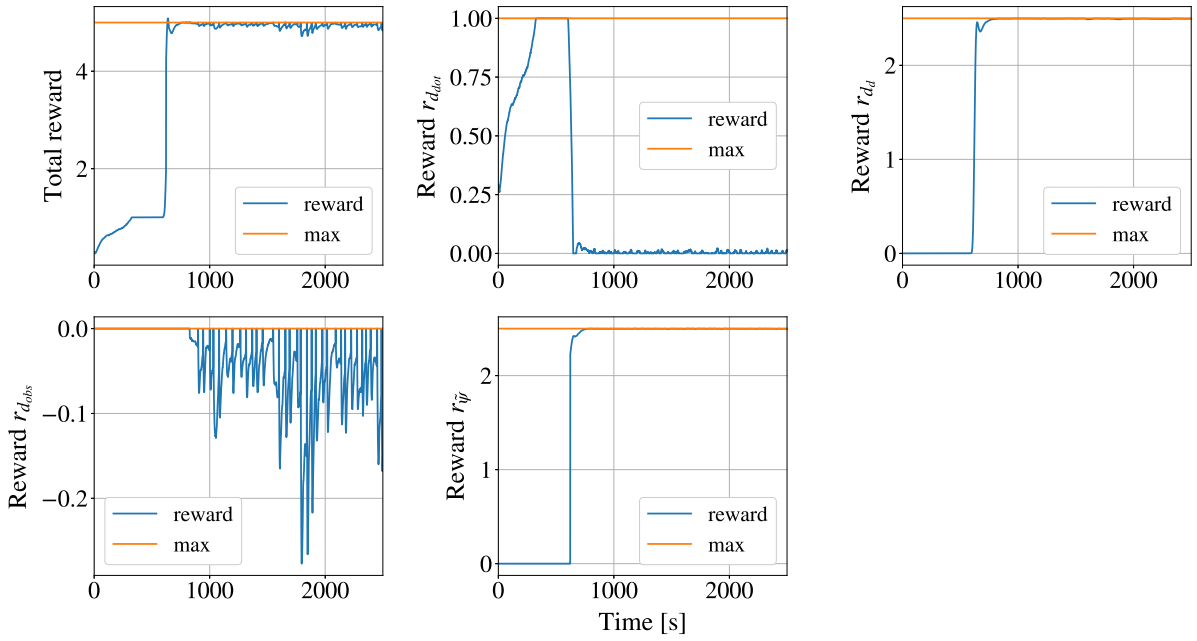


(a) No ocean current

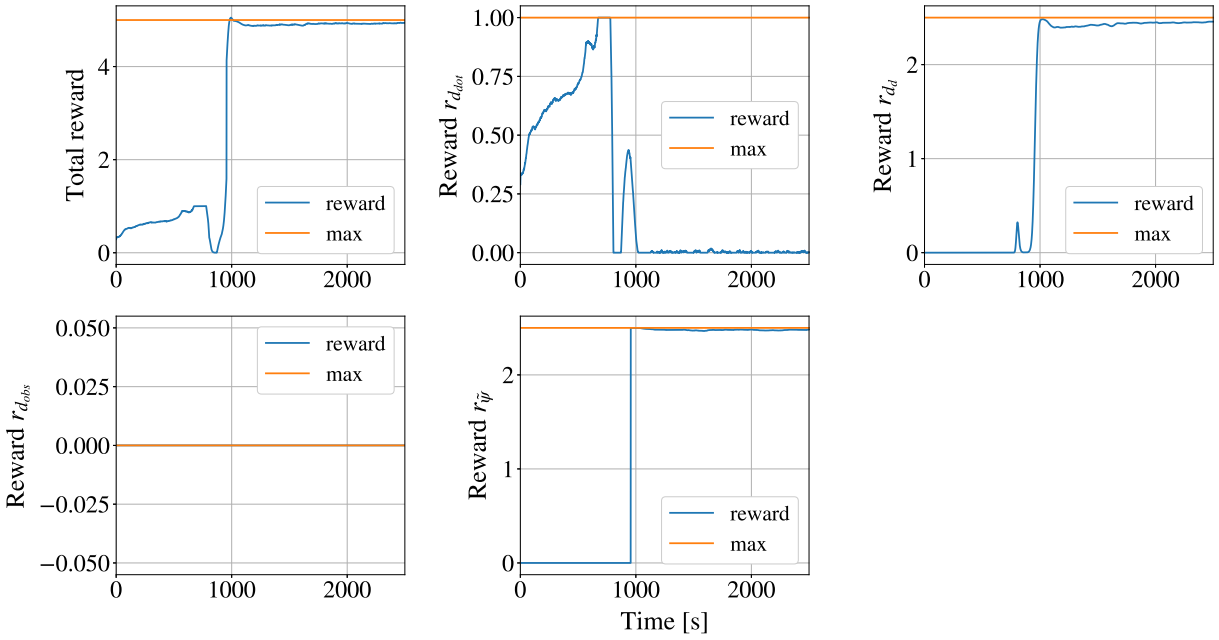


(b) Ocean current

Figure 4.29: State plots for LP4:Distance berthing agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[312.183 m, 876.256 m, -0.384 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].

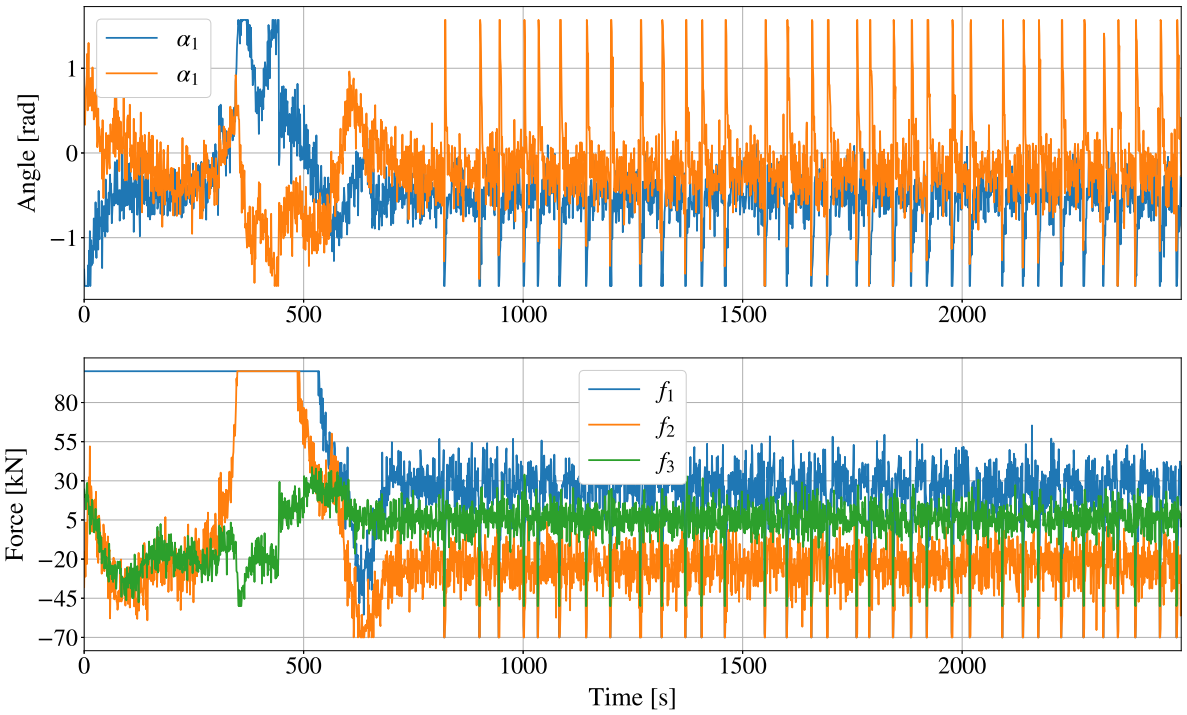


(a) No ocean current

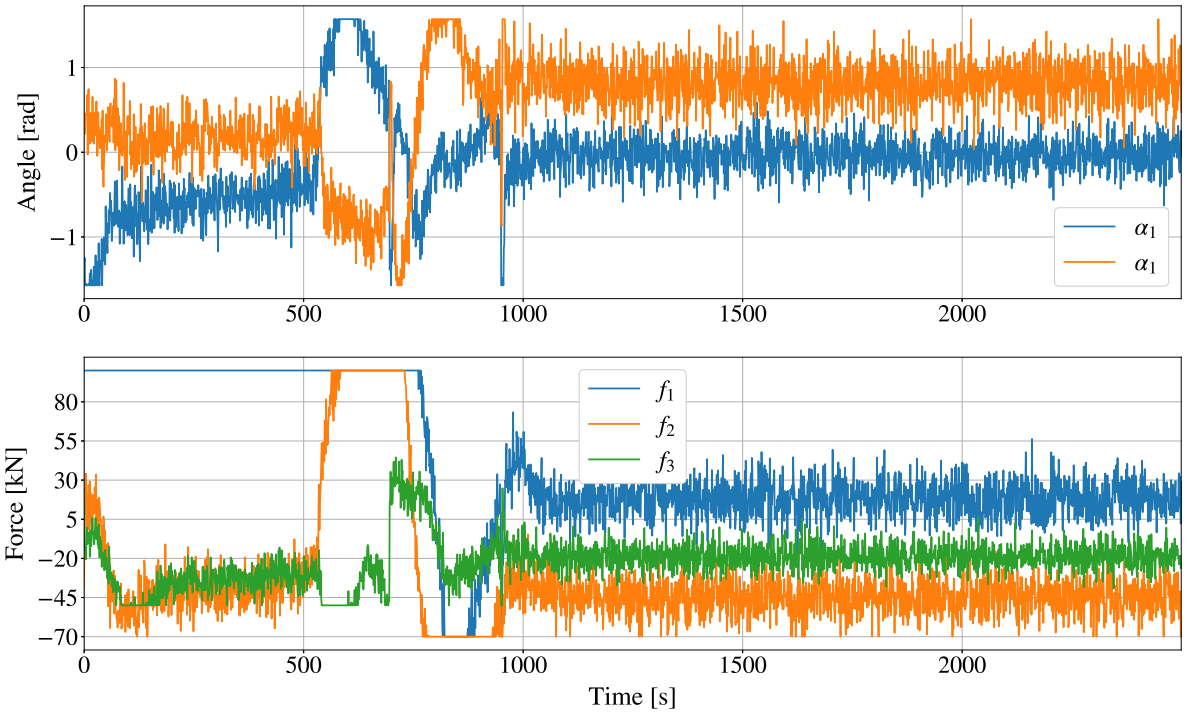


(b) Ocean current

Figure 4.30: Reward plots for LP4:Distance berthing agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[312.183 m, 876.256 m, -0.384 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].

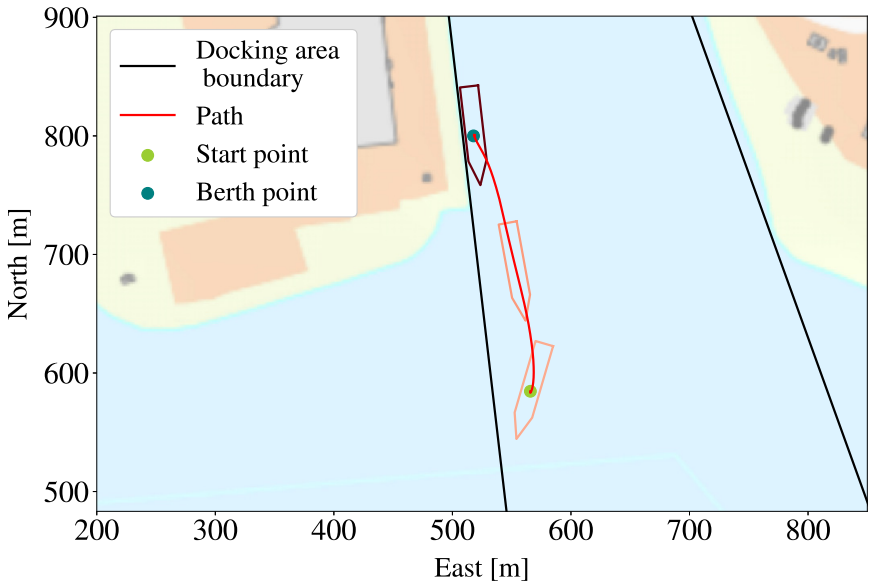


(a) No ocean current

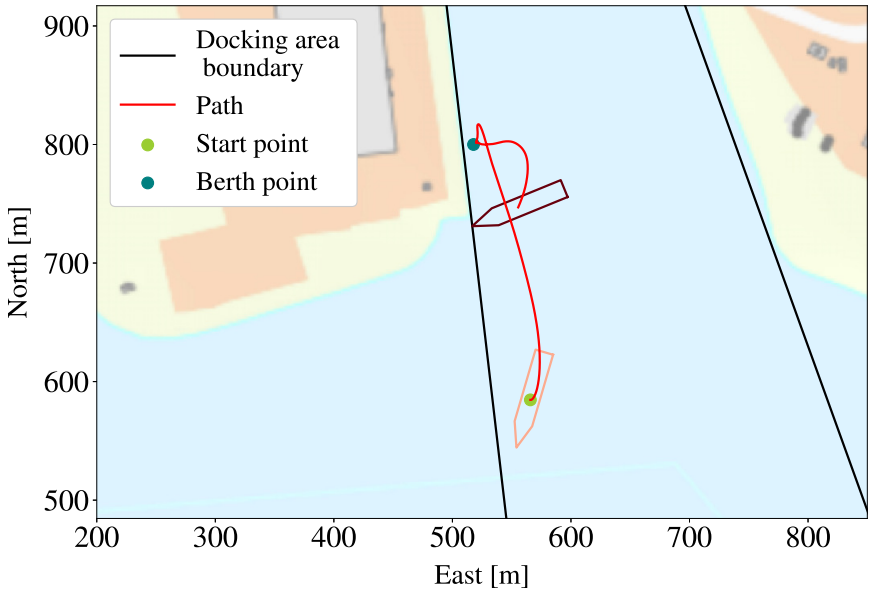


(b) Ocean current

Figure 4.31: Thruster plots for LP4:Distance berthing agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[312.183 m, 876.256 m, -0.384 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].



(a) No ocean current



(b) Ocean current

Figure 4.32: Trajectory plots for LP4:Distance berthing agent when crashing with current. Test episodes starts with initial state $\boldsymbol{\eta}$ =[585 m,566 m, -2.86 rad] and $\boldsymbol{\nu}$ = [0.23 m/s ,0 m,/s, 0 rad/s].

4.5.3 Analysis of SHAP values

4.5.3.1 General contribution of states

The general contribution of states to the DRL agents selection of control input was analysed using the mean absolute SHAP values for each state to each control input, using 400 random samples from the environment. The valid sets of states for selecting random samples from the environment are listed in Table 3.7. The accuracy of the explanation model to capture the DRL-agents policy is visualised in Figure D.3 in Appendix D.3.

The mean absolute SHAP values are illustrated in Figure 4.33. The SHAP values are calculated based on raw and unscaled data of the thrusters, so the absolute value of the SHAP values between the control inputs should not be compared. The relationship between the contribution of the states is more interesting than the values themselves. Higher values of mean absolute SHAP values of one state for one control input shows that on average, the state contributed more than the other states to the average control input.

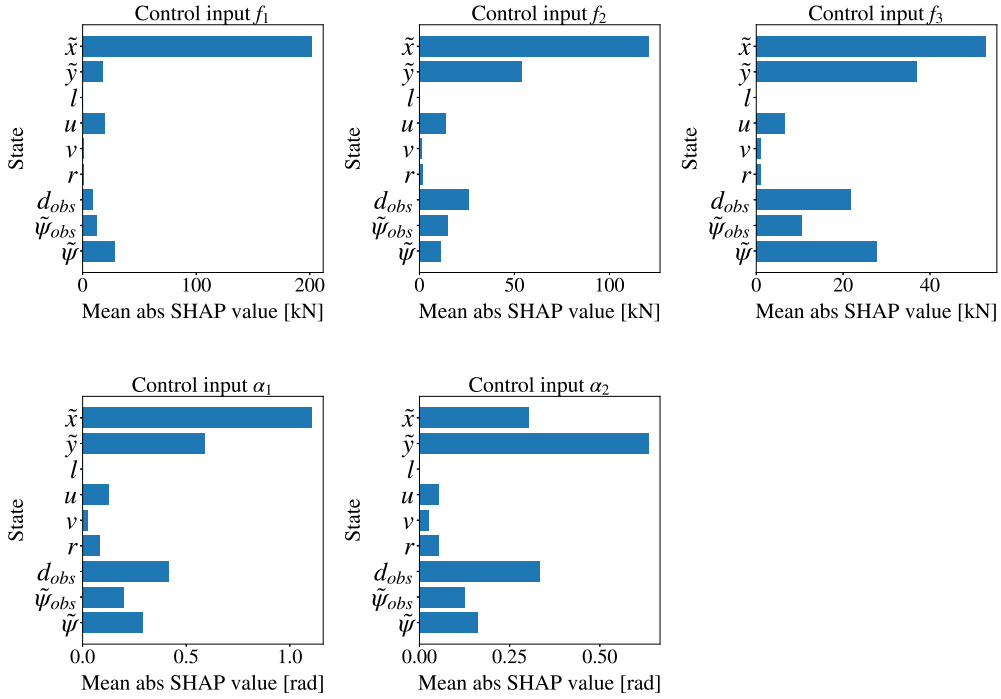


Figure 4.33: Mean absolute SHAP values of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$

From Figure 4.33 it can be observed that the distances to the berth $\tilde{x}$ and $\tilde{y}$ was the states that generally contribute the most to the selection of all control inputs. The second most

important attribute appeared to be the distance to the closest obstacle. The velocities and angles seemed to be contributing less. The random samples used in the SHAP analysis was only collected when the vessel was on the sea, which led to the land binary variable l having zero contribution for all the thrusters, due to how the SHAP values are calculated.

The states contributed differently to the thruster angles and forces, as observed when analysing the states mean SHAP value for each control input in Figure 4.33. This seems intuitive, as the different thrusters had different physical roles in controlling the vessel. For instance, distance to obstacle d_{obs} had a higher general contribution to tunnel thruster f_3, than to the azimuth thrusters. This means that the selection of thruster force f_3 was more impacted by the distance to the obstacle, than the azimuth thrusters. It can also be recognised that the distance d_{obs} generally contributed similarly to the selection of azimuth thruster forces $[f_1, f_2]$ and angles $[\alpha_1, \alpha_2]$. This indicates that the additional training with the vessel starting at the port and starboard side (as relative to the berth) is working well, as recognised in LP2:Berthing.

4.5.3.2 Contribution of states in an episode

The SHAP values calculated based on instances of a specific episode can give insight into why a specific action was selected by the DRL agent at a certain time. The episode analysed here is the same as illustrated in Section 4.5.2, without current. The episode has a length of 1500 seconds, and can be summarised as:

- The vessel starts in the middle of the harbour.
- The vessel increases its velocity from 0.5 to 1.5 m/s until approximately 600 seconds and decelerates after that to approximately zero velocity when reaching the berth.
- At around 800 seconds the vessel reaches the berth and stays there until the end of the episode. The berth is placed close to the quay.

The accuracy of the explanation model is visualised in Figure D.4 in Appendix D.3.

The SHAP values of the states for each timestep of the episode are illustrated in Figure 4.34. The SHAP values at a certain time will not necessarily be comparable to other time instances or between controller inputs, due to the dependency of the size of the value of the control input, as previously mentioned. For instance, the SHAP values of distance to target $\tilde{x}$ for thruster force f_1 had higher values at 0 seconds, than at 1000 seconds. When analysing the action plot from the episode, in Figure 4.31, the force f_1 was closer to 200 kN at 0 seconds and was reduced to 20 kN at 1000 seconds. The relation between the SHAP values at a

certain time instance are easier to compare. The relative contributions of state $\tilde{x}$ to thruster force f_1 were 60 % and 50 % respectively at those time instances, indicating that the state $\tilde{x}$ contributed similarly in the selection of action, in those two cases. The relative contribution of a state compared to the other states contribution was calculated using equation (3.39).

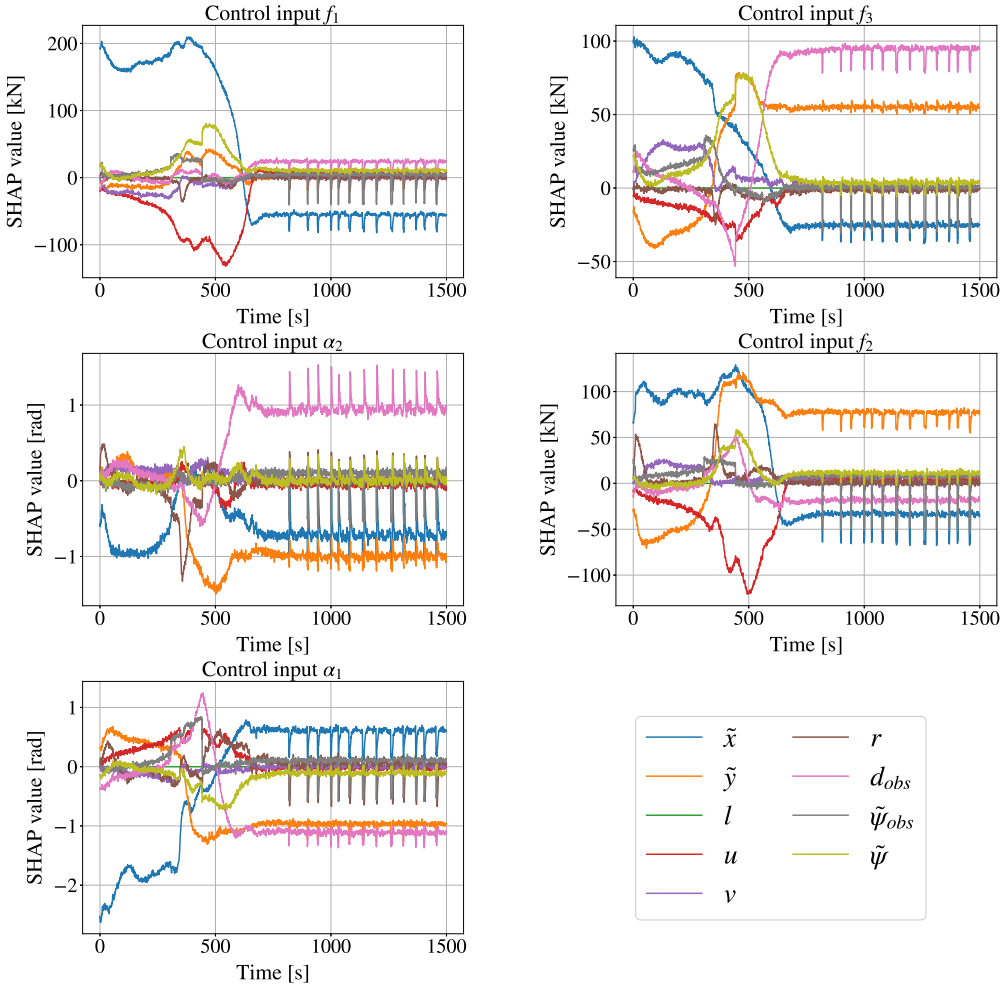


Figure 4.34: SHAP values of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$ from one test episode of LP4:Distance berthing.

The relative contributions of states at each timestep of the episode are presented in Figure 4.35. Generally, through this episode, the distances to the berth $\tilde{x}$ and $\tilde{y}$ and the distance to closest obstacle d_{obs}, had the highest relative contribution, as observed in the analysis of general contributions in the previous section.

For all the control inputs, the relative contribution of $\tilde{x}$ to the action, decreased as the vessel

got closer to the berth, at approximately 800 seconds, from Figure 4.35. In contrast, the relative contribution of $\tilde{y}$ increased as the vessel got closer to the berth for almost all the control inputs. The distance $\tilde{x}$ represents the distance to the berth in the direction of vessels heading, while $\tilde{y}$ represents the perpendicular distance to the berth. This can be interpreted as though the agent found it more efficient to first reduce the distance in the direction of the vessels heading, before reducing the perpendicular distance $\tilde{y}$ when closer to the berth. The relative contribution of heading error $\tilde{\psi}$ also increased for most control inputs as the vessel got closer to the berth. This might indicate that the agent first desires to come close to a berth, before controlling it into a specific pose.

From the general analysis of SHAP-values, it was remarked that not all states contributed equally to all the control actions. From Figure 4.35 it can be observed that the distance to the target $\tilde{x}$ contributed more to the selection of thruster force f_1 than $\tilde{y}$ during berthing, while for other control inputs the state $\tilde{x}$ contributed more during transit while $\tilde{y}$ contributed more during berthing. From the plot of the trajectory, shown in Figure 4.31, the thruster force f_1 appeared to have higher values during transit in the beginning. This might indicate that thruster force f_1 might be used more for propulsion, while the others more for manoeuvering.

The relative contribution of d_{obs} increased as the vessel got closer to the berth for all the control inputs, as observed in Figure 4.35. The increase in relative contribution could indicate that the agent understood that obstacles are only dangerous when they are close to the vessel, as in the case of the quay near the berth. The oscillations in the relative contribution of d_{obs} can be due to discretisation in the measurements of the distance to the obstacle, as seen in Figure 4.29.

The relative contribution of the forward velocity u also increased as the vessel got closer to the berth, but decreased as the vessel reached zero forward velocity, as seen in Figure 4.35. This can indicate that the agent understood that it was crucial to decrease its speed to reach the berth without crashing. This tendency was also observed also for the velocities sway u and yaw r.

From the analysis of the relative contributions of the states for LP4:Distance berthing, it appears that the agent had understood the intention of controlling a vessel away from obstacles and to reach the berth. The agent also appears to be acting in an intuitive way.

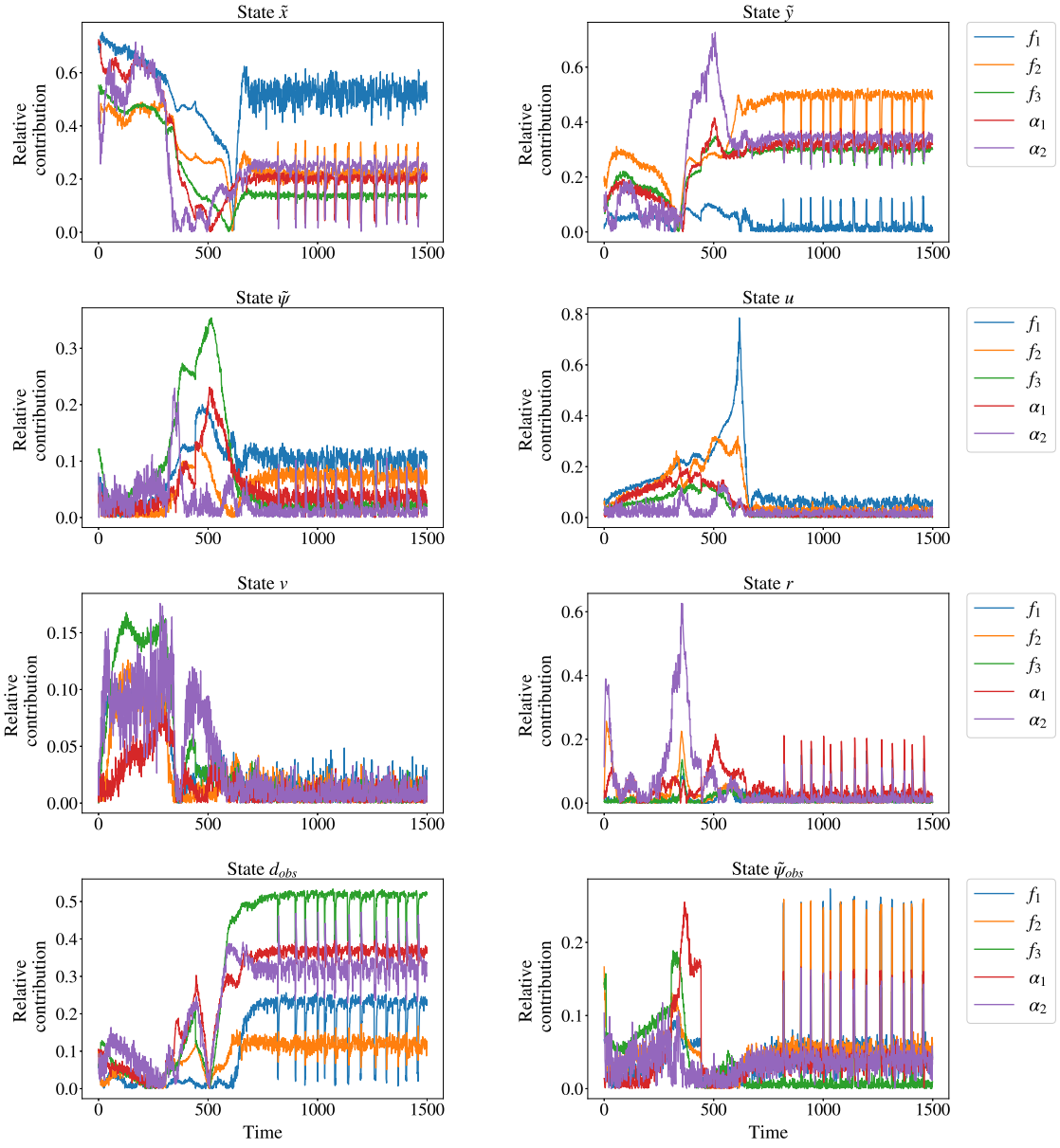


Figure 4.35: Relative contributions of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$ from one test episode of LP4:Distance berthing

4.6 LP5: Docking

The LP5:Docking agent performed all the docking phases, described in Section 3.1. The objective of the agent was to reduce the vessels forward velocity so that it obeyed the speed limit of the harbour while controlling it towards the berth and kept it there. The vessel started an episode within a range of 400 m from the berth, with a start forward velocity of 2.5 - 3.5 m/s if the vessel started outside the harbour, or -0.5 - 0.5 m/s if inside the berth. The agent controlled the vessel through the thruster forces and angles $[\alpha_1, \alpha_2, f_1, f_2, f_3]$. The LP5:Docking agent was trained with PPO, using training episodes described in Section 3.3.5.

The reward function of LP5:Docking consists of five reward components,equation (3.29), and is an extension of the reward function used in LP4:Distance berthing. The same reward function parameters of reward components from LP4:Distance Berthing were used. The additional reward component r_u, for LP5:Docking, penalises breaking the speed limit within the harbour. The weight of the reward component C_u was set to 1, which is lower than the respective weights of 2.5, 2.5 and 1 of the other reward components. The modest weight was found sufficient, based on the experience that the agent quickly preferred lower velocities. The reward function parameters are summarised in Table 3.3.

4.6.1 Training progress

The average return of an episode during the training of the PPO agent is illustrated in Figure 4.27. The figure shows that the average return converged after approximately 7.5 million interactions between the environment and the agent, to a steady-state value of approximately 2300. The maximum received reward for one episode was 6000, which is only achievable if the vessel starts at the target pose and stayed there.

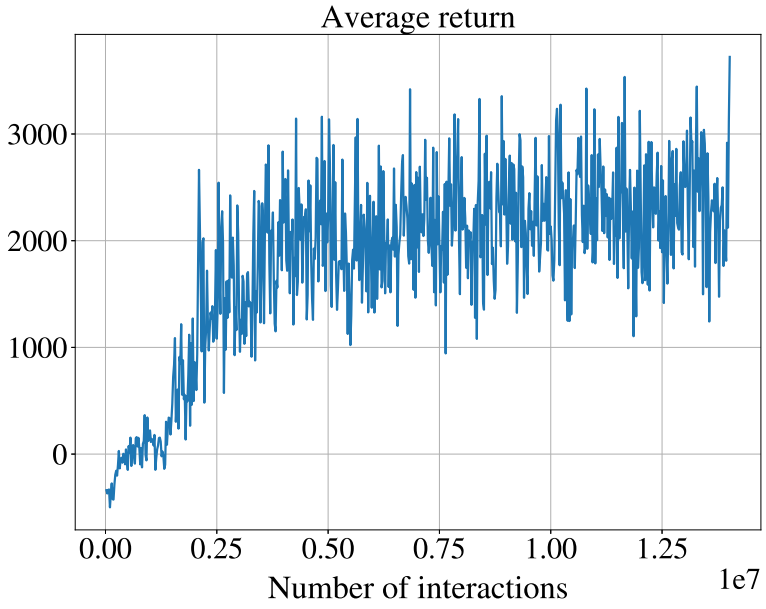


Figure 4.36: The average return during training as a function of number of interactions between agent and environment, for the LP5:Docking agent.

4.6.2 Performance

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta}$ =[317 m, 664 m, 0.334 rad] and $\boldsymbol{\nu}$ = [3 m/s, 0 m/s, 0 rad/s]. The trajectory plot of the vessel in this episode is visualised in Figure 4.28 and corresponding video LP5-Docking-PPO. It shows that the PPO agent controlled the vessel nicely towards the berth and kept it there, with a best achieved accuracy of 0.005 meters. The agent also obeyed the speed limit, reducing the speed from 3 m/s before the harbour. The vessel's trajectory is naturally affected by the high initial forward velocity, ending up taking a larger turn before converging towards the berth.

The state plot of the vessel in the test episode is illustrated in Figure 4.38. Even though the higher velocities affected the trajectory, the distances to the berth $\tilde{x}$ and $\tilde{y}$ appears to have decreased quite efficiently and converged well towards zero. This is also reflected in the reward plots, where the reward components $r_{d_d} + r_{\tilde{\psi}}$,for being close to the berth pose, increased steadily, as seen in Figure 4.39.

The state plot of the vessel, in Figure 4.38, shows that the vessel started with a surge of 3 m/s, but quickly reduced it and reached a speed below 2.5 m/s before entering the harbour. From the reward plot, in Figure 4.39, it can be observed that the agent does not receive a penalty for being above the speed limit in the harbour. One interesting observation is that the reward component $r_{d_{dot}}$, for moving towards the target, increased after the agent had reduced the speed before entering the harbour. This can be interpreted as the agent first wanted to slow

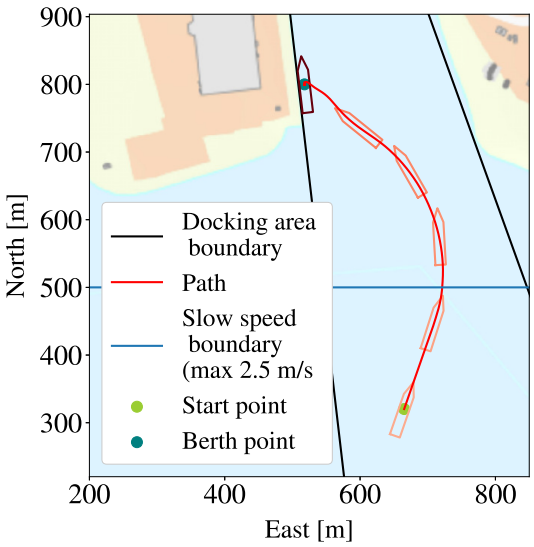


Figure 4.37: LP5 - maps

down the surge, to gain control, and after that increased it to reach the harbour faster. This is supported by analysing the actuator plot 4.40. The agent exhibited a high backwards thrust, to begin with, trying to reduce the vessels speed. Other than the backward thrust, in the beginning, the agent's use of the thrusters is similar to that of the LP4:Distance berthing agent, being quite aggressively.

The results from performing LP5:Docking in several test episodes are presented in Table 4.7. The vessel was demonstrated capable of solving the task of avoiding obstacles in several test episodes, reaching the berth and obeying the speed limit. The maximum return is 10000, and is only achievable if the vessel starts and stays at the target during the entire episode. Unfortunately, the agent was not always able to solve the task, and even sometimes made contact with the quay, typically at low speeds in the range of 0.05-0.2 m/s. This was often the case in episodes when the vessel was started close to obstacles. The agent did however manage to avoid making contact in the case of LP4:Berthing, which in all respects is identical to LP5:Docking except with the added speed limit. Not being able to always avoid obstacles, when adding a speed limit, can be due to:

- The agent is only exposed to higher velocities in a smaller area right outside the harbour, and therefore might be experiencing difficulties with learning.
- The agent needs more information about the velocity constraint, through additional states or different design of reward function.
- The agent can be started too close to land at higher velocities during training, and thereby have relatively lower chance of stopping the vessel before crashing. One

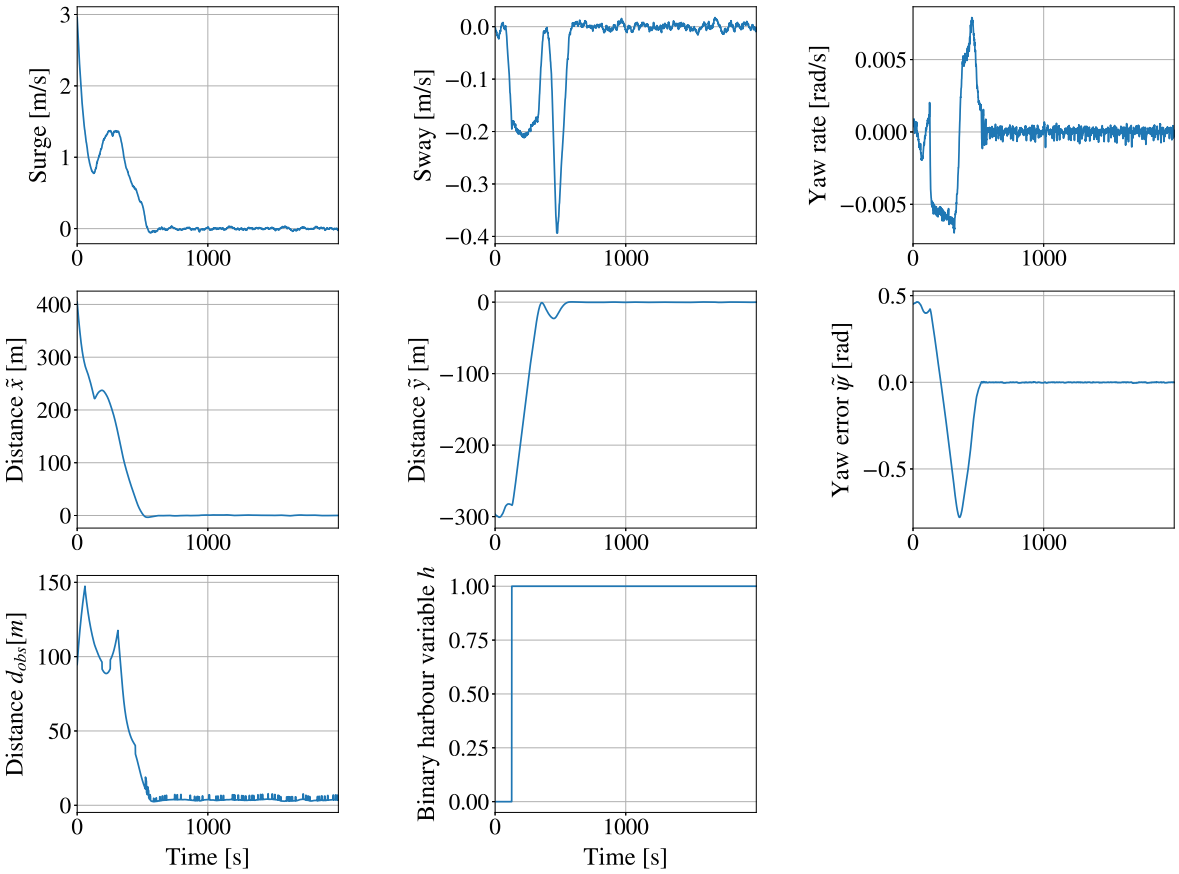


Figure 4.38: State plots for LP5:Docking agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[317 m, 664 m, 0.334 rad] and $\boldsymbol{\nu}$ = [3 m/s, 0 m/s, 0 rad/s].

possible fix to this, would be to perform a thorough study of how to generate valid scenarios for the vessel.

- Learning in this complex setting may be too much for one agent trained with methods such as PPO in a simple two-hidden-layer ANN. It might be necessary to use other techniques, which enables more high level thinking.

This is further discussed in Section 5.

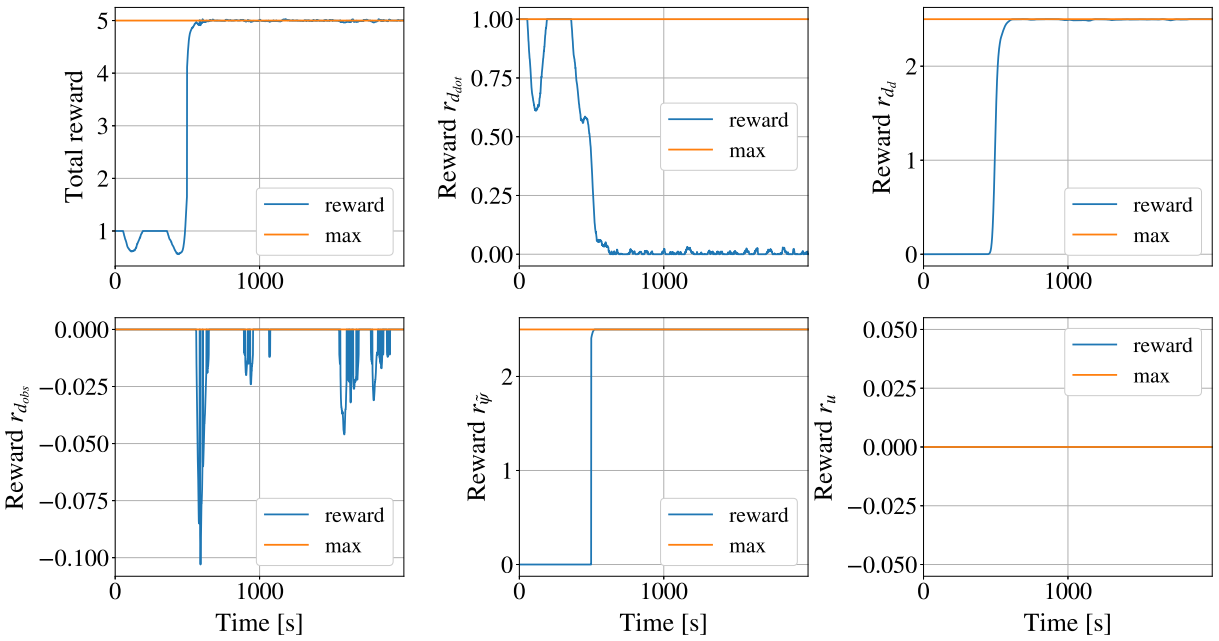


Figure 4.39: Reward plots for LP5:Docking agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[317 m, 664 m, 0.334 rad] and $\boldsymbol{\nu}$ = [3 m/s, 0 m/s, 0 rad/s].

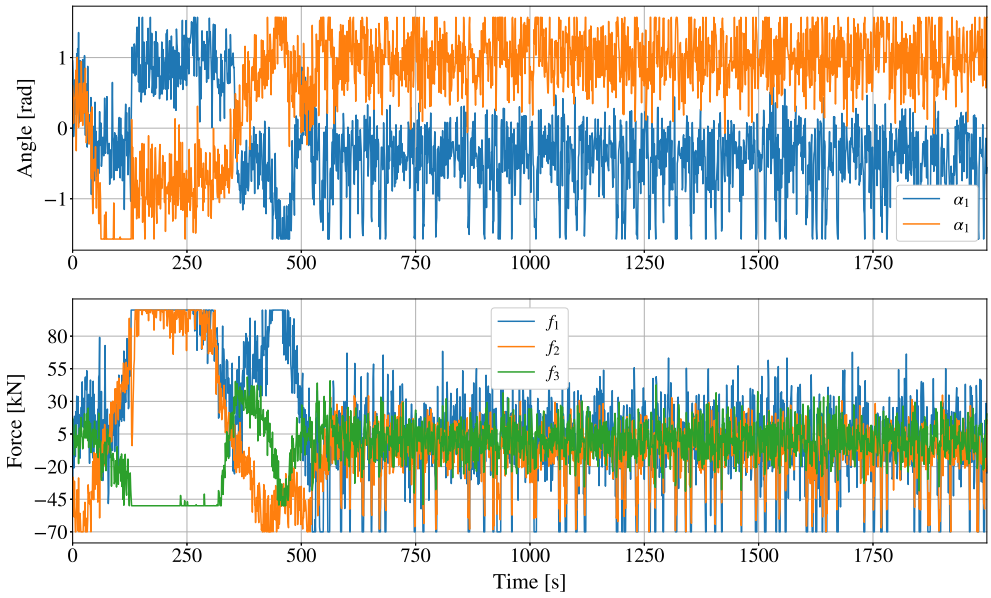


Figure 4.40: Thruster plots for LP5:Docking agent. Test episodes starts with initial state $\boldsymbol{\eta}$ =[317 m, 664 m, 0.334 rad] and $\boldsymbol{\nu}$ = [3 m/s, 0 m/s, 0 rad/s].

Table 4.7: The results from the LP5:Docking on several test episodes. The test episodes are run for 2000 seconds.

Initial position x, y, ψ, u, v, r **[m,m,rad,m/s,m/s,rad/s]**	**Min absolute** d_d, **m**	**Mean absolute** d_d **m**	**Return**
317, 664, 0.334,3, 0, 0	0.005, 0.000	58.970, 0.100	7919.742
672, 664 0.297,0.30, 0., 0.	(0.013, 0.000)	(20.211, 0.048)	8149.024
811, 600, -0.0275,-0.329, 0., 0.	(0.076, 0.000)	(6.807, 0.029)	8885.813
292, 638, -0.277,1.178, 0., 0.	0.061, 0.000	99.017, 0.099	7071.631
277,643, 0.734, 3.21, 0. 0.	0.052, 0.000	105.033, 0.191	6829.862
524,639, -2.598,0.218, 0., 0.	28.712, 0.002	102.880, 0.369	15

4.6.3 Analysis of SHAP values

4.6.3.1 General contribution of states

The general contributions of the states to the selection of control action (thrusters) were analysed using the mean absolute SHAP-value from 400 random samples of the environment. The valid sets of states for selecting random samples from the environment are listed in Table 3.7. The accuracy of the explanation model to capture the DRL-agents policies are visualised in Figure D.5 in Appendix D.4.

Figure 4.41 visualises the general contribution of states to the selection of thrust of the LP5:Docking agent. The general contribution of states for LP5:Docking agent and those of LP4:Distance berthing, are different both in absolute values and the relationship between the contribution of states, comparing Figure 4.33 and 4.41. In the same manner as for LP4:Docking, the distances to the berth $\tilde{x}, \tilde{y}$ and distance to obstacles d_{obs} still represent a significant general contribution to several of the control inputs. However, they don't have as a high relative general contribution, as for LP4:Docking. Several of the other states have a higher relative general contribution to the control inputs. For instance, the forward velocity u had a higher general contribution to all the control inputs, compared to LP4:Distance Berthing. The additional state h, signifying whether the vessel is inside the harbour or not, also contributed significantly in the selection of thrust. This might be interpreted as the agent has understood the concept of speed as being more important in this learning phase.

4.6.3.2 Contribution of states in an episode

The docking episode used as an example of the contribution of states to the control inputs (thrusters), is the same episode as presented in Section 4.6.2. The episode had a length of 1500 seconds, and can be summarised as:

- The vessel started outside the harbour at 0 seconds, relatively far from any obstacles.
- The vessel was inside the harbour at around 100 seconds.
- At around 800 seconds, the vessel reached the berth and stayed there until the end. The berth was placed close to obstacles, as in LP4.
- The vessel decelerated from 3 m/s in forward velocity at the start of the episode to approximately zero when reaching the berth.

In the following section, the relative contribution of the states to the control inputs will be discussed. The development of the SHAP values for each control input can be found in Figure

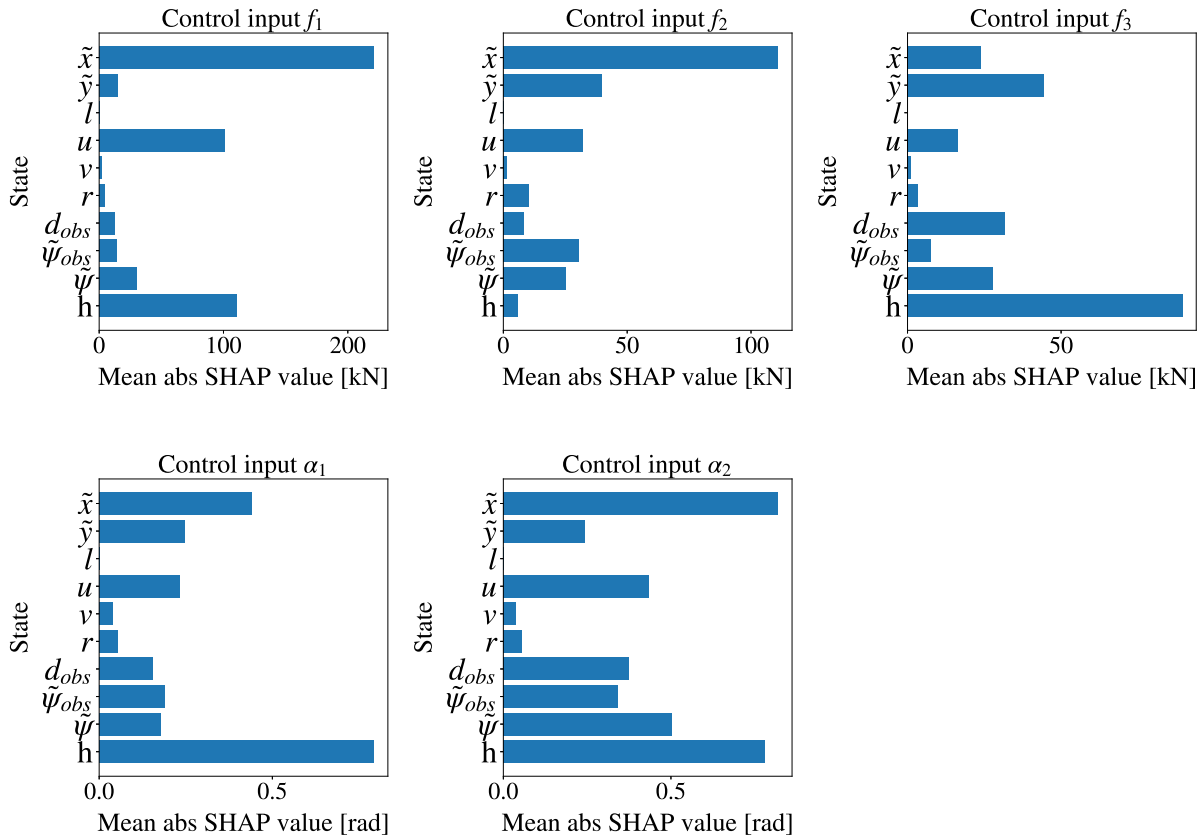


Figure 4.41: Mean absolute SHAP values of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$

D.7, in Appendix D.4. The accuracy of the explanation model to capture the DRL-agent's policy during the test episode is visualised in Figure D.6 in Appendix D.4.

The relative contributions of each state to each control input are represented in Figure 4.42. The plots show that the binary harbour state h had high relative contribution to all control inputs throughout the entire episode. For all the control inputs, the relative contribution of forward velocity u appears to be higher, than for LP4:Distance berthing, especially during berthing. This might indicate that with higher initial forward velocities, it is more important for the agent to reach lower velocities close to the berth to avoid colliding with the quay. The velocities sway v and yaw rate r, had small relative contributions to the control inputs during transit, and even less important when the agent is trying to holds its position.

The relative contribution of the distances to the berth $\tilde{x}$ and $\tilde{y}$ to the control inputs had similar development as LP4:Distance docking, during the episode. The relative contribution heading error $\tilde{\psi}$ exhibited higher values while proceeding to the berth, compared to LP4:Distance

berthing. This might suggest that the agent manoeuvred the vessel at high velocities more based on angles than for LP4:Distance berthing, to reduce speed and avoid obstacles until the vessel had corrected its initial heading. We can also see from the relative contributions of state $\tilde{\psi}$ that it contributed more when closer to the berth again.

The agent experienced some cases of colliding with the quay. However the relative contribution of the distances to the obstacles d_{obs}, to all the control inputs was still high, compared to LP4:Distance berthing. One may consequently suggest that the agent experienced some problems due to incomplete training scenarios or insufficient training space within the harbour, as discussed in the previous section.

From the analysis of the relative contributions of the states for LP5:Docking, it appears that the agent had a less intuitive logic of the objectives of the learning phase than for LP4:Distance berthing. The LP5:Docking agent also had some performance problems, which might indicate that with a less intuitive explanation, the agent might not be as trustable.

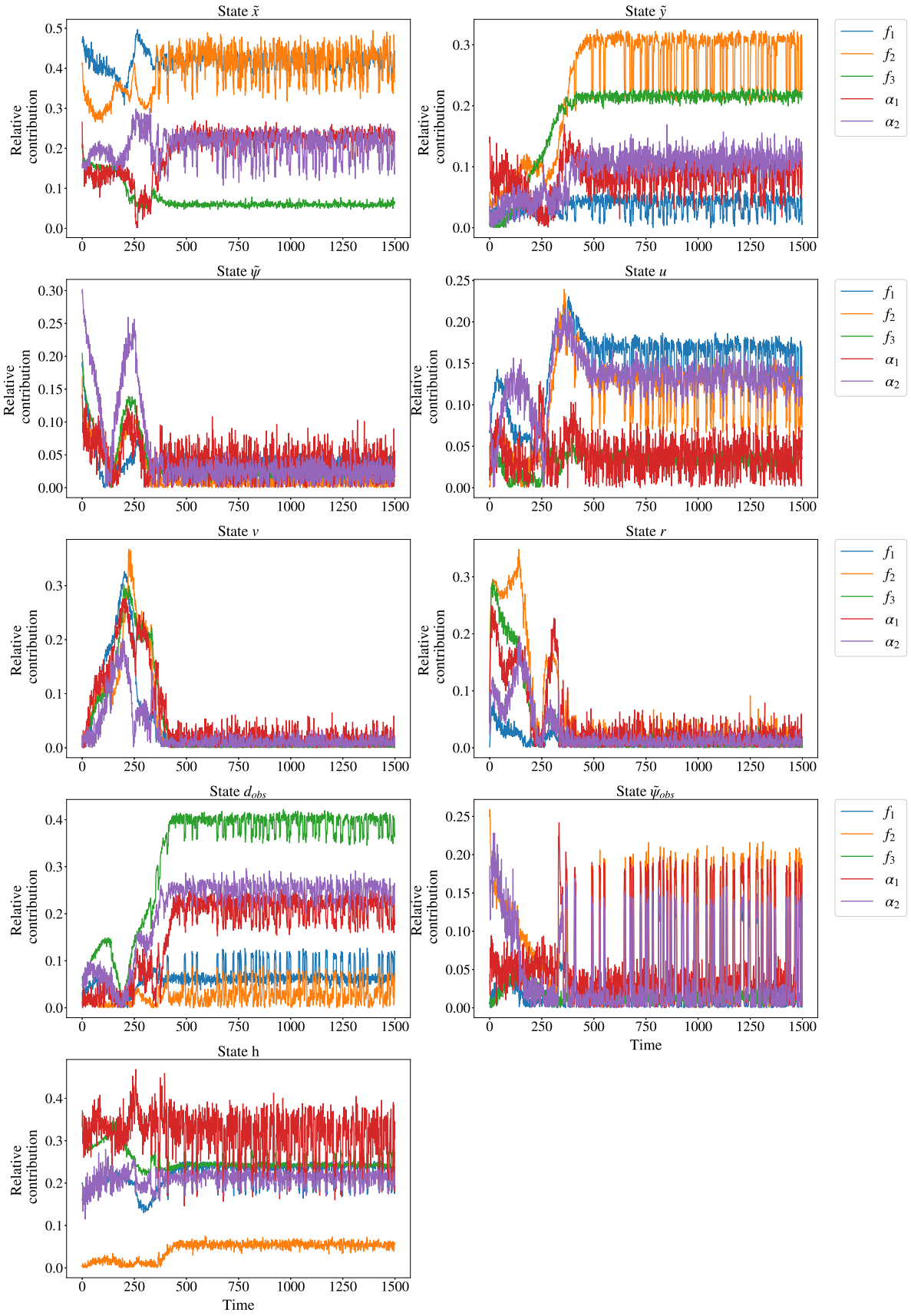


Figure 4.42: Relative contributions of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$ from one test episode of LP4:Distance berthing.

4.7 Reduction of aggressive use of control inputs

Through all the learning phases the agent demonstrated aggressive use of the thrusters. It was experimented with reducing this for LP4:Distance berthing, as a proof of concept. In Section 3.3.3.8 two alternative ways of doing this were presented:

- **LP4:Added penalty**: The reward function from LP4:Distance berthing with and additional penalty for reducing rapid changes in control inputs $r_{\dot{a}}$.
- **LP4:Extended state**: The same as the previous step, but in addition the state vector from LP4:Distance berthing was extended with the thruster angles $\boldsymbol{\alpha}$ and forces $\mathbf{f}$, and their derivatives $\dot{\boldsymbol{\alpha}}$ and $\dot{\boldsymbol{f}}$.

The LP4:Added penalty and LP4:Extended state was trained with PPO, using the same training episodes as for LP4:Distance berthing, presented in Table 3.5. The same set of reward function parameters from LP4:Distance docking was also used here.

The penalty $r_{\dot{\alpha},\dot{f}}$ for aggressive use of the control inputs, are a weighted summation of the absolute value of the derivative of the control inputs. The weights of the reward component $r_{\dot{\alpha},\dot{f}}$ was set to penalise the derivatives equally. This meant the weight of the derivative of thruster angle $\dot{\alpha}$ was $C_{\dot{\alpha}} = -0.15/\dot{\alpha}_{max}$, and the derivative of thruster force $\dot{f}$ was $C_{\dot{f}} = -0.15/\dot{f}_{max}$. It was hard to calibrate these weights. With higher values of the weights, the agent had problems with exploration during training, probably due to receiving a to harsh penalty for trying to move around. The described weight was the lowest penalty which lead to a reasonable solution within reasonable training time.

4.7.1 Training progress

Figure 4.43 presents the development during the training of the average return of the agents as a function of the number of interactions between the environment and agent. The LP4:Added penalty and LP4:Extended state agents, trained slower than LP4:Distance berthing, as shown in Figure 4.27. Adding more objectives through one scalar signal, the shaping of the reward function becomes increasingly difficult. It seems that the agent needed a lot more training time to decode the different reward components embedded into the single reward. It is however observed that the LP4:Extended state agent, which had the extended state vector, achieved a higher average return faster than the agent with the only penalty.

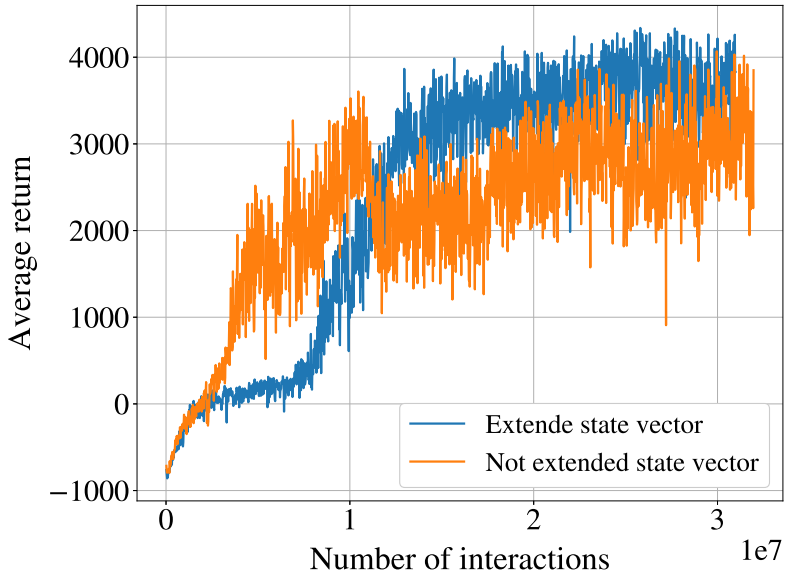


Figure 4.43: The average return during training as a function of number of interactions between agent and environment, for the LP4:Extended state and LP4:Added penalty agents.

4.7.2 Performance

The test episode illustrated in this section started with the vessel in the initial state $\boldsymbol{\eta}$ =[345 m, 760m, -0.1 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s]. The trajectory is illustrated in Figure 4.44. It shows that the agent without extended state vector, selected a less direct route to the berth than the agent with extended state vector. Even though the agents selected different paths, they achieved similar best accuracies, of 1.174 m for the agent with extended state vector and 0.751m for the agent without extended state vector. The state plots during the test episode is appended in Appendix C.2, in Figure C.4.

The reward that the agents received during the test episode are visualised in Figure 4.45. These show that both the LP4:Added penalty and LP4:Extended state agents, tried to reduce the penalty for aggressive use of the thrusters. It can be observed, however, that the LP4:Extended state agent achieved alternating large and small penalties for the use of the control inputs, through reward component $r_{\dot{\alpha},\dot{f}}$. This can be caused by the fact that the LP4:Extended state agent optimises the control input usage by performing rapid and alternating changes of the control inputs such that they did not change them simultaneously. This is visualised in Figure 4.46. The LP4:Extended state agent "hacks" the reward function by maximising the reward function, but without reaching the objective as intended by the creators. Extending the state vector might have enabled the LP4:Extended state agent to relate more directly the states to the penalties, and thereby creating a solution to "hack" the system. The control input for the LP4:Added penalty agent, on the other hand tried to reduce the derivatives of

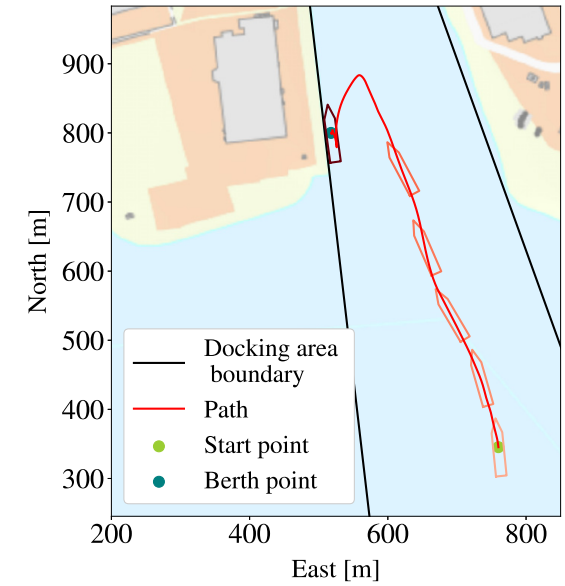
the thrusters.

The original LP4:Distance berthing agent was compared in several episodes with the LP4:Extended state and LP4:Added penalty agents. These results are presented in table 4.8, where the mean absolute norm of control inputs f, α gives an indication of the average control inputs through the episode. The results showed that generally, the LP4:Distance berthing agent had the highest achieved accuracy of target, with the most aggressive use of the thrusters. The LP4:Extended state and LP4:Added penalty achieved sufficiently high accuracy to the berth in most episodes, but at some occurrences the vessel touched the quay right next to the berth. This might indicate that when very close to the berth, the penalty for aggressive thruster use should have been diminished when close to the berth.

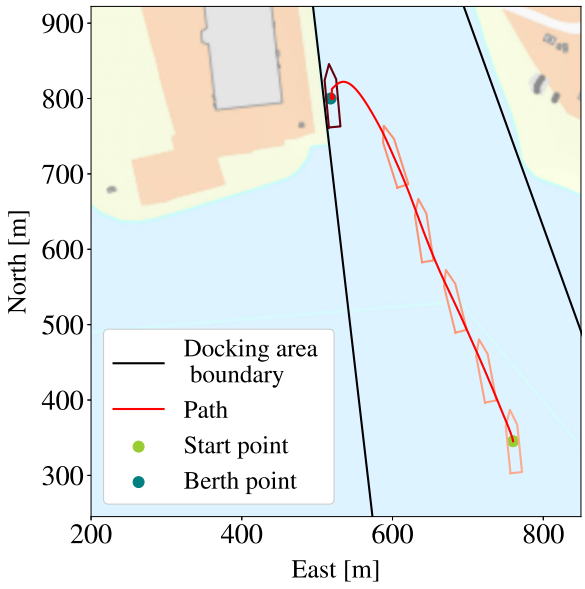
When the LP4:Extended state and LP4:Added penalty crashed into the quay, the agents had low velocities. A vessel would in real life travel at a sufficiently small velocity towards the berth to allow the fender to absorb the kinetic energy. This indicates that the objective of the training of the agents perhaps should be changed from keeping the pose (performing DP) at the end state, to park a vessel close to the quay, and allows small velocities, and not hold the position. Vessel of the size of the simulated container ship can have a maximum speeds towards mooring structures of no more than 0.1-0.4 m/s, according to international standards [137]. Allowing touching the quay at low velocities and not hold its position, would simplify matter greatly. As seen in LP5:Docking, the agent almost seemed "afraid" of the quay close to the berth, in Figure 4.32. This is more discussed in Section 5.

Table 4.8: The results from the LP4:Extended state and LP4:Added penalty agents on several test episodes. The test episodes are run for 1500 seconds. The type "Extended + penalty" is the LP4:Extende state agent, and "Penalty" is the LP4:Added penalty, and "Normal" is LP4:Distance berthing agent.

Type	**Initial position** $[x, y, \psi, u, v, r]$ [m,m.rad,m/s,m/s,rad/s]	**Min absolute** $d_d, \tilde{psi}$ **[m,rad]**	**Mean absolute** $d_d, \tilde{psi}$ **[m,rad]**	**Mean abs norm** $\dot{f}, \dot{\alpha}$	**Return**
Extended + penalty	345,760,-0.1,0.3,0,0	1.174, 0.000	42.553, 0.027	0.143, 0.158	10190
Penalty		0.751, 0.000	103.881, 0.111	0.116, 0.140	7393
Normal		0.276, 0.000	41.975, 0.047	0.171, 0.251	10786
Extended + penalty	317,664,0.33,3,0,0	2.188, 0.000	53.172, 0.060	0.152, 0.160	9687
Penalty		255.518, 0.000	459.929, 0.104	0.019, 0.015	-313
Normal		0.252, 0.000	91.606, 0.080	0.167, 0.235	9786
Extended + penalty	672,664,0.30,0.30,0, 0	337.013, 0.000	339.753, 0.001	0.003, 0.002	-602
Penalty		0.751, 0.000	29.717, 0.047	0.124, 0.172	10400
Normal		0.162, 0.000	24.978, 0.015	0.168, 0.243	10955
Extended + penalty	811, 600, -0.03,-0.3, 0, 0	0.763, 0.000	22.550, 0.060	0.226, 0.159	10534
Penalty		1.279, 0.000	27.759, 0.129	0.139, 0.123	10134
Normal		0.795, 0.000	17.252, 0.081	0.176, 0.301	11239

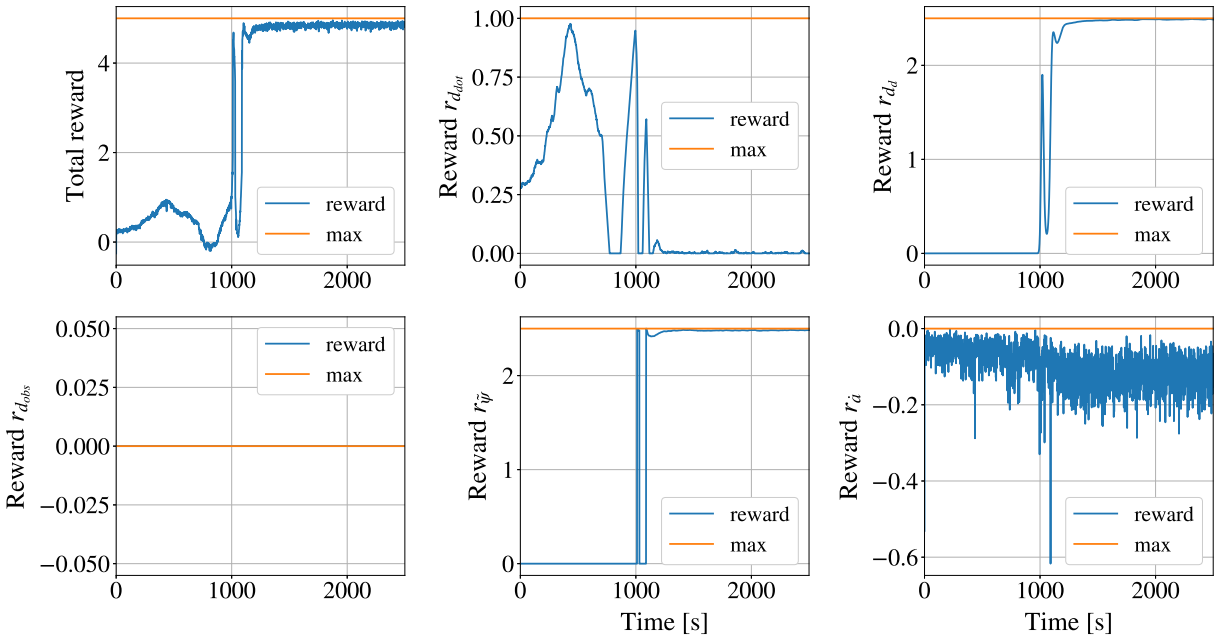


(a) LP4:Added penalty

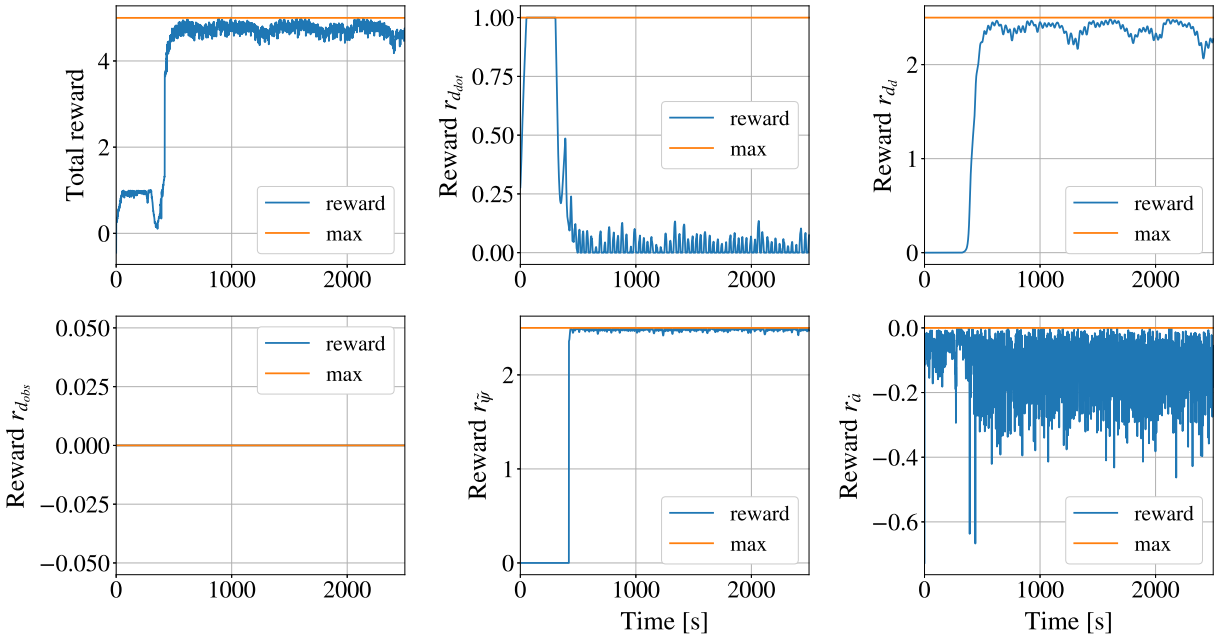


(b) LP4:Extended state

Figure 4.44: Trajectory plots for the LP4:Extended state and LP4:Added penalty agents. Test episodes starts with initial state $\boldsymbol{\eta}$ =[345 m, 760m, -0.1 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].

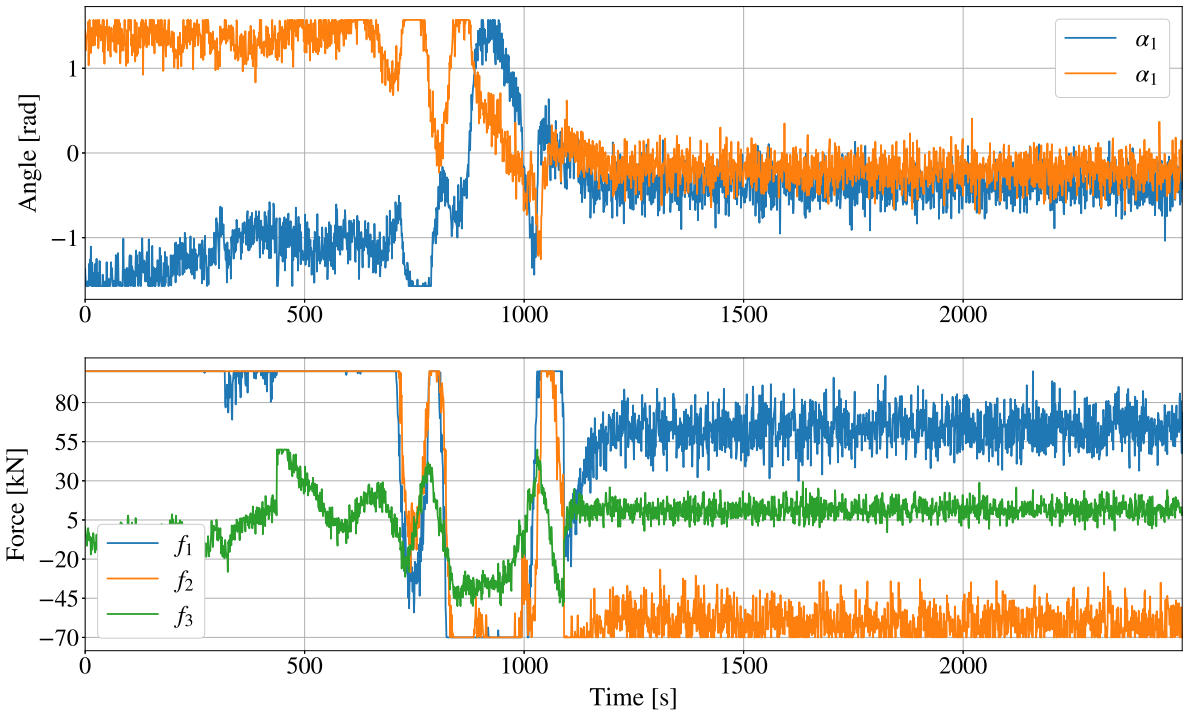


(a) Penalty + not extended state vector

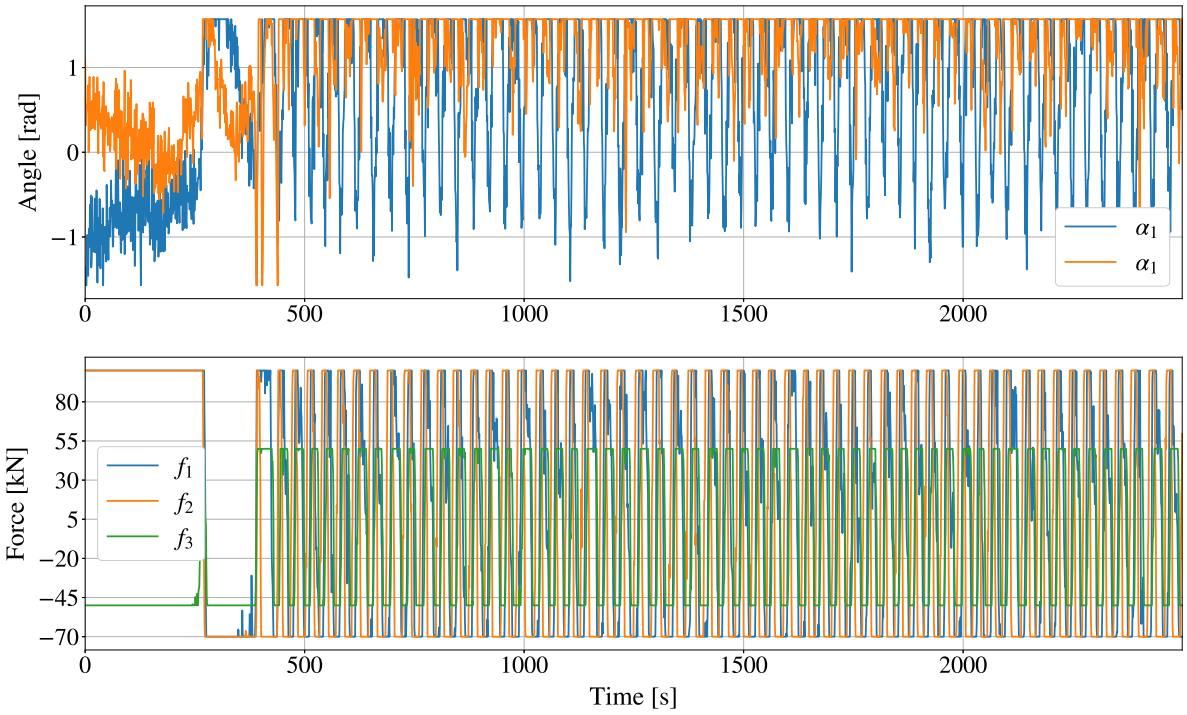


(b) Penalty + extended state vector

Figure 4.45: Reward plots for the LP4:Extended state and LP4:Added penalty agents. Test episodes starts with initial state $\boldsymbol{\eta}$ =[345 m, 760m, -0.1 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].



(a) LP4:Distane berthing, without state



(b) LP4:Distance berthing, with extra states

Figure 4.46: Thruster plots for the LP4:Extended state and LP4:Added penalty agents. Test episodes starts with initial state $\boldsymbol{\eta}$ =[345 m, 760m, -0.1 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s].

Chapter 5

Future work

Designing and implementing deep reinforcement learning for the docking scenario have had it's challenges. This section will discuss some of the challenges faced and possible future work based on this thesis.

5.1 Simulation of a harbour and vessel

In this thesis, a simplified harbour environment was implemented. This was primarily to decrease the complexity of the problem and limit the training time. Future work would probably be trying to perform docking in a more complex simulation. It was, for instance, discussed that the reason for the LP5:Docking solution experiencing problems could be due to unfortunate training scenarios. Factors could be insufficient space for proper exploration with speed limits and too many scenarios at high initial forward velocities, where the agent had little chance of avoiding crashing. A more complex simulation could, for instance, solve this problem by providing sufficient space for proper exploration with speed limits and taking precautions to avoid unrealistic initial values. This would relieve the agent from experiencing unnecessary "traumas" during learning, due to penalties received.

A more complex simulator could also take into account the dynamics of the thrusters, such as delays and inertias, that would both pose new challenges and optionally affect the intensive control input usage by imposing natural low-pass filters.

5.2 Design of state vector

The state space is an important factor in how the agent perceives the world, but what should be on the mind when designing it? Through this thesis, different combinations of states were tested, and the design of states was crucial to the results. The most challenging state to design was representing information about obstacles. This was performed by giving the relative distance to the closest obstacle, calculated using relatively simple geometry. In the real world, the information of obstacles would most likely be given differently, which would lead to a different design of those parts of the state vector. The information could, for instance, be given by using simultaneous localisation and mapping (SLAM), where the DRL-agent could receive information from SLAM with an estimation of an obstacle map and the vessel's state in it [138].

Another possibility is giving the DRL agent directly data from sensors, such as LIDAR or RADAR in the cases of obstacle avoidance data. Data from LIDAR has, for instance, proven to work well for DRL agents to extract useful information concerning obstacle avoidance [122]. It has also been shown that a DRL agent could be upgraded to use convolution neural networks (CNNs) to more efficiently generalise the environmental state using LIDAR or RADAR data [139].

Future work could also be to upgrade the state information given to the DRL agent and for instance, using techniques such as CNNs to support to use of sensors, possibly be paving the way for testing the DRL docking model in real life.

5.3 Design of reward function and selection of DRL-algorithms

Through the learning phases LP1:DP, LP2:Berthing, LP3:Target tracking and LP4:Distance berthing without current, the environment seemed to be working pretty well, with the vessel efficiently reaching the target with high accuracy. For the learning phases LP5:Docking and LP4:Distance berthing with reduction of aggressive thruster (LP4:Added penalty and LP4:Extended state), some complications were experienced.

One of the complications for LP4:Docking and LP4:Distance berthing with reduction of aggressive thruster was the aversion towards touching the quay close to the berth. When analysing the failing scenarios, the agents were punished for touching the quay even at very low velocities. With docking in real life, the vessel would have low velocities during berthing and indeed touch the quay, but not get any damage due to the use of fenders and modern vessel construction allowing a certain force towards the hull without damage to it

[137]. The reward function could be improved accordingly, for instance by designing the reward function so that the agent would not receive a penalty if the vessel makes contact with the quay at sufficiently low speeds.

The reward component r_u for reducing the forward speed when entering the harbour could be redesigned to achieve a better result. The reward component could, for instance, start to give a penalty for having high velocities when the vessel is getting closer to the harbour.

The DRL algorithms tested in this thesis, PPO and DDPG, have a relatively small likelihood of solving a sparse reward function. Through shaping the reward function, they were more likely to solve the task. Reward shaping does, however, often demand expert domain knowledge, and would otherwise represent an misinterpretation of the real objective at hand. Through designing the reward function for the docking tasks, it was experienced that shaping the multi-objective problem into one scalar was difficult, especially for the later learning phases. Meeting all criteria simultaneously and selecting weighting of different objectives, is a classical multi-objective optimisation challenge. It could, therefore, be interesting to explore DRL algorithms capable of handling sparse-reward functions. One instance of this is DDPG with an extension of hindsight experience replay (HER) [140]. Hindsight experience replay allows sample-efficient learning from sparse reward functions, meaning less or even no need for complicated reward engineering. Another alternative is testing algorithms created especially for handling multi-objective problems such as multi-objective reward functions [141] and hierarchical reinforcement learning [142].

For reducing the use of the control inputs, further research could be performed into constrained policy optimisation [143]. Using constrained policy optimisation, specific constraints on the use of the thrusters can be posed, so that the agent always could give satisfactory use of thrusters.

5.4 Explainable AI

This thesis used techniques from explainable AI to try to interpret the decisions made by the agent. Using SHAP provided a value signifying the contribution of states to a given action. However, the values will need to be analysed using domain knowledge to gain more advantage of these interpretations. It would, therefore, be interesting in future work to analyse these SHAP-values with a marine engineer, to gain even more insight into the DRL-agents inherent logic, and possible shortcomings due to simulators, algorithms and reward structures.

One important aspect of SHAP is whether the explanation model is capturing the actual

inherent logic of the model. From the results it was observed that the explanation model was able to capture the original DRL model quite well, see Appendix, D.1 , D.3, D.4. From the literature search, it was not found a clear answer as to when one has reached an explanation model which captures the intent of the original DRL model. One method could, for instance, be to perform sensitivity analysis on the background data, to see if the explanation model changed under certain circumstances. In this thesis, the background data was sampled uniformly from the environment under certain preconditions. It was explored with different volume and instances in the background dataset. However, when the data was generated under the same preconditions, the explanations did not change significantly. From the given background dataset the explanation model for LP4:Distance berthing appeared to be logical for most of the analysis.

It was also performed experiments with sampling data from the environment under different preconditions, such as higher values of velocities v and r. It was seen that the SHAP-values changed significantly when increasing v and r. This shows that the background data should be carefully selected. In this thesis, it was done by setting preliminaries which should represent the average values of the environment. A perhaps better way of doing this would be to sample states from when the DRL agent solved docking scenarios. This would mean that the SHAP values would be created based on potentially more similar states.

In addition to the initiatives mentioned above, it would be interesting to:

- Compare the explanations of DDPG and PPO in learning phases were both methods worked, looking for possible similarities or differences of behaviour.
- Compare the explanations of the DRL models in scenarios where the agent failed the assignment, such as the instance when it came into contact with quay. This could give insight into what should be changed to avoid these behaviours.
- Try to use different explainable AI techniques to make explanation models and see how they differ. For instance, creating a more transparent RL-agent [71].

Chapter 6

Conclusion

This thesis has investigated how to use deep reinforcement learning (DRL) to create an end-to-end docking model for a fully-actuated autonomous surface vessel in a harbour. This was solved using a progressive methodology, consisting of five learning phases. The end product is an agent capable of controlling the vessel from just outside the harbour to a berth inside the harbour.

The first two phases consisted of creating an agent capable of performing berthing. The result of the third learning phase was an agent performing target tracking, from right outside the harbour to a point close to the berth. These steps were solved using both deterministic policy gradient (DDPG) and proximal policy optimisation (PPO). The solutions of learning phase one to three using PPO exhibited good performances overall, being able to perform target tracking, dynamic positioning and obstacle avoidance with high accuracy. It was found that PPO proved best fitted for the docking task in this thesis, finding the solutions faster and more reliably than DDPG. Proximal policy optimisation was, therefore, the only DRL algorithm used in the last two learning phases. SHAP values were used to evaluate insufficient training in learning phase two with berthing. It showed that that the vessel needed to be trained with a quay at both starboard and port side relative to the berth, to give a general solution. This insight helped to correct the agent's behaviour in learning phases four and five.

The fourth phase combined the ideas from the target tracking and berthing-phases to create an agent performing berthing from larger distances. This agent achieved good performance and could withstand a constant current as long as the agent was started far enough away from the berth, to allow the vessel to adapt to the circumstances. Using SHAP, the relative contribution of state information to the behaviour of the agent was shown to correspond well with what may be defined as general expectations of the situation.

In the fifth learning phase, the agent should steer the vessel from just outside the harbour to the berth and should obey the speed limits inside the harbour. The agent was able to solve the task in some scenarios, but not in all of them. This is most likely due to the insufficient space of the harbour for a vessel entering at higher velocities. Analysing the behaviour of the agent using explainable AI, the agent appeared to exhibit less logical reasoning than in the fourth learning phase, giving support to the assertion that the scenario proved overly difficult or complex.

The simulation results indicate that the DRL-based controllers can provide good performance, with high accuracy and efficiency, in several tasks of docking. Using SHAP on the policy of the DRL-agent, showed promising results by giving intuitive insights into the reasoning of the DRL created control polices. This was an early trial of using SHAP to explain the DRL agents reasoning, and more work remains to facilitate easier use and consistency. More work is also needed to successfully implement speed constraints, to better withstand ocean currents and to reduce the number of rapid changes in the control inputs.

References

[1] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. CoRR. 2015;abs/1509.0.

[2] Ray A, Askell A, Garfinkel B, Dennison C, Devin C, Zeigler D. Ray A, editor. Spinning Up. OpenAI; 2018. Visited on 2019-09-03. Available from: https://spinningup.openai.com/en/latest/index.html.

[3] Martinsen AB, Lekkas AM, Gros S. Autonomous docking using direct optimal control. IFAC PapersOnLine. 2019;52:97–102.

[4] Martinsen AB, Lekkas AM. Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning. IFAC-PapersOnLine. 2018;51(29):329–334.

[5] Martinsen AB, Lekkas AM. Curved Path Following with Deep Reinforcement Learning: Results from Three Vessel Models. Institute of Electrical and Electronics Engineers (IEEE); 2018. Available from: http://hdl.handle.net/11250/2596760.

[6] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. CoRR. 2016;abs/1603.0. Available from: http://arxiv.org/abs/1603.04467.

[7] Oliphant T. NumPy : A guide to NumPy. Trelgol Publishing; 2019. Visited on 2019-11-14. Available from: http://www.numpy.org/.

[8] Hunter JD. Matplotlib: A 2D graphics environment. Computing in Science & Engineering. 2007;9(3):90–95.

[9] Lundberg S. Explainers — SHAP latest documentation. Read the Docs, Inc; 2018. Visited on 2019-12-03. Available from: https://shap.readthedocs.io/en/latest/.

[10] McKinney W. Data Structures for Statistical Computing in Python. In: Proceedings of the 9th Python in Science Conference,; 2010. p. 51–56. Available from: http://conference.scipy.org/proceedings/scipy2010/mckinney.html.

[11] Ltd J. draw.io. JGraph Ltd; 2019. Visited on 2019-08-28. Available from: https://about.draw.io/.

[12] Marine RRC. Rolls-Royce and Finferries demonstrate world's first Fully Autonomous Ferry. Rolls-Royce Holdings plc; 2018. Visited on 2019-10-21. Available from: https://www.rolls-royce.com/media/press-releases/2018/03-12-2018-rr-and-finferries-demonstrate-worlds-first-fully-autonomous-ferry.aspx.

[13] Skredderberget A. The first ever zero emission, autonomous ship. Yara International ASA; 2018. Visited on 2019-09-15. Available from: https://www.yara.com/knowledge-grows/game-changer-for-the-environment/.

[14] Cowan D. Autonomous ships: are they the future of the seven seas? Raconteur Media Ltd.; 2018. Visited on 2019-08-25. Available from: https://www.raconteur.net/finance/autonomous-ships.

[15] Fosen J. Maritime autonomous surface ships on the horizon. Gard AS; 2019. Visited on 2019-11-20. Available from: http://www.gard.no/web/updates/content/27107214/maritime-autonomous-surface-ships-on-the-horizon.

[16] Engineering RAo. Innovation in autonomous systems; 2015. Visited on 2019-09-30. Available from: https://www.raeng.org.uk/autonomoussystems.

[17] Club TS. The Standard for service and security; 2018. Visited on 2019-10-09. Available from: https://www.standard-club.com.

[18] Kusslid KT. Et havnekraftig Trøndelag Strategiplan 2019-2030. Trondheim havn; 2019. Visited on 2019-11-17. Available from: https://trondheimhavn.no/wp-content/uploads/2019/06/trondheim-havn-strategi-2030.pdf.

[19] veritas B. NI641 Guidelines for autonomous shipping | Marine And Offshore. Bureau veritas; 2018. Visited on 2019-11-24. Available from: https://marine-offshore.bureauveritas.com/ni641-guidelines-autonomous-shipping.

[20] Demirel E, Bayer D. The Further Studies On The COLREGs (Collision Regulations). TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation. 2015;9(1):17–22. Available from: http://www.transnav.eu/Article_The_Further_Studies_On_The_COLREGs_Demirel,33,551.html.

[21] Kretschmann L, Burmeister HC, Jahn C. Analyzing the economic benefit of unmanned autonomous ships: An exploratory cost-comparison between an autonomous and a conventional bulk carrier. Research in Transportation Business and Management. 2017 12;25:76–86.

[22] Kim DG, Hirayama K, Okimoto T. Ship Collision Avoidance by Distributed Tabu Search. TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation. 2015 4;9(1):23–29.

[23] for Maritime Logistics FC, Services. Research in Maritime Autonomous Systems Project Results and Technology Potentials. European Union and Fraunhofer CML and Marintek and Aptimar AS and Chalmers and Hochshule Wismar and Marine Soft and Marokka and UCC; 2016. Visited on 2019-10-02. Available from: www.unmanned-ship.org.

[24] Marina VI. The Differences Between Anchoring, Mooring & Docking - Van Isle Marina. Van Isle Marina; 2019. Visited on 2019-08-29. Available from: https://vanislemarina.com/anchoring-mooring-docking/.

[25] Magheralex. Mooring, berthing, docking and picking up a buoy - MarinaReservation.com. MarinaReservation.com Ltd.; 2018. Visited on 2019-09-05. Available from: https://www.marinareservation.com/articles/mooring-docking-and-berthing/.

[26] org H. Port information guide. International harbour masters association, International association of ports and harbours, UK admiralty, International task force port call optimization, Sea traffic management; 2019. Visited on 2019-11-04. Available from: https://portcalloptimization.org/images/Port%20Information%20Guide%201.1%20-%20clean.pdf.

[27] Nguyen VS. Research on a support system for automatic ship navigation in fairway. Future Internet. 2019 2;11(2).

[28] Im NK, Nguyen VS. Artificial neural network controller for automatic ship berthing using head-up coordinate system. International Journal of Naval Architecture and Ocean Engineering. 2018 5;10(3):235–249.

[29] Bui VP, Kawai H, Kim YB, Lee KS. A ship berthing system design with four tug boats. Journal of Mechanical Science and Technology. 2011 5;25(5):1257–1264.

[30] Sørensen AJ. Marine Control Systems Propulsion and Motion Control of Ships and Ocean Structures Lecture Notes; 2013. Visited on 2019-03-03.

[31] Murdoc E, Dand IW, Clarke C. A Masters Guide to Berthing 2nd edition. Standard house; 2012. Visited on 2019-11-16. Available from: https://www.academia.edu/34141248/A_Masters_Guide_to_Berthing_2nd_edition.pdf.

[32] Djouani K, Hamam Y. Minimum Time-Energy Trajectory Planning for Automatic Ship Berthing. IEEE Journal of Oceanic Engineering. 1995;20(1):4–12.

[33] Tran VL, Im N. A study on ship automatic berthing with assistance of auxiliary devices. International Journal of Naval Architecture and Ocean Engineering. 2012 9;4(3):199–210. Available from: https://linkinghub.elsevier.com/retrieve/pii/S2092678216303429.

[34] Sorfonn I, Holmlund-Sund M. World's first Autodocking installation successfully tested by Wärtsilä. Wärtsilä Corporation; 2018. Visited on 2019-09-03. Available from: https://www.wartsila.com/media/news/26-04-2018-world-s-first-autodocking-installation-successfully-tested-by-wartsila-2169290.

[35] Breivik M, Loberg JE. A virtual target-based underway docking procedure for unmanned surface vehicles B. In: IFAC Proceedings Volumes (IFAC-PapersOnline). vol. 44. IFAC Secretariat; 2011. p. 13630–13635.

[36] Mizuno N, Saka N, Katayama T. A Ship's Automatic Maneuvering System Using Optimal Preview Sliding Mode Controller with Adaptation Mechanism. IFAC-PapersOnLine. 2016;49(23):576–581.

[37] Mizuno N, Uchida Y, Okazaki T. Quasi real-time optimal control scheme for automatic berthing. In: IFAC-PapersOnLine. vol. 28; 2015. p. 305–312.

[38] Fossen TI. Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd; 2011.

[39] Rae GJS, Smith SM. A fuzzy rule based docking procedure for autonomous underwater vehicles. In: OCEANS 1992 - Proceedings: Mastering the Oceans Through Technology. vol. 2. Institute of Electrical and Electronics Engineers Inc.; 1992. p. 539–546.

[40] Teo K, Goh B, Chai OK. Fuzzy Docking Guidance Using Augmented Navigation System on an AUV. IEEE Journal of Oceanic Engineering. 2015 4;40(2):349–361.

[41] Zhang Y, Hearn GE, Sen P. A Multivariable Neural Controller for Automatic Ship Berthing. IEEE Control Systems. 1997;17(4):31–45.

[42] Shuai Y, Li G, Cheng X, Skulstad R, Xu J, Liu H, et al. An efficient neural-network based approach to automatic ship docking. Ocean Engineering. 2019 11;191.

[43] Kalaiselvam S, Parameshwaran R. Kalaiselvam S, Parameshwaran R, editors. Fuzzy-Logic Control - an overview (pdf) | ScienceDirect Topics. Boston: Academic Press; 2014. Available from: https://www.sciencedirect.com/topics/engineering/fuzzy-logic-control/pdf.

[44] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–444.

[45] Russell S, Norvig P. Artificial Intelligence: A Modern Approach. 3rd ed. Pearson; 2010.

[46] Sutton RS, Barto AG. Reinforcement Learning - An introduction. The MIT Press; 2018.

[47] Bertsekas DP. Reinforcement Learning and Optimal Control. Massachusetts Institute of Technology; 2019. Visited on 2019-09-25. Available from: https://web.mit.edu/dimitrib/www/RL_MONOGRAPH1.pdf.

[48] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks. 2015 1;61:85–117.

[49] Goodfellow I, Bengio Y, Courville A. Deep Learning. Adaptive computation and machine learning. Cambridge, Mass: MIT Press; 2017.

[50] François-Lavet V, Henderson P, Islam R, Bellemare MG, Pineau J. An Introduction to Deep Reinforcement Learning. Foundations and Trends®in Machine Learning. 2018;11(3-4):219–354.

[51] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning. 2013 12;Available from: http://arxiv.org/abs/1312.5602.

[52] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015 2;518(7540):529–533.

[53] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016 1;529(7587):484–489.

[54] Gandhi D, Pinto L, Gupta A. Learning to Fly by Crashing. arXivorg. 2017;Available from: http://search.proquest.com/docview/2074098729/.

[55] You Y, Pan X, Wang Z, Lu C. Virtual to Real Reinforcement Learning for Autonomous Driving. CoRR. 2017;abs/1704.0. Available from: http://arxiv.org/abs/1704.03952.

[56] Deng Y, Bao F, Kong Y, Ren Z, Dai Q. Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. {IEEE} Transactions on Neural Networks and Learning Systems. 2017 3;28(3):653–664.

[57] Tuyen LP, Layek A, Vien NA, Chung T. Deep reinforcement learning algorithms for steering an underactuated ship. In: 2017 {IEEE} International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI). IEEE; 2017. .

[58] Zhuang Y, Gu Q, Wang B, Luo J, Zhang H, Liu W. Robust Auto-parking: Reinforcement Learning based Real-time Planning Approach with Domain Template; 2018.

[59] Zhang P, Xiong L, Yu Z, Fang P, Yan S, Yao J, et al. Reinforcement Learning-Based End-to-End Parking for Automatic Parking System. Sensors. 2019 9;19(18):3996. Available from: https://www.mdpi.com/1424-8220/19/18/3996.

[60] Morrison S, Fisher A, Zambetta F. Towards Intelligent Aircraft Through Deep Reinforcement Learning. RMIT University; 2018.

[61] Wang Z, Li H, Wu H, Shen F, Lu R. Design of Agent Training Environment for Aircraft Landing Guidance Based on Deep Reinforcement Learning. In: Proceedings - 2018 11th International Symposium on Computational Intelligence and Design, ISCID 2018. vol. 2. Institute of Electrical and Electronics Engineers Inc.; 2018. p. 76–79.

[62] Anderlini E, Parker GG, Thomas G. Docking Control of an Autonomous Underwa-ter
 Vehicle Using Reinforcement Learning. Applied Sciences. 2019 8;9(17):3456. Available from: https://www.mdpi.com/2076-3417/9/17/3456.

[63] Amendola J, Tannuri EA, Cozman FG, Reali AH. Batch Reinforcement Learning of Feasible Trajectories in a Ship Maneuvering Simulator. Sociedade Brasileira de Computacao - SB; 2019. p. 263–274.

[64] Cheng Y, Zhang W. Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. Neurocomputing. 2018 1;272:63–73.

[65] Shen H, Hashimoto H, Matsuda A, Taniguchi Y, Terada D, Guo C. Automatic collision avoidance of multiple ships based on deep Q-learning. Applied Ocean Research. 2019 5;86:268–288.

[66] Bitar G, Martinsen AB, Lekkas AM, Breivik M. MPC-based Automatic Docking for ASVs with Full-Scale Experiments. In: Submitted to IFAC CAMS 2020; 2020. .

[67] Taylor ME, Stone P. Transfer Learning for Reinforcement Learning Domains: A Survey. J Mach Learn Res. 2009;10:1633–1685. Available from: http://dl.acm.org/citation.cfm?id=1577069.1755839.

[68] Dosilovic FK, Brcic M, Hlupic N. Explainable artificial intelligence: A survey. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings. Institute of Electrical and Electronics Engineers Inc.; 2018. p. 210–215.

[69] Hayes B, Shah JA. Improving Robot Controller Transparency Through Autonomous Policy Explanation. 2017 Mar;2:134–144.

[70] Van Der Waa J, Van Diggelen J, Van Den Bosch K, Neerincx M. Contrastive explanations for reinforcement learning in terms of expected consequences. Ithaca; 2018. Available from: http://search.proquest.com/docview/2092800288/.

[71] Lee JH. Complementary reinforcement learning towards explainable agents. Ithaca; 2019. Available from: http://search.proquest.com/docview/2162922094/.

[72] Adamson L, Brown N. IMO takes first steps to address autonomous ships. International maritime organization; 2018. Visited on 2019-08-17. Available from: http://www.imo.org/en/MediaCentre/PressBriefings/Pages/08-MSC-99-MASS-scoping.aspx.

[73] og fiskeridepartementet N. Forskrift om forebygging av sammenstøt på sjøen (Sjøveisreglene) - Lovdata. Lovdata; 1975. Visited on 2019-10-23. Available from: https://lovdata.no/dokument/SF/forskrift/1975-12-01-5/.

[74] og beredskapsdepartementet J. Lov om sjøfarten (sjøloven) - Lovdata. Lovdata; 2019. Visited on 2019-11-07. Available from: https://lovdata.no/dokument/NL/lov/1994-06-24-39.

[75] Kartverket. Den norske los. Kartverket; 2019. Visited on 2019-11-16. Available from: https://www.kartverket.no/Kart/Nautiske-hjelpemidler/Den-norske-los/.

[76] maritime organization I. Convention on the International Regulations for Preventing Collisions at Sea, 1972 (COLREGs). International maritime organization; 1977. Visited on 2019-08-10. Available from: http://www.imo.org/en/About/Conventions/ListOfConventions/Pages/COLREG.aspx.

[77] Organization IM. Guidelines for Voyage Planning; 2000. Visited on 2019-11-15. Available from: http://www.imo.org/en/KnowledgeCentre/IndexofIMOResolutions/Assembly/Documents/A.893(21).pdf.

[78] maritime organization I. International Convention for the Safety of Life at Sea (SOLAS), 1974. International maritime organization; 1974. Visited on 2019-10-16. Available from: http://www.imo.org/en/About/Conventions/ListOfConventions/Pages/International-Convention-for-the-Safety-of-Life-at-Sea-(SOLAS),-1974.aspx.

[79] Solas Chapter V - Regulation 34 - Safe navigation and avoidance of dangerous situations. International maritime organization; 1974. Available from: https://mcanet.mcga.gov.uk/public/c4/solas/solas_v/Regulations/regulation34.htm.

[80] Fagerdal SO. Email exchange with harbour captain at Trondheim harbour;.

[81] Transverse Tunnel Thrusters fincantieri / marine systems and components. Fincantieri marine systems and components; 2014. Available from: https://www.fincantieri.com/globalassets/prodotti-servizi/sistemi-e-componenti/sistemi-e-componenti-navali/transverse-tunnel-thrusters_mp-05-14.pdf.

[82] Sinha T. An Introduction to Tunnel Thrusters in Ships - Design and Application. Marine insight; 2019. Visited on 2019-09-14. Available from: https://www.marineinsight.com/naval-architecture/introduction-to-tunnel-thrusters-ships/.

[83] Mitchell TM. Machine Learning. 1st ed. McGraw-Hill Education; 1999.

[84] Priddy KL, Keller PE. Artificial neural networks an introduction. SPIE; 2005.

[85] Galante L, Banisch R. A Comparative Evaluation of Anomaly Detection Techniques on Multivariate Time Series Data. 2019 01;.

[86] Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov RR. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. 2014;15:1929–1958.

[87] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXivorg. 2015;Available from: http://search.proquest.com/docview/2081710836/.

[88] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016;p. 770–778.

[89] Belleter DJW, Paliotta C, Maggiore M, Pettersen KY. Path Following for Underactuated Marine Vessels. FAC-PapersOnLine. 2016;49(18):588–593.

[90] Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA. A Brief Survey of Deep Reinforcement Learning. CoRR. 2017;abs/1708.0.

[91] Tangkaratt V, Abdolmaleki A, Sugiyama M. Guide Actor-Critic for Continuous Control. In: International Conference on Learning Representations; 2018. Available from: https://openreview.net/forum?id=BJk59JZ0b.

[92] Watkins CJCH, Dayan P. Technical Note: Q-learning. Machine Learning. 1992;8(3/4):279–292.

[93] Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning. 1992;8(3-4):229–256.

[94] Sutton RS, Mcallester DA, Singh SP, Mansour Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: Solla SA, Leen TK, Müller KR, editors. Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]. The MIT Press; 1999. p. 1057–1063.

[95] Konda VR, Tsitsiklis JN. On Actor-Critic Algorithms. SIAM J Control Optim. 2003 4;42(4):1143–1166. Available from: https://doi.org/10.1137/S0363012901385691.

[96] Peters J, Schaal S. Natural Actor-Critic. Neurocomputing. 2008 3;71(7-9):1180–1190.

[97] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust Region Policy Optimization. CoRR. 2015;abs/1502.0. Available from: http://arxiv.org/abs/1502.05477.

[98] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. CoRR. 2017;abs/1707.0. Available from: http://arxiv.org/abs/1707.06347.

[99] Gu S, Lillicrap TP, Sutskever I, Levine S. Continuous Deep Q-Learning with Model-based Acceleration. CoRR. 2016;abs/1603.0. Available from: http://arxiv.org/abs/1603.00748.

[100] Amos B, Xu L, Kolter JZ. Input Convex Neural Networks. In: Precup D, Teh YW, editors. Proceedings of the 34th International Conference on Machine Learning. vol. 70 of Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR; 2017. p. 146–155. Available from: http://proceedings.mlr.press/v70/amos17b.html.

[101] Konda V. Actor-critic Algorithms. Cambridge, MA, USA; 2002.

[102] Silver D, Heess N, Degris T, Wierstra D, Riedmiller M. Deterministic Policy Gradient Algorithms; 2014.

[103] Maxim Lapan. Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q ... - Maxim Lapan - Google Books. Packt Publishing; 2018.

[104] Kakade S, Kakade S, Langford J. Approximately Optimal Approximate Reinforcement Learning. In proeedings 19th international conference on machine learning. 2002;p. 267–274. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.7.7601.

[105] Hunter DR, Lange K. A Tutorial on MM Algorithms. American Statistician. 2004;58(1):30–37.

[106] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust Region Policy Optimization. 2015 2;Available from: http://arxiv.org/abs/1502.05477.

[107] Achiam J, Abbeel P. Proximal Policy Optimization — Spinning Up documentation. OpenAI; 2018. Visited on 2019-10-19. Available from: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[108] Schulman J, Moritz P, Levine S, Jordan MI, Abbeel P. High-dimensional continuous control using generalized advantage estimation. In: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings. International Conference on Learning Representations, ICLR; 2016. .

[109] Riedmiller M, Hafner R, Lampe T, Neunert M, Degrave J, Van De Wiele T, et al. Learning by Playing-Solving Sparse Reward Tasks from Scratch. Ithaca; 2018. Available from: http://search.proquest.com/docview/2071685952/.

[110] Wiewiora E. Reward Shaping. In: Encyclopedia of Machine Learning. Springer US; 2010. p. 863–865.

[111] Zou H, Ren T, Yan D, Su H, Zhu J. Reward Shaping via Meta-Learning. 2019 1;Available from: http://arxiv.org/abs/1901.09330.

[112] Lundberg SM, Allen PG, Lee SI. A Unified Approach to Interpreting Model Predictions;. Visited on 2019-12-05. Available from: https://github.com/slundberg/shap.

[113] Ribeiro MT, Singh S, Guestrin C. "Why should i trust you?" Explaining the predictions of any classifier. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. vol. 13-17-August-2016. Association for Computing Machinery; 2016. p. 1135–1144.

[114] Lipton ZC. The Mythos of Model Interpretability. Communications of the ACM. 2018;61(10):36–43.

[115] Arrieta AB, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, et al. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. 2019 10;Available from: http://arxiv.org/abs/1910.10045.

[116] Molnar C. Interpretable Machine Learning. Christoph Molnar; 2019. Visited on 2019-01-08. Available from: https://christophm.github.io/interpretable-ml-book/.

[117] Shapley L. In: A value for n-person games. vol. 28; 1983. p. 307–317.

[118] Molnar C. Interpretable Machine Learning; 2019. Visited on 2019-12-05. Available from: https://christophm.github.io/interpretable-ml-book/cite.html.

[119] Lundberg SM, Erion GG, Lee S. Consistent Individualized Feature Attribution for Tree Ensembles. CoRR. 2018;abs/1802.03888. Available from: http://arxiv.org/abs/1802.03888.

[120] Institute NM. Yr – Havvarsel for 63,44180 10,36698 63,44180 10,36698, Hav (sted). YR Joint weather online; 2019. Visited on 2019-11-16. Available from: https://www.yr.no/sted/Hav/63,44180_10,36698/.

[121] Younhee Lee, Woong Lim. Shoelace Formula: Connecting the Area of a Polygon with Vector Cross Product. The Mathematics Teacher. 2017;110(8):631.

[122] Manuelli L, Florence P. Reinforcement Learning for Autonomous Driving Obstacle Avoidance using LIDAR;. Visited on 2019-09-03. Available from: http://www.peteflorence.com/ReinforcementLearningAutonomousDriving.pdf.

[123] Lenart A. Approach Parameters in Marine Navigation – Graphical Interpretations. TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation. 2017 10;11(3):521–529.

[124] Hostetler R, Larson R. Precalculus with Limits. Charlie VanWagner; 2011. Available from: https://www.amazon.com/Precalculus-Limits-Hostetler-Robert-2007-11-08/dp/B01N8YECC8.

[125] Mahmood AR, Korenkevych D, Vasan G, Ma W, Bergstra J. Benchmarking Reinforcement Learning Algorithms on Real-World Robots; 2018. Available from: https://github.com/kindredresearch/SenseAct.

[126] Zhao L, Roh MI. COLREGs-compliant multiship collision avoidance based on deep reinforcement learning. Ocean Engineering. 2019 11;191.

[127] Bhatt A, Argus M, Amiranashvili A, Brox T. CrossNorm: On Normalization for Off-Policy TD Reinforcement Learning. Ithaca; 2019. Available from: http://search.proquest.com/docview/2182960211/.

[128] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. CoRR. 2015;abs/1412.6.

[129] Baselines S. stable_baselines.ppo2.ppo2 — Stable Baselines 2.10.0a0 documentation. Read the Docs; 2019. Visited on 2019-11-15. Available from:

https://stable-baselines.readthedocs.io/en/master/_modules/stable_baselines/ppo2/ppo2.html.

[130] Mahmood AR, Korenkevych D, Vasan G, Ma W, Bergstra J. Benchmarking Reinforcement Learning Algorithms on Real-World Robots;. Visited on 2019-12-11. Available from: https://github.com/kindredresearch/SenseAct.

[131] Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D. Deep Reinforcement Learning that Matters. Ithaca; 2019. Available from: www.aaai.org.

[132] Islam R, Henderson P, Gomrokchi M, Precup D. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. arXivorg. 2017;Available from: http://search.proquest.com/docview/2075689963/.

[133] LONZA A. Reinforcement learning algorithms with python. PACKT Publishing Limited; 2019.

[134] Fujimoto S, Van Hoof H, Meger D. Addressing Function Approximation Error in Actor-Critic Methods; 2018.

[135] Liessner R, Schmitt J, Dietermann A, Bäker B. Hyperparameter optimization for deep reinforcement learning in vehicle energy management. In: ICAART 2019 - Proceedings of the 11th International Conference on Agents and Artificial Intelligence. vol. 2. SciTePress; 2019. p. 134–144.

[136] Matheron G, Perrin N, Sigaud O. The problem with DDPG: understanding failures in deterministic environments with sparse rewards. arXivorg. 2019;Available from: http://search.proquest.com/docview/2319067568/.

[137] Design Manual Shibata Fender. ShibataFenderTeam AG, Germany; 2019. Available from: https://www.shibata-fender.team/files/content/Downloads/SFT-Design-Manual-A4-English-2019.pdf.

[138] Mustafa KA, Botteghi N, Sirmaçek B, Poel M, Stramigioli S. Towards continuous control for mobile robot navigation: A reinforcement learning and slam based approach; 2019. .

[139] Lei X, Zhang Z, Dong P. Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning. Journal of Robotics. 2018 09;2018:1–10.

[140] Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, et al. Hindsight experience replay. vol. 2017-. Neural information processing systems foundation; 2017. p. 5049–5059.

[141] Friedman E, Fontaine F. Generalizing Across Multi-Objective Reward Functions in Deep Reinforcement Learning. CoRR. 2018;abs/1809.0.

[142] Osa T, Tangkaratt V, Sugiyama M. Hierarchical Reinforcement Learning via Advantage-Weighted Information Maximization. CoRR. 2019;abs/1901.0.

[143] Achiam J, Held D, Tamar A, Abbeel P. Constrained Policy Optimization. In: ICML; 2017. .

Appendix A

Constants in the simulation of the vessel and harbour

A.1 Constants of vessel dynamics

The container ship used in this project was selected from a paper about autonomous docking [3]. The vessel consisted of:

- Inertia matrix **M**, (A.1)
- Dampening matrix **D**, (A.2)
- Thruster allocation $\boldsymbol{\tau}(\boldsymbol{\alpha}, \mathbf{f})$, (A.3)

$$\mathbf{M} = \begin{bmatrix} 6.767e+03 & 0. & 0. \\ 0. & 1.135e+04 & -3.403e+04 \\ 0. & -3.403e+04 & 4.445e+06 \end{bmatrix} \tag{A.1}$$

$$\mathbf{D} = \begin{bmatrix} 7.707e+01 & 0. & 0. \\ 0. & 2.55e+02 & -2.034e+03 \\ 0. & -6.726e+02 & 3.850e+05 \end{bmatrix} \tag{A.2}$$

$$\boldsymbol{\tau}(\boldsymbol{\alpha}, \mathbf{f}) = \begin{bmatrix} \cos(\alpha_1) & \cos(\alpha_2) & 0 \\ \sin(\alpha_1) & \sin(\alpha_2) & 1 \\ -35\sin(\alpha_1) - 7\cos(\alpha_1)) & -35\sin(\alpha_2) + 7\cos(\alpha_2)) & 35 \end{bmatrix} \mathbf{f} \tag{A.3}$$

A.2 Constants of the vessel shape set

The shape of the container ship used in this thesis was expressed using the following constants:

- $\mathbf{A}_b$, (A.4)
- $\mathbf{b}_b$, (A.5)

$$\mathbf{A}_b = \begin{bmatrix} -1 & 0 \\ 2.72 & -1. \\ -2.72 & -1. \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \tag{A.4}$$

$$\mathbf{b}_b = \begin{bmatrix} -7.7 \\ 41.91 \\ 41.91 \\ 7.7 \\ -41.91 \end{bmatrix} \tag{A.5}$$

A.3 Constants of docking area set

The shape of the docking area used in this thesis was expressed using the following constants:

- $\mathbf{A}_s$, (A.6)
- $\mathbf{b}_s$, (A.7)

$$\mathbf{A}_s = \begin{bmatrix} -8.57 & -1. \\ 0 & -1. \\ -0.52 & -1. \\ -2.77 & -1. \\ 0 & -1 \end{bmatrix} \tag{A.6}$$

$$\mathbf{b}_s = \begin{bmatrix} 5163.85 \\ 1242.0 \\ 1503.81 \\ 2846.56 \\ 120.0 \end{bmatrix} \tag{A.7}$$

A.4 Constants of harbour set

The shape of the harbour used in this thesis was expressed using the following constants:

- $\mathbf{A}_h$, (A.8)
- $\mathbf{b}_h$, (A.9)

$$\mathbf{A}_h = \begin{bmatrix} -8.64 & -1. \\ 0 & -1. \\ -0.52 & -1. \\ -2.76 & -1. \\ 0 & -1 \end{bmatrix} \tag{A.8}$$

$$\mathbf{b}_h = \begin{bmatrix} 5195.87 \\ 1242.0 \\ 1503.81 \\ 2839.07 \\ 120 \\ 500 \end{bmatrix} \tag{A.9}$$

Appendix B

Calculate area of polygon from vertexes

The area of a vessel inside a target-rectangle was calculated using the Shoelase algorithm, explained in [121]. The following Python implementation was used:

```
def create_corners(x,y):
    corners = []
    for i in range(len(x)):
        corners.append((y.item(i),x.item(i)))

    return corners

def sort_corners(corner_list):
    n = len(corner_list)
    cx = float(sum(x for x, y in corner_list)) / n
    cy = float(sum(y for x, y in corner_list)) / n
    corner_angle_list = []
    for x, y in corners:
        an = ((np.arctan2(y - cy, x - cx) + 2.0 * np.pi)
                % (2.0 * np.pi))
        corner_angle_list.append((x, y, an))
    corner_angle_list.sort(key = lambda tup: tup[2])

    return [ (x, y) for (x, y,an) in corner_angle_list]

def calculate_polygon_area(corner_list_sorted):
```

```
    n = len(corner_list_sorted)
    area = 0.0
    for i in range(n):
        j = (i + 1) % n
        area += (corner_list_sorted[i][0] *
                 corner_list_sorted[j][1])
        area -= (corner_list_sorted[j][0] *
                 corner_list_sorted[i][1])
    area = abs(area) / 2.0
    return area

def get_vessel_area(x_b_n, y_b_n):
    corner_list = create_corners(x_b_n, y_b_n)
    corner_list_sorted = sort_corners(corner_list)
    area = calculate_polygon_area(corner_list_sorted)
    return area
```

Appendix C

Additional plots of the learning phases

C.1 The LP3:Target tracking DDPG agent

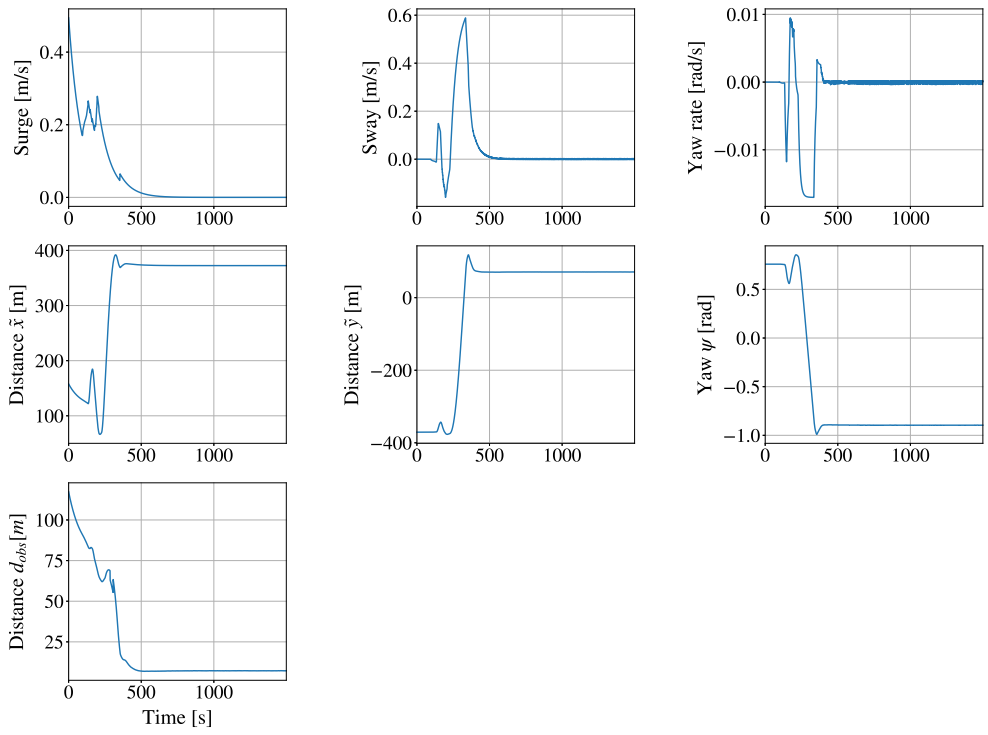


Figure C.1: State plots for LP3:Target tracking DDPG agent. Test episodes started with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad].

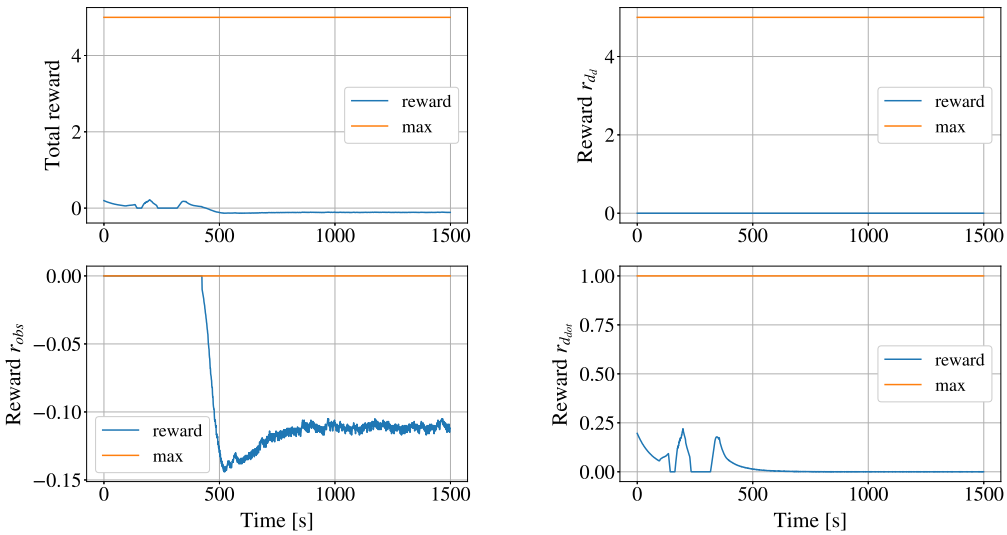


Figure C.2: Reward plots for LP3:Target tracking DDPG agent. Test episodes started with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad]

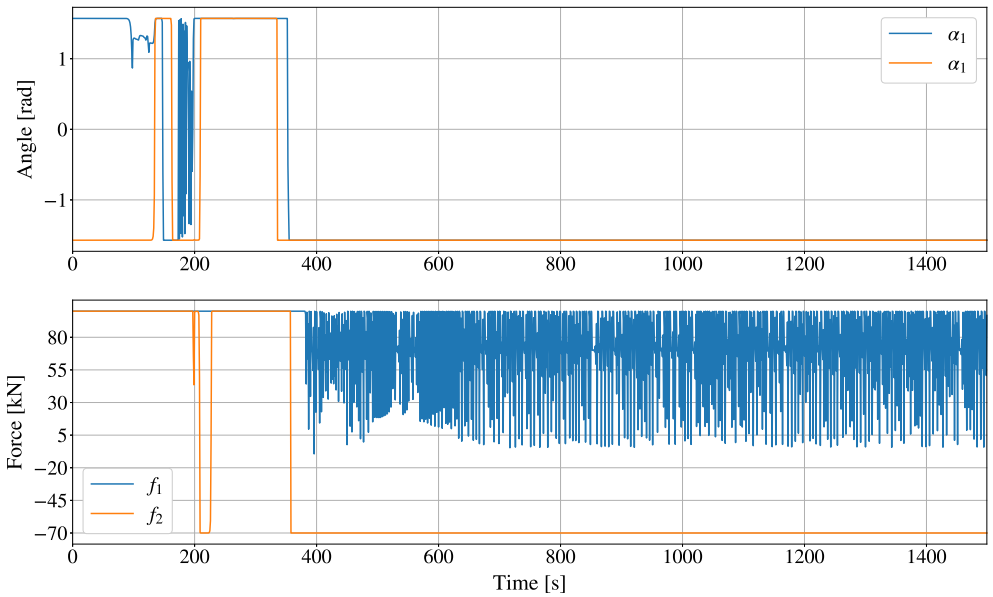
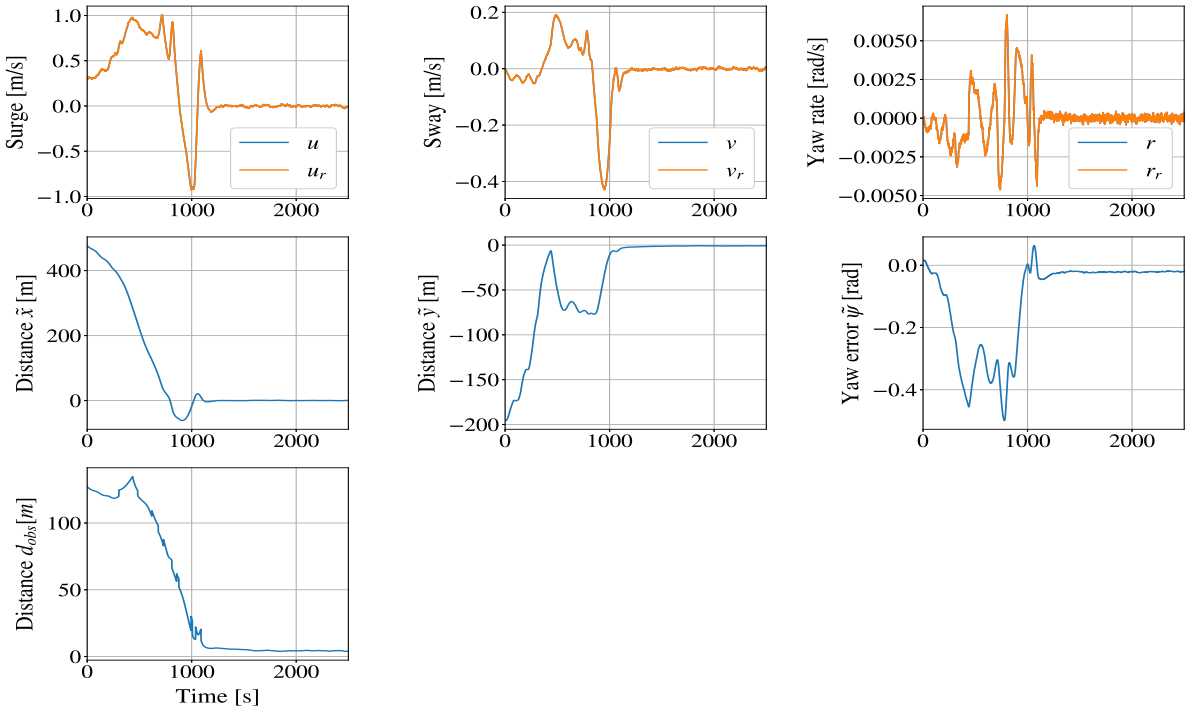
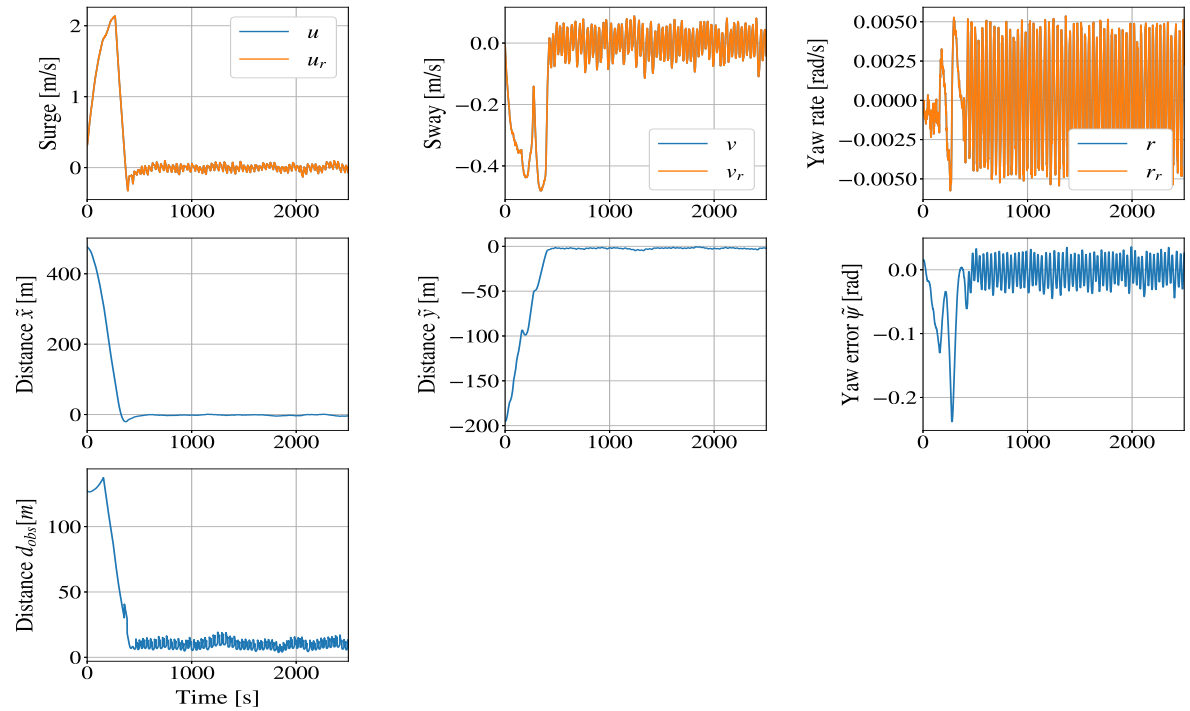


Figure C.3: Thruster plots for LP3:Target tracking DDPG agent. Test episodes started with initial state $\boldsymbol{\eta}$ =[330 m,760 m,0.759 rad] and $\boldsymbol{\nu}$ = [0.5 m/s,0 m/s,0 rad]

C.2 Additional plot of LP4:Add penalty and LP4:



(a) LP4:Extended state



(b) LP4:Added penalty

Figure C.4: State plots for the LP4:Extended state and LP4:Added penalty agents. Test episodes started with initial state $\boldsymbol{\eta}$ =[345 m, 760m, -0.1 rad] and $\boldsymbol{\nu}$ = [0.3 m/s, 0 m/s, 0 rad/s]

Appendix D

Additional plots from SHAP analysis

D.1 The LP2-η:Berthing agent trained on one side

The explanation model used to calculate the general contribution of states to the LP2-η:Berthing agent trained to berth on port side, was able to capture the DRL-agents policy with an root-mean-squared-error (RMSE) of: $f_1 = 54.2651486494$ kN, $f_2 = 48.2704491108$ kN, $f_3 = 26.7010037629$ kN, $\alpha_1 = 0.784122432178$ radians and $\alpha_2 = 0.871529764973$ radians. The explanation model vs. the original LP2-η:Berthing agent trained to berth on port side, is plotted for the 400 test instances in Figure D.1.

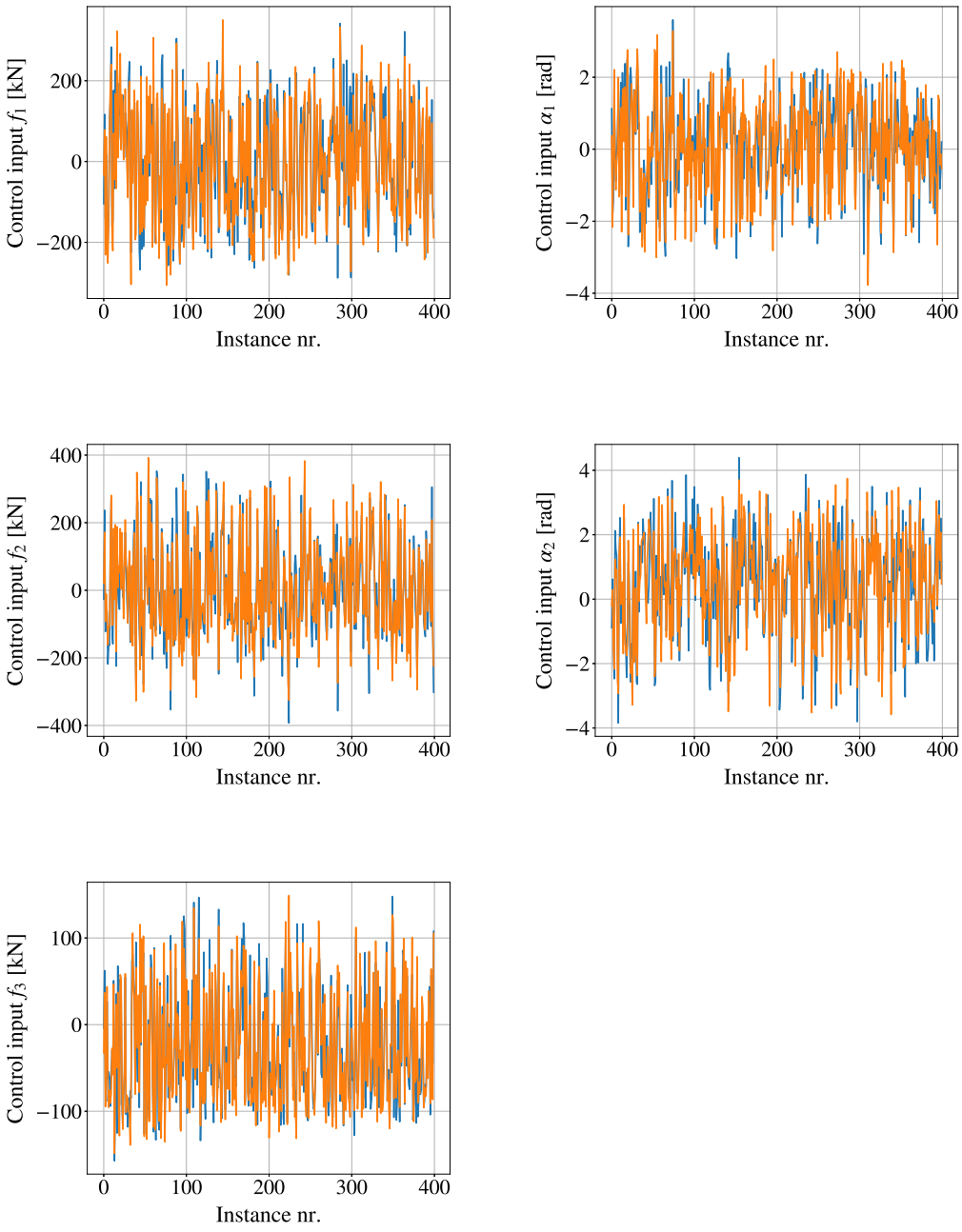


Figure D.1: The explanation model vs. the LP2-η:Berthing agent trained to berth on port side.

D.2 The LP2-η:Berthing agent trained on two sides

The explanation model used to calculate the general contribution of states to the LP2-η:Berthing agent trained to berth on port and starboard side, was able to capture the DRL-agents policy with an root-mean-squared-error (RMSE) of: $f_1 = 31.2992311969$ kN, $f_2 = 47.7426127612$ kN, $f_3 = 17.842803762$ kN, $\alpha_1 = 0.57080957131$ radians and $\alpha_2 = 0.658779061736$ radians. The explanation model vs. the original LP2-η:Berthing agent trained to berth on port and starboard side, is plotted for the 400 test instances in Figure D.2.

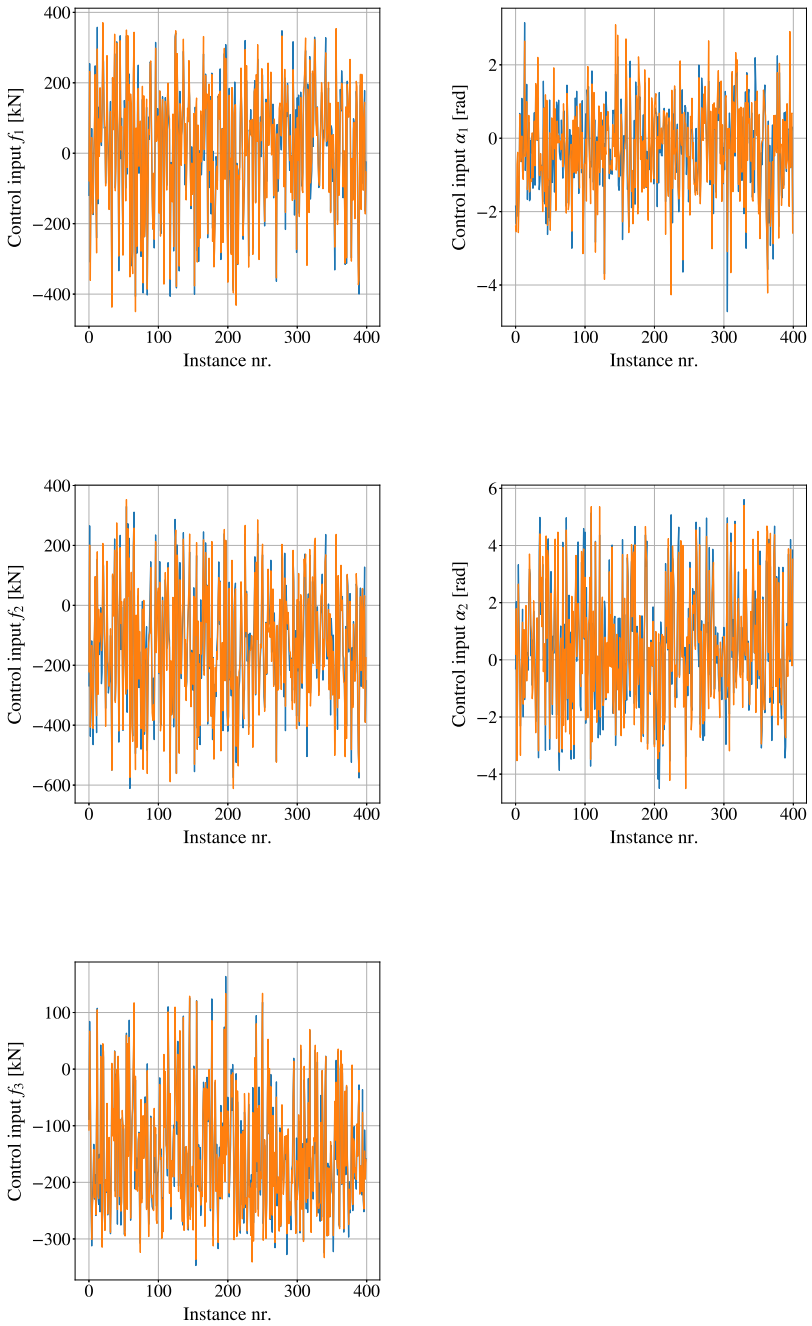


Figure D.2: The explanation model vs. original the LP2-η:Berthing agent trained to berth on port and starboard side

D.3 The LP4:Distance berthing agent

The explanation model used to calculate the general contribution of states to the LP4:Distance berthing agent was able to capture the DRL-agents policy with an root-mean-squared-error (RMSE) of: $f_1 = 14.6159067371$ kN, $f_2 = 15.956755779$ kN, $f_3 = 9.67568454281$ kN, $\alpha_1 = 0.27189706152$ radians and $\alpha_2 = 0.29576505413$ radians. The explanation model vs. the original LP4:Distance berthing agent is plotted for the 400 test instances in Figure D.3.

The explanation model used to calculate the contributions of states to LP4:Distance berthing agent for one episode, was able to capture the DRL-agents policy with an root-mean-squared-error (RMSE) of: $f_1 = 14.9353847924$ kN, $f_2 = 15.4361509312$ kN, $f_3 = 9.9043901805$ kN, $\alpha_1 = 0.252529376413$ radians and $\alpha_2 = 0.28632913474$ radians. The explanation model vs. the original LP4:Distance berthing agent is plotted for one episode in Figure D.4.

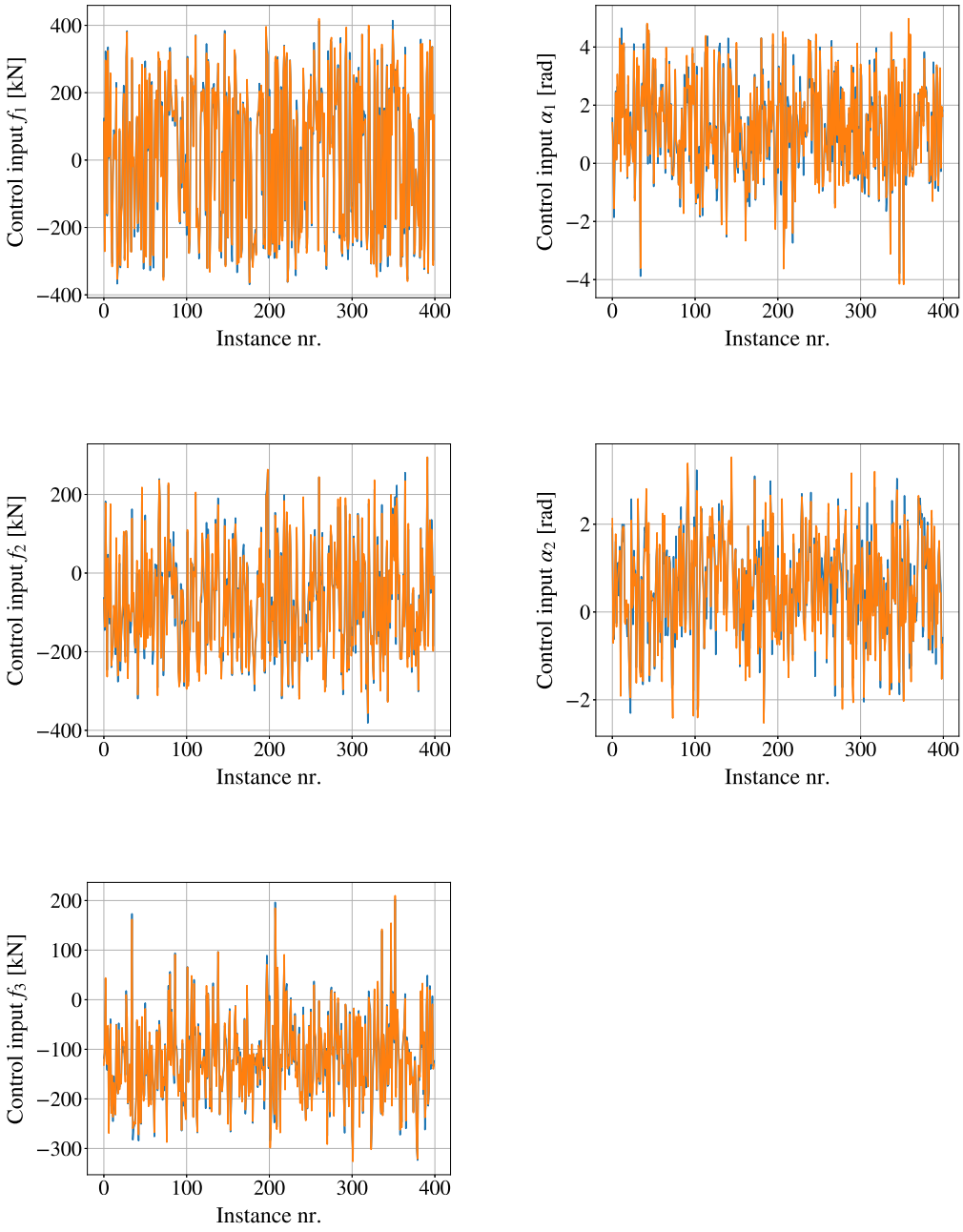


Figure D.3: The explanation model used to calculate the general contribution of states vs. the LP4:Distance berthing agent.

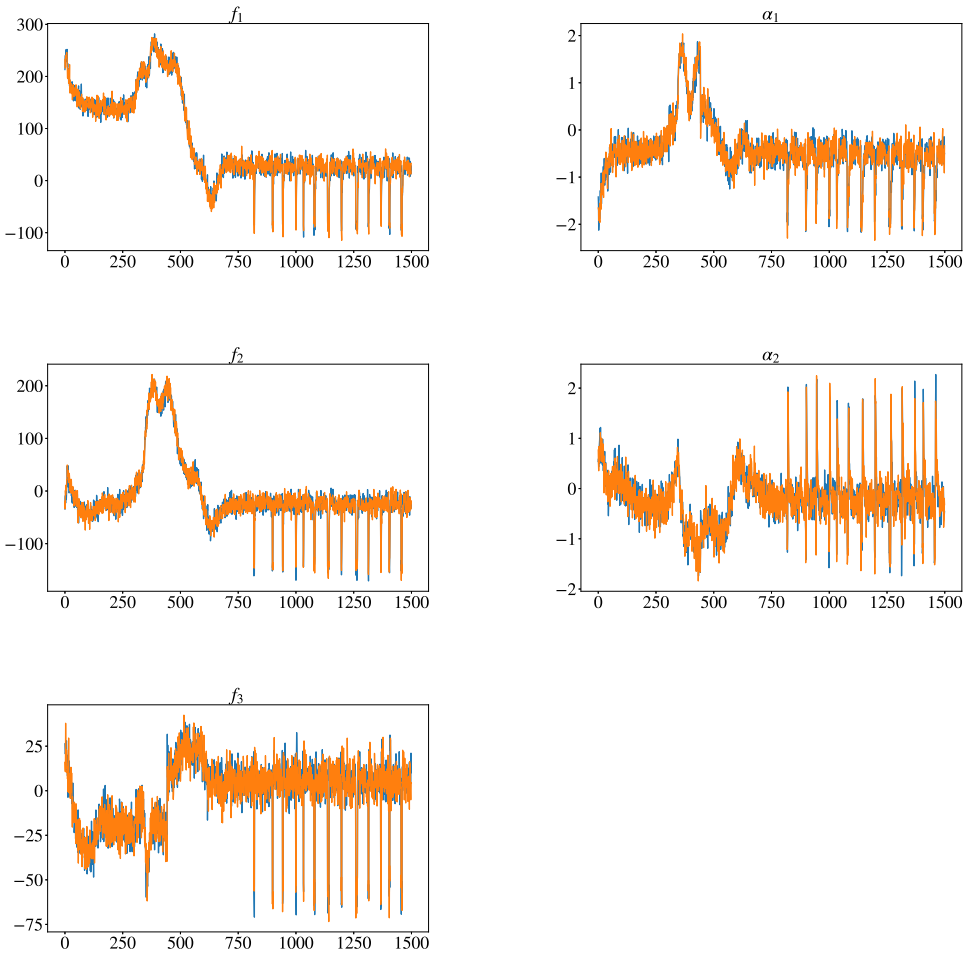


Figure D.4: The explanation model used to calculate the contributions of states for one episode vs LP4:Distance berthing agent

D.4 The LP5:Docking agent

The explanation model used to calculate the general contribution of states to the LP5:Docking was able to capture the DRL-agents policy with an root-mean-squared-error (RMSE) of: $f_1 = 29.085796383847814$ kN, $f_2 = 18.04912958025381$ kN, $f_3 = 13.811696642205655$ kN, $\alpha_1 = 0.4410659028660526$ radians and $\alpha_2 = 0.48624768452652267$ radians. The explanation model vs. the original LP5:Docking agent is plotted for the 400 test instances in Figure D.5.

The explanation model used to calculate the contributions of states to LP4:Distance berthing agent for one episode, was able to capture the DRL-agents policy with an root-mean-squared-error (RMSE) of: $f_1 = 29.494548978169792$ kN, $f_2 = 18.122132598683525$ kN, $f_3 = 14.107190017007362$ kN, $\alpha_1 = 0.45039695297377574$ radians and $\alpha_2 = 0.4873803052306718$ radians. The explanation model vs. the original LP5:Docking agent is plotted for the docking episode in Figure D.6. The development for SHAP values during the docking episode is visualised in Figure D.7.

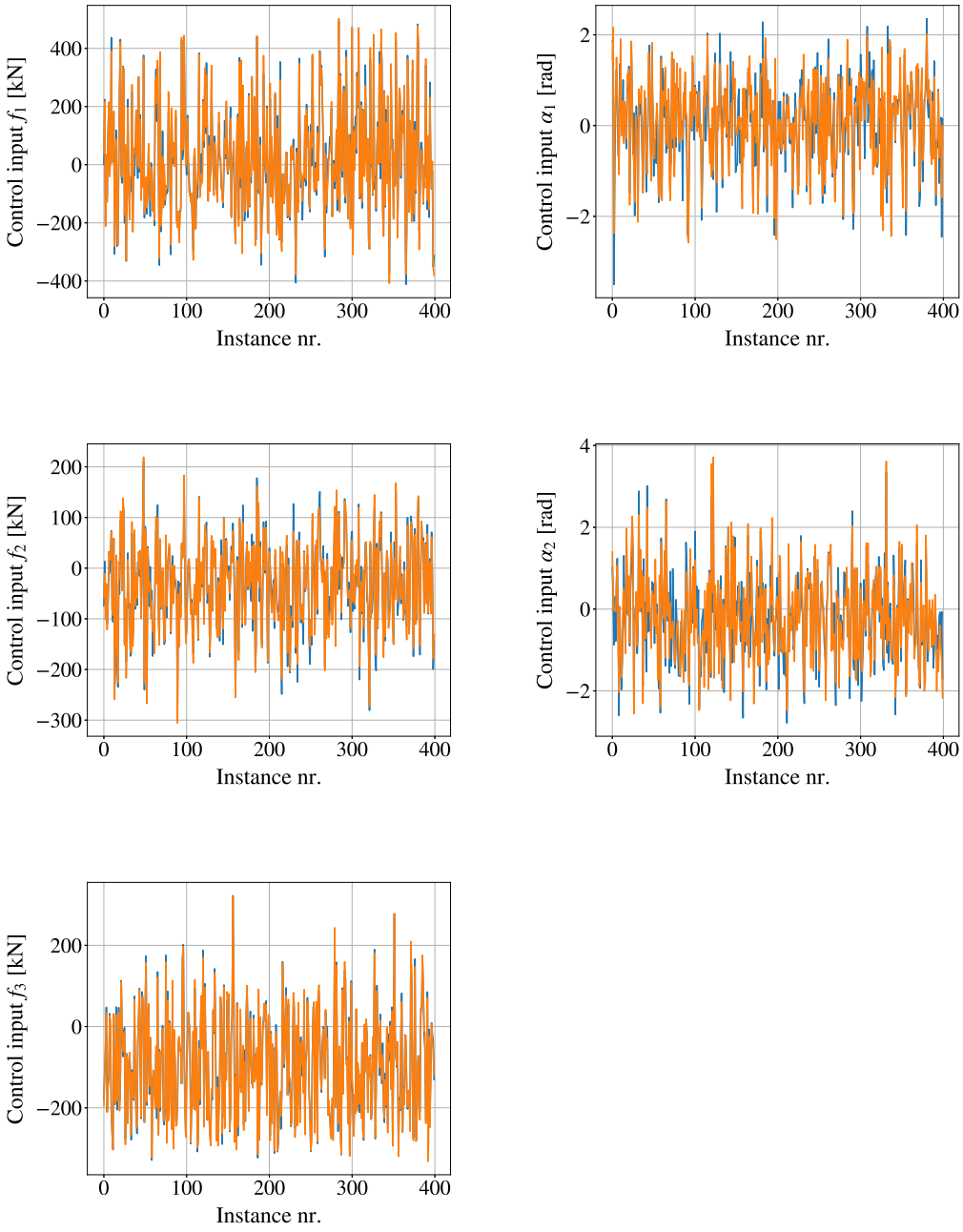


Figure D.5: The explanation model used to calculate the general contribution of states vs. the LP5:Docking agent.

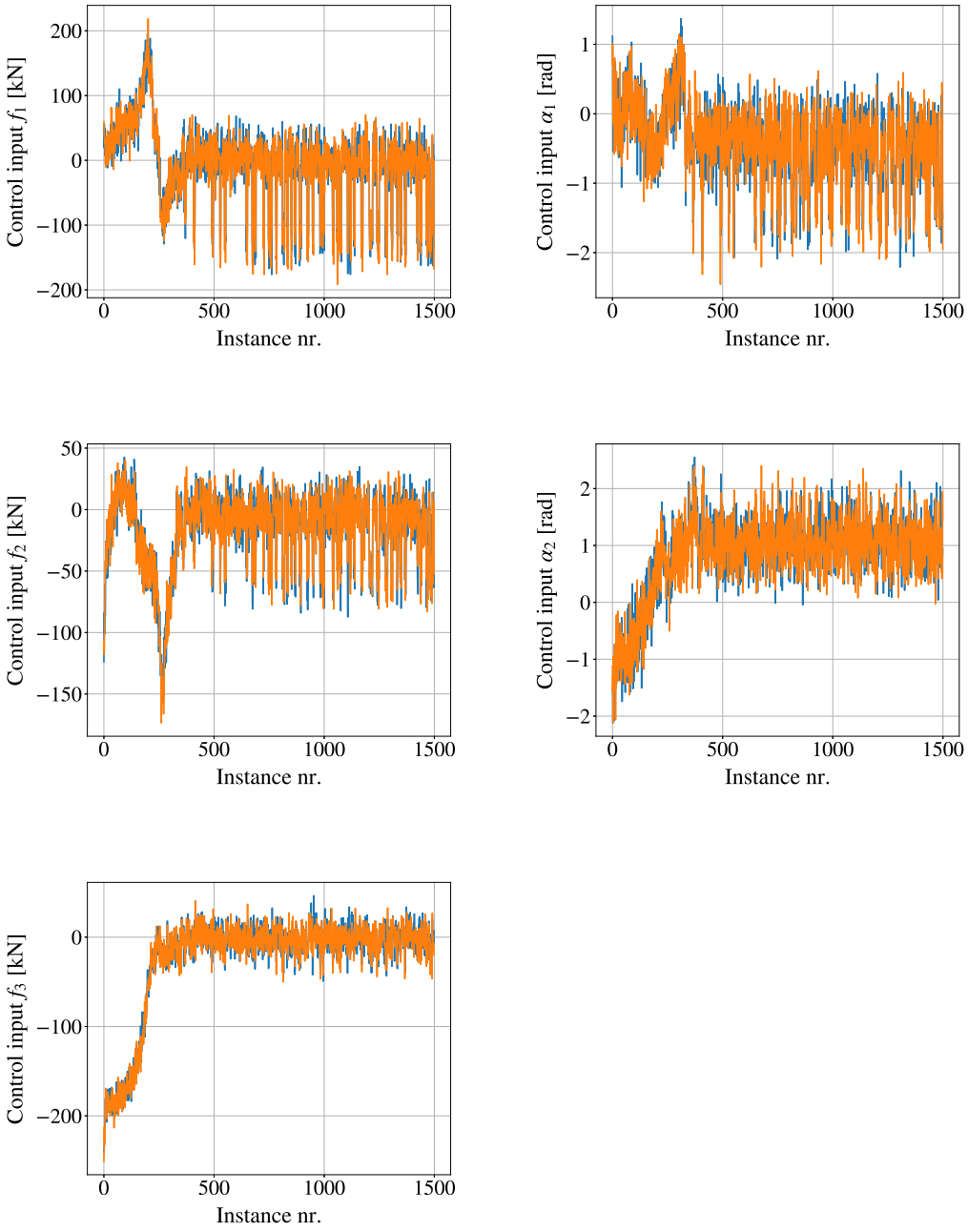


Figure D.6: The explanation model used to calculate the contributions of states for one episode vs LP5:Docking agent

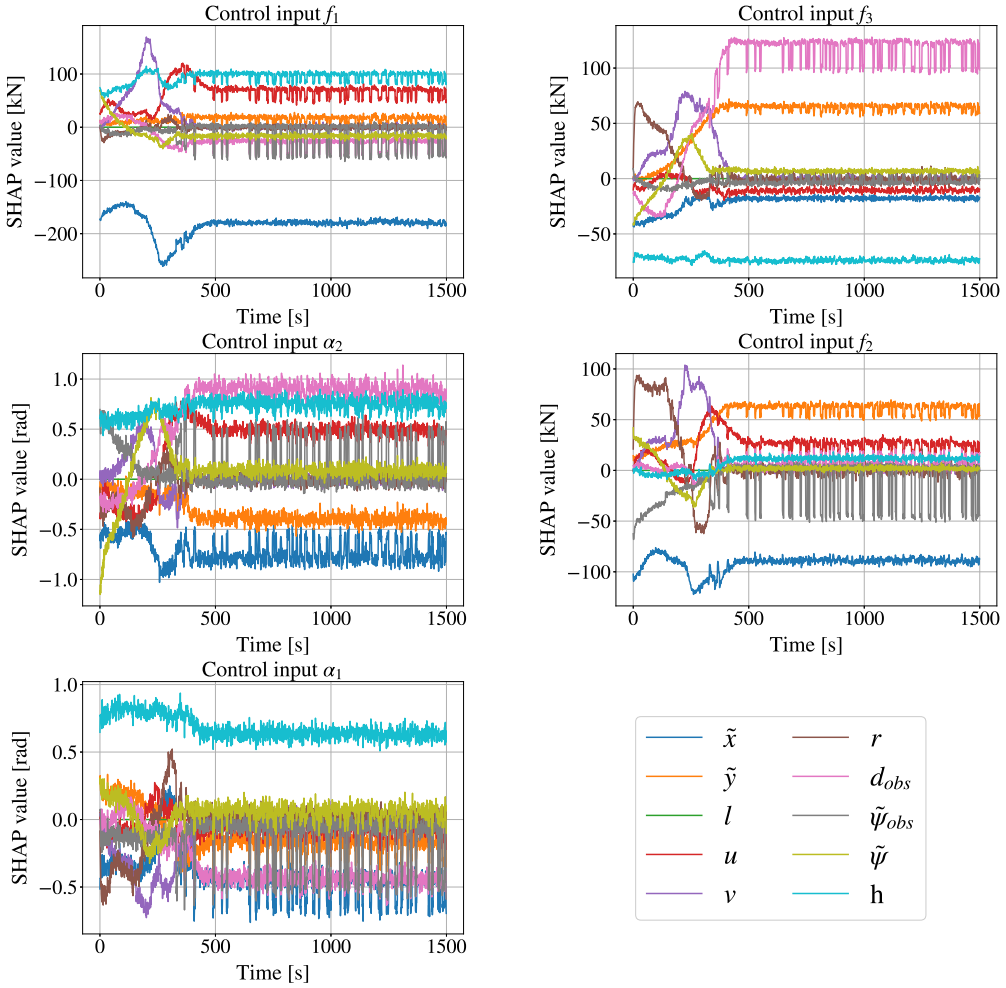


Figure D.7: SHAP values of each state to each thruster force $[f_1, f_2, f_3]$ and angle $[\alpha_1, \alpha_2]$ from one test episode of LP5:Docking agent

Ella-Lovise Hammervold Rørvik

Automatic Docking of an Autonomous Surface Vessel

NTNU
Norwegian University of
Science and Technology