Aurora Opheim Sauar

Explainable Artificial Intelligence for the docking of a marine vessel using SHAP and LIME values

Specialization project
Department of Engineering Cybernetics
Norwegian University of Science and Technology

Masteroppgave i Kybernetikk og Robotikk
Veileder: Anastasios Lekkas
Februar 2023

Aurora Opheim Sauar

Explainable Artificial Intelligence for the docking of a marine vessel using SHAP and LIME values

Specialization project
Department of Engineering Cybernetics
Norwegian University of Science and Technology

Masteroppgave i Kybernetikk og Robotikk
Veileder: Anastasios Lekkas
Februar 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

1.5em

Explainable Artificial Intelligence for the docking of a marine vessel using SHAP and LIME values

Author:
Aurora Opheim Sauar

Supervisor:
Anastasios Lekkas

Specialization project
Department of Engineering Cybernetics
Norwegian University of Science and Technology

February 24, 2023

Preface

This master's thesis is the culmination of a master's degree program in Cybernetics and Robotics at NTNU, Trondheim. It was written during the fall of 2022 and is a follow-up to a master project completed in the spring of the same year. The thesis covers relevant background theory and provides a brief introduction to artificial intelligence and ship dynamics. The methods employed to solve the docking of a marine vessel are explained in detail, and the code used for the project was executed on a Mac Book Air using Visual Studio Code and the PyTorch library.

Working on this thesis has been both challenging and rewarding. It has been a rewarding process to see the code work after hours of debugging, and interesting to gain insights into the automation of ships and reinforcement learning.

Acknowledgements

I would like to acknowledge several stakeholders who have contributed to this thesis. First and foremost, I am grateful to my supervisor, Professor Anastasios Lekkas, for guiding me in choosing the topic and providing regular feedback and valuable insights on relevant theory and methods. I am also thankful to the Ph.D candidates, Vilde Gjærum and Sindre Benjamin Remman, for their assistance in debugging, theoretical discussions and continuous interest in my work.

Furthermore, I am grateful to the OpenAI chatbot, ChatGPT, for providing valuable assistance in correcting my writing during this master thesis. The ability to receive immediate feedback and suggestions has greatly improved the quality of my work.

Finally, I would like to acknowledge the role of UKA and Samfundet, volunteer organizations in Trondheim, in providing me with opportunities to develop my skills in communication, prioritization, and efficiency. Their support and encouragement have made my time in Trondheim not only productive, but also enjoyable and fulfilling.

Executive summary

Autonomous docking of ships has been a focus of research for decades, with the first automatic docking system developed in the 1960s. Advances in technology have led to increasingly sophisticated systems that can be tested using simulations and field trials to ensure safety and reliability. AI and Deep Reinforcement learning have emerged as powerful tools for developing autonomous docking systems that can adapt to changing environmental conditions.

However, it is crucial for the captain on a ship to understand how the AI system operates, especially in situations where its decision-making process could impact the safety of the vessel and crew. Explainable AI provides insights into the decision-making process of the system, ensuring transparency, understandability and increase the quality in the decision making process for a captain that considers to overrule the decision of an AI system. In the context of autonomous docking, Explainable AI is critical in identifying areas for improvement.

Explainable AI has its roots in machine learning, where algorithms focus on accurate predictions without providing an explanation for their decisions. The Explainable AI aims to ensure transparency and accountability, making it an important tool in developing autonomous docking systems. By using Explainable AI, researchers can understand how the system makes decisions.

The primary objective of this report is to investigate the feasibility of utilizing a proximal policy optimization algorithm to govern the docking maneuvers of a 3 Degree Of Freedom autonomous vessel. Additionally, the report endeavors to explicate the behavior of the system through the application of LIME and SHAP values. The proximal policy optimization algorithm is an effective algorithm for designing autonomous docking systems, while LIME and SHAP values offer a mechanism for elucidating the behavior of the system. In this study, two distinct models were examined and assessed for performance, and their behavior were analyzed through the use of SHAP and LIME values. Furthermore, one of the models was trained using different learning rates. The observation vector encompassed nine parameters, which were employed by the reward function to yield a reward, while the action vector comprised five parameters for controlling the two azimuth thrusters and the one tunnel thruster on the vessel.

By using a proximal policy optimization algorithm and Explainable AI techniques such as LIME and SHAP values, it is possible to develop highly effective autonomous docking systems that can be used in a wide range of applications. This study found the Proximal policy algorithm to be effective for a 3 Degree Of Freedom autonomous ship. In addition, it provided insight into the feature attribution of each the states for each of the controlling actions. This is useful for developing autonomous docking systems that are safe, reliable,

and transparent.

It is important to emphasize the need for further work to fully exploit the potential of Explainable AI for the docking of a marine vessel. While these techniques can provide insight into how an autonomous docking system makes decisions, it is still necessary to investigate how this insight can be transferred to practical use. Further research and development of Explainable AI will be critical to developing autonomous docking systems that are even more reliable and safe, and that can be used in increasingly demanding environments. For example, additional work is required to improve the robustness of the autonomous docking system, which involves implementing ocean currents, wind, and moving obstacles. Additionally, it may be interesting to further investigate Counterfactuals, as this technique can provide a more personal and qualitative explanation of decisions and help users understand how to modify input variables to achieve the desired output variable.

Sammendrag

Autonom dokking av skip har vært et forskningsområde i flere tiår, og det første automatiske dokkingssystemet ble utviklet på 1960-tallet. Fremskritt innen teknologi har ført til stadig mer sofistikerte systemer, som kan testes ved hjelp av simuleringer og feltprøver for å sikre trygghet og pålitelighet. Kunstig intelligens (KI) og dyp forsterkende læring er kraftige og viktige verktøy for å utvikle autonome dokkingssystemer som kan tilpasse seg endrede miljøforhold.

Det er imidlertid essensielt at kapteinen forstår hvordan KI-systemet fungerer, spesielt i situasjoner der beslutningsprosessen kan påvirke skipets og mannskapets sikkerhet. Forklarende KI gir innsikt i beslutningsprosessen til systemet og sikrer gjennomsiktighet og forståelighet. Dermed forstår kapteinen også bedre når hen skal overstyre KI-systemets beslutning. I sammenheng med autonom dokking er forklarende KI avgjørende for å identifisere områder for forbedring.

Forklarende KI har sitt opphav i maskinlæring, der algoritmer fokuserer på prediksjoner uten å gi en forklaring på sine beslutninger. Utviklingen av forklarende KI har som mål å sikre gjennomsiktighet og ansvarlighet, og gjør det til et viktig verktøy i utviklingen av autonom dokkingssystemer. Ved å bruke forklarende KI kan både forskere og kapteinen forstå hvordan systemet tar beslutninger.

Hovedmålet med denne oppgaven er å undersøke bruken av en proximal policy optimization-algoritme for å styre et autonomt skip med 3 frihetsgrader under dokking. I tillegg tar rapporten sikte på å forklare systemets atferd gjennom anvendelse av LIME og SHAP-verdier. Proximal policy optimization-algoritmen er en effektiv algoritme for å utforme autonome dokkingssystemer, mens LIME- og SHAP-verdiene tilbyr en mekanisme for å forklare systemets atferd. I denne studien ble to forskjellige modeller undersøkt og vurdert for ytelse, og deres atferd ble analysert ved hjelp av SHAP- og LIME-verdier. Videre ble en av modellene trent ved hjelp av ulike læringsrater. Observasjonsvektoren omfattet ni parametere, som ble brukt av belønningsfunksjonen for å gi en belønning, mens handlingvektoren omfattet fem parametere for å kontrollere de to azimut-thrustere og en tunnel-thruster på fartøyet.

Ved å bruke en proximal policy optimization-algoritme og forklarende KI-teknikker som LIME og SHAP-verdier, er det mulig å utvikle svært effektive autonome styringssystemer for dokking som kan brukes i en rekke applikasjoner. Resultatene i denne oppgaven viser at proximal policy optimization-algoritmen er en effektiv algoritme for et dokking problem. I tillegg gir resultatene innsikt i atttribusjonen til hver av tilstandene for hver av kontrollhandlingene. Dette er nyttig for å utvikle autonome dokkingssystemer som er sikre, pålitelige og transparente.

Det er viktig å fremheve behovet for ytterligere arbeid for å fullt ut utnytte potensialet til forklarende KI for dokkingproblemet. Selv om disse teknikkene kan gi innsikt i hvordan et autonomt dokkingssystem tar beslutninger, er det fortsatt nødvendig å undersøke hvordan denne innsikten kan overføres til praktisk bruk. Videre forskning og utvikling av forklarende KI vil være avgjørende for å utvikle autonome dokkingssystemer som er enda mer pålitelige og sikre, og som kan brukes i stadig mer krevende miljøer. For eksempel kreves det ytterligere arbeid for å forbedre robustheten til det autonome dokkingssystemet, som omfatter implementering av havstrømmer, vind og bevegelige hindringer. Videre kan det også være interessant å undersøke Counterfactuals nærmere, ettersom teknikken kan gi en mer personlig og kvalitativ forklaring på beslutninger og hjelpe brukere med å forstå hvordan man kan endre input-variabler for å oppnå ønsket output-variabel.

Table of Contents

List of Tables

List of Figures

Abbreviations

Abbreviation	Description
ANN	Artificual Neural Networks
AI	Artificial intelligence
CP	Controllable-pitch propellers
DOF	Degrees of freedom
DRL	Deep reinforcement learning
FP	Fixed-pitch propellers
GNSS	Global Navigation Satellite System
HRL	Human-in-the-loop reinforcement learning
LIME	Local interpretable modelagnostic explanations
MDP	Markov Decision Process
NED	North-East-Down
PPO	Proximal policy optimization
RL	Reinforcement learning
SHAP	Shapley additive explanations
XAI	Explainable artificial intelligence

1 Introduction

The introduction chapter of this master's thesis serves as an overview of the research project and sets the stage for the subsequent chapters. The chapter begins by providing a background and motivation for the research, explaining why the topic is relevant and important. Next, the chapter provides an overview of past work that has been done in the field, highlighting key research studies and theoretical frameworks that have informed the research project. This include a review of relevant literature and an analysis of gaps in the existing research, identifying areas where further investigation is needed. The scope and objectives of the thesis are then presented, outlining the specific questions that the research project aims to answer. Finally, the chapter concludes with an overview of the thesis structure. The chapter consist of the following sections:

- Section 1.1: Background and motivation
- Section 1.2: Overview of past work
- Section 1.3: Scope and objective
- Section 1.4: Overview over thesis

1.1 Background and Motivation

The concept of intelligent robots and inanimate objects is not a recent idea, but has historical roots in ancient Greek, Chinese, and Egyptian civilizations (Russell and Norvig, 2010). However, the Dartmouth Summer Research Project on Artificial Intelligence in 1956 is widely regarded as the starting point for the modern development of AI (McCarthy et al., 1955). Prior to this, science fiction had already popularized the idea of artificial intelligence robots, such as the *Wizard of Oz* and the impersonated Maria in *Metropolis*, which influenced mathematicians, scientists, and philosophers to explore the possibility of AI in the real world (Nilsson, 2014)

In the early 1940s, the demand for new technology increased during the second world war. This led a British mathematician, Alan Turing, to explore the mathematical possibilities of artificial intelligence and suggest that computers should be able to make decisions by utilizing previous knowledge and available information, similar to how humans do (Anderson, 1987). However, there were several challenges at the time, including the high cost of computers, which were only available to prestigious universities and big technology companies, and the inability of early computers to store commands or information, a key requirement for Turing's proposed AI (Anyoha, 2017).

The next 50 years AI slowly became available for everyone to research as computers became cheaper, faster and more accessible. But, it was not until 1997 a chess playing computer program, IBM's *Deep Blue*, was able to defeat Gary Kasparov, the chess master at that time. At the same time *Windows* also released their first speech recognition software. Today, AI, is available for everyone and used everywhere. However, as AI become more and more complex, especially when working with Deep Neural Network, and usually presented as a black box, the transparency became a challenge. In several applications AI take on cognitive work with social dimensions which were previously performed by humans, and this raise several ethical challenges. For instance, imagine applying for a loan in a bank and being denied based on their AI algorithm. Then it would be very frustrating to get no reason behind the denial. This is the main motivation behind Explainable Artificial Intelligence (XAI) (Bossom and Yudkowsky, 2009).

As the field of AI received more attention, Robotics and Automation also became a subject of more interest. However, trust was an even more important factor. Using AI to control robots could have fatal outcomes in the case of any errors. For instance, imagine a vessel, controlled by AI, going to the berth. However, suddenly the AI turns the vessel to the right, without the captain knowing why, and, hence, making the vessel collide next to the berth and destroy the houses located by the wharf next door. Maybe the AI believed the bird to the left was a human swimming in the water, and, therefore, made the turn to the right or just some noise in the sensors. Without the explain ability no one would never know, and the AI would not be trusted. Therefore, a main goal behind XAI is to develop AI systems that people trust. Interestingly, the forms of explanation that fostered the most human trust did not correspond to the learning algorithms that produced the best task performance. Hence, this implies how important it is for robot design to consider both good task performance and trustworthy explanations as an important step toward enabling humanbeings and robots to work together. (Edmonds and Zhu, 2020)

As briefly mentioned one use case for XAI is the docking of a marine vessel. There are several reasons for using AI for controlling vessels. With about 90% of the world's trade carried by sea, the vessels are among the largest contributors to carbon dioxide emissions (Salyer, 2022), and this is an application where AI posibly could help to reduce the emissions from the vessels by having the AI optimize the speed for fuel saving. Another reason for applying AI to vessels is the safety aspect. An example is the AI controlled ABB Ability Marine Braking Assistance used to minimize the stop time and distance, as well as to ensure a safe controlled stopping process (ABB, 2022). In the recent years Marine Autonomous Surface Ships have emerged as a new application of vehicle automation as a result of a rapid technology development, risk and safety science. Despite of a the high focus on safety shipping industry still has safety and material damage consequently a high rate of fatal injuries and severe incidents. More interestingly a study, by Hansen and Frydenberg in 2002, found that most accidents happened performing daily routine duties, and not during occasions with bad weather or other more extreme external factors. (Hansen et al., 2002) When investigating in the underlying causes for marine accidents, human errors where found to be the single greatest contributor and involved in 76%-95% of all accidents. (Rothblum, 2000) However, despite the frequency of human errors in the marine industry, there are reasons why human should be involved in the loop, and hence, the aim is to develop human-AI interaction in autonomous ships. For instance, in a situation of "tail risk"; namely, the risk arising when specific tasks like navigation is performed in an unfamiliar environment or under new conditions, human interaction would be beneficial (Veitch and Andreas Alsos, 2022). However, a human-AI interaction would require the AI to explain the reason behind it's decisions. Hence, motivating the use and research of XAI in marine automation processes.

1.2 Overview of past work

The idea of autonomous docking of a marine vessel is not new and has been studied with several methods. Some of the proposed techniques are optimal control theory, artificial neural networks and expert systems (Martinsen et al., 2019).

However, the autonomous docking of a marine vessel has not only been explored in theory, but also in practice. In 2018 the technology group Wärtsilä actually carried out the world´s first successful testing of autodocking on a ferry. The test started on *"Folgfonn"*, a Norwegian 83-meter long ferry, owned by Norled, and 2000 meters from the berth and included the gradual decrease in speed, the line-up and a safe voyage to the berth until it was fully secured (Sorfonn and Holmlund-Sund, 2018). The control system used was based on Wärtsiläs's existing dynamic positioning system and used GNSS as a primary sensor and Wärtsilä Guidance Marine CyScan AS as a secondary position sensor when approaching the berth (Wärtsilä, 2018). Later, in 2020, Kongsberg succesfully testet out the world's first fully automatic crossing with passengers onboard from dock to dock using an anti-collision system, radar and electro optical sensors (Midtbø, 2020). However, a fully autonomous ship without any passengers was first carried out in 2022 by The Nippon Project, in 2022 (lines, 2022). This ship used an AI-based collision avoidance with Q-learning and Neural Networks implemented as in figure 1.1 (Hashimoto et al., 2021). The Q-learning algorithm was used for risk assessment and resulted in a table shown in figure 1.2.

One important conclusion from this work was that in order to use this for full-scale practical application, it would be desirable to strengthen the visualization of AI's manoeuvring instructions. This would help crew members to understand the intentions of the AI and to develop a man-machine interface for approving those intentions, which highlights the importance of XAI.

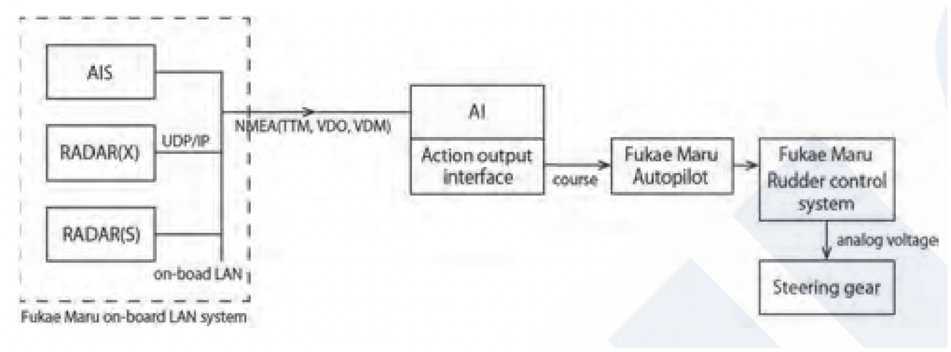
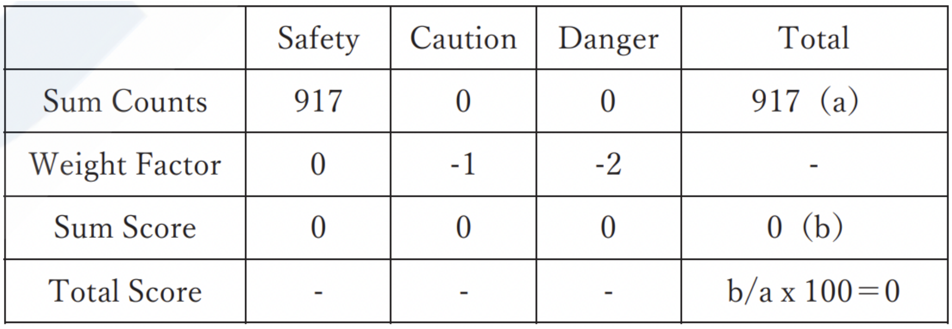


Figure 1.1: Block diagram of AI-based automatic collision avoidance system



	Safety	Caution	Danger	Total
Sum Counts	917	0	0	917 (a)
Weight Factor	0	-1	-2	-
Sum Score	0	0	0	0 (b)
Total Score	-	-	-	b/a x 100=0

Figure 1.2: Example of collision avoidance manoeuvring evaluation results.

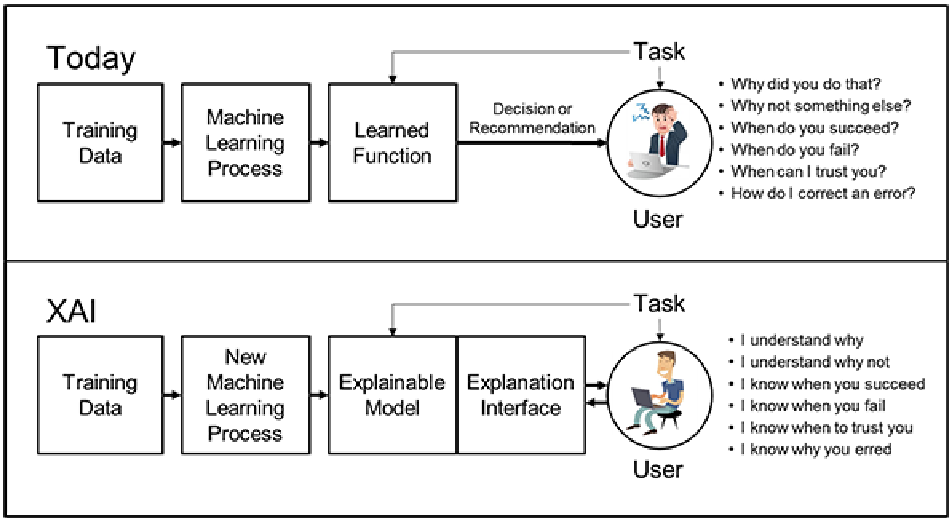


Figure 1.3: Example of collision avoidance manoeuvring evaluation results (Darpa, 2016).

Figure 1.3 visually explains the concept of XAI and what the objectives of XAI is and raises three main challenges:

1. How to produce more explainable models?
2. How to design the explanation interface?
3. How to understand the psychological requirements for effective explanations?

Hence, the XAI should aim to develop general techniques and methods that can be applied in general AI, and to create both visualization, communication in different languages and strategies to create effective and interpretive explanations, in addition to apply psychological theories to assist the developers and users of XAI (Darpa, 2016). In XAI there are five main local model-agnostic methods; Individual conditional expectation curves, Local surrogate models (LIME), Scoped rules (anchors), Counterfactuals, Shapley values and SHAP. Local agnostic models explain individual predictions, in contrast to global models, which usually describes the average behaviour, and would not be beneficial for a captain when trying to figure out why the AI is acting the way it is (Molnar, 2020). Explainable AI is quite new and, therefore, not applied to many areas. In addition, the use of XAI in real time for the real world would require a fast algorithm where both SHAP and Local interpreatable modelagnostic explainer (LIME) are too slow (Løver, 2021). In 2022, Karulus and Lindner applied upward Counterfactuals on Human-in-the-loop Reinforcement learning to see if the method could help to improve learning to increase the training rate, especially during the early stages (Karalus and Lindner, 2022). There has also been tests with model tree based methods for explaining deep reinforcement learning agents actions, which were applied to the docking of marine vessels (Gjærum et al., 2023). However, this requires a tree based model and, hence, due to the complexity of explanations, a limited size of inputs and outputs.

1.3 Scope and objectives

The goal of this thesis is to apply reinforcement learning to the docking of autonomous marine vessel and, hence, evaluate using SHAP and LIME values. Manouvering a ship is a complex task and this thesis will focus on the docking of the vessel. The assumption for this thesis is a convex environment, without any obstacles and a constant current.

The main contribution of this thesis is a methodology to solve the docking of a marine vessel using proximal policy optimization (PPO), a deep reinforcement learning method, using explainable artificial intelligence (XAI). The thesis is partly a continuation of the project thesis, written spring 2022, where PPO was applied to a path following problem and SHAP values where used.

A list of contributions:

1. A ship environment with spatial constraints
2. An adjusted PPO model from the project thesis
3. An adjusted implementation of the SHAP values from the project thesis

It is noteworthy to acknowledge that the scope of this thesis bears a resemblance to Jakob Lover's master thesis Løver (2021). Nonetheless, this thesis strives to delve more comprehensively into the explainable AI methods and employs distinct variants of the PPO model as this has been the most promising method in previous studies Rorvik (2020). This differs from Lover's work, as his study tested several models.

1.4 Overview over thesis

The thesis contains of six chapters:

- Chapter 1 introduces the thesis including relevant background and problem description.
- Chapter 2 presents the relevant theory for this problem. This includes both vessel dynamics, reinforcement learning and XAI.
- Chapter 3 explains the method in detail and how the problem is to be solved.
- Chapter 4 presents the simulation results and discuss them.
- Chapter 5 gives a brief conclusion of this work.
- Chapter 6 discusses the limitations and assumptions for this work and discusses possible future work.

2 Theory

This chapter will present relevant theory for this thesis. This following sections will be presented:

- 2.1 The docking problem
- 2.2 Kinetics and kinematics of a marine vessel
- 2.3 Reinforcement learning
- 2.4 Explainable AI

2.1 The docking problem

The docking is the final and most complex part of a vessels voyage. The operation is also referred to as berthing where the ship should be controlled safely and precisely to the stationary berth. This requires gently and precise control by the captain. Most of the time this is a successful operation. However, the consequences of an unsuccessful operation can be hazardous, giving rise to loss of life, environmental pollution and property damage (Murdoch et al., 2014). The docking process consist of three main phases: *The approach face*, where the vessel travels from open sea to more confined water, *the berthing phase*, where the vessel is parked, and *the mooring phase*, where the vessel is fastened to the berth.

2.1.1 Ship Dynamics

Understanding the dynamics of the ship is important for the manoeuvrability of the ship. This includes the inertia, the wind current, the sea currents, propulsion and hydrodynamic effects. In order to control the ships heading and speed actuators thrusters and control systems are utilized. When the ships decrease speed the forces acting on the ship, such as wind and current forces, have a more significant effect on the manoeuvrability. In addition, the characteristics of thrusters change. The captain need to have a good knowledge of the ships dynamics to successfully berth the vessel.

2.1.2 Thrusters

Thrusters are used in order to give ships more maneouvrability, especially at low speeds, and provide a level of redundancy. They can be controlled by the use of propellers. There are two different types of propellers that can be used. The fixed-pitch (FP) propellers which are the easiest to control and the least expensive, but have limitations in speed and low-end thrust. The controllable-pitch (CP) propellers have, on the other hand, blades that can rotate around their axis, are more expensive and works better at varius speed.
There are three main types of thrusters:

- Azimuth thruster - Which can be rotated 360 degrees.
- Lateral thruster or tunnel thruster - A propellor installed in a athwartship tunnel.
- Jet thruster - A pump taking suction from the keel and discharge to either side.

The azimuth thrusters, Figure 2.1 produce two force components in the horizontal plane and can produce forces in different directions and are therefore frequently used in DP systems. The thruster makes it easier to manoeuvre the ship in the right angle towards the quay and can work in high speeds as well. The thrusters generates a generalized force, τ, in surge, sway and yaw direction, that yield:

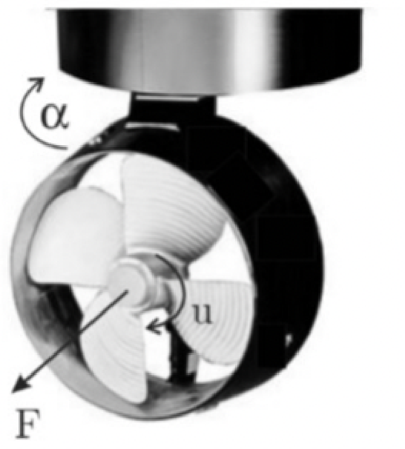
$$\tau = [Fcos(\alpha), Fsin(\alpha), 0, 0, 0, l_x Fsin(\alpha) - l_y cos(\alpha)]^T \tag{2.1}$$

where the force F is (Fossen, 2021):

$$\begin{aligned} FPpropeller &: F = \rho D^4 K_T(0)|n|n \approx T_{|n|n}|n|n \\ DPpropeller &: F \approx T_{\theta|n|n}\theta|n|n \end{aligned} \tag{2.2}$$

where D is the propeller diameter, n the propeller revolution, T the thrust, and K_T is given:

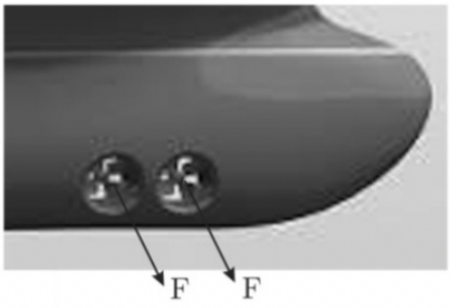
$$K_T = \frac{T}{\rho D^4 |n|n} \tag{2.3}$$



(Fossen, 2021)

Figure 2.1: Azimuth thruster

The tunnel thruster is usually placed in the bow of a ship, such as in Figure 2.2, and is highly effective at low speeds, but not in transit. Hence, it will mostly be used when the ship is at very low speed, below 2km/h, to manoeuvre the ship until it has stopped.



(Fossen, 2021)

Figure 2.2: Tunnel thruster

The propeller produces a transverse force and yield the generalized force:

$$\tau = [0, F_y, 0, 0, 0, l_x F_y]^T \tag{2.4}$$

where,

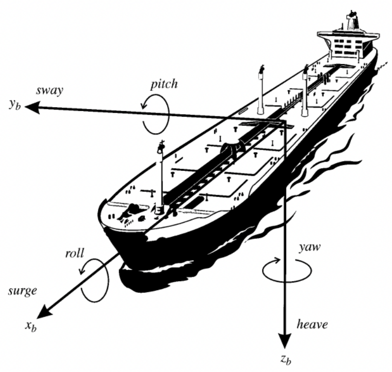
$$\begin{aligned} FPpropeller &: F_y = \rho D^4 K_T(0)|n|n \approx T_{|n|n}|n|n \\ DPpropeller &: F_y \approx T_{\theta|n|n}\theta|n|n \end{aligned} \tag{2.5}$$

where D is the propeller diameter, n the propeller revolution, T the thrust, and K_T is given:

$$K_T = \frac{T}{\rho D^4 |n| n} \tag{2.6}$$

2.2 Kinetics and kinematics of a marine vessel

The marine vessel experience a 6 degrees of freedom (DOF) model when maneuvering as in Figure 2.3.



(Fossen, 2011)

Figure 2.3: Motion in 6 degrees of freedom (DOF)

The 6 motions are listed in the table below 2.1.

Plane	Motion	Description
Horizontal plane	Surge Sway	Longitudinal motion (motion in x direction) Sideways motion (motion in y direction)
Vertical plane	Heave	Motion in vertical direction (motion in z direction)
Rotational	Roll Pitch Yaw	Rotation about the longitudinal axis (x) Rotation about the transverse axis (y) Rotation about the vertical axis (z)

Table 2.1: The 6 degrees of freedom

Each of the motions are described by forces, moments, linear and angular velocities and positions and euler angles where the notation is presented below in table 2.2 by SNAME (1950):

The motions can be given in several reference frames where NED is the north-east-down coordinate system n = x_n, y_n, z_n which is relative to earths centre and the body-fixed reference frame b = x_b, y_b, z_b which is the reference frame with origin that is moving with the craft. The rotation between the two frames, NED and Body, can be denoted R_{Body}^{NED}

DOF	Forces and Moments	Linear and Angular velocities	Positions and Euler angles
Surge	X	u	x
Sway	Y	v	y
Heave	Z	w	z
Roll	K	p	ϕ
Pitch	M	q	θ
Yaw	N	r	ψ

Table 2.2: The notation of SNAME (1950) for marine vessels

where NED is the reference frame which the result will be in, and Body is the reference frame that the vector is converting from.

The dynamics of the shift can be described by kinematics and kinetics where the kinematics describes the motion and kinetics describes how the craft acts when forces is applied to it. The rigid-body dynamics can be expressed in vectorial form according to Fossen (1994) as :

$$\eta = J_\Theta(\eta)v \tag{2.7}$$

$$M_{RB}\dot{v} + C_{RB}(v)v = \tau_{RB} \tag{2.8}$$

$$\tau_{RB} = \tau_{hyd} + \tau_{hs} + \tau_{wind} + \tau_{wave} + \tau_{control} \tag{2.9}$$

where 2.7 describes the kinematics of the system and 2.8 and 2.9 describes the kinetics. The matrices are:

- $\dot{\eta}$: The position vector
- J_Θ: The transformation matrix
- v: The velocity vector
- M_{RB}: The rigid-body inertia matrix
- C_{RB}: The rigid-body Coriolis and centripetal forces
- τ_{RB}: A vector of generalized forces

2.2.1 3 Degree Of Freedom (DOF) model dynamics

For most conventional ships one can assume the angle for roll, θ, and the angle for pitch, ϕ, to be sufficiently small and neglecting the heave and, hence, simplify the ship dynamics to a 3 DOF model, which yields (Fossen, 2011):

$$\dot{\eta} = R(\psi)v \tag{2.10}$$

$$M\ddot{v} + D(v)v = \tau \tag{2.11}$$

where **R**(ψ):=**R**$_{z,\psi}$, **v** $= [u, v, r]^T$ and $/eta = [N, E, \psi]^T$ and are the rotation matrix, the velocity and the position vector, including surge, sway and yaw, respectively. The rotational matrix **J**($\psi \in$ SO(3) is given by:

$$J(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

and describes the rotation from the body reference frame to NED reference frame. The second equation 2.11 contains the inertia matrix, **M**, the dampening matrix, **D**, and the control input vector τ.

2.2.2 Thrust configurations

The control input vector is decided by the thrust configurations where:

$$\tau = T(\alpha)f \tag{2.13}$$

where **T** is the configuration matrix that maps the thrust, f, from each thruster into the surge, sway and yaw forces for the thruster angle α. With a number i of thruster, the right side of the equation can be written out:

$$T(\alpha)f = \begin{bmatrix} F_x \\ F_y \\ F_y l_x - F_x l_y \end{bmatrix} = \begin{bmatrix} f_i cos(\alpha_i) \\ f_i sin(\alpha_i) \\ f_i(l_x sin(\alpha_i) - l_y cos(\alpha_i)) \end{bmatrix} \tag{2.14}$$

where α_i is the orientation of the thruster in the body fixed reference frame and f_i is the force that thruster i produces. Hence, in order to produce the desired control input vector, τ, the thruster configuration can be controlled. In order to optimize this performance some physical constraints can be added to the thrusters roce saturation and azimuth sectors to solve the allocation problem from Johansen and Fossen 2013.

$$\alpha_{i,min} \leq \alpha_i \leq \alpha_{i,max} \tag{2.15}$$

$$f_{i,min} \leq f_i \leq f_{i,max} \tag{2.16}$$

2.3 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning that involves an agent interacting with an environment in order to learn a policy for taking actions that maximize some reward. Reinforcement learning is a relatively new era inside machine learning, and it tries to learn an agent to make that action that maximize reward. Common machine learning is usually supervised, meaning that the agent will receive feedback whether the action made was correct or wrong. However, for reinforcement leaning it is no right or wrong, and in the absence of a training set the agent has to learn from its experience. There are a few elements that generally define a RL task space and these are:

- An agent
- A policy function
- A set of actions
- A set of states
- A reward function

In the task the agent tries to gain maximum reward from the reward function by choosing the action, using the developed value function, that makes the agent get into the ”best” state. In this section aspects of Reinforcement learning such as Markov decision processes (MDP), reward functions and proximal policy optimization (PPO) will be described in more detail.

2.3.1 Markov decision process

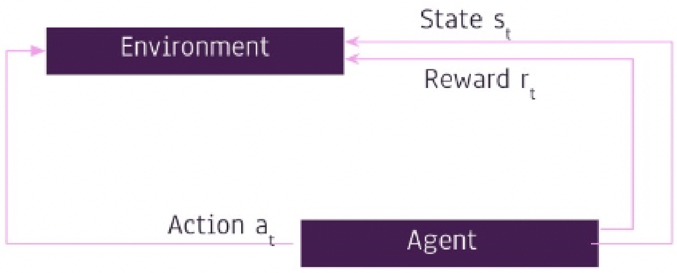


Figure 2.4: Markov Decision Process

The Markov decision process (MDP), Figure 2.4, is the process of the agent trying to decide the best action based on the current state repetitively and contains:

- A set of possible states, S.
- A set of models which gives the effect of an action in a given state, also called the transition function, P(s´- s,a).
- A set of possible actions, A.
- A reward function R(s,a), which is the expected reward gives state, s, and action, a.
- A policy, which is the solution of a MDP.

The Markov decision process can be either deterministic, meaning that the actions chosen will be performed, or stochastic, meaning that there might only be a 80% chance that it is the action chosen that actually will be performed. The MDP assumes that the agent can observe the state and, hence, at all times know its current position, and that this state only will depend on a fixed number of earlier states. In reinforcement learning the goal is to find the optimal policy, given a MDP. This optimal MDP is the optimal value function and is described as:

$$V^*(s) = \max_{\pi}[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1})|\pi, s_0 = s] \tag{2.17}$$

and is the sum of discounted rewards starting from state, s, and choosing the optimal action in every state. H is the horizon of the problem and can be either finite, which it would be in an environment with a finite number of states, or infinite, which it would be in an environment with a continuous state space.

2.3.2 Model-based vs Model-free

Reinforcement learning algorithms can be divided into two broad categories: model-based and model-free.

Model-based RL algorithms involve learning a model of the environment, which allows the agent to make predictions about the consequences of its actions. The agent can then use this model to plan its actions by searching through the space of possible sequences of actions in order to find the one that is most likely to maximize the reward. One advantage of model-based RL is that it can be more sample efficient than model-free RL, as the agent can use its model to simulate the consequences of different actions without actually having to take those actions in the real environment (Sutton and Barto, 2018).

Model-free RL algorithms, on the other hand, do not involve learning a model of the environment. Instead, they learn a policy directly from experience by using trial and error to update the values of actions based on the rewards they receive. Model-free RL algorithms are typically simpler to implement and can be more flexible than model-based algorithms, as they do not require the agent to learn a model of the environment. However, they may require more samples to converge to a good policy compared to model-based algorithms (Sutton and Barto, 2018).

2.3.3 Reward function

In reinforcement learning, the reward function defines the goal of the learning process. It specifies the reward or punishment that the agent receives for taking certain actions in the environment. The agent's objective is to learn a policy for taking actions that maximize the expected reward over time.
The general form of the reward function in reinforcement learning can be represented as follows:

$$R(s, a) = r + \gamma * R(s', a') \tag{2.18}$$

where R(s, a) is the reward received by the agent for taking action, a, in state, s, r is the immediate reward received by the agent for taking the action, s', is the next state that the agent transitions to after taking the action a', is the next action taken by the agent in state s', and γ is the discount factor. The discount factor determines the importance of future rewards compared to immediate rewards, with a value of 0 indicating that only immediate rewards are considered and a value of 1 indicating that all future rewards are considered equally.

The reward function can be defined in various ways, depending on the specific problem being solved. It can be a simple scalar value, or it can be a vector of multiple values. It can also be a function of the current state of the environment and the actions taken by the agent, or it can depend on the sequence of states and actions leading up to the current one.

The choice of the reward function is crucial for the learning process, as it determines the optimization problem that the agent is trying to solve. It is important to carefully design the reward function to reflect the desired behavior of the agent and to avoid providing rewards for undesirable actions (Sutton and Barto, 2018).

2.3.4 The Exploration vs Exploitation dilemma

In RL, exploration refers to the process of taking actions that allow the agent to learn more about the environment and the consequences of its actions. This can involve trying out different actions in order to discover which ones lead to higher rewards, or visiting states that have not been visited before in order to gather more information about the environment.

Exploitation, on the other hand, refers to the process of taking the actions that are known to lead to the highest rewards based on the current knowledge of the agent. For example, if the agent has learned that taking a certain action in a certain state leads to a high reward, it may choose to exploit this knowledge by taking that action repeatedly in order to maximize the reward.

The trade-off between exploration and exploitation is a central challenge in RL. On the one hand, it is important for the agent to explore the environment in order to learn as much as possible about it. On the other hand, the agent needs to exploit its current knowledge in order to maximize the reward. Balancing these two objectives is known as the exploration-exploitation dilemma.

One approach to addressing the exploration-exploitation dilemma is to use an exploration policy that gradually decreases the amount of exploration over time as the agent becomes more confident in its current policy. This allows the agent to initially explore more of the environment in order to gather more information, while eventually focusing more on exploitation as it becomes more confident in its policy (Sutton and Barto, 2018).

2.3.5 Artificial Neural Networks

Artificial neural networks (ANN´s) are a type of machine learning model that are inspired by the structure and function of the brain. They consist of multiple interconnected units called ”neurons,” which are inspired by the neurons in the brain. ANN´s are commonly used in RL to approximate the value function or the policy of the agent.

In RL, the value function represents the expected return (i.e. the sum of the discounted rewards) that the agent can expect to receive by following a particular policy. The agent is an entity that interacts with an environment to learn how to perform a task. The value function can be used to evaluate the quality of different actions and to choose the action that is most likely leading to the highest return as the agent´s task is to learn an optimal policy that maximizes its expected cumulative reward. ANN´s can be used to approximate the value function by training an algorithm that maps states or state-action pairs to their corresponding values.

The policy of an RL agent defines the actions that the agent should take in each state. ANN´s can be used to approximate the policy by learning a function that maps states to the probabilities of taking each action. This can be useful when the space of possible actions is large or continuous, as it allows the agent to choose actions that are more likely to lead to high rewards.

ANN´s can be trained using a variety of algorithms, such as Q-learning and policy gradient methods. These algorithms typically involve adjusting the weights and biases of the ANN based on the rewards and transitions experienced by the agent in the environment(Sutton and Barto, 2018).

2.3.6 Proximal policy optimization (PPO)

Proximal policy optimization (PPO) is a RL algorithm that aims to improve the stability and efficiency of policy gradient methods (Schulman et al., 2017). It is based on the idea of "trust region" optimization, which involves constraining the change in the policy to be within a certain range in order to prevent it from making too large of a update (Schulman, 2020).

In PPO, the policy is updated by maximizing a "surrogate objective" function that is closely related to the expected reward of the policy (Schulman et al., 2017). The "surrogate objective" function is defined as the ratio of the new policy to the old policy, multiplied by the advantage function, the difference between the expected reward of the new policy and the expected reward of the old policy (Schulman, 2020). By maximizing this objective function, the PPO algorithm can update the policy in a way that increases the expected reward while still being close to the old policy.

The general form of the PPO optimization objective can be represented as follows (Schulman et al., 2017):

$$L = E[min(r(\theta) * A, clip(r(\theta), 1 - \epsilon, 1 + \epsilon) * A)] \tag{2.19}$$

where L is the optimization objective, r(θ) is the ratio of the current policy function to the previous policy function, A is the advantage function, ϵ is the clipping threshold, and θ are the parameters of the policy function.

The advantage function, A, is a measure of how much better or worse a particular action is compared to the average action taken by the policy. The advantage function is typically calculated using the difference between the expected return for a particular action and the expected return for the average action. The ratio of the current policy function to the previous policy function, r(θ), is used to measure the change in the policy function from one iteration to the next. The clipping term, clip(r(θ), 1 - ϵ, 1 + ϵ), limits the change in the policy function by clipping the value of r(θ) to the range [1 - ϵ, 1 + ϵ]. This helps to stabilize the learning process and prevent the policy function from changing too rapidly. The optimization objective, L, is then defined as the expected value of the minimum of the unclipped and clipped versions of the policy function ratio multiplied by the advantage function. This objective encourages the policy function to make large updates when the advantage of taking a particular action is high, but limits the size of the update when the advantage is low or the change in the policy function is large.

PPO algorithms typically use a neural network to represent the policy, and they use a combination of gradient ascent and gradient descent to update the weights of the network

(Schulman et al., 2017). They also use a number of techniques to stabilize the learning process, such as using generalized advantage estimation to estimate the advantage function and using mini-batches of data to update the policy (Schulman, 2020).

PPO is often used in autonomous systems, including autonomous ships. It is a type of on-policy algorithm, which means that it uses data generated from the current policy to update the policy. One of the main advantages of PPO is that it is relatively simple to implement and can achieve good performance with relatively few hyperparameters.

Additionally, PPO is known for being stable and robust, which is important in the context of autonomous ships where safety is a critical concern (Cooper, 2019). It is also relatively sample efficient, meaning that it can learn effectively using relatively small amounts of data, which is important in the context of autonomous ships where data collection may be difficult or expensive. Finally, PPO is able to handle continuous action spaces and can learn policies that are able to take advantage of complex, nuanced environments, which is important in the context of autonomous ships that operate in complex and dynamic environments.

2.4 Explainable AI

Explainable artificial intelligence (XAI) refers to the development of artificial intelligence systems that are able to provide explanations for their actions or decisions. This is an important area of research, as many AI systems are designed to operate autonomously and may make decisions that have significant consequences. In order to build trust and accountability in these systems, it is important to be able to understand how they arrived at their decisions.

2.4.1 Shapley values

Shapley values are a mathematical tool that can be used to explain the contribution of individual features or variables to the prediction made by a machine learning model. They were developed by Lloyd Shapley in the 1950s, and used in the field of cooperative game theory.

Shapley values can be used to provide an ”attribution” for each feature, which is a measure of the contribution that the feature made to the overall prediction made by the model. The Shapley value for a particular feature is calculated by considering all possible combinations of features and their contributions to the prediction, and then averaging these contributions across all possible combinations. The result is a measure of the marginal contribution of the feature to the overall prediction, taking into account the interactions between features.

The general form of the Shapley value for a state or action, s, in a Markov Decision Process (MDP) can be represented as follows (Shapley, 1997):

$$\phi(s) = \Sigma[(|S| - 1)!/(|S - s|)!(|s|)!] * V(S) \tag{2.20}$$

where S is the set of states in the MDP, —S— is the size of the set S, —S-s— is the size of the set S without the state s, —s— is the size of the set containing only the state s, and V(S) is the value of the MDP with the set of states S. The Shapley value, ϕ, for a state or action is calculated by summing over all possible coalitions of states in the MDP. For each coalition, the Shapley value is calculated as the product of a combinatorial factor and the value of the MDP with the given coalition of states. The combinatorial factor is a function of the size of the coalition and the size of the set containing the state of interest.

The Shapley value can be used to evaluate the importance of a state or action in the learning process by calculating the Shapley value for each state or action and comparing the values. States or actions with high Shapley values are likely to be more important for the agent to visit or take in order to maximize the reward.

For instance, considering a docking scenario where the aim is to predict the time it will take for a vessel to dock at a particular port based on various factors such as wind speed, wave height, and water depth. The dataset consists of 500 docking operations, and an objective is to understand which factors are most important for making accurate predictions. In this example the Shapley value can be used to attribute the prediction of a machine learning model to its input features. For example, the Shapley value can determine the contribution of each factor to the predicted docking time of a vessel. To calculate the Shapley value for a factor, all possible subsets of factors are considered and the difference in the model's prediction with and without the factor is calculated. The Shapley value for a factor is the average of these differences, weighted by the number of subsets that include the factor.

2.4.2 SHAP values

SHAP (SHapley Additive exPlanation) values are a method for interpreting the output of machine learning models by decomposing the model's predictions into the contributions of each feature or variable. SHAP values can be used in reinforcement learning to understand the importance of different states or actions in the learning process, and to identify states or actions that may be important for the agent to visit or take in order to maximize the reward (Lundberg and Lee, 2017).

The general form of the SHAP value for a state or action, s, can be represented as follows:

$$\phi(s) = \Sigma[f(S) - f(S - s)] \tag{2.21}$$

where f is the model's prediction function, S is the set of states in the Markov Decision Process (MDP), and —S-s— is the size of the set S without the state s.

The SHAP value for a state or action is calculated by summing over all possible coalitions of states in the MDP. For each coalition, the SHAP value is calculated as the difference between the model's prediction with the full set of states and the model's prediction with the given coalition of states.

The SHAP value can be used to evaluate the importance of a state or action in the learning process by calculating the SHAP value for each state or action and comparing the values. States or actions with high SHAP values are likely to be more important for the agent to visit or take in order to maximize the reward. SHAP values can be calculated using a variety of methods, including the SHAP Kernel method and the SHAP TreeExplainer method (Ribeiro et al., 2016).

Using the example presented in the Shapley section, the SHAP values also provide a way to explain the output of any machine learning model. SHAP values assign each factor an importance value for a particular prediction, providing a unified framework for interpreting model predictions. For example, the SHAP values can be used to understand the importance of each factor in predicting the docking time of a particular vessel.

2.4.3 LIME

LIME (Local Interpretable Model-agnostic Explanations) is a method for interpreting the output of machine learning models by generating explanations that are locally faithful to the model's predictions. LIME can be used in reinforcement learning to understand the importance of different states or actions in the learning process, and to identify states or actions that may be important for the agent to visit or take in order to maximize the reward (Ribeiro et al., 2016).

The general form of the LIME value, ϕ for a state or action, s, can be represented as follows:

$$\phi(s) = \Sigma[f(S) - f(S - s)] \tag{2.22}$$

where f is the model's prediction function, S is the set of states in the Markov Decision Process (MDP), and —S-s— is the size of the set S without the state s.

The LIME value for a state or action is calculated by summing over all possible coalitions of states in the MDP. For each coalition, the LIME value is calculated as the difference between the model's prediction with the full set of states and the model's prediction with the given coalition of states.

The LIME value can be used to evaluate the importance of a state or action in the learning process by calculating the LIME value for each state or action and comparing the values. States or actions with high LIME values are likely to be more important for the agent to visit or take in order to maximize the reward. LIME values can be calculated using a variety of methods, including sampling and optimization techniques.

For example, for the aim to predict the time it will take for a vessel to dock the LIME values provide local explanations by fitting a simple model to the local neighborhood around a particular prediction and approximating the original model's behavior using this simple model. In this situation the LIME values can be used to explain the predicted docking time of a particular vessel based on its wind speed, wave height, and water depth.

2.4.4 Counterfactuals

Counterfactuals are statements or statements that describe a hypothetical situation or event that is contrary to the facts or circumstances that actually occurred. In reinforcement learning, counterfactuals can be used to understand the impact of different states or actions on the learning process, and to identify states or actions that may be important for the agent to visit or take in order to maximize the reward (Pearl, 2009).

The general form of a counterfactual statement in reinforcement learning can be represented as follows: ”If the agent had taken action a in state s, rather than the actual action a', the reward would have been r rather than r'.” where a is the hypothetical action taken by the agent, s is the state in which the action is taken, a' is the actual action taken by the agent, r is the hypothetical reward received by the agent, and r' is the actual reward received by the agent.

Counterfactuals can be used to evaluate the importance of different states or actions in the learning process by comparing the hypothetical reward received by the agent with the actual reward received. States or actions that lead to a higher hypothetical reward are likely to be more important for the agent to visit or take in order to maximize the reward. Counterfactuals can be calculated using a variety of methods, including counterfactual regret minimization and inverse reinforcement learning.

3 Method

The docking of a ship is considered a complex task, where there are several restrictions that has to be kept simultaneously. The main contribution for this thesis is the implementation of a 3 Degree Of Freedom vessel ship berthing and the use of deep reinforcement learning. After model had been trained XAI was implemented using SHAP and LIME in order to gain insight of the models choices. Counterfactuals was not implemented due to the limitations in size of the state vector and action vector.
In the sections in this chapter the implementation and design of the problem is discussed and includes:

- 3.1 Reference frames and notation

- 3.2 Reinforcement learning applied to docking problem

- 3.3 Proximal policy optimization

- 3.4 Explainable AI

3.1 Reference frames and notation

For the docking of a marine vessel there were two different reference frames used:

- The North-East-Down (NED) frame: The global frame where the spatial constraint, the goal constraint and the vessel constraints was defined.
- The bodyframe: The local frame from the vessel´s point of view where the velocities were adjusted by the thrusters.

The frames are visualized in figure 3.1 where North and East represents the NED frame and u and v are parameters from the bodyframe.

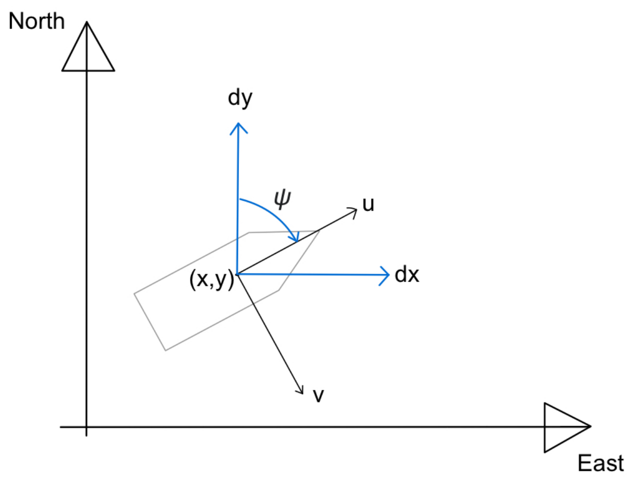


Figure 3.1: The ship with bodyframe in NED frame.

The notation for the problem was the position vector $\eta = [x, y, \psi]^T$ and the velocity vector v = $[u, v, r]$, with the Cartesian coordinates (x,y), the yaw angle, ψ, the linear velocities, u and v, and the yaw rate, r. In order to convert between the two different reference frames a rotation matrix, J, was used. This matrix is given:

$$\mathbf{J}(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

3.1.1 The vessel

For this thesis the vessel model is based on the SV Northern Clipper ship from Fossen. The model is a 3 degree of freedom model with the dynamic equations:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{J}_{\psi}\boldsymbol{v} \\ \boldsymbol{M}\dot{\boldsymbol{v}} + \boldsymbol{D}(\boldsymbol{v})\boldsymbol{v} &= \boldsymbol{\tau} \end{aligned} \tag{3.2}$$

where J, M, D and τ are the rotational matrix, the inertia matrix, the dampening matrix and the control input vector, respectively.
..

The thruster configurations were specified by the thrust configuration matrix $\mathbf{T}(\alpha)$ and maps each thrust **f** into surge, sway and yaw. For this vessel to azimuth thruster were used and one tunnel thruster. These were placed as seen in figure 3.2 where 1 and 2 are the azimuth thrusters and 3 is the tunnel thruster in the front:

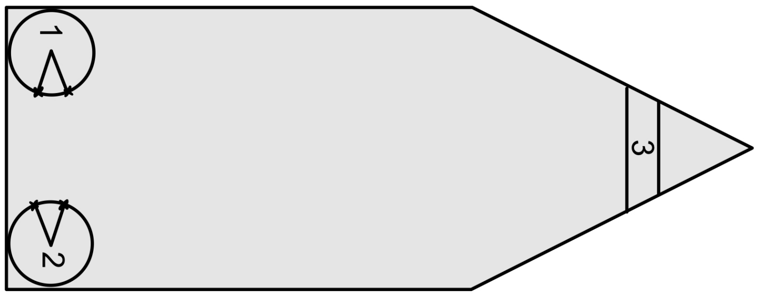


Figure 3.2: The thruster placement where 1 and 2 are the azimuth thrusters and 3 is the tunnel thruster.

The thrusters positions and angle were also set to:

Thruster	x-position	y-position	Angle
Azimuth thruster 1	l_{x_1} = -35m	l_{y_1} = 7m	α_1
Azimuth thruster 2	l_{x_2} = -35m	l_{y_2} = -7m	α_2
Tunnel thruster 3	l_{x_3} = 35m	l_{y_2} = 0m	$\alpha_3 = \frac{\pi}{2}$

Table 3.1: Thruster position and angle

A constraint of 1/30 of the vessels mass was added for the maximum thrust for the azimuth thrusters, and 1/60 of the vessels mass for the tunnel thruster. The azimuth thruster also had a limitations with a 20 degree forbidden zone, shown in figure 3.2 in order to avoid that the thrusters just worked opposite of each other. In addition, their maximum turnaround time was set to 30 seconds per revolution and a maximum angle of 170 degrees.

In order to control the vessels movement the force from each thruster was mapped to surge, sway and yaw by the configuration matrix $\mathbf{T}(\alpha)$ given the thruster angles, α_i. Then this was added to the dynamics 3.2 as:

$$\boldsymbol{\tau} = \mathbf{T}(\alpha_i)\boldsymbol{f} \tag{3.3}$$

where each column in $\mathbf{T}(\alpha)$ maps to each thruster i and each row maps to the force in surge, sway and yaw direction. With the parameters from 3.1 the control configuration matrix becomes:

$$\begin{aligned}\mathbf{T}_i(\alpha)f_i &= \begin{bmatrix} \boldsymbol{F_x} \\ \boldsymbol{F_y} \\ \boldsymbol{F_y l_x} - \boldsymbol{F_x l_y} \end{bmatrix} = \begin{bmatrix} f_i cos(\alpha_i) \\ f_i sin(\alpha_i) \\ f_i(lxsin(\alpha_i - l_y cos(\alpha_i)) \end{bmatrix} \\ &= \begin{bmatrix} cos(\alpha_1) & cos(\alpha_2) & 0 \\ sin(\alpha_1) & cos(\alpha_2) & 0 \\ lx_1 sin(\alpha_1 - l_{y_1} cos(\alpha_1) & lx_2 sin(\alpha_2 - l_{y_2} cos(\alpha_2) & l_{x_3} \end{bmatrix}\end{aligned} \tag{3.4}$$

where α and **f** corresponds to the orientation and force of each thruster respectively.

3.1.2 The docking environment

Docking is a complex problem and can be simulated using several different methods, such as path-following problem method, collision avoidance or other methods. For this thesis the docking is simulated using spatial constraints. Firstly, the spatial constraint was created as a convex pentagon with the coordinates [(260,585), (650,800), (1000,540), (800,40), (400,40)], as seen in figure 3.3 marked in blue, with no obstacles in order to simulate the docking where there are land around. Secondly, the goal constraints, marked as green points and lines in figure 3.3, were implemented as a pentagon close to one of the spatial lines in order for the ship to simulate a berth close to the land. The coordinates for the goal was set to [(317,499), (290,535), (287,489), (310, 402), (344,412)] which was a larger area than the ship itself in order to make the task easier to solve for the agent. The initial position area where the center was initialized using a random generator, where x could be between 650 and 750 and y could be between 100 and 200, are marked on figure 3.3 in red.

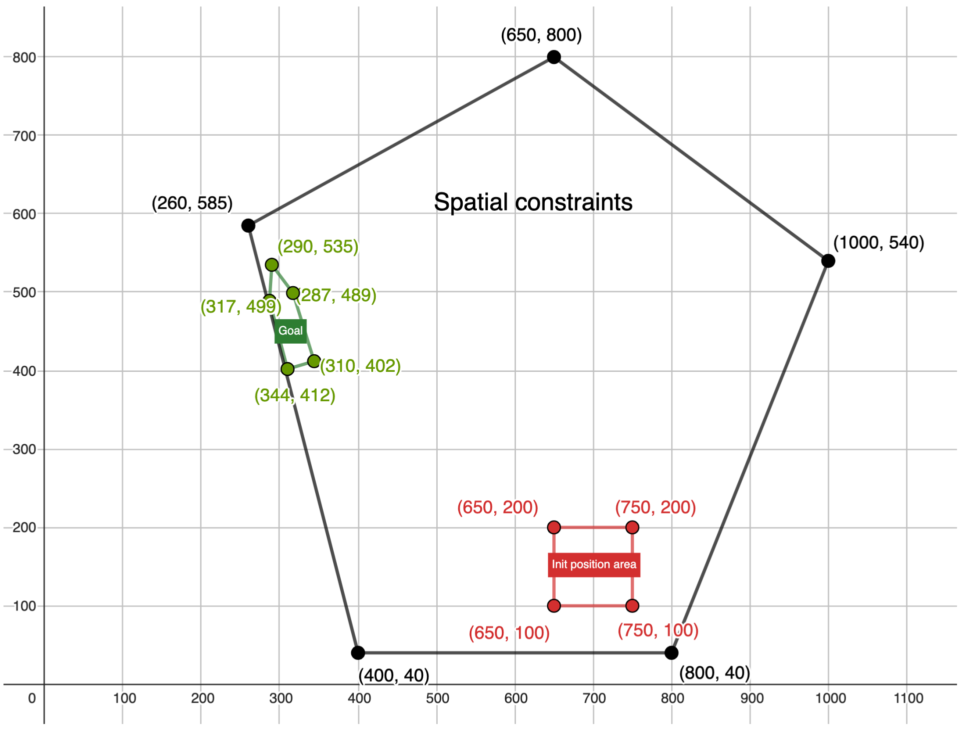


Figure 3.3: The spatial constraints, the goal and the initial area for the environment.

In addition, the speed in surge, forward, direction were initialized to 2.5 and the velocity in sway and the yaw rate were set to 0. An overview of the initialization parameters are given in table. 3.2.

Parameter	Value range	Units
Ship centre x value	(650,750)	meters
Ship centre y value	(100,200)	meters
Yaw angle ψ	(-10, 10)	degrees
Surge velocity u	2.5	meters per second
Sway velocity v	0	meters per second
Yaw rate r	0	degrees per second

Table 3.2: Initialization parameters for the docking environment

3.1.3 Simulation of vessel movement

For the simulation of the vessel the environment was ran with a maximal timesteps per episode of 1200. The environment´s task is to calculate the rewards and states based on the actions given by the PPO and the previous states. The following subsection will go into detail about how this was implemented.

Dynamics of the vessel

The dynamics of the vessel is the dynamics of 3 degree of freedom marine vessel and can be presented as:

$$\boldsymbol{\eta} = \boldsymbol{R}(\psi)\boldsymbol{v} \tag{3.5}$$

$$\boldsymbol{M}\dot{\boldsymbol{v}} + \boldsymbol{D}(\boldsymbol{v})\boldsymbol{v} = \boldsymbol{\tau} \tag{3.6}$$

where η is the position vector, R the rotation matrix, ψ the angle of the vessel, **v** the velocity vector, **M** the mass matrix, **D** the dampening matrix and τ is the control input matrix.

For this problem the PPO initializes the environment, receives the action vector from the neural network and then run the step function in the environment. The action vector is scaled in the step function, which is further explained in section 3.3.1, with 1/30*the vessels mass for the forces by the azimuth thruster, and 1/60*the vessels mass for the force by the tunnel thruster. The angles where scaled between -30 to 30 degrees. The update function of the environment is based on eulers formula and the update rule is implemented as follows: where timespan = 10 and the timestep is 0,1 second which results in 100

Algorithm 1 Update position η and velocity v

```
$dt \leftarrow 0,1$ $t$ in range $\frac{timespan}{dt}$
$J \leftarrow$ rotation matrix
$\tau \leftarrow$ control input matrix
$\eta \leftarrow \eta + J \times v$
$\dot{v} \leftarrow solve(M \times \dot{v} = \tau - D \times v)$
$v \leftarrow \dot{v} \times dt + v$
```

iterations as a tradeoff between accuracy, stability and efficiency. After the position and the velocity are updated, the observation parameters are calculated. For this problem the following observations were selected:

- $\tilde{x}$: The distance between the goal center and the vessel´s center in x direction.
- $\tilde{y}$: The distance between the goal center and the vessel´s center in y direction.
- $\tilde{\psi}$: The distance between the goal angle and the vessel´s angle.
- u: The velocity of the ship in u direction (forward in bodyframe coordinates).
- v: The velocity of the ship in v direction (sideways in bodyframe coordinates).
- l: A binary variable describing whether the vessel is in contact with the land where True implies that is in contact with the spatial constraints and False that it is inside the spatial constraints.
- f_g: Number of corners inside berth area.
- d_{obs}: Distance to closest spatial constraint.

3.2 Reinforcement learning applied to docking problem

The Deep Reinforcement learning algorithm, proximal policy optimization (PPO) was applied in this thesis. This method has had success earlier with robotic systems and have been applied several times to control marine vessels (Rorvik, 2020). PPO can be used on environments with both discrete and continuous action vectors and observation vectors and is a model free and online algorithm. In addition, it is on on-policy which means that it learns based on experience from the current policy. The implementation details will be described in the sections below.

3.3 Proximal policy optimization

The PPO algorithm used in this thesis was based on the project thesis, written spring 2022, and the guide made by Eric Yang Yu (Yu, 2020). The PPO consists of six main files:

- **main.py**: Where the code is executed which reads from the config file and pass it on the PPO.
- **config.py**: The configurations for the PPO, where learning rate, gamma, number of games and number of hidden layers are set.
- **PPO.py**: The PPO model, which initializes the environment and do the learning process.
- **network.py**: A simple network which consists of a Feed-Forward Neural Network.
- **EvaluatePolicy.py**: An independent module which test the trained policy on the environment.

The PPO algorithm is designed to be a simple, efficient and stable algorithm and was implemented using PyTorch and Python to train the agent. The algorithm uses a policy (x) to generate control actions a and a value function V(x) to estimate the cumulative discounted reward of being in state x. The value function is estimated using TD(1)-estimation, and the advantage is estimated using GAE-lambda (Schulman et al., 2017).

The PPO algorithm is based on the original PPO-clip actor-critic implementation of Schulman et al. (Up, 2018). The algorithm uses a clipped surrogate objective to ensure reasonable policy updates, but it is still possible to end up with a new policy that is too far from the previous policy. To prevent this, ”early stopping” was employed by stopping to perform gradient steps if the mean KL-divergence of the new policy compared to the previous policy grew beyond a certain threshold. The threshold used was 0.015, based on the recommendations from Spinning Up (Up, 2018).

The policy and value-functions were approximated using two fully-connected neural networks, with each hidden layer consisting of 64 and 128 hidden units receptively and ReLU (Rectified linear unit) as the activation function. The ReLU function was used in order to overcome the vanishing gradient problem and allows for faster and better learning as it maps negative inputs to zero and positive inputs to the same positive value (Brownlee, 2019). The states were normalized using an upper and lower limit for each value. The policy is stochastic, meaning

$$\pi = \mu(x) + \sigma, \tag{3.7}$$

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 the input state vector, x, and output action vector, a. The standard deviation is calculated during the training process (Schulman et al., 2017).

The policy in equation 3.7 can give actions higher than 1, so it is necessary to saturate this value before being applied to a physical system, as performed by OpenAI in their Baseline implementation of PPO (OpenAI, 2023b). The saturated value is thereafter scaled before being applied to the marine vessel. The value network, the critic, has state vector 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 (Schulman et al., 2017).

During training, the optimizer used is ADAM, as recommended by Schulman (Up, 2018) and it is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments (Kingma and Ba, 2014). The parameters used relevant to the performance of the training are mini-batch size, replay buffer size, discount rate, γ, actor learning rate, critic learning rate, number of epoch updates with mini-batch (K), GAE parameter, λ, and clipping range. These parameters were based on recommendations from Spinning Up and adjusted by means of trial and error. The clip ratio was set to 0,2 to ensure that the policy update was limited to at most 20 percent from the previous policy to improve stability and help prevent the policy from changing too quickly. Choosing a higher value for the clip ratio would have resulted in a larger policy updates, and it could also have resulted in more oscillations, which could lead to instability. On the other hand, a lower clip rate could have resulted in a slower convergence, but a more stable learning. This means that for a this task, which does not have sparse reward, i.e. rewards are given after every move, a higher clip ratio would be preferable in order to allow for a less cautious update of the policy. The PPO algorithm assumes an episodic problem and the maximum length of an episode was set to 1200 seconds in the docking scenario for all the learning scenarios (Schulman et al., 2017).

For the implementation of the PPO it was important to ensure that it was possible to use on different environments for future work. The environment was, therefore, initialized as a call on a separate file. Thereafter the main steps of a PPO algorithm was implemented:

- A batch of transitions was collected, including observation, action, reward, next observation and finished, from the environment.
- The advantage estimates were computed by the value network (critic).
- The Policy network was updated to minimize the PPO loss.
- The value network was updated in order to minimize the mean square (MSE) loss.

The timesteps for each batch was set to 20000, the maximum timesteps per episode to 1200 and the saving frequency to 10 as proposed by Eric Yang Yu in his PPO guide (Yu, 2020) and as Ella Lovise Røvik did in her master thesis (Rorvik, 2020). The MSE (mean squared error) loss function was used to measure the difference between the predicted action and the actual action taken by the agent.

The action vector will be described more in detail in the next section, but in brief, the output layer consisted of five nodes, where each was a continuous number that controlled each of the possible parameters to act on. The relevant parameters used for the testing scenarios are:

- Batch size: 20 000
- Learning rate: 0.01
- Discount rate γ: 0.95
- Number of updates per iteration: 80
- Clip ratio: 0.2

3.3.1 The action vector

The action vector for this problem consisted of five parameters:

- f_1: The force which should be applied to the azimuth thruster 1.
- f_2: The force which should be applied to the azimuth thruster 2.
- f_3: The force which should be applied to the tunnel thruster.
- α_1: The angle that should be applied to the azimuth thruster 1.
- α_2: The angle that should be applied to the azimuth thruster 2.

which controlled the three thrusters on the vessel. As docking of a vessel is a complex task which requires manoeuvrability in backward, forward and sideways directions the action space had to be a dimension of five parameters even though the aim was to keep it as simple as possible. The action was chosen by the five output nodes from the actor network and evaluated by the critic network.

3.3.2 Reward function

The reward function is an important part of a RL problem and gives feedback on the performance of the agent. This feedback is used in order for the agent to learn how to reach its goal and, hence, needs to reflect the objective of the problem. The reward function for the PPO algorithm is designed to be differentiable and, therefore, optimizable, using gradient-based optimization algorithm such as Adam. This differs from other reinforcement learning algorithms where the reward function is typically a hand-designed, scalar function that assigns a reward to each state-action pair (Sutton and Barto, 2018). The reward function has a significant impact on the performance of the agent, hence, for this problem two different reward functions were implemented and tested.

Reward function 1

The first reward was based on trial and error with an aim not to let any of the parts dominate over the other parts in order to secure an action loss for the actor to learn. The function provided aims to guide the marine vessel towards the desired goal, which is to approach the berth. It considers various aspects of the vessel's behavior, including its position, velocity, rotation, proximity to obstacles, and whether it has made contact with the land.

The reward function is written as:

$$reward = \begin{cases} -(x^2 + y^2) - 1.0(u^2 + v^2) - 1.0r + 10f_g - 10 & \text{if } d_{obs} < 10 \\ -(x^2 + y^2) - 1.0(u^2 + v^2) - 1.0r + 10f_g - 1000 & \text{if } l = True \\ -(x^2 + y^2) - 1.0(u^2 + v^2) - 1.0r + 10f_g + 1000 & \text{if } f_g = 5 \text{ and } \tilde{\psi} \leq 15 \\ -(x^2 + y^2) - 1.0(u^2 + v^2) - 1.0r + 10f_g & \text{otherwise} \end{cases} \tag{3.8}$$

where x, y, u, v, r, l, f_g and d_{obs} are the input variables of the function which are explained in the paragraph named Dynamics of the vessel above. The first component of the reward function encourages the vessel to get closer to the goal by penalizing the distance between the vessel's position and the berth. This is achieved by subtracting the squared values of the differences in x and y positions between the vessel and the berth. The second component of the reward function encourages the vessel to reduce its velocity. This is achieved by penalizing the square of the vessel's velocity in the u and v directions. The third component of the reward function encourages the vessel to reduce its rotational velocity by penalizing the rotational velocity in the body frame. The fourth component of the reward function rewards the vessel for being inside the goal area. This is achieved by adding a positive reward that is proportional to the number of corners of the vessel that is inside the goal area. The fifth component of the reward function penalizes or rewards the vessel for rare occasions. If the vessel is closer than 10 meters from obstacles, a fixed value of 10 is subtracted from the reward. However, if l is True, meaning that the marine vessel is in contact with the land, a value of 1000 is subtracted from the reward function. On the other hand, if all corners are inside the goal area and the angle difference is less than 15 degrees the agent is in a "win" state and is rewarded with a fixed value of 1000.

The finished parameter are used to indicate when the vessel has achieved the goal or is out of the observation space. If all four corners are inside and the vessel is also oriented within 15 degrees of the desired orientation, a positive reward is added, and the episode is terminated as the marine vessel achieved its goal. However, if the marine vessel is in contact with the land, l = true, then the finished variable is set to true, indicating that the episode is terminated. The function provides both positive and negative rewards, depending on the vessel's behavior, to guide it towards the desired goal.

Reward function 2

After evaluating the first reward function a slightly more adjusted reward function was proposed. The reward function was given:

$$reward = \begin{cases} -(\tilde{x}^2 + \tilde{y}^2) - 0.5 \times |u| - 1 * \times |v| - 0.8 \times |r| + 10 \times f_g - \tilde{\psi}^2 & \text{if } f_g \geq 1 \\ -(\tilde{x}^2 + \tilde{y}^2) - 0.5 \times |u| - 1 * \times |v| - 0.8 \times |r| + 10 \times f_g - 1000 & \text{if } l = True \\ -(\tilde{x}^2 + \tilde{y}^2) - 0.5 \times |u| - 1 * \times |v| - 0.8 \times |r| + 10 \times f_g + 1000 & \text{if } f_g = 5 \text{ and } \tilde{\psi} < 8 \\ -(\tilde{x}^2 + \tilde{y}^2) - 0.5 \times |u| - 1 * \times |v| - 0.8 \times |r| + 10 \times f_g \text{otherwise} \end{cases} \tag{3.9}$$

where the velocity, u was penalized less in order to motivate the agent to move forward and v, where penalized as in reward function to avoid sideways movement. The yaw rate, r, was also penalized with a lower scalar as an observation from the first reward function was that the vessel tended to travel sideways to get to the goal rather than turning around and driving forwards. The $\tilde{\psi}$ was also adjusted to be below 8 degrees for the episode to terminate to achieve a higher goal. This was set to 15 in the first reward function only to see that the vessel was able to learn to get to the goal area.

3.4 Explainable AI

3.4.1 SHAP - SHapley Additive exPlanations

For this problem SHAP values are implemented in order to see the feature importance as a way to evaluate and understand learned policy. Therefore, the SHAP method is only used in the testing phase of the code. In the implementation of the code an evaluate policy function is responsible for evaluating the policy on an environment and logging various information about the episode. The rollout function simulates each episode and returns relevant information such as episode length and return and the logs of state and action. Then the code employs SHAP for model interpretability. It calculates the importance of each feature for a given prediction using the a SHAP function. The function takes the agent (policy), state logs, feature names, and action logs as inputs. It then creates a deep explainer object using the SHAP library that can compute the SHAP values for the given state logs and the policy. The function also produces a summary plot of the SHAP values, which visualizes the contribution of each feature towards the policy's action prediction.

3.4.2 LIME

LIME is a popular XAI technique that generates explanations for predictions made by a machine learning model by fitting a simple, interpretable model locally around the prediction. The LIME (Local Interpretable Model-Agnostic Explanations) are implemented in a similar way as SHAP where a function in the rollout function calls for the LIME visualization using the machine learning model, state data, feature names and action data. The input data is first preprocessed and reshaped to ensure compatibility with the LIME library. The LIME library is then used to fit a local interpretable model around the prediction made by the input machine learning model. The LIME explanation is then visualized using a bar plot, which shows the feature importance of each input feature in determining the prediction.

4

Results and Discussion

In this chapter the main results of the Proximal policy optimization agent on the docking environment are presented and discussed. The chapter consists of the following sections:

- Section 4.1: Reward function 1. Presents and discusses the results, including the SHAP and LIME values of the reward function 1 on the docking environment.
- Section 4.2: Reward function 2. Presents and discusses the results, including the SHAP and LIME values of the reward function 2 on the docking environment. This also includes three different learning rates.
- Discussion: A brief dicussion is presented at the end to discuss the main results from the different scenarios.

4.1 Reward function 1

The agent performing the docking with reward function 1 had the objective to control the boat to the desired goal area with a somewhat correct angle. The marine vessel is controlled by the agent through the three thruster forces $[f_1,\ f_2,\ f_3]$ and the two angles $[\alpha_1, \alpha_2]$. The reward function, as written in chapter 3, is written as:

$$reward = \begin{cases} -(x^2+y^2) - 1.0(u^2+v^2) - 1.0r + 10f_g - 10 & \text{if } d_{obs} < 10 \\ -(x^2+y^2) - 1.0(u^2+v^2) - 1.0r + 10f_g - 1000 & \text{if } l = True \\ -(x^2+y^2) - 1.0(u^2+v^2) - 1.0r + 10f_g + 1000 & \text{if } f_g = 5 \text{ and } \tilde{\psi} \leq 15 \\ -(x^2+y^2) - 1.0(u^2+v^2) - 1.0r + 10f_g & \text{otherwise} \end{cases} \tag{4.1}$$

where x, y, u, v, r, l, f_g and d_{obs} are the input variables of the function.

4.1.1 The training process

This study investigates the performance of a Proximal Policy Optimization (PPO) agent trained on a docking environment. The docking environment is designed to simulate an

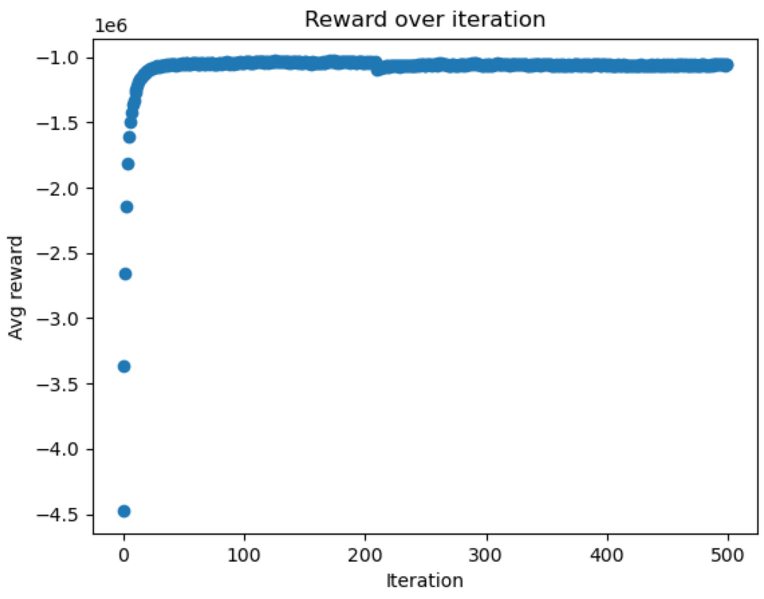


Figure 4.1: The average return during 500 iterations of training using reward function 1.

autonomous docking scenario where the agent must navigate to a target location while avoiding obstacles. The reward function used in the training process is a combination of distance from the target location, velocity, collision avoidance, and docking success/failure. The reward function is formulated to encourage the agent to reach the target location quickly while avoiding the land and achieving docking success. Figure 4.1 shows the average reward over iterations for 500 iterations, where the graph converges at around 25 iterations, which is the same as around 500,000 episodes. The results indicate that the PPO agent is effective in learning the optimal policy for the docking task. The trained agent demonstrated a high success rate in achieving docking while avoiding the land. This study highlights the effectiveness of PPO in autonomous navigation tasks and provides a valuable contribution to the field of autonomous robotics.

4.1.2 SHAP values

The study conducted an analysis of the contribution from each state to each action using SHAP values. A summary plot was employed, where the magnitude of each bar corresponds to the degree of contribution to the output state, which in the case of the Proximal Policy Optimization (PPO) agent is the action. Additionally, each bar is subdivided into sub-bars, represented by each class, which is proportional to the contribution from the state to each class (action).

The SHAP values were implemented in the test phase of the training scenario, specifically for the first 50 and 100 iterations. It is noteworthy that the agent had undergone

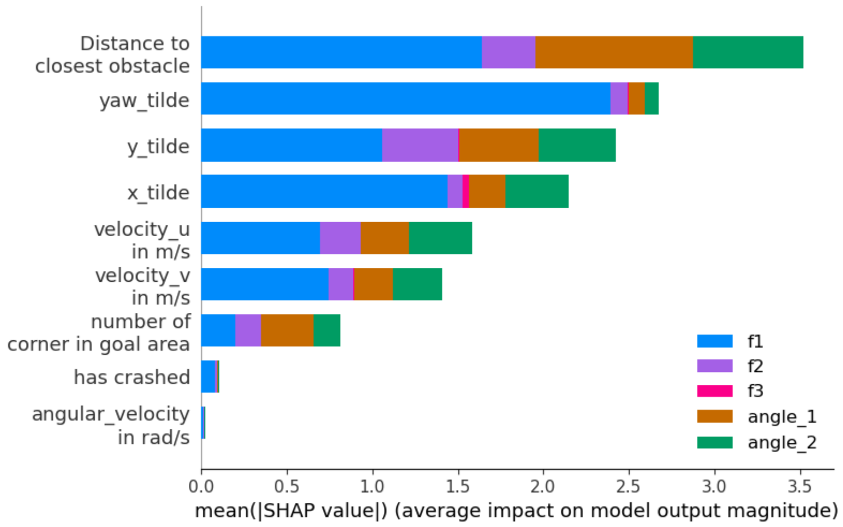


Figure 4.2: SHAP values after 50 iterations for reward function 1.

720 iterations or 14400000 episodes at the time of SHAP implementation. The outcomes of the SHAP values obtained from iteration 50 and 100 are presented in Figure 4.2 and Figure 4.3. Upon analyzing the selected episode after 50 iterations, it is evident that the observation parameter representing the distance to the nearest obstacle has the highest contribution towards action selection. This observation implies that the vessel is in proximity to land and, as a result, attempts to avoid collision. The $\tilde{\psi}$ parameter follows as the second most significant contributor to action selection, followed by $\tilde{y}$ and $\tilde{x}$. Conversely, the "has crashed" parameter and angular velocity are deemed the least significant features in action selection. The first action f_1, which dominates this figure, is predominantly governed by $\tilde{\psi}$ as the primary factor in decision making, followed by distance to obstacle and $\tilde{x}$.

Conversely, when examining the selected episode after 100 iterations, the figure portrays a distinct scenario, where f_3 evidently dominates the figure. This observation suggests that f_3 is the action primarily influenced by the observations. For this episode, the number of vessel corners in the goal area is identified as the most crucial feature in action selection, particularly when selecting f_3.

For figure 4.4 and 4.5, an average was taken over the first 50 and 100 episodes of the trained model, respectively. The first figure after 50 iterations depicts a more evenly distributed plot than the plots for each episode mentioned above. This observation is not surprising, as the average was obtained by aggregating the contributions. In this plot, it is apparent that for the azimuth forces, namely $\tilde{x}$, $\tilde{y}$, the number of corners in the goal area, the distance to the nearest obstacle, and the linear velocity (u) are all significant contributors in action selection. This could be attributed to the reward function where all these parameters hold importance. For the angles, $\tilde{x}$, $\tilde{y}$, the number of corners in the goal area, and the distance to the nearest obstacle are the prominent observations in decision making.

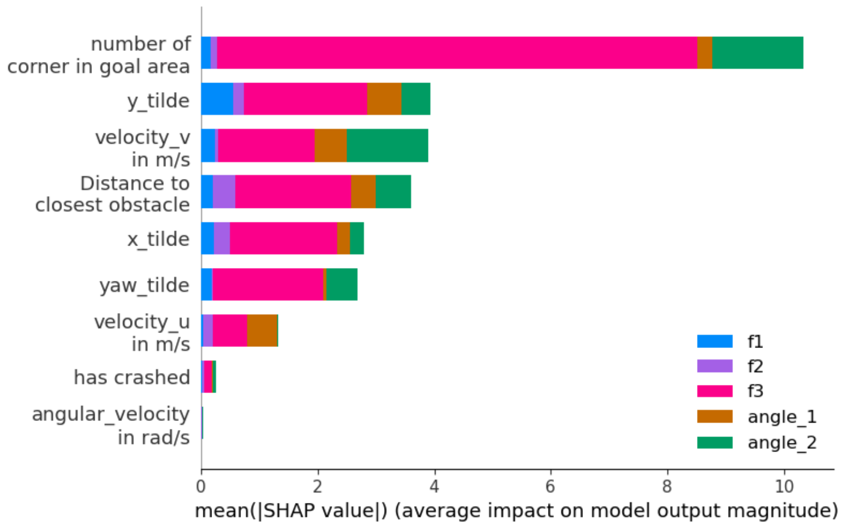


Figure 4.3: SHAP values after 100 iterations for reward function 1.

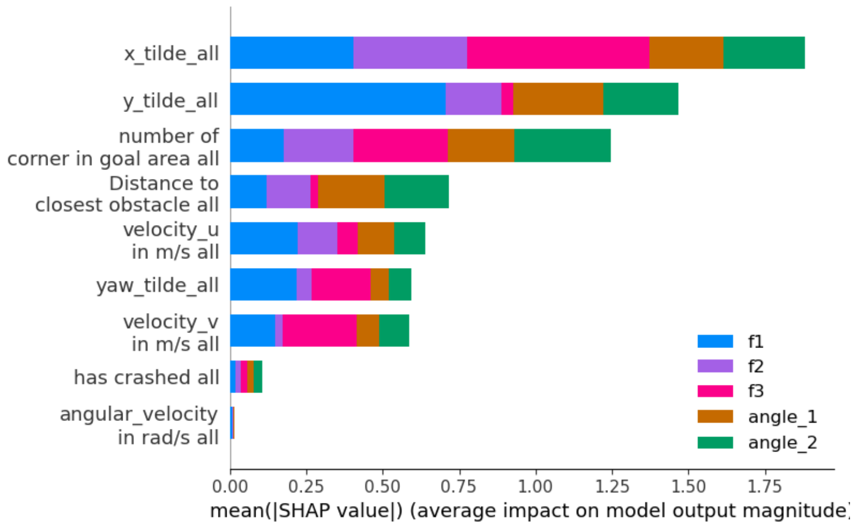


Figure 4.4: Average SHAP values after 50 iterations for reward function 1

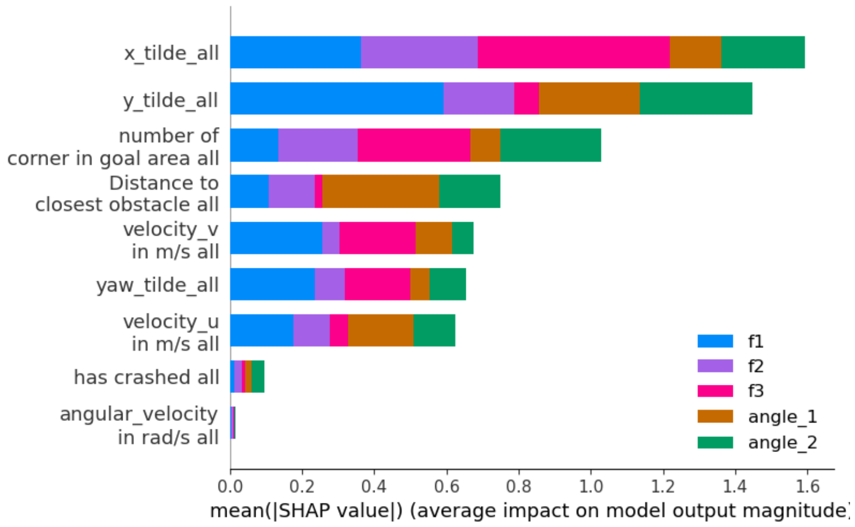


Figure 4.5: Average SHAP values after 100 iterations for reward function 1

After 100 iterations, the plot is even more uniformly distributed, but $\tilde{x}$ and $\tilde{y}$ become even more dominant. In both the 50 and 100 iterations, the "has crashed" parameter and angular velocity parameters are at the bottom of the list, which is unsurprising, as a trained model will tend to avoid crashing into the land, and the agent has likely learned that excessive turning is not the optimal strategy to approach the goal.

In Figures 4.6, 4.7, 4.8, 4.9 and 4.10 the relative feature contribution is plotted for each of the 5 actions during one episode. This episode has a length of 12 seconds and can be summarized as:

- The vessel starts 280 meters to the east of the goal area and 254 meter south for the goal and accelerates to get closer to the goal.
- The vessel reaches the goal area after 12 seconds.

As the size of the SHAP value depends on the value of the control input, the SHAP values at a certain time can not be compared to other instances of time. However, the relation between the SHAP values would be easier to compare. The results show that the features do not equally contribute to the control actions. For example, $\tilde{x}$ has a relatively higher contribution to the actions controlling the azimuth thruster 2 due to the initial position of the vessel. It was observed that $\tilde{x}$ and $\tilde{y}$ have a positive contribution in the first part of the episode for f_1 and f_2. However, at timestep 6, the contribution becomes negative, indicating that the velocities u and v become more important as the agent gets closer to the goal, and $\tilde{x}$ and $\tilde{y}$ become less important. The contribution of the first angle, α_1, is relatively low in the beginning, but increases during the episode, suggesting that the left thruster is not very active initially as the vessel turns towards the goal area. However, as the vessel gets closer to the goal, α_1 becomes more important to adjust the control action.

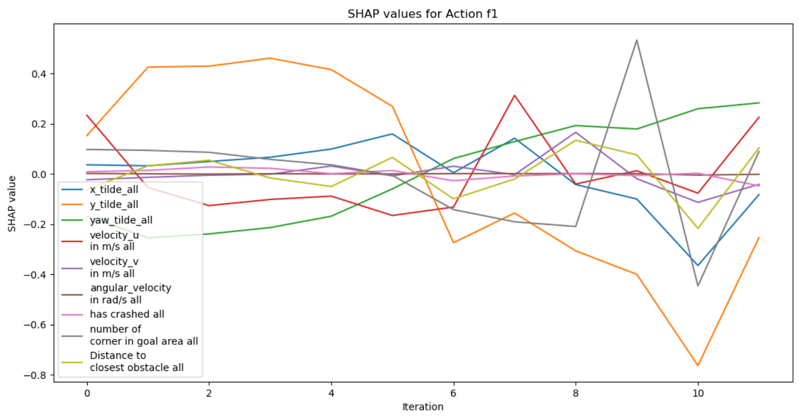


Figure 4.6: SHAP plot - Feature contribution to control action f_1 over one episode for reward function 1

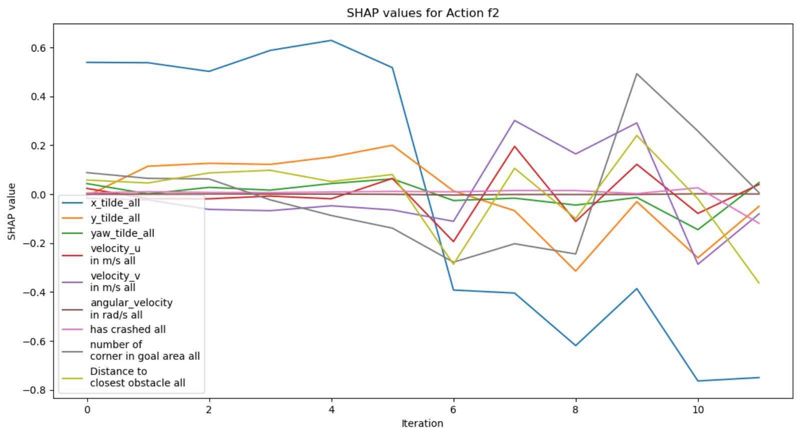


Figure 4.7: SHAP plot - Feature contribution to control action f_2 over one episode for reward function 1

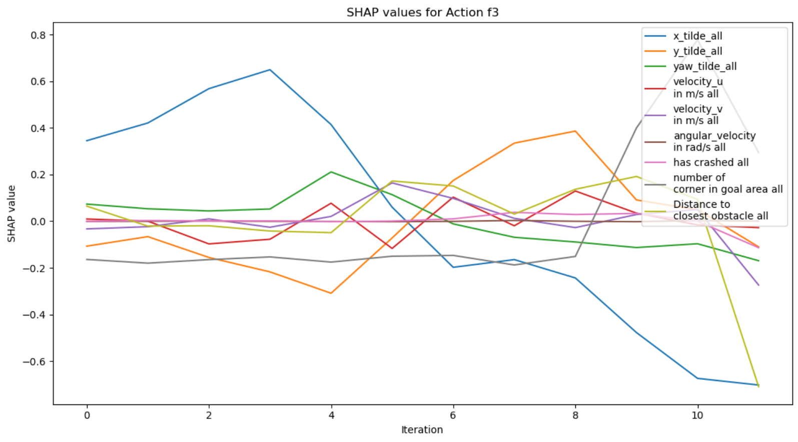


Figure 4.8: SHAP plot - Feature contribution to control action f_3 over one episode for reward function 1

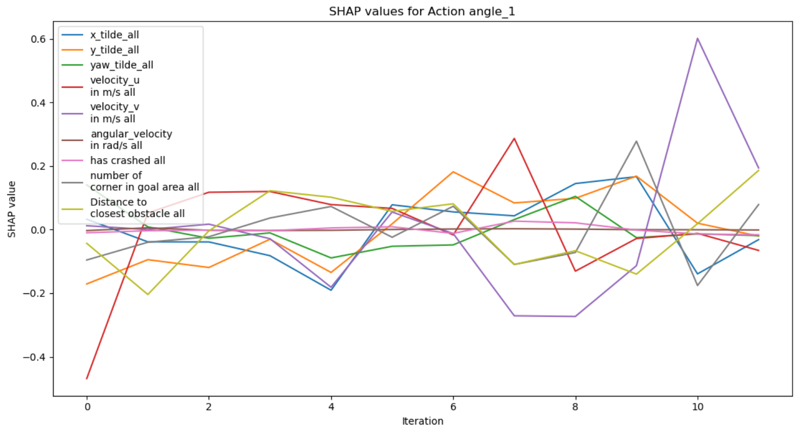


Figure 4.9: SHAP plot - Feature contribution to control action α_1 over one episode for reward function 1

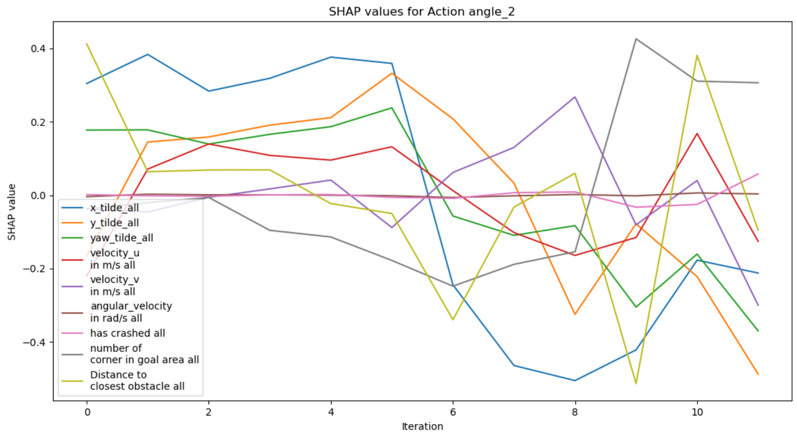


Figure 4.10: SHAP plot - Feature contribution to control action α_2 over one episode for reward function 1

In summary, it can be observed that the agent operates in an intuitive way and that the agent understood the intention of controlling the vessel towards the goal and keep it away from the land. The study finds that the features' contributions to control actions change over time, with velocities becoming more important as the agent gets closer to the goal, and certain angles becoming more important to adjust the control action. The study highlights the importance of understanding the contribution of each state to each action to improve the performance of the agent.

4.1.3 LIME values

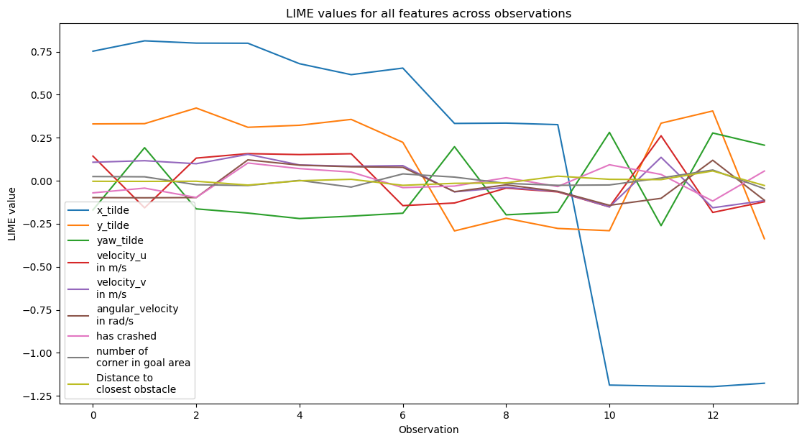


Figure 4.11: Lime values for one episode with reward function 1 after training

In this study, LIME values were utilized to examine the contribution of individual features to the decision-making process in an autonomous vessel docking task. The obtained LIME values were visualized in Figure 4.23, which displays the contribution of each of the nine features to the agent's decision-making process during a 13-timestep episode where the vessel was initialized 290 meters east and 254 meters south from the goal area. However, a drawback of LIME values is that they do not consider the overall sample space when constructing the model, which may lead to noisier feature attribution in general Løver et al. (2021). Although this effect is not very evident in this short episode, it can still be observed.

The plot in Figure 4.23 illustrates that $\tilde{x}$ and $\tilde{y}$ have a large positive contribution in the beginning of the episode, which gradually decreases. On the other hand, $\tilde{x}$ has a negative influence in the beginning when the vessel needs to adjust, indicating that the model output decreases as $\tilde{x}$ increases. Additionally, it can be observed that the LIME value for $\tilde{\psi}$ fluctuates at the end of the episode when the vessel attempts to adjust its angle ψ to terminate the episode. This behaviour is similar to what was reported for the SHAP values in the previous analysis.

4.2 Reward function 2

The agent that performs the docking with reward function 2 exhibits a similarity to the agent using reward function 1. Both agents utilize the same observation states, namely: $\tilde{x}$, $\tilde{y}$, $\tilde{\psi}$, u, v, r, l, f_g and d_{obs}. However, in this case, the velocities are subject to less penalization as a modification made after realizing that the yaw rate was penalized exces-

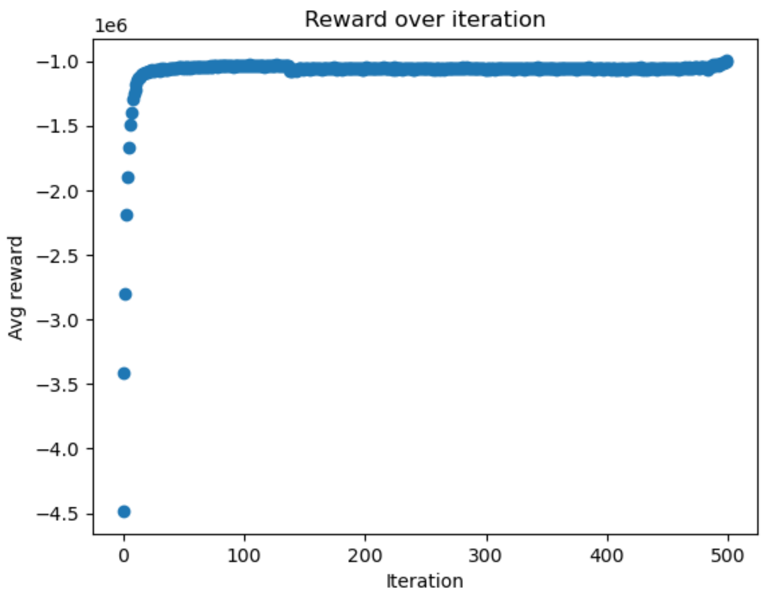


Figure 4.12: The average return during 500 iterations of training using reward function 2.

sively in the training of the agent with reward function 1. Additionally, a more stringent termination requirement has been imposed on the angle η.

The reward function implemented in the agent with reward function 2 is expressed as follows:

$$reward = \begin{cases} -(\tilde{x}^2+\tilde{y}^2) - 0.5\times|u| - 1*\times|v| - 0.8\times|r| + 10\times f_g - \tilde{\psi}^2 & \text{if } f_g \geq 1 \\ -(\tilde{x}^2+\tilde{y}^2) - 0.5\times|u| - 1*\times|v| - 0.8\times|r| + 10\times f_g - 1000 & \text{if } l = True \\ -(\tilde{x}^2+\tilde{y}^2) - 0.5\times|u| - 1*\times|v| - 0.8\times|r| + 10\times f_g + 1000 & \text{if } f_g = 5 \text{ and } \tilde{\psi} < 8 \\ -(\tilde{x}^2+\tilde{y}^2) - 0.5\times|u| - 1*\times|v| - 0.8\times|r| + 10\times f_g \text{otherwise} \end{cases} \tag{4.2}$$

The reward function is described in more detail above in chapter 3.

4.2.1 The training process

The agent trained using reward function 4.2 underwent 500 training iterations, and a visualization of the mean reward obtained for each iteration is presented in figure 4.12. The plot presented in Figure 4.12 showcases the performance of a Proximal Policy Optimization (PPO) algorithm trained for a docking task in a marine vessel. The average reward per iteration (with each iteration comprising of 20,000 episodes) is used to evaluate the algorithm's effectiveness. The plot illustrates the average reward values obtained over 500 iterations. The trend shows convergence towards a negative reward of -1.0e6 after approximately 25 iterations. This suggests that the algorithm may have reached a performance

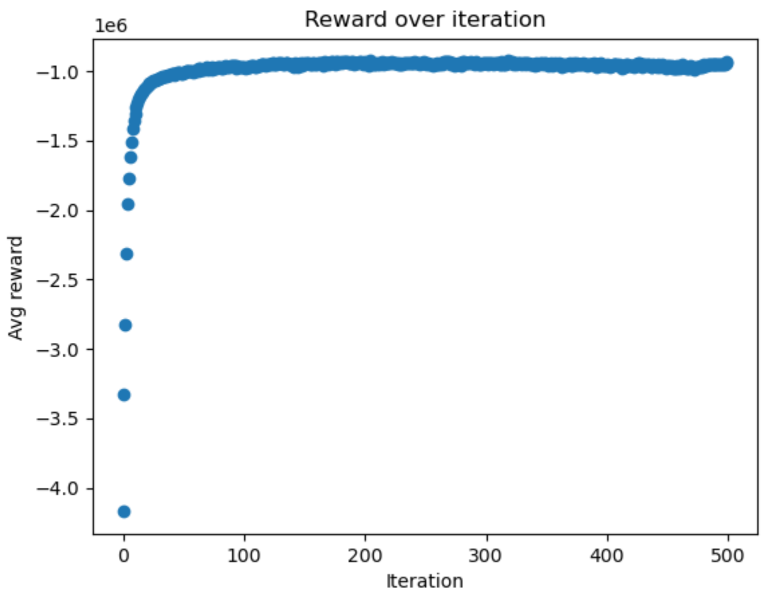


Figure 4.13: The average return during 500 iterations of training using reward function 2 and learning rate of 0,005.

limit or encountered a challenging part of the problem that it was unable to overcome. However, further analysis revealed that the agent had only reached a local optimum, with the vessel crashing instead of reaching the goal area. This is in contrast to the desired behavior achieved by the agent trained with reward function 1. The change in requirement for the angle ϕ from 15 degrees to 8 degrees may have led to this outcome. In an attempt to address this issue, lower learning rates were tested, as shown in Figure 4.13 and 4.14. Nonetheless, the agent still failed to learn the desired behavior.

4.2.2 SHAP values

Figure 4.15 and Figure 4.16 present a SHAP bar plot for iteration 50 and iteration 100. Figure 4.15 presents the feature contributions from each state to the action for iteration 50. The plot indicates a relatively evenly distributed feature attribution in comparison to the other SHAP summary plots for a single iteration. For the first action, f_1, the number of corners in the goal area was the most significant contributor, followed by $\tilde{y}$. For the second action, f_2, $\tilde{x}$ had a comparatively high impact, followed by $\tilde{\eta}$ and the distance to the closest obstacle. The third action, f_3, was relatively equally influenced by all the parameters except angular velocity and velocity v. For angle 1, $\tilde{\eta}$ and the distance to the closest obstacle had the most influence, and for the last action, angle 2, $\tilde{x}$ had a significantly higher impact than the other states.

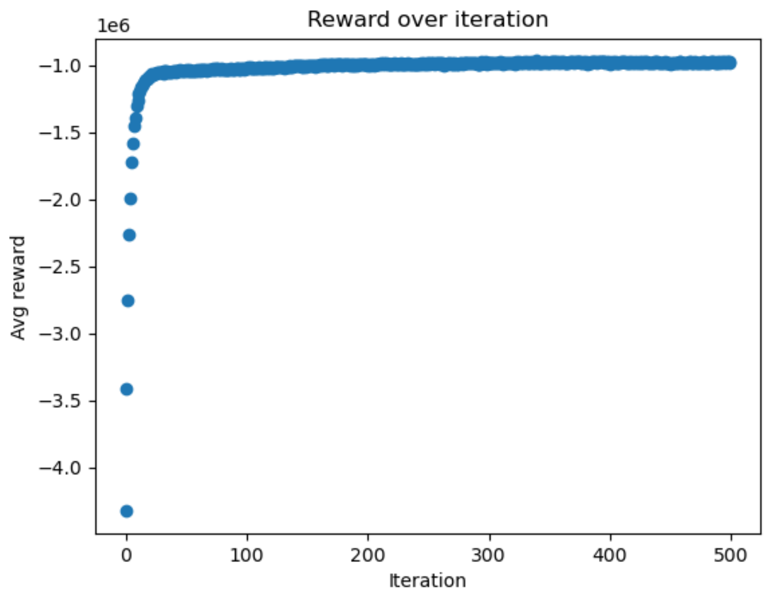


Figure 4.14: The average return during 500 iterations of training using reward function 2 and learning rate of 0,001.

Notably, $\tilde{x}$ had a higher contribution compared to the other parameters and mostly influenced f_2 and angle 2. This finding is not surprising as the ship was initially positioned to the right of the goal area, and the azimuth thruster on the right side, controlled by angle 2 and f_2, had to be used to turn the ship to the left and towards the goal area.

For iteration 100, in Figure 4.16 it can be observed the action f_1 is mostly affected by the number of corners in the goal area, f_g and the velocity v. For the second action, f_2, the state $\tilde{x}$ has the highest contribution, followed by the number of corners in the goal area, $\tilde{y}$, the velocity u, the distance to the closest obstacle d_{obs}, and the heading angle $\tilde{\psi}$. The third force, f_3, has relatively low contributions from the states, with velocities u and v being the most influential. For the angle parameters, angle 1 is mainly influenced by $\tilde{x}$ and v, while angle 2 is mostly affected by $\tilde{x}$ and the number of corners in the goal area.

There are some notable insights that can be drawn from this plot. Firstly, it is interesting to observe that $\tilde{x}$ has a much larger contribution than $\tilde{y}$ relatively. This could be attributed to the possibility that the vessel started a few meters more to the right than usual, resulting in $\tilde{x}$ having a greater impact on the reward function than $\tilde{y}$. Secondly, it is evident that angle 2 and f_2 significantly dominate the feature bars. This observation supports the first claim that the vessel might have started a few meters more to the right than usual, resulting in thruster 2, which is placed on the right side, being activated more to steer the vessel more to the left.

Figure 4.17 present the average SHAP values over 100 iterations, and is a follow-up to the plot presented in Appendix A, which showcases the average SHAP values after 50

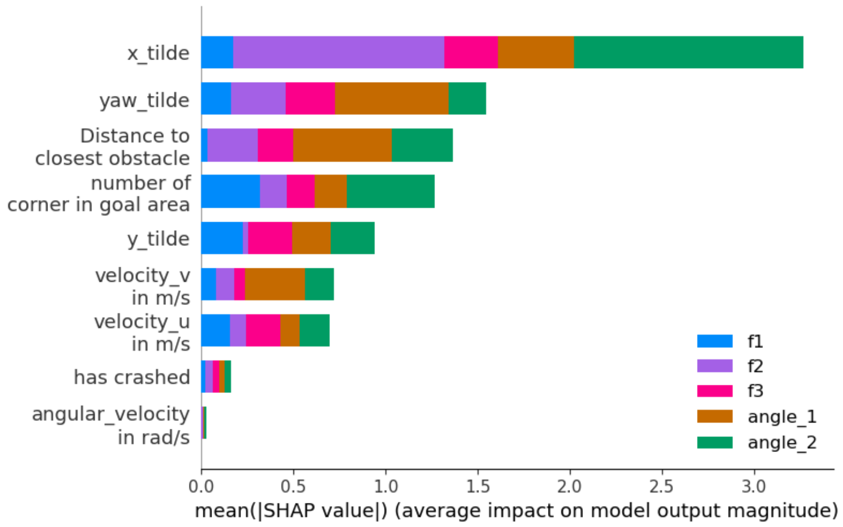


Figure 4.15: SHAP values after 50 iterations for reward function 2.

iterations. However, as the plot after 50 iterations is almost identical to the plot after 100 iterations, it has not been included in the current discussion. In contrast to the relationship between feature contributions seen in Figure 4.5 for reward function 1, Figure 4.17 reveals a different trend, with the state $\tilde{\psi}$ ranking as the second most influential feature followed by the distance to the closest obstacle. This trend is sensible as this reward function places a greater emphasis on achieving a specific angle ψ at the end of the episode.

In Figures 4.18, 4.19, 4.20, 4.21 and 4.22, the contribution of each feature to the reward function is plotted for each of the 5 actions during a single episode. The length of this episode is 18 seconds and can be summarized as follows:

- The vessel starts 287 meters to the east of the goal area and 258 meter south for the goal and accelerates to approach the goal. After 5 seconds, the vessel starts to slow down as it approaches the goal area.
- The vessel crashes and hit land after 18 seconds which terminates the episode. At this point, the vessel has two corners inside the goal area but has a heading angle, $\tilde{\psi}$, of 90 degrees.

In summary, Figures 4.18, 4.19, 4.20, 4.21 and 4.22 exhibit more noise compared to the previous reward function. However, the tendency of $\tilde{x}$ contributing more in the first half of the episode is still observed, and $\tilde{\psi}$ seems to influence more towards the end of the episode. The velocity, v, is particularly noisy, and the observations during the episode reveal that the velocity fluctuates a lot. One possible explanation for the failure of the agent to reach the desired behavior could be the fluctuation of velocity during the episode, which may have made it difficult for the agent to learn and maintain a stable trajectory towards the goal area. Another factor could be the relatively high requirement for the angle ψ at

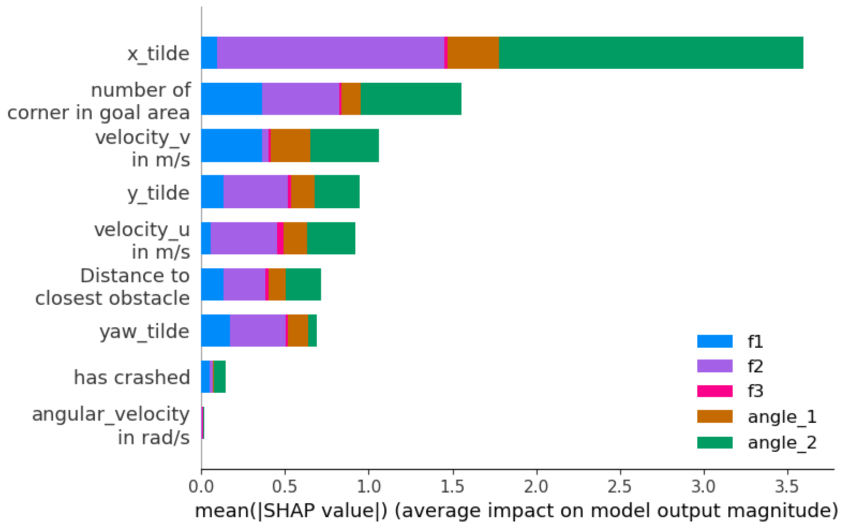


Figure 4.16: SHAP values after 100 iterations for reward function 2.

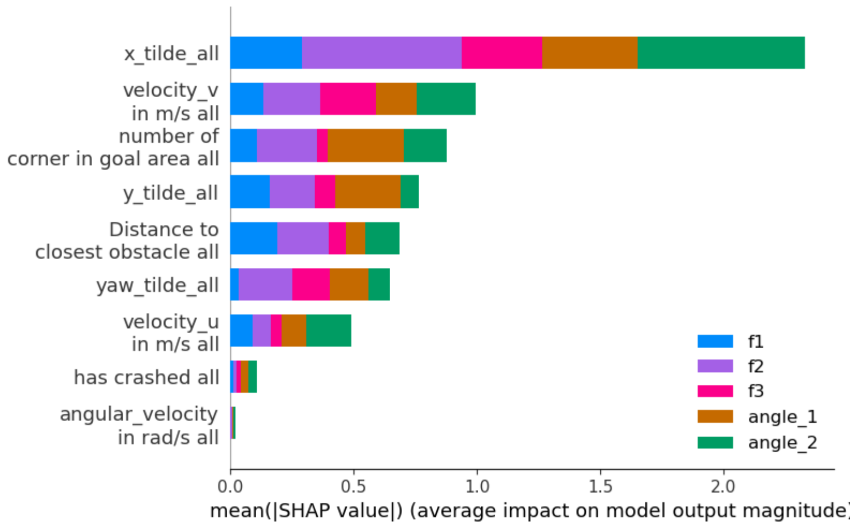


Figure 4.17: Average SHAP values after 100 iterations for reward function 2

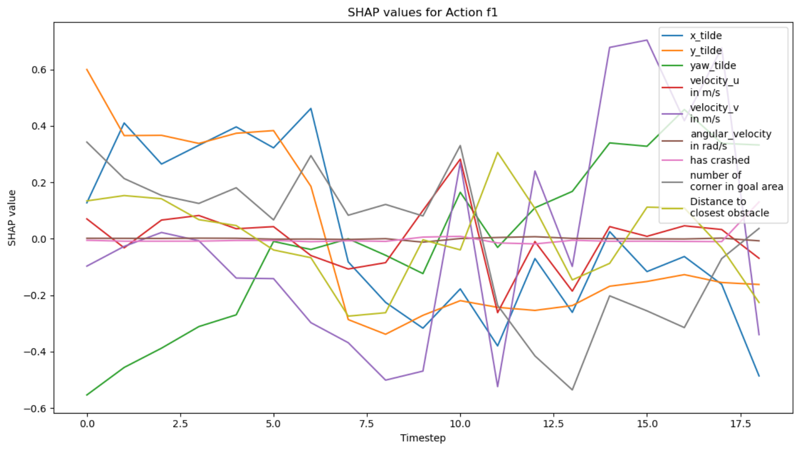


Figure 4.18: SHAP plot - Feature contribution to control action f_1 over one episode for reward function 2

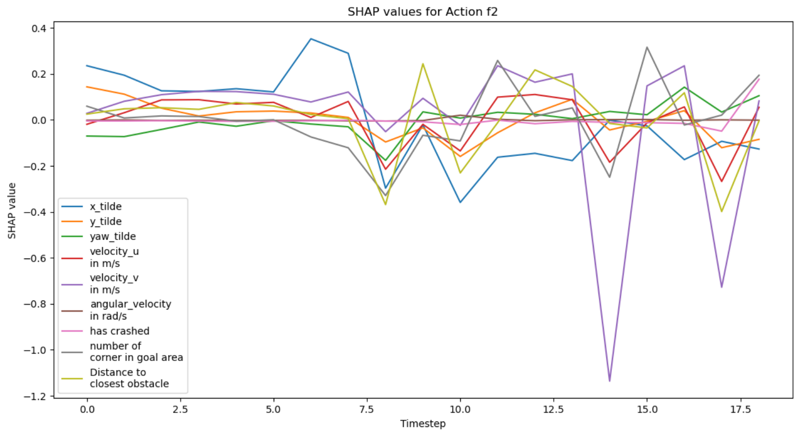


Figure 4.19: SHAP plot - Feature contribution to control action f_2 over one episode for reward function 2

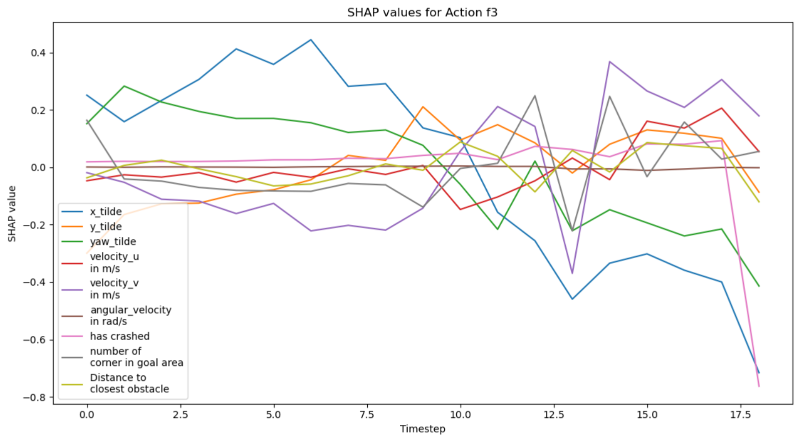


Figure 4.20: SHAP plot - Feature contribution to control action f_3 over one episode for reward function 2

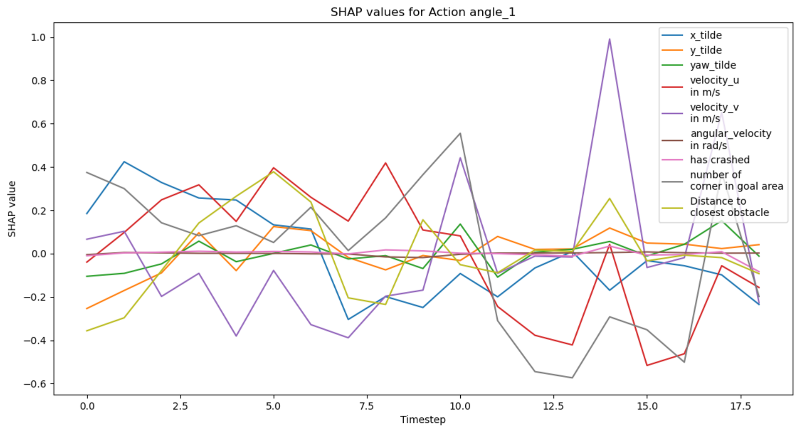


Figure 4.21: SHAP plot - Feature contribution to control action α_1 over one episode for reward function 2

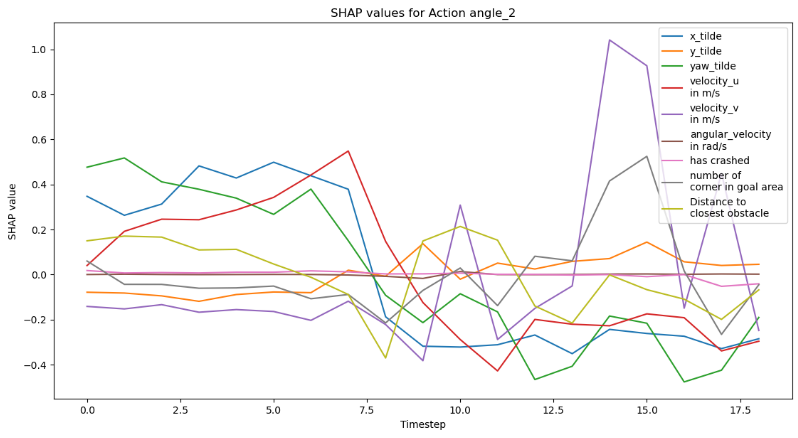


Figure 4.22: SHAP plot - Feature contribution to control action α_2 over one episode for reward function 2

the end of the episode, as indicated by the higher contribution of $\tilde{\psi}$ in the latter part of the episode.

4.2.3 LIME values

Figure 4.23 illustrates the LIME values for reward function 2, which appear to be more erratic compared to the previous scenario. Despite this, a pattern emerges where $\tilde{x}$ and $\tilde{y}$ contribute positively in the first half of the episode, while they contribute negatively in the second half. This observation is consistent with the previous finding where $\tilde{x}$ was found to have a greater influence in the earlier part of the episode. It is worth noting that the noisy nature of the LIME values may be attributed to the high level of variability observed in the velocity feature during the episode, which can cause fluctuations in the LIME values. Given these findings, it is possible that the fluctuating velocity may be contributing to the difficulty of achieving the desired behavior by the agent. Further investigation may be necessary to confirm this possibility.

4.3 Discussion

To summarize the results and discussions presented above, several key findings can be highlighted:

- The feature $\tilde{x}$ generally contributes more in the beginning of the episode and has a particularly strong influence on the actions controlling azimuth thruster 2. This is likely due to the vessel starting to the east of the goal area and azimuth thruster 2 being required to turn the vessel to the west.

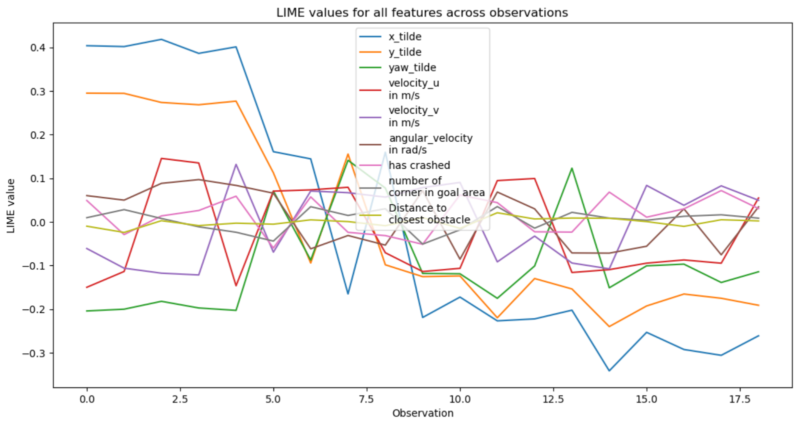


Figure 4.23: Lime values for one episode with reward function 2 after training

- In the case of reward function 1, the velocities u and v should have been penalized more, as the vessel tended to prioritize reaching the goal quickly rather than maintaining a lower speed. This may be attributed to the observation scaling in the environment implementation, where the velocities' maximum and minimum values were set high. Adjusting these values to be lower would have led to greater punishment for high velocities.
- For reward function 2, the agent was unable to reach the goal. It is presumed that because the velocities were not penalized as severely, they became too high, causing the vessel to approach the goal area too quickly. With a stricter angle constraint, the vessel continued towards the land and was unable to adjust the angle in time to avoid crashing.
- An important observation from the results is that the agent was able to successfully achieve the desired behavior in scenario 1, where reward function 1 was employed, whereas it failed to do so in scenario 2, where the proposed improved reward function 2 was used. This underscores the significance of fine-tuning reward functions to ensure effective performance of an agent in a given environment.

5 Conclusion

This research study aimed to investigate the use of the Proximal Policy Optimization (PPO) algorithm for the docking of an autonomous marine vessel, with the objective of reaching the goal area at the right angle. Furthermore, the study aimed to evaluate the contribution of feature variables to the agent's decision-making process using SHAP and LIME explainable AI techniques.

The results of the study indicate that the PPO can be successful in solving the docking. The performance converged after approximately 25 iterations for the two implemented reward functions. The agent was able to learn the optimal policy to achieve the desired goal of reaching the goal area at the right angle for the first scenario. The successful results of the PPO algorithm suggest that it is an effective method for solving similar problems and that it can be used in various autonomous systems.

Moreover, the study also evaluated the contribution of feature variables to the agent's decision-making process using SHAP and LIME values. The SHAP and LIME values provided valuable insights into the agent's decision-making process, which can help in improving the agent's performance in future iterations. The results showed that the distance to the goal area in general were the most significant features contributing to the agent's decision-making process.

The successful results of the first implementation of the PPO algorithm in the docking, coupled with the analysis using SHAP and LIME values, suggest that explainable AI techniques can provide valuable insights into the decision-making process of autonomous systems. By providing interpretable and transparent models of the decision-making process, the SHAP and LIME techniques enable researchers and developers to gain insight into the model, the choices made and enables the ability to improve the model. However, it is important to note that the reward function should be further tuned to reach the desired behaviour with a stricter requirement for the angle and more penalty for the velocities.

The findings of this study have important implications for the field of autonomous systems, particularly in the area of docking. The use of the PPO algorithm can be extended to other similar problems, and the SHAP and LIME techniques can be utilized to evaluate the contribution of feature variables to the decision-making process of the agent. The insights gained from the SHAP and LIME analyses can be used to improve the performance of the agent and optimize the system further.

In conclusion, this study has demonstrated a promising use of the PPO algorithm for a docking of an autonomous marine vessel, with the objective of reaching the goal area at the right angle. The study also showed the possibilities and given insight by using SHAP and LIME techniques to evaluate the contribution of feature variables to the decision-making process of the agent. The use of SHAP and LIME to explain the reasoning of the PPO agent in this study was an initial attempt, and further research is needed to simplify its use and ensure its consistency. Additionally, to enhance the robustness of the autonomous docking system, more work is required which involves implementation of ocean currents, wind, and moving obstacles.

6 Limitations, assumptions and further work

In this chapter the limitations and assumptions of this thesis is presented and discussed. In addition, a brief section of possible further work is presented. The chapter consists of the following sections:

- Section 6.1: Presents and discusses the limitations and assumptions of this master thesis.
- Section 6.2: Presents possible further work.

6.1 Limitations and assumptions

This study has a few limitations that must be considered when interpreting the results. Firstly, the model was implemented on a MacBook Air with the M1 chip. This was a limitation as a MacBook Air has a limited capacity compared to a high-performance computing environment, such as a cluster of GPU's. This impacted the duration of the training and the speed of convergence.As a result these simulations may not be possible to generalize for more complex or larger scale problems. Secondly, Pytorch introduced a limitation in terms of scalability and computational efficiency as it has some limitations in design and parallel processing. Thirdly, a limitation was the time limit for this thesis. Reinforcement learning with many states and a set of five actions with implemented SHAP values is quite slow. As there were a few technical difficulties during the work with this thesis, the actual training was started later than planned and the results, which lead to fewer iterations in the training. Ideally, the training phase should had been started earlier in order tests more reward function to achieve better convergence. Finally, as the PPO was only implemented on a single case study of a docking scenario, further evaluation and testing would be required in order to establish robustness and generalizability of this approach. These limitations are important to acknowledge and consider for future work. Particularly, future studies with the same scope should aim to address these limitations using more powerful computing resources and look for alternative frameworks in order to achieve better results. In addition, further evaluation and testing would be needed to establish the robustness of this approach for other domains and use cases.

6.2 Further work

A goal for this thesis was to implement counterfactuals. However, during the fall of 2022 Vilde Gjærum wrote her Ph.D on counterfactuals and was not satisfied with the results of counterfactuals in reinforcement learning as it was very ineffective and became very complex on systems with more than one state. In addition, there are very little to no sources on how to actually implement counterfactuals in reinforcement learning problems. Due to the limitations of time and the complexity this was, therefore, not implemented. But it would be interesting to see how counterfactuals could improve and to see it implemented on reinforcement learning in the future. An example implementation of this could be to define a custom function, similar to the one implementing LIME and SHAP, that takes the model and input data as inputs and then generates counterfactual explanations based on the desired methodology. Pseuso code is provided below.

Algorithm 2 Explain with Counterfactual

```
Input: Model $model$, state log $state\_log$, feature names $feature\_names$, action log $action\_log$, environment $env$, method $method$
Output: Counterfactual explanations
1: Convert statelog from numpy array to a tensor: $statelogtensor \leftarrow T.fromnumpy(statelog).float()$
2: if method = 'simple' then
3:     Generate counterfactual explanations using the simple method: $counterfactualexplanations \leftarrow simplecounterfactualexplanation(model, statelogtensor, actionlog)$
4: else if method = 'advanced' then
5:     Generate counterfactual explanations using the advanced method: $counterfactualexplanations \leftarrow advancedcounterfactualexplanation(model, statelogtensor, actionlog)$
6: end if
7: Plot the counterfactual explanations: $plotcounterfactualexplanations(counterfactualexplanation$
```

In this pseudo code the 'simple counterfactual explanation' and 'advanced counterfactual explanation' functions would be custom functions that would be needed to define in order to implement the desired counterfactual explanation method. The 'plot counterfactual explanations' function would be used for visualization.

Bibliography

ABB, 2022. Automatisk bremseassistent gir sikrere nødstopp av store skip .

Anderson, J.A., 1987. Alan turing and the mathematical objection. British Journal for the Philosophy of Science 38, 499–524.

Anyoha, R., 2017. The history of artificial intelligence .

Bossom, N., Yudkowsky, E., 2009. The ethics of artificial intelligence .

Brownlee, J., 2019. A gentle introduction to the rectified linear unit (relu) URL: `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-network`

Cooper, J., 2019. Growing capability and capacity in the bluetech startup ecosystem. Marine Technology Society Journal 53, 11–14.

Darpa, 2016. Explainable artificial intelligence (xai) URL: `https://www.darpa.mil/program/explainable-artificial-intelligence`.

Edmonds, M., Zhu, Y., 2020. People prefer robots to explain themselves – and a brief summary doesn't cut it .

Fossen, T.I., 1994. Guidance and control of ocean vehicles. University of Trondheim, Norway, Printed by John Wiley & Sons, Chichester, England, ISBN: 0 471 94113 1, Doctors Thesis .

Fossen, T.I., 2011. Handbook of marine craft, hydrodynamics and motion control. John Wiley $ Sons.

Fossen, T.I., 2021. Lecture notes ttk 4190 guidance, navigation and control of vehicles - chapter 9 control forces and moments .

Gjærum, V.B., Strümke, I., Løver, J., Miller, T., Lekkas, A.M., 2023. Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications. Neurocomputing 515, 133–144.

Hansen, H.L., Nielsen, D., Frydenberg, M., 2002. Occupational accidents aboard merchant ships. Occupational and environmental medicine 59, 85–91. doi:10.1136/oem.59.2.85.

Hashimoto, H., Nishimura, H., Nishimaya, H., Higuchi, G., 2021. Development of ai-based automatic collision avoidance system and evaluation by actual ship experiment. ClassNK Technical Journal, Special feature: Autonomous Ships .

Johansen, T.A., Fossen, T.I., 2013. Control allocation—a survey. Automatica 49, 1087–1103.

Karalus, J., Lindner, F., 2022. Accelerating the learning of tamer with counterfactual explanations .

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization URL: https://arxiv.org/abs/1412.6980, doi:10.48550/ARXIV.1412.6980.

lines, M.O., 2022. World's first successful sea trial of autonomous sailing on a commercial container ship voyage URL: https://www.mol.co.jp/en/pr/2022/22007.html.

Løver, J., 2021. Explaining a Deep Reinforcement Learning Agent Using Regression Trees. Master's thesis. NTNU.

Lundberg, S.M., Lee, S.I., 2017. A unified approach to interpreting model predictions. Advances in neural information processing systems 30.

Løver, J., Gjærum, V.B., Lekkas, A.M., 2021. Explainable ai methods on a deep reinforcement learning agent for automatic docking. IFAC-PapersOnLine 54, 146–152. URL: https://www.sciencedirect.com/science/article/pii/S2405896321014889, doi:https://doi.org/10.1016/j.ifacol.2021.10.086.

Martinsen, A.B., Lekkas, A.M., Gros, S., 2019. Autonomous docking using direct optimal control. IFAC-PapersOnLine 52, 97–102. URL: https://www.sciencedirect.com/science/article/pii/S2405896319321755.

McCarthy, J., Minsky, M.L., Rochester, N., Shannon, C.E., 1955. A proposal for the Dartmouth Summer Research Project on Artificial Intelligence. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1904/1802.

Midtbø, G.H., 2020. Automatic ferry enters regular service following world-first crossing with passengers onboard URL: https://www.kongsberg.com/no/maritime/about-us/news-and-media/news-archive/2020/first-adaptive-transit-on-bastofosen-vi/.

Molnar, C., 2020. Interpretable machine learning. Lulu. com.

Murdoch, E., Clarke, D., Dand, I.W., Glover, B., Busoniu, L., Yang, S.X., 2014. Master's guide to berthing - the standard club .

Nilsson, N.J., 2014. The quest for artificial intelligence: A history of ideas and achievements. Cambridge University Press.

OpenAI, 2023a. Openai .

OpenAI, 2023b. Openai baselines. GitHub repository .

Pearl, J., 2009. Causality. Cambridge university press.

Ribeiro, M.T., Singh, S., Guestrin, C., 2016. " Why should i trust you?" Explaining the predictions of any classifier.

Rorvik, E.L.H., 2020. Automatisk dokking av et autonomt overflatefartøy. Master's thesis. NTNU.

Rothblum, A.M., 2000. Human error and marine safety. volume 7.

Russell, S.J., Norvig, P., 2010. Artificial intelligence: A modern approach. Pearson Education.

Salyer, K., 2022. Autonomous shipping is making waves .

Schulman, J., 2020. Proximal policy optimization. OpenAI URL: `https://openai.com/blog/openai-baselines-ppo/`.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

Shapley, L.S., 1997. A value for n-person games. Classics in game theory 69.

Sorfonn, I., Holmlund-Sund, M., 2018. World's first autodocking installation successfully tested by wärtsilä. Wärtsilä Corporation URL: `https://www.wartsila.com/media/news/26-04-2018-world-s-first-autodocking-installation-successfully-tested`

Sutton, R.S., Barto, A.G., 2018. Reinforcement learning: An introduction. MIT press.

Up, S., 2018. Spinning up. OpenAI .

Veitch, E., Andreas Alsos, O., 2022. A systematic review of human-ai interaction in autonomous ship systems. Safety Science 152, 105778. URL: `https://www.sciencedirect.com/science/article/pii/S0925753522001175`, doi:`https://doi.org/10.1016/j.ssci.2022.105778`.

Wärtsilä, 2018. Wärtsilä tests automated dock-to-dock technology on folgefonn. Ship Technology URL: `https://www.ship-technology.com/news/wartsila-testing-technology-folgefonn/`.

Yu, E.Y., 2020. Coding ppo from scratch with pytorch .

Appendix

A Appendix A - SHAP values

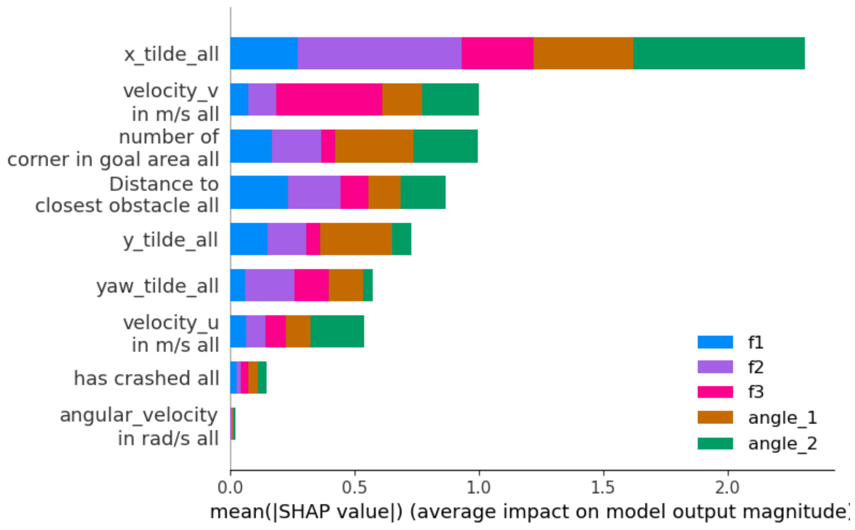
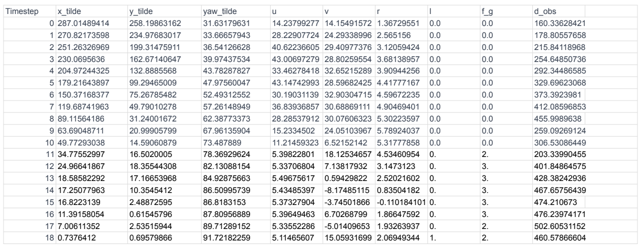


Figure 6.1: Average SHAP values after 50 iterations for reward function 2

B Appendix B - Observations for the episode used for SHAP values per timestep for reward function 2



Timestep	x_tilde	y_tilde	yaw_tilde	u	v	r	l	f_g	d_obs
0	287.01489414	258.19863162	31.63179631	14.23799277	14.15491572	1.36729551	0.0	0.0	160.33628421
1	270.82173598	234.97683017	33.66657943	28.22907724	24.29338996	2.565156	0.0	0.0	178.80557658
2	251.26326969	199.31475911	36.54126628	40.62236605	29.40977376	3.12059424	0.0	0.0	215.84118968
3	230.0695636	162.67140647	39.97437534	43.00697279	28.80259554	3.68138957	0.0	0.0	254.64850736
4	204.97244325	132.8885568	43.78287827	33.46278418	32.65215289	3.90944256	0.0	0.0	292.34486585
5	179.21643897	99.29465009	47.97560047	43.14742993	28.59682425	4.41777167	0.0	0.0	329.69623068
6	150.37168377	75.26785482	52.49312552	30.19031139	32.90304715	4.59672235	0.0	0.0	373.3923981
7	119.68741963	49.79010278	57.26148949	36.83936857	30.68869111	4.90469401	0.0	0.0	412.08596853
8	89.11564186	31.24001672	62.38773373	28.28537912	30.07606323	5.30223597	0.0	0.0	455.9989638
9	63.69048711	20.99905799	67.96135904	15.2334502	24.05103967	5.78924037	0.0	0.0	259.09269124
10	49.77293038	14.59060879	73.487889	11.21459323	6.52152142	5.31777858	0.0	0.0	306.53086449
11	34.77552997	16.5020005	78.36929624	5.39822801	18.12534657	4.53460954	0.	2.	203.33990455
12	24.96641867	18.35544308	82.13088154	5.33706804	7.13817932	3.1473123	0.	3.	401.84864575
13	18.58582292	17.16653968	84.92875663	5.49675617	0.59429822	2.52021602	0.	3.	428.38242936
14	17.25077963	10.3545412	86.50995739	5.43485397	-8.17485115	0.83504182	0.	3.	467.65756439
15	16.8223139	2.48872595	86.8183153	5.37327904	-3.74501866	-0.110184101	0.	3.	474.210673
16	11.39158054	0.61545796	87.80956889	5.39649463	6.70268799	1.86647592	0.	3.	476.23974171
17	7.00611352	2.53515944	89.71289152	5.33552286	-5.01409653	1.93263937	0.	2.	502.60531152
18	0.7376412	0.69579866	91.72182259	5.11465607	15.05931699	2.06949344	1.	2.	460.57866604

Figure 6.2: Observations for the episode used for SHAP values per timestep for reward function 2