

Doctoral thesis

Doctoral theses at NTNU, 2023:134

Vilde Benoni Gjærum

Machine Learning in Robotics: Explaining Autonomous Agents in Real-Time

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics



Norwegian University of
Science and Technology

Vilde Benoni Gjærum

Machine Learning in Robotics: Explaining Autonomous Agents in Real-Time

Thesis for the Degree of Philosophiae Doctor

Trondheim, April 2023

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



Norwegian University of
Science and Technology

NTNU

Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

© Vilde Benoni Gjærum

ISBN 978-82-326-5788-9 (printed ver.)
ISBN 978-82-326-6955-4 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2023:134

Printed by NTNU Grafisk senter

Summary

Artificial intelligence (AI) and machine learning (ML) offer a number of benefits in multiple applications within the field of robotics, such as computer vision, object grasping, motion control, and planning. Although AI methods can boost performance in many robotic tasks, these methods' utility value is limited by the fact that humans struggle to understand how these methods operate. AI or ML models that are so complex that we cannot understand them are called *black boxes*. Our lack of understanding of these black boxes can lead to a lack of trust in systems working perfectly well or too much trust in systems that might not be trustworthy. Additionally, understanding the black boxes can help us improve them, detect their weaknesses and thus better assess which scenarios the black box can be applied to in a safe manner and ensure that the black box obeys laws and regulations. These are some of the shortcomings of AI that the field of explainable artificial intelligence (XAI) addresses.

This thesis presents topics related to XAI in robotics. The main part of the thesis is a collection of four peer-reviewed papers, two journal papers and two conference papers. Additionally, one submitted conference paper is included. In addition to the paper collection, the first part of the thesis contains an introduction to the thesis as well as an introduction of the main topics of the thesis, namely ML in robotics, XAI and linear model trees (LMTs). This first part provides context to the publications and puts the different publications in relation to each other. In this thesis, LMTs are used as an XAI method. LMTs are decision trees (DTs) with a linear prediction function in the leaf nodes. The LMTs divides the input space into distinct regions and fits a linear function to each region, and the LMTs thus makes out a piece-wise linear function approximator. The LMTs can be used as an XAI method by approximating the black box and subsequently analysing the LMT to gain a better understanding of the black box. The first thing that needs to be done when using LMTs for explainability is to build the tree to approximate the black box. To do so, we must gather data from the world and collect the corresponding output responses from the black box. We then use this dataset to build the LMT in a supervised manner. The validity of the explanations depends on how similar the LMT is to the black box, so great care must be taken when gathering the dataset and building the tree. We found that introducing domain knowledge to the building process improved the tree's accuracy and building time.

We use the LMTs as a post-hoc, model-agnostic surrogate model, which means that the LMTs is an XAI method that mimics any type of black box model that is already built. In addition to being able to give explanations in the form of

feature attributions and counterfactuals, LMTs also is an explanation in itself since the trees' structure and linear prediction functions represent the black box model in a simpler manner. We show that the LMTs are capable of generating feature attribution and counterfactuals in real-time, even for complex, robotic applications.

Once the explanations have been generated, we must make sure the explanations are effectively communicated to the user of the AI system. How an explanation best can be communicated depends on the system to be explained, which application the system is used on, and who the recipient of the explanation is. We suggest two different visualizations of feature attributions to two different end-users based on their background knowledge and characteristics.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work has been carried out at, and funded by, the Department of Engineering Cybernetics (ITK) as part of the EXAIGON project (project number 304843), with Associate Professor Anastasios Lekkas as my main supervisor and Professor Timothy Miller and Dr Inga Strümke as my co-supervisors. The Research Council of Norway funded the research stay at the University of Melbourne as a part of the EXAIGON project.

Acknowledgements

This thesis is the result of over three years of hard work, support from the people that have been around me, and just a little dash of luck. I want to use this opportunity to thank my supervisors, colleagues, friends, and family for all of their support.

First of all, I would like to thank my main supervisor, Tasos, for giving me the opportunity to do this PhD. I am grateful for all our discussions and your valuable input and feedback. I want to thank Tim for sharing your knowledge and inviting me to Melbourne. I am so grateful for how including you and all the lovely people at the Agent lab were; I truly had a blast! The biggest hug goes to Inga. Your ability to motivate and see the break in the clouds whenever things were rough has been so invaluable to me. Thank you for all the support and advice, but more importantly, for being such a good friend. I would also like to thank Associate Professor Ole Andreas Alsos for sharing your knowledge and insights regarding the visualizations of the explanations.

I am grateful to my previous classmate, Ella-Lovise Hammervold Rørvik, for letting me use the results of her master's thesis as the black box to explain and for happily helping me set it all up and answering any questions I had. I have had the pleasure of working with two master's students, Nicolas Blystad Carbone and Jakob Løver, during the course of my thesis. Nicolas' idea for using LMTs for vehicle control changed the entire course of my PhD, and the collaboration with Jakob resulted in two publications. Thank you both for all the interesting discussions!

I have been so lucky to have Bjørn, Sindre, Thomas, and Andreas as my office mates. Thank you for all the GeoGuessr matches, coffee breaks, distractions and

motivation, and for making D444 a great place to be. Katrine, I really appreciate our friendship and am so glad we got to know each other.

I want to thank my family for all the love and support. To dad, for all the encouragement and for always having the time to talk whenever I call. To mom, for all the support and care packages. To Wiggo, for always cracking up jokes. To my sister, Lina, for always listening to my worries and cheering on my accomplishments. To my brother, Henrik, and his family, for welcoming me to Trondheim 8 years ago and for all the time we have spent together since then. And to Karl's family for being so warm and welcoming.

Finally, the biggest thank you of them all goes to Karl for the endless love and support and for always having more faith in me than I tend to have myself.

Started making it.
Had a breakdown.
Bon appétit !

James Acaster

Table of Contents

List of Figures	vi
List of Tables	vii
1 Introduction	1
1.1 Motivation	1
1.2 Publications	3
1.3 Contributions at a glance	6
1.4 Outline	6
2 Background	7
2.1 Machine learning in robotics	7
2.2 The need for explanations in AI-based robotics	10
2.3 Explainable artificial intelligence	12
2.4 Linear model trees	17
2.5 Test applications	24
2.5.1 The docking problem	24
2.5.2 The inverted pendulum problem	25
3 Contributions and discussion	26
3.1 Building LMTs	26
3.2 LMTs as an XAI-method	29
3.3 Visualization of explanations in robotics	30
4 Conclusions and further work	34

5 Publications	35
5.1 Paper A	36
5.2 Paper B	53
5.3 Paper C	70
5.4 Paper D	94
5.5 Paper E	121
Bibliography	136

List of Figures

1 Overview of the relation between the black box model, the XAI method, and the end user.	3
2 Publication and contribution overview.	4
3 Deep neural network	8
4 Reinforcement learning and control loop	11
5 Characterization of XAI-methods	13
6 Illustration of a decision tree.	18
7 How regression trees divide the input space into regions.	19
8 How linear model trees divide the input space into regions.	19
9 Overview over the pipeline of using linear model tree as explainer for a reinforcement learning agent	23
10 Example of a run in the simulated docking environment.	25
11 Illustration of the inverted pendulum from Paper E [20].	25
12 Examples of different visualizations of systems and explanations	33

List of Tables

1	Terms in reinforcement learning	10
2	Overview over explanation types	15
3	Motivations for transparency	16
4	Characteristics of LMTs affecting interpretability	24

1 Introduction

1.1 Motivation

Recent developments within the field of robotics include solutions utilizing AI methods, and more specifically ML methods. In [44], ML is defined as “.. a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty” while IBM¹ describes ML as “.. a branch of AI and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy”. In essence, ML allows the machines to learn patterns in the data useful for solving some problem or task by themselves. Whereas [44] argues that the best tools to attack ML-problems come from probability theory, [7] argues that most ML-problems can and be solved accurately by the use of optimization techniques. Whichever of these two approaches is taken to solve the ML-problem, considering the swooningly amount of data we produce every day, it is easy to see that there is a seemingly endless amount of problems for ML to wrestle with.

There are numerous examples of successful reinforcement learning (RL) agents, perhaps most prominently within computer and board games, such as in [6], [43], [45], and [61]. In [6], agents playing hide and seek developed new strategies to counter the opposing teams’ strategy multiple times, leading to a continuously evolving game of cat-and-mouse. In [43], agents learned how to play Atari games, and in [45] five RL-agents started beating beginner human players in the competitive, team-based, real-time-strategy game Dota 2. As famously presented in [61], an RL-algorithm mastered to play the classic games of Chess, Go, and Shogi at a superhuman level.

Apart from RL agents’ ability to learn and dominate in games, they also achieved significant feats in the field of robotics. In [24], a sample-efficient RL algorithm was used to train a Minitaur robot to learn how to walk from scratch, while RL agents in [56] learned different gaits for different simulated robots. A RL algorithm controlling a five-fingered humanoid hand learned to solve the Rubik’s cube. In [33], a model-free RL-algorithm was trained to master control of over 20 different simulated physical tasks. In [49], a RL-agent learned to master the manipulation of a robotic lever, and in [46], the very same algorithm that was used in [45] was used to control a dexterous hand. Additionally, there are numerous examples of RL being used to perform maritime operations [3, 38, 39, 41, 59, 71].

¹<https://www.ibm.com/cloud/learn/machine-learning>

As AI methods become ubiquitous and get an increasingly important role in our everyday lives, it becomes clear that for us to be able to predict how these methods will impact us, we need to study these algorithms [48]. The Defence Advanced Research Project Agency (DARPA)², a research agency of the United States Department of Defence, defined the term XAI as

“AI systems that can explain their rationale to a human user, characterize their strengths and weaknesses, and convey an understanding of how they will behave in the future.” [23].

DARPA differs between XAI and other similar terms (such as interpretable, comprehensible, or transparent AI) to focus more on making the AI systems human-understandable through communicating effective explanations. This research initiative was started to address the critical shortcomings of AI. The European Union’s XAI project³ focuses on explaining opaque AI/ML models to enable better collaboration between humans and machines, ensuring good communication, trust, clarity, and understanding. As AI systems are entering our everyday life through industries such as healthcare, banking, advertising, transportation, and so on, the need for understanding these systems arises. EXAIGON (Explainable AI systems for Gradual Industry Adoption)⁴ is a Norwegian research initiative that concerns with the challenges that must be addressed before trustworthy AI systems can be deployed in social environments and business-critical applications. XAI has not only gotten the attention of research communities, but also many companies are seeing the commercial value of XAI and are offering their solutions, such as IBM’s open-source toolkit, *AI Explainability 360* [4], Interpretable AI⁵, Google’s XAI tool⁶, Seldon⁷, the open source explainable AI Toolkit (XAITK)⁸.

Take the field of healthcare. If you go to the doctor and get diagnosed with some disease, you most likely would ask your doctor on what basis they concluded with this. Given an AI system with the same, or higher even, skill level, you would still be left with the same questions. This need for explanation is not limited to the field of healthcare. Quite the opposite, this reasoning behind the decisions made by an AI system is wanted (and in most cases, also needed) for all safety critical systems and systems that must comply with regulations and legislation. This need for robotic applications becomes especially clear for autonomous systems interacting with or working close to humans. However, understanding the system

²<https://www.darpa.mil/program/explainable-artificial-intelligence>

³<https://xai-project.eu/>

⁴<https://www.ntnu.edu/exaigon>

⁵<https://www.interpretable.ai/>

⁶<https://cloud.google.com/explainable-ai>

⁷<https://www.seldon.io/>

⁸<https://xaitk.org/>

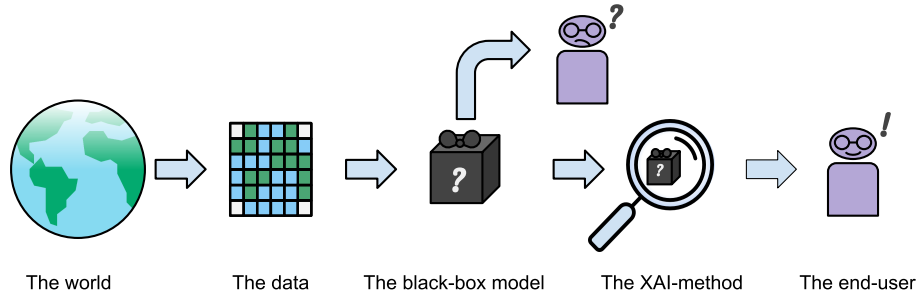


Figure 1: Overview of the relation between the black box model, the XAI method, and the end user.

is crucial for applications with risks associated with equipment or the robot, such as autonomous vehicles. AI methods of which we do not understand the inner workings are often referred to as *black boxes*, not because what is inside them is secret or hidden but simply because we do not know how the method uses the input to compute the output. Figure 1 outlines a simple overview of how the black box, the XAI method, and the users relate to each other is shown. We gather data that ideally (but rarely) perfectly represents the world on which the AI model, or the black box, is built. Since we do not understand how this black box works, the end user is left wondering and perhaps sceptical. The goal of XAI is to answer the questions the end-user has so that they understand and trust the black box.

1.2 Publications

This thesis is based on two journal papers and three conference publications (where two are accepted and one under review). Additionally, two conference publications that are not included in this thesis were published during the PhD.

The contributions of this thesis can be divided into three main parts, namely: 1) suggesting tactics and emphasizing considerations that must be taken into account when building the LMTs, 2) demonstrating and appraising the LMTs' utility value as an XAI method for robotic applications, and 3) considering the end-user's perspective and suggesting user-adapted visualizations of the feature attributions. An overview of the main contributions and the publications linked to the different contributions is given in Figure 2. A detailed discussion regarding the different contributions and how they relate to each other is given in Section 3.

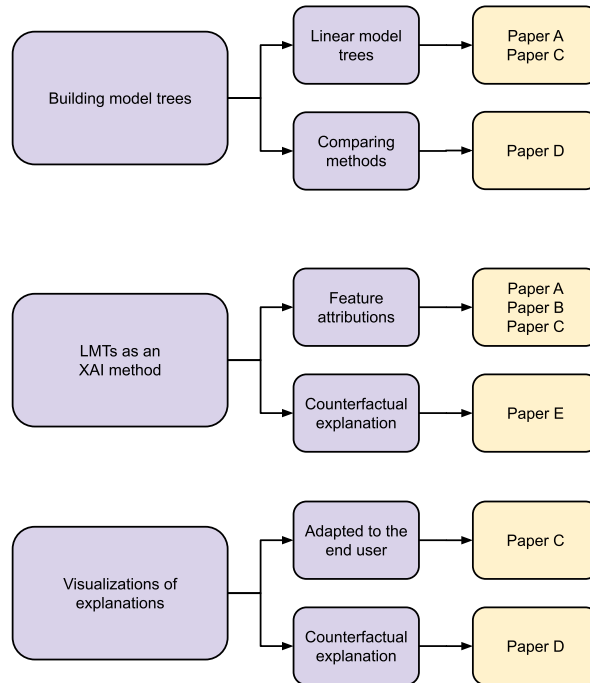


Figure 2: Publication and contribution overview.

Conference publications

Paper A

[17]: **Vilde B. Gjørnum**, Ella L. H. Rørvik and Anastasios M. Lekkas. ‘Approximating a deep reinforcement learning docking agent using linear model trees’. In: *The 19th European Control Conference(ECC)(2021)*, pp. 1465-1471, doi: <https://doi.org/10.23919/ECC54610.2021.9655007>

Paper B

[35]: Jakob Løver, **Vilde B. Gjørnum** and A. M. Lekkas. ”Explainable AI methods on a deep reinforcement learning agent for automatic docking”. In: *14th IFAC Conference on Control Applications in Marine Systems, Robotics,*

and *Vehicles (CAMS)* (2021) doi: <https://doi.org/10.1016/j.ifacol.2021.10.086>

Paper E

[20]: **Vilde B. Gjørum**, Inga Strümke, Anastasios M. Lekkas, and Timothy Miller, "Real-Time Counterfactual Explanations For Robotic Systems With Multiple Continuous Outputs". Accepted to: *The 22nd World Congress of the International Federation of Automatic Control (IFAC WC)* (2023) doi: <https://doi.org/10.48550/arXiv.2212.04212>

Journal publications

Paper C

[18]: **Vilde B. Gjørum**, Inga Strümke, Ole Andreas Alsos, and Anastasios M. Lekkas. "Explaining a deep reinforcement learning docking agent using linear model trees and user adapted visualizations". In: *Journal for Marine Science and Engineering* 9(11) 1178 (2021). doi: <https://doi.org/10.3390/jmse9111178>

Paper D

[19]: **Vilde B. Gjørum**, Inga Strümke, Jakob Løver, Timothy Miller, and Anastasios M. Lekkas. "Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications". In: *Neurocomputing* 515 (2022), pp. 133–144. doi: <https://doi.org/10.1016/j.neucom.2022.10.014>

Conference publications not included in this thesis

[25]: Anne Håkansson, Aya Saad, Akhil Anand, **Vilde B. Gjørum**, Haakon Robinson and Katrine Seel. "Robust Reasoning for Autonomous Cyber-Physical Systems in Dynamic Environments". In: *Proceedings of the 25th International Conference on Knowledge Based and Intelligent Information and Engineering Systems (KES)* 192 (2021) pp. 3966-3978, doi: <https://doi.org/10.1016/j.procs.2021.09.171>

[2]: Akhil Anand, Katrine Seel, **Vilde B. Gjørnum**, Anne Håkansson, Haakon Robinson and Aya Saad. "Safe Learning for Control using Control Lyapunov Functions and Control Barrier Functions: A Review". In: *Proceedings of the 25th International Conference on Knowledge Based and Intelligent Information and Engineering Systems(KES) 192 (2021)* pp. 3987 - 3997, doi: <https://doi.org/10.1016/j.procs.2021.09.173>

1.3 Contributions at a glance

This thesis has contributed to the literature at the intersection of robotics and XAI in the following ways:

- Proposed the novel idea of using LMTs as a surrogate XAI method for robotic applications by extracting feature attributions from the LMT's linear prediction functions.
- Demonstrated that LMTs could be used as a post-hoc explanation method for complex, robotic applications.
- Proposed a method for extracting counterfactual explanations utilizing the LMT's structure.
- Proposed two significantly different ways of visualizing the explanations to two end-users taking their background knowledge, situation, and needs into account.
- Compared the LMTs to two other post-hoc, model-agnostic XAI methods in light of robotic applications.
- Compared four different methods for building model trees (MTs) for the specific application of using them as post-hoc explanation methods for robotic applications.

1.4 Outline

The rest of the thesis is structured as follows: Section 2 contains the background on the topics covered in the the publications. Section 3 presents and discusses the contributions of the thesis. In Section 4, the conclusion is given, along with some reflections on possible further work. Finally, in Section 5, the publications written as a result of the work done in this thesis is presented. The references for each publication are not included in the bibliography at the end of the thesis.

2 Background

In this chapter, the necessary background is presented. Without AI there is no need for XAI, so in Section 2.1, the AI methods, and more specifically the ML models, that we want to understand better is introduced. In Section 2.3, an introduction to XAI is given and in Section 2.2, what must be taken into account when applying XAI to robotic applications is presented. In Section 2.4, a thorough walkthrough of LMTs, the XAI model used in all the publications, is given. Lastly, the docking problem, which was used as the test application in all the publications, is briefly introduced in Section 2.5.1.

2.1 Machine learning in robotics

ML can be divided into three main subfields, namely supervised learning (SL), unsupervised learning (UL), and RL. In SL, the goal is to learn a mapping between the inputs to the outputs given a set of correct input-output pairings. If the output is categorical, it is called a classification problem, and if the output is continuous it is called a regression problem. The second branch of ML is UL. In UL, there is no correct or incorrect output. The goal is rather to discover interesting patterns in the data. Lastly, in RL, the AI agent discovers the best (and worst) actions through trial and error and a reward signal. Unlike in SL, where every prediction must be labelled either correct or wrong, it is only the "overall goal" that must be defined in RL. As mentioned in the introduction, RL has made great achievements within both games and robotics, but there are also numerous examples of RL being used to perform maritime operations [3, 38, 39, 41, 59, 71].

When referring to ML, including these three subfields, we often mean deep learning (DL). DL is a subfield of AI that allows us to avoid spending time on finding a mathematical model that accurately describes our system, hard-coding rules based on apriori knowledge, or creating complex, specialized algorithms for the particular task. DL make us of deep neural networks (DNNs), a function approximator with a structure that is inspired by the human brain. A DNN is a neural network (NN) with at least two hidden layers, as shown in Figure 3. At first glance, DL may seem like the perfect short-cut to everything we could ever want from an AI based system, and some even argue that RL is the way to go for exactly that [63]. However, this "short-cut" through the use of DL includes obstacles like tedious fine-tuning of parameters and reward or cost functions, carefully choosing model structures, prolonged training times requiring a lot of computational resources, and crucial preprocessing of data to ensure they

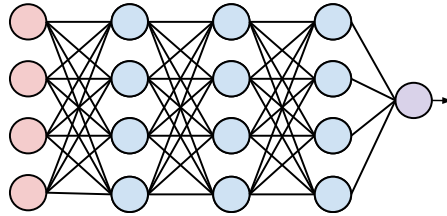


Figure 3: Illustration of a deep neural network with a four-dimensional input layer (red nodes), three hidden layers (blue nodes), and a one-dimensional output layer (purple node).

adequately represent the real world.

Even though all three methods can be useful in robotics, RL is an especially fitting methodology for problems where we know what we want the agent to do but do not necessarily know what is the best way to achieve said goal, which is the case for many robotic tasks. Additionally, for many applications, labelled data is very expensive, or even not feasible at all, to require.

RL solves sequential decision-making problems, meaning that the outcome (or the behaviour) of the system does not come from a single prediction but rather many decisions made consecutively. Many problems fit this description, especially within robotics, such as driving a car, flying a drone or controlling an industrial robot. The *agent* includes the physical or simulated entity that interacts with the environment as well as the model (which for deep RL (DRL) will be some sort of NN) that it is controlled by. The *environment* is the world the agent lives in, either the real world or a simulated one. The *state* gives a complete description of the world, whereas the *observation* is an incomplete description of the world to that the agent has access. Say that the agent is an autonomous vacuum robot, and your apartment is its environment. The observation will be all the information the little robot can gather through its sensors. Even if the sensors could gather information about everything relevant, most real-world problems will still only be partially observable due to modelling errors or noise in the environment or in the sensors. It should be noted that state and observation often are used interchangeably, even though there are distinct differences. The *action space* includes the various ways the agent can interact with its environment, which for example, can be through controlling the motors. The agent has a goal, something it wants to achieve in the environment, which for the vacuum robot will be to make the floors in your apartment clean and dust-free. The agent is learning through testing different actions in different states and receiving *rewards* from

an *reward function*. The reward function is crucial for the agent’s learning and should describe the agent’s goal or desired behaviour. For the vacuum robot, this could be the percentage of the area cleaned. The agent wants to maximize the *return*, which is the accumulated reward over one *episode*. One episode is all the time steps from the starting point in the environment until the environment terminates either due to failure or success or after a number of time steps have passed. In Figure 4a, the RL loop where the agent receives a state from the environment, performs one action and then receives both a reward and a new action is shown. As pointed out by [66], the boundary between the agent and the environment is not always clear and may vary from one application to another. The agent does not usually include the entire physical body of the robot since the motors and sensors often are included in the environment. The corresponding control loop is shown in Figure 4b. Even though the relationship between the terms is not completely one-to-one, the controller can be thought of as the agent (or the policy), the plant as the environment, the control input as the state, and the measured output of the plant as the state. An *experience* includes one state, one action, and one reward signal. A *trajectory* is the sequence of all experiences during one episode. The *policy*, sometimes referred to as the controller, is a mapping from state to action and is what the agent uses to determine which action to take. An overview of the different terms commonly used in RL is given in Table 1. The goal of the RL-algorithm is to find a policy that maximizes the return for every episode. The main RL-problem can be described as

$$\pi^* = \max_{\pi} J(\pi), \quad (1)$$

where π^* is the optimal policy and the J is the expected return which can be denoted as

$$J(\pi) = \mathbb{E}[R(\pi)] = \int_{\tau} P(\tau|\pi)R(\tau), \quad (2)$$

and the probability of a trajectory is denoted as $P(\tau|\pi)$, and R is the return. There exists numerous RL-algorithms applying different tactics to find π^* , such as [58, 57, 62, 28].

RL excels in applications where it is extremely challenging to model the system dynamics but relatively easy to model the objective (and thus the reward function) given that a reliable training environment is available or that it is possible to collect experience from a real-world implementation without any risk for safety. This applies to problems where the system or the environment is very complex, unknown factors affect the system, or the dynamics are changing.

Table 1: Terms in reinforcement learning

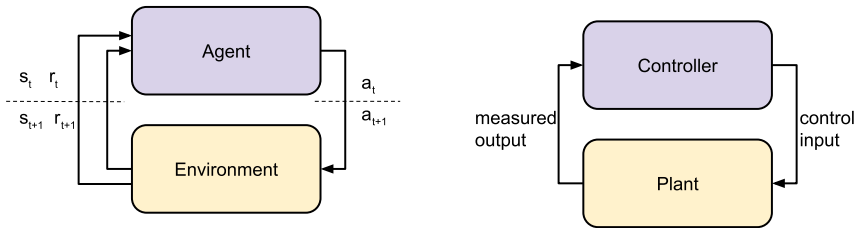
Term	Symbol	Meaning
Agent		The entity that interacts with the environment
Policy	π	A function that maps states to actions
Action	a_t	Describes how the agent interacts with the environment at time t
State	s_t	Describes the environment at time t
Observation	o_t	Describes the agent’s observation of the environment at time t
Reward	r_t	The value of either the state or the state, action pair at time t assigned by the reward function
Return	R	The accumulated rewards throughout an episode
Trajectory	τ	All the states, actions, and rewards for one episode
Episode		All the time steps from the beginning until the end of the environment

2.2 The need for explanations in AI-based robotics

Even though DL-models have proven themselves to be highly accurate, there are also numerous examples of cases where the DL-model is unfair or untrustworthy. As stated in [40]:

“... fairness is the *absence of any prejudice or favouritism toward an individual or group based on their inherent or acquired characteristic*”.

In [51], a logistic regression classifier was trained to highlight that if we are not careful, the classifier might learn the correlations within the dataset instead of the causations. The classifier was trained to classify images of huskies and wolves and to showcase this issue, a biased dataset was handpicked such that all the pictures of wolves had snow in the background while the pictures of huskies did not. By only looking at the model’s accuracy, it seemed like the classifier did well. Without any additional information regarding how the model made its prediction, more than a third of the people asked in the study trusted the model. When presented with the prediction along with the most important features (which turned out to be the background) for the prediction, only around



(a) The reinforcement learning loop, based on [66].

(b) The control loop

Figure 4: Reinforcement learning and control loop

1/10 of the people trusted the model. In [8], it is showcased that gender bias can be found in the word embeddings of natural language processing models. In some cases, such as with king/man and queen/woman, gendered connections are desired and useful, but gendered connections along the lines of doctor/man and nurse/woman, computer programmer/man and homemaker/woman are harmful. In [21], it is shown that two methods trying to remove sexist word embeddings are not reliable in doing so since they turned out to merely cover them up. In [11], two datasets used for benchmarking facial recognition models are found to have an overwhelmingly disproportionate share of lighter-skinned individuals. When three commercially used gender classification systems are tested on an evenly distributed testing set, it becomes clear that the systems are significantly better at recognising light-skinned individuals compared to dark-skinned individuals and also significantly better at recognising male faces compared to female faces, both of which significantly inhibits the applicability of the classification system. These examples not only highlight the fact that accuracy alone is not enough for us to decide whether or not a model is trustworthy but also that the foundation of the decision-making is crucial to help us gauge the level of trust to place in the system.

ML is capable of making sense of the large amounts of data coming from the sensors on robotic systems, thus creating more intelligent robots that can factor in more information regarding the environment it is operating in and adapt accordingly. ML, and more specifically DL, has boosted the performance of many robotics systems, and it is clear that unfair ML-systems can lead to unfair, or even dangerous, autonomous robots. This issue is not limited to the issue of fairness. The utility value of these robots heavily depends on whether or not they can gain our trust. Letting AI models we do not have a complete picture of how works control robotic systems in real-world applications raises concerns for

safety since any misstep could pose a huge risk to the safety of not only people but also the robot itself as well as other equipment and valuables. Applying AI methods to real-world robotic use cases requiring safety insurance or stability guarantees is a challenging task. It is imperative that human end users at different levels are guaranteed that an AI-based controller will perform as expected. Even with meticulous testing, we cannot cover the entire state space and thus not be sure that there does not exist an edge scenario where the controller fails. Traditional control methods are not except for failure but the fact that we have a fundamental understanding of how they work allows us to trust them enough to use them even in safety-critical applications. For those scenarios where AI methods are deployed, and more specifically when they involve black boxes, taking explainability into account may allow the user to [1, 55]:

- **discover** the unknown dynamics, how to optimize the performance, new aspects of situational awareness and so on.
- have better **control** over the system. Having a general understanding of how a system works is crucial for controlling the system. If we are going to have a human in the loop, the human must understand the system, for example, to know when to take control of the system.
- **justify** the system’s decisions and behaviour. Systems that are working for or alongside humans must be able to communicate the reasoning behind their decisions. Additionally, we must be able to justify that the system complies with legislation.
- **improve** the system in terms of optimality or by pinpointing weaknesses that should be improved.
- achieve the appropriate amount of **trust** not only for the system as a whole but also for the system’s reaction to different situations it may encounter. If a perfect system cannot gain our trust, it will not be used, but on the other hand, trusting a system too much can be dangerous.

2.3 Explainable artificial intelligence

Even though the use of complex ML methods brings multiple benefits such as high performance in complex tasks, the models that we refer to as black boxes also bring in some challenges of their own. The field of XAI attempts to address these challenges through ”unboxing” the black boxes, i.e. by understanding *why* a model made the prediction it did. Different XAI methods apply different tactics

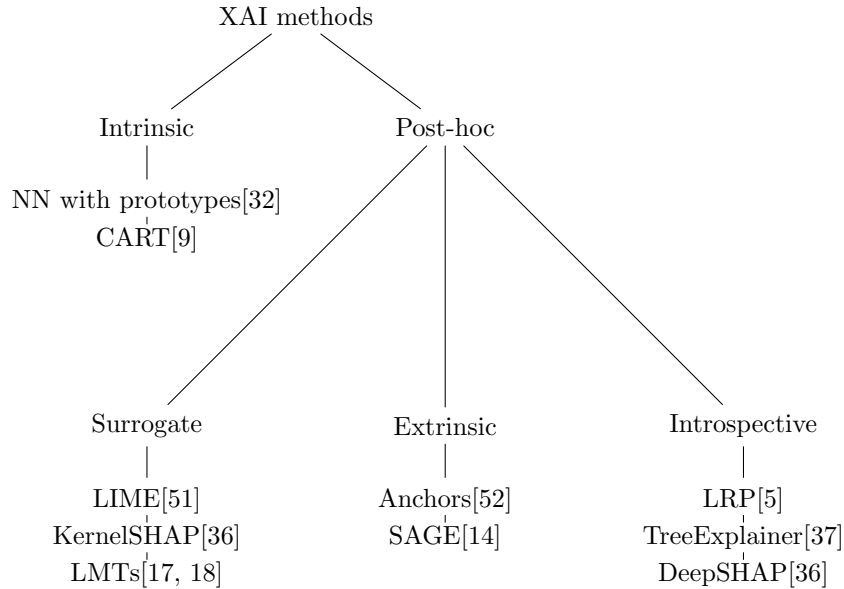


Figure 5: Characterization of XAI-methods with examples of each type.

for unboxing the black box at hand. Since XAI aims to explain how extremely complex models work to humans the field has quite a challenging translation task at hand as well. There are many benefits of uncovering these models’ inner workings and knowing how and why they make the predictions and the decisions that they do. We need to be able to see precisely what factors went into a decision to justify that they are fair and unbiased by giving the reasoning behind different outcomes. Gaining a better understanding of how the model works will give us better control of the model because we know exactly what the model can and cannot do, and thus we would be able to properly evaluate whether it is capable of handling the task at hand. Knowing the model’s weaknesses and limitations will not only let us have better control over the model but it also makes the task of improving the model easier. Lastly, if the model discovers solutions, patterns, or strategies previously unknown to humans we would like to be able to learn from the model and ideally make the same discoveries.

The terminology of XAI is sometimes used interchangeably, and do to some extent overlap. In short, in this thesis, the three most commonly used terms are defined as follows. Transparency relates to whether or not the relevant information is available to humans in an understandable form. Interpretability relates to how

easy it is for humans to understand how a model works, including predicting what it will do in different situations. Explainability relates to the model being able to explain the reason behind its decisions and behaviour. Interpretability can be seen as a passive trait of the model, whereas explainability relates to the active trait of being able to give explanations or reasoning.

If explainability is considered before building the model such that it is either self-explainable or self-explaining, it is called an *intrinsic* XAI method. In essence, the XAI part is embedded in the AI method. Self-explainable methods, such as linear regression at one end of the scale, usually have limited expressive power and thus struggle with achieving high accuracy and good performance in comparison with black boxes such as DNNs. A post-hoc method, on the other hand, is a method that is applied on top of, or in addition to, the model that needs to be explained. Among the post-hoc methods, there are three main types, namely *surrogate models*, *extrinsic methods*, and *introspective methods*. Surrogate models, or imitation models, are simpler models that attempt to approximate the complex black box model. By looking at how the surrogate model makes its predictions, we can learn something about how the black box model makes its predictions. The surrogate model can be local, global, or somewhere in between. A global surrogate model approximates the entire black box model. An example of such a method is presented in Section 2.4. A local approximation method generates a surrogate model around the instance to be explained. An example of such a method can be found in [51] where a linear function is built around the instance. Extrinsic methods treat the model to be explained strictly as a black box and only make use of the input and output. On the other hand, Introspective methods make use of the black box model’s structure. For example, with NN this can be making use of the activation functions, the weights, or the gradients [5, 64, 65, 69]. XAI methods can be either *model-agnostic* or *model-specific*. A model-agnostic method can be applied to any model. In contrast, a model-specific method can only be applied to one specific type of method such as DTs, NNs, or perhaps NNs with a specific structure. By definition, intrinsic methods are always model-specific as they only explain themselves.

In [67], they highlight that transparency is beneficial but not necessarily a universal good and should rather be treated as a means to an end instead of as a goal in itself. The goal should be robust and fair systems which we understand and thus can attribute the appropriate amount of trust. Transparency can be an important tool or characteristic towards that goal. In Table 3, some motivations for transparency based on [67] are given. The developer’s motivations for transparency which include the user introduces some interesting situations where giving *truthful* explanations might not be in the developer’s best interest. If the system is unfair or biased and the goal still is to make the user trust the sys-

Table 2: An overview of the different explanation types along with examples of methods that give such explanations.

Explanation type	Examples of methods that give this explanation type
Sensitivity explanations	LIME[51], SHAP[36], LMTs[17, 18], TreeExplainer[37], LRP[5]
Counterfactual explanations	LMTs[17, 18]
Prototypes	NN with embedded prototypes[32],ProtoX [47]
Concepts	TCAV[30]
Rule-based explanations	Anchors [52]

tem, the system either must be improved, or the user must be led to believe that the system is fair and unbiased. In addition to often having to make a trade-off between explainability and accuracy when building intrinsic methods, the possibility of "bad" intentions from the developer's side motivates the development of post-hoc, extrinsic methods since they can be applied to any model without knowledge or access to any more than the black box's inputs and outputs.

When we refer to the scope of the explanations, we refer to how much the explanations cover. On one end of the scale, we have *local* explanations that explain a single instance. In contrast, on the other end of the scale, we have *global* explanations that explain the black box model's overall behaviour in the entire state space. This is not a binary characteristic, as explanations can cover a group of instances, a certain region in the state space, or a certain part of the model's behaviour. Following, and in Table 2, a non-exhaustive list of different types of explanation types is presented.

Sensitivity explanations comes in many forms, such as *feature attributions* and *saliency maps*. In essence, these explanations tell us something about how sensitive the output of the model is to the different parts of the model's input. Sensitivity explanations in the form of feature attribution, more specifically, tell us how much each input feature contributed to the outcome. Think about the outcome of the model as the outcome of a volleyball match and the input of the model as all the players. The feature attributions then say something about how important each player was for the outcome of the match, good or bad. As the name indicates, saliency maps highlight which parts of an image were most important when a model made a prediction or classification. It can be seen as a representation of where the model paid the most attention to in the image.

Table 3: Motivations for transparency, based on [67]’s list of types and goals of transparency.

Developer	End-user	Other
To understand how the system works to improve or debug the system.	Provide a sense for what the system is doing and the system’s reasoning behind it, in addition to gaining trust towards the system	Giving society a basic understanding of the system, and becoming comfortable with the technology.
To make the user take the appropriate (or wanted) action	For the user to understand a particular prediction so that they can check for themselves that the system worked appropriately and possibly challenge this prediction	To facilitate safety guarantees and monitoring of edge cases
To make the user trust the system so that they continue using it		

Counterfactual explanations answers the hypothetical questions that are contrastive to how the situation actually is. Such questions typically go along the lines of *”but what if?”*. For a classification problem, the counterfactual explanation is defined as follows.

Definition 1 (Counterfactual explanation [22]). *Given a classifier b that outputs the decision $y = b(x)$ for an instance x , a counterfactual explanation consists of an instance x' such that the decision for b on x' is different from y , i.e., $b(x') \neq y$, and such that the difference between x and x' is minimal.*

For a regression problem, this must be modified to the output being significantly different in terms of some distance metric. Which metric to be used and how different the counterfactual output $b(x')$ depends on the problem at hand.

Prototypes gives explanations in a case-based reasoning manner. Prototypes are stereotypical examples of a certain type of input, and when used as an explanation, it lets us know which prototype the new instance is most similar to. In essence, the explanation takes the form of *”I predicted this because that’s what I*

did in this similar scenario". In [32], a way of building an intrinsically explainable NN by introducing a prototype layer to the structure is shown. In [47], a post-hoc method for giving prototypes as explanations for a RL-agent is presented. Each prototype constitutes a stereotypical state, taking the agent's behaviour and the environment into account.

Concepts are global explanations aiming at discovering human-understandable concepts that the black box has understood. An example of such a concept could be 'stripes' when deciding whether an image contains a zebra. These concepts can help us understand what the NN overall is basing its predictions on.

Rule-based explanations presents the explanations either as a set of rules or an ordered list of rules. These rules can either locally describe the rules used for one instance or globally describe the most important rules over all.

Previous work on XAI in DRL can be divided into four main categories[27], namely video games ([16, 29]), guidance and navigation tasks ([26]), system control ([31, 53, 70]), and robotic manipulation([15, 49, 50]). The use cases in robotics introduce some additional aspects and considerations to XAI. This includes (but is not limited to) the following:

- Many robotic applications are sequential decision-making problems. This means that it is not enough to analyze only one decision but rather that several consecutive decisions must be analyzed as a whole to understand the behaviour within a time frame.
- Robotic applications are governed by the laws of physics which must be taken into account to ensure meaningful explanations.
- Many of the problems are complex by nature, and thus, the explanations might also be complex. Therefore, care must be taken to communicate the explanations efficiently and understandably.

2.4 Linear model trees

DTs are one of the most commonly used ML techniques, both for regression and classification problems. A DT consists of a root node, branch nodes, and leaf nodes, as illustrated in Figure 6. The root node is the top node, it has no parent node and is where data instances enter the tree. Just like the root node, the branch node contains a splitting condition that decides if the data instance should follow the left or the right path coming out of the node. The leaf nodes have no descendants or splitting condition, but they do contain a prediction.

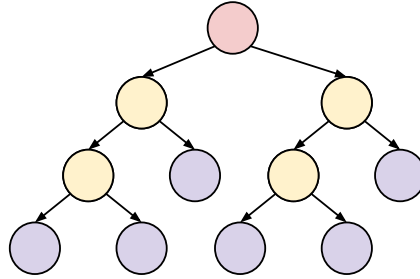


Figure 6: Illustration of a decision tree. The red node is the root node, the yellow nodes are branch nodes, and the purple nodes are leaf nodes.

This prediction is a class for classification trees, and for regression trees, this prediction is a constant number. The splitting conditions in the branch nodes usually take the form of checking whether or not a certain input feature is larger or smaller than a set threshold. By repetitively splitting the input space in two with different splitting conditions, the structure of the tree corresponds to a given partitioning of the input space, and each leaf node corresponds to one region. This is illustrated in Figure 7 for the case of regression trees. MTs is an umbrella term for all DTs with a non-constant prediction in the leaf nodes. The leaf nodes can contain all types of prediction functions, such as simple linear functions or even huge NNs. Trees with leaf nodes that contain a linear function are called LMTs. Both regression trees and LMTs constitute piece-wise linear functions, as can be seen in Figure 7 and Figure 8. However, by comparing Figure 7 and Figure 8 it is easy to see why LMTs can approximate functions much more accurately than regression trees even when the LMT is shallower than the regression tree.

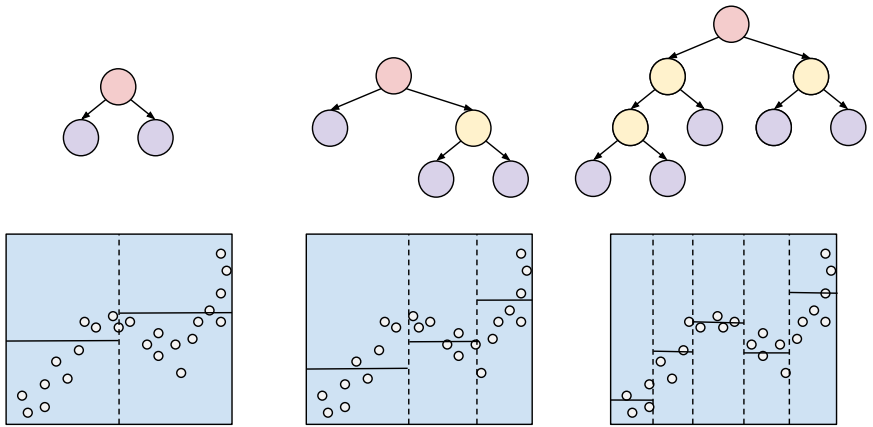


Figure 7: Illustration of how different regression trees divide the input space and assign a constant prediction to each region. The red node is the root node, the yellow nodes are branch nodes, and the purple nodes are leaf nodes.

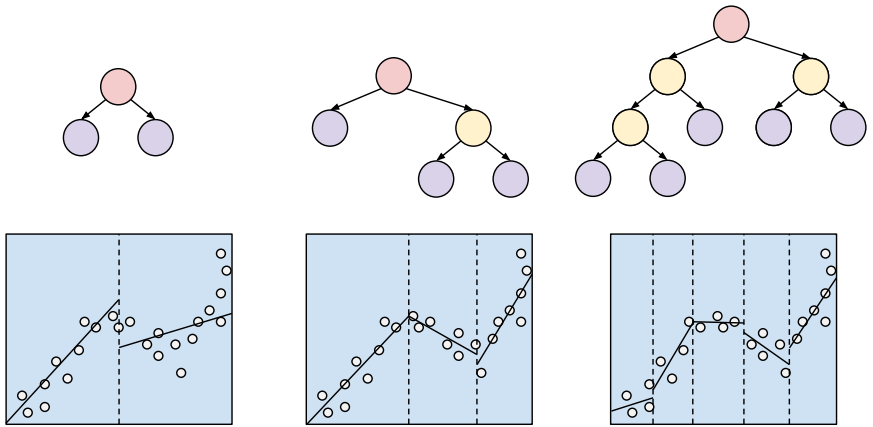


Figure 8: Illustration of how different linear model trees divide the input space and fit a linear function to each region. The red node is the root node, the yellow nodes are branch nodes, and the purple nodes are leaf nodes.

Generally, the tree's structure is determined by either all or some of the following parameters:

- **The maximum depth** states the maximum depth a leaf node in the tree can be at. Nodes at this level cannot have any children nodes and thus must be a leaf node.
- The tree stops growing when the **maximum number of leaf nodes** is achieved.
- **Minimum number of samples** states the minimum number of samples that must belong to each leaf node. If this parameter is too low, the tree might be prone to overfitting.

In [7], the authors show that LMTs can be formalized as a mixed integer optimization (MIO)-problem. A simplified version of the MIO-equations for a LMT with univariate splitting conditions in (16) is given below, for the full version, including measures taken to account for numerical instability and multivariate splitting conditions the reader is referred to [7].

Since the linear prediction function for the t 'th leaf node, y_t , can be described as

$$y_t(x_i) = \beta_t^T x_i + \beta_{0t}. \quad (3)$$

The overall prediction, f , for the LMT can be described as

$$f(x_i) = \sum_{t=1}^{\mathcal{T}_l} y_t(x_i) z_{it}, \quad (4)$$

where $z_{it} \in \{0, 1\}$ indicates whether or not the i 'th data instance x_i belongs to the t 'th leaf node, \mathcal{T}_l is the total number of leaf nodes, and n the size of \mathbf{x} . The loss function can then be formulated as

$$\begin{aligned} L &= \sum_{i=0}^n |Y_i - f(x_i)| \\ &= \sum_{i=0}^n (|Y_i - \sum_{t=0}^{\mathcal{T}_l} (\beta_t^T x_i + \beta_{0t}) z_{it}|), \end{aligned} \quad (5)$$

where Y_i is the correct output for the input x_i . Since we are minimizing over L we can linearize (5) to

$$\begin{aligned} L_i &\geq f_i - y_i, \forall i \in [n], \\ L_i &\geq -f_i + y_i, \forall i \in [n]. \end{aligned} \quad (6)$$

(3) can be linearized to

$$\begin{aligned} f_i - (\beta_t^T x_i + \beta_{0t}) &\geq -M_f(1 - z_{it}), \\ f_i - (\beta_t^T x_i + \beta_{0t}) &\leq M_f(1 - z_{it}). \end{aligned} \quad (7)$$

Here, M_f must be larger than the maximum value of $f_i - (\beta_t^T x_i + \beta_{0t})$ for any i . If z_{it} equals 1 f_i must equal $(\beta_t^T x_i + \beta_{0t})$ and if z_{it} is zero the constraint is inactive. To ensure that each data point is assigned to one, and one only, leaf node. As stated, z_{it} tells us if data point i belongs to the t 'th leaf node. Each leaf node must contain a minimum number of samples, N_{min} and if l_t is a binary variable stating whether or not leaf node t contains any sample the allocation of data points to leaf nodes can be described as

$$\sum_{i=1}^n z_{it} \geq N_{min} l_t, \forall t \in \mathcal{T}_L \quad (8)$$

and

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \forall i \in [n]. \quad (9)$$

To ensure that all leaf nodes are valid and contain at least the minimum amount of samples required, the following is introduced:

$$z_{it} \leq l_t, \forall t \in \mathcal{T}_L. \quad (10)$$

The next thing we need to include is the splitting conditions. The variable d_t assigns which branch nodes apply a splitting condition or not. This is just for the convenience of not having to remove or introduce new variables to the optimization problem if the tree should be shallower than the maximum depth. If d_t is zero, all data instances will be sent the same way. The splitting conditions used in LMTs are given by

$$a_t x_i < b_t, \forall t \in \mathcal{T}_B \quad (11)$$

and

$$\sum_{j=1}^p a_{jt} = d_t, \forall t \in \mathcal{T}_B \quad (12)$$

ensures that the splitting conditions are univariate since both $a_{jt} \in \{0, 1\} \forall j \in [p]$ and $t \in \mathcal{T}_B$ where (n, p) is the dimension of \mathbf{x} and \mathcal{T}_B is all the branch nodes. (11) can be linearized in the same fashion as (3) which gives us

$$\begin{aligned} a_m^T x_i &< b_m + M_1(1 - z_{it}), \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \\ a_m^T x_i &\geq b_m - M_2(1 - z_{it}), \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \end{aligned} \quad (13)$$

where M_1 and M_2 must be sufficiently large so that the constrain disappears when z_{it} equals zero. The set of right-hand ancestors of node t is denoted $\mathcal{R}(t)$ and the set of left-hand ancestors of node t is denoted $\mathcal{L}(t)$. Given that $x_i \in [0, 1]^p$, we have that

$$0 \leq b_t \leq d_t, \forall t \in \mathcal{T}_B. \quad (14)$$

A branch node cannot apply a splitting if its parent node does not. We enforce this by

$$d_t \leq d_{p(t)}, \forall t \in \mathcal{T}_B \setminus \{1\}. \quad (15)$$

Combining the equations above allows us to express the problem of finding the optimal LMT with maximum depth d in the following way:

$$\begin{aligned} \min \quad & \sum_{i=0}^n L_i & (16) \\ \text{s.t.} \quad & L_i \geq f_i - y_i & \forall i \in [n], \\ & L_i \geq -f_i + y_i & \forall i \in [n], \\ & f_i - (\beta_i^T x_i + \beta_{0i}) \geq -M_f(1 - z_{it}), & \forall t \in \mathcal{T}_L, \\ & f_i - (\beta_i^T x_i + \beta_{0i}) \leq M_f(1 - z_{it}), & \forall t \in \mathcal{T}_L, \\ & \sum_{i=1}^n z_{it} \geq N_{min} l_t, & \forall t \in \mathcal{T}_L, \\ & \sum_{t \in \mathcal{T}_L} z_{it} = 1, & \forall i \in [n], \\ & z_{it} \leq l_t, & \forall t \in \mathcal{T}_L, \\ & a_m^T x_i < b_m + M_1(1 - z_{it}), & \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \\ & a_m^T x_i \geq b_m - M_2(1 - z_{it}), & \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \\ & 0 \leq b_t \leq d_t, & \forall t \in \mathcal{T}_B, \\ & \sum_{j=1}^p a_{jt} = d_t, & \forall t \in \mathcal{T}_B, \\ & d_t \leq d_{p(t)}, & \forall t \in \mathcal{T}_B \setminus \{1\}, \end{aligned}$$

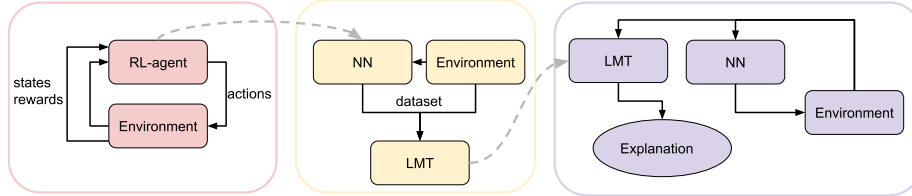


Figure 9: Overview of the pipeline of using the linear model tree as an explainer for a reinforcement learning agent. The training of a reinforcement learning agent is shown in the red field, building the LMT is shown in the yellow field, and how to combine them during run-time is shown in the purple field.

DTs are often considered completely transparent or interpretable because of their intuitive structure. Many DTs are hard for humans to understand. Incomprehensible trees can be built by using features that are hard to understand, unintuitive splitting conditions, complex prediction functions, or by simply letting the tree grow so deep that we lose track of all the different paths from the root node to the different leaf nodes. Therefore, it is better to consider DTs as possibly interpretable because they *can* be interpretable if interpretability is taken into account when building them. Of course, an intrinsically explainable method is preferred over a black box with a post-hoc explanation method. Still, more often than not, we must accept that there is a trade-off between interpretability and accuracy. In Table 4, an overview of the different characteristics of LMTs that affect the trees' interpretability is given.

Additionally, DTs must be trained in a supervised manner, which makes them challenging to use for problems better solved by RL. Since it is easier to train a NN with adequate accuracy, LMTs can instead be used as a post-hoc explanation method. In Figure 9, how an LMT can be used as a post-hoc explanation method for a RL-agent. First, a black box model is trained by an RL-agent. For this case, we assume that it is a NN. By sampling states from the environment and passing them through the NN a dataset that can be used to build an LMT is formed. Given that the LMT managed to approximate the NN properly, the LMT can run in parallel with the NN. Thus, the NN will still control the system, and the LMT will provide explanations.

Table 4: Characteristics of LMTs affecting interpretability

Characteristic	Positive impact	Negative impact
Depth of tree	Shallow trees	Deep trees
Splitting condition	Simple, univariate, linear	Complex, multivariate, non-linear
Prediction function	Simple, univariate, linear	Complex, multivariate, non-linear
Features	Intuitive, real-world based	Complex, over-engineered

[htb]

2.5 Test applications

The concepts presented in this thesis were tested on the two robotic applications presented in this chapter, namely the docking problem presented in Section 2.5.1 and the pendulum presented in Section 2.5.2.

2.5.1 The docking problem

The docking problem was originally presented in [54]. Docking is the process of taking a vessel to the desired berthing point along the quay and subsequently holding that position. Docking is a critical skill that must be mastered in order to achieve autonomous ships. This process is considered to be both a challenging and a high-risk process due to nonlinear system behaviour and reduced manoeuvrability.

In [54], a DRL agent was trained to perform docking in a simulated environment based on Trondheim harbour. An example of a run in the simulated environment is given in Figure 10. The simulated vessel had 9 input features concerning the vessel’s position and velocity and 5 actions that controlled 3 thrusters. The DRL agent’s policy takes the form of a DNN with two hidden layers with 400 neurons each. The agent was trained with the proximal policy optimization (PPO) algorithm [57]. It is this agent’s DNN which is used as the black box to be explained in all the publications of this thesis.

For further details regarding the agent or the environment, the reader is referred to [54].

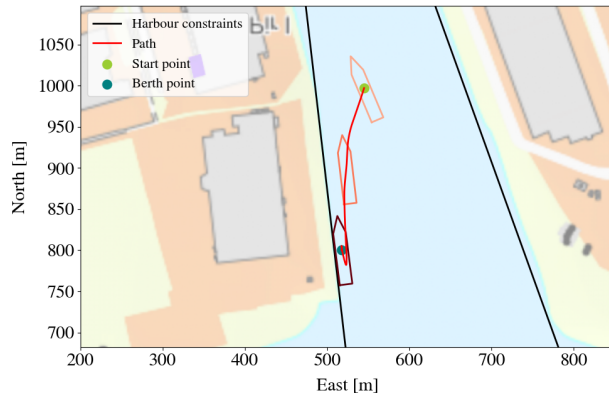


Figure 10: Example of a run in the simulated docking environment.



Figure 11: Illustration of the inverted pendulum from Paper E [20].

2.5.2 The inverted pendulum problem

The second robotic application used in this thesis is the inverted pendulum problem⁹. The goal is to balance the pendulum in the upright position by applying force to the free end of the pendulum. An illustration of the inverted pendulum is given Figure 11.

⁹https://www.gymnasium.com/environments/classic_control/pendulum/

3 Contributions and discussion

In this chapter, a more thorough discussion regarding the three main contributions: first, proposing tactics and highlighting important considerations for developing LMTs in Section 3.1; second, evaluating the usefulness of LMTs as an XAI approach for robotics in Section 3.2; and third, taking into account the end-user’s perspective and suggesting customized visualizations of feature attributions in Section 3.3.

3.1 Building LMTs

In Paper A ([17]), a heuristic algorithm for building LMTs was presented. A demonstration showing that the LMTs are capable of approximating the DNN controlling a vessel performing docking in a simulated harbour environment well enough to act as a post-hoc explanation method is given. In Paper C ([18]), an improved version of the algorithm presented in Paper A ([17]) was presented. In Paper D ([19]), four different algorithms for building MTs were investigated.

The algorithms presented in both Paper A ([17]) and In Paper C ([18]), are heuristic methods that follow a top-down approach, meaning that they start from the root node and the tree then grows deeper as opposed to taking the entire tree into consideration as can be done by taking a MIO-approach. As noted by [7], the number of binary variables in the MIO-formulation shown in (16) quickly increases as the number of data points increases. Given a data set with 1000 data points and a tree of depth 5, there are 320,000 binary variables needed to keep track of which leaf node each data point belongs to. For this reason, the MIO approach requires a lot of computational resources for applications that require larger datasets, and we, therefore, chose a heuristic approach instead.

LMTs and quadratic MTs have been used to replace an RL agent and thus serve as an intrinsic explainable method by calculating feature attributions from the prediction functions in the leaf nodes [12]. The MTs were not capable of achieving the same accuracy as the RL agent and this replacement thus came with a performance cost. The LMT implementation from Paper A ([17]) was used in [34], where three regression tree methods were compared. The implementations from [34] were used as a basis for the implementations in Paper D ([19]). To the author’s best knowledge, in Paper A ([17]), LMTs were used as a surrogate model to generate explanations in the form of feature attributions for the first time in the literature.

The CART-like implementation of LMTs is an extension of the implementation

of LMTs from [68]. Instead of using the maximum depth of the tree, we used the maximum number of leaf nodes to include more asymmetric trees in the search space. Additionally, we introduced some randomization to both finding the next node to be split and to the search for the threshold in the splitting conditions so that the trees can be built in parallel with the same parameters and still give different results.

To allow the algorithm to explore more of the solution space, the maximum number of leaf nodes was chosen as a hyperparameter instead of using the maximum depth of the tree. For binary trees (meaning trees that splits the nodes into two children nodes), the maximum depth indirectly sets the maximum number of leaf nodes since a complete tree with depth D has a total number of 2^D leaf nodes. On the other hand, if 2^D is used as the maximum number of leaf nodes, the tree can grow deeper than depth D in regions that require that and shallow for the more linear regions.

A critical drawback with heuristic methods is that we have no guarantee for the optimality of the resulting tree. Even though using a prebuilt LMT is straightforward, building them can be more tedious. Firstly, there are quite a few hyperparameters to be defined. These hyperparameters include the maximum number of leaf nodes, the minimum number of samples required for a leaf node to be valid, and the required improvement in the loss metric for a branch node to be split. It is important to note that we are searching for the best tree given these hyperparameters and not the best tree in general. Take the MIO-formulation of LMTs where we are searching for the optimal tree with a given depth instead of searching for the optimal tree with any depth. For example, if we are searching for the optimal tree with a depth of 4 but the globally optimal tree has a depth of 6, we will never find the globally optimal tree. The same goes for any of the hyperparameters that we need to set. Secondly, the LMT cannot approximate the NN for situations that are not well-represented by the dataset. Thus, great care must be taken when gathering data from the environment and the agent. One measure, introduced in Paper A ([17]), taken to ensure good datasets to build the LMTs upon was the strategy referred to as *iterative tree building*. First, data was gathered evenly distributed from the environment and paired with the output of the NN and an LMT was built upon that dataset. Then, the outputs of the NN and the LMT in this dataset was compared to see where in the state space more data collection was required, followed by building a new LMT based upon this improved dataset. This process is then repeated until the LMT approximates the black-box accurately enough. It is hard to set a hard threshold for what accurate enough should be, as there is no good metric to determine how similar two different models are. Instead, this should be investigated through several different metrics and using these metrics as indicators for when the LMT accurately

enough approximates the black box.

Paper C ([18]) introduced domain knowledge by stating which features could be used in the splitting conditions in the branch nodes at the different depths of the tree, which is called *ordered feature splitting*. Not only does this significantly speed up the algorithm since the number of features that must be searched through is lower, but the algorithm also finds more promising trees more consistently. This ordering should be done with the application at hand in mind as it imposes some rather strict limitations to the structure of the LMT. Ordered feature splitting can have a positive impact, given that it removes the parts of the solution space that includes bad performing trees and keep the part of the solution space that includes the optimal and near-optimal solutions.

Within the field of computer science, a lot of work has been done on different algorithms for building DTs, such as the pruning of trees to prevent overfitting or other methods for simplifying the trees [10]. Lessons from previous work on DTs should be utilized to a higher degree to improve the algorithm for building LMTs as well.

As stated earlier, LMTs' structural properties are both beneficial in terms of being intuitive for humans so that we can easily follow an instance's path from the root node to its respective leaf node and because it is relatively easy to extract explanations from the trees. Even though it is always possible to follow the path, the number and length of the paths will vary with the size of the tree, which is directly linked to how easy it is to get an overview of the tree. Therefore, the size of the tree is also directly linked to how easily humans can understand it.

There are many ways of building not only DTs in general but also LMTs specifically. As shown in Paper B ([35]), LMTs structure makes them well-suited to be formalized as a MIO-problem but solving them for real-world problems is difficult and we thus must resort to heuristic and greedy methods. Paper D ([19]) compared three different methods for building LMTs and one method for building non-linear MTs. The work is limited to only containing methods that could handle multiple, continuous inputs and outputs and used non-constant prediction functions within the leaf node of the trees. With some adjustments, other methods could apply to these kinds of robotic problems but only methods that were ready to go were considered. Two of these methods, optimal regression tree with linear prediction function (ORT-L) and ORT-L with hypersplits (ORT-LH), are commercially used methods offered by *Intepretable AI*¹⁰. At the time, ORT-LH had issues with numerical instability during run-time and a structured search for the best hyperparameters was not possible which most likely resulted in subop-

¹⁰<https://www.interpretable.ai/>

timal trees.

3.2 LMTs as an XAI-method

DTs is a well-known SL method, with CART dating back to 1984 [9]. Most DT-algorithms do not apply to RL-problems as they must be built from scratch to introduce new data. The differentiable decision trees (DDTs) from [60] can be updated online as the splitting conditions in the branch nodes are sigmoid functions but the sigmoid functions are discretized after training to ensure discrete splits. Sigmoid functions are differentiable and backpropagation can therefore be applied to the trees. DDTs as presented in [60] is only applicable to applications with a discrete action space. In this thesis, the LMTs is used as a post-hoc, surrogate XAI method trained as a SL problem. As emphasized in Section 3.1, it is crucial to have a dataset that properly represents the agent’s behaviour across the entire state space to build an LMT that accurately represents the black box model. For a surrogate model to be able to run in parallel with the black box and give explanations in real time for robotic applications, the surrogate model must be fast enough to keep up with the black box. The fact that LMTs are capable of this is a significant motivation for using them as an XAI method.

LMTs can be seen as a global explanation as the black box model’s complex structure is ported to the more easily understood structure of LMTs. However, as discussed in Paper D ([19]) and Paper C ([18]), a large DT is not necessarily interpretable for humans without the use of additional analytical tools as it simply is too much information to be able to get a good overview over the model. That the LMTs’ structure is easy to understand, meaning it is easy to follow all the decisions made from the root node along the path to the leaf node and finally, the linear prediction function must not be confused with that the LMT as a whole is easy to get an overview of.

In Paper B ([35]), three post-hoc, model-agnostic methods generating explanations in the form of feature attributions were implemented and evaluated for a marine robotic application. The methods that were tested out and evaluated were linear model agnostic explanations (LIME) [51], Kernel-SHAP [36], and the first iteration of LMTs from Paper A ([17]). Since both LIME and SHAP adapt a local surrogate model around the instance to be explained, they can easily adapt to new data, whereas LMTs must have all data available at the time it is built. LIME gave somewhat noisy explanations in addition to being too slow to run in real-time. Due to the LMT being only piece-wise linear, it can have quite big and abrupt changes in its predictions and, thus also in its explanations in areas close to the borders of the leaf nodes’ regions. SHAP gave smoother explanations, in

terms of not changing too suddenly in comparison to both LIME and the LMT, but only the LMT was able to give explanations in real time. Since linear regression is vulnerable to dependant features, so are the feature attributions generated based upon these linear functions. Suppose even just for a region two (or more) features are dependent. In that case, the linear function may only need to use one of these features to get an accurate prediction and the features not used will thus be given no importance by the feature attributions. SHAP also assumes independent features, which is pointed out and addressed in Causal-SHAP [13].

In addition to porting the complex structure of the black box to a more simplistic form, LMTs are capable of generating explanations in the form of feature attributions as seen in Paper A ([17]), Paper B ([35]), Paper C ([18]), Paper D ([19]) and counterfactual explanations (CFEs) as seen in Paper E ([20]). In Paper E ([20]), it is shown that CFEs can be generated from the LMT without any adjustments to either the structure of the tree or the building process of the tree. The LMT is only used to *locate* the counterfactual state and then this counterfactual state is passed through the black box to find the counterfactual action. The counterfactual state and action together make out the counterfactual explanation. Since the black box is used to determine the counterfactual action, the counterfactual explanation is always true, but we cannot be sure that it is the *best* explanation. Best, in this case, being the counterfactual example that has maximized the objective function used to find the counterfactual example with the smallest change to the state while still giving the largest change in the action. Neither feature attributions nor CFEs constitute a complete local or global explanation but can still give insightful information about the black box’s decision-making process.

3.3 Visualization of explanations in robotics

As stated in [42], an explanation is the transfer of information. The *explainer* has some causal information regarding a decision made by themselves or someone (or *something*) else that they want to explain to the *explainee*. There could be many reasons for wanting this transfer of information, such as the explainee wanting to understand something (think of the explainee as a student and the explainer as a teacher) or that the explainee wants to check that the explainer has understood something (think of the explainee as the teacher wanting to check what the student, the explainer, has learned). In the context of XAI, the explainer would be the XAI method explaining itself (if it is an intrinsic method) or a black box model to be explained to the human user of the system by a post-hoc XAI method.

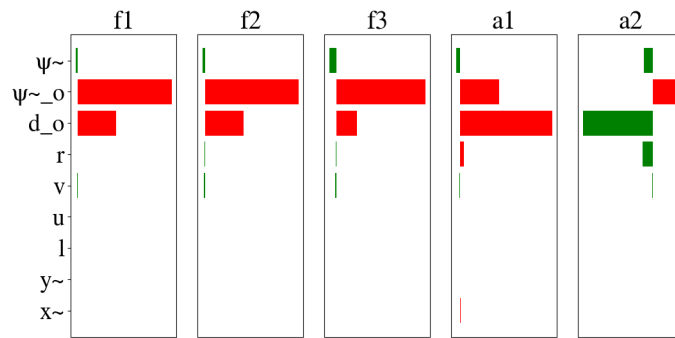
In Paper A ([17]), the feature attributions were presented in five bar plots, one

for each action, which can be seen in Figure 12a. If an input feature contributed positively (pushed the action towards a higher value) the bar plot is green, and red if it contributed negatively. The bar plots are more efficient than just looking at the raw numbers. However, there are still connections that must be made, such as considering that the state of the azimuth thrusters is collectively determined by f_1 and a_1 , and f_2 and a_2 . The colours should be changed to be more colour-blind friendly. Even though the explanations are given in real-time, it is challenging for the end-user to process them in real time.

In Paper C ([18]), the focus lies on how to best communicate the explanations depending on who the explaine, which is referred to as the end-user, is. Many different aspects and characteristics of the different end-users can affect how the explanations should be communicated. Two central questions to ask when considering the end-user is 1) "What questions are the end-user asking?", and 2) "What is the end-user going to use the explanations for?". To answer the first question, we must know what the end-user knows beforehand, what their expectations for the system are, and why they are asking questions. To answer the second question, the end-user's role (such as captain, developer, or passenger of an autonomous surface vessel (ASV)) must be considered. In Paper C ([18]), two end-users were considered, namely the developer of the black-box model and the captain of the ship the black-box model is used upon. There are many differences between these two end-users, ranging from different background knowledge to different personal risks associated with failures of the system. Since the developer does not need to process the explanations in real time, all the information regarding an episode was presented through plots of the episodes, states, and feature attributions. The captain, on the other hand, must process the information in real-time and irrelevant information (such as too detailed information) is filtered out. One of the developer's plots can be seen in Figure 12d, and the visualization of the state, action, and explanation given to the captain is shown in Figure 12c. The end user and the industry they are intended for should evaluate the explanations and how they are visualised. Additionally, several end-users, such as passengers of the ASV should be included.

In Paper E ([20]), CFEs were generated both for the docking problem and the inverted pendulum problem. For the docking problem, the difference between the factual state and action and the counterfactual state and action was given in a tabular form next to the visualizations as presented in Paper C ([18]). This is not an ideal way of presenting the CFEs, and work concerning how these explanations best can be presented for this application should be done. For the inverted pendulum case, however, the counterfactual state and counterfactual action could be presented in the same way as for the system, only in another colour. This can be seen in Figure 12b, where the pendulum's state and the

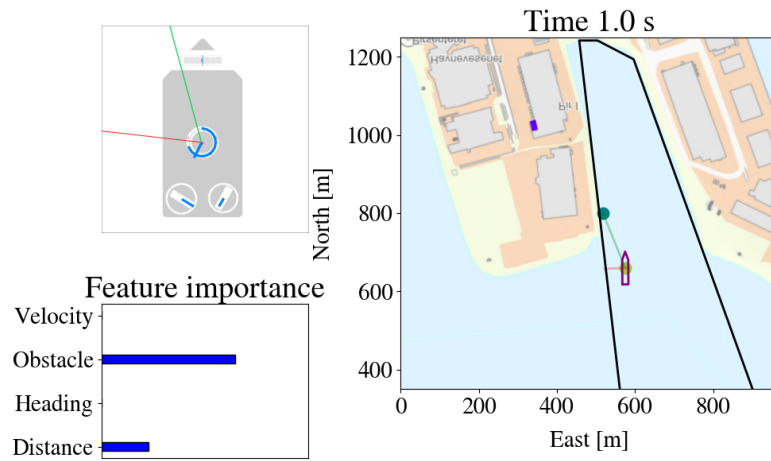
agent's action are shown in red, and the counterfactual state and action are shown in black.



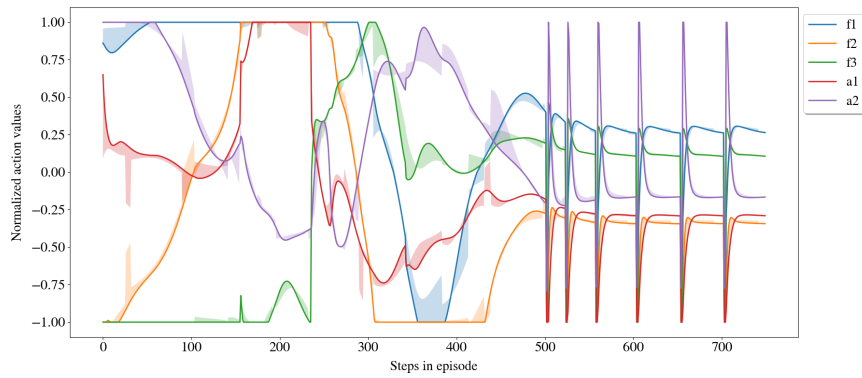
(a) Bar plot of the feature attributions from Paper A ([17]).



(b) Visualization of counterfactual explanations from Paper E ([20]).



(c) Visualizations intended for the captain of the system and the feature attributions from Paper C ([18]).



(d) One of the visualizations of the state, actions, and feature attributions intended for the developer from Paper C ([18]).

Figure 12: Examples of different visualizations of systems and explanations from Paper A ([17]), Paper C ([18]), and Paper E ([20]).

4 Conclusions and further work

This thesis contains contributions with novel solutions to the problem of explaining AI based robotic systems in real-time, adapting the visualization and communication of the explanation to two different users of the AI based system, and how LMTs can be built and used as a post-hoc, surrogate XAI method for robotic applications.

There is a clear need for XAI within robotic applications that utilize black box models such as DNNs. This thesis, and the publications it contains, has shown that LMTs have several traits that make them a useful XAI tool, such as

- being real-time runnable,
- being able to deal with multiple, continuous inputs and outputs,
- being a global surrogate model with an intuitive structure,
- and being able to give both feature attributions and counterfactual explanations.

Even though LMTs can be formalized as a MIO-problem, solving them optimally is still very challenging for large-scale problems. Within the field of computer science, a lot of work has been done on how to build DTs, and further work should utilize different methods from this field, such as pruning the trees. It is shown that LMTs can give two forms of explanations: feature attributions and CFEs. The feature attributions are generated based on the linear functions in the leaf nodes of the LMT, while the CFEs are generated by searching through the closest leaf nodes until a CFE is found. Additionally, the LMTs are by themselves a form of explanation as they transform the black box into a simpler structure. However, very large DT are arguably not transparent as it will be very challenging to get a good overview of the trees. Both the feature attributions and the CFEs are local explanations. Thus, working on generating more global explanations from the LMTs would be an interesting direction for further work.

The trees benefit from having domain knowledge introduced to the structure of the tree under the building. Having multiple actions for each tree leads to a simpler structure that is more easily understood by humans and more consistently accurate trees. Measuring the similarity between the black box model and the LMT is not straightforward. Both the error on a test set and the behaviour of the two models when controlling the robotic system were compared to get some indications of how similar they were. Further work should include finding more reliable similarity measures and preferably some benchmark metric.

How the explanations are communicated can affect the receivers' understanding of the black box model, the XAI method and the system. Different people want different explanations, so it is crucial to consider the receiver when presenting the explanations given by the XAI method. Two significantly different ways of visualizing the explanations to two different users of the black box model is given, and further work should include having such visualizations evaluated by their respective receivers.


The field of robotics introduces some additional requirements and challenges to the XAI problem. This includes, but is not limited to, explaining sequential decisions, making sure not to give explanations that do not violate the laws of physics, and giving the explanations in real-time.

5 Publications

This chapter contains reprints of the publications which were written as a result of the work in this thesis. The publications are formatted to fit the format of the thesis.

5.1 Paper A

Postprint of [17]: **Vilde B. Gjørsum**, Ella L. H. Rørvik and Anastasios M. Lekkas. ‘Approximating a deep reinforcement learning docking agent using linear model trees’. In: The 19th *European Control Conference(ECC)(2021)*, pp. 1465-1471, doi: <https://doi.org/10.23919/ECC54610.2021.9655007>

©2021 Vilde B. Gjørsum, Ella L. H. Rørvik, and Anastasios M. Lekkas. Reprinted and formatted to fit the thesis under the terms of the Creative Commons Attribution License 

Approximating a deep reinforcement learning docking agent using linear model trees

Vilde B. Gjørsum¹ and Ella-Lovise H. Rørvik² and Anastasios M. Lekkas¹

Abstract

Deep reinforcement learning has led to numerous notable results in robotics. However, deep neural networks (DNNs) are unintuitive, which makes it difficult to understand their predictions and strongly limits their potential for real-world applications due to economic, safety, and assurance reasons. To remedy this problem, a number of explainable AI methods have been presented, such as SHAP and LIME, but these can be either too costly to be used in real-time robotic applications or provide only local explanations. In this paper, the main contribution is the use of a linear model tree (LMT) to approximate a DNN policy, originally trained via proximal policy optimization (PPO), for an autonomous surface vehicle with five control inputs performing a docking operation. The two main benefits of the proposed approach are: a) LMTs are transparent which makes it possible to associate directly the outputs (control actions, in our case) with specific values of the input features, b) LMTs are computationally efficient and can provide information in real-time. In our simulations, the opaque DNN policy controls the vehicle and the LMT runs in parallel to provide explanations in the form of feature attributions. Our results indicate that LMTs can be a useful component within digital assurance frameworks for autonomous ships.

Index Terms

Deep Reinforcement Learning, Explainable Artificial Intelligence, Linear Model Trees, Docking, Berthing, Autonomous Surface Vessel

I. INTRODUCTION

Deep reinforcement learning (DRL) is a powerful tool with many application areas within robotics, such as perception and control. One of DRL's attributes is that it enables end-to-end learning, which refers to mapping sensory input directly to control actions. This mapping allows for optimizing the overall system performance, instead of having several, locally optimized systems in cascade, which often is the case for model-based systems. In [1], the learned policy was able to perform various manipulation tasks with a dexterous, robotic hand. In [2], a real-world Minitaur robot learned to walk on a flat surface and was able to handle somewhat challenging surfaces and obstacles without having seen these obstacles during training. In [3], one of DRL's greatest

This work was supported by the Research Council of Norway through the EXAIGON project, project number 304843. An additional thanks to Nicolas B. Carbone for his contribution through several valuable discussions.

¹Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway. Email: vilde.gjarum, anastasios.lekkas}@ntnu.no

²Department of Artificial Intelligence, TrønderEnergi, Trondheim, Norway. Email: elh.rorvik@gmail.com

strengths is demonstrated, namely discovering strategies through the exploration of the state- and action space in the multi-agent environment of playing hide and seek. The agents adapted and came up with new strategies to combat the opponents' latest strategy, even going as far as using parts of the environment in ways not originally intended. The applicability of DRL has also been demonstrated in motion control tasks for autonomous surface vessels (ASVs), which often operate in complex and uncertain environments that are challenging to model. In [4], DRL was used to perform curved-path following on a surface vessel and performed well compared to line-of-sight guidance in simulations. In [5], a DRL-agent was trained to perform both path-following and collision avoidance. In [6], a DRL agent is trained to perform the approach and berthing phases of docking of an ASV.

Even though DRL is a promising tool for advancing the level of autonomy in many fields, its potential applications in real life are strongly limited by the lack of transparency of the deep neural networks (DNNs) involved. This is crucial in all cost- and safety- critical applications. To be able to understand the agent's actions, a *global explainer* is needed, or as a bare minimum, *local explanations* for each prediction. There has been done a lot of work on addressing this problem in the field of eXplainable Artificial Intelligence (XAI) in the recent years. The goal of XAI is to uncover the inner workings of black-box models. One of the most prominent explainers is the Local Interpretable Model-agnostic Explainer (LIME) [7], which trains an interpretable, surrogate model around the instance it is explaining based on neighboring instances. LIME is a post-hoc (it explains previously trained methods), model-agnostic (it can explain any type of model) XAI method. The neighboring datapoints are weighted based on how far away from the instance to be explained they are. One weakness of LIME is that it does not perform as well when explaining instances it has not seen before. This problem is addressed in [8] by the same authors, where the interpretable surrogate model is replaced by a set of IF-THEN rules called *anchors*. Another prominent XAI method is Shapley Additive exPlanations (SHAP) from [9]. The SHAP method explains a prediction by assigning importance to the input features for that prediction. The importance of the features is calculated by utilizing *Shapley values* from game theory, in combination with the coefficients of a local linear regression. SHAP is a model-agnostic, post-hoc method. The assigned contributions of the input features should add up to the original prediction, thus SHAP is an *additive feature attribution* method. Also, although SHAP is mainly a local explanation method, it can give indications of how the black-box model works as a whole through calculating the Shapley values for every instance and analyzing the resulting matrix of Shapley values. It should be noted that SHAP is a very computationally demanding method. Both LIME and SHAP form their explanations by perturbing the inputs and computing how these perturbations affect the output of model to be explained. In [10], it was shown that XAI methods relying on input perturbations are vulnerable to adversarial attacks aiming to hide their classification bias from the XAI method. One of

the reasons such methods are vulnerable to adversarial attacks is that the data sampled from input perturbation often are irrelevant, and the model is forced to explain input samples it has never seen before [11]. Even if the model to be explained does not intend to fool the explainer, if the samples created by perturbing the inputs are unrealistic, the explanations will be based upon predictions not fairly representing the model. Additionally, in [11] it is pointed out that SHAP assumes complete independence between the input features, which is often not the case for robotic systems. In [12], the method called Integrated Gradients was presented. As the name implies, the gradients are integrated along a straight-line path between the instance to be explained and an information-less baseline instance to extract the explanations directly from the neural network. Integrated Gradients is a post-hoc, model-specific (it can only explain one type of model). In this paper, the focus is on linear model trees, a type of decision tree (DT). The rule-based nature of DTs make them inherently transparent and interpretable, since it is trivial to follow the path from the leaf node (output) to the root node (input) of the tree. The most basic form of a decision tree for continuous data - a simple regression tree - has univariate splits and each leaf node predicts a constant value. Model trees are regression trees with a different type of prediction model at the leaf nodes. In linear model trees (LMTs), linear regression is used in the leaf nodes instead, which makes it easy to extract explanations of the predictions in the form of feature attributions, in addition to being transparent. Linear model trees, as presented in this paper, are fast enough to be used in real-time, are model-agnostic, and can be used to understand both individual predictions and the system as a whole. To the authors' best knowledge, there is no existing literature where LMTs are used to approximate DNN policies controlling robotic systems. The main contributions of this paper are:

- We use an LMT to approximate a DRL policy, previously trained in [6] via proximal policy optimization (PPO), to perform autonomous docking in a simulated environment.
- Compared to the standard way of building LMTs, we have added randomization to the search for thresholds and the process of choosing which node to split next. Moreover, to ensure a sufficient dataset from the areas of interest, an iterative approach to the training and data collection was used.
- We run the LMT in parallel with the policy in order to provide real-time correlation between input features and the selected control inputs computed by the policy.

II. DOCKING AS A DEEP REINFORCEMENT LEARNING PROBLEM

In this section, the docking problem and the reinforcement learning docking agent are briefly introduced. For further details about the implementation and training, the reader is referred to [6].

A. The docking problem

Docking pertains to reaching a fixed location along a quay, where the vessel can moor, and can be split in three stages: 1) Moving from open seas to confined waters (*the approach phase*), 2) parking the vessel (*the berthing phase*), and 3) fastening the vessel to the dock (*the mooring phase*). Docking is a complex motion control scenario and requires a lot of intricate maneuvering, since the vessel operates close to the harbor infrastructure with little to no space for deviations, and under the influence of external disturbances that gain increased importance at lower speeds. Such circumstances are challenging for most traditional control systems since they often depend on accurate mathematical models in contrast to RL-methods that can learn the model guided by the reward function. In [6], a PPO policy was trained in a simulated environment based on the Trondheim harbor environment.

B. The docking agent

Deep RL is a subfield of machine learning where learning occurs by selecting actions via an exploration/exploitation scheme and receiving reinforcement signals for these actions. The reinforcement signals, called rewards, are given by the reward function, which is user-defined. The agent is tasked to find the state-action mapping (i.e. the policy) that optimizes the *return*, which represents the expected cumulative reward during an episode. Thus the reward function is crucial for the agent's learning process and its resulting behavior. The policy used in this work, was trained extensively in [6] with the PPO method from [13]. It performs the approach and berthing phases of the docking process from up to 400 meters distance from the quay. The PPO method uses a trust region to prevent the agent from overreacting to a training batch and thus risking getting stuck in a local minimum. A trust region is defined as the region where the policy approximation used for gradient descent is adequately accurate. Instead of having the trust-region as a hard constraint, PPO includes it in the objective as a penalty for leaving the trust region, which makes the training less rigorous. The policy is trained to perform the approach- and berthing phase of the docking. The training algorithm has no prior knowledge of the inner dynamics of the vessel and it utilizes the feedback from the reward function as the agent takes action and the outcomes of these actions are evaluated. Selecting the input features vector is crucial for the reward function and the agents learning, and thorough work was done in [6] to develop an effective reward function for the task in hand. The fully-actuated vessel to be controlled has two azimuth thrusters and one tunnel thruster, hence giving the following control states:

$$\mathbf{A} = [f_1, f_2, f_3, \alpha_1, \alpha_2], \quad (1)$$

where f_1, f_2 (from -70 to 100 kN) and α_1, α_2 (from -90 to 90 degrees) are the forces and rotation angles of the two azimuth thrusters, whereas the tunnel thruster can exert only a lateral force f_3 (from -50 to 50 kN). The thrusters' placement on the vessel

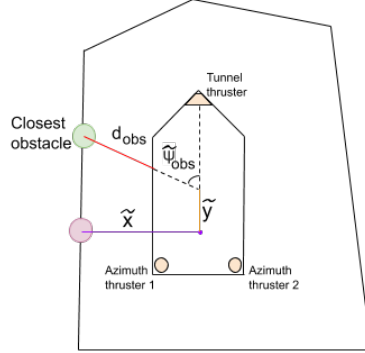


Fig. 1: The features, or states, representing the vessels position relative to the closest obstacle and the positioning of the vessels three thrusters.

can be seen in Figure 1. The state vector, which is also the input feature is composed of 9 states:

$$\mathbf{x} = [\tilde{x}, \tilde{y}, \tilde{\psi}, u, v, r, l, d_o, \tilde{\psi}_o], \quad (2)$$

where \tilde{x} and \tilde{y} represent the distance to the berthing point in the body frame, while $\tilde{\psi}$ represents the difference in the heading compared to the desired heading. The vessel velocities are given by the variables u , v , and r . The binary variable l indicates whether or not the vessel has made contact (crashed) with an obstacle. The relative position to the closest obstacle in body frame is given by d_o and $\tilde{\psi}_o$.

The PPO-trained policy network involves two hidden layers, consisting of 400 neurons each. The *ReLU* activation function was used for the hidden layers, and the hyperbolic tangent was used as the activation function for the output layer, ensuring actions in the range $[-1,1]$. The PPO-trained policy converged after approximately 6 million interactions with the environment. The DRL agent was trained in [6] to perform both the approach and berthing phase of docking, but without consideration for any speed regulations within the harbor.

III. APPROXIMATION VIA LINEAR MODEL TREES

A decision tree is a rule-based prediction method, which splits the input space into smaller regions and makes a prediction for each region [14]. A tree consists of branch nodes, where the splitting happens, and leaf nodes, where the prediction happens. LMTs perform linear regression in all the regions separately instead of attempting to fit the function for the entire feature space at once. To preserve the transparency of the tree in this work, the splits are univariate. Increasing the complexity of the prediction- or splitting model significantly increases the computational time required to build a tree, in addition to limiting their transparency. More generally, given any

Algorithm 1: Building LMTs

Require:*training data \mathcal{D}* *Maximum number of leaf nodes N* *Minimum number of data samples for leaf nodes M*

```
while number of leaf nodes is less than  $N$  do
  if there exist a node that fulfills all splitting criteria then
    Choose node to split
    Perform splitting
    Calculate best potential split for the newly created nodes
  else
    return root node
  end
end
```

black-box model $f: x \rightarrow y$, LMTs make out a piece-wise linear approximation function $f': x \rightarrow y'$, where $y \simeq y'$. LMTs are useful because they are easy to implement and can give explanations in real-time, which is crucial for most robotic applications. The implementation of the LMTs in this paper is based on [15] which again is based on Classification and Regression Trees (CART) from [16]. The following modifications have been made to [15]'s implementation:

- We added randomization in the process of searching for thresholds and choosing the next node to split.
- We replaced the maximum depth of the tree with maximum number of leaf nodes.

The tree building process as implemented in this paper is shown in Algorithm 1.

For a node to be split, and two new nodes to be created, there must exist at least one node which fulfills the splitting criteria. That is, a node where a split will result in two nodes with at least M data samples each, and gives an improvement in the model's loss. When a splitting has occurred, the best potential split for the newly created nodes are calculated. It is this loss improvement value that is used when choosing the next node to be split. When searching for these split conditions, it is not possible to check for all possible thresholds, so instead a grid search is done. There is no guarantee that the optimal threshold will be found in this grid search, so some randomness is introduced. Not having the process be deterministic is beneficial because the process will generate a different tree every time it is run, which allows us to explore different outcomes. Equations 3-5 show how the split variables for a node are calculated.

$$f, t_n = \underset{f, n}{\operatorname{argmin}}(\operatorname{loss}(\mathcal{D}_L) + \operatorname{loss}(\mathcal{D}_R)) \quad (3)$$

$$\begin{aligned}\mathcal{D}_L &= (x \in \mathcal{D} : x_f \leq t_n) \\ \mathcal{D}_R &= (x \in \mathcal{D} : x_f > t_n)\end{aligned}\tag{4}$$

$$t_n = \min(\mathcal{D}_f) + (n + r) \frac{(\max(\mathcal{D}_f) - \min(\mathcal{D}_f))}{N}\tag{5}$$

where f is the feature the split should be performed on, t_n is the threshold number n in the grid search, where $n \in [0, 1, 2, \dots, N]$, N is the size of the grid search, and r is a random number between $\pm 2\%$. The variable \mathcal{D}_f denotes the values of feature f in the set \mathcal{D} , thus $\min(\mathcal{D}_f)$ and $\max(\mathcal{D}_f)$ denotes the minimum and maximum values of feature f in dataset \mathcal{D} . It is important to note that the LMT training process makes local, greedy choices, which gives no guarantee of global optimality. For example, if an extremely good split comes after an apparently bad split, it may never be found. This is a common problem for heuristic decision tree training methods. We alleviate this issue by adding some randomness to the process of choosing the next node to be split, as shown in the following equation:

$$n_s = \underset{n}{\operatorname{argmax}}((1 + r)(\operatorname{loss}(\mathcal{D}_L^n) + \operatorname{loss}(\mathcal{D}_R^n)))\tag{6}$$

where n_s is the node to be split next, r is a random number between $\pm 2\%$, and \mathcal{D}_L^n and \mathcal{D}_R^n are as defined in Equation 4 given the best split conditions f and t for node n .

The tree has no maximum depth condition, instead it has a maximum number of leaf nodes it is allowed to have. This lets us directly state how many regions the tree is allowed to divide the input feature space into. Additionally, the tree is allowed to grow more asymmetrically, which again allows the tree to grow deeper in the area that covers the most complex regions of the feature space, while keeping the parts of the tree that covers simpler regions shallower.

The aspect that proved to be most important, and most challenging, was getting a balanced data set. The number of data points a certain area in the feature space requires in order to be represented adequately, depends on how far away from linearity the problem is in that area. To account for this, iterative tree-building was used. First, an initial data set is created through randomly sampling the feature space. Then, an initial tree is created based on that data set. The data set is then improved by letting the tree run through the environment and further samples from episodes that did not end successfully. To form the local explanations, the linear functions in the leaf nodes are utilized. The linear functions take the form of Equation 7 where a_f is feature f 's coefficient and x_f is the sample x 's value for feature f , and C is a constant. The importance I_f for each input feature concerning each output feature can be calculated as shown in Equation 8, similarly to LIME and SHAP.

$$y = \sum_f a_f x_f + C\tag{7}$$

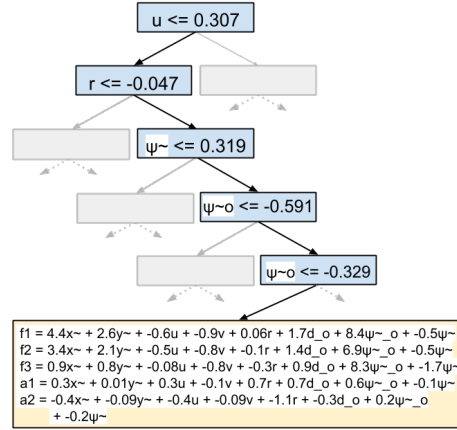
$$I_f = \frac{a_f x_f}{\sum_{j \in \forall f} |a_j x_j|} \quad (8)$$

Transparency can be divided into three categories, namely simulatable-, decomposable-, and algorithmic transparency. In [17], simulatable transparency is defined as the model not being more complex than what a human can easily comprehend. Therefore, given that the input features are understandable by humans (or at least domain experts) and the tree is not too deep, a linear model tree can be simulatable transparent. Decomposable transparency means that every part of the model must be understandable by a human without any additional tools. Since the linear model trees have univariate splits and linear function in the leaf nodes and the input and output features are understandable they are decomposable transparent, even if they grow big. Algorithmic transparency takes into account if it is possible to analyze the model with help of mathematical tools, which it is. Thus, linear models with univariate splits can be simulatable transparent but are always decomposable- and algorithmic transparent. The explanations given by the LMTs are *local, feature relevant explanations*, which means that, for each prediction, an explanation in the form of showing how much a feature pulled in a certain direction is given.

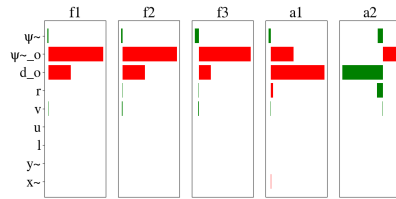
IV. SIMULATION RESULTS

For this application, there are five control inputs to be predicted. This can either be solved by fitting one tree to each output feature or combining their losses when evaluating splits. The LMT made for this work used the latter and had 681 leaf nodes, where the shallowest leaf nodes were at depth 5, and the deepest leaf nodes were at depth 15. Thus, for all practical means, the resulting tree is only decomposable transparent, and not simulatable transparent. In Figure 2a, the path along the tree from the root node to leaf node is highlighted, and in Figure 2b the explanations given in form of relative feature contribution is shown. Figure 2 is from the last time instance shown in Figure 4b. Like the PPO-agent, the LMT can act as a controller on its own. Thus, how well the LMT approximated the PPO-trained policy network can be evaluated through the difference between their outputs when they are given the same input. Table I shows the analysis of the LMT’s error through its deviation from the target output by the PPO-agent from 1000 episodes with random starting points. In most episodes, the vessel has arrived at the berthing point at step 800. After this it enters a cycle of repeating states. To prevent these states to skew the evaluation since the LMT approximates the PPO-agent quite well in the region close to the berthing point, the episodes are stopped at step 800. The highest errors usually occurs in the beginning of the episode, when the vessel still is far from the harbour, where the LMT’s actions follows the curvature of the PPO-agent’s actions but are somewhat noisy, causing an increase in the average error. Overall, the magnitude of the average error and standard deviation is moderate.

An alternative way to evaluate the LMT’s approximation of the PPO-trained DRL agent is to look at their paths when starting from the same initial point and aiming



(a) Tree path for actions. Left arrow means that the condition in branch node above was true, right arrow means it was false.



(b) Relative importance for input features for the actions

Fig. 2: Explanations and path from root node to leaf node predicting the actions in last step of Figure 4b.

for the same berthing point. A successful run is here defined as the vessel reaching the berthing point without making contact with any obstacle, while a failed run is defined as the vessel making contact with an obstacle (crashing). This criterion is not meant to evaluate the PPO-agent's behavior, because berthing can be successful even if it makes contact with the harbor if it happens slowly enough (i.e a small bump is usually tolerated), but rather as a way of evaluating how well the LMT managed to approximate the PPO-trained policy. Of note, neither of the agents have any episodes that does not end by either successfully berthing the vessel or by making contact with an obstacle. An example of a successful run by both the LMT and the PPO-agent is shown in Figure 4a. It is clear that for this starting point, the LMT has approximated the PPO-trained policy very well. The LMT fails approximately 3% more often than

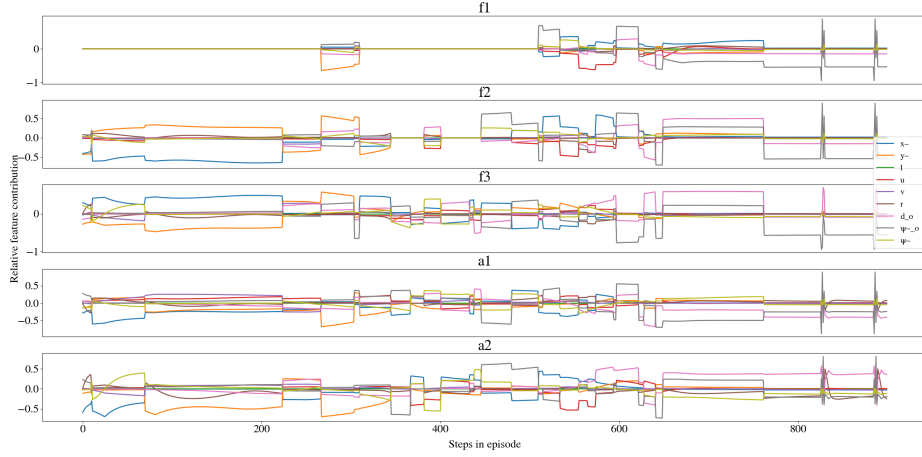


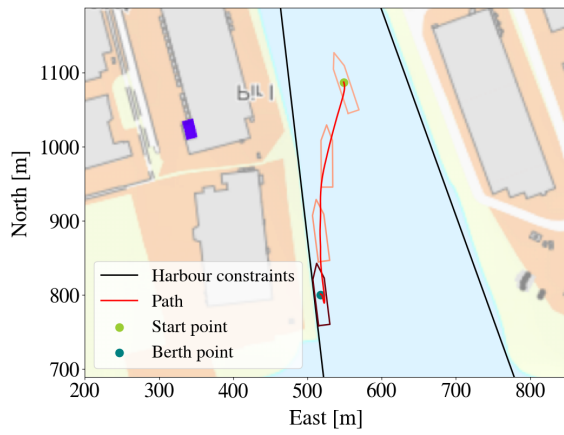
Fig. 3: Relative feature contributions given by the LMT for the episode shown in Figure 4b.

Output feature	Mean absolute error	Error standard deviation
f_1 (kN)	15.84(9.3%)	25.6(15.05%)
f_2 (kN)	14.23(8.3%)	21.7(12.76%)
α_1 (deg)	16.61(9.2%)	23.49(13.05%)
α_2 (deg)	13.75(7.63%)	20.62(11.45%)
f_3 (kN)	9.08(9.08%)	15.9(15.9%)

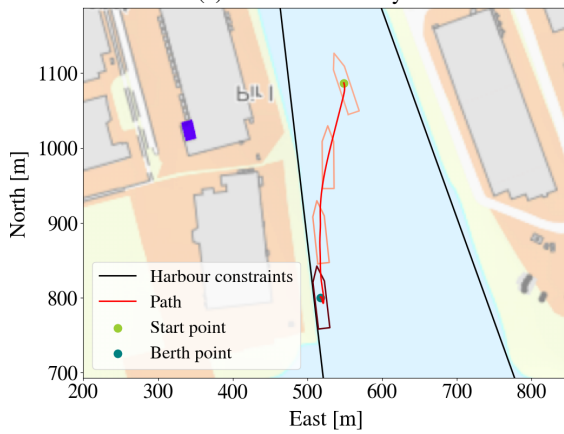
TABLE I: Output error analysis

the PPO-agent, but when looking closer at the situations where the LMT fails and the PPO-agent succeeds, it is apparent that such episodes typically unfolds similar to the episode shown in Figure 5. Even though their outcome is different, they act very similarly, so the explanations are still useful. However, the biggest difference between the two agents is most apparent when the PPO-agent fails, as can be seen in Figure 6. This could either be due to the LMT not having seen enough data from this area, that this is a more complex area so the LMT needs to grow deeper, or that the PPO-agent has not found a proper strategy for this area (which in turn can be due to the starting position being extremely hard or even impossible, for example, if the boat has an initial speed towards the harbor that is too high. However, if this deviation is detected, it might be used to raise an alarm of some sort, to alert an overseer. The explanations for the episode shown in Figure 4b are shown in Figure 3. Since the LMT only uses the linear functions in the leaf nodes as a basis for its explanations it does not take the splits along the path from the root node to the respective leaf node into consideration, even though it intuitively is relevant. This can for example be seen in the two flat areas in the

first 500 steps of the episode for output f_1 . The PPO-agent reaches the berthing point at around step number 750, and both the output of the PPO-agent and the explanations from the LMT goes into a rather repetitive cycle. LMT assigns most importance to $\tilde{\psi}_o$ and \tilde{d}_o for all actions. When looking closer at what features are changing in this part of the episode, it is clear that it is in fact $\tilde{\psi}_o$, r , and \tilde{d}_o that are changing the most, while \tilde{x} and \tilde{y} are virtually constant. Since feed-forward neural networks are one-to-one, the changing parameters are causing the change in outputs and are therefore the correct explanations. In the first ~ 250 steps it seems like the PPO-agent cares most about the three input features regarding the vessel's position relative to the berthing point (\tilde{x} , \tilde{y} and $\tilde{\psi}$). The part where the explanations are the least decisive is from approximately step number 250 to 750. LMT's explanations changes fast, which reflects that it is only piece-wise smooth. LMTs are somewhat time-demanding to build, but when it is built they can easily give real-time explanations. The LMTs only give explanations for one output feature at a time. The problem with this is that the 5 outputs are controlling the same vessel and thus dependent on each other. Additionally, f_1 and α_1 , and f_2 and α_2 are controlling the same motor. Explaining dependent factors independently will not give the whole picture. Even though relative feature contributions cannot serve as a full-fledged explanation in itself, it can be an important component of technical assurance [18].

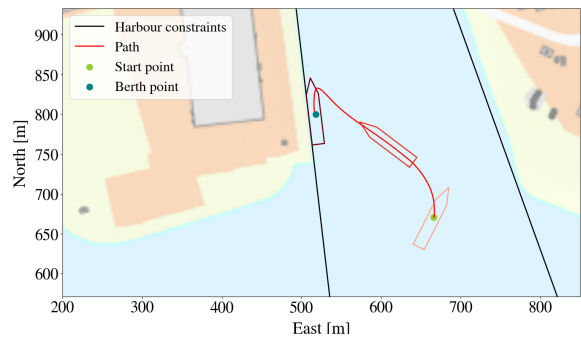


(a) Successful run by the LMT.

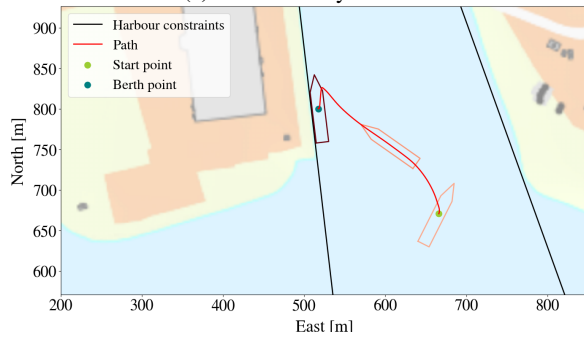


(b) Successful run by the PPO-agent.

Fig. 4: Paths of PPO-agent and LMT from same starting point.

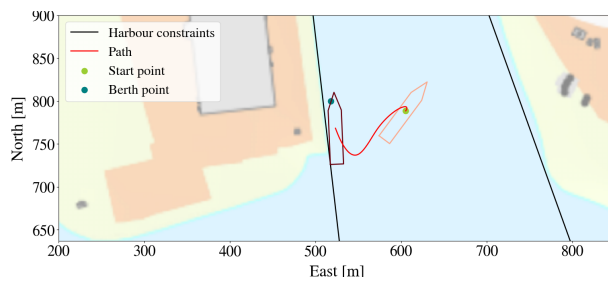


(a) Failed run by the LMT.

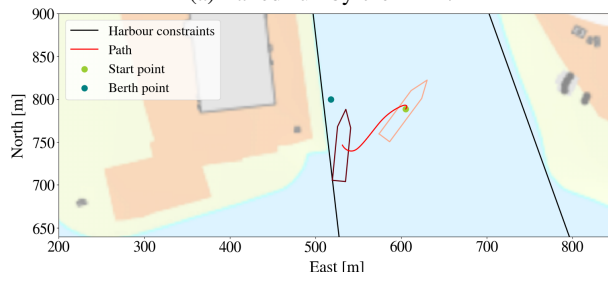


(b) Successful run by the PPO-agent.

Fig. 5: Paths of PPO-agent and LMT from same starting point.



(a) Failed run by the LMT.



(b) Failed run by the PPO-agent.

Fig. 6: Paths of PPO-agent and LMT from same starting point.

V. CONCLUSIONS AND FUTURE WORK

Linear model trees (LMTs) can contribute to tracing the outputs of a deep reinforcement learning (DRL) policy by directly linking them to the input features. In this paper, this potential was demonstrated by approximating a DRL policy controlling an autonomous surface vessel with five control inputs in a complex motion control scenario, namely docking. Although LMTs do not approximate the deep neural network in an optimal way, our results indicate that their performance is close enough to that of the original policy. In addition, the fact that LMTs are fast enough to be applicable in real-time applications, make them good candidates as components a digital assurance framework explaining the actions of black box models during operation. Future work includes improving the accuracy of the trees, and utilizing domain knowledge in both the process of building the trees and the process of extracting information about the system from the trees to make them truly understandable to several categories of end users.


REFERENCES

- [1] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, “End-to-end robotic reinforcement learning without reward engineering,” *Robotics: Science and Systems*, 2019.
- [2] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, “Learning to walk via deep reinforcement learning,” *Robotics: Science and Systems (RSS)*, 2019.
- [3] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, “Emergent tool use from multi-agent autocurricula,” in *International Conference on Learning Representations*, 2020.
- [4] A. B. Martinsen and A. M. Lekkas, “Curved path following with deep reinforcement learning: Results from three vessel models,” in *OCEANS 2018 MTS/IEEE Charleston*, pp. 1–8, 2018.
- [5] E. Meyer, A. Heiberg, A. Rasheed, and O. San, “COLREG-compliant collision avoidance for unmanned surface vehicle using deep reinforcement learning,” *IEEE Access*, vol. 8, pp. 165344–165364, 2020.
- [6] E.-L. H. Rørvik, “Automatic docking of an autonomous surface vessel : Developed using deep reinforcement learning and analysed with Explainable AI,” MA thesis. Trondheim, Norway: Norwegian University of Science and Technology(NTNU), 2020.
- [7] M. T. Ribeiro, S. Singh, and C. Guestrin, ““why should i trust you?”: Explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 1135–1144, Association for Computing Machinery, 2016.
- [8] M. T. Ribeiro, S. Singh, and C. Guestrin, “Anchors: High-precision model-agnostic explanations,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.
- [9] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems 30, 2017*, pp. 4765–4774, Curran Associates, Inc., 2017.
- [10] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, “Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods,” *AIES '20*, pp. 180–186, ACM, 2020.
- [11] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger, and S. Friedler, “Problems with shapley-value-based explanations as feature importance measures,” *Proceedings of the International Conference on Machine Learning*, pp 8083-8092, 2020.
- [12] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 3319–3328, PMLR, 06–11 Aug 2017.
- [13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, 2017.

- [14] K. P. Murphy, *Machine learning : a probabilistic perspective*. Adaptive computation and machine learning, Cambridge: MIT Press, 2012.
- [15] A. Wong, "Building model trees." https://github.com/ankonzoid/LearningX/tree/master/advanced_ML/model_tree, 2020.
- [16] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.
- [17] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information fusion*, vol. 58, pp. 82–115, 2020.
- [18] J. Glomsrud, A. Ødegårdstuen, A. Clair, and O. Smogeli, "Trustworthy versus explainable AI in autonomous vessels," ISSAV - International Seminar on Safety and Security of Autonomous Vessels, 2019.

5.2 Paper B

Postprint of [35]: Jakob Løver, **Vilde B. Gjørnum** and A. M. Lekkas. "Explainable AI methods on a deep reinforcement learning agent for automatic docking". In: *14th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles (CAMS) (2021)* doi: <https://doi.org/10.1016/j.ifacol.2021.10.086>

©2019 IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. Reprinted and formatted to fit the thesis with permission from Jakob Løver, Vilde B. Gjørnum, and Anastasios M. Lekkas under the terms of the Creative Commons Attribution License .

Explainable AI methods on a deep reinforcement learning agent for automatic docking^{*}

Jakob Løver^{*} Vilde B. Gjørnum^{*} Anastasios M. Lekkas^{*}

^{*} *Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, NO-7491 Norway (e-mail: loverjakob@gmail.com, vilde.gjarum@ntnu.no, anastasios.lekkas@ntnu.no).*

Abstract: Artificial neural networks (ANNs) have made their way into marine robotics in the last years, where they are used in control and perception systems, to name a few examples. At the same time, the black-box nature of ANNs is responsible for key challenges related to interpretability and trustworthiness, which need to be addressed if ANNs are to be deployed safely in real-life operations. In this paper, we implement three XAI methods to provide explanations to the decisions made by a deep reinforcement learning agent: Kernel SHAP, LIME and Linear Model Trees (LMTs). The agent was trained via Proximal Policy Optimization (PPO) to perform automatic docking on a fully-actuated vessel. We discuss the properties and suitability of the three methods, and juxtapose them with important attributes of the docking agent to provide context to the explanations.

Keywords: Marine control systems, Explainable Artificial Intelligence, Deep Reinforcement Learning, Autonomous ships, Docking

1. INTRODUCTION

Despite the progress in artificial intelligence over the past decade, there is still a significant trade-off between interpretability and accuracy. As neural networks become increasingly complex in dimensionality and design, understanding the underlying decision-making becomes equally difficult. Being unable to explain the reasoning behind black box decisions is unacceptable for safety-critical applications. Explainable Artificial Intelligence (XAI) is a relatively recent movement in the AI community, referring to tools that allow humans and/or

^{*} This work was supported by the Research Council of Norway through the EXAIGON project, project number 304843

machines to understand the decision-making rationale of AI systems. XAI systems are usually characterized as being either intrinsically interpretable, or post-hoc interpretable. XAI systems that only work for specific predictors are characterized as model-specific. If they are not concerned about the internal predictor structure, they are model-agnostic. Interpretability is defined by Miller (2019) as the degree to which an observer can understand the cause of a decision, and can roughly be divided into two classes Molnar (2019)

Local Interpretability Being able to explain reasoning behind single decisions or groups of decisions.

Global Interpretability Understanding the reasoning behind the entire model behavior on a holistic or modular level.

An explainer should be interpretable, locally faithful, model-agnostic, and should provide a global perspective Ribeiro et al. (2016). A locally faithful explainer is one that exerts local fidelity. There are several kinds of networks that are not inherently interpretable. For example, convolutional neural networks with multiple successive matrix convolutions end up with being far too complex for humans to understand. This requires the use of explainers that can be applied to any black box algorithm post-hoc. In this paper, model-agnostic methods will be discussed.

Achieving holistic global interpretability is often hard to achieve in practice. Being able to understand the model on a modular level is however much closer in reach. For linear regression models with a large feature space for example, modular interpretability can be achieved through looking at the weights, but holistic model interpretability is hard to achieve because feature spaces with dimensions larger than three are simply inconceivable for humans Molnar (2019).

1.1 Related work and motivation

The motivation for this paper was the scarcity of research on XAI for cyber-physical systems and deep reinforcement learning. Most of the available literature are survey papers. Additionally, Explainable AI is a fairly young research topic. To the author’s best knowledge, no paper has compared the explanations from white-box and black-box models for a cyber-physical system using the methods described in this paper. Shapley-based methods such as SHAP has been proposed by researchers as a possible first step to achieve a global understanding of reinforcement learning agents Heuillet et al. (2021). SHAP was successfully implemented for a DRL agent in Liessner. et al. (2021); He et al. (2021), but no comparison to other methods were performed. It is therefore natural to investigate the method SHAP, and its related method LIME. SHAP and LIME build on many of the same ideas, but SHAP provides some desirable guarantees that LIME do not. However, LIME is often lauded as a fast explainer. It was therefore of interest to see how its explanations compared to SHAP.

2. BACKGROUND

This section will present an overview of the docking problem, as well as the docking agent which was developed by Rørvik (2020). Brief theory behind the three XAI methods applied to the agent will also be presented. Local Interpretable Model-Agnostic Explanations (LIME) is an explainer that samples the locality of the sample to explain, and builds a linear regression model around the sample to provide explanations. Shapley Additive Explanations (SHAP) is an explainer rooted in a series of fairness axioms. Finally, the method Linear Model Trees (LMTs) will be presented, which builds a regression tree that estimates the agent.

2.1 Docking

Docking involves various complex maneuvers to steer a vessel from the open sea towards a designated area in the harbor area called a berth. It has been characterized as one of the hardest problems to solve within ship control Tran and Im (2012). Not only does the vessel need to take into account the speed limits of the harbor, distance to other ships and obstacles, but it has to simultaneously deal with extremely nonlinear motions, reduced maneuverability at low speeds, and environmental forces.

Historically, auxiliary devices such as tug boats have been used to dock large vessels, but with the increased freedom of maneuverability in the form of azimuth thrusters and tunnel thrusters, more sophisticated strategies can be employed. Performing automatic docking using auxiliary devices together with neural networks have already proven successful Tran and Im (2012); Ahmed and Hasegawa (2013); Im and Nguyen (2018). For example, Im and Nguyen (2018) used a neural network architecture with one hidden layer to perform supervised learning. The artificial neural network (ANN) was trained from data collected by observing a skilled captain berth the vessel, but there are a multitude of reasons why this is a sub-optimal approach. The captain would have to berth the ship perfectly every time, which is not possible in practice. Errors are bound to happen, and the ANN will be entirely limited to the data provided. Even though the captain may have experience docking the vessel, the procedure the captain follows may not necessarily be the most efficient for any given scenario. A major drawback with some of these aforementioned implementations is that they do not generalize well from one arbitrary port to another. These methods also had strict limits from what angle the vessel may approach the berth. Recent advances such as Nguyen (2020) allowed an ANN to berth both starboard and port side on multiple ports successfully without re-training, but did not take into account environmental forces such as wind and waves. Some publications were found using deep reinforcement learning to solve similar problems, but most of them were applied to underwater vehicles Anderlini et al. (2019). Other methods that have been proposed are backstepping controllers Zhang et al. (2020) and model predictive control Martinsen et al. (2019).

2.2 Automatic docking as a deep reinforcement learning problem

Deep reinforcement learning (DRL) agents have already been shown to perform well in collision avoidance and trajectory following Meyer et al. (2020). Using a deep reinforcement learning agent has been proposed as a solution to the docking problem Rørvik (2020). The docking agent was proven to successfully solve the docking scenario from a variety of poses relative to the berth. It was trained using Proximal Policy Optimization (PPO) with two hidden layers of 400 hidden units each, *ReLU* activations for both hidden layers, and a hyperbolic tangent activation for the output layer.

The agent was trained on a three degrees-of-freedom vessel. It is 76.2 meters long, weighing 6000 tonnes in dead weight Martinsen et al. (2019). The vessel actuators are three thrusters: one tunnel thruster, and two azimuth thrusters. Their numbering and location on the vessel is shown in Figure 1. The tunnel thruster is used to create a side force on the vessel, while the two azimuth thrusters are mounted in the aft of the ship, and are rotatable thrusters. The action vector is described in Equation 1. f_i is the applied thrust measured in newton, and a_i is the azimuth angle in radians for thruster i .

$$y = [f_1 \ f_2 \ f_3 \ a_1 \ a_2] \quad (1)$$



Fig. 1. Thruster numbering on vessel Martinsen et al. (2019)

The state vector is described in Equation 2, and the respective state descriptions in Table 1.

$$x = [\tilde{x} \ \tilde{y} \ l \ u \ v \ r \ d_{obs} \ \tilde{\psi}_{obs} \ \tilde{\psi}] \quad (2)$$

3. IMPLEMENTATION

3.1 Computational Hardware

The results were produced using a workstation running a virtualized Ubuntu 20.04 environment. The workstation has an AMD Ryzen 9 3950X CPU with 32GB of allocated RAM.

3.2 LIME

Local Interpretable Model-Agnostic Explanations (LIME) creates locally interpretable explanations for a single data point. LIME creates a linear surrogate

State	Description
\tilde{x}, \tilde{y}	The Cartesian distances from origin of the vessel to the target position, in body frame. x-direction is north-south, y-direction is east-west.
$\tilde{\psi}$	The relative difference between heading of vessel and heading of the desired target.
u, v, r	Linear and rotational velocities of the vessel, in body frame.
l	A binary variable describing whether the vessel is in contact with land. Only used when training the agent.
$d_{obs}, \tilde{\psi}_{obs}$	The distance from the vessel’s edge to the closest obstacle and the relative heading between the vessel and the closest obstacle.

Table 1. Description of states Rørvik (2020).

model that approximates the local behavior of the predictor for a single prediction. By creating a perturbed dataset made up of local perturbations around the decision in question, the importance of each feature in the black box predictor can be inferred. The output from LIME is a vector of coefficients that suggests how increasing or decreasing variables affect the prediction. As described in Ribeiro et al. (2016), Equation 3 illustrates how LIME creates an explanation $\xi(x)$ for an instance x :

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \quad (3)$$

$\mathcal{L}(f, g, \pi_x)$ is a measure of how unfaithful a model g is when approximating f in the locality π_x . All predictions are weighted according to π_x , also called a kernel. As the samples stray further from the prediction LIME is explaining, they will carry lower weight. $\Omega(g)$ is a measure of complexity of the explanation. LIME balances out this equation by minimizing $\mathcal{L}(f, g, \pi_x)$ while keeping $\Omega(g)$ low enough to be human interpretable.

For tabular data, LIME will also have to be supplied a background dataset. The dataset will be used to compute the mean, standard deviation, and discretize the feature into quartiles Ribeiro (2018), which are used to scale the data. When LIME samples perturbed instances, it first samples from a normal distribution. Then, the samples are multiplied by the standard deviation, and the mean is added.

The GitHub LIME implementation made available by Ribeiro (2018) was used to explain the docking episode. The number of neighbors were set to 5000, which is the amount of perturbations per sample LIME will perform to evaluate feature importance. For every second of the docking episode, LIME was applied to each data point one time for every action.

3.3 Kernel SHAP

The Shapley value is a coalitional game theory concept based on a set of fundamental axioms of fairness. It is proven that the Shapley value is the only solution that satisfies these axioms. The Shapley value therefore provides a unique solution to distribution of reward based on work contributed Young (1985). The Shapley value serves as the basis for the Shapley Additive Explanations (SHAP) framework to quantify how much each feature in a neural network contributes to the output.

Several variations of SHAP exist, but this paper focuses on Kernel SHAP. Kernel SHAP is a model-agnostic method of approximating feature attributions. The feature attributions are approximated instead of calculated exactly, because computing them exactly is time-consuming. Kernel SHAP builds on the LIME framework, as shown by Lundberg and Lee (2017) when the following parameters are inserted into Equation 3:

$$\Omega(g) = 0 \tag{4}$$

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|} |z'| (M-|z'|)} \tag{5}$$

$$\mathcal{L}(f, g, \pi_x) = \sum_{z' \in Z} [f(h_x^{-1}(z')) - g(z')]^2 \pi_x(z') \tag{6}$$

$|z'|$ is the number of present features in z' , and h_x is a function that maps a coalition z' to a valid instance. h_x is needed because in practice, "removing" a feature is not trivial for models with a fixed-size input. Setting the feature values to 0 is also not always desirable either. Instead, Kernel SHAP "removes" features by sampling the background data and replacing the missing features with feature values from one of these random samples.

The most significant difference from LIME is the weighting kernel. Instead of weighting samples according to how close they are to the sample to be explained, Kernel SHAP weighs small and large coalitions heavier through the Shapley kernel $\pi_x(z')$. Small coalitions—coalitions where many features are missing—allows the significance of presence of a feature to be studied. Large coalitions, where few features are missing, allows the algorithm to see how the model prediction changes with respect to the absence of features. Mid-sized coalitions do not say much about either situation, and are weighted less.

The *KernelExplainer* object of the SHAP implementation provided by Lundberg and Lee (2017) was used to create SHAP values for the states in the docking episode. The background data for SHAP was first summarized using a K-means summarizer with 100 neighbors to reduce computation time.

3.4 Linear Model Trees

A decision tree (DT) is a machine learning model that divides the input domain into subregions by performing multiple evaluations on data, and assigns a prediction to each of the subregions. This can be visualized as a tree-like structure, where each of the splits are called internal nodes. When there are no more splits to be evaluated, a leaf node has been reached. The prediction depends on which leaf node, or subregion, the input data falls into. Regression trees are decision trees where the predicted output is a constant real number. Model trees on the other hand can output predictions based on any type of model. Linear Model Trees (LMT) is a regression tree where the only structural difference is that instead of constant predictions in the leaf nodes, an LMT uses a linear function to form its prediction. For DTs, the splits in the input data can be either multivariate or univariate. Univariate splits are splits that depends on only one input feature at a time. When the splits are done using multiple input features, they are called multivariate splits. DTs with multivariate splits are called oblique DTs. For the LMT implementation used, the splits on the input data are univariate. This is done to retain interpretability and reduce computation time. Growing an LMT is done by greedily splitting the data. This gives no guarantees for global optimality, as a seemingly bad split may cause a good split to never be found Gjørø et al. (2021).

By using an LMT to form a piece-wise linear approximation of a black box predictor, the simpler structure of the LMT can be used to understand the predictions made by the black box predictor. The resulting tree sacrifices some accuracy to give more interpretability. By weighting the linear regression coefficients in the leaf nodes, the predictions by the LMT can be interpreted. From Gjørø et al. (2021), the linear functions in the leaf nodes can be written on the form

$$y = \sum_{f \in \mathcal{F}} a_f x_f + C \quad (7)$$

where y is the model prediction. a_f is the linear regression coefficient for the feature x_f , and C is a constant. \mathcal{F} is the set of all features. Total feature importance I_f for each feature f can then be calculated as

$$I_f = \frac{a_f x_f}{\sum_{j \in \mathcal{F}} |a_j x_j|} \quad (8)$$

The LMT implementation used is based on an adaptation of classification and regression trees from Wong (2020). The modifications done allowed the tree to grow to a maximum number of leaf nodes instead of a maximum depth, and added randomization in the process of searching for thresholds and choosing the next node splits Gjørø et al. (2021). The LMT used contains 681 leaf nodes,

with the shallowest leaf node situated at depth 5, and the deepest at depth 15. The LMT was trained by collecting the states and actions from the PPO agent from 1000 docking episodes. The starting points were chosen randomly for each docking episode in order to capture as much of the vessel dynamics as possible.

3.5 Background Data

As mentioned in Section 3.2 and Section 3.3, we need to supply these algorithms with a background dataset. Motivated by observed values, the following table was proposed in Rørvik (2020) as a representative dataset for the DRL agent.

Variable	Valid range
x_d [m]	800
y_d [m]	517.8
x [m]	$(x_d - 400, x_d + 400)$
y [m]	$(y_d - 400, y_d + 400)$
ψ [rad]	$(-\frac{\pi}{4}, \frac{\pi}{4})$
u [m/s]	$(-0.5, 0.5)$
v [m/s]	$(-0.05, 0.05)$
r [rad/s]	$(-0.005, 0.005)$

Table 2. Valid ranges for the dataset.

Data was first collected by letting the DRL agent perform ten docking episodes from ten different locations. From this data, 2000 data points that fell within the valid ranges of Table 2 were sampled as part of the dataset. The final dataset to serve as background data for SHAP and LIME therefore had the dimensions (2000,9).

4. RESULTS

The results were inspected by plotting the vessel’s position in Figure 2 together with the vessel’s actions in Figure 3. Figure 4 is a screenshot of a video setup that allowed inspection of the force and torque vector of the vessel in body frame together with actions and explanations. The dial in the bottom right contains an orange line, and a blue circular bar plot. The orange line is the calculated force vector applied on the vessel in body frame measured from the center. For example, 100 % of max force in the north-westward direction results in an orange line from the center which stops at the outer edge of the 45 degree mark. The blue torque vector denotes how much torque the vessel is applying, and in what direction. When the actuators exert 25 % of max torque, the circular blue bar plot will be filled a quarter of the way, stopping at the 270 degree mark. These visualizations contributed to put the explanations in context.

Figure 5 shows the feature importances from Kernel SHAP for an entire episode. Note that the y-axes of the plots represent how much each state contributed to the output of the respective action in the positive or negative direction. The sum of all

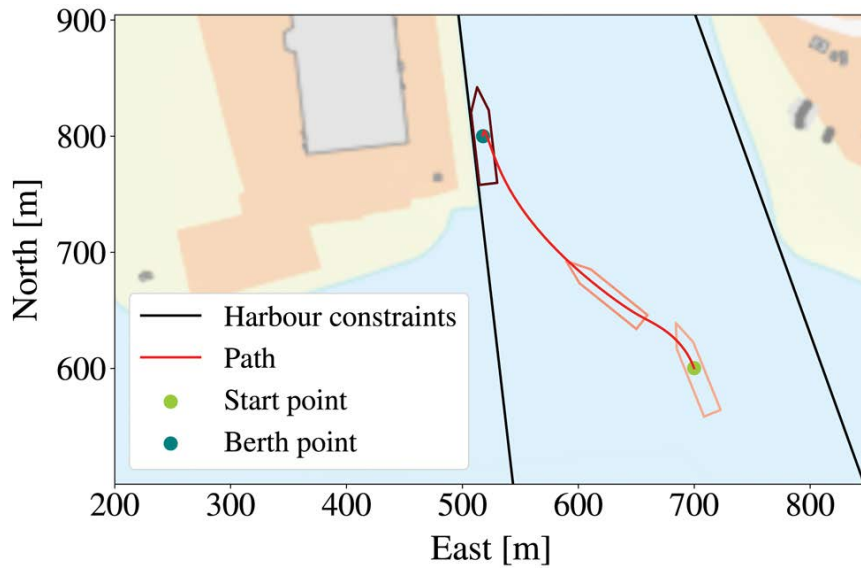


Fig. 2. Trajectory of vessel during a docking episode.

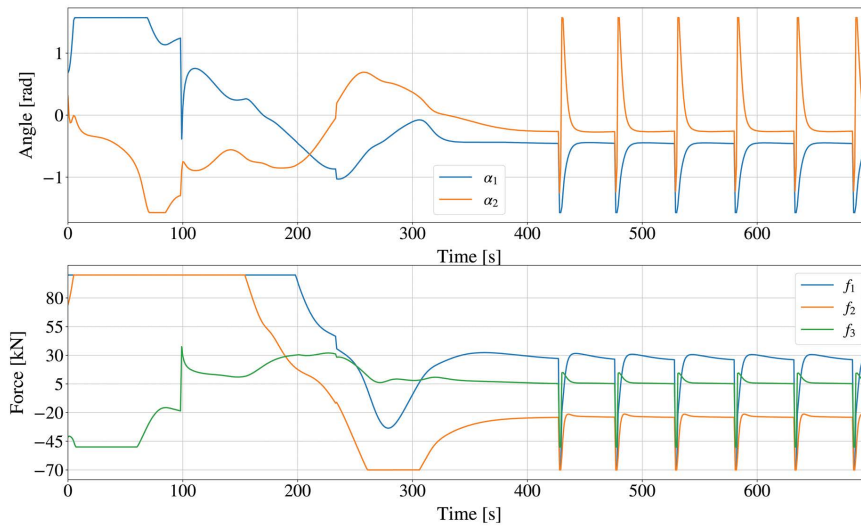


Fig. 3. Actions for a docking episode.

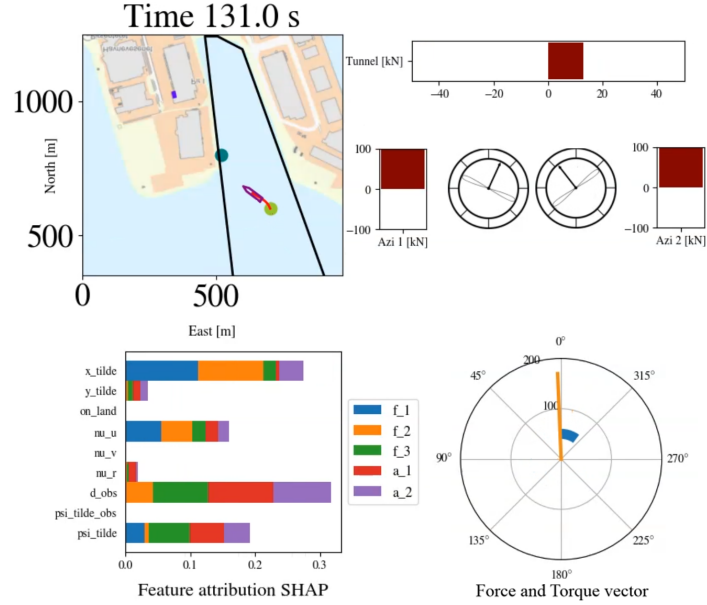


Fig. 4. Visualization of force and torque in body frame together with actions, trajectory, and feature attributions. The bottom left bar plot represents a weighted sum of the absolute value of the feature attribution per state across all actions such that the sum of all bars equal 1.

feature attribution values equal 0. Kernel SHAP provided smooth explanations, especially during the approach phase, where the vessel is approaching the berth. The actuators are not changing too much in each time step, and this approach phase is fairly slow. The explanations also intuitively make sense.

In the beginning of the episode, high importance is attributed to \tilde{x} for the thruster forces and d_{obs} for the azimuth angles. As the vessel performs for a clockwise rotation at around 120 seconds into the episode by applying a positive tunnel thruster force, the importance of the rotational velocity r begins to increase for f_3 . The vessel slightly overshoots the berth, and needs to align itself with the berth. The surge velocity is almost solely the main contributors to a negative thruster force on the azimuths to slow down the vessel. At this point, the sway velocity v increases in importance, and the vessel moves closer in the \tilde{y} -direction to the berth. During the final seconds of the berthing phase, d_{obs} and $\tilde{\psi}_{obs}$ begins to increase in importance to bring the vessel into its final position.

The vessel reaches the berth in about 400 seconds, and begins to slightly oscillate by the berth in a "steady state." The actuators first apply a sharp corrective action counter-clockwise, with high importance for the state $\tilde{\psi}_{obs}$, seemingly to

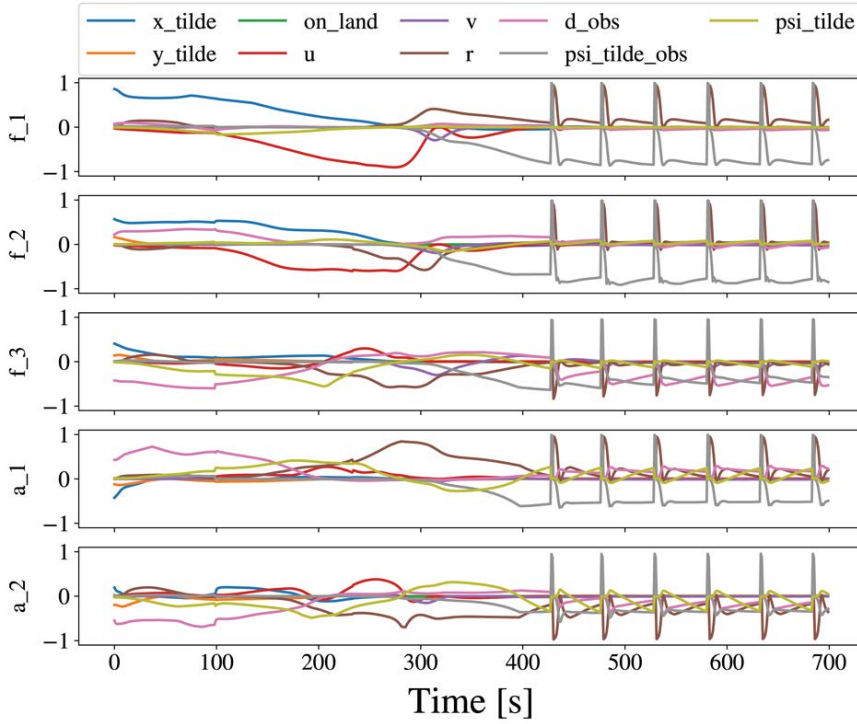


Fig. 5. Feature attributions for Kernel SHAP.

correct the pose of the vessel. Shortly after, a longer lasting clockwise torque is applied with importance for r to stop the vessel from rotating and correct its alignment to the berth.

LIME has a large disadvantage in that it does not take into account the global sample space when building the linear models as opposed to Kernel SHAP. This can be seen in practice from the LIME explanations in Figure 6, and comparing them to Kernel SHAP. It is observed that the feature attributions from LIME are clearly noisier. In general, the explanations from LIME does seem to follow the explanations from Kernel SHAP, but are not nearly as smooth.

The LMT exhibits a more discrete behaviour than Kernel SHAP and LIME. This is shown in Figure 7. There are also several points where their explanations differ, but they are equally intuitive. For example, when the vessel is making its final move towards the berth after overshooting it around 300 seconds into the episode, the agent applies a large force vector backwards towards the berth. The most dominating feature at this point is d_{obs} . LIME seemed to agree that d_{obs}

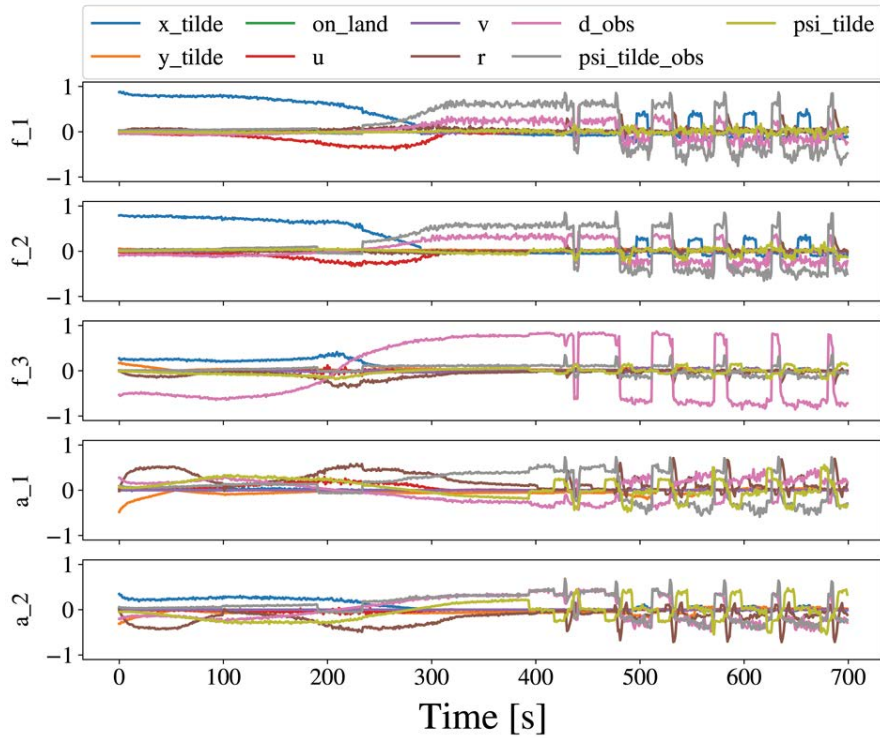


Fig. 6. Feature attributions for LIME.

was quite important. This is however quite different from SHAP, which attributed more importance to the surge velocity and the rotational states.

LIME and SHAP are versatile, as the background data can be continuously modified. The LMT can not easily be modified this way, as it needs to be retrained. The number of samples in the background dataset mattered greatly when leaving out a K-means summarizer. When using 2000 samples as the background data, running Kernel SHAP without a K-means summarizer was infeasible, as the computer eventually ran out of memory.

LIME was concluded to be ill-suited for this application. LIME was implemented with the expectation of being able to run real-time, but spent about five times longer than Kernel SHAP to explain a data sample. The weighting kernel in LIME is also entirely arbitrary, and introduces another unnecessary tuning parameter. LMT was fastest method out of all three, and can be implemented to run real-time. Its explanations were intuitive, but were not as smooth as those from SHAP due to its piece-wise linearity.

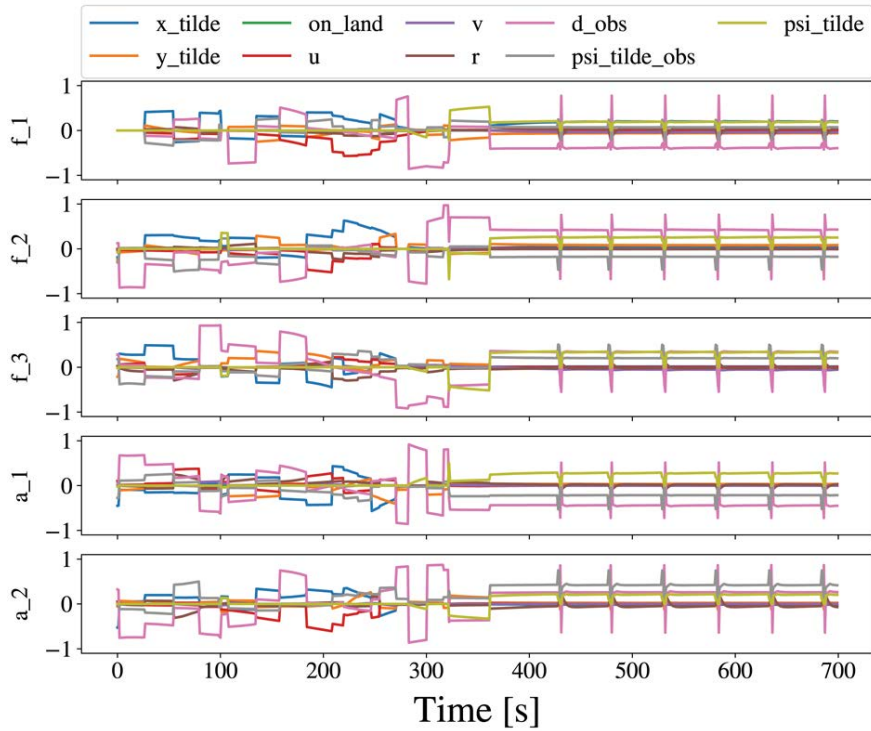


Fig. 7. Feature attributions for LMT.

Kernel SHAP was efficient, but assumes that the features are independent of each other. This is indeed very problematic for many applications. It is not well known as to what degree the features in the model correlate. There may for example be some correlation between \tilde{x} and \tilde{y} . Recent attempts have been made to remedy this assumption of independent features. Tree SHAP—another method of approximating feature attributions—does not rely as heavily on this assumption Lundberg et al. (2019). This method does not however deliver satisfactory accuracy, and may even give highly inaccurate results when dependent features are present Aas et al. (2020). An overview of the XAI methods and their pros and cons can be found in Table 3.

Even though Shapley-value based explainers have been widely used as a measure of feature importance, they may not be suitable as explainers for neural networks Kumar et al. (2020). SHAP may for example not alarm about any potential biases in the data, or whether the model accuracy would increase with or without a feature present. Post-hoc perturbation based methods such as SHAP and LIME are also vulnerable to deliberate attacks. Scaffolding is a technique that effectively

hides the biases of any given classifier by allowing an adversarial entity to craft an arbitrary desired explanation Slack et al. (2020). This has the potential to create biased predictions with innocuous explanations, which does not sit well with the safety-critical nature of a robotic system such as an ASV.

Explainer	Pros	Cons	Time per explanation
LIME	<ul style="list-style-type: none"> • Can easily adapt to new data 	<ul style="list-style-type: none"> • Slow • Noisy explanations • Only local explanations 	16.57 s
Kernel SHAP +K-means	<ul style="list-style-type: none"> • Can easily adapt to new data • Smooth explanations 	<ul style="list-style-type: none"> • Slow • Only local explanations 	3.6486 s
LMT	<ul style="list-style-type: none"> • Fast • Can run in real-time • Drop-in replacement for model • Can provide global explanations 	<ul style="list-style-type: none"> • Discrete explanations • Long time to train 	0.0012 s

Table 3. Overview of XAI methods used.

5. CONCLUSION

XAI algorithms may be employed to provide explanations in a deep reinforcement learning agent for robotic applications. Through visualizing the explanations together with the states of the system, we can gain insight into the reasoning behind black box predictors. This insight can assist in validating the performance of the autonomous system, and provide assistance during the certification process. Methods such as LMTs are fast enough to provide intuitive explanations real-time, which can be used in real-time visualizations or control loops. Perturbation-based methods such as SHAP and LIME are slow, and might not be suitable for real-time explanations. SHAP is however a viable alternative that gives smooth and intuitive explanations post-hoc, but should be used with caution due to its vulnerability to create biased predictions with innocuous explanations.

Further work may involve implementing Optimal Regression Trees (ORT), a method which expresses the node splitting as an mixed integer optimization problem Bertsimas and Dunn (2019) instead of greedily growing the tree as with the LMTs. Near-optimal Nonlinear Regression Trees (NNRT) Bertsimas et al. (2021) should also be investigated. NNRT is another tree method where

parameters for multivariate splits in the tree are found through gradient descent to build non-linear prediction functions in the leaf nodes.


REFERENCES

- Aas, K., Jullum, M., and Løland, A. (2020). Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence*, 298, 103502.
- Ahmed, Y.A. and Hasegawa, K. (2013). Automatic ship berthing using artificial neural network trained by consistent teaching data using nonlinear programming method. *Engineering Applications of Artificial Intelligence*, 26(10), 2287 – 2304.
- Anderlini, E., Parker, G., and Thomas, G. (2019). Docking control of an autonomous underwater vehicle using reinforcement learning. *Applied Sciences*, 9, 3456.
- Bertsimas, D. and Dunn, J. (2019). *Machine learning under a modern optimization lens*. Dynamic Ideas LLC.
- Bertsimas, D., Dunn, J., and Wang, Y. (2021). Near-optimal nonlinear regression trees. *Operations Research Letters*, 49(2), 201–206.
- Gjærum, V., Rørvik, E.L.H., and Lekkas, A.M. (2021). Approximating a deep reinforcement learning docking agent using linear model trees. *Submitted to IEEE European Control Conference (ECC)*.
- He, L., Nabil, A., and Song, B. (2021). Explainable deep reinforcement learning for uav autonomous navigation.
- Heuillet, A., Couthouis, F., and Díaz-Rodríguez, N. (2021). Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214, 106685.
- Im, N.K. and Nguyen, V.S. (2018). Artificial neural network controller for automatic ship berthing using head-up coordinate system. *International Journal of Naval Architecture and Ocean Engineering*, 10(3), 235 – 249.
- Kumar, I.E., Venkatasubramanian, S., Scheidegger, C., and Friedler, S. (2020). Problems with shapley-value-based explanations as feature importance measures. In *International Conference on Machine Learning*, 5491–5500.
- Liessner, R., Dohmen, J., and Wiering, M. (2021). Explainable reinforcement learning for longitudinal control. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, 874–881. INSTICC, SciTePress.
- Lundberg, S.M., Erion, G.G., and Lee, S.I. (2019). Consistent individualized feature attribution for tree ensembles.
- Lundberg, S.M. and Lee, S.I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, 4765–4774. Curran Associates, Inc.
- Martinsen, A.B., Lekkas, A.M., and Gros, S. (2019). Autonomous docking using direct optimal control. *IFAC-PapersOnLine*, 52(21), 97–102.

- Meyer, E., Robinson, H., Rasheed, A., and San, O. (2020). Taming an autonomous surface vehicle for path following and collision avoidance using deep reinforcement learning. *IEEE Access*, 8, 41466–41481.
- Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267, 1–38.
- Molnar, C. (2019). *Interpretable Machine Learning*. <https://christophm.github.io/interpretable-ml-book>.
- Nguyen, V. (2020). Investigation of a multitasking system for automatic ship berthing in marine practice based on an integrated neural controller. 8, 1–23.
- Ribeiro, M., Singh, S., and Guestrin, C. (2016). “why should i trust you?”: Explaining the predictions of any classifier. 97–101.
- Ribeiro, M.T. (2018). LIME. URL <https://marcotcr.github.io/lime/>.
- Rørvik, E.L.H. (2020). Automatic Docking of an Autonomous Surface Vessel. *Master thesis. Norwegian University of Science and Technology (NTNU)*.
- Slack, D., Hilgard, S., Jia, E., Singh, S., and Lakkaraju, H. (2020). Fooling lime and shap: Adversarial attacks on post hoc explanation methods. 180–186.
- Tran, V.L. and Im, N.K. (2012). A study on ship automatic berthing with assistance of auxiliary devices. *International Journal of Naval Architecture and Ocean Engineering*, 4(3), 199–210.
- Wong, A. (2020). *ankonzoid/LearningX*. URL <https://github.com/ankonzoid/LearningX>.
- Young, H.P. (1985). Monotonic solutions of cooperative games. *International Journal of Game Theory*, 14(2), 65–72.
- Zhang, Y., Zhang, M., and Zhang, Q. (2020). Auto-berthing control of marine surface vehicle based on concise backstepping. *IEEE Access*, 8, 197059–197067.

5.3 Paper C

Postprint of [18]: **Vilde B. Gjørum**, Inga Strümke, Ole Andreas Alsos, and Anastasios M. Lekkas. "Explaining a deep reinforcement learning docking agent using linear model trees and user adapted visualizations". In: *Journal for Marine Science and Engineering* 9(11) 1178 (2021). doi: <https://doi.org/10.3390/jmse9111178>

©2021 Vilde B. Gjørum, Inga Strümke, Ole Andreas Alsos, and Anastasios M. Lekkas. Reprinted under the terms of the Creative Commons Attribution License 

Article

Explaining a deep reinforcement learning docking agent using linear model trees with user adapted visualization

Vilde B. Gjørum¹, Inga Strümke¹, Ole Andreas Alsos² and Anastasios M. Lekkas^{1,3}

¹ Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway

² Department of Design, Norwegian University of Science and Technology, Trondheim, Norway

³ Centre for Autonomous Marine Operations and Systems, Norwegian University of Science and Technology, Trondheim, Norway

* Correspondence: vilde.gjarum@ntnu.no

Abstract: Deep neural networks (DNNs) can be useful within the marine robotics field, but their utility value is restricted by their black-box nature. Explainable artificial intelligence methods attempt to understand how such black boxes make their decisions. In this work, linear model trees (LMTs) are used to approximate the DNN controlling an autonomous surface vessel in a simulated environment and then run in parallel with the DNN to give explanations in the form of feature attributions in real time. How well a model can be understood depends not only on the explanation itself, but also on how well it is presented and adapted to the receiver of said explanation. Different end users may need both different types of explanations, as well as different representations of these. The main contributions of this work are 1) improving accuracy and build time of a greedy approach for building LMTs by introducing ordering of features in the splitting of the tree, and 2) suggesting a visualization of the docking agent, the environment, and the feature attributions given by the LMT for when the developer is the end user of the system, and another visualization for when the seafarer or operator is the end user, based on their different needs and characteristics.

Keywords: Deep Reinforcement Learning, Autonomous Surface Vessel, Explainable Artificial Intelligence, Linear Model Trees

Citation: Gjørum, V. B.; Strümke, I.; Alsos, O. A.; Lekkas, A. M.

Explaining a deep reinforcement learning docking agent using linear model trees with user adapted visualization. *J. Mar. Sci. Eng.* **2021**, *1*, 0. <https://doi.org/>

Received:

Accepted:

Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2023 by the authors. Submitted to *J. Mar. Sci. Eng.* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Machine learning is the sub-field of artificial intelligence (AI) dedicated to self-learning systems that use data to adjust their predictions. Among the most remarkable advancements in machine learning methods has been the evolution from artificial neural networks to deep architectures, known as deep neural networks (DNNs), forming the class of deep learning [1,2]. Reinforcement learning (RL) is a branch of machine learning where an agent learns a strategy, referred to as a *policy*, for controlling an agent in an environment based on an evaluation of the agent's interactions with the environment [3], called *rewards*. That is, the policy maps from a state to an action, similar to a controller. Several noteworthy accomplishments have been made with the use of deep reinforcement learning (DRL), such as learning to play atari games directly from image pixels [4], or discovering new strategies in a simulated hide-and-seek environment [5]. DRL has also shown to be a very useful tool for accomplishing difficult tasks in robotics, one advantage being that it does not require a mathematical model of the agent or the environment. In [6], DRL was used to perform 20 different simulated physical tasks. In [7], DRL was used to conduct various manipulation tasks with a dexterous, robotic hand. In [8], a DRL-agent learned to walk on flat surfaces, but could also handle unseen, more challenging surfaces as well. The potential for reduced costs and increased safety has inspired the work towards autonomous ships, and the

36 industry has already shown promising autonomous surface vessels (ASVs), such as
37 Falco [9] and Yara Birkeland [10]. DRL has also been used for marine operations, such
38 as autonomous path-following [11–13], collision avoidance [14,15], and for docking [16,
39 17]. In [17], a DRL-agent was trained to perform docking of an ASV in a simulated
40 environment based on Trondheim harbour. Even though the agent showed promising
41 and rather convincing results, its applicability to real-life problems is reduced by the
42 lack of understanding of how the DNN-policy makes its decisions. This is because
43 the many parameters and interconnections of DNNs makes their inner workings hard
44 for humans to interpret. For this reason, DNNs are considered to be *black boxes*. The
45 field of explainable artificial intelligence (XAI) is dedicated to develop methods for
46 explaining such black box models [18]. The objective is to gain increased understanding
47 regarding how the black box model works and why it behaves the way it does. XAI
48 methods can thus be used to interpret and justify the decisions made by a black-box
49 model, control and prevent erroneous actions, improve the model, and even discover
50 new strategies, correlations in the data set or application [19]. In [20], the importance of
51 explaining AI-systems to non-expert users is highlighted, especially with consideration
52 for the preparation for wider-scale operations of ASVs. The combination of explanations
53 and thorough testing of the AI system is crucial for gaining the trust needed for the
54 autonomous system to be deployed [21]. The authors of [21] also argue that depending on
55 the role and needs of recipient of the explanation, the explanations should be customized
56 regarding several aspects:

- 57 1. Whether all the details of an explanation should be provided, or if it is preferable
58 to highlight only the most relevant parts, with respect to the specific end-user, of
59 the explanation.
- 60 2. Whether the end-user need to process the explanations in real-time or not.
- 61 3. In what way the explanation will be presented to the end-user.

62 In this work, we consider two different end-users, the developer and the seafarer/operator.
63 Their main differences lies in their background knowledge, how much risk they asso-
64 ciate with the predictions made by the DNN, and how fast they need to evaluate the
65 predictions from the DNN and the explanations from the explainer.

66 Among the most widely used explanation methods are local interpretable model-
67 agnostic explanations (LIME) [22], Anchors [23], integrated gradients (IG) [24], Shapley
68 additive explanations (SHAP) [25] and SAGE [26]. The main characteristics of XAI-
69 methods are outlined in Table 1. IG is a *model-specific* method, only applicable to
70 differentiable models, while LIME, Anchors, SHAP and Shapley additive global im-
71 portance (SAGE) are *model-agnostic* methods. While SAGE provides *global explanations*,
72 SHAP, LIME, Anchors, and IG give *local explanations*. Preliminary work was presented
73 in [27], where we approximated the DRL-policy from [17] with a linear model tree (LMT)
74 and used the linear functions in the active leaf node to form explanations in the form of
75 feature attributions. The LMTs was built by a greedy method, which was quite sensitive
76 to the dataset. To remedy this, a very time demanding iterative data sampling process
77 was used to ensure that the LMT got enough samples from regions it did not approx-
78 imate as well. In this paper, we improve the approximation by enforcing the order of
79 which features to better match guidance system logic when searching for the splits in
80 each branch node at different depths of the tree. Not only does this speed up the time it
81 takes to build one tree, but the iterative data sampling process was deemed unnecessary
82 when ordered feature splitting was used. Additionally, two different visualizations for
83 two different end-users with regards for their characteristics and needs is suggested.
84 Following the main characteristics of XAI-methods, LMTs is a post-hoc, model-agnostic
85 explanation method giving local explanations in the form of feature attributions. Even
86 though the feature attributions used to form the local explanations in this work, it should
87 be noted that instead of creating an explanation model for specific data points, the LMT
88 approximates the full model across its whole range of validity. So, the explanations pro-
89 vided by the LMT are local, but since decision trees (DTs) are considered interpretable,

Table 1: Main characteristics of XAI-methods

Scope of explanation	Local/ Global	The scope of the explanations range from local explanations, where only one instance is explained, to global, where the entire model is explained. This is not a binary categories, as for example groups of similar instances can be explained at the same time.
Complexity of model to be explained	Intrinsic/ Post-hoc	Models that are self-explanatory, such as simple linear regression, are called intrinsically explainable models. More complex methods, such as most DNNs or other models considered to be black-boxes however, cannot be easily understood by humans, so a post-hocXAI-method must be applied to the model toaid with the understanding of it.
Applicability of XAI-method	Model-agnostic/ Model-specific	A model-agnostic XAI-method treats the model to be explained as a black-box, that is the XAI-method only cares about the inputs and outputs of the model to be explained. Thus, it can be applied to any model. A model-specific XAI-method, as the name implies, can only be applied to one specific model.

in theory the LMT also yields a global explanation of the full model. However, note that for all practical means, the global interpretability of a DT is reduced quickly as the size of the tree increases. The LMT is intended to run in real-time, parallel to the full model, to provide explanations in the form of feature attributions for its predictions. The ability of LMTs to run real-time combined with their inherent transparency are the two main benefits of using them to explain black-box models used in robotic applications such as docking. The terms used in relation to LMTs and DRL are described in Table 2.

Our main contributions are the following:

- An improved and faster building process of LMTs from [27] by introducing re-ordering to the splitting sequence of the input features, to better match the way guidance systems work. This made the iterative data sampling process slowing down the building process from [27] unnecessary.
- An overview of the background knowledge, skills, needs, and requirements the different end-users of the docking agents have.
- Two different visualizations of the explanations based on the characteristics of each end-user.

2. Preliminaries

In this section, the DRL agent as well as the training environment used for its development are presented. For more details, the reader is referred to [17].

2.1. The ASV docking problem

Docking is the process of taking a vessel from being in open waters to being fastened to a specific location along the quay, referred to as the *berthing point*. The process can be divided into the following three stages:

1. The approach phase.
2. The berthing phase.

Table 2: Description of terms used in relation with the LMT and DRL.

LMT	DRL	Description
Input features	States	Information the model is trained and later used to predict on, in this case a description of the environment as provided to the policy and the policy approximator, given in Equation (2).
Predictions	Actions	The model output, given in Equation (1)
Policy approximator	Policy	The model itself, providing a mapping from input features to predictions or states to actions, respectively. The policy corresponds to the <i>controller</i> in robotics, while the policy approximator is in this case only used to generate explanations
Explainer	Agent	The application of the model. In the current setting, the agent comprises the policy and the vessel, while the explainer is produced using the LMT to generate feature attributions and visualisations

115 3. The mooring phase.

116 During the approach phase, the vessel moves from open seas to confined waters. In
 117 the berthing phase, the vessel maneuvers inside confined waters until it is parked at a
 118 location close to the berthing point. In the mooring phase, the vessel is fastened to the
 119 berthing point. Docking is considered to be a challenging task since it requires complex
 120 decision making and significant fine-tuning of actions. In addition to being difficult to
 121 model, external disturbances affect the vessel more at low speeds than at high speeds.
 122 Thus, their impact on the movement of the vessel increases when the vessel operates at
 123 low speeds close to obstacles. The simulation environment used in this work is based on
 124 Trondheim harbor, and is the same as the one used in [17]. Figure 1a) shows a snapshot
 125 of the simulation environment. An illustration of the vessel is shown in Figure 1b). The
 126 vessel has three thrusters: a tunnel thruster in the front and two azimuth thrusters at the
 127 back. The vessel is controlled using the control inputs

$$128 \quad \mathbf{A} = [f_1, f_2, f_3, \alpha_1, \alpha_2], \quad (1)$$

129 where f_1, f_2 are restricted to the range $[-70 \text{ kN}, 100 \text{ kN}]$ and α_1, α_2 to the range
 130 $[-90 \text{ degrees}, 90 \text{ degrees}]$ represent the force and the angle of the two azimuth thrusters,
 131 respectively. The tunnel thruster is controlled by changing its force, f_3 , in the range
 132 $[-50 \text{ kN}, 50 \text{ kN}]$. The features representing the vessel's state and relative position in the
 133 environment form the vector

$$134 \quad \mathbf{x} = [\tilde{x}, \tilde{y}, \tilde{\psi}, u, v, r, l, d_{obs}, \tilde{\psi}_{obs}]. \quad (2)$$

135 Here, \tilde{x} and \tilde{y} represent the relative distance to the berthing point in the vessel's body
 136 frame, in which u, v, r represent the vessel's velocity. $\tilde{\psi}$ represents the difference between
 137 the actual heading and the heading desired at the berthing point. Note that since \tilde{x}
 138 and \tilde{y} are in body frame, they are only aligned with the environment axis if $\tilde{\psi}$ is zero
 139 and aligned with the environment frames. That is, \tilde{x} is not necessarily in the south-
 140 north direction, and \tilde{y} is not necessarily in the west-east direction. The binary variable
 141 l indicates whether or not the vessel has made contact (i.e. collided) with an obstacle,
 142 which, as discussed in Section 2.2, is mainly used during training. The variables d_{obs} and
 143 $\tilde{\psi}_{obs}$ represent the relative position to the closest obstacle in body frame. A restriction
 144 in the simulated environment is that the agent is not allowed to make any contact with
 145 the harbour under any circumstances, although gentle contact with the harbour under

low speeds while berthing is usually allowed in real life. The binary variable l is only used during training of the DRL-policy through giving a large penalty and ending the episode.

2.2. The docking agent

As previously discussed, the problem of docking is challenging due to several reasons, one being that it is hard to achieve adequate mathematical models of all the aspects affecting the operation, which is crucial of most traditional control methods. In [17], an RL-agent learned how to perform berthing from substantial distances – up to 400 meters from the berthing point, corresponding to LP4:Distance berthing in [17] – without using any additional models of the environment or vessel. RL is the branch of machine learning dedicated to learning by interacting with the environment and receiving rewards for different states based on their desirability. The reward function is chosen or designed by the programmer, and is crucial for the learning process of the agent. Extensive work was done in engineering a fitting reward function for the task in this paper, and the objectives for the RL-agent are the following:

1. Avoiding any obstacles, specifically keeping $d_{obs} > 0$.
2. Reaching and staying at the berthing point, specifically achieve a stable situation with $\tilde{x} = \tilde{y} = \tilde{\psi} = 0$.

Note that COLREG(Convention on the International Regulations for Preventing Collisions at Sea)¹ is not taken into consideration. These objectives are given to the RL-agent through the following reward function

$$R(\tilde{x}, \tilde{y}, l, d_{obs}) = R_d + R_{\tilde{\psi}} + R_{obs} + R_{\tilde{d}}. \quad (3)$$

Here, R_d rewards the agent for minimizing the distance to the berthing point. Given that the distance to the berthing point is small enough, rewards for achieving the desired heading is given through $R_{\tilde{\psi}}$. The agent was given significant penalties for getting close to, and especially, making contact with any obstacles through R_{obs} since this is of high priority. However, this made the agent hesitant to get close to the berthing point since this is very close to the harbour, which in the agent's point of view is an obstacle. Therefore, the reward component $R_{\tilde{d}}$, which rewards decreasing the distance to the berthing point, was designed.

To train the DRL-agent, the proximal policy optimization (PPO) algorithm from [28] was used. It is a *stochastic, on policy* algorithm that uses a trust region to prevent too large updates to the policy based on a training batch, which can lead to getting trapped in a local minimum. The trust region is the area in which the approximation of the gradient descent of the policy is accurate. To prevent the training to become too constrained to

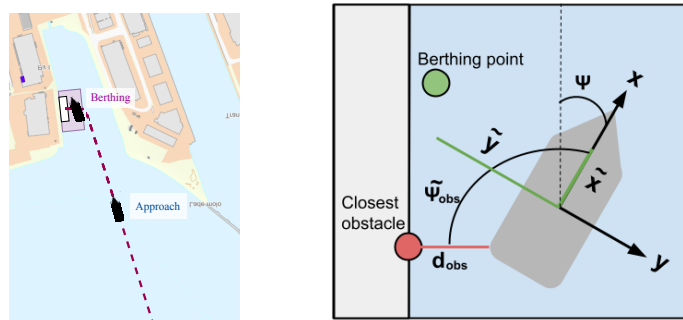


Figure 1. a) The simulation environment, and b) an illustration of the vessel's states.

¹ <https://www.imo.org/en/About/Conventions/Pages/COLREG.aspx>

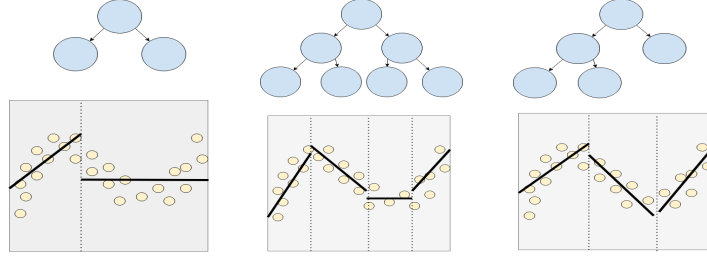


Figure 2. Illustrations of how two perfect LMTs of depth **a)** two, **b)** three, and **c)** an imperfect LMT of depth three, fit to the same data set. The number of regions estimated by a linear function correspond to the number of leaf nodes.

181 this trust region, the trust region does not set hard boundaries for the exploration, but
 182 is rather included in the objective by giving penalties to the updates that encourage
 183 not leaving the trust region. The resulting PPO-trained neural network has two hidden
 184 layers, consisting of 400 neurons each. The hidden layer's nodes use the rectified
 185 linear units (ReLU) activation function [29], while the output layer uses the hyperbolic
 186 tangent function, restricting the outputs to the range $[-1, 1]$. The agent converged after
 187 approximately 6 million interactions, i.e. cycles of having a state, performing an action
 188 and receiving a reward.

189 3. Linear Model Trees

190 Decision trees (DTs) form a class of machine learning algorithms based on condi-
 191 tional control statements, and are capable of solving many classification and regression
 192 problems. Their main advantages are being both easily visualized and interpretable for
 193 humans. A DT consists of branch and leaf nodes, where the branch nodes perform data
 194 splitting based on the control statements, and the leaf nodes perform the DT model's
 195 prediction. In its simplest form, a DT has univariate splits, i.e. it splits based on only
 196 one feature at a time, and each leaf node has a constant prediction. *Oblique DTs* have
 197 multivariate splits in the branch nodes, making them less interpretable and significantly
 198 increasing their building time, due to which they are not used in this work. *Model trees*
 199 are DTs where the constant predictions in the leaf nodes are replaced by a prediction
 200 model, for example a linear regression model or a DNN, so that the tree maps the input to
 201 the appropriate model. The simplest version of model trees are *linear model trees (LMTs)*,
 202 which have a linear function in the leaf nodes. As illustrated in Figure 2, an LMT makes
 203 out a piecewise linear function, and the number of regions resulting from the splits of
 204 the tree correspond to the number of leaf nodes.

205 The problem of building an LMT for a data set (\mathbf{X}, \mathbf{Y}) can be expressed as

$$206 \min_{a, t, w} = \sum_{\forall (x, y) \in (X, Y)} (y - f(x))^2, \quad (4)$$

207 where $f(x)$ is the prediction made by the LMT, which we express as follows

$$208 f(x) = \sum_{l \in \forall \text{ leaf nodes}} f_l(x) \prod_{n \in l^a} a_n^T x < t_n \prod_{n \in l^r} a_n^T x \geq t_n. \quad (5)$$

209 Here, a_n is a standard basis vector in the chosen coordinate basis, which is in our case
 210 that of the vessel, while t_n is the *threshold* value upon which node n is split. A leaf node
 211 l 's ascendants are its left and right ascendants, l^a and l^r . The linear function f_l in leaf
 212 node l , is given by

$$213 f_l(x) = \sum_{f=1}^F (w_f x_f) + w_{F+1}, \quad (6)$$

214 where F is the number of input features, i.e. states.

215 It is sometimes stated that DTs are fully interpretable models, but this is an oversim-
 216 plification for most practical means. As outlined in [30], transparency can be understood
 217 as simulatable, decomposable, or algorithmic transparent. To be *simulatable transparent*,
 218 the model as a whole must be simple enough that a human can easily interpret it. This
 219 also implies that both the input features and the predictions must be easily understand-
 220 able. Provided that the inputs and outputs are understandable, and that the trees have
 221 a reasonable size, both DTs and LMTs are simulatable transparent. To be *decomposable*
 222 *transparent*, all parts of the model must be simulatable transparent. This means that
 223 an LMT that is too big to be simulatable transparent, is still decomposable transparent,
 224 since all its parts, i.e. its subtrees, are still simulatable transparent. Finally, *algorithmic*
 225 *transparent* methods are those that can be analyzed using mathematical tools. Thus,
 226 LMTs are always decomposable and algorithmic transparent, and whether they are also
 227 simulatable transparent depends on the size of the specific tree.

228 3.1. Heuristic tree building

229 The LMTs used here and presented in [27] are constructed using Algorithm 1. Since
 230 building an optimal DT given a data set \mathcal{D} is an NP-hard problem, our approach is
 231 heuristic, which is common for most approaches to building DTs (see e.g. CART [31],
 232 ID3 [32], and C4.5 [33]).

Algorithm 1: The LMT algorithm from [27].

```

Require:
  Training data  $\mathcal{D}$ 
  Maximum number of leaf nodes  $N$ 
  Minimum number of data samples for leaf nodes  $M$ 
while number of leaf nodes is less than  $N$  do
  | if there exist a node that fulfills all splitting criteria then
  | | Choose node to split
  | | Perform splitting
  | | Calculate best potential split for the newly created nodes
  | else
  | | return root node
  | end
end

```

233 A so-called *perfect* DT is a tree with binary splits where all the leaf nodes have the
 234 same depth. The trees in Figure 2a), Figure 2b) are examples of perfect DTs. However,
 235 as pointed out in [34], perfect DTs are often unnecessarily big. Consider the docking
 236 problem, the complexity of the agent's behaviour will vary in different parts of the
 237 harbor and with different positions and velocities. For example, it is expected that the
 238 maneuvers required close to the berthing point will be more intricate than at open seas.
 239 If the DT is to be perfect, it will either not be deep enough to approximate the behavior
 240 close to the berthing point, or it will overfit to the behavior for open seas. For the same
 241 reason, the stopping criteria was changed from maximum depth to maximum number
 242 of leaf nodes, which allows the DT to grow deeper in areas that require more splits,
 243 resulting in an imperfect tree. Figure 2b) and Figure 2c) illustrates the difference between
 244 an imperfect DT and a perfect DT. One way of searching for splitting conditions for
 245 a node is to order the values for each feature, and try threshold values in the middle
 246 between two neighboring feature values. However, for large data sets, this procedure is
 247 very computationally expensive. Therefore, a search grid evenly distributed from the

248 lowest to the highest feature values is used to find the split thresholds. The splitting
249 condition for a node is found via

$$250 \quad \mathcal{F}, t_n = \underset{\mathcal{F}, n}{\operatorname{argmin}}(\operatorname{loss}(\mathcal{D}_L) + \operatorname{loss}(\mathcal{D}_R)), \quad (7)$$

251 where \mathcal{F} and t_n are the feature and threshold, respectively, the node is to split upon,
252 given the data samples in \mathcal{D} . That is, the non-zero entry of the basis vector a_n for node n
253 in Equation (5) corresponds to \mathcal{F} . The data sets \mathcal{D}_L and \mathcal{D}_R are subsets of \mathcal{D} , and result
254 from the split of a node. Each branch node splits the data it receives into a left and right
255 part, so all data points end up in exactly one leaf node. Each node splits the data it
256 receives according to

$$257 \quad \begin{aligned} \mathcal{D}_L &= x \in \mathcal{D} \quad \text{if} \quad x_{\mathcal{F}} \leq t_n, \\ \mathcal{D}_R &= x \in \mathcal{D} \quad \text{if} \quad x_{\mathcal{F}} > t_n, \end{aligned} \quad (8)$$

258 where $x_{\mathcal{F}}$ is the data sample x 's value for feature \mathcal{F} . Since not all possible thresholds are
259 explored, and there is no guarantee for global optimality since this method is greedy,
260 there is no need for the algorithm to be deterministic and yield exactly the same tree
261 in each run. Instead, having the process include some randomness leads to a wider
262 exploration in the same runtime, if run in parallel. The n 'th threshold is

$$263 \quad t_n = \min(\mathcal{D}^{\mathcal{F}}) + (n+r) \frac{(\max(\mathcal{D}^{\mathcal{F}}) - \min(\mathcal{D}^{\mathcal{F}}))}{N}, \quad (9)$$

264 where $\mathcal{D}^{\mathcal{F}}$ are all the values of feature \mathcal{F} in the data set \mathcal{D} , N is the number of thresholds
265 in the grid search, and r is a random number that alters the threshold value in the range
266 $\pm 2\%$. The next node n_s to split is chosen using

$$267 \quad n_s = \underset{n}{\operatorname{argmax}}((1+r)(\operatorname{loss}(\mathcal{D}_L^n) + \operatorname{loss}(\mathcal{D}_R^n))), \quad (10)$$

268 where \mathcal{D}_L^n and \mathcal{D}_R^n are the losses of the left and right child nodes, respectively, of node
269 n , given its best split variables \mathcal{F} and t_n . The linear functions showed in Equation (6)
270 in the leaf nodes are calculated by performing ordinary least squares regression on the
271 data \mathcal{D}_l belonging to leaf node l .

272 As our aim is for the LMT to be a faithful explanation model for the DRL model,
273 the loss in Equation (10) is calculated as the mean squared error (MSE) between the
274 prediction of the DRL model and that of the linear function fitted by linear regression in
275 the leaf nodes.

276 When tested, Algorithm 1 turned out be very sensitive to the data set \mathcal{D} , which is a
277 well-known problem for DTs. How many data points are needed to represent an area
278 properly, depends on how complex the DRL model is in that area.

279 To mitigate this, we performed the data sampling and tree building iteratively,
280 according to the following algorithm

Algorithm 2: The data sampling process from [27].

Require:
Maximum number of iterations Max_it
Maximum number of leaf nodes N
Minimum number of data samples for leaf nodes M
 $it = 0$
while *number of iterations it is less than* Max_it **do**
 $\mathcal{D}_{it+1} \leftarrow \text{sample_from_environment}(LMT_{it})$
 $LMT_{it} \leftarrow \text{Algorithm 1}(\mathcal{D}_{it}, \mathbf{M}, \mathbf{N})$
 $it++$
end

281 This process was repeated for ten iterations, before checking which resulting LMT
282 performed best on an independent validation set. The best chosen LMT tree was built
283 on the ninth iteration.

284 In [27], an imperfect LMT with a total of 681 leaf nodes was trained to approximate
285 and serve as explanation model for the DRL-model presented in Section 2.2. The tree
286 model is inarguably too large to be considered simulatable transparent, but it can still
287 be used to map the input features, i.e., the states, to the predictions, i.e., the actions.
288 Furthermore, sub-parts of the tree are still considered simulatable transparent. The
289 maximum depth of the tree was 15, while the shallowest leaf node was at depth 5.

290 As the vessel has five control inputs, the DRL model has five outputs, and so must
291 the LMT. This can be achieved either by building one LMT for each control input, or by
292 building one LMT for all the control inputs. In the latter case, every leaf node contains a
293 fitted linear function for each of the control inputs, and the average loss is used when
294 fitting and evaluating the splits. Consequently, this approach requires the control inputs,
295 respectively the LMT outputs, to be normalized. The latter approach was used both
296 in [27] and the present work, because it is challenging to understand, and because it is
297 much more time demanding to build five trees instead of just one.

298 3.2. Building linear model trees utilizing ordered feature splitting

299 Although the LMT used in [27] did show promising results, there are two important
300 drawbacks. Primarily, the process of building it is slow because several trees must be
301 built, and data sampling has to be done for several iterations. This leads to a larger data
302 set, which again increases the time it takes to build an LMT each iteration. Secondly, the
303 resulting tree is very large and thus, as mentioned, in no way simulatable transparent,
304 which is a significant drawback since the LMT is used as an explanatory model.

305 To address this, we set the order in which the LMT building process searches for
306 feature splits. This is done by letting the LMT search for splits on the following features,
307 and in the following order:

- 308 1. $\tilde{x}, \tilde{y}, \tilde{\psi}$
- 309 2. $d_{obs}, \tilde{\psi}_{obs}$
- 310 3. u, v, r .

311 As mentioned, the binary variable l is only for penalizing the DRL-agent during training
312 and ending the episode, and it will therefore not be used for the LMTs. The order is
313 set to better match guidance system logic, however, a more systematically approach
314 remains future work. During training, the criteria for a split to be valid is that the overall
315 loss decrease, and that the child node receive a minimum number of data samples, here
316 M . Once these criteria are met, the node is split. If the criteria aren't met after trying all
317 features in the three feature groups, the tree stops growing.

318 As expected, this approach to searching the features for splitting decreases the
319 time needed to train the LMT, since the number of feature and threshold pairs are
320 reduced in the split search. Additionally, with ordered feature splits the iterative data

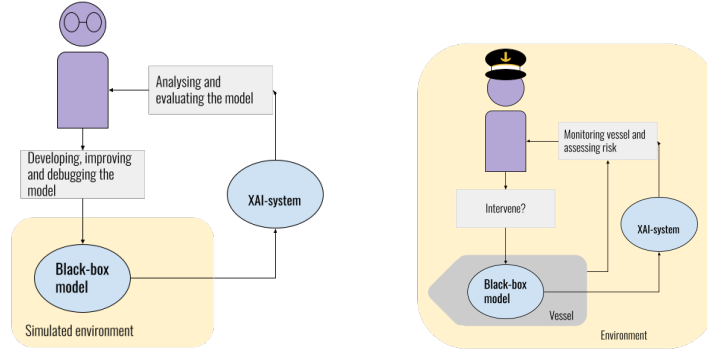


Figure 3. Illustration of **a)** the developer's and **b)** the operator/seafarer's different relations to the agent and the environment.

321 sampling process shown in Algorithm 2 was deemed unnecessary. More importantly,
 322 the resulting tree is more interpretable for humans since similar features are close to each
 323 other, making it easier to locate which parts of the tree are relevant in different situations.
 324 Furthermore, the resulting LMT is sufficiently small to be simulatable transparent.

325 4. Increasing model interpretability using linear model trees

326 The goal of approximating the DRL model with an LMT is to use the inherent
 327 transparency of the LMT and its intuitive structure to efficiently obtain an importance
 328 ranking of the input features, i.e. the states of the vessel. However, as previously
 329 emphasized, although DTs, and consequently LMTs, are transparent, this does not
 330 necessarily make them easily understandable for humans. In this section, we first
 331 discuss how the linear functions in the leaf nodes can be used to obtain explanations for
 332 a prediction in the form of feature attributions. Next, we demonstrate how these feature
 333 attributions can be visualized together with the environment as well as the states and
 334 actions of the vessel to obtain a more comprehensible picture.

335 4.1. Extracting feature attributions from the leaf nodes

336 LMTs can give local explanations in the form of feature attributions, which can be
 337 seen as giving credit or blame to the input features for the output, in essence feature
 338 attributions are answering the question "how much did each input feature affect the
 339 model's output?". The local explanations are calculated utilizing the coefficient in the
 340 linear function in the leaf nodes and the values of the instance to be explained. The
 341 linear functions in the leaf nodes take the form of Equation (6), and the importance of a
 342 feature \mathcal{F} is

$$343 I_{\mathcal{F}} = \frac{w_{\mathcal{F}}x_{\mathcal{F}}}{\sum_{f \in \mathcal{V}_{\mathcal{F}}} |w_f x_f|}, \quad (11)$$

344 where $w_{\mathcal{F}}$ is the coefficient of the linear function in Equation (5) of the leaf node making
 345 the prediction, and $x_{\mathcal{F}}$ the value of the sample for feature \mathcal{F} . Note that the constant
 346 coefficient $w_{\mathcal{F}} + 1$ from Equation (5) is not included in Equation (11), which means that
 347 if the linear function in a leaf node is a constant function no feature attributions can be
 348 calculated. Additionally, it should be noted that when forming these local explanations,
 349 only the function in the leaf node is taken into consideration, even though the path
 350 from root node to this leaf node is not irrelevant and most likely should be considered.
 351 However, including the paths in (both local and global) explanations should not be done
 352 carelessly since even irreducible DTs can have irrelevant splits [35].

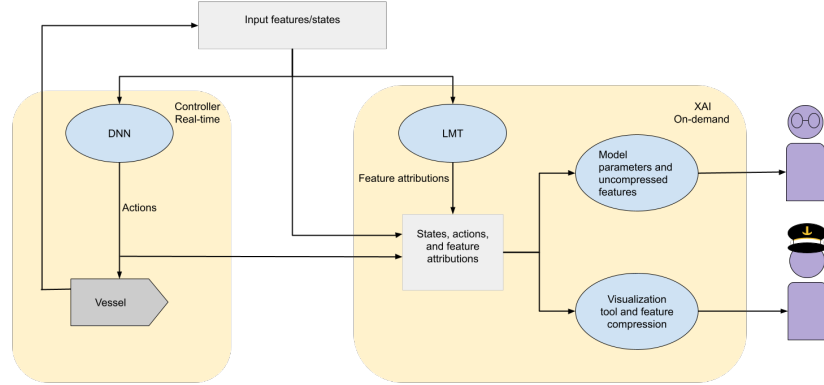


Figure 4. Pipeline after both the LMT and DNN are trained.

353 4.2. Visualization of feature attributions

354 Different users require both different explanations and different representations of
 355 the explanations, states, and actions. For this work two users of the black-box model is
 356 considered, namely the *developer* and the *seafarer/operator*. The developer wants to use the
 357 XAI-method to verify that the black-box model works as intended, detect edge cases or
 358 erroneous behaviour to improve the model, as well as understanding of how the model
 359 behaves. On the other hand, the seafarer/operator uses the XAI-method as a supporting
 360 tool to monitor and control the autonomous agent's behaviour to assess whether or
 361 not they should intervene to prevent a dangerous situation or accident. An important
 362 difference is that the operator/seafarer has a personal risks associated with erroneous
 363 behaviour of the model, whereas the developer has not. The different relation to the
 364 black-box model, the environment, and the XAI-system for the two users is shown in
 365 Figure 3. Where the developer can carefully inspect the models behavior in a simulated
 366 environment with no time pressure, the seafarer/operator must make assessments within
 367 a short time span with risk of serious consequences for vessel, crew, and equipment.
 368 Additionally, the seafarer/operator has a lot of other sources of information, both from
 369 other sensors and displays, but also from their own senses. The main differences that
 370 needs to be taken into account when deciding how to convey the explanations to the
 371 specific user are outlined in Table 3.

372 To aid the developer in thoroughly investigate the step-by-step state-action pairs
 373 with their corresponding feature attributions the plots in Figure 5 are suggested. In
 374 Figure 5a), the feature attributions are plotted for each step. The feature attributions
 375 should be studied together with the state and action plots of Figure 5b) and Figure 5c).
 376 Figure 5 contains a lot of information that requires a lot of time to analyse, so this type of
 377 visualization cannot be used real-time. For a user like the operator/seafarer, another type
 378 of representation of this information is needed. One aspect that makes it hard for humans,
 379 and even domain experts such as operators/seafarers, to process the information is the
 380 fact that the vessel has 9 state features and 5 control inputs. Additionally, f_1 and α_1 , and
 381 f_2 and α_2 are controlling the same motors, and it is not possible to understand the agent's
 382 behavior as a whole while looking at cooperating actions independently. To remedy this,
 383 the actions are mapped to and visualized on the vessel for faster comprehension as can
 384 be seen Figure 6. Additionally, feature attributions for the 5 actions are combined as
 385 follows

$$386 I^{\mathcal{F}} = \sum_{a \in \mathcal{A}} |I_a^{\mathcal{F}}| \quad (12)$$

Table 3: Differences between developer and operator/seafarer

	Developer	Operator/Seafarer
Background knowledge	Good analytical skills, but not necessarily domain knowledge	Domain knowledge, but not necessarily good analytical skills
Environment	Works in simulated environments or digital twins without risk of physical damage	Works with the physical vessel, with risks for material damage and potentially personnel injury
Risk	Works with a risk-free simulated environment	Works in a physical environment where errors can compromise safety of units involved
Urgency	Analyses the model offline with no time pressure	Monitors the controller via the XAI module real-time under time pressure
Tools	Has access to analytical and mathematical tools	Has no analytical or mathematical tools available
Information design	Prefers information enabling thorough and analytic investigation of the controller's behaviour	Prefers information suitable for fast processing, and related to the vessel
Level of detail	Desires high level of detail, has low risk of cognitive overload as information originates from one source only and the working environment is stress-free	Only interested in the necessary information, having several sources of information and a potentially stressful working environment, creating a risk for cognitive overload
Event frequency	Interested in examining the controller's behaviour over the entire state space	Not interested in experiencing states that might lead to undesired behaviour or dangerous situations
Edge cases	Uses edge cases to detect undesirable or unexpected behavior	Does not wish to experience edge cases that involve higher risk of faulty controller behavior
Intervention	Does not intervene if undesirable or unexpected behavior is discovered	Intervenes if entering or experiencing state that lead to undesired behavior to avoid dangerous situations

Table 4: Feature compression

Compressed features	Features	Compressed feature importance
Distance	\tilde{x}, \tilde{y}	$I^D = I^{\tilde{x}} + I^{\tilde{y}}$
Heading	$\tilde{\psi}$	$I^H = I^{\tilde{\psi}}$
Obstacle	$d_{obs}, \tilde{\psi}_{obs}$	$I^O = I^{d_{obs}} + I^{\tilde{\psi}_{obs}}$
Velocity	u, v, r	$I^V = I^u + I^v + I^r$

Table 5: Overview of the structures of the different LMTs. LMT+ OFS denotes LMTs where ordered feature splitting was utilized, while the number denotes the total number of leaf nodes the LMT has.

Name of LMT	Number of leaf nodes	Depth of deepest node(s)	Depth of shallowest node(s)
LMT 467	467	16	3
LMT OFS 100	100	11	3
LMT OFS 312	312	12	3

Table 6: Run time for the different algorithms for trees of different sizes.

Algorithm	Build time for 10 leaf nodes	Build time for 50 leaf nodes
LMT	74.75 s	171.45 s
LMT+ OFS	52.748 s	117.91 s

387 where $I^{\mathcal{F}}$ is the overall importance for the feature \mathcal{F} . Still, having to consider feature
388 attributions for 9 features is too much to take into consideration in a stressful environ-
389 ment with time pressure, so the feature attributions are further compressed as shown
390 in Table 4. It is important to note that the feature importance is not confused with the
391 actual values of the features. A high importance for the velocity does not mean that the
392 vessel has a high velocity, it just means that the velocity played an important part when
393 the the action was predicted. The pipeline between the DRL-agent, the LMT and the
394 visualization tools, and the end-users after the DNN is trained and the LMT is built is
395 shown in Figure 4.

396 5. Results

397 To create the data set to build the LMTs 1000 unique starting points was found,
398 whereas 800 of these were used as starting points for the training set, 50 for the validation
399 set, and the remaining 150 for the test set. The complete data sets consisted of data from
400 runs performed by the RL-agent with these starting points. In this chapter the LMT
401 process presented in Section 3.1 and the LMT process utilizing ordered feature splitting
402 presented in Section 3.2 will be evaluated and compared.

403 5.1. Structure of linear model trees

404 There is an important difference between building *the optimal tree given a data set*,
405 and building *the optimal tree given a specific structure of the tree and a given data set*. In this
406 work, we take on the problem of building an LMT given a maximum number of leaf
407 nodes, univariate, binary splits, and a given data set. In Table 5, the structures of the two
408 best LMTs built using ordered feature splitting and one LMT built by the purely greedy
409 approach. Utilizing ordered feature splitting resulted in slightly smaller trees than when
410 building LMTs without utilizing ordered feature splitting.

411 5.2. Computational complexity

412 To compare the computational complexity of building LMTs with the algorithm
413 presented in Section 3.1 and the version that limit the number of features considered
414 at each split as presented in Section 3.2 the time it takes to build trees with 10 and 50
415 leaf nodes is compared. The different run times are presented in Table 6. LMT OFS is
416 significant faster than LMT, and the difference (naturally) increases when the size of the
417 trees increases.

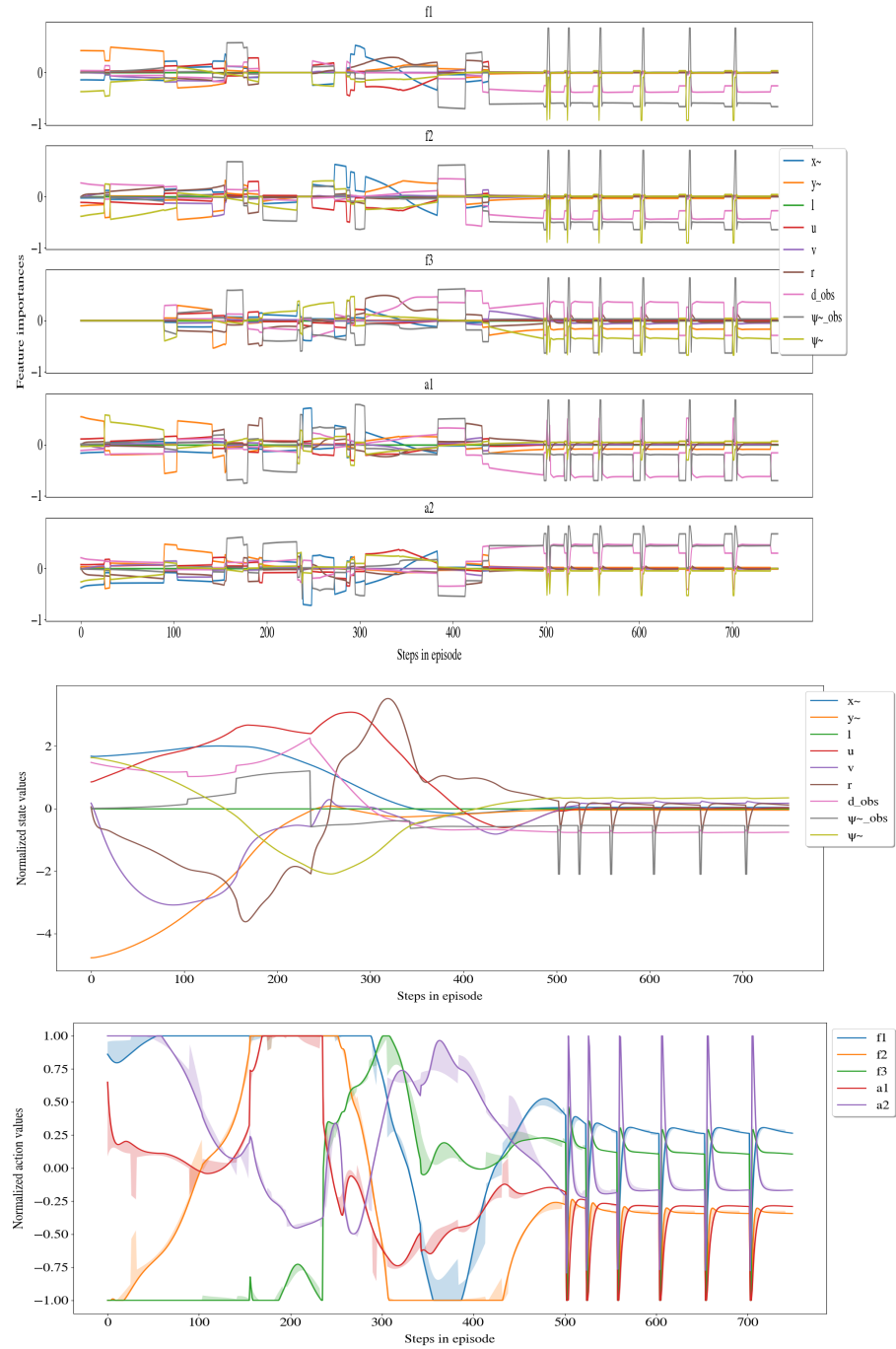


Figure 5. The visualization of the states, actions and feature attributions from one episode for the developer. The shaded area in the action-plot shows the difference between the actions taken by the DRL-agent and predicted by the LMT.

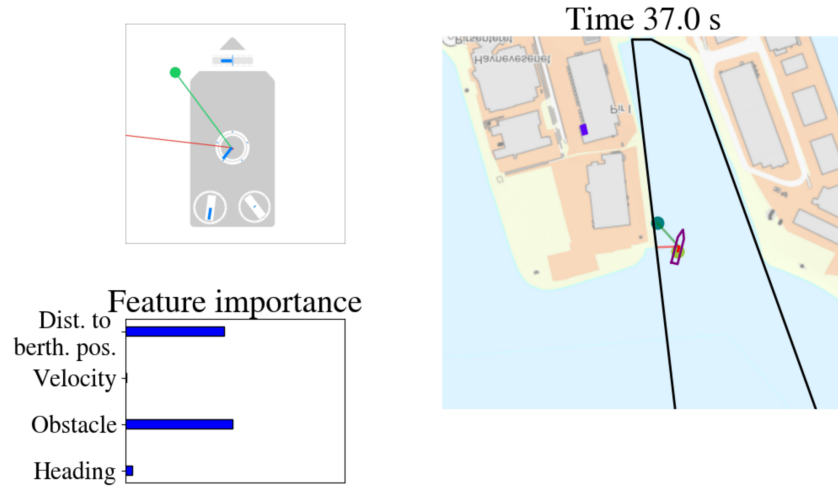


Figure 6. The visualization of the environment, vessel, and compressed feature attributions for the seafarer/operator. The compressed feature *Distance to berthing position* is shortened to *Dist. to berth. pos.*

418 5.3. Evaluating the fidelity

419 The most important aspect when choosing which tree to use is how well the tree
420 approximated the DRL-agent, i.e. the *fidelity*, which will be evaluated based on the
421 following metrics:

- 422 1. The average error between the DRL-agent's output and the tree's output given the
423 same input state as presented in Section 5.3.1.
- 424 2. The trees' path when running the vessel in the simulator compared to the path
425 taken by the DRL-agent as presented in Section 5.3.2.
- 426 3. The error between the resulting forces and moment based on the predicted actions
427 as presented in 5.3.3.

428 5.3.1. Output error

429 The mean absolute error and the standard deviation can be seen in Table 7. Both the
430 LMT OFS 100 and LMT OFS 312 has better accuracy and precision than LMT 467, despite
431 LMT 467 being significantly larger. LMT OFS 312 also has better accuracy and precision
432 than LMT OFS 100 on all actions. Additionally, using ordered feature splitting gave
433 consistently better results than without, and the building process became less sensitive
434 to the dataset.

435 5.3.2. Comparing the paths of the agent and of the linear model trees

436 If the LMT has approximated the PPO-policy well enough, the LMT should be able
437 to replicate the PPO-policy's behavior. Therefore, one way of evaluating how well the
438 LMT has approximated the PPO-policy is to compare the paths of their runs given the
439 same starting point. Plots for the four agents from four different starting points can be
440 seen in Figure 7 and Figure 8. Figure 8 shows a difficult scenario where the agent must
441 first steer the vessel backwards followed by straightening the yaw while simultaneously
442 controlling the surge and sway. Unlike Figure 7, Figure 8 does not require a turn, but
443 the path is close to the boundaries and deviations from this path will quickly lead to
444 the vessel making contact with the harbor limits. The DRL-agent's behavior is shown
445 in Figure 7a) and Figure 8a), the LMT OFS 100's behaviour in Figure 7b) and Figure
446 8b), the LMT OFS 312's behaviour in Figure 7c) and Figure 8c), and finally the LMT

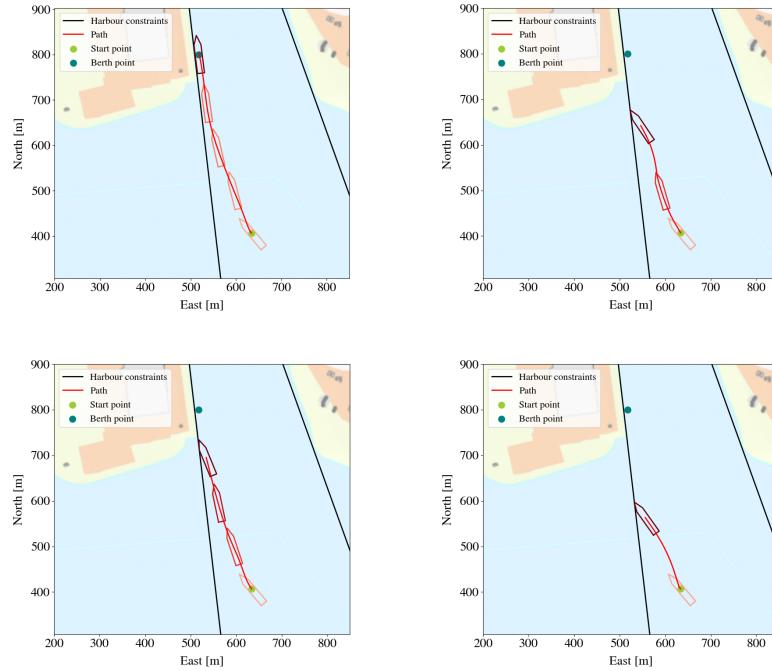


Figure 7. a) Successful run by the PPO-policy, b) failed run by the LMT OFS 100, c) failed run by the LMT OFS 312, and d) failed run by the LMT 467.

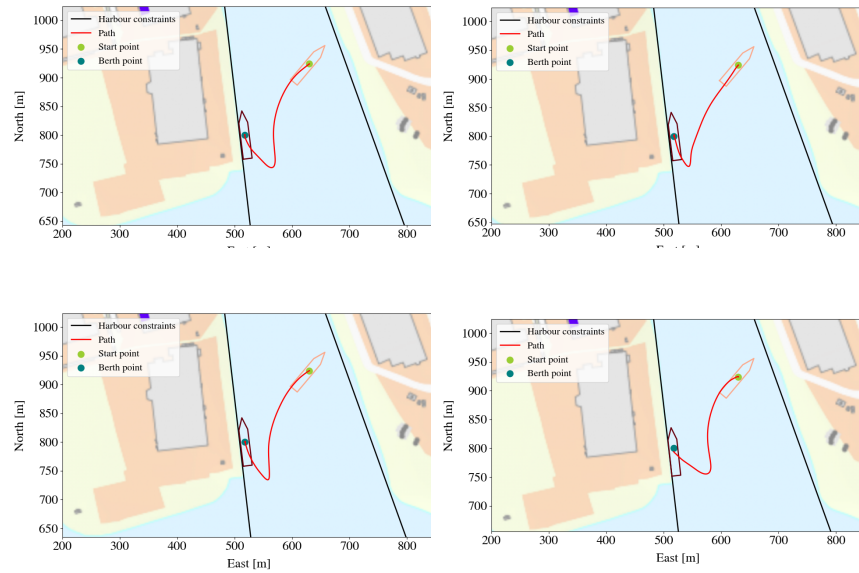


Figure 8. a) Successful run by the PPO-policy, b) failed run by the LMT OFS 100, c) failed run by the LMT OFS 312, and d) failed run by the LMT 467.

LMT OFS 100		
Output feature	Mean absolute error	Error standard deviation
f_1 (kN)	4.57 (2.68%)	9.31 (5.5%)
f_2 (kN)	4.018 (2.36%)	7.261 (4.2%)
α_1 (deg)	3.43 (1.9%)	7.66 (4.3%)
α_2 (deg)	4.81 (2.67%)	8.04 (4.4%)
f_3 (kN)	1.77 (1.77%)	4.049 (4.05%)
LMT OFS 312		
Output feature	Mean absolute error	Error standard deviation
f_1 (kN)	3.55 (2.08%) (-7,22%)	7.78 (4.57%) (-7,27%)
f_2 (kN)	3.33 (1.95%) (-6,35%)	7.085 (4.16%) (-5,675%)
α_1 (deg)	2.463 (1.36%) (-7,84%)	6.93 (3.85%) (-6,12%)
α_2 (deg)	3.66 (2.15%) (-5,48%)	8.03 (4.45%) (-3,42%)
f_3 (kN)	1.302 (1.3%) (-7,78%)	3.513 (3.51%) (-12,387%)
LMT 467		
Output feature	Mean absolute error	Error standard deviation
f_1 (kN)	11.85(6.97%)	19.07(11.22%)
f_2 (kN)	9.039(5.32%)	17.38(10.22%)
α_1 (deg)	7.9(4.3%)	14.32(7.96%)
α_2 (deg)	14.09(7.83%)	18.91(10.51%)
f_3 (kN)	3.84(3.84%)	6.83(6.83%)

Table 7: Output error analysis for the three different LMTs LMT OFS 100, LMT OFS 312, and LMT 467. The improvements from the LMT presented in [27] are highlighted in red

447 467's behaviour in Figure 7d) and Figure 8d). In the episode shown in Figure 7 it is clear
 448 that the LMT OFS 312 performs best out of the three. In the episode shown in Figure
 449 8 only LMT DK 100 and LMT OFS 312 completes the episode, while LMT 467 makes
 450 contact with the harbor limits while attempting the last part of the docking. LMT OFS
 451 312 mimics the behavior of the DRL-agent better than LMT OFS 100, as can be seen in
 452 Figure 8.

453 5.3.3. Comparison of resulting forces and moment on vessel

To further investigate the behavior of the policy and the LMT, we look at the forces acting on the vessel that result from the actions taken. This is because there are many combinations of actions that may result in the same overall forces. This also means that small deviations in each action may accumulate, causing the policy and the LMT to predict very different forces, despite their action predictions being similar. On the other

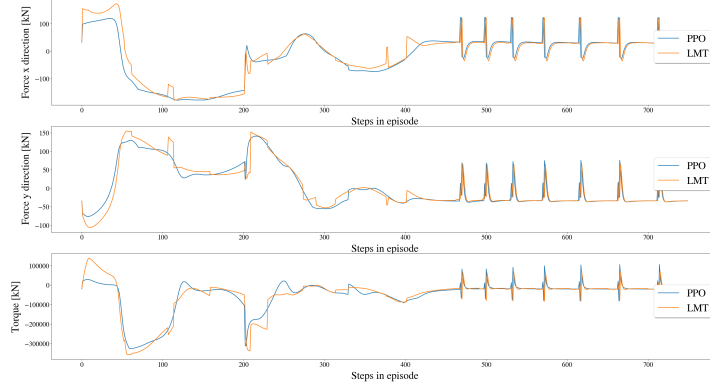


Figure 9. Plot of total force and moment predicted by both the LMT and the PPO-policy for an episode where the PPO-policy actions are given to the vessel.

hand, if the force of a thruster is zero then its angle does not matter, but it may still look like an important error. We calculate the overall forces predicted by the two models as

$$F_x = \sum_{i=1}^3 f_i \cos(\alpha_i), \quad (13)$$

$$F_y = \sum_{i=1}^3 f_i \sin(\alpha_i), \quad (14)$$

$$T = \sum_{i=1}^3 f_i (l_{i_x} \sin(\alpha_i) - l_{i_y} \cos(\alpha_i)), \quad (15)$$

454 where F_x denotes the applied force in the x -direction, F_y the applied force in the y -
 455 direction, and T the applied torque, all three in the body frame. The forces' arms of
 456 moment are given by l_{i_x} and l_{i_y} . Figures 9 and 10 show the forces and moments predicted
 457 by both the PPO-policy and the LMT. The LMT predictions do not follow the PPO-policy
 458 forces and moment perfectly, but the behaviour is very similar. Note that, as was also the
 459 case for the actions and feature attributions, the actions predicted by the LMT sometimes
 460 change abruptly. This happens when there is a change in which leaf node in the tree is
 461 being used to make the prediction.

462 5.4. Comparison of rewards

463 The LMT is trained without any knowledge of the reward function, whereas the
 464 DRL-agent's training relies heavily on it. However, it is expected that the LMT receives
 465 approximately the same rewards as the DRL-agent throughout an episode since they
 466 should behave similarly. In Figure 11, an episode where the LMT and PPO-policy
 467 behaves very similarly and their corresponding rewards can be seen. Since they have so
 468 similar paths their rewards are also similar, though with small deviations. The docking
 469 problem is a complex problem with many possible solutions, and thus, many different
 470 reward functions ought to lead to a viable solution. An example of two different paths
 471 successfully leading to the berthing point from the same starting point can be seen
 472 in Figure 12. Even though both the LMT and the PPO-policy successfully brings the
 473 vessel to the berthing point, the PPO-policy receives a higher cumulative reward. In
 474 cases like this, where the LMT ends up taking a different, but still viable, path than the
 475 PPO-policy does, the output error will be high. It might be interesting to evaluate the

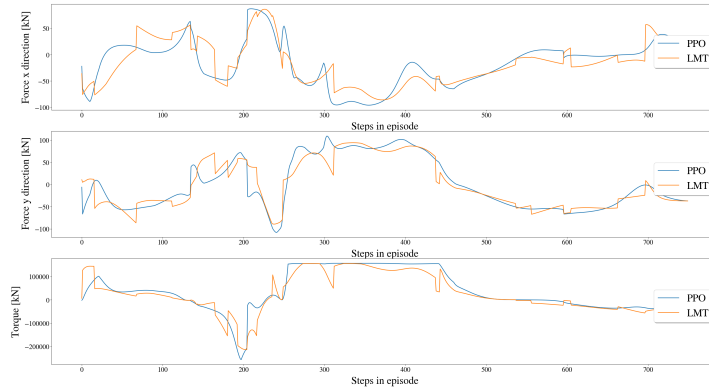


Figure 10. Plot of total force and moment predicted by both the LMT and the PPO-policy for an episode where the PPO-policy actions are given to the vessel.

476 LMT in the same way as the acPPO-policy, because if the LMT behaves as well as the
 477 PPO-policy, the LMT could replace the PPO-policy entirely, which would be beneficial
 478 because then we would be certain that the feature attributions would be completely
 479 correct. Nevertheless, as could be seen in Figure 7, the PPO-policy performs better than
 480 the LMTs.

481 6. Discussion

482 The main drawback with the building process outlined in Algorithm 1 is that it is
 483 a heuristic, greedy method. This means there that there is no guarantee of an optimal
 484 approximation of the black-box method, nor any guarantee of optimality given a dataset
 485 or a given tree structure. Introducing feature ordering to the splits improved the accuracy
 486 of the trees while decreasing their size, but still good splits can be hidden behind bad
 487 splits which will not be found due to the building process' greedy nature. This have
 488 been addressed by adding some randomness to the building process to further explore
 489 the solution space. Algorithm 1 can be sensitive to outliers in the dataset, since a larger
 490 range in the features' values will stretch out the thresholds' grid search. The linear
 491 regression may also be affected by outliers. For this reason, alongside the fact that the
 492 LMT cannot learn aspects of the black-box model's behaviour that is not represented
 493 in the dataset, it is important to have a good dataset. For this application, one tree
 494 with five linear functions in each leaf node was chosen due to the fact that building
 495 five trees is much more computational demanding than building one. As discussed,
 496 transparency for both DTs in general and LMTs depends on the size of the tree. If a small
 497 enough tree can be made with decent fidelity to the black-box model it's approximating,
 498 an assessment between accuracy and interpretability must be made. In this work, all
 499 the trees considered are too big to be categorized as simulatable transparent, thus only
 500 accuracy should be taken into account when choosing which tree should be the explainer
 501 model for the black-box model. In this work, the explanations come in the form of
 502 feature attributions which are calculated by using the linear function in the activated
 503 leaf node. This means that the splits along the path from the root node to the activated
 504 leaf node is not taken into account when forming the explanation, even though it clearly
 505 is important. Say that a leaf node gives out a constant prediction through the coefficient
 506 $w_F + 1$ from Equation (5), then the feature attributions will all be zero, and thus there
 507 are no explanations for this region. For this problem, the thruster's force and angle
 508 is controlled directly instead of having the vessel be controlled through a total force

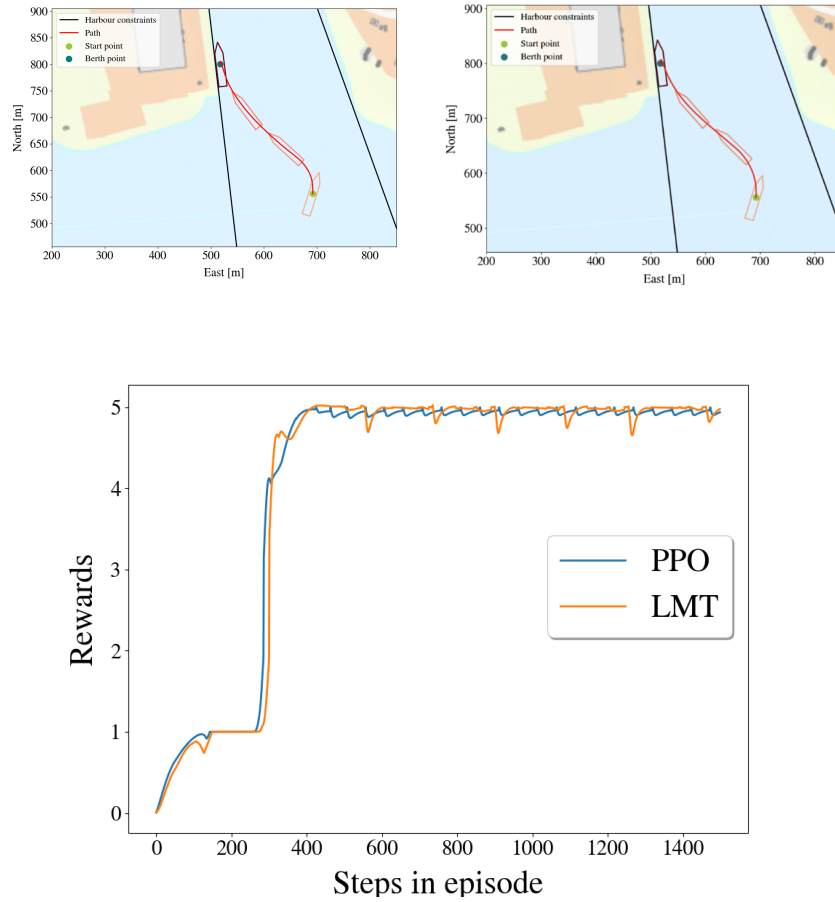


Figure 11. a) A successful run by the LMT, b) a successful run by the PPO-policy, and c) the LMT's and PPO-policy's rewards for their respective runs with the same starting points.

509 and torque applied to the vessel. This is desirable because the DRL-agent gets more
 510 freedom to learn new strategies, but it provides an additional challenge to the XAI-
 511 method because it gets less clear what the DRL-policy is attempting to do because there
 512 will be many combinations of forces and angles of the thrusters that equals the same total
 513 force and torque. Additionally, a_1 and f_1 controls the same azimuth thruster, and how
 514 each of these actions affect the vessel is heavily dependant on each other. For example, if
 515 f_1 gives no force, the angle, a_1 , of the thruster does not affect the vessel in any way. As
 516 pointed out by [36], interpretability is not a concept that is easily objectively measured,
 517 and how the explanation is communicated to the end user is of great importance to how
 518 well the model will be understood. Thus, the visualizations of the vessel's states, actions,
 519 and corresponding feature attributions should be evaluated by the users themselves in
 520 terms of how factors such as how efficient the information is communicated, and how
 521 they affect the users trust towards the system. Additionally, how the trees' structure can
 522 be used to form better explanations should be investigated, and a more systematically
 523 approach to the reordering of the features used in the splitting should be looked into.

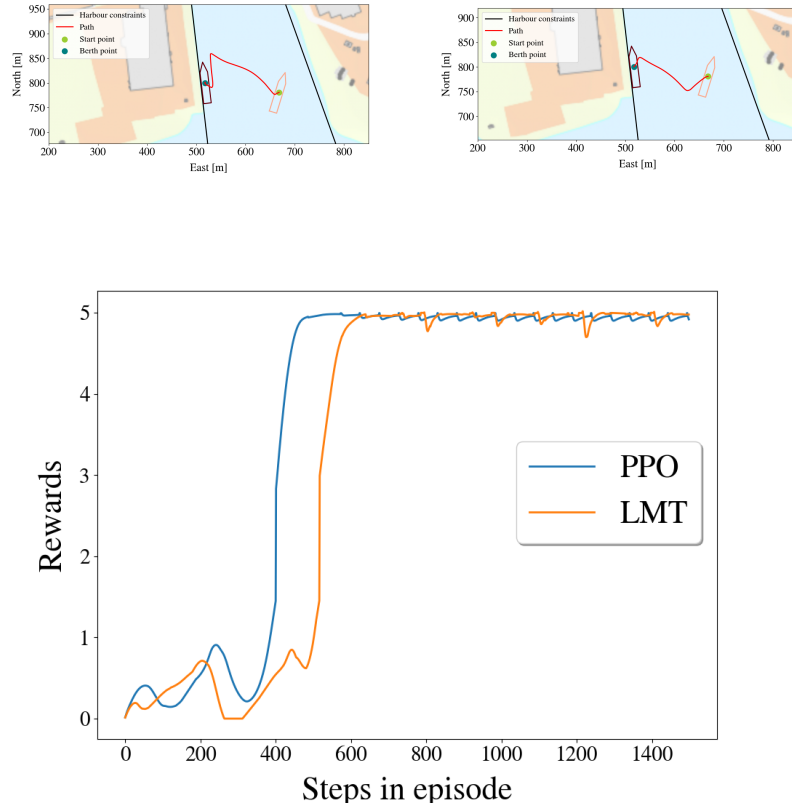


Figure 12. a) A successful run by the LMT, b) a successful run by the PPO-policy, and c) the LMT's and PPO-policy's rewards for their respective runs with the same starting points.

524 7. Conclusions

525 The need for XAI-methods for black-box models such as DNNs to be used for
 526 marine robotics in general, but also more specifically ASVs. In this work, the preliminary
 527 work from [27] was significantly extended through improving the algorithm, more
 528 thoroughly testing of the approximation, and better communication of the feature
 529 attributions through user adapted visualizations. The algorithm was improved by
 530 introducing ordered feature splitting to the trees, both in terms of more accurate trees
 531 and in faster building time. This makes the LMTs capable of tackling more complex
 532 problems with higher dimensions. Different users require different types of explanations,
 533 as well as different representation of both the information about the ASV and the
 534 explanation given by the LMTs. Therefore, two different visualizations were suggested
 535 for two different users, the developer and the seafarer/operator. The visualizations of
 536 the feature attributions do not serve as a full explanation of the model, but can be used
 537 as a step towards understanding, or at least trusting, the model.

538 **Acknowledgments:** This work was supported by the Research Council of Norway through the
 539 EXAIGON project, project number 304843.

540 **Conflicts of Interest:** The authors declare no conflict of interest.


1. References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* **2015**, *521*, 436–44. doi:10.1038/nature14539.
2. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep learning*; MIT press, 2016.
3. Sutton, R.; S., Barto, A.G. *Reinforcement learning: An introduction.*; MIT Press, 1998.
4. Mnih, V.; Kavukcuoglu, K.; Silver, D.; et al.. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. doi:https://doi.org/10.1038/nature14236.
5. Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G.; McGrew, B.; Mordatch, I. Emergent Tool Use From Multi-Agent Autocurricula, 2020, [arXiv:cs.LG/1909.07528].
6. Lillicrap, T.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *CoRR* **2016**, *abs/1509.02971*.
7. Singh, L.Y.; K. Hartikainen, C.F.; Levine, S. End-to-end robotic reinforcement learning without reward engineering. *Robotics: Science and Systems* **2019**.
8. Haarnoja, T.; Ha, S.; Zhou, A.; Tan, J.; Tucker, G.; Levine, S. Learning to walk via deep reinforcement learning. *Robotics: Science and Systems (RSS)* **2019**.
9. Rolls-Royce. Marine RRC. Rolls-Royce and Finferries demonstrate world's first Fully Autonomous Ferry. <https://www.rolls-royce.com/media/press-releases/2018/03-12-2018-rr-and-finferries-demonstrate-worlds-first-fully-autonomous-ferry.aspx>, 2018. [Online; accessed 2021-05-18].
10. Skredderberget A., Y.I.A. The first ever zero emission, autonomous ship. <https://www.yara.com/knowledge-grows/game-changer-for-the-environment/>, 2018. Online; accessed 2021-05-18.
11. Shen, H.; Guo, C. Path-following control of underactuated ships using actor-critic reinforcement learning with mlp neural networks. *Sixth International Conference on Information Science and Technology (ICIST), IEEE*, p. pp. 317–321.
12. Martinsen, A.; Lekkas, A. "Curved-path following with deep reinforcement learning: Results from three vessel models. *OCEANS MTS/IEEE* **2018**.
13. Martinsen, A.; Lekkas, A. Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning. *IFAC-PapersOnLine* **2018**, p. 329–334.
14. Meyer, E.; A. Heiberg, A.R.; San, O. COLREG-compliant collision avoidance for unmanned surface vehicle using deep reinforcement learning. *IEEE Access* **2020**, *8*, 165344–165364.
15. Zhao, L.; Roh, M.I. COLREGs-compliant multiship collision avoidance based on deep reinforcement learning. *Ocean Eng.* **2019**, *191*.
16. Anderlini, E.; Parker, G.; Thomas, G. Docking Control of an Autonomous Underwater Vehicle Using Reinforcement Learning. *Applied Sciences* **2019**, *9*.
17. Rørvik, E.L.H. Automatic Docking of an Autonomous Surface Vessel : Developed using Deep Reinforcement Learning and analysed with Explainable AI; MA thesis. Trondheim, Norway: Norwegian University of Science and Technology (NTNU), 2020.
18. Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; Pedreschi, D. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* **2018**, *51*. doi:10.1145/3236009.
19. Adadi, A.; Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access* **2018**, *6*, 52138–52160. doi:10.1109/ACCESS.2018.2870052.
20. Veitch, E.; Alsos, O.A. Human centered explainable artificial intelligence for marine autonomous surface vehicles: concepts, strategies, and case study using Interaction Design (IxD) perspectives. *Journal of Marine Science and Engineering*. **9**(8). **2021**(Manuscript in preparation).
21. Glomsrud, J.; Ødegårdstuen, A.; Clair, A.; Smogeli, O. Trustworthy versus explainable AI in autonomous vessels. *ISSAV - International Seminar on Safety and Security of Autonomous Vessels* **2019**.
22. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* **2016**.
23. Ribeiro, M.T.; Singh, S.; Guestrin, C. Anchors: High-Precision Model-Agnostic Explanations. *AAAI Conference on Artificial Intelligence (AAAI)* **2018**.
24. Sundararajan, M.; Taly, A.; Yan, Q. Axiomatic Attribution for Deep Networks. Proceedings of the 34th International Conference on Machine Learning - Volume 70. JMLR.org, 2017, ICML'17, p. 3319–3328.
25. Lundberg, S.M.; Lee, S.I. A Unified Approach to Interpreting Model Predictions. Proceedings of the 31st International Conference on Neural Information Processing Systems; Curran Associates Inc.: Red Hook, NY, USA, 2017; NIPS'17, p. 4768–4777.
26. Covert, I.; Lundberg, S.; Lee, S.I. Understanding Global Feature Contributions With Additive Importance Measures, 2020, [arXiv:cs.LG/2004.00668].
27. Gjørnum, V.B.; Rørvik, E.L.H.; Lekkas, A.M. Approximating a deep reinforcement learning docking agent using linear model trees. *ECC* **2021**.
28. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms, 2017, [arXiv:cs.LG/1707.06347].
29. Agarap, A.F. Deep Learning using Rectified Linear Units (ReLU) **2018**.
30. et al., A.B.A. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* **2020**, *58*, 82–115.

31. Breiman, L.; Friedman, J.; Olshen, R., Stone, C. *Classification and Regression Trees*. *Wadsworth* **1984**.
32. Quinlan, R. Induction of Decision Trees. *Mach. Lear* **1986**, *1*, 81–106.
33. Quinlan, R. C4.5: programs for machine learning. *Elsevier* **2014**.
34. Avellaneda, F. Efficient Inference of Optimal Decision Trees. *AAAI Conference on Artificial Intelligence (AAAI)* **2020**, *34*, 3195–3202.
35. Izza, Y.; Ignatiev, A.; Marques-Silva, J. On Explaining Decision Trees **2020**.
36. Dinu, J.; Bigham, J.; Kolter, J.Z. Challenging common interpretability assumptions in feature attribution explanations, 2020, [[arXiv:cs.LG/2012.02748](https://arxiv.org/abs/cs.LG/2012.02748)].

5.4 Paper D

Postprint of [19]: **Vilde B. Gjørum**, Inga Strümke, Jakob Løver, Timothy Miller, and Anastasios M. Lekkas. "Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications". In: *Neurocomputing* 515 (2022), pp. 133–144. doi: <https://doi.org/10.1016/j.neucom.2022.10.014>

©2022 Vilde B. Gjørum, Inga Strümke, Jakob Løver, Timothy Miller, and Anastasios M. Lekkas. Reprinted under the terms of the Creative Commons Attribution License 

Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications

Vilde B. Gjørum¹, Inga Strümke², Jakob Løver³,
Timothy Miller⁴, Anastasios M. Lekkas¹

Abstract

Deep reinforcement learning has shown useful in the field of robotics but the black-box nature of deep neural networks impedes the applicability of deep reinforcement learning agents for real-world tasks. This is addressed in the field of explainable artificial intelligence, by developing explanation methods that aim to explain such agents to humans. Model trees as surrogate models have proven useful for producing explanations for black-box models used in real-world robotic applications, in particular, due to their capability of providing explanations in real time. In this paper, we provide an overview and analysis of available methods for building model trees for explaining deep reinforcement learning agents solving robotics tasks. We find that multiple outputs are important for the model to be able to grasp the dependencies of coupled output features, i.e. actions. Additionally, our results indicate that introducing domain knowledge via a hierarchy among the input features during the building process results in higher accuracies and a faster building process.

Index Terms

Explainable Artificial Intelligence, Model trees, Reinforcement Learning, Robotics

1. INTRODUCTION

The need for explaining artificial intelligence (AI) systems for decision making has become more prominent over the recent years [1], [3], [12], [16], [21]. This is also the case in the field of robotics, where *black-box* methods, such as deep neural

This work was supported by the Research Council of Norway through the EXAIGON project, project number 304843.

¹ V. B. Gjørum and A. M. Lekkas are with the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). E-mails: {vilde.gjaerum, anastasios.lekkas}@ntnu.no

² I. Strümke is with the Department of Computer Science at NTNU. E-mail: inga.strumke@ntnu.no,

³ J. Løver is with Nordic Semiconductor. E-mail: lover.jakob@gmail.com,

⁴ T. Miller is with the School of Computing and Information Systems at the University of Melbourne. E-mail: tmiller@unimelb.edu.au,

* The author is currently with Nordic Semiconductor.

networks, have shown great potential [14], [15], [20], [25], [28], but their usage for real-world applications with potential risk for humans and equipment is limited due to our lack of understanding of their decision making processes. The field of explainable artificial intelligence (XAI) has experienced rapid growth in response to this, and a large number of methods have been developed [8], [17], [21], [22], [29].

Reinforcement learning (RL) is a form of machine learning where an agent learns by autonomously exploring an environment. The environment responds to the actions of the agent by providing it with states and a reward. In machine learning (ML) terms, the states of the environment are thus the agent model's input features, while its outputs constitute the actions performed by the agent. We study explanation methods in the context of robotics and reinforcement learning. These problems often involve large state spaces, as the agent can explore a potentially open environment, which results in large training data sets.

In robotics settings, the states of the environment are often represented as continuous variables. For instance, autonomous vehicles use sensors that provide continuous measurements, such as position and velocity. Furthermore, the actions of robotic agents are often continuous. Some robotic applications also require real-time explanations, which constitutes a challenging additional requirement for the XAI methods.

In total, these requirements are challenging to meet for existing explanation methods. As seen in [18], both SHapley Additive exPlanations (SHAP) [17] and Local Interpretable Model-agnostic Explainer (LIME) [21] are not fast enough to be used in real-time for real-world robotic applications. Similar to LIME [21] and Anchors [22], both LOcal Rule-Based Explainer (LORE) [13] and FOILTREE [30] build a local interpretable model(a decision tree) around the instance to be explained and then searches that decision tree for explanations but neither are applicable to regression problems with multiple outputs. Additionally, they might not be fast enough to use in real-time for robotic applications since it needs to build a model for each instance to be explained as was shown to be a problem for both SHAP [17] and LIME in [19].

One class of methods that meet these aforementioned requirements is model trees when used as a surrogate model to gain insight into a black-box model as first presented in [10]. In short, the decision tree's intuitive structure can be utilized to extract explanations about a black-box given that the decision tree has adequately approximated the black-box model. The process of extracting explanations from a linear model tree in the form of feature attributions is thoroughly presented in Section 3. Decision trees are widely used for classification and regression tasks, and are popular for their more intuitive structure, especially compared to black-box models such as deep neural networks, which allows for a better understanding of their decision-making process. A decision tree model is a directed graph, consisting of one root node, branch nodes, and leaf nodes. The root node is the top node and the place where a data instance enters the model. Both the root node and the branch nodes contain a decision statement each, which split the data. Finally, the leaf nodes contain the model outcome for a data

instance following the root and branch node splits. The leaf nodes are at the bottom of the tree’s structure and have no children nodes. For classification trees, each leaf node predicts a class, i.e. a constant integer, while for regression trees, each leaf node predicts a constant. Thus, each leaf node has a constant prediction function.

The constant prediction in decision trees’ root nodes can be replaced by any type of prediction model, and the resulting decision tree is then commonly referred to as a *model tree*. The simplest form of model trees is regression trees with linear prediction functions in the leaf nodes, called linear model trees (LMTs). The structure of the tree corresponds to a specific partitioning of the feature space, and each region of this partitioning corresponds to one leaf node and its prediction function. Thus, an LMT is a piece-wise linear function approximator. In this work, we focus on methods for building model trees for explaining deep reinforcement learning (DRL) models acting as agents in robotics settings. Specifically, the methods must handle continuous output, yield non-constant prediction function output, and be able to deal with large datasets. They must also handle large data sets, and provide explanations quickly enough to be used in real time. In [11], an LMT was built by a heuristic, greedy algorithm that was used to provide explanations in the form of feature attributions of a deep neural network (DNN) controlling an autonomous surface vessel (ASV) in a simulated environment. Having approximated the DNN, the LMT ran in parallel with the controller and generated explanations in real time. The problem of finding the optimal LMT given the dataset can be represented as a mixed integer optimization problem, as shown in [5]. However, solving the optimal tree problem as a mixed integer optimization problem quickly becomes intractable for complex problems with large data sets which is the case for most robotic applications. Therefore, this is not a viable approach for our problem.

In [27], it is demonstrated that model trees can be used to build policies for reinforcement learning problems, but this method requires binary actions and is therefore not applicable to problems featuring continuous actions. Other commonly used method types for building decision trees include *satisfiability*(SAT) solvers, various constraint programming methods, and branch-and-bound search methods. Most SAT solvers, such as [4], [24], and constraint programming methods, such as [31], [32], work on classification problems with binary input features and are therefore not applicable to robotic applications. Additionally, many of the SAT solvers and constraint programming solvers do not scale well to larger data sets, one reason being that these methods add one constraint per sample in the dataset. Finally, branch-and-bound search methods, such as [2], also require binary input features.

Given the aforementioned requirements, the following is, to the best of our knowledge, an exhaustive list of *relevant* methods for building model trees in a robotics setting

- Optimal regression trees with linear prediction functions (ORT-L) [5]
- Optimal regression trees with linear prediction functions and hypersplits (ORT-LH) [5]
- Heuristic linear model tree (H-LMT) [11]
- Near optimal non-linear regression tree (NNRT) [6] .

The paper’s main contributions are the following:

- We provide an overview of the decision tree methods applicable to use as explanation methods for real-time, real-world robotic applications.
- We implement three methods(ORT-L, ORT-LH, and NNRT) on a robotic system with the intent of adding insight into a black-box’s decision making which, to the best of our knowledge, has not been done before.
- We present a comparison of the methods’ performances on a real-world problem involving an autonomous docking agent in a complex environment.
- We evaluate the four relevant methods in terms of
 - their strength and weaknesses,
 - their fidelity to the black-box model,
 - and their level of interpretability.

The paper is structured as follows. In Section 2, the methods are explained. How LMTs can be used as explanation methods is explained in Section 3. In Section 4, the docking problem used for the case study is presented, and the results from the case study are presented in Section 5. In Section 6, the methods’ results are analysed, and their strengths and weaknesses are discussed. Finally, we draw our conclusions in Section 7.

2. BACKGROUND

To approximate and explain DRL agents in complex environments using continuous states and producing continuous actions, the model tree algorithm must have the following capabilities:

- Ability to deal with large data sets
- Handle continuous input,
- Non-constant prediction function.

TABLE I: Characteristics of the different model tree algorithms.

	H-LMT	Sep. H-LMT	NNRT	ORT-L	ORT-LH
Continuous input	✓	✓	✓	✓	✓
Continuous output	✓	✓	✓	✓	✓
Piece-wise linear function	✓	✓	✗	✓	✓
Piece-wise polynomial function	✗	✗	✓	✗	✗
Univariate splits	✓	✓	✗	✓	✗
Multivariate splits	✗	✗	✓	✗	✓
Ordered feature splitting	✓	✓	✗	✗	✗
Optimality guarantees	✗	✗	✗	✗	✗
Multiple output	✓	✗	✗	✗	✗

Given these, the following is, as mentioned, an exhaustive list of relevant methods:

- Optimal regression trees with linear prediction functions (ORT-L), presented in Section 2-A,
- Optimal regression trees with linear prediction functions and hypersplits (ORT-LH), presented in Section 2-B,
- Heuristic linear model tree (H-LMT), presented in Section 2-C,
- Near optimal non-linear regression tree (NNRT), presented in Section 2-D.

Briefly summarised, the different methods work as follows. ORT-L and ORT-LH search for the best tree by locally improving the tree until convergence. H-LMT searches using a greedy approach, and NNRT uses gradient descent to determine the splitting functions and ridge regression for the prediction functions. ORT-L, ORT-LH, and H-LMT have linear prediction functions, while NNRT has polynomial prediction functions. ORT-LH and NNRT have multivariate split functions, also referred to as *hypersplits*, while ORT-L and H-LMT have univariate split functions.

ORT-L and ORT-LH are both built by using the Python package *Interpretable AI*¹. Unfortunately, the implementation of ORT-LH is unstable at the time of writing, which makes performing a systematic hyperparameter search challenging and time-consuming. The implementation of NNRT is an improved version of the implementation presented in [18]. H-LMT and separated H-LMT’s (sep. H-LMT) implementation is as presented in [11]. The only difference between H-LMT and sep. H-LMT is that sep. H-LMT builds one tree per output, while H-LMT builds one tree with several prediction functions in the leaf nodes.

The different methods are further explained in the following, and Table I contains an overview of their characteristics.

¹<https://docs.interpretable.ai/stable/>

2.1. *ORT-L*

ORT-L is a method for building LMTs with univariate splits in the branch nodes, whose overall objective function can be expressed as

$$\min_{\beta, \beta_0} \frac{1}{\hat{L}} \sum_{i=1}^n (y_i - f_i)^2 + \lambda \sum_{t \in \mathcal{T}_L} \|\beta_t\|_1, \quad (1)$$

where y_i is the prediction target, f_i is the ORT-L model prediction, \mathcal{T}_L is the set of all leaf nodes, \hat{L} is the baseline error in the training data, β are the coefficients of the linear prediction function in the leaf node t , and λ is a regularization factor. The objective function in Equation (1) concerns both the structure of the tree, which maps an input sample to its respective leaf node and prediction function, represented as f , and the fitting of all the prediction functions in the leaf nodes. The problem of finding the structure of the tree and fitting the linear prediction function in the leaf nodes can be divided into two separate problems. The objective function can thus be written as

$$\min_{\beta, \beta_0} \frac{1}{2|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \frac{\lambda \hat{L}}{2|\mathcal{I}_t|} \|\beta_t\|_1, \quad (2)$$

where \mathcal{I}_t is the set of data points that fall into leaf node t , and $|\mathcal{I}_t|$ is the number of data points that fall into leaf node t . The *local search* method provided by [5] searches for the solution to Equation (1) by iteratively improving the tree until it reaches a local optimum. For each iteration, this method locally optimizes the splits in all branch nodes in random order, followed by fitting the linear prediction functions in the leaf nodes, given the current structure of the tree.

One challenge in the process of finding the optimal LMT is that even small changes to the structure of the tree can require recalculating many of the prediction functions in the leaf nodes. This is because even adding just a few samples to the regression problem can cause significant changes to the final leaf node function. A regression method that addresses this is the GLMNet algorithm [9], which solves the regression problem efficiently by using coordinate descent.

2.2. *ORT-LH*

ORT-LH is a method for building LMTs with multivariate splits, or *hypersplits*, in the branch nodes. The only difference between ORT-LH and ORT-L lies in the search for the best splits, where ORT-L searches for the best splitting criteria on one feature, while ORT-LH searches for the best *splitting function*. The splitting criteria for ORT-L are expressed as

$$x_j < t, \quad (3)$$

while the splitting criteria for the hypersplits in ORT-LH are expressed as

$$\sum_{j \in \mathcal{J}} \beta_j x_j < t, \quad (4)$$

where t is the threshold, \mathcal{J} is the set of features, and β are the coefficients of the hypersplit function.

2.3. H-LMT

H-LMT, presented in [11], is a heuristic method for building LMTs with univariate splits. The method, presented in Algorithm 1, follows a procedure similar to CART [7].

Algorithm 1: H-LMT

Require:

Training data \mathcal{D}

Maximum number of leaf nodes N

Minimum number of data samples for leaf nodes M

Order of which features can be used for splits **OFS**

Number of thresholds to search for when performing splitting \mathbf{T}_s

Number of thresholds to search for calculating the potential split value \mathbf{T}_{psv}

do:

Split the root node by using Equation (6)

Calculate the potential split for the new nodes using Equation (6) searching through **OFS**(depth) and \mathbf{T}_{psv} thresholds

while number of leaf nodes is less than N **do**

if there exists a node that fulfills all splitting criteria **then**

 Choose node to split using Equation (5)

 Perform splitting using Equation (6)

 Calculate best potential split for the newly created nodes using

 Equation (6) searching through **OFS**(depth) and \mathbf{T}_{psv} thresholds

else

return root node

end

end

As in CART, the method starts from the root node and incrementally expands the tree as long as this is possible, given the restrictions set by the hyperparameters of the algorithm. H-LMT does not take maximum depth as a restriction but rather the maximum number of leaf nodes allowed. Using maximum number of leaf nodes instead of maximum depth allows for growing asymmetric trees because the tree can grow deeper in the more complex parts of the feature space while leaving the parts of the tree covering simpler regions shallow. For this to happen the tree cannot grow one branch at a time as deep as it allows (as done in CART) but must rather make some intuition about which regions should be partitioned further.

When choosing the next node to split, all the leaf nodes' *potential loss value* are compared. How to choose the next node to be split can be expressed as

$$l_{tbs} = \max_{l \in L} \text{potential split value}(l), \quad (5)$$

where l_{tbs} is the leaf node that gives the highest improvement of error for the model called the *potential split value*, and L is all leaf nodes. The potential loss value is calculated by performing a split search on the children of a newly split node as in Equation (6) but searching through fewer possible thresholds. If a particular child is later on chosen as the next node to be split, a thorough split search is done.

The method provides no guarantee of optimality and there is therefore no need for the method to output the same tree at every run given the same data set and parameters. To further explore the solution space, randomness is used when searching for the thresholds in the splits and when deciding which node to split next. If run in parallel, several different trees can be built without increasing the run-time. Equation (6) is used both when performing splitting and when calculating the best potential split for newly created nodes in Algorithm 1, the difference being the number of thresholds explored. The feature \mathcal{F} and the threshold t_n to be split upon are found via

$$\mathcal{F}, t_n = \underset{\mathcal{F}, n}{\operatorname{argmin}} (\text{loss}(\mathcal{D}_L) + \text{loss}(\mathcal{D}_R)), \quad (6)$$

where

$$\begin{aligned} \mathcal{D}_L &= x \in \mathcal{D} \quad \text{if} \quad x_{\mathcal{F}} \leq t_n, \\ \mathcal{D}_R &= x \in \mathcal{D} \quad \text{if} \quad x_{\mathcal{F}} > t_n, \end{aligned} \quad (7)$$

$$t_n = \min(\mathcal{D}^{\mathcal{F}}) + (n+r) \frac{(\max(\mathcal{D}^{\mathcal{F}}) - \min(\mathcal{D}^{\mathcal{F}}))}{N}, \quad (8)$$

and

$$n_s = \underset{n}{\operatorname{argmax}} ((1+r) (\text{loss}(\mathcal{D}_L^n) + \text{loss}(\mathcal{D}_R^n))). \quad (9)$$

The data sets belonging to the left and right child node of the current node are denoted \mathcal{D}_L and \mathcal{D}_R , respectively. The search includes N evenly distributed thresholds for each feature in the range spanned by that feature. The randomness is introduced through the value r , which changes the threshold value or the potential loss value by up to 2%.

In [11], a restriction on which features could be used for finding the best split at different depths of the tree was set called ordered feature splitting (OFS). OFS maps different depths of the tree to a set of features that can be used for the splitting condition in the branch nodes at this depth. This means that we can manipulate the trees' structures, and utilize domain knowledge or improve the trees' interpretability by getting a more intuitive structure or grouping splits on certain features. This significantly speeds up the algorithm, since it decreases the number of features that

must be searched through when calculating the best split combination. Such ordering of features must be done with care – since it strongly affects the resulting trees – preferably by testing various options and adapting specifically to the task at hand.

2.4. NNRT

NNRTs is a method for building model trees with multivariate splits and polynomial prediction as a nonlinear optimization problem with a near-optimal solver, introduced in [6]. NNRT’s overall prediction function is expressed as

$$g(x) = \sum_{p \in L} g_p(x) \prod_{l \in \mathcal{L}_p^<} \mathbb{1}\{\mathbf{a}_l^T \mathbf{x} < b_l\} \prod_{l \in \mathcal{L}_p^>} \mathbb{1}\{\mathbf{a}_l^T \mathbf{x} \geq b_l\}, \quad (10)$$

where x is the input sample, L is the set of all leaf nodes, $\mathcal{L}_p^<$ and $\mathcal{L}_p^>$ denotes the set of all the left and right parents of the node with their corresponding splitting criteria $\{\mathbf{a}_l^T \mathbf{x} < b_l\}$ or $\{\mathbf{a}_l^T \mathbf{x} \geq b_l\}$. The polynomial prediction function in the leaf nodes is given by

$$g_p(x) = f_{p,D+1} + \sum_{i=1}^D f_{p,i} \prod_{l=l_1}^{l_i} |\mathbf{a}_l^T \mathbf{x} - b_l|, \quad (11)$$

where D is the depth of the tree. As we can see from Equation (11), the polynomial function in a leaf node consists of the linear splitting functions from the branch nodes along the path from the root node to the leaf node. However, since

$$|\mathbf{a}_l^T x - b_l| \mathbb{1}\{\mathbf{a}_l^T \mathbf{x} < b_l\} = \max(b_l - \mathbf{a}_l^T \mathbf{x}, 0) \quad (12)$$

$$|\mathbf{a}_l^T x - b_l| \mathbb{1}\{\mathbf{a}_l^T \mathbf{x} \geq b_l\} = \max(\mathbf{a}_l^T \mathbf{x} - b_l, 0) \quad (13)$$

the problem can also be formulated as a nonlinear optimization problem as follows:

$$g(x) = \sum_{p \in L} f_{p,i}(x) \prod_{l \in \mathcal{L}_{i_i}^<} \max\{b_l - \mathbf{a}_l^T x, 0\} \prod_{l \in \mathcal{L}_{i_i}^>} \max\{\mathbf{a}_l^T x - b_l, 0\}, \quad (14)$$

The polynomial functions in the leaf nodes of the tree are thus made out of the linear functions (the hypersplits) in the branch nodes on the path from the root node to the activated leaf node. This means that the depth of the tree determines the power of the polynomial prediction functions in the leaf nodes. The final objective to be minimized is thus

$$\min_{\mathbf{a}, \mathbf{b}, \mathbf{c}} L(\mathbf{a}, \mathbf{b}, \mathbf{f}) = \frac{1}{n} \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda (\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2 + \|\mathbf{f}\|_2^2). \quad (15)$$

NNRT searches for the optimal solution to 15 by alternating between applying gradient descent to the hypersplit parameters a and b of the branch nodes, and by fitting the leaf nodes parameters f via ridge regression.

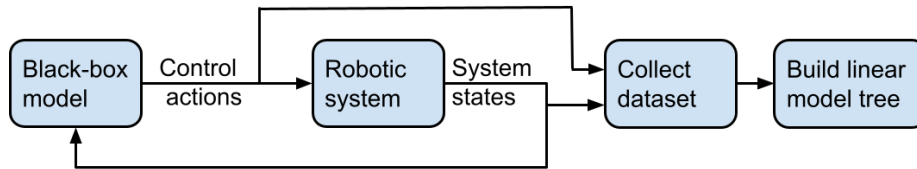


Fig. 1: Illustration of the pipeline for how to build an LMT for a reinforcement learning agent trained on a robotic system.

3. LINEAR MODEL TREES AS EXPLAINERS

One strategy within XAI is to use a simpler, more interpretable model as a *surrogate model* for the complex black-box model. The surrogate model mimics the behaviour of the black-box model locally around an instance to be explained or globally for the entire black-box model and we can then learn something about the black-box model by analysing the surrogate model [1]. An LMT can be used as a surrogate model to support explanation whenever the tree is capable of approximating the black-box model to be explained. For problems suited for supervised learning, training the surrogate model is straightforward since it can just be trained on the same data as the black-box model was trained on. For robotic applications, supervised learning is often not possible because the correct output is not known and therefore reinforcement learning is often used. Figure 1 illustrates how we train an LMT for a reinforcement learning agent. For the LMT to approximate the model properly we first need to gather the appropriate data. Such data can be obtained by sampling states from the environment and running these states through the black-box model to get its action and using this state-action dataset to build the LMT. If the LMT approximates the black-box model accurately enough, it serves as a piece-wise linear function approximator to the black-box model, and can run in parallel and give explanations in real-time as illustrated in Figure 2 [11]. When LMTs are built by the process shown in Figure 1 and used as shown in Figure 1, the LMT is a post-hoc, model-agnostic explanation method. Post-hoc meaning applied to a preexisting model, and model-agnostic meaning it can be applied to any type of model [1]. Using LMTs as explanation models yield the following advantages:

- 1) their structure is intuitive and their decision flow directly available to humans,
- 2) they merely split, not transform, the input features, leaving these interpretable to humans
- 3) they are fast enough to run in real time even for robotic applications.

Gjaerum et al. [10], [11] present a way to retrieve explanations in the form of feature attributions – the same type of explanations given by other explanation methods such as SHAP and LIME – from an LMT. Since each leaf node corresponds to a certain region in the state space, feature attributions are local explanations. Even though decision trees are often considered to be transparent and interpretable for humans, analysing

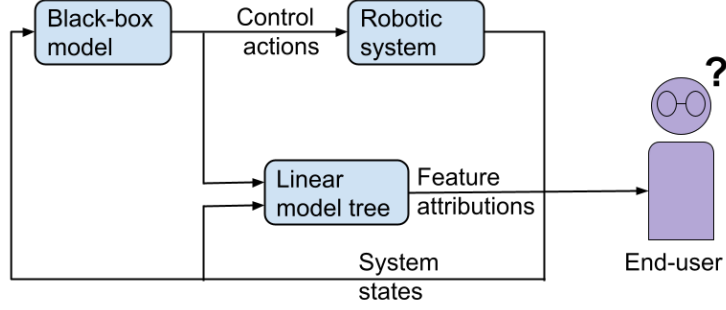


Fig. 2: Illustration of how the black-box model controlling the robotic system and the LMT explaining the black-box model runs in parallel.

the structure and splits of the tree is very time consuming depending on the tree's size. The linear functions $f_i(x)$ in the leaf nodes can be written as

$$f_i(x) = \sum_{f=1}^F (w_f x_f) + w_{F+1}, \quad (16)$$

where w_f is the coefficient of feature f in the input vector x . The importance I_F attributed to feature F by the model is then

$$I_F = \frac{w_F x_F}{\sum_{f \in \mathcal{V}_F} |w_f x_f|}. \quad (17)$$

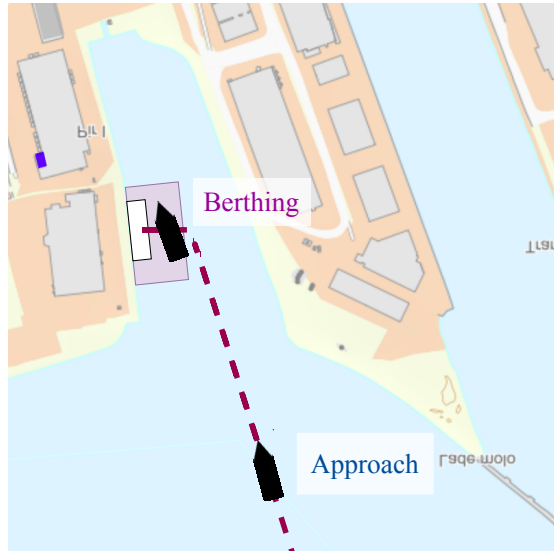


Fig. 3: Overview of the simulated environment from [23].

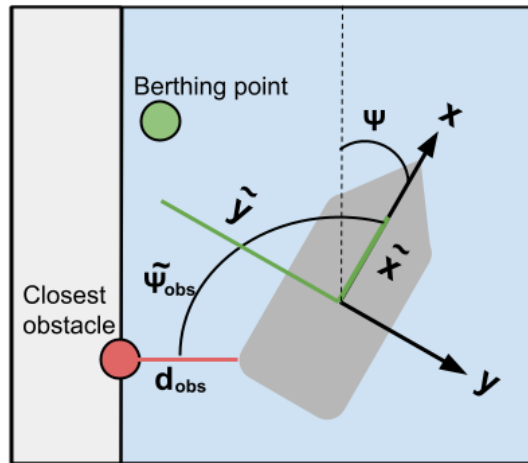


Fig. 4: Illustration of simulated vessel and the states representing it, from [11].

4. CASE STUDY: THE DOCKING PROBLEM

To evaluate the four methods' utility values in robotic applications, we apply them to an RL agent solving the problem of docking an ASV in a simulated environment based on a physical harbour, shown in Figure 3. The physical harbour is located in Trondheim, Norway harbour and the RL docking agent used is the same as in [23]. The vessel has three thrusters which can be controlled through five input actions given by

$$\mathbf{A} = [f_1, f_2, f_3, \alpha_1, \alpha_2], \quad (18)$$

where the two azimuth thrusters at the back of the vessel are controlled by setting their force and angle, f_1, f_2 and α_1, α_2 . The tunnel thruster at the front of the vessel is controlled by setting its force f_3 . The forces of the azimuth thrusters are restricted to $[-70 \text{ kN}, 100 \text{ kN}]$, and the angles are restricted to $[-90 \text{ degrees}, 90 \text{ degrees}]$. The tunnel thruster's force is restricted to $[-50 \text{ kN}, 50 \text{ kN}]$. The vessel's states are illustrated in Figure 4 and expressed as

$$\mathbf{x} = [\tilde{x}, \tilde{y}, \tilde{\psi}, u, v, r, l, d_{obs}, \tilde{\psi}_{obs}]. \quad (19)$$

where \tilde{x}, \tilde{y} , and $\tilde{\psi}$ gives the vessels position relative to the berthing point in the vessel's body frame. The velocity is represented by u, v , and r . The vessel's position relative to the closest obstacle is given by d_{obs} and $\tilde{\psi}_{obs}$. Additionally, the binary variable l indicates whether or not the vessel has made contact with an obstacle. The variable l is mainly used under the training of the RL-agent. The RL-agent used to control the ASV is a neural network with two hidden layers of 400 neurons each trained with the proximal policy optimization (PPO) algorithm from [26]. The task of docking an ASV is a challenging problem that requires complex decision making and accurate fine-tuning of the actions, especially when the ASV is close to the harbour and needs to be very precise in its movements.

5. RESULTS

In this section, the four methods H-LMT, NNRT, ORT-L, and ORT-LH are compared on a case study of the docking problem.

5.1. Tree structure

In Table II, the depth and number of leaf nodes are given for NNRT, ORT-L, and ORT-LH. H-LMT and sep. H-LMT's structure is only given by the number of leaf nodes since the depth of the leaf nodes may vary which can be seen in Figure 5. Additionally, the total number of leaf nodes and the total number of prediction functions are also given in Table II. Even though H-LMT only has a total number of 18 leaf nodes, it's still important to remember that its number of prediction functions is five times higher, namely 90. Still, having multiple outputs per tree significantly reduces the number of splitting functions and regions the dataset is split into since all actions use the same tree structure. Besides ORT-L building somewhat bigger trees than the

TABLE II: Overview of the trees’ structures, in terms of depth and number of leaf nodes denoted by *.

	Depth of tree					Total number of leaf nodes	Total number of prediction functions
	f_1	f_2	f_3	a_1	a_2		
H-LMT			18*			18	90
NNRT	4(16*)	4(16*)	4(16*)	4(16*)	4(16*)	80	80
ORT-L	2(4*)	4(16*)	6(64*)	3(8*)	7(128*)	220	220
ORT-LH	4(16*)	4(16*)	4(16*)	4(16*)	4(16*)	80	80
sep. H-LMT	12*	13*	13*	22*	27*	87	87

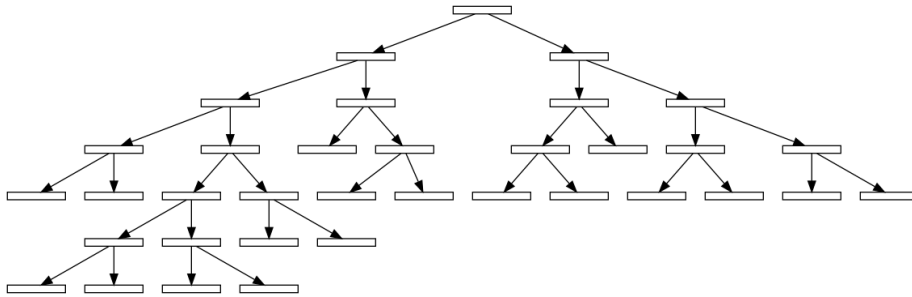


Fig. 5: H-LMT’s structure.

other methods, mostly due to the trees built for a_2 and f_3 being quite large, no method sticks out for building more sparse models compared to the other methods. NNRT and ORT-LH are both expected to build shallower trees since these use multivariate splits instead of univariate splits. ORT-LH is smaller than ORT-L, but neither ORT-LH nor NNRT are significantly smaller than H-LMT or sep. H-LMT. Compared to the original policy network which consisted of two hidden layers of 400 neurons each, the trees are considerably smaller in addition to being structured in a way more intuitive to humans.

5.2. Prediction error

In Table III, the mean absolute error and standard deviation of the different trees’ predictions are given. The dataset consists of 80% randomly sampled starting positions and 20% is sampled from paths where the DRL-agent controls the vessel given a random starting point. Since practically all paths end up at the berthing point, many of the states in different runs will be very similar. This is why most of the data used for calculating these error measures are from randomly sampled starting points so that the dataset is not skewed to this particular part of the state space Sep. H-LMT has

TABLE III: MAE and standard deviation for all actions.

	Average error				
	f_1 (kN)	f_2 (kN)	f_3 (kN)	a_1 (deg)	a_2 (deg)
H-LMT	8.45(4.9%)	8.04(4.7%)	2.8(2.8%)	9.17(5.09%)	10.31 (5.73%)
NNRT	21.92(12.89%)	17.53(10.31%)	7.82(7.82%)	18.9(10.5%)	26.92(14.96%)
ORT-L	10.02(5.89%)	8.57(5.04%)	2.92(2.92%)	10.31(5.73%)	10.89(6.05%)
ORT-LH	7.16(4.2%)	8.39(4.9%)	3.96(3.96%)	11.46(6.37%)	12.61(7%)
Sep. H-LMT	6.98(4.1%)	7.23 (4.25%)	3.27 (3.27%)	8.6 (4.78%)	9.74(5.41%)
	Standard deviation				
	f_1 (kN)	f_2 (kN)	f_3 (kN)	a_1 (deg)	a_2 (deg)
H-LMT	14.2(8.35%)	12.2(7.18%)	4.51(4.51%)	12.6(7%)	13.75(7.64%)
NNRT	24.65(14.5%)	20.66(12.15%)	9.71(9.71%)	20.63(11.46%)	20.05(11.14%)
ORT-L	12.81(7.54%)	13.46(7.91%)	4.9(4.9%)	13.18(7.32%)	14.9(8.28%)
ORT-LH	11.54(6.79%)	12.73(7.48%)	6.13(6.13%)	14.32(7.96%)	14.9(8.28%)
Sep. H-LMT	10.82 (6.36%)	11.6(6.82%)	4.92 (4.92%)	12.6(7%)	13.18(7.32%)

the lowest average error on all actions except f_3 , and the lowest standard deviation on all actions except f_3 and a_1 where H-LMT has the same standard deviation. Overall, ORT-L has a somewhat higher error and standard deviation compared to sep. H-LMT and H-LMT. NNRT has both very high error and standard deviation, for some actions even more than twice as high as the other methods. ORT-LH performs similarly to H-LMT and sep. H-LMT on f_1 , f_2 , and f_3 but struggles more with approximating a_1 and a_2 .

5.3. Path comparison

One way of evaluating how well a tree approximates the DRL-agent is by letting the tree control the vessel and comparing their behaviour related to the DRL-agent's behaviour. A good approximation performs similarly to the DRL-agent not only when the DRL-agent performs well but also when it performs poorly. We do not want the approximation model to find solutions to the areas where the DRL-agent struggles since then the fidelity of the explanations will be weakened. An example of this can be seen in Figure 6, where ORT-L manages to arrive close to the berthing point but it deviates from the DRL-agent's path along the way. H-LMT and ORT-LH on the other hand do not get as close to the berthing point but their paths more closely resemble the DRL agent's path, and are thus better approximations in this situation.

Since sep. H-LMT achieves lower error than H-LMT, as shown in Table III, sep. H-LMT is expected to be more capable than the other methods to control the vessel.

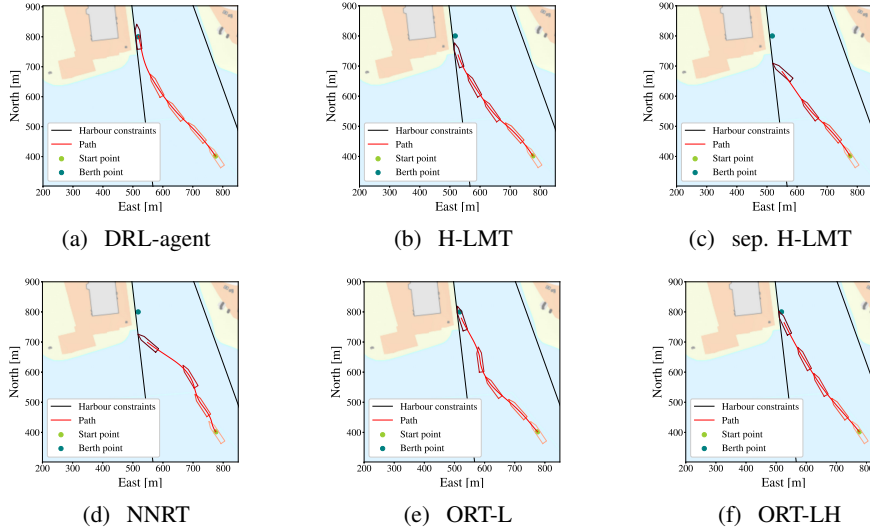


Fig. 6: Path plots of the vessel controlled by the DRL-agent and the trees given the same starting point.

However, sep. H-LMT at best performs marginally better than H-LMT, as can be seen in Figure 7. In Figure 7, we can see that in the last steps sep. H-LMT approaches the berthing point from the right, H-LMT approaches slightly from above and must go backwards the last steps but the DRL-agent approaches the berthing point first from the right and then moves forward the last few steps. On the other hand, for most of the scenarios and as can be seen in Figure 6, H-LMT performs better than sep. H-LMT. Specifically, in Figure 6, sep. H-LMT crashes with the harbour limits due to not having the right angle. This could be due to that sep. H-LMT consists of five trees, one per each action, while H-LMT only consist of one tree which predicts all five outputs. Decoupling these very dependent actions most likely leads to a loss of general understanding of what these actions do. Since f_1 and a_1 control the left azimuth thruster together, a mistake in only one of them will still lead to a wrong control action for that thruster. Some mistakes will also have more effect on the final control input the vessel receives than others. Let's consider the two following prediction errors:

- 1) f_1 is supposed to be -5 kN but $+5$ kN is predicted
- 2) f_1 is supposed to be 50 kN but 40 kN is predicted

Both of these scenarios have an absolute error of 10 kN, but the first error will affect the outcome much more since this will make the thruster give a force in the opposite direction.

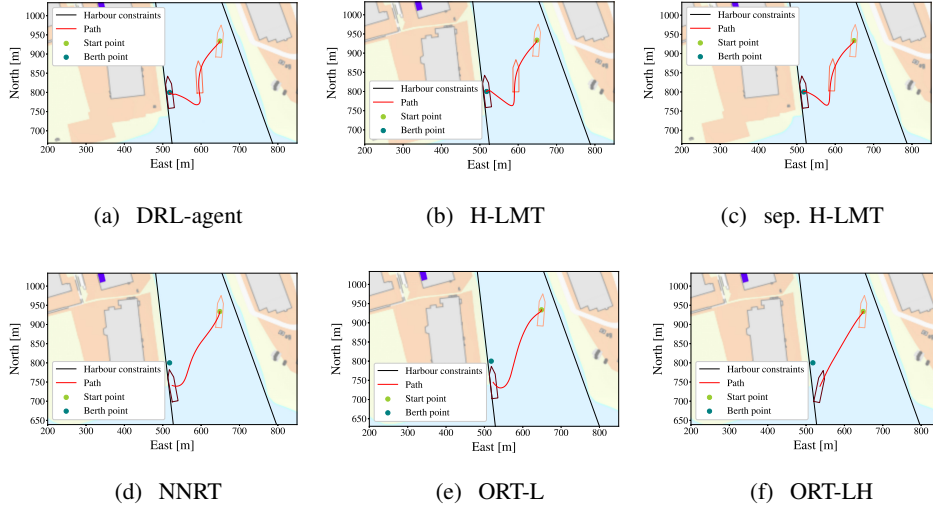


Fig. 7: Path plots of the vessel controlled by the DRL-agent, H-LMT, and sep. H-LMT given the same starting point.

As expected based on the errors shown in Table III, NNRT performs rather poorly as can be seen in Figure 6. NNRT has not managed to grasp the overall concept, but there are some situations where NNRT performs somewhat better, such as the one we can see in Figure 8 where the vessel is positioned in front of the berthing point and moves backwards towards the berthing point. ORT-LH manages to get the vessel close to the berthing point via a similar path as the DRL-agent in the situation shown in Figure 6, but shows the poorest performance in the situation showed in Figure 8. In Figure 7, NNRT, ORT-L, and especially ORT-LH perform poorly.

5.4. Action comparison

Another way of evaluating how well the tree methods approximated the neural network is by looking closer at how close the actions predicted by the neural network and the tree given the same states. We do this by letting the DRL-agent control the vessel and running the trees in parallel, predicting actions for the same states. One important difference between these episodes and the ones shown in Section 5-C, is that for those episodes, the actions of the trees do not affect the next state, since the DRL-agent controls the vessel.

In Figure 9, we can see the difference between what the DRL-agent and the NNRT give as actions given the same states. The NNRT struggles with abrupt changes in the actions, which can be seen from around step 550 and after. At this part of the episode,

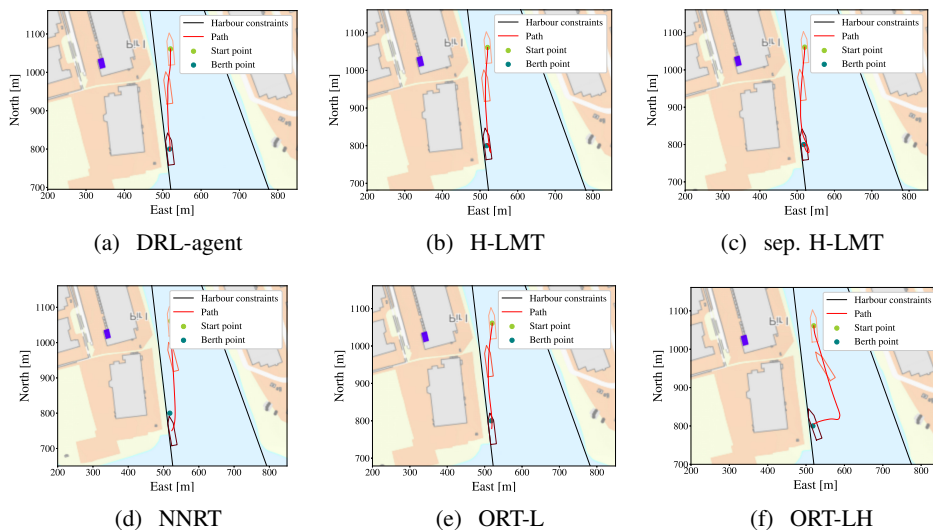


Fig. 8: Path plots of the vessel controlled by the DRL-agent, NNRT, ORT-L, and ORT-LH given the same starting point.

the vessel has reached the berthing point and stabilises the vessel around this point through repetitive actions. We can see the same thing happening for the NNRT, and to some degree also for ORT-L and ORT-LH in Figure 10. Figure 10 shows the difference in actions for states in the episode showed in Figure 6 controlled by the DRL-agent. As expected given previously shown results for NNRT, NNRT predicts quite different actions as the DRL-agent given the same states. The simulated docking environment only accepts actions within range of $[-1, 1]$ so actions outside of this range will be clipped if the model is controlling the vessel. However, this will affect the feature attributions and thus the explanations' fidelity. Another aspect that is important to note is that since this is an *end-to-end control process* (meaning that the agent controls the actuators directly and not through a control allocation system) is that there are an endless amount of action combinations that will lead to the same combined control action. For example, f_1 equal to 10 kN and a_1 equal to 30 degrees yield the exact same thruster status as having f_1 equals to -10 kN and a_1 equal to -30 degrees. For the purpose of controlling the vessel, there is no difference between the two action-pairs, but for the case of the explanations, it is two vastly different outcomes. Overall, as can also be seen in Figure 10 after approximately step 500, H-LMT, sep. H-LMT, and to some degree ORT-L and ORT-LH has managed to approximate the DRL-agent quite well. This could be due to this being an area that is well-represented in the training

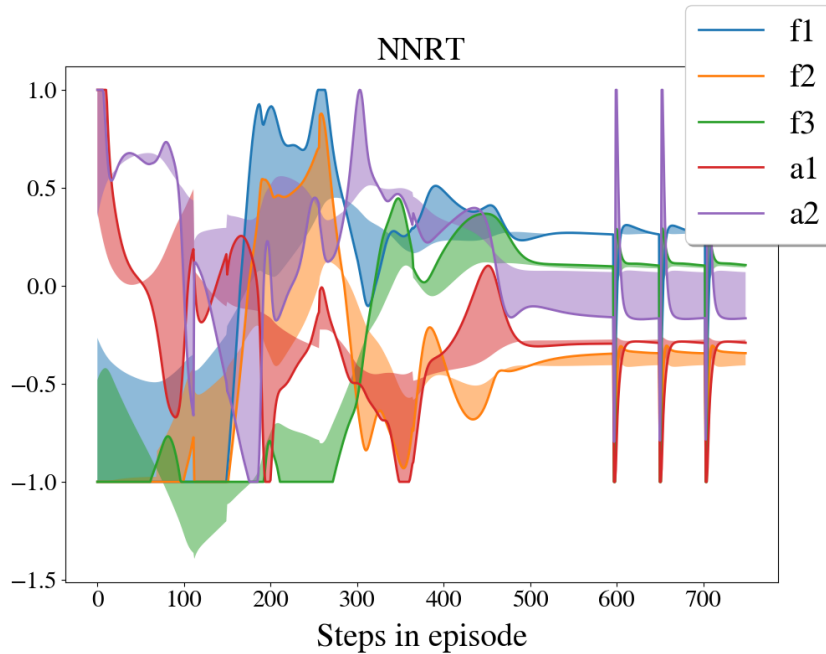


Fig. 9: The difference between the DRL-agent’s actions and the NNRTs actions is highlighted by the shaded area. These actions are for the episode shown in Figure 7 where the DRL-agent controls the vessel and the NNRT predicts actions for the states without controlling the vessel.

data since it appears for most episodes regardless of the starting point. Some of the leaf nodes in ORT-LH have either constant prediction functions, or linear prediction functions relying on few input features leading to nearly constant predictions. This can be seen in f_1 in Figure 10. The methods often have similar under- and overestimations of the predicted actions which could indicate they have misunderstood the mapping from state to action in the same manner, or perhaps have discovered the same (perhaps irrelevant) pattern.

5.5. Interpretability metrics

In Table IV, a categorization of the different tree methods’ level of transparency and structural properties is given.

In [16], the transparency of a method is divided into three different levels, namely simulatable, decomposable, and algorithmic transparency. For a method to be simulatable transparent, it must be so simple that a human can simulate the outcome of

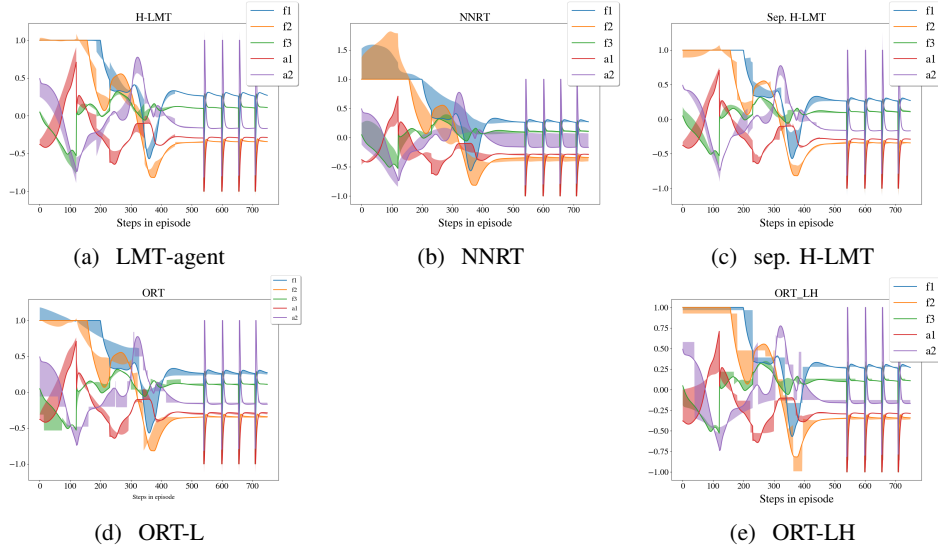


Fig. 10: The difference between the DRL-agent's actions and the trees' actions given the same states from the path shown in Figure 6

a method given a certain input. For a method to be decomposable transparent, the model cannot alter the data in a way that makes them unreadable to humans, for example through complex feature engineering. Lastly, for a method to be algorithmic transparent, it must be possible to evaluate the method mathematically [3]. For the categorization of the methods' level of transparency, we assume that the trees are of the same size. This especially becomes important when deciding whether or not the different trees are simulatable transparent. We decide that only H-LMT is simulatable transparent since it's the only one that has all three of the structural properties on what we've considered more interpretable, namely univariate splits, linear prediction functions, and multiple outputs. Univariate splits are considered more interpretable than multivariate splits since univariate splits only consist of one variable and one threshold, whereas multivariate splits consist of several variables and their coefficients, and a threshold. Thus, univariate splits are much more sparse than multivariate splits. Linear functions are considered more interpretable than polynomial functions because they, given that they consist of the same number of variables, are easier to compute, especially for humans. It is also easier to know how a change in a variable will affect the outcome of a linear function than for a polynomial function. Lastly, we consider trees with multiple outputs more interpretable than multiple trees with single output because 1) the model consists of significantly fewer splits, and 2) it is considerably easier to see

TABLE IV: The decision trees’ levels of transparency and structural properties that affect their interpretability.

Transparency			
	Simulatable	Decomposable	Algorithmic
ORT-L	No	Yes	Yes
ORT-LH	No	Yes	Yes
H-LMT	Yes	Yes	Yes
Sep. H-LMT	No	Yes	Yes
NNRT	No	Yes	Yes
Structural properties			
Split	Prediction function	Num. of outputs	
ORT-L	Univariate	Linear	Single
ORT-LH	Multivariate	Linear	Single
H-LMT	Univariate	Linear	Multiple
Sep. H-LMT	Univariate	Linear	Single
NNRT	Univariate	Polynomial	Single

how the different outputs change in relation to each other since the prediction functions belong to the same regions of the input space. All three of the structural properties related to interpretability are affected by the size of the trees. It’s not easy to say how much bigger a tree with univariate splits, linear prediction functions, and multiple outputs can be compared to multiple small trees with multivariate splits and polynomial prediction functions to still be considered most interpretable. This consideration will of course also be affected by both the end-user of this model, as well as the application they are used on. Following this intuition, H-LMT is considered to be the most interpretable followed by ORT-L and sep. H-LMT in second. ORT-LH is considered the second least interpretable, and NNRT is considered the least interpretable method.

6. DISCUSSION

NNRT,ORT-L, and ORT-LH cannot build trees with multiple outputs and must therefore build one tree for each action. This is an important drawback because 1) the vessel is controlled through the combination of the actions and by building one model for each action the dependencies between the actions are lost, and 2) the model becomes less interpretable. Only H-LMT can build a tree that gives five outputs, while sep. H-LMT builds five H-LMTs, one for each action. As shown in III, separated H-LMT achieves lower errors than H-LMT with multiple actions, and one could therefore expect that sep. H-LMT would perform better at controlling the vessel. Instead, the two methods perform similarly at the task, and in most situations the H-LMT yields superior performance to sep. H-LMT. A likely explanation for this is that when building one tree per action the codependency between the actions is lost.

NNRTs consistently performs poorly. This could be due to the fact that expanding the size of the tree comes at the cost of increased complexity of the polynomial prediction function in the leaf node. Since the power of the polynomial in the leaf node corresponds to the depth of the tree the prediction function quickly becomes unnecessary big while the number of leaf nodes and thus the number of regions the feature space is divided into still is too small.

Considering H-LMT is a greedy method and that ORT-L performs a much more thorough search for the optimal tree, it is expected that ORT-L would build better trees than H-LMT. However, this is not the case and this could be due to several reasons:

- H-LMT uses OFS,
- H-LMT uses maximum number of leaf nodes as a parameter instead of maximum depth,
- H-LMT is able to produce multiple outputs for each tree.

Using OFS may improve the trees because the tree is given a structure that is likely to help the tree produce better regions based on domain knowledge. As previously mentioned, this should be done carefully with the specific task at hand in mind. For this application, the following feature ordering is used

- 1) $\tilde{x}, \tilde{y}, \tilde{\psi}$;
- 2) d_{obs}, ψ_{obs} ;
- 3) u, v, r .

Using maximum number of leaf nodes as a hyperparameter instead of maximum depth gives H-LMT a bigger solution space to explore than ORT-L since it allows for more asymmetrical trees. One drawback of having one tree with multiple outputs becomes prominent if the different outputs have very different complexity. If one output requires a very deep branch to be predicted accurately the other outputs must also go through the same branches although this might not be necessary. Additionally, although sep. H-LMT achieves slightly lower errors compared to H-LMT, having one tree with multiple outputs seems to be beneficial over having one tree per output.

Due to instability issues in the implementation of ORT-LH, the search for the best parameter settings for this method was not done as rigorously as for the other methods in our experiments. Hence, we cannot rule out that it is possible to build significantly better performing trees with different hyperparameters.

It is important to remember that the explanations given by the tree cannot bring any insight regarding the black-box model if they are not similar enough, i.e. if the tree has not approximated the model with adequate accuracy. This accuracy should be taken into account when analysing the model through the explanations of the tree, for example by looking at how different the output of the tree and the model is.

Even though sparse model trees are more interpretable than large neural networks, most model trees are still too complex to be understood in real time. It is therefore needed that the necessary and relevant information is extracted and presented to the end-users in a suitable way. Feature attributions on their own do not constitute a

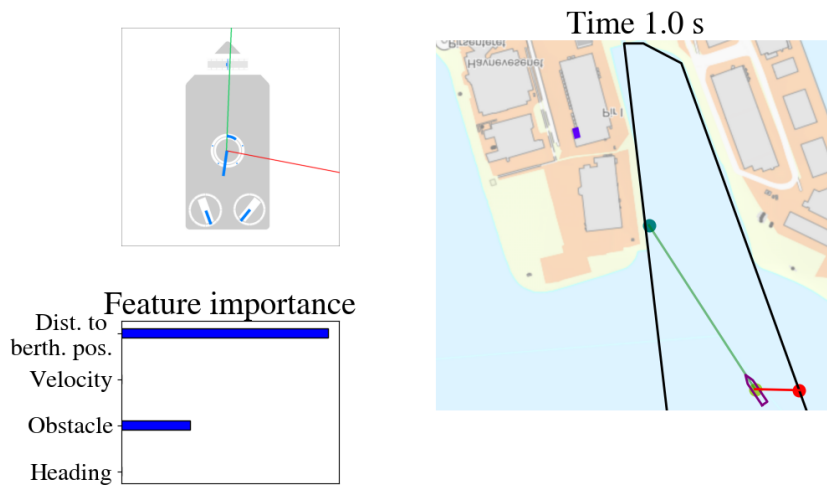


Fig. 11: The visualizations of the feature attributions given by the H-LMT, the vessel's states and actions aimed for the operator or seafarer [11].

complete explanation of a model, but can serve as part of one by highlighting how the inputs affect the outputs of a model, or in terms of a robotic application, how the states of the environment affects the action of the control agent. What constitutes a complete explanation depends on the field, application, use-case, and end-user. In [11], two different visualizations and level of detail catered to two different end-users, namely the operator or seafarer of the vessel and the developer of the system, was suggested. The visualization of the vessel's state, actions, and explanations in the form of feature attributions is shown in Figure 11. Global explanations, meaning explanations that explain the entire inner workings of the black-box model are desirable to understand the behaviour of the control agent as a whole [1]. By looking at the linear prediction function in a leaf node, we can learn something about the behaviour of the model in a specific region of the input space, but the feature attributions are only applicable to one specific input instance and are therefore a local explanation. However, by looking at the feature attributions together with the states and actions for longer periods of time we can find patterns in the behaviour and explanations of the model.

7. CONCLUSION

We have compared different methods for building model trees used as post-hoc explanation methods for real-world, robotic tasks with multiple continuous inputs and outputs in terms of both accuracy and interpretability. It is crucial for the model trees to achieve high accuracy for the explanations to have any value but it is also important for the surrogate model to be simplistic so that the surrogate model is easy to analyse. We found that the methods benefit from being able to predict multiple outputs for a single tree, both in terms of interpretability and in being able to better grasp the properties of the model to be explained. Additionally, we find that it can be beneficial to introduce domain knowledge during the tree building process by determining which features can be used for splitting at different depths of the tree. Overall, simpler tree models with univariate splits, linear prediction functions, and multiple outputs per tree are favored because these are more sparse, which is favorable in terms of interpretability, but still capable of approximating even complex models. Future work includes investigating how the splitting conditions in the branch nodes and the overall structure of the tree can be used to produce more global explanations and/or other types of explanations to complement the feature attributions. The possibility of adding an uncertainty metric with the explanations based on how similar the output of the black-box model and the model tree also warrants further research.

ACKNOWLEDGEMENTS

An additional thanks to Dr. Daisy Zhuo from Interpretable AI for the support with ORT-LH.

REFERENCES


- [1] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.
- [2] G. Aglin, S. Nijssen, and P. Schaus. Learning optimal decision trees using caching branch-and-bound search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3146–3153, 2020.
- [3] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, Salvador ang Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.
- [4] F. Avellaneda. Efficient inference of optimal decision trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3195–3202, 2020.
- [5] D. Bertsimas and J. Dunn. *Machine learning under a modern optimization lens*. Dynamic Ideas LLC, 2019.
- [6] D. Bertsimas, J. Dunn, and Y. Wang. Near-optimal nonlinear regression trees. *Operations Research Letters*, 49:201–206, 03 2021.
- [7] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth, 1984.
- [8] I. Covert, S. Lundberg, and S.-I. Lee. Understanding global feature contributions with additive importance measures. *4th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- [9] J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.
- [10] V. B. Gjørørum, E. L. H. Rørøvik, and A. M. Lekkås. Approximating a deep reinforcement learning docking agent using linear model trees. *European Control Conference(ECC)*, 2021.

- [11] V. B. Gjørum, I. Strümke, O. A. Alsos, and A. M. Lekkas. Explaining a deep reinforcement learning docking agent using linear model trees and user adapted visualizations. *Journal for Marine Science and Engineering(JMSE)*, 8(9), 2021.
- [12] J. A. Glomsrud, A. Ødegårdstuen, A. L. S. Clair, and O. Smogeli. Trustworthy versus explainable AI in autonomous vessels. *International Seminar on Safety and Security of Autonomous Vessels(ISSAV)*, 2019.
- [13] R. Guidotti, A. Monreale, F. Giannotti, D. Pedreschi, S. Ruggieri, and F. Turini. Factual and counterfactual explanations for black box decision making. *IEEE Intelligent Systems*, 34(6):14–23, 2019.
- [14] T. Haaroja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *Robotics: Science and Systems (RSS)*, 2019.
- [15] T. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations(ICLR)*, 2016.
- [16] Z. C. Lipton. The myths of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *ACM Queue*, 16(3):31–57, 2018.
- [17] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems(NeurIPS)*, page 4768–4777, 2017.
- [18] J. Løver. Explaining a deep reinforcement learning agent using regression trees. MA thesis. Trondheim, Norway: Norwegian University of Science and Technology(NTNU), 2021.
- [19] J. Løver, V. B. Gjørum, and A. M. Lekkas. Explainable AI methods on a deep reinforcement learning agent for automatic docking. *IFAC-PapersOnLine*, 54(16):146–152, 2021. 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles (CAMS).
- [20] S. B. Remman and A. M. Lekkas. Robotic lever manipulation using hindsight experience replay and shapley additive explanations. *European Control Conference(ECC)*, 2021.
- [21] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, 2016.
- [22] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.
- [23] E.-L. H. Rørvik. Automatic docking of an autonomous surface vessel : Developed using deep reinforcement learning and analysed with Explainable AI. MA thesis. Trondheim, Norway: Norwegian University of Science and Technology(NTNU), 2020.
- [24] A. Schidler and S. Szeider. SAT-based decision tree learning for large data sets. *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, (5):3904–3912, 2021.
- [25] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on learning representation(ICLR)*, 2016.
- [26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [27] A. Silva, T. Killian, I. D. J. Rodriguez, S.-H. Son, and M. Gombolay. Optimization methods for interpretable differentiable decision trees in reinforcement learning. *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics(AISTATS)*, 2020.
- [28] L. Y. Singh, C. F. K. Hartikainen, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *Robotics: Science and Systems(RSS)*, 2019.
- [29] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, page 3319–3328, 2017.
- [30] J. van der Waa, M. Robeer, J. van Diggelen, M. Brinkhuis, and M. Neerinx. Contrastive explanations with local foil trees. *arXiv preprint arXiv:1806.07470*, 2018.
- [31] H. Verhaeghe, S. Nijssen, G. Pesant, C.-G. Quimper, and P. Schaus. Learning optimal decision trees using constraint programming. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4765–4769, 2020.

- [32] S. Verwer and Y. Zhang. Learning optimal classification trees using a binary linear program formulation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1625–1632, Jul. 2019.

5.5 Paper E

Postprint of [20]: **Vilde B. Gjørum**, Inga Strümke, Anastasios M. Lekkas, and Timothy Miller, "Real-Time Counterfactual Explanations For Robotic Systems With Multiple Continuous Outputs". Accepted to: *The 22nd World Congress of the International Federation of Automatic Control (IFAC WC)* (2023) doi: <https://doi.org/10.48550/arXiv.2212.04212>

©2021 Vilde B. Gjørum, Inga Strümke, Anastasios M. Lekkas, and Timothy Miller. Reprinted and formatted to fit the thesis under the terms of the Creative Commons Attribution License 

Real-Time Counterfactual Explanations For Robotic Systems With Multiple Continuous Outputs[★]

Vilde B. Gjørum^{*} Inga Strømke^{**} Anastasios M. Lekkas^{*} Timothy Miller^{***}

^{*} *Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway, (e-mail: vilde.gjarum, anastasios.lekkas ,@ntnu.no).*

^{**} *Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway, (e-mail: inga.strumke@ntnu.no)*

^{***} *School of Computing and Information Systems, University of Melbourne, Melbourne, Australia(e-mail: tmiller@unimelb.edu.aus)*

Abstract: Although many machine learning methods, especially from the field of deep learning, have been instrumental in addressing challenges within robotic applications, we cannot take full advantage of such methods before these can provide performance and safety guarantees. The lack of trust that impedes the use of these methods mainly stems from a lack of human understanding of what exactly machine learning models have learned, and how robust their behaviour is. This is the problem the field of explainable artificial intelligence aims to solve. Based on insights from the social sciences, we know that humans prefer contrastive explanations, i.e. explanations answering the hypothetical question “*what if?*”. In this paper, we show that linear model trees are capable of producing answers to such questions, so-called *counterfactual explanations*, for robotic systems, including in the case of multiple, continuous inputs and outputs. We demonstrate the use of this method to produce counterfactual explanations for two robotic applications. Additionally, we explore the issue of infeasibility, which is of particular interest in systems governed by the laws of physics.

Keywords: Explicability and transparency in Cyber-physical and human systems, Reinforcement learning and deep learning in control, data-driven control, autonomous robotic systems, explainable artificial intelligence for robotics, counterfactual explanations for robotic systems

^{*} This work was supported by the Research Council of Norway through the EXAIGON project, project number 304843.

1. INTRODUCTION

Artificial intelligence (AI) has shown to be useful for robotics, control, and autonomous systems in several ways, for example by deep learning (DL) boosting the performance of image processing and thus also giving robots better perception. Reinforcement learning (RL) is a subfield of machine learning (ML) where the agent learns through exploring its environment, and through trial and error improving its strategy. One of the main benefits of using RL is that we do not need to label every prediction as right or wrong but can rather encourage desired behaviour through a reward function describing the wanted outcome of the agent's behaviour. Deep RL (DRL) is well-suited for problems where the system and/or the environment cannot be modelled accurately making it a good approach for many challenging control problems. There are numerous examples of RL-agents successfully performing robotic tasks (Remman and Lekkas, 2021; Martinsen and Lekkas, 2018; Haarnoja et al., 2019; Lillicrap et al., 2016). When a model is so complex that humans no longer can understand how they make their decisions or predictions we call it a *black-box model*. For AI-methods of black-box nature, such as deep neural networks, to reach their full potential in applications with real-life risk associated we need to better understand their inner workings. By using explainable artificial intelligence (XAI)-methods we can get increased trust, better knowledge, improved control, and the ability to justify decisions and predictions (Adadi and Berrada, 2018). The fields of robotics and control have some additional requirements for the explanations given by XAI-methods, such as:

- explanations to be used under live operations must be given in real-time,
- the methods must be able to handle large, continuous input and output spaces
- the methods must be able to handle large datasets.

As seen in Løver et al. (2021), these are requirements that some XAI-methods struggle to meet. In Gjørnum et al. (2021a) it is shown that linear model trees (LMTs), a decision tree (DT) with linear functions in the leaf nodes, are well-suited for acting as a post-hoc, explanation method for a RL-agent performing docking of a surface vessel. In Gjørnum et al. (2021a), the LMT gave explanations in the form of feature attributions that states a direct mapping from the inputs to the outputs. Another popular form of explanation is counterfactual explanations (CFEs). CFEs answers the hypothetical question “*but what if (the state was different)?*” or the more specific question “*why did you do this instead of that?*”. A commonly used example for showing the utility value of counterfactual explanations is the case of someone getting a loan application rejected, and the importance of giving an *actionable* explanation is emphasized. For example, “*if you had more savings your loan application would have been accepted* is an actionable explanation since this is something that can be changed, whereas “*if you were 10 years younger your loan application would have been accepted* is not an actionable explanation. Similar examples can be found in other fields, such

as job and university applications, predicting the risk of disease in the future, or disbursing government aid (Mothilal et al., 2020). However, this notion of actionable explanations does not apply to the field of robotics as it does not make sense to change the input features(the state). Instead, the focus should be more on the *feasibility* of the counterfactual explanations, in the sense of whether or not the counterfactual state and the counterfactual action are physically possible (Guidotti, 2022). In Gjørnum et al. (2021a), it is shown that LMTs are capable of giving explanations in the form of feature attributions in real time for robotic applications. LMTs are decision trees where the constant prediction in the leaf nodes is replaced by a linear function. Since the branch node and its splitting conditions divide the input space into different regions, and each region has a corresponding linear function, the LMTs constitute a piece-wise linear function. If the LMT has accurately enough approximated black-box model, such as a Neural network (NN), it can be used as a post-hoc explanation method. In Carreira-Perpiñán and Hada (2021), an exact method for computing CFEs for classification trees with both univariate and multivariate splits in the branch nodes is proposed. Regression trees with univariate splitting conditions are solved directly by exploiting the fact that the problem is separable within the leaf nodes, meaning that solving the problem in every leaf node and then choosing the leaf node with the best CFE solves the problem globally. For the classification trees with multivariate splits, the problem of finding CFEs are expressed as a mixed-integer problem and the solution is found by using linear or quadratic problem solvers depending on whether the objective function is expressed as a linear or quadratic problem. A model-agnostic method for finding CFEs is presented in Karimi et al. (2019). The problem of finding CFEs is expressed as a sequence of *satisfiability*(SAT) problem. Neither Carreira-Perpiñán and Hada (2021) nor Karimi et al. (2019) are applicable to regression problems with multiple outputs and thus not applicable to most robotic applications. In Sokol and Flach (2019), CFEs are found using a *leaf-to-leaf counterfactual distance matrix* describing how much an instance belonging to a leaf node would have to change to belong in another leaf node. However, the method proposed is for classification trees and details for implementation lacks. Both Local Rule-Based Explainer (LORE) Guidotti et al. (2019) and FOILTREE van der Waa et al. (2018) build a DT locally around the instance to be explained and searches through the DT for CFEs but neither apply to robotic applications since they do not apply to regression problems with multiple outputs. Another problem with methods that builds a new local interpretable model for every explanation is that they are often not fast enough to be used in real-time (Løver et al., 2021). In this paper, we present an algorithm for finding counterfactual explanations from an LMT and show that this method is applicable to robotic applications.

The paper’s main contributions are as follows:

- Algorithm for getting counterfactual explanations in real-time for a black-box model through an LMT serving as a surrogate model.

- Using the algorithm on two robotic applications, one with singular, continuous outputs and one with multiple, continuous outputs.
- Identify the issue of infeasibility that arises when producing CFEs for real-world problems where the laws of physics apply.
- Suggesting remedies for avoiding infeasible counterfactuals.

The paper is structured as follows. In Section 2.1, LMTs will be presented, followed by an introduction to CFEs in Section 2.3. In Section 3, we will present how LMTs can be used to find CFEs. In Section 5, the results are presented and the discussion is given in Section 6.

2. BACKGROUND

In this section, the necessary background will be introduced. First, the LMTs will be introduced in Section 2.1 followed by the two applications we test the LMTs on in Section 2.2. Finally, the CFEs will be introduced in Section 2.3.

2.1 Linear Model Trees

DTs consist of branch nodes and leaf nodes. The branch nodes have a univariate splitting condition, meaning they split the data based on whether or not a certain input feature is smaller or bigger than a threshold. In this way, the branch nodes split the input space into distinct regions and each region has a corresponding leaf node. LMTs are decision trees with linear functions in the leaf nodes and LMTs are piece-wise linear function approximators. As presented in Gjærnum et al. (2021b), several methods for building LMTs for robotic applications exist but for this work LMTs trained with the method presented in Gjærnum et al. (2021a) will be used.

2.2 Test applications

In this section, we introduce the two robotic applications for which the LMT produced counterfactual explanations.

Pendulum: The inverted pendulum¹ is a classic control theory problem where the goal is to balance the pendulum in the upright position by applying force to the free end of the pendulum. The pendulum has three states, namely the angular velocity $\dot{\theta}$ and the coordinates of the free end of the pendulum x and y . The force applied to the free end of the pendulum is continuous and is the only action for this environment. A NN has been trained with the RL-method Proximal Policy Optimization (PPO) (Schulman et al., 2017) to balance the pendulum.

¹ See https://www.gymnasium.ml/environments/classic_control/pendulum/

Docking environment: The docking agent and the harbour environment are thoroughly presented in Gjørnum et al. (2021a). The vessel has eight input features describing the vessel’s relative position and velocity in the harbour. The vessel has three thrusters, one tunnel thruster at the front and two azimuth thrusters at the back. The tunnel thruster is controlled by setting the force of the thruster, while the azimuth thrusters are controlled by setting the force and angle of the thruster. The NN that performs the docking was trained by PPO as originally presented in Rørvik (2020).

2.3 Counterfactual explanations

Definition 1. (Counterfactual explanation (Guidotti, 2022)). Given a classifier b that outputs the decision $y = b(x)$ for an instance x , a counterfactual explanation consists of an instance x' such that the decision for b on x' is different from y , i.e., $b(x') \neq y$, and such that the difference between x and x' is minimal.

Following this definition, a CFE can be formulated as:
If state s had been Δs different, the corresponding action a would have been Δa different.

The most frequently asked contrastive question that applies to robotic applications is “*Why is action A used in state S, rather than action B?*” (Krarup et al., 2021). The classifier b in Definition 1 is usually a black-box model such as a deep neural network, and we are looking for another instance with as similar input features as our original instance but with a different output. The meaning of a different output is straightforward for classification problems as the counterfactual example will be the closest instance (based on some distance metric on the input features) with a different class as output. This is more challenging for regression problems since it is not as clear what the counterfactual action is since the meaning of *different enough output* is context-dependent. Counterfactual explanations for problems with multiple, continuous input and output features can be defined as follows.

Definition 2. (Cont. counterfactual explanation). Given a predictor b that outputs a continuous prediction of dimension n , $y = b(x)$, for a continuous instance x of dimension m , a counterfactual explanation consists of an instance x' such that the distance from b ’s prediction on x' from y is maximal, i.e., $b(x') \neq y$, and such that the difference between x and x' is minimal.

3. METHOD

CFEs can be found by using an LMT as shown in Alg. Algorithm 1 by ordering the leaf nodes from closest to furthest and then solving an optimization problem within that region instead of over the entire state space. To ensure the *correctness* of the explanations, the LMT is only used to locate the counterfactual example and the black-box is used when formulating the explanation.

We make the following two main assumptions:

Algorithm 1 CFEs from LMTs

Require:

x : instance to be explained
 num_exp : number of explanations wanted
 $constraints$: known constraints for input and output
 $ordered_leaf_nodes$: the trees leaf nodes ordered from closest to furthest relative to the leaf node x belongs to
 f : Objective function

$i = 0$

while $i < num_exp$ **do:**

$leaf_node \leftarrow ordered_leaf_nodes[i]$

$constraints \leftarrow$ boundaries of $leaf_node$

$x' \leftarrow$ minimize f subject to $constraints$

$y' \leftarrow$ black_box(x')

end while

- (1) We assume that the LMT is successfully trained and thus has approximated the black-box model with sufficient accuracy (Gjærum et al., 2021b).
- (2) We assume that the tree has placed leaf nodes of regions that are similar in vicinity of each other.

3.1 Leaf node ordering

To avoid searching through the entire state space for counterfactual explanations, we exploit the fact that the LMT has already split the state space into regions. We want to order the regions by how close they are to the instance to be explained. Since each region corresponds to a specific leaf node we can do this by ordering the leaf nodes, which again can be done by looking at the structure of the tree. This ordering of leaf nodes can be found by counting how many branch nodes must be traversed to get from the instance to every other leaf node. An example of how the leaf nodes would be ordered is shown in Figure 1 and the algorithm for ordering the leaf nodes is presented in Algorithm 2.

3.2 Counterfactual explanations from optimization

Given that we already have the leaf nodes ordered from closest to furthest relative to the leaf node the instance to be explained x belongs to, the counterfactual example can be found by solving the optimization problem given in 1.

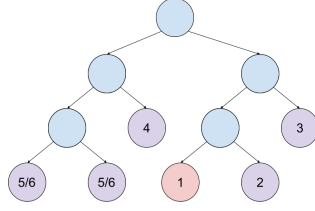


Fig. 1. Illustration of how the leaf nodes will be ordered if the instance to be explained belongs to the red leaf node.

Algorithm 2 Leaf node ordering

Require: instance x to be explained

$\mathcal{L}_c \leftarrow$ the leaf node x belongs to $\triangleright \mathcal{L}_c$ is the current leaf node

$LNO \leftarrow \mathcal{L}_c$ \triangleright List containing the Leaf Node Order

$\mathcal{L}_c \leftarrow \text{Parent}(\mathcal{L}_c)$

while \mathcal{L}_c is not the root node **do**

Traverse the subtree starting for \mathcal{L}_c

Add leaf nodes to LNO in the order they're found.

$\mathcal{L}_c \leftarrow \text{Parent}(\mathcal{L}_c)$

end while

$$\begin{aligned}
 \min_{x'} z = & |x - x'| - (y - y')^2 - \text{sparsity}(x') \\
 \text{s.t.} \quad & x'_{j_i} \leq t_i, & i \in \mathcal{P}_{left} \\
 & x'_{j_i} > t_i, & i \in \mathcal{P}_{right} \\
 & x'_{j_i} > t_i, & i \in \mathcal{B}_{upper}^{input} \\
 & x'_{j_i} < t_i, & i \in \mathcal{B}_{lower}^{input} \\
 & y'_{j_i} > t_i, & i \in \mathcal{B}_{upper}^{output} \\
 & y'_{j_i} > t_i, & i \in \mathcal{B}_{lower}^{output} \\
 & y' = f_l(x')
 \end{aligned} \tag{1}$$

where x' is the counterfactual example and its corresponding output y' . The left and right parent nodes along the path from the root node to the leaf node are denoted by $\mathcal{P}_{left/right}$, while t_i is the threshold used in the splitting condition on input feature j . The lower and upper boundaries on the input are given by $\mathcal{B}_{lower/upper}^{input}$, while the lower and upper boundaries on the output are given by $\mathcal{B}_{lower/upper}^{output}$. The objective function is given by z in 1 and the expression consists of three parts:

- $|x - x'|$: The distance between the input values of the instance to be explained and the counterfactual example should be minimized
- $(Y - y')^2$: The distance between the output value of the instance to be explained and the counterfactual example should be maximized
- $\text{sparsity}(x')$ and $\text{sparsity}(y')$: Simple explanations are preferred and thus it is better with explanations where a few features have been changed a lot rather than many features changed slightly.

For different applications, it may be beneficial to change these metrics with for example different distance functions or different sparsity measurements.

Requesting specific explanations Given the objective function in 1, we are searching for the counterfactual problem that balances finding an example with a as small as possible change in input features while having a as big as possible change in the output feature. However, if a specific explanation is requested, such as “*Why was the action y taken instead of action Y ?*”, we are no longer searching for a counterfactual example in general but rather the counterfactual example with the smallest change in the input and output as close as possible to Y . This specific counterfactual can be found by using

$$z = |x - x'| - (Y - y')^2 - \text{sparsity}(x') - \text{sparsity}(y'), \quad (2)$$

where

$$\text{sparsity}(x') = |x - x'|_0, \quad (3)$$

and

$$\text{sparsity}(y') = |y - y'|_0. \quad (4)$$

3.3 Pipeline

How the system, the NN, and the LMT are connected is illustrated in Figure 2. For both the case of the pendulum and the docking agent the controller is a NN but since the LMT is a model-agnostic method any type of model can be used to control the system. The NN gets the state from the system and returns the action for that state. The LMT receives the same state for the system and calculates what the counterfactual state is, gives this counterfactual state to the NN and the NN returns the counterfactual action. By combining the counterfactual state and action we get the counterfactual explanation. By doing it this way we are certain that the counterfactual explanation is true because we are sure that this is the action the NN would have taken given the state found by the LMT.

4. INFEASIBLE EXPLANATIONS

Given a black-box controlling a robot operating in the real world, we argue that explanations must make sense from a physical point of view. However, this may contradict the desire for a sparse explanation. Take the inverted pendulum as an example. Following the desire for sparse explanations, a counterfactual example where only one of x or y is changed is preferred. However, since the

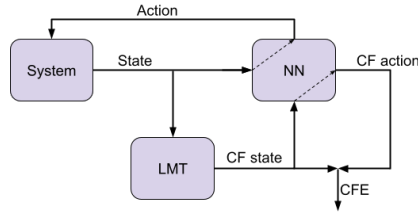


Fig. 2. Pipeline describing how the system, the NN, and the LMT works together.

free end of the pendulum always will be at a point along a circle with a radius the length of the pendulum, thus, with only one exception, all counterfactual examples changing only one of the two coordinates will be physically impossible and thus make no sense. We address this through feature engineering as presented in Sec. Section 4.1, adding constraints to the optimization problem as presented in Section 4.2.

4.1 Feature engineering

One way of ensuring that the tree does not return infeasible counterfactuals is by making sure that the features, both input and output, are not dependent. This can be achieved using feature engineering to find new features that are independent but represent the original problem (or an approximation of it).

In the case of the pendulum problem, this could be done by using θ and $\dot{\theta}$ as input features instead of x, y , and $\dot{\theta}$.

4.2 Adding constraints

If the relationship between the dependent features is known and can be formalized as a function they can be added to the optimization solver. It is important to note that the constraint functions form affects which optimization solvers can be used. For the case of the pendulum, the following constraint can be added:

$$L = x^2 + y^2, \quad (5)$$

where L is the length of the pendulum, and x and y the position of the end of the pendulum.

5. RESULTS

In this section, we show that the LMTs are capable of giving counterfactual explanations for robotic systems with both singular and multiple continuous inputs and outputs in real time. For both the inverted pendulum environment and the docking environment a NN is trained to perform the respective tasks, and an LMT is built to approximate the NNs. The LMT approximating the

Application	Average time
Pendulum	0.21 s
Docking env.	0.036 s

Table 1. Average time for computing the counterfactual explanation for each of the test applications.

docking agent was trained and presented in (Gjærum et al., 2021a), while the LMT and the NN for the pendulum were built for this paper. In Table 1, the average time it took to compute the counterfactual explanations for 250 different states on an Intel $\text{\textcircled{R}}$ CoreTMi9-9980HK CPU @ 2.40GHz. The explanations can be computed within a quarter of a second which is faster than humans can interpret the explanations and the explanations are thus suitable for use in real-time. The LMT giving explanations for the pendulum-agent has 220 leaf nodes, while the LMT giving explanations for the docking agent has 312 leaf nodes. The most time consuming part of the algorithm is the ordering of all the leaf nodes. This can be speeded up by only ordering the a limited number of the closest leaf nodes instead of all of the leaf nodes. This is especially helpful for large trees. Despite the LMT for the docking problem being larger and dealing with more inputs and outputs than the LMT for the pendulum problem, the LMT for the docking problem finds the counterfactual explanations faster than the LMT for the pendulum problem. This is due to the fact that the docking problem is a more complex problem with faster changing outputs than the pendulum problem, and it is thus easier to find different enough outputs.

In Figure 3a, a counterfactual explanation for the inverted pendulum is shown. The thick, red line is the position of the pendulum, while the black line is the position of the pendulum in the counterfactual state. The torque applied to the pendulum by the NN is shown with a red, circular line while the counterfactual action is shown with a black, circular line.

In Figure 3b, an example of an infeasible counterfactual explanation is shown. This explanation is infeasible because the counterfactual state is infeasible. In fact, when building an LMT using the features θ and $\dot{\theta}$ instead of x, y , and \dot{x}, \dot{y} , as discussed in Section 4.1, we find that the LMT always gives feasible counterfactuals. This can be seen in Figure 3c, where a feasible counterfactual explanation for the same state as in Figure 3b is shown.

In Figure 4, a counterfactual explanation for the docking agent is shown. Since the vessel has so many input and output features we found it easier to express the counterfactual explanation in a table rather than in words. Still, the explanation could be communicated faster with appropriate visualizations.

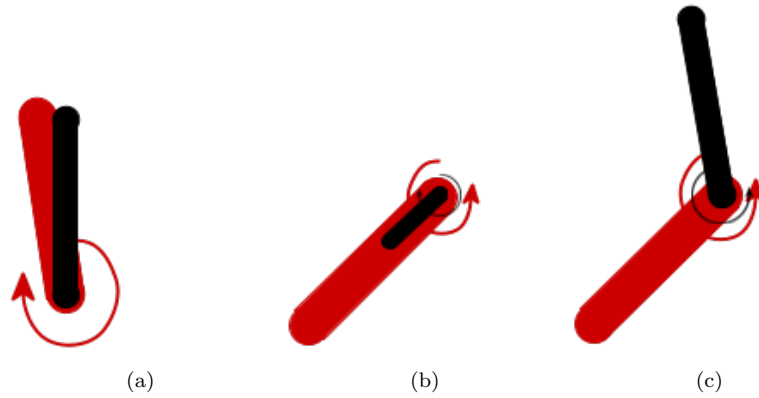


Fig. 3. The pendulums state is given by the red rod and the torque applied by the black-box model is shown in red. The counterfactual explanation is given by the counterfactual state (the black rod) and the counterfactual action (black torque).
 (a) shows a feasible counterfactual explanation,
 (b) shows an infeasible counterfactual explanation,
 and
 (c) shows the corresponding feasible counterfactual explanation found by using feature engineering.



Fig. 4. The table shows the counterfactual explanation by stating how much the state and the action would change in the counterfactual example for the situation given by the vessels point of view(top left) and how the vessel is situated in the harbour (right).

6. DISCUSSION

We have shown that LMTs are suitable for generating counterfactual explanations even for complex robotic systems with multiple, continuous outputs. Which counterfactual explanation is found is determined solely by 1, and the tuning

of this function is crucial because it determines the trade-off between a large distance in the output, a small distance in the input and the sparsity of the explanation. Since the LMT only identifies the counterfactual state, while the NN is used to complete the counterfactual explanation by finding the counterfactual action, we know that the counterfactual explanation is necessarily true. However, we do not know whether there exist better (in terms of distance in input and output) counterfactual explanations that could have been found by using another cost function. The CFEs could also say something about the agents or the environments stability around a certain point by looking at how far (in the input space) we have to look before finding a significantly different output. Actions that are changing a lot, even when the state is just slightly different, can be due to either the agent being in an especially complex region or that the agents behaviour is unstable. As shown for the pendulum case, the LMT may find infeasible states if the input features are not independent. One way of handling this problem is by performing feature engineering on the input features so that they become independent. This approach was successful in the case of the pendulum but would be significantly harder in a more complex environment, as is the case for the docking agent. In some cases, not all the relevant dependencies are defined or even known. If they are known, they can be added to the optimization problem as constraints, which again can make the optimization problem harder to solve. Evaluating the usefulness of an explanation is difficult because explanations are subjective, and different recipients prefer different types of explanations. Additionally, how an explanation is communicated is sometimes as important as the explanation itself, especially for complex systems. Therefore, future work should include investigating both to what extent these explanations can improve the understanding of the system and how to communicate these explanations most effectively. The usefulness of infeasible explanations should also be investigated.

REFERENCES

- Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6, 52138–52160. doi: 10.1109/ACCESS.2018.2870052.
- Carreira-Perpiñán, M.a. and Hada, S.S. (2021). Counterfactual explanations for oblique decision trees: Exact, efficient algorithms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8), 6903–6911.
- Gjørnum, V.B., Strümke, I., Alsos, O.A., and Lekkas, A.M. (2021a). Explaining a deep reinforcement learning docking agent using linear model trees and user adapted visualizations. *Journal for Marine Science and Engineering(JMSE)*, 8(9).
- Gjørnum, V.B., Strümke, I., Løver, J., Miller, T., and Lekkas, A.M. (2021b). Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications. *Submitted to Neurocomputing*.

- Guidotti, R. (2022). Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*, 1–55. doi:10.1007/s10618-022-00831-6.
- Guidotti, R., Monreale, A., Giannotti, F., Pedreschi, D., Ruggieri, S., and Turini, F. (2019). Factual and counterfactual explanations for black box decision making. *IEEE Intelligent Systems*, 34(6), 14–23. doi:10.1109/MIS.2019.2957223.
- Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. (2019). Learning to walk via deep reinforcement learning. *Robotics: Science and Systems (RSS)*.
- Karimi, A.H., Barthe, G., Belle, B., and Valera, I. (2019). Model-agnostic counterfactual explanations for consequential decisions. *The 22nd International Conference on Artificial Intelligence and Statistics*.
- Krärup, B., Krivic, S., Magazzeni, D., Long, D., Cashmore, M., and Smith, D.E. (2021). Contrastive explanations of plans through model restrictions. *Journal of Artificial Intelligence Research*, 72, 533–612.
- Lillicrap, T., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations (ICLR)*.
- Løver, J., Gjørnum, V.B., and Lekkas, A.M. (2021). Explainable ai methods on a deep reinforcement learning agent for automatic docking**this work was supported by the research council of norway through the exaigon project, project number 304843. *IFAC-PapersOnLine*, 54(16), 146–152. doi:https://doi.org/10.1016/j.ifacol.2021.10.086. URL <https://www.sciencedirect.com/science/article/pii/S2405896321014889>. 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2021.
- Martinsen, A.B. and Lekkas, A.M. (2018). "curved-path following with deep reinforcement learning: Results from three vessel models. *OCEANS MTS/IEEE*.
- Mothilal, R.K., Sharma, A., and Tan, C. (2020). Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, FAT* '20*, 607–617. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3351095.3372850. URL <https://doi.org/10.1145/3351095.3372850>.
- Remman, S.B. and Lekkas, A.M. (2021). Robotic lever manipulation using hindsight experience replay and shapley additive explanations. *European Control Conference (ECC)*.
- Rørvik, E.L.H. (2020). Automatic docking of an autonomous surface vessel : Developed using deep reinforcement learning and analysed with Explainable AI. MA thesis. Trondheim, Norway: Norwegian University of Science and Technology (NTNU). URL <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2656724>.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Sokol, K. and Flach, P. (2019). Desiderata for interpretability: Explaining decision tree predictions with counterfactuals. *Proceedings of the AAAI Conference*

on Artificial Intelligence, 33, 10035–10036. doi:<https://doi.org/10.1609/aaai.v33i01.330110035>.

van der Waa, J., Robeer, M., van Diggelen, J., Brinkhuis, M., and Neerincx, M. (2018). Contrastive explanations with local foil trees. doi:10.48550/ARXIV.1806.07470. URL <https://arxiv.org/abs/1806.07470>.

Bibliography

- [1] A. Adadi and M. Berrada. ‘Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)’. In: *IEEE Access* 6 (2018), pp. 52138–52160. DOI: 10.1109/ACCESS.2018.2870052.
- [2] A. Anand et al. ‘Safe Learning for Control using Control Lyapunov Functions and Control Barrier Functions: A Review’. In: *Procedia Computer Science, Proceedings of the 25th International Conference on Knowledge Based and Intelligent Information and Engineering Systems(KES)* 192 (2021), pp. 3987–3997. DOI: 10.1016/j.procs.2021.09.173.
- [3] E. Anderlini, G. Parker and G. Thomas. ‘Docking Control of an Autonomous Underwater Vehicle Using Reinforcement Learning’. In: *Applied Sciences* 9.3456 (2019).
- [4] Vijay Arya et al. *One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques*. 2019. URL: <https://arxiv.org/abs/1909.03012>.
- [5] Sebastian Bach et al. ‘On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation’. In: *PloS one* 10.7 (2015). DOI: DOI:10.1371/journal.pone.0130140.
- [6] Bowen Baker et al. *Emergent Tool Use From Multi-Agent Autocurricula*. 2020. arXiv: 1909.07528 [cs.LG].
- [7] Dimitri Bertsimas and Jack Dunn. *Machine learning under a modern optimization lens*. Dynamic Ideas LLC, 2019. URL: <https://books.google.no/books?id=g3ZWygEACAAJ>.
- [8] Tolga Bolukbasi et al. ‘Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings’. In: *The 30th Conference on Neural Information Processing Systems (NeurIPS)* (2016).
- [9] L. Breiman, J. Friedman and C. Olshen. R. Stone. ‘Classification and Regression Trees’. In: *Wadsworth* (1984).
- [10] Leonard A. Breslow and David W. Aha. ‘Simplifying decision trees: A survey’. In: *The Knowledge Engineering Review* 12.01 (1997), pp. 1–40. DOI: 10.1017/S0269888997000015.
- [11] Joy Buolamwini and Timnit Gebru. ‘Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification’. In: *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*. Ed. by Sorelle A. Friedler and Christo Wilson. Vol. 81. Proceedings of Machine Learning Research. PMLR, Feb. 2018, pp. 77–91. URL: <https://proceedings.mlr.press/v81/buolamwini18a.html>.

-
- [12] Nicolas Blystad Carbone. ‘Explainable AI for path following with Model Trees’. MA thesis. Norwegian University of Science and Technology (NTNU), 2020.
- [13] ‘Causal Shapley Values: Exploiting Causal Knowledge to Explain Individual Predictions of Complex Models’. In: *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)* (2020).
- [14] Ian Covert, Scott Lundberg and Su-In Lee. *Understanding Global Feature Contributions With Additive Importance Measures*. 2020. DOI: 10.48550/ARXIV.2004.00668. URL: <https://arxiv.org/abs/2004.00668>.
- [15] Francisco Cruz et al. ‘Explainable robotic systems: understanding goal-driven actions in a reinforcement learning scenario’. In: *Neural Computing and Applications* (2021). DOI: <https://doi.org/10.1007/s00521-021-06425-5>.
- [16] Nathan Douglas et al. ‘Towers of Saliency: A Reinforcement Learning Visualization Using Immersive Environments’. In: *Proceedings of the 14th ACM International Conference on Interactive Surfaces and Spaces (ISS)* (2019).
- [17] Vilde B. Gjærums, Ella-Lovise H. Rørvik and Anastasios M. Lekkas. ‘Approximating a deep reinforcement learning docking agent using linear model trees’. In: *2021 European Control Conference (ECC)*. 2021, pp. 1465–1471. DOI: 10.23919/ECC54610.2021.9655007.
- [18] Vilde B. Gjærums et al. ‘Explaining a deep reinforcement learning docking agent using linear model trees and user adapted visualizations’. In: *Journal for Marine Science and Engineering* 9.1178 (2021). DOI: <https://doi.org/10.3390/jmse9111178>.
- [19] Vilde B. Gjærums et al. ‘Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications’. In: *Neurocomputing* 515 (2022), pp. 133–144. DOI: <https://doi.org/10.1016/j.neucom.2022.10.014>.
- [20] Vilde B. Gjærums et al. ‘Real-Time Counterfactual Explanations For Robotic Systems With Multiple Continuous Outputs’. In: *Submitted to the 22nd World Congress of the International Federation of Automatic Control (IFAC WC)* (2023).
- [21] Hila Gonen and Yoav Goldberg. ‘Lipstick on a Pig: Debiasing Methods Cover up Systematic Gender Biases in Word Embeddings But do not Remove Them’. In: *North American Chapter of the Association for Computational Linguistics (NAACL)* (2019).
- [22] Riccardo Guidotti. ‘Counterfactual explanations and how to find them: literature review and benchmarking’. In: *Data Mining and Knowledge Discovery* (Apr. 2022), pp. 1–55. DOI: 10.1007/s10618-022-00831-6.

-
- [23] David Gunning and D. W. Aha. ‘DARPA’s Explainable Artificial Intelligence (XAI) Program’. In: *AI Magazine* 40.2 (2019), pp. 44–58. DOI: <https://doi.org/10.1609/aimag.v40i2.2850>.
- [24] T. Haarnoja et al. ‘Learning to walk via deep reinforcement learning’. In: *Robotics: Science and Systems (RSS)* (2019).
- [25] A. Håkansson et al. ‘Robust Reasoning for Autonomous Cyber-Physical Systems in Dynamic Environments’. In: *Procedia Computer Science, Proceedings of the 25th International Conference on Knowledge Based and Intelligent Information and Engineering Systems(KES)* 192 (2021), pp. 3966–3978. DOI: 10.1016/j.procs.2021.09.171.
- [26] Lei He, Nabil Aouf and Bifeng Song. ‘Explainable Deep Reinforcement Learning for UAV autonomous path planning’. In: *Aerospace Science and Technology* 118 (2021), p. 107052. ISSN: 1270-9638. DOI: <https://doi.org/10.1016/j.ast.2021.107052>.
- [27] Thomas Hickling et al. ‘Explainability in Deep Reinforcement Learning, a Review into Current Methods and Applications’. In: *arXiv:2207.01911* (2022).
- [28] ‘Hindsight Experience Replay’. In: *Proceedings of the 31st Conference on Neural Information Processings Systems (NeurIPS)* 30 (2017). URL: <https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf>.
- [29] Ho-Taek Joo and Kyung-Joong Kim. ‘Visualization of Deep Reinforcement Learning using Grad-CAM: How AI Plays Atari Games?’ In: *IEEE Conference on Games (CoG)*. 2019, pp. 1–2. DOI: 10.1109/CIG.2019.8847950.
- [30] Been Kim et al. ‘Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)’. In: *International Conference on Machine Learning(ICML)*. 2018.
- [31] O. Kotevska et al. ‘Methodology for Interpretable Reinforcement Learning Model for HVAC Energy Control’. In: *IEEE International Conference on Big Data*. 2020, pp. 1555–1564. DOI: 10.1109/BigData50022.2020.9377735.
- [32] Oscar Li et al. ‘Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network That Explains Its Predictions’. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI’18/IAAI’18/EAAI’18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.
- [33] T. Lillicrap et al. ‘Continuous control with deep reinforcement learning’. In: *4th International Conference on Learning Representations(ICLR)* (2016).
-

-
- [34] Jakob Løver. ‘Explaining a Deep Reinforcement Learning Agent Using Regression Trees’. MA thesis. Norwegian University of Science and Technology (NTNU), 2021.
- [35] Jakob Løver, Vilde B. Gjørnum and A. M. Lekkas. ‘Explainable AI methods on a deep reinforcement learning agent for automatic docking’. In: *14th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles (CAMS)* (2021).
- [36] Scott M. Lundberg and Su-In Lee. ‘A Unified Approach to Interpreting Model Predictions’. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NeurIPS’17. Curran Associates Inc., 2017, pp. 4768–4777. ISBN: 9781510860964.
- [37] Scott M. Lundberg et al. ‘From local explanations to global understanding with explainable AI for trees’. In: *Nature Machine Intelligence* 2 (2020), pp. 56–67.
- [38] A.B. Martinsen and A.M. Lekkas. ‘”Curved-path following with deep reinforcement learning: Results from three vessel models.’ In: *OCEANS MTS/IEEE* (2018).
- [39] A.B. Martinsen and A.M. Lekkas. ‘Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning’. In: *IFAC-PapersOnLine* 51(29) (2018), pp. 329–334.
- [40] Ninareh Mehrabi et al. ‘A Survey on Bias and Fairness in Machine Learning’. In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3457607. URL: <https://doi.org/10.1145/3457607>.
- [41] E. Meyer, A. Rasheed A. Heiberg and O. San. ‘COLREG-compliant collision avoidance for unmanned surface vehicle using deep reinforcement learning’. In: *IEEE Access* 8 (2020), pp. 165344–165364.
- [42] Tim Miller. ‘Explanation in artificial intelligence: Insights from the social sciences’. In: *Artificial Intelligence* 267 (2019), pp. 1–38. DOI: <https://doi.org/10.1016/j.artint.2018.07.007>.
- [43] V. Mnih et al. ‘Human-level control through deep reinforcement learning’. In: *Nature* 518 (2015), pp. 529–533. DOI: <https://doi.org/10.1038/nature14236>.
- [44] Kevin P. Murphy. *Machine Learning: A probabilistic perspective*. The MIT press, 2012.
- [45] OpenAI. *OpenAI Five*. <https://blog.openai.com/openai-five/>. 2018.
- [46] OpenAI et al. *Learning Dexterous In-Hand Manipulation*. 2018. DOI: 10.48550/ARXIV.1808.00177. URL: <https://arxiv.org/abs/1808.00177>.
-

-
- [47] Ronilo J. Ragodos et al. *ProtoX: Explaining a Reinforcement Learning Agent via Prototyping*. 2022. DOI: 10.48550/ARXIV.2211.03162. URL: <https://arxiv.org/abs/2211.03162>.
- [48] Iyad Rahwan et al. ‘Machine Behaviour’. In: *Nature* 568.7753 (2019), pp. 477–486. DOI: 10.1038/s41586-019-1138-y.
- [49] Sindre Benjamin Remman and Anastasios M. Lekkas. ‘Robotic Lever Manipulation using Hindsight Experience Replay and Shapley Additive Explanations’. In: *European Control Conference(ECC)* (2021). DOI: 10.23919/ACC53348.2022.9867807.
- [50] Sindre Benjamin Remman, Inga Strümke and Anastasios M. Lekkas. ‘Causal versus Marginal Shapley Values for Robotic Lever Manipulation Controlled using Deep Reinforcement Learning’. In: *American Control Conference(ACC)* (2021).
- [51] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778.
- [52] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. ‘Anchors: High-Precision Model-Agnostic Explanations’. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* 32.1 (2018).
- [53] Stefano Giovanni Rizzo, Giovanna Vantini and Sanjay Chawla. ‘Reinforcement Learning with Explainability for Traffic Signal Control’. In: *IEEE Intelligent Transportation Systems Conference (ITSC)* (2019), pp. 3567–3572. DOI: 10.1109/ITSC.2019.8917519.
- [54] Ella-Lovise Hammervold Rørvik. ‘Automatic Docking of an Autonomous Surface Vessel’. MA thesis. Norwegian University of Science and Technology (NTNU), 2020.
- [55] Wojciech Samek, Thomas Wiegand and Klaus-Robert Müller. *Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models*. 2017. DOI: 10.48550/ARXIV.1708.08296. URL: <https://arxiv.org/abs/1708.08296>.
- [56] J. Schulman et al. ‘High-dimensional continuous control using generalized advantage estimation’. In: *International Conference on learning representation(ICLR)* (2016).
- [57] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347. URL: <https://arxiv.org/abs/1707.06347>.
-

-
- [58] John Schulman et al. ‘Trust Region Policy Optimization’. In: *Proceedings of the 32nd International Conference on Machine Learning* 37 (2015), pp. 1889–1897.
- [59] H. Shen and C. Guo. ‘Path-following control of underactuated ships using actor-critic reinforcement learning with mlp neural networks’. In: *Sixth International Conference on Information Science and Technology (ICIST), IEEE* (2016), pp. 317–321.
- [60] Andrew Silva et al. ‘Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning’. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, Aug. 2020, pp. 1855–1865. URL: <https://proceedings.mlr.press/v108/silva20a.html>.
- [61] David Silver et al. ‘A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play’. In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.
- [62] David Silver et al. ‘Deterministic Policy Gradient Algorithms’. In: *Proceedings of the 31st International Conference on Machine Learning* 32.1 (2014), pp. 387–395. URL: <https://proceedings.mlr.press/v32/silver14.html>.
- [63] David Silver et al. ‘Reward is enough’. In: *Artificial Intelligence* 299.103535 (2021). ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2021.103535>.
- [64] Karen Simonyan, Andrea Vedaldi and Andrew Zisserman. ‘Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps’. In: *Workshop at International Conference on Learning Representations*. 2014.
- [65] Mukund Sundararajan, Ankur Taly and Qiqi Yan. ‘Axiomatic Attribution for Deep Networks’. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3319–3328.
- [66] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [67] Adrian Weller. ‘Transparency: Motivations and Challenges’. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer International Publishing, 2019, pp. 23–40. ISBN: 978-3-030-28954-6. DOI: 10.1007/978-3-030-28954-6_2.
- [68] Anson Wong. *Building Model Trees*. 2020. URL: https://github.com/ankonzoid/LearningX/tree/master/advanced_ML/model_tree.
-

-
- [69] Matthew D. Zeiler and Rob Fergus. ‘Visualizing and Understanding Convolutional Networks’. In: *Proceedings of the 13th European Conference on Computer Vision*. 8689 (2014).
- [70] Ke Zhang et al. ‘Explainable AI in Deep Reinforcement Learning Models for Power System Emergency Control’. In: *IEEE Transactions on Computational Social Systems* 9.2 (2022), pp. 419–427. DOI: 10.1109/TCSS.2021.3096824.
- [71] L. Zhao and M.-I. Roh. ‘COLREGs-compliant multiship collision avoidance based on deep reinforcement learning’. In: *Ocean Eng.* 191.106436 (2019). DOI: 10.1016/j.oceaneng.2019.106436.

ISBN 978-82-326-5788-9 (printed ver.)
ISBN 978-82-326-6955-4 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)



NTNU

Norwegian University of
Science and Technology