Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 54-16 (2021) 146–152

Explainable AI methods on a deep reinforcement learning agent for automatic docking ⋆

Jakob Løver * **Vilde B. Gjærum** * **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: Artifical 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 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 meth-

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

10.1016/j.ifacol.2021.10.086

ods 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 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.

$$y = [f_1 \ f_2 \ f_3 \ a_1 \ a_2] \tag{1}$$

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.

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

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 model that approximates the local behavior of the predictor for a single prediction. By creating a perturbed dataset made up of

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).

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:

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

$\mathcal{L}(f, g, \pi_x)$ 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 Gjærum 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 Gjærum et al. (2021), the linear functions in the leaf nodes can be written on the form

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

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

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

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 Gjærum 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.

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.

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.

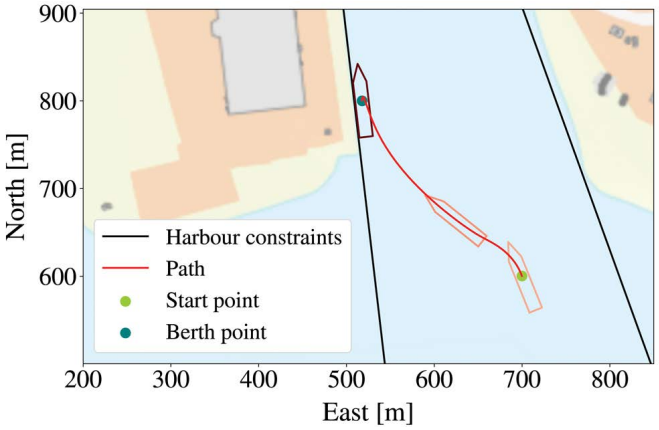

Fig. 2. Trajectory of vessel during a docking episode.

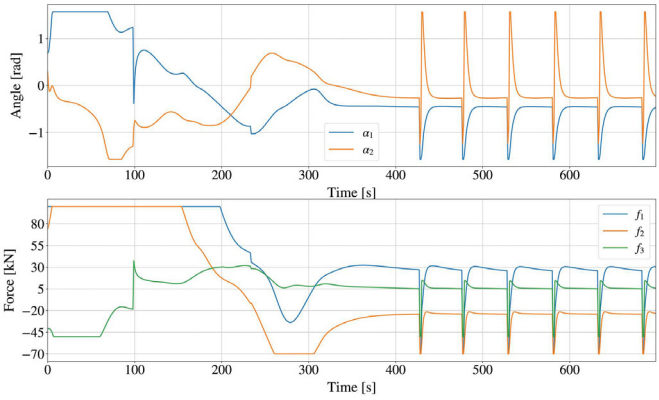

Fig. 3. Actions for a docking episode.

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

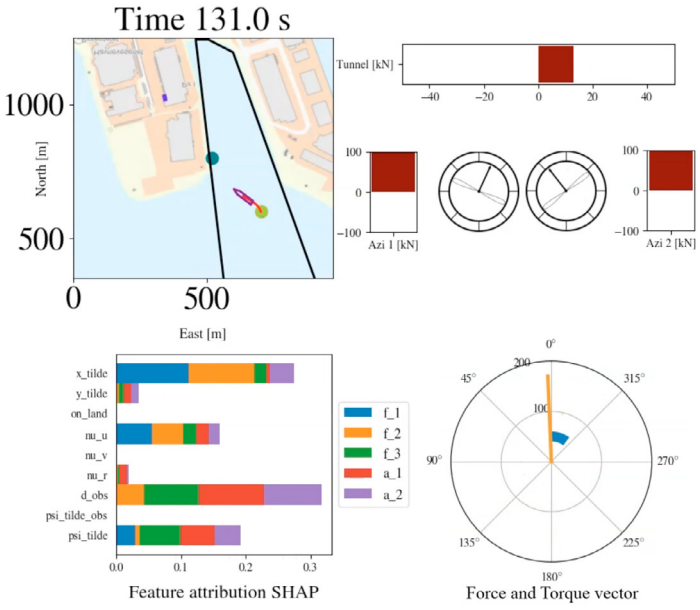

Fig. 4. Visualization of force and torque in body frame together with actions, trajectory, and feature attributions. The bottom left bar plot represents a weighted sum of the absolute value of the feature attribution per state across all actions such that the sum of all bars equal 1.

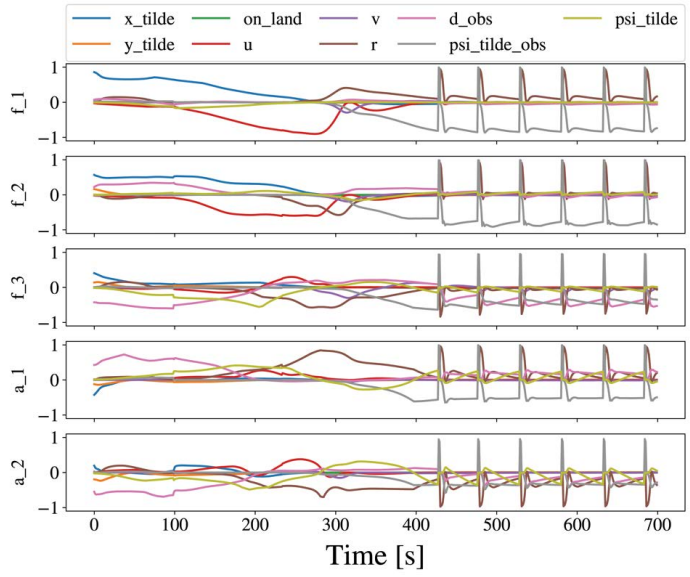

Fig. 5. Feature attributions for Kernel SHAP.

the respective action in the positive or negative direction. The sum of all 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 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 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.

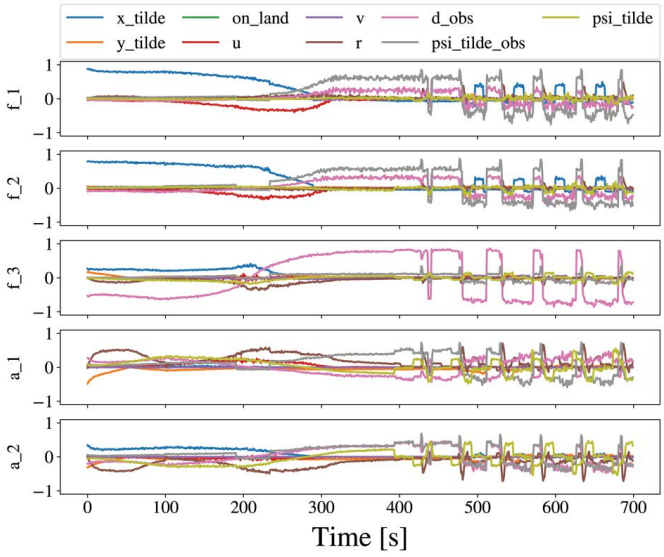

Fig. 6. Feature attributions for LIME.

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} 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

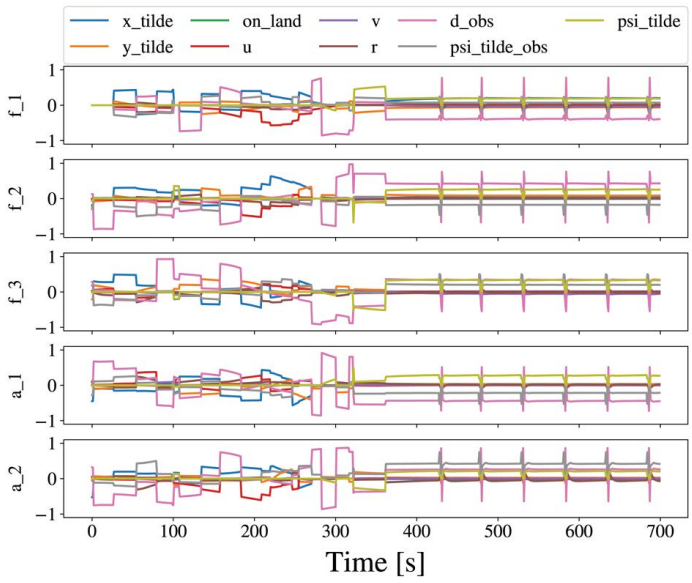

Fig. 7. Feature attributions for LMT.

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.

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	• Can easily adapt to new data	• Slow • Noisy explanations • Only local explanations	16.57 s
Kernel SHAP +K-means	• Can easily adapt to new data • Smooth explanations	• Slow • Only local explanations	3.6486 s
LMT	• Fast • Can run in real-time • Drop-in replacement for model • Can provide global explanations	• 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.