# Approximating a deep reinforcement learning docking agent using linear model trees

Vilde B. Gjærum[1] and Ella-Lovise H. Rørvik[2] and Anastasios M. Lekkas[1]

***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 be 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 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

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.

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

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

perturbating 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 pertubation 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 perturbating 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 in stead, 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:

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

where $f_1$,$f_2$ (from -70 to 100 kN) and $\alpha_1$,$\alpha_2$ (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 can be seen in Figure 1. The state vector, which is also the input feature is composed of 9 states:

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

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.

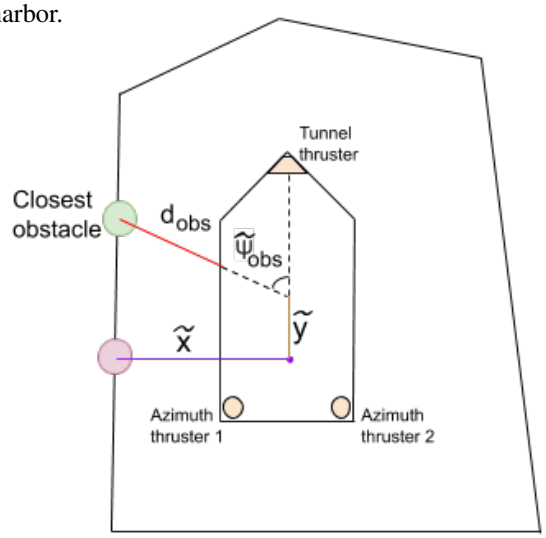



Fig. 1: The features, or states, representing the vessels position relative to the closest obstacle and the positioning of the vessels three thrusters.

## 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 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.

**Algorithm 1:** Building LMTs

```
Require:
training data 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
```

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.

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

$$\mathcal{D}_L = (x \in \mathcal{D} : x_f \leq t_n)$$
$$\mathcal{D}_R = (x \in \mathcal{D} : x_f > t_n) \quad (4)$$

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

where $f$ is the feature the split should be performed on, $t_n$ is the threshold number $n$ in the grid search, where $n \in [0, 1, 2, ..., N]$, 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:

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

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.

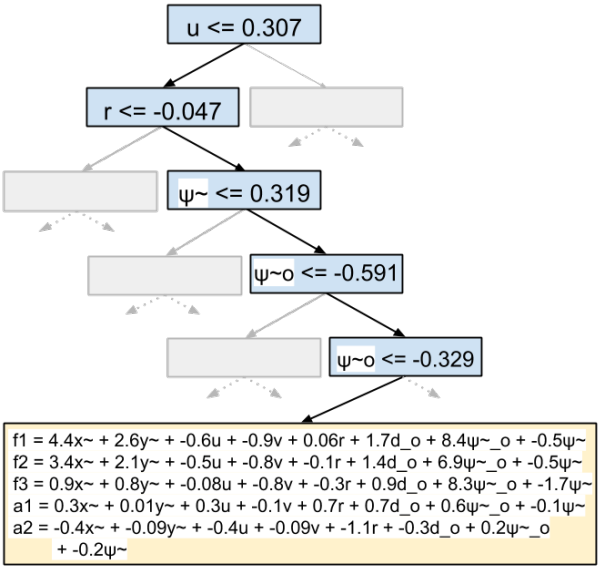
The tree has no maximum depth condition, instead it has a maximum number of leaf nodes it is allowed to have. This lets us directly state how many regions the tree is allowed to divide the input feature space into. Additionally, the tree is allowed to grow more asymmetrically, which again allows the 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.

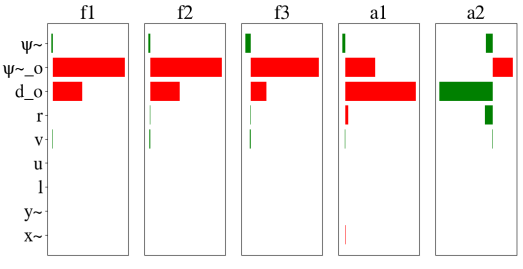
$$y = \sum_f a_f x_f + C \quad (7)$$

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

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.




(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.

## 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.

| 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%) |
| $\alpha_1$( deg ) | 16.61(9.2%) | 23.49(13.05%) |
| $\alpha_2$( 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

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 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 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 $d_o$ 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 $d_o$ 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 $\sim 250$ 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. LMTs 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 $\alpha_1$, and $f_2$ and $\alpha_2$ 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].

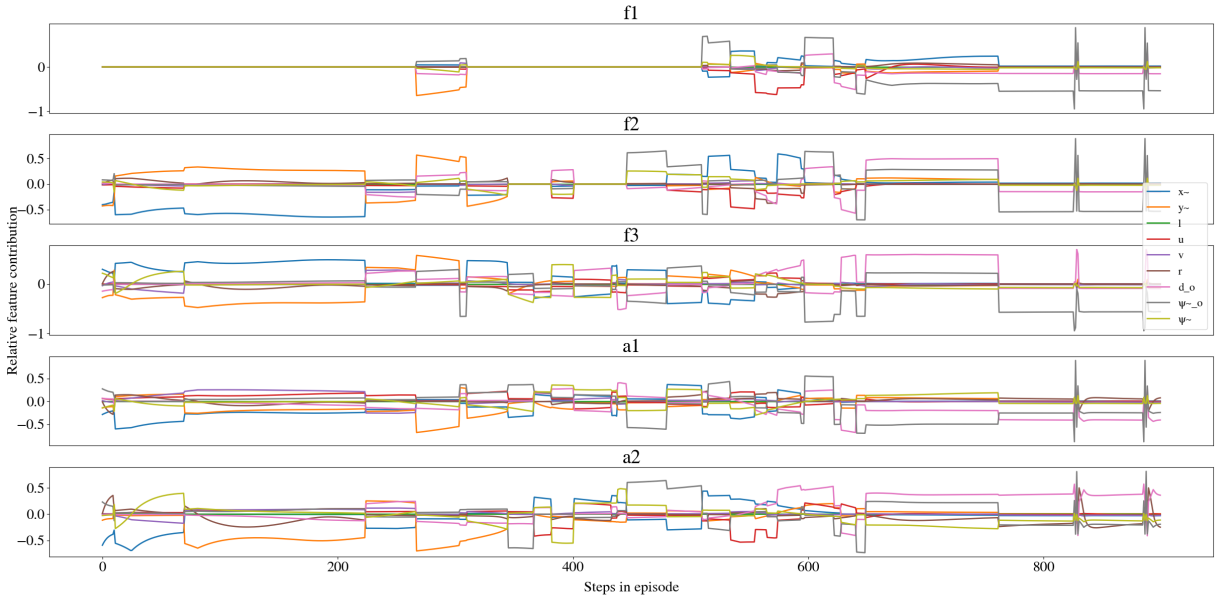
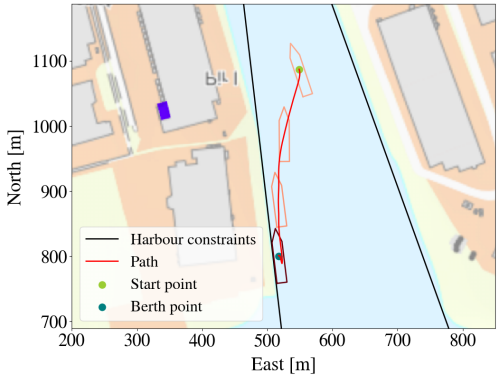
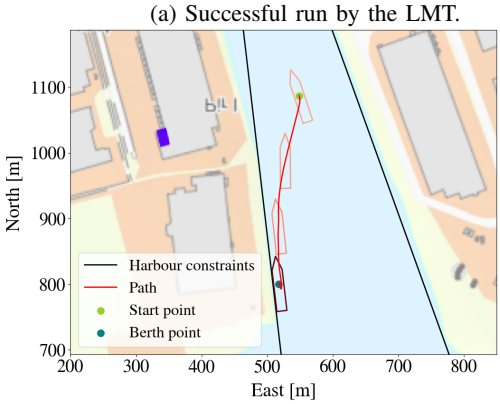



Fig. 3: Relative feature contributions given by the LMT for the episode shown in Figure 4b.

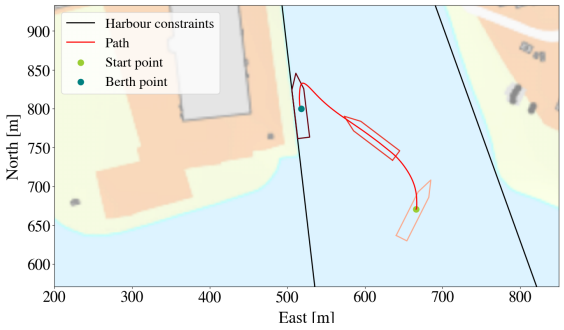



(a) Successful run by the LMT.

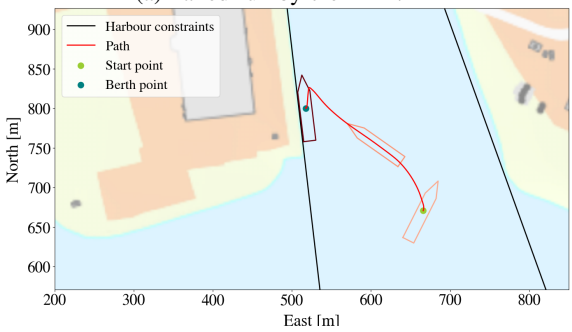



(b) Successful run by the PPO-agent.

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




(a) Failed run by the LMT.

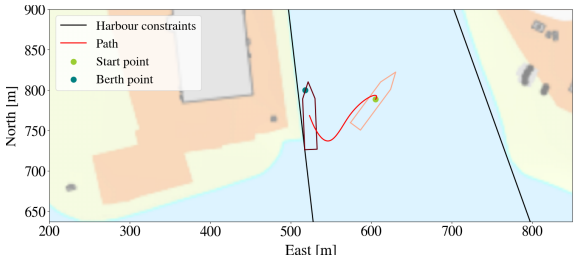



(b) Successful run by the PPO-agent.

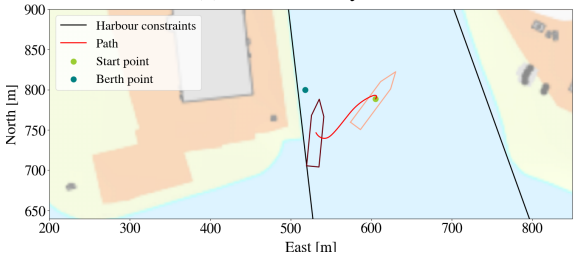
Fig. 5: Paths of PPO-agent and LMT from same starting point.

## 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




(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.

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*.

[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.