Causal versus Marginal Shapley Values for Robotic Lever Manipulation Controlled using Deep Reinforcement Learning

Sindre Benjamin Remman[1], Inga Strümke[2] and Anastasios M. Lekkas[3]

Abstract— **We investigate the effect of including application knowledge about a robotic system states' causal relations when generating explanations of deep neural network policies. To this end, we compare two methods from explainable artificial intelligence, KernelSHAP, and causal SHAP, on a deep neural network trained using deep reinforcement learning on the task of controlling a lever using a robotic manipulator. A primary disadvantage of KernelSHAP is that its explanations represent only the features' direct effects on a model's output, not considering the indirect effects a feature can have on the output by affecting other features. Causal SHAP uses a partial causal ordering to alter KernelSHAP's sampling procedure to incorporate these indirect effects. This partial causal ordering defines the causal relations between the features, and we specify this using application knowledge about the lever control task. We show that enabling an explanation method to account for indirect effects and incorporating some application knowledge can lead to explanations that better agree with human intuition. This is especially favorable for a real-world robotics task, where there is considerable causality at play, and in addition, the required application knowledge is often handily available.**

Index Terms— **Deep reinforcement learning, robotics, explainable artificial intelligence, Shapley additive explanations, causal SHAP**

I. Introduction

Data-driven control methods have become widespread over the last years due to their ability to capture unforeseen changes in the surroundings and the system dynamics and adapt accordingly. Reinforcement learning (RL) based methods show great promise in terms of adaptability in robotics applications. However, this adaptability comes at a cost, as RL methods are often paired with a function approximator such as deep neural networks (DNNs), which are in general not interpretable by humans. The combination of RL with DNNs is called deep reinforcement learning (DRL) and has been prominent in the last decade because of its high performance on several tasks that have been considered difficult for computers to achieve superhuman performance [1], [2]. DRL has also had success within robotic manipulation [3]–[5]. However, the non-interpretable nature of DNNs implies that using DRL methods to control a real-world cyber-physical system during safety-critical operation comes with considerable risks.

Non-interpretable models permeate the state-of-the-art methods in machine learning (ML). In response to this, researchers are putting effort into investigating how the decisions of ML agents can be explained. The field addressing these issues is called *explainable artificial intelligence* (XAI), from which a steadily increasing number of methods are being developed. Among the first and most widely used XAI methods is Local Interpretable Model-agnostic Explanations (LIME), presented in [6]. This method locally approximates the non-interpretable model using an interpretable model, for instance, a linear model. Linear LIME is an *additive feature attribution method*. Methods of this type create an explanation model that is a linear function of binary variables. This definition was formalized by [7], whose authors also realized that several existing explanation methods share this property, thus unifying several explanation methods and introducing SHapley Additive exPlanations (SHAP). The SHAP framework produces feature attributions satisfying the axioms of the Shapley decomposition, a solution concept from cooperative game theory [8]. Adapting linear LIME to satisfy the Shapley axioms [8] results in the feature attribution method KernelSHAP [7].

One can use SHAP to explain any ML model for which the target value in the data is known, and the explanation is given in the form of a feature importance attribution. In the SHAP value calculation, the binary variables mentioned above indicate the presence or absence of a model feature. All SHAP implementations thus rely on calculating an ML model's expected outcome in the absence of model features. In KernelSHAP [7], this calculation is done using a marginal distribution of the excluded features, which amounts to assuming independence between the model features. As argued by [9], explanations generated using the marginal distribution can only represent the direct effects of features on the model, not the indirect effects a feature can have on the output by in turn affecting other features [10]. Taking the causal structure in the data into account, [9] present a modification to the SHAP package, named *causal* SHAP.

The contributions of this paper are the following:

- Developing a `Python` version of causal SHAP [9], available in [11].
- We employ causal SHAP for explaining a deep neural network controller (policy) performing a robotic manipulation task. To achieve this, we consider the geometry of the problem and, for each state, identify whether an intervention on the state value could influence other states.

[1]Sindre Benjamin Remman is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway `sindre.b.remman@ntnu.no`

[2]Inga Strümke is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway `inga.strumke@ntnu.no`

[3]Anastasios M. Lekkas is with the Department of Engineering Cybernetics, Centre for Autonomous Marine Operations and Systems (AMOS), Norwegian University of Science and Technology (NTNU), Trondheim, Norway `anastasios.lekkas@ntnu.no`

- We compare the explanations generated by KernelSHAP and causal SHAP. In doing so, we investigate the effect of taking indirect feature effects into account, thereby obtaining feature attributions based on a more complete physical description of the system at hand.

This paper is organized as follows: in Section II, we present the necessary theory behind DRL, SHAP and causal SHAP; in Section III, we describe the task to be solved using DRL, the experimental design, and how we use SHAP and causal SHAP to explain the decision-making agent; in Section IV, we present and discuss our results; and finally, in Section V, we draw our conclusions.

II. Preliminaries

This section gives an overview of the theory and terminology necessary to understand the remainder of this paper. Firstly, we provide an overview of the fundamentals of DRL. Secondly, the theory behind SHAP is explained. Lastly, we look at how SHAP is modified to create causal SHAP values.

A. Deep Reinforcement Learning

Reinforcement learning considers two parts: the *agent*, which learns and makes decisions, and the *environment*, which consists of everything in the problem other than the agent [12]. The interactions between these two parts are illustrated in Figure 1. The agent receives a state from the environment, performs an action based on this state, and receives a new state together with a reward from the environment. This cycle then repeats for the whole operation. The goal of RL is to find a *policy* that maps states to actions. The so-called *optimal policy* does the mapping in a way such that a long-term expected reward is maximized, here defined by the discounted infinite horizon model:

$$E[\sum_{t=0}^{\infty} \gamma^t r_t] \,, \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor, and r_t is the reward received at time t [13, pp.13-15].

As previously stated, DRL refers to RL where the function approximator is a DNN. Here, we train a DRL agent using the Deep Deterministic Policy Gradient (DDPG) algorithm [14]. This is an actor-critic algorithm, which means that it trains two neural networks, the *actor-network* and the *critic-network*. The actor-network functions as the policy, which means that it maps states to actions, and the critic-network is used to guide the training of the actor-network. From a control engineering point of view, the policy is then akin to a controller. DDPG trains a deterministic policy, which means that a specific policy will always give the same output for the same input.

B. Shapley Additive Explanations

The Shapley decomposition, introduced by Lloyd Shapley in 1953 [8], has in recent years been applied extensively in the XAI literature. Originating in cooperative game theory, the Shapley decomposition decomposes the outcome of a cooperative multiplayer game and attributes the outcome

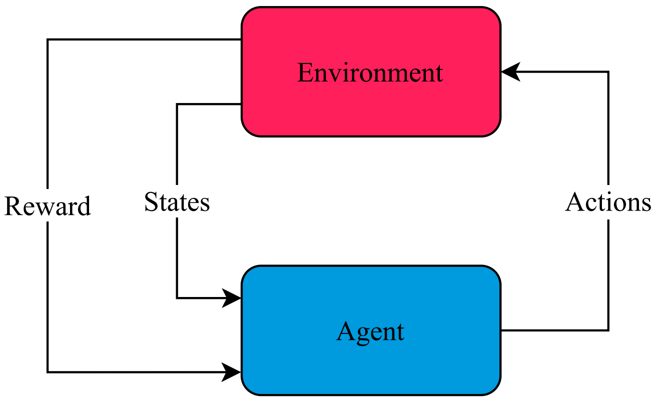

Fig. 1: The reinforcement learning loop.

to the game's players. This attribution is arguably fair to all the players since it is the only solution that satisfies specific properties; see, for instance, Theorem 2 in [15]. These properties are:

- Efficiency (called *local accuracy* in [7])
  - The contribution attributed to all players should add up to the difference between the outcome and the outcome with no players in the game.
- Monotonicity (called *consistency* in [7])
  - If the game changes such that a player's contribution increases or stays the same regardless of the other players, that player's attribution should not decrease.
- Equal treatment (also called *symmetry*)
  - Two players that contribute the same in all coalitions should be given the same attribution.
- Dummy (also called *missingness*)
  - If a player does not change the outcome whether or not they are added to any coalition, they should receive an attribution of zero.

The Shapley value of participant i is calculated as a weighted mean over all subsets $\mathcal{S} \subseteq \mathcal{N}$ of the game's N participants, not containing participant i:

$$\phi_i = \sum_{\mathcal{S} \subseteq \mathcal{N}} \frac{|\mathcal{S}|!\,(N - |\mathcal{S}| - 1)!}{N!} \left(v(\mathcal{S} \cup \{j\}) - v(\mathcal{S})\right) \,. \tag{2}$$

Here, $v(\mathcal{S})$ is the *characteristic function*, which fully characterizes the game by mapping any set of participants in the game to a single real number $2^{\mathcal{N}} \rightarrow \mathbb{R}$.

Table I shows how the game-theoretic concepts can be viewed in the context of XAI and this paper's use-case. The analogy shows that the Shapley decomposition can be used to obtain a feature attribution for an ML model. The feature attribution distributes the model's output among the model's input features, quantifying how each of these affect the model output. The output of an ML model f trained on a set of data with features $\mathbf{x}$, for a feature vector with specific values $\mathbf{x} = \mathbf{x}^*$, can be decomposed as

$$f(\mathbf{x}^*) = \phi_0 + \sum_{i=1}^{N} \phi_i^* \,, \tag{3}$$

TABLE I:
GAME THEORETIC CONCEPTS IN THE CONTEXT OF XAI AND THIS PAPER'S USE-CASE

Game theory interpretation	XAI interpretation	This paper's use case
The game	A model performing a prediction task	A DRL model controlling a manipulator
The game's outcome	The model's prediction/output	The actions chosen by the DRL model
The game's participants	The model's input features	The system's states

with N the total number of features in the model, ϕ_0 the expected value of the model output across the data set, $E[f(\mathbf{x})]$, and ϕ_i^* the Shapley value for the specific output on $\mathbf{x}^*$.

Two main challenges are associated with calculating Shapley values for feature attribution: First, the calculation is very computationally expensive. A model using N features would need to be evaluated 2^N times, once for each feature's inclusion or exclusion, as is readily seen from (2). Second, it is, in general, not possible to evaluate a fitted ML model with sets of features missing. For example, for a neural network that is trained using N-dimensional input features, the architecture of the network does not permit the usage of $(N-1)$-dimensional input features. To circumvent these challenges, implementations such as the widely used SHAP, introduced by Lundberg and Lee [7], rely on approximations. SHAP's characteristic function, in the notation of [16], is

$$v(\mathcal{S}) = E[f(\mathbf{x})|\mathbf{x}_{\mathcal{S}} = \mathbf{x}^*_{\mathcal{S}}]\,, \tag{4}$$

which is an estimate of the expected model output, conditional upon the included features with values $\mathbf{x}_{\mathcal{S}} = \mathbf{x}^*_{\mathcal{S}}$. As such, SHAP values attribute the change in the expected model output to each model feature. In other words, SHAP values are the Shapley values of the original model's conditional expectation function [7]. The SHAP values are calculated using an estimation of how much each feature contributes to driving the model output away from its mean output across a data set. KernelSHAP, introduced in [7], estimates (4) with absent features by sampling from the marginal distribution, which amounts to the assumption of independence between the included and excluded features.

Various changes to the sampling procedure used for estimating the expected model output have since been suggested, and particularly relevant in this context are [9], [16]–[18].

C. Causal SHAP

To account for mutual dependence among the model features, the SHAP calculation can use the conditional distribution of the excluded features, instead of the marginal, as first suggested by Aas et al. [16]. Furthermore, *causal structure* of the features can be taken into account by conditioning the absent features upon the values of the included features by *intervention*. To achieve this, the sampling procedure for calculating the expected model output in the SHAP calculation must use the interventional distribution. Interventions on random variables are commonly represented mathematically using the so-called do-operator. Judea Pearl famously invented the do-calculus [19], consisting of rules for mapping probability distributions. With this, one can infer interventional probabilities from conditional probabilities. As shown by Heskes et al. [9], modifying conditional SHAP by applying the do-calculus rules thus allows us to use the interventional distribution in the SHAP calculation. Then, (4) becomes

$$v(\mathcal{S}) = E[f(\mathbf{x})|do(\mathbf{x}_{\mathcal{S}} = \mathbf{x}^*_{\mathcal{S}})]\,, \tag{5}$$

which we calculate by integrating over the absent features $\overline{\mathcal{S}}$, as detailed in [9]. The main advantage of the resulting so-called *causal* SHAP values is that both direct as well as *indirect* effects of the model features are taken into account. The direct effects represent the change in the model's output due to a change in a feature without changing the absent features. The indirect effects, on the other hand, represent the change caused in the missing features by the intervention upon a feature, see [9, eq (5)]. The inclusion of these indirect effects constitutes the main difference between causal SHAP values and the marginal SHAP values introduced earlier, as the latter by construction only represent direct effects.

The causal SHAP implementation used in this paper [11] was created by the first author, by adapting the `R` implementation by [9] to `Python`. In the implementation, we specify the causal structure of the data via a causal ordering in a nested list. Each list is defined as causally dependent on the elements in the preceding list(s). In addition, we use a second, separate list to specify whether the dependencies within each nested list result from a confounding factor or mutual interactions between the components in the nested list.

III. METHODOLOGY

In this section, we describe the objective and training process of the DRL agent, the data set creation, and finally, how we analyze the data using the two different SHAP implementations.

A. Lever manipulation task

The robotic manipulator that is used in this paper is the OpenMANIPULATOR-X[1] by Robotis, which can be seen in Figure 2. This manipulator has five degrees of freedom, four for the joints and one for the gripper. We do not use the first joint during this lever manipulation task, which corresponds to a rotation about the manipulator's base. This is both because this makes the training more efficient, but also because the angle of this joint is trivial to solve for using

$$\theta_1 = \text{arctan2}(y_{lever}, x_{lever}),$$

where y_{lever} and x_{lever} is the y- and x- coordinates of the lever expressed in the inertial frame of the manipulator.

[1]`https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/`

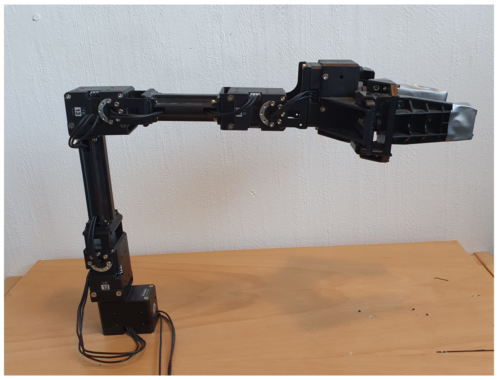

Fig. 2: The OpenMANIPULATOR-X.

The task involves moving the lever from a random start angle to a random target angle. These target and start angles are selected uniformly according to

$$\theta_{start}, \theta_{target} \in \mathbb{R} : \theta_{start}, \theta_{target} \in [-1.0 \text{ rad}, 1.0 \text{ rad}],$$

and $|\theta_{start} - \theta_{target}| > 0.4$ rad, to ensure that the target and start angles have viable minimum and maximum values, while also not initializing the lever too close to the goal

B. States and Actions

In this task, the dimension of the state vector is eight and consists of: the joint angles of the manipulator, three in total, since we do not use the first joint, as explained in section III-A, denoted by q_1, q_2, q_3; the distance between the two fingers of the gripper, q_4; the horizontal and vertical distance from the end-effector to the lever, d_x, d_z; and the current and desired angles of the lever, $\theta_{lever}, \theta_{target}$. See Figure 3 for a visualization of the system's states.

The action vector is of dimension four, where the first three entries correspond to the desired change in angle of the shoulder, elbow, and wrist joints, respectively. The fourth entry in the action vector indicates whether the gripper should open or close:

$$a_4 \geq 0, \rightarrow \text{Gripper should open}$$
$$a_4 < 0, \rightarrow \text{Gripper should close.}$$

Then the state vector $\mathbf{s}$ and the action vector $\mathbf{a}$ consist of:

$$\mathbf{s} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 & d_x & d_z & \theta_{lever} & \theta_{target} \end{bmatrix}^T$$
$$\mathbf{a} = \begin{bmatrix} \Delta q_1 & \Delta q_2 & \Delta q_3 & a_4 \end{bmatrix}^T$$

C. Training procedure

The agent was trained using the DDPG algorithm together with the technique Hindsight Experience Replay (HER) [20]. HER enables the usage of sparse rewards by treating each configuration of the lever angle as a separate goal. In other words, even if the agent did not reach the original goal, it treats the lever angle that was reached as an alternative goal, and it receives rewards accordingly. Sparse rewards avoid overcomplicating the reward engineering, and we, therefore, give sparse rewards according to

$$r = \begin{cases} -1, & \text{if } |\theta_{lever} - \theta_{target}| \geq 0.025 \text{ rad} \\ 0, & \text{if } |\theta_{lever} - \theta_{target}| < 0.025 \text{ rad} \end{cases},$$

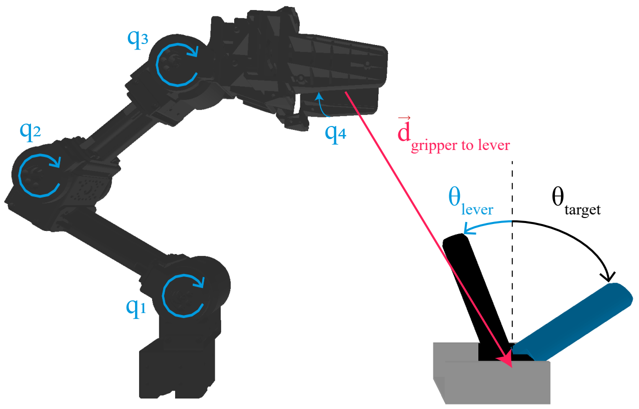

Fig. 3: Visualization of the system's states.

where 0.025 rad is the chosen precision for the lever manipulation task. The agent is trained using simulations, and we use two different simulators for this. The first simulator is PyBullet, a fast simulator but not close enough to the real-world environment for our purposes. The agent trained in PyBullet is then transfer learned in Gazebo, which is slower but more like the real-world environment. After transfer learning in Gazebo, we can deploy the agent in the real-world environment. This training procedure is described in more detail in [21], where the main difference here is that we have reduced the number of states, which we did primarily to make the feature attributions simpler to interpret.

D. Data set and explanations

To sample the excluded features, implementations of SHAP rely on background data sets. Furthermore, we wish to choose interesting decisions by our agent to explain. To this end, we collect a data set by letting the fully trained DRL agent operate in the real-world environment by running 15 *test episodes* with randomly selected target and start lever angles. From these test episodes, we identify interesting events and compare the explanations generated by the two different SHAP implementations on these. We remove the episodes where these events occur and use the resulting data set as our background data set. In this data set, the input features x are the environment's states, while the targets y are the actions chosen by the DRL agent. We chose one event from Episode 1 and two events from Episode 3, meaning that our background data set consists of episodes 2 and $4-15$.

To display the results, we create a plot similar to the force plot available in the SHAP package[2]. The force plot illustrates the "force" of each feature on the prediction, showing how the features force the prediction away from the mean prediction and towards the model's prediction. In our adaptation, we show one force plot for each of the agent's actions on the same figure.

As stated in Section II-C, the causal SHAP implementation requires the causal structure of the data to be specified by a causal ordering. The causal ordering we use is

$$[[\theta_{target}], [q_1, q_2, q_3], [q_4, d_x, d_z], [\theta_{lever}]].$$

[2]https://github.com/slundberg/shap

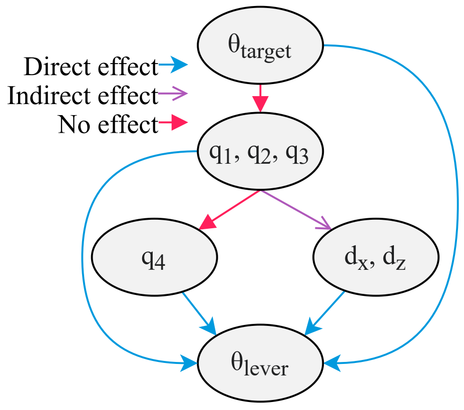

Fig. 4: Visualization of the chosen causal ordering.

In addition, we assume that none of the features are influenced by a confounding factor but instead have only mutual interactions. We found the causal ordering by considering which of the other features would change if we performed an intervention on one feature. For instance, the lever position θ_{lever} would change if we changed one of the joint features such that the manipulator pushes the lever. A visualization of our causal ordering is shown in Figure 4, where blue arrows indicate each feature's direct effect on the target, purple arrows indirect effects via other features, and the red arrows indirect effects that we know are not present. More details on this are given below.

Our setup reveals a weakness in the causal SHAP implementation: The feature θ_{target} does not have any causal connection with any of the other features, but because of the way causal SHAP is implemented, it still has to be defined in the causal ordering. We list this feature first in the causal ordering to avoid the indirect effects of all the other features flowing through this independent feature (which would happen if we placed it after other features). We assume now and show in the results sections that although this feature is put first in the causal ordering, the causal SHAP algorithm will uncover that this feature only has a direct effect on the prediction and therefore assign it an indirect causal connection strength close to 0 to the following features in the causal ordering.

Another weakness is that, according to our causal ordering, q_1, q_2, q_3 (the joint features), have a causal effect on q_4 (the gripper feature). This is not necessarily true, but the current implementation does not allow two features to affect a third feature without affecting each other in the causal ordering. Figure 4 highlight these two issues, indicating that θ_{target} does not influence features succeeding it in the causal ordering and that other joint variables not influencing q_4.

IV. Results and Discussion

In this section, we discuss and compare the results from using causal SHAP and KernelSHAP to explain the actions of the agent performing the robotic lever manipulation task. We select three events from the data set described above and examine interesting aspects of the events' SHAP values. We group the discussion into where the methods agree and

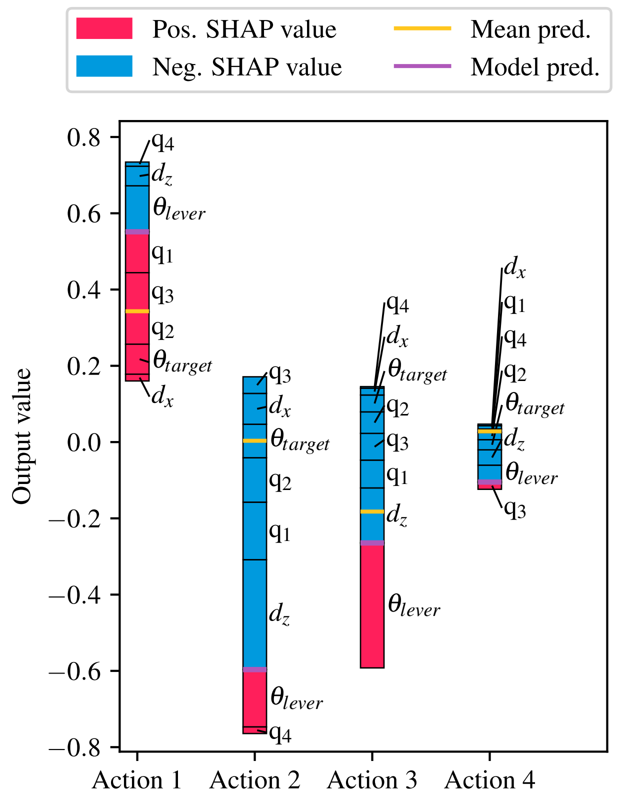

Fig. 5: Episode 1: pushing event, causal SHAP

where the methods disagree with each other for each event. The selected events are defined as follows:

Pushing event: The first of the events is from episode 1, where the manipulator pushes the lever from the start angle to the target angle with the gripper closed. We select a time-step in which the manipulator is actively pushing the manipulator and analyze it.

Grasping event: The second event selected is from episode 3, in which the manipulator is grasping the lever before pulling it. We here analyze the exact time-step in which the manipulator grasps the lever.

Pulling event: The third and final event we analyze also belongs to episode 3 and takes place just after the grasping event when the manipulator is being used to pull the lever from the start angle to the target angle.

A. *Pushing event*

Where the methods agree: For the pushing event, force plots for causal SHAP and KernelSHAP are shown in Figures 5 and 6, respectively. In both plots, for action 4, all but one feature have negative SHAP values, implying that most aspects of this situation inform the agent to keep the gripper closed. For both methods, for the actions corresponding to moving the joints (actions 1 to 3), the feature q_4 has the lowest SHAP value. This means that both methods agree that knowing whether the gripper is closed is not important for the movement of the joints. Both methods attribute similar importance to θ_{lever} and θ_{target}. However, causal SHAP assigns slightly lower values to θ_{lever}

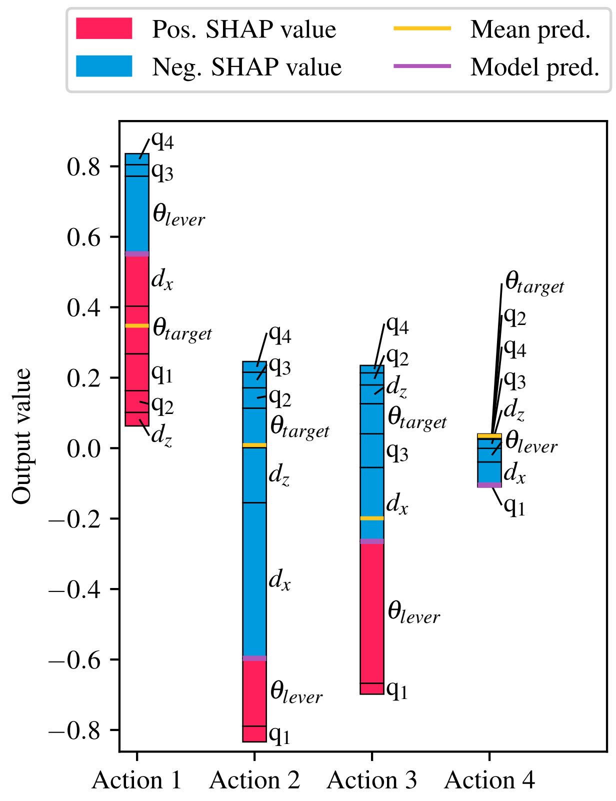

Fig. 6: Episode 1: pushing event, KernelSHAP

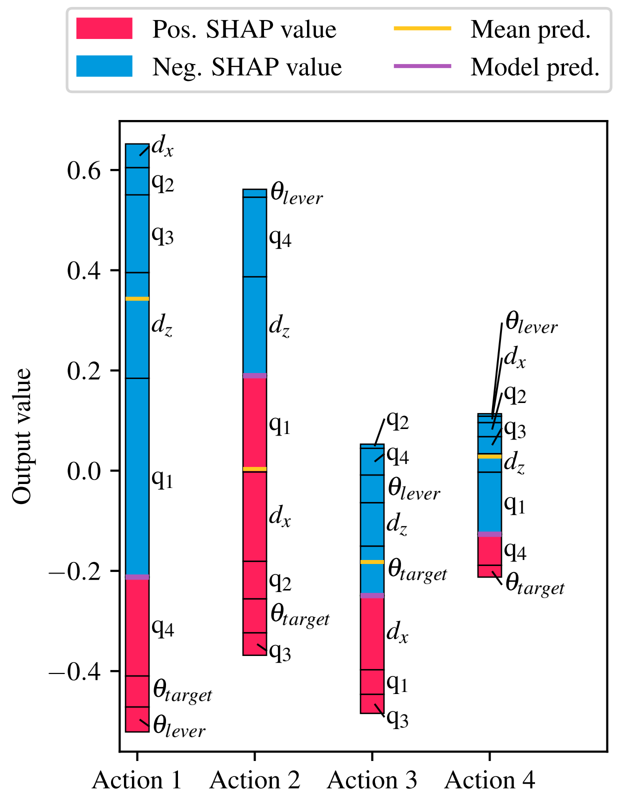

Fig. 7: Episode 3: grasping event, causal SHAP

compared to KernelSHAP, likely because this feature is at the bottom of the causal ordering. Nevertheless, θ_{lever} is still among the features with the highest SHAP values. This implies that θ_{lever} is an important predictor variable for how the manipulator should be used to move the lever. Simply put, it is important to know where the lever is to move it.

Where the methods disagree: Overall, causal SHAP attributes higher importance to the joint variables, q_1, q_2, q_3. In contrast, KernelSHAP attributes higher importance to d_x and d_z, the features that together form a Cartesian vector from the end-effector to the lever's base. The joint variables are higher in the causal ordering, and they cause changes in d_x and d_z, and thus causal SHAP assigns more importance to them. This indicates that some of the effects attributed to d_x and d_z in KernelSHAP are captured as indirect effects of the joint variables in causal SHAP. The joint variables, d_x and d_z, contain much of the same information, that is, information about the manipulator's position. However, in addition to this information about the manipulator's position, the joint variables contain information about the manipulator's orientation. However, d_x and d_z contain information about where the lever is situated in relation to the manipulator. The exclusive information that these two sets of features contain makes each of them valuable for performing the lever manipulation task. However, because the joint variables are what is being controlled by actions $1-3$, and it is difficult to manipulate something one does not know where is, it makes sense these features should have among the highest SHAP values.

B. *Grasping event*

Where the methods agree: For the grasping event, the plots for causal SHAP and KernelSHAP are shown in Figure 7 and Figure 8, respectively. In contrast to the plots for the pushing event, q_4 has a much higher SHAP value for both methods. This is presumably because the agent is in the process of pulling the lever, and q_4 is important for knowing that the lever still needs to be grasped by the manipulator.

Where the methods disagree: The difference between the two methods is much more significant for θ_{lever} than before. KernelSHAP still attributes the most importance to this feature, while causal SHAP attributes the lowest overall importance to θ_{lever} in the grasping event. This is curious since the position of the lever should be necessary for deciding how to grasp it. The reason is that causal SHAP assigns part of the contribution from θ_{lever} as indirect effects to the features above it in the causal ordering. Similar to the pushing event, the joint variables are generally more critical according to causal SHAP than they are according to KernelSHAP. Conversely, d_x and d_z are generally more critical according to KernelSHAP than what they are to causal SHAP. However, there are some exceptions to this: Consider q_2, which has approximately the same SHAP values for both plots' corresponding actions, and action 3, for which the SHAP value of q_3 has a larger magnitude for KernelSHAP than for causal SHAP.

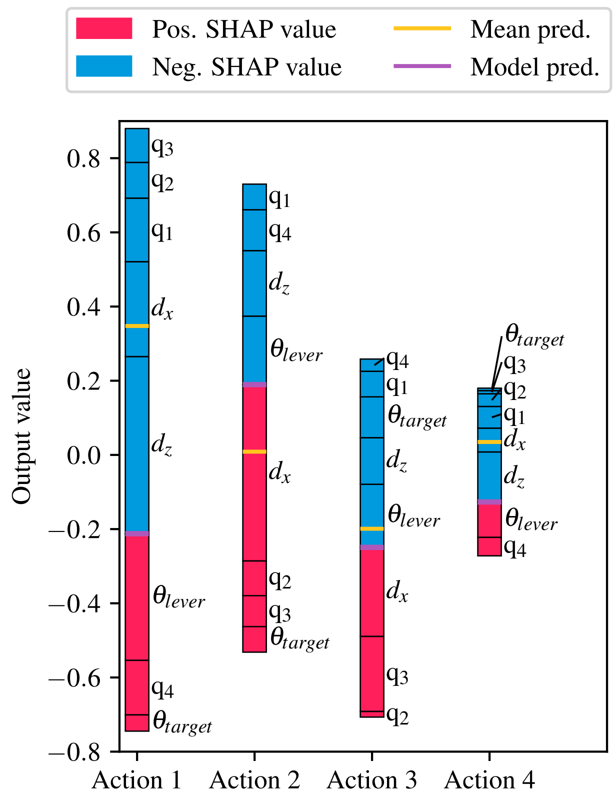

Fig. 8: Episode 3: grasping event, KernelSHAP

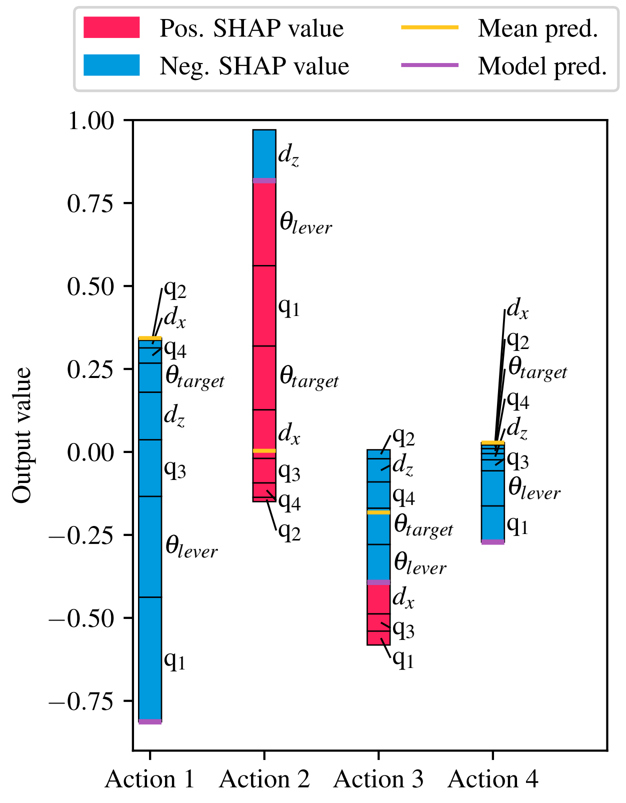

Fig. 9: Episode 3: pulling event, causal SHAP

C. Pulling event

Where the methods agree: Figure 9 and Figure 10 show the results from causal SHAP and KernelSHAP, respectively. As discussed in Section III-D, θ_{target} was put on the top of the causal ordering by necessity. However, we can see that this feature has approximately the same SHAP value for all actions in the pulling event for both methods. This was also the case for the pushing and grasping events described above. This suggests that causal SHAP has discovered that this feature has only a direct effect, and therefore given an indirect causal connection strength of approximately 0 to the features succeeding it in the causal graph, in agreement with our expectation.

Both methods assign only negative SHAP values to actions 1 and 4. This means both methods agree that the gripper should be closed, which corresponds to having a negative action 4. If action 4 had a positive value, the manipulator would lose its grip on the lever, which is undesirable in this situation. In addition, since action 1 controls the joint closest to the manipulator's base, this action is most effective at moving the end-effector. For example, moving q_1 a specific angular distance would result in a much more significant difference in the end-effector position than moving q_3 the same angular distance. Therefore, it seems reasonable that both methods give negative SHAP values to all features for action 1. This is because this is the most effective way to move the end-effector backward, which corresponds to pulling the lever in this case.

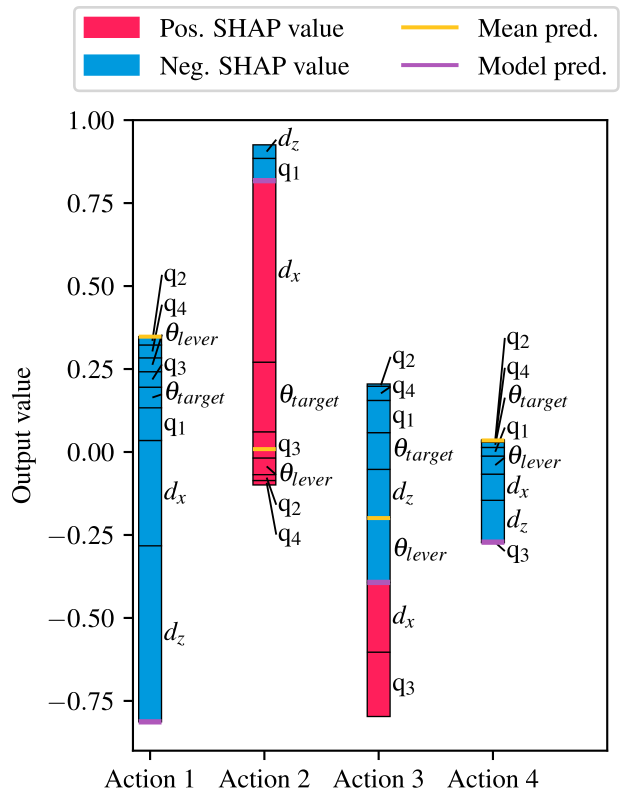

Fig. 10: Episode 3: pulling event, KernelSHAP

Where the methods disagree: As for the two previous events, we observe again that KernelSHAP prioritizes d_x and d_z over the joint variables, and vice versa for causal SHAP. In fact, d_x and d_z account for over half of the total magnitude of all the SHAP values for actions 1, 2 and 4 in Figure 10.

V. Conclusion

We have shown how causal SHAP can be used to explain not only the direct effects but also the indirect effects features can have on the decisions of a deep reinforcement learning agent controlling a real-world robotics system. In addition, we have shown how causal SHAP can incorporate application knowledge in the explanation generation process. We expect causal relations to be an essential part of explainable artificial intelligence (XAI) in the future, especially for explaining physical systems.

Due to the complex nature of the presented explanations, we regard these as most useful for data scientists, model developers, and others with experience in data analysis and XAI. The explanations would require processing to produce explanations suitable for non-technical end-users. According to [22], explanations should be contrastive, and explanations based on *counterfactuals* [23] are more intuitive to humans. Work has been done towards unifying feature attribution and counterfactuals [24], and towards generating counterfactuals from SHAP [25]. Further work could thus investigate whether transforming feature attributions to counterfactual explanations could make them more accessible to non-experts. Taking a leaf from [26], further work can also include adjusting explanations based on the characteristics of a specific end-user audience, which is particularly applicable to a real-world robotics application.

More specific to causal SHAP, further work can i.a. consist in modifying causal SHAP to account for fully independent features (such as θ_{target} in this paper). Another valuable extension would be the possibility to specify which indirect causal connections have strength 0 (for causally independent subsystems).

Acknowledgment

The Research Council of Norway supported this work through the EXAIGON project, project number 304843.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, and D. Hassabis, *AlphaZero Shedding new light on chess, shogi, and Go*, 2018 (accessed 15-Oct- 2020). [Online]. Available: https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

[2] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari Human Benchmark," in *International Conference on Machine Learning*. PMLR, 2020, pp. 507–517.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end Training of Deep Visuomotor Policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," in *Conference on Robot Learning*. PMLR, 2018, pp. 651–673.

[5] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[6] 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*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144.

[7] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 4765–4774.

[8] L. S. Shapley, *A VALUE FOR N-PERSON GAMES.* Defense Technical Information Center, 1952.

[9] T. Heskes, E. Sijben, I. G. Bucur, and T. Claassen, "Causal Shapley Values: Exploiting Causal Knowledge to Explain Individual Predictions of Complex Models," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 4778–4789.

[10] J. Pearl, "Direct and indirect effects," in *Probabilistic and Causal Inference: The Works of Judea Pearl*, 2022, pp. 373–392.

[11] S. B. Remman, "Causal shap python," https://github.com/sbremman/causal_shap_python, 2021.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2017.

[13] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Springer Publishing Company, Incorporated, 2014.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning." in *ICLR*, 2016.

[15] H. P. Young, "Monotonic solutions of cooperative games," *International Journal of Game Theory*, vol. 14, pp. 65–72, 1985.

[16] K. Aas, M. Jullum, and A. Løland, "Explaining individual predictions when features are dependent: More accurate approximations to Shapley values," *Artificial Intelligence*, vol. 298, p. 103502, 2021.

[17] C. Frye, C. Rowat, and I. Feige, "Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1229–1239.

[18] D. Janzing, L. Minorics, and P. Bloebaum, "Feature relevance quantification in explainable ai: A causal problem," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 2907–2916.

[19] J. Pearl, "Causal diagrams for empirical research," *Biometrika*, vol. 82, no. 4, pp. 669–688, 1995.

[20] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight Experience Replay," *Advances in neural information processing systems*, vol. 30, 2017.

[21] S. B. Remman and A. M. Lekkas, "Robotic Lever Manipulation using Hindsight Experience Replay and Shapley Additive Explanations," in *2021 European Control Conference (ECC)*. IEEE, 2021, pp. 586–593.

[22] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[23] D. Hume *et al.*, *An enquiry concerning human understanding: A critical edition*. Oxford University Press, 2000, vol. 3.

[24] R. Kommiya Mothilal, D. Mahajan, C. Tan, and A. Sharma, "Towards unifying feature attribution and counterfactual explanations: Different means to the same end," *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, May 2021.

[25] S. Rathi, "Generating counterfactual and contrastive explanations using shap," *arXiv preprint arXiv:1906.09293*, 2019.

[26] V. B. Gjærum, I. Strümke, O. A. Alsos, and A. M. Lekkas, "Explaining a deep reinforcement learning docking agent using linear model trees with user adapted visualization," *Journal of Marine Science and Engineering*, vol. 9, no. 11, p. 1178, 2021.