Robotic Lever Manipulation using Hindsight Experience Replay and Shapley Additive Explanations

Sindre Benjamin Remman[1] and Anastasios M. Lekkas[2]

***Abstract*— This paper deals with robotic lever control using Explainable Deep Reinforcement Learning. First, we train a policy by using the Deep Deterministic Policy Gradient algorithm and the Hindsight Experience Replay technique, where the goal is to control a robotic manipulator to manipulate a lever. This enables us both to use continuous states and actions and to learn with sparse rewards. Being able to learn from sparse rewards is especially desirable for Deep Reinforcement Learning because designing a reward function for complex tasks such as this is challenging. We first train in the PyBullet simulator, which accelerates the training procedure, but is not accurate on this task compared to the real-world environment. After completing the training in PyBullet, we further train in the Gazebo simulator, which runs more slowly than PyBullet, but is more accurate on this task. We then transfer the policy to the real-world environment, where it achieves comparable performance to the simulated environments for most episodes. To explain the decisions of the policy we use the SHAP method to create an explanation model based on the episodes done in the real-world environment. This gives us some results that agree with intuition, and some that do not. We also question whether the independence assumption made when approximating the SHAP values influences the accuracy of these values for a system such as this, where there are some correlations between the states.**

***Index Terms*— Deep Reinforcement Learning, Hindsight Experience Replay, Robotics, Explainable Artificial Intelligence, SHapley Additive Explanations**

I. Introduction

Deep Reinforcement Learning (DRL), which is the fusion of traditional Reinforcement Learning (RL) and Artificial Neural Networks (ANNs), has since the early 2010s been used to solve a variety of difficult problems. The first successful application of DRL was done in [1], where the authors made a DRL algorithm, called Deep Q-network (DQN), that could learn how to play Atari 2600 video games directly from high-dimensional screen pixel input. After that, DRL has shown promise in various fields: DRL was used to create AlphaZero, which beat world-champion computer programs at Chess, Shogi, and Go [2]; Agent57 was created in 2020, the first DRL agent that learned to play all 57 Atari 2600 games in the OpenAI gym with super-human performance [3]; DRL has also made great strides in the field of robotics [4]–[8].

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

[2]Anastasios M. Lekkas is with 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

One commonly used DRL algorithm is Proximal Policy Optimization (PPO), which was presented in [9]. PPO has been used in various tasks such as: quadrotor control [10], electrical vehicle charging with uncertain wind power [11], and for developing a step climbing method for a crawler type rescue robot [12]. A problem with PPO is that it has a poor sample efficiency. This is because it is an *on-policy* algorithm, which means that it cannot reuse past experiences, and needs to collect new samples every time it optimizes its parameters [5]. The algorithm used in this paper is the Deep Deterministic Policy Gradient (DDPG) algorithm, presented in [4], which unlike PPO, is an *off-policy* algorithm. This means that it can reuse past experiences. DDPG has been used in tasks such as: traffic signal control [13], control of unmanned surface vehicles [14], and flight control [15]. A disadvantage with DDPG is that it is sensitive with regards to hyperparameters, which can require significant effort when tuning the algorithm [5]. PPO and DDPG use a stochastic and deterministic policy respectively. Stochastic policies are appropriate for an agent that needs to adapt to stochasticity in the environment, but for this paper, deterministic policies are more suitable. This is because a deterministic policy's decisions arguably will be easier to explain.

Even though DRL shows great promise, and has been used to solve remarkably different problems, there are still many drawbacks with DRL: it is hard to do intelligent exploration, instead of just random exploration; training a policy can take a significant amount of time; safety guarantees during safety-critical operations are still lacking; it can be difficult and time-consuming to design a good reward function that does not have any unwanted side-effects; and it can be difficult to understand how a DRL policy makes its decision, which ties into whether or not the policy can be trusted. The last drawback mentioned here is mainly because of the black-box nature of ANNs.

Based on the recent advances in Machine Learning (ML), researchers are now looking at how to explain the decisions made by ML agents. Many of the most prominent applications of ML are now done using models that have a black-box nature. This makes it especially demanding to understand how the agents make their decisions. The field that handles these types of explanation problems is collectively called eXplainable Artificial Intelligence (XAI) and has garnered increasing interest these last years [16], [17]. Local Interpretable Model-agnostic Explanations (LIME) is one of the more popular XAI methods, which learns an interpretable model locally around the prediction of the ML model [18]. Another XAI method is the *Integrated Gradients* method,

which integrates the gradients of an ANN on a straight-line path from a baseline x' to an input x to explain how the inputs to an ANN affect its prediction [19]. The XAI method used in this paper is called SHapley Additive exPlanations (SHAP) [20]. Similar to LIME, SHAP is an additive feature attribution method. SHAP has some desirable properties that LIME does not have, but LIME can generally generate explanations faster. SHAP has also been shown empirically to give explanations that better agree with human intuition [20].

In this paper, we examine how to alleviate the last two of the drawbacks mentioned above by using the RL technique Hindsight Experience Replay (HER) [6] and SHAP, where the objective is to manipulate a lever using a robotic manipulator. HER enables the usage of a simple reward function for a complex system such as a robotic manipulator, and SHAP provides an approximation of the contribution of each state to each action. The results of this paper are an extension of the first author's Master's thesis [21]. We do these experiments in a platform that involves lever control with a robotic manipulator, more specifically the contributions of this paper are:

- The design of an experimental setup including the OpenMANIPULATOR-X by ROBOTIS, which involves lever angle control and can correspond to many problems in the physical world. In addition to the real-world experimental setup, corresponding simulated environments including a model of the lever was created. We trained the policy in two simulated environments. After training, the policy was implemented on the physical manipulator.
- We customize the sparse reward function to the lever manipulation problem and train the simulated system using DDPG and HER.
- We implement SHAP to understand how the actions are selected by the policy. In this way, we provide insight and can figure out if the system has learned behavior that conforms to human intuition.

A related approach can be seen in [22], where the authors also apply SHAP to a DRL problem and then investigate how different background data influence the explanations. However, this is done on a much less complicated system than our system, which only has four states and one action, where we have 20 states and four actions. Another related approach is shown in [23], where they use ML to predict failure modes in robot grasping, and compare how these failures can be explained using both inherently interpretable ML models and black-box ML models. Unlike our work, they do this solely in a simulated environment, where we use a real-world environment in addition to simulated environments.

The remaining sections in this paper are organized as follows: the Preliminaries, where we discuss some of the necessary background theory; the Methodology, where we discuss the task and experimental setup; the Results and Discussion, where we show and discuss the results from the lever-manipulation task performed in this paper; and finally the Conclusion, where we give a brief overview of the main points from this paper.

II. Preliminaries

This section briefly examines and explains the necessary theory for the rest of the paper. Firstly, we give the fundamental concepts in RL and DRL. Secondly, the HER technique is motivated and explained. Lastly, the theory and properties of SHAP are examined.

A. Reinforcement learning

In RL, we assume that we can model the environment as a Markov Decision Process (MDP). An MDP is defined as a tuple $< \mathcal{S}, \mathcal{A}, T, R >$, where $\mathcal{S}$ is a state-space, $\mathcal{A}$ is an action-space, T is a Markovian transition model, and R is a reward function. A Markovian transition model is a proper probability distribution over the next possible states given the current state and the action taken in the current state. A Markovian transition model satisfies the Markov property, which means that only the current state and action are relevant for the distribution of the next state [24, p.11]. The reward function returns a scalar number, the reward, based on the transition $R(s_t, a_t, s_{t+1}) = r_t$.

The goal for an RL agent is to find the optimal policy π^*, which is the policy that maximizes the long-term expected reward, 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 a hyperparameter called the discount factor [24, pp.13-15].

The DDPG algorithm used in this paper is an off-policy actor-critic algorithm that can tackle problems with continuous state- and action-spaces. As briefly mentioned in the introduction, off-policy method can use samples generated any time during training for optimization [25]. Being actor-critic, DDPG trains two ANNs. The first is the *actor-network*, which serves the role of the policy, that is, it takes in the state of the environment and outputs what the action should be (equivalent to a controller). The second is the *critic-network*, which is used to approximate the value of taking a certain action in a certain state (called a Q-value). The critic's role is to evaluate the performance of the actor. This evaluation is subsequently used for optimizing the actor's parameters. There were two crucial components for DRL that were introduced in [1] and [26] in the context of DQN. Both of these components are also used in DDPG. The first component is the *experience replay*, which ensures that training samples are independent, a requirement when training ANNs. This requirement is crucial to avoid overfitting because of the strong correlation between sequential samples [1]. Since most optimization algorithms assume that samples are independent of each other, the training might be unsuccessful if samples are used for optimization in just the order they appear. Experience replay is used as a buffer that all transitions $\{s_t, a_t, r_t, s_{t+1}\}$ are stored in and recalled

randomly from during ANN training. This means that the transitions are independent of each other, and that previous transitions are not forgotten and can be used multiple times. Experience replay buffers can only be used by off-policy algorithms [1]. The second component introduced in [1] and [26] is the *target network*, which are copies of the original networks that are, for instance, gradually changed towards the parameters of the original networks (Polyak averaging), or copied from the original networks at a certain interval. Having a target that is changing fast becomes similar to trying to catch up to a moving target, so by using target networks the correlation with the target is decreased, which improves the training stability [4], [26].

We would like to note that there exist improvements over the DDPG algorithm such as the state-of-the-art Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm presented in [27]. However, in RL, there are no guarantees that any algorithm will work, and DDPG has worked well for this system from the start. This choice should also not affect the later focus on explainability.

We could also have included comparisons with other learning-based control methods for robotic manipulators. However, in the context of this paper, it is not in the scope to do this. This paper aims to study the sequence from the simulator to the real-world to explainability.

B. Hindsight experience replay

Arguably, one of the hardest parts of implementing RL and DRL concerns how to engineer a good reward function. Give the reward too sparsely, and the agent will learn slowly, or not at all; give the reward too frequently, and the behavior of the agent may already be specified by the reward function.

A novel technique that can enable RL agents to learn from sparse rewards was presented in [6]. The approach is called Hindsight Experience Replay (HER), where the idea is to substitute the actual goals with virtual goals, which represent the goal-state that was actually achieved by the agent [28]. Consider an agent that has achieved a trajectory of $s_1, \ldots, s_T$, but the goal state is not in this trajectory. In this case, with sparse rewards and ordinary DRL, the agent would receive the same reward for each step of the trajectory, and would not have any useful feedback for optimization. Even though the agent has not discovered how to reach the goal state, from another point of view, it has discovered how it can reach every other state in the trajectory. When using HER, any of the states within the trajectory can then be used as substituted goals, where the agent gets the same reward as it would if it achieved the real goal. In this way, the agent will get feedback which it can then use to optimize its parameters.

In this paper, the strategy that is used for selecting which state that should eligible for being substituted goals is called "future". For every transition stored in the experience replay, k new versions of the transition are also stored, where the goal states are substituted with randomly selected achieved goal states that came from the same episode, but were observed after the transition.

When using HER, it is useful to consider the state-space as consisting of two parts. The first part is the observation of the environment, the second part is the goal states. This way, when substituting goals, only the part of the state that contains the goal states needs to be substituted with the actual achieved goals.

C. Shapley Additive Explanations

SHAP is a method developed in [20], followed by a library in python[1] that can help to identify which inputs are most important for a function's output. SHAP is a model-agnostic and post-hoc XAI method, which means that it can work with any type of model or function, and it can explain decisions based on already generated data. It does this by approximating the Shapley values [29] for the input compared to the output. The Shapley values explain how each input contributes to the magnitude of the output. SHAP is an additive feature attribution method, which means that it is a linear function of binary variables, of the form:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i, \tag{2}$$

"where $z' \in \{0, 1\}$, M is the number of simplified input features, and $\phi_i \in \mathbb{R}$" [20]. It can be shown that additive feature attribution methods have a single unique solution which has three desired properties [20]:

- Local accuracy
  - When using an explanation model to approximate another model for a specific input x, the explanation model's output will match the other model's output for the simplified input x' which corresponds to the input x.
- Missingness
  - Requires features missing from the original input x to have no impact:

$$x'_i = 0 \rightarrow \phi_i = 0.$$

- Consistency
  - If a model changes so that a simplified input's contribution increases or stays the same, then that input's impact should not decrease.

As stated above, there is only one solution to the explanation model (2) that satisfies the three properties just stated. This solution corresponds to the Shapley values [20]:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \backslash i)], \tag{3}$$

where $|z'|$ is the number of non-zero entries in z', $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries of the simplified inputs x', $f_x(z') = f(h_x(z'))$, f is the original prediction model, and h_x is a mapping function that maps simplified inputs to the

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

original inputs [20]. Equation (3) is a difficult equation to solve, as it relies on $f_x(z' \backslash i)$, which is the model that is going to be explained, without feature i present. This is not straight-forward to achieve, for instance, for an ANN, because we would have to train several new models without the feature(s) present. The SHAP library provides several methods to address these challenges and approximate these values. Some of these methods take advantage of a model's structure to more efficiently approximate the Shapley values. The SHAP method that is used in this paper is called *Deep SHAP* and is specific for ANNs.

III. Methodology

For the experiments done in this paper, we used both a physical and two simulated versions of the OpenMANIPULATOR-X robotic manipulator by ROBOTIS[2]. Two simulators were used, Gazebo and PyBullet. The manipulator has 5 degrees-of-freedom (DOFs), four for the joints of the manipulator, and one for the gripper. The physical robot and its Gazebo counterpart were controlled using the Robot Operating System (ROS), with packages developed by ROBOTIS[3]. Also, we developed an environment in the PyBullet simulator to accelerate training, as will be explained in Section III-D. The OpenAI Gym framework was used to create the RL environments. The DDPG+HER implementation used the PyTorch deep learning framework and was adapted from a GitHub repository created by A. Imran[4].

A. Lever manipulation task

The goal for each episode in this task is to move the lever to a randomly selected goal angle. The starting position and goal position of the lever are randomly selected according to

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

with the additional constraint that $|\theta_{start} - \theta_{goal}| > 0.4$ rad.

All policies in this paper are trained using HER and because of this, the reward function can be designed to only give sparse rewards on this task. The reward is given according to

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

where 0.025 rad is approximately $1.43°$, which we deemed to be a sufficient accuracy for a task such as this.

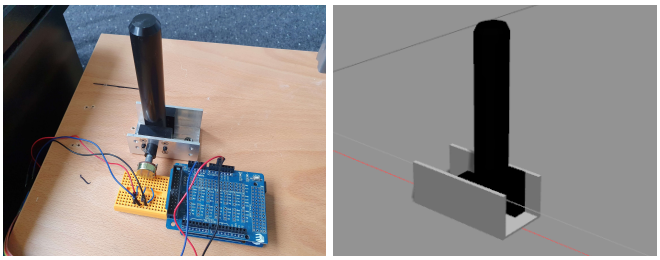
B. Experimental design

The lever that is shown in Figure 1a was used for the experiments. The angle of the lever was measured using a potentiometer and an Arduino Uno. Using the open-source 3D graphics software Blender, a mesh for this lever was also created and is shown in Gazebo in Figure 1b, and

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

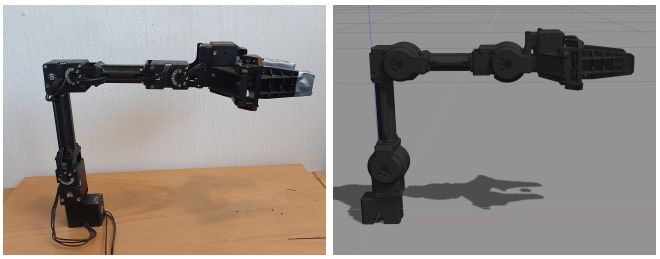
[3]https://github.com/ROBOTIS-GIT/open_manipulator

[4]https://github.com/alishbaimran/Robotics-DDPG-HER



(a) (b)

Fig. 1: **(a)**: Real lever with potentiometer and Arduino. **(b)**: Lever model in Gazebo.



(a) (b)

Fig. 2: **(a)**: Real-world manipulator. **(b)**: Manipulator in Gazebo.

a Unified Robot Description Format (URDF) model was created with the mesh as a base. The URDF-model requires various properties, such as the specification of the links and joints of a model, visual and collision meshes, dynamics, and friction. These properties were approximated by trial-and-error using the real-world environment and the simulated environment. Both PyBullet and Gazebo can use models of the URDF format.

The OpenMANIPULATOR-X can be seen in the real-world environment in Figure 2a, and the simulated manipulator can be seen in Gazebo in Figure 2b (the PyBullet version looks similar, and has therefore not been included).

C. State and Action

The state-space of this task is of dimension 20. The first 19 entries in the state-space are the observation of the environment. This observation consists of the angles and velocities of the manipulator's joints, the Cartesian position of the lever's base relative to the manipulator's base, the relative distance between the end-effector and the lever's base, and the current angle of the lever. The remaining entry in the state-space is the goal state, and in this task, the goal state is the desired lever angle. When substituting goals with HER, the desired lever angle is then substituted with the achieved lever angle.

The agent is given a restriction in that it cannot move the manipulator's first joint. This is the joint that rotates around the base of the manipulator. This was done to make training faster, and also because it is trivial to solve for the angle of this joint given the Cartesian x- and y-coordinates of the lever's base:

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

Hyperparameter	Value
Learning rate actor (PyBullet), α_a	0.001
Learning rate critic (PyBullet), α_c	0.001
Learning rate actor (Gazebo), α_a	0.0008
Learning rate critic (Gazebo), α_c	0.0008
Discount factor, γ	0.98
l2 regularization, λ	1
noise_eps, ϵ_n	0.2
random_eps, ϵ_r	0.18
HER ratio to be replaced, k	4
HER replay strategy	future
Mini-batch size	256
Neurons in input layer, actor	20
Neurons in input layer, critic	24
Hidden layers, actor and critic	3
Neurons in hidden layers, actor and critic	256
Neurons in output layer, actor	4
Neurons in output layer, critic	1
Activation functions hidden layers	ReLU
Activation functions input layer, actor	Tanh
Activation functions input layer, critic	Linear
a_s	0.1

TABLE I: DDPG hyperparameters

Since the agent is not able to move the first joint, the action-space is of dimension four. The first three entries correspond to the desired relative angles of joints 2-4, and the fourth entry corresponds to whether the gripper should open or close:

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

D. Training Procedure

The training procedure for this task involves to first train the policy in a simulated environment, then transfer the policy to the real environment. The first training takes place in PyBullet, which runs faster on this task compared to Gazebo. PyBullet also has more suitable functionality for DRL, for instance, a dedicated step function that steps the simulator one time-step forward. The policy was trained in PyBullet for 50 epochs with 30 *training episodes* in each epoch. The policy was then more finely tuned by transfer learning in Gazebo for additional 300 episodes. Gazebo is more accurate concerning the real-world environment on this task. Except for the learning rate, the remaining hyperparameters were the same for Gazebo and PyBullet. The hyperparameters can be seen in Table I. The action selected by the actor is scaled by a factor of a_s before the action is applied to the manipulator. After transfer learning in Gazebo, experiments using the physical manipulator were conducted.

For this paper, exploration is done using both ϵ-greedy exploration and by adding exploration noise to the action. By ϵ-greedy exploration, with probability $(1 - \epsilon_r)$, the agent performs the greedy action, i.e. the best available action according to the actor's knowledge at that time. With probability ϵ_r, the agent performs a random action, hence allowing the agent to explore, with $\epsilon_r \in [0, 1]$. For the exploration noise, when the greedy action is selected by the ϵ-greedy exploration, Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = \epsilon_n$ is added to the greedy action.

During training, the manipulator is randomly selected to start in a grasping position on the lever for half of the episodes. This is done to make the training faster and was inspired by how the authors in [6] did the pick-and-place task.

As previously mentioned, the goal and start angle of the lever was randomized between episodes. In addition to this, the position of the lever's base relative to the manipulator's base was also randomized between episodes when training in the simulators. In the real-world environment, the lever was attached to a wooden plate together with the manipulator, and could not be moved between episodes. To give the agent the information it needs to calculate the position of the lever when doing the real-world experiments, the distance between the manipulator's base and the lever's base was measured by hand and given as a constant in the state returned from the environment.

After the training is completed, we run five *test episodes* in both the simulated environments and the physical environment, the results of this will be shown in Section IV. In these plots, the error measured by the difference in the achieved and the goal lever angle is plotted on the y-axis over the episodes. A video of the real-world experiments is also available[5].

E. Explaining the agent's decisions with SHAP

The SHAP method was used in an attempt to get a better understanding of the agent's decisions. From the five test episodes, four episodes were used to generate the explanation model, and the remaining episode was explained. The data for the explanations were generated using the real-world manipulator. The SHAP values over the entire first episode are plotted using a force plot for each of the four actions. Features that push the prediction higher or lower are respectively shown in red or blue. The height of each feature contribution shows the magnitude of each feature's contribution.

IV. Results and Discussion

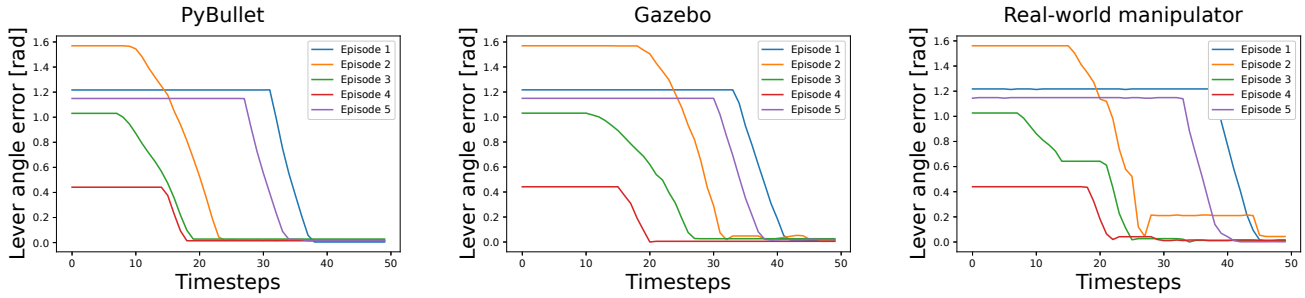
In this section, we first discuss and compare the performance of the agent on the three different platforms. We discuss reasons why the agent performed more poorly when transferred to the real manipulator. Lastly, we try to get an insight into how the agent makes its decisions by looking at how SHAP assigns importance to the input features for the actions selected.

A. Lever manipulation

The results from the lever manipulation experiments can be seen in figures 3a to 3c, where the results were achieved by randomly selecting five different start and goal conditions, and running these over five episodes for both the simulated and physical platforms. First we will discuss the results from the simulated environments in Figure 3a and Figure 3b. These results are quite good and show smooth trajectories

[5] https://youtu.be/eyPmYoAZqNA

Fig. 3: Lever angle errors for all episodes and platforms



(a) Error, PyBullet (b) Error, Gazebo after transfer learning (c) Error, real world manipulator

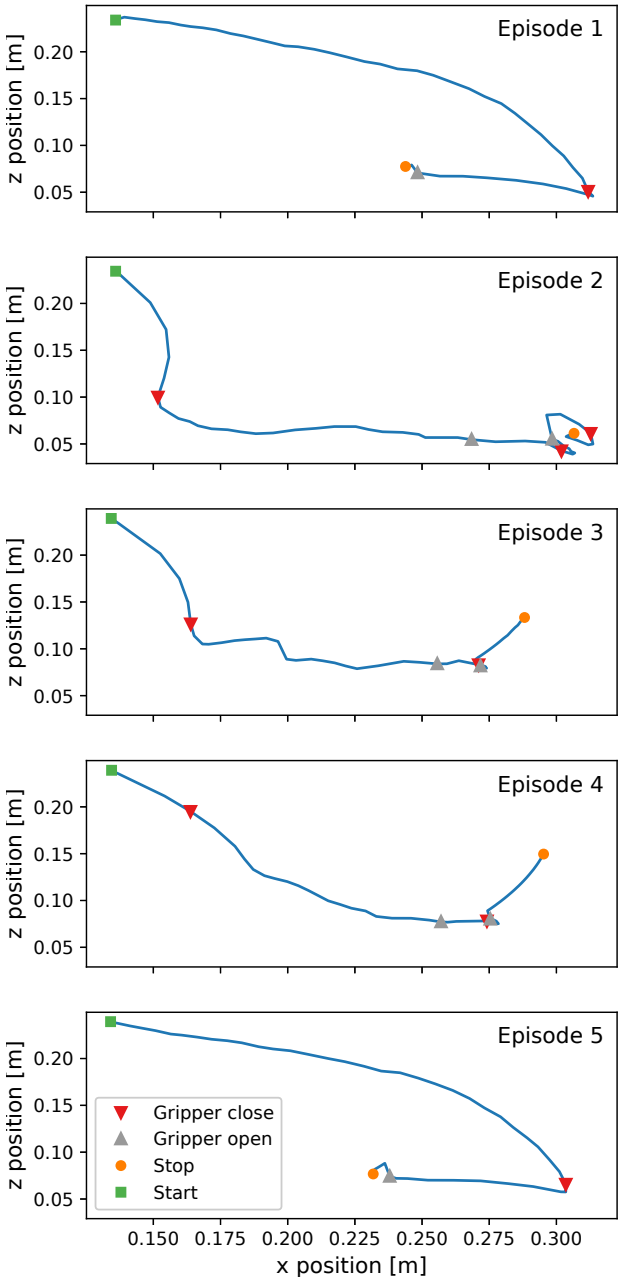


Fig. 4: Task space trajectories for the real-world manipulator, with indications for changes in gripper state. The y-coordinate is constant since the first joint is not moved.

Episode	$\lvert\theta_{start} - \theta_{goal}\rvert$	Initial distance from end-effector to lever
1	1.21719255 rad	0.2674m
2	1.56957698 rad	0.1763m
3	1.03069065 rad	0.1742m
4	0.44181838 rad	0.1833m
5	1.14932491 rad	0.2508m

TABLE II: Initial situations for the experiments

for all episodes. Some episodes take more time to complete than others, for instance, Episode 1 and Episode 5. From Table II we can see that these two episodes are when the lever initiates furthest away from the manipulator, and the initial lever angle error is also among the largest, so naturally, these episodes are the most time-consuming. The performance in both simulated environments is also quite similar, with Gazebo being somewhat slower for all episodes. The operation in Gazebo could be faster if more episodes of transfer learning were done.

We will now discuss the differences between the physical environment and simulated environments. From the plots of Episode 2 and Episode 3 in Figure 3c, which show the performance in the physical environment, it is noticeable that the performance is not as good as in the simulated environments. This can also be seen in Figure 4, especially in Episode 2, where the trajectory in the bottom right corner seems chaotic. This discrepancy in the real-world results likely stems from modeling errors in the simulated environments compared to the real-world environment. This phenomenon is described in [30] as under-modeling. This under-modeling is especially noticeable in Episode 2, where the manipulator first moves the lever too far, and then when it tries to correct this, it first tries to grasp too low on the lever and instead grasps the base of the lever. In Episode 3 and Episode 4, the manipulator also first moves the lever too far. This could indicate that both the dimensions of the lever and the friction of the lever could be under-modeled in the simulators. If more effort was made to more accurately model the friction and dimensions of the lever, these problems could have been avoided. However, friction is notoriously hard to model, and it is also possible that the effort to make the simulated lever more similar to the real lever, would fail. It is important to note that the agent was not trained at all on the real manipulator. If safe training on the physical manipulator could be implemented, this would likely increase the performance by enabling the agent to learn how to act

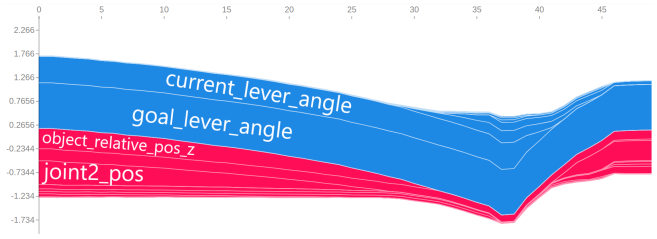


Fig. 5: SHAP values for action a_1 in Episode 1

Fig. 6: SHAP values for action a_2 in Episode 1

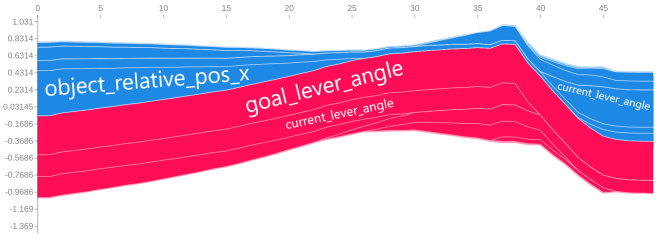


Fig. 7: SHAP values for action a_3 in Episode 1

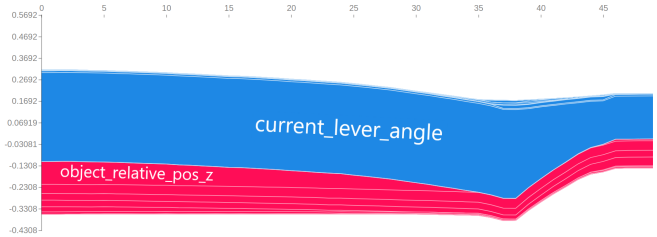


Fig. 8: SHAP values for action a_4 in Episode 1

according to the real environment, and not according to the simulation.

A possible way to improve the results from the real-world experiments could be to use *Dynamics randomization*. This technique was proposed in [8] and involves using a Recurrent Neural Network (RNN) to enable the agent to approximate the dynamics of the system while randomizing the dynamics of the simulated system between episodes. This technique could, for instance, be applied so that the agent can approximate the dynamics of the real-world lever.

As can be seen in the literature, the advantages and disadvantages of DRL compared to more traditional robotic control methods are many. However, the main disadvantage of robotic DRL is arguably the lack of explanation in the agent's decisions and the general trustworthiness of the agent. This disadvantage was attempted to be remedied by using SHAP in this paper.

B. SHAP explanations

The SHAP values plotted over the operation in Episode 1 can be seen in figures 5 to 8. In the plots, the x-axis corresponds to the step numbers on the x-axis of the plot in Figure 3c. According to these SHAP values, the most important states for the agent to make its decisions are the current lever angle and the desired lever angle. This conforms with intuition since, without these two states, it would be virtually impossible for the agent to move the lever to the desired lever angle. Other important states according to SHAP are the relative distance on the x- and z-axis between the end-effector and the lever's base. This also makes sense, since these two states, combined with the current lever angle, can be used to calculate where the end-effector is relative to the lever. However, it is surprising that the manipulator's joint angles contribute so little to the actions. Intuitively, the angles of the manipulator's joints should be important for knowing which action to execute, considering that actions 1-3 correspond to how these joints should be moved. It is also possible that the validity of these SHAP values might be put into question. SHAP assumes that all states are independent, which they might not be in most robotics applications [31].

Other than which states are important for the decisions of the agent, it is difficult to interpret these plots in a way that either increases or decreases the trust in the agent's decisions. An interesting point was made by [31] regarding whether SHAP contributes to understanding whether or not a decision is correct. They state that humans often explain by using contrastive statements (e.g. why A rather than B). It might be difficult to use a feature attribution method for this since they only describe why and not why not. It might be that for such complex problems as robotic manipulation, it is not that helpful to use a single tool to explain all problems. Instead, it might be beneficial to attempt to tailor the explanation mechanism to the use-case [31]. The creators of LIME say in [18], that for an explanation to be easy to understand, the features themselves should be easy to understand. They also say that the features used by the ML model and inputs used by the explanation model need not necessarily be the same if this can make the explanations more understandable. It is possible that the quality of the explanations could be increased if the features were transformed into an input that is more easily understood (e.g. by going from joint space to task space).

DRL research often seems to have the goal to either increase the performance of the agents or to make algorithms that can solve even more complex environments. Better performance is arguably important, but as long as the decisions of the agents cannot be explained, DRL will likely remain a tool for solving toy problems, and will not be used in safety-critical applications. This is why increasing the understanding of the agents, for instance, through XAI methods might be necessary. The implementations in this paper aim at demonstrating how well a state-of-the-art XAI method works on a robotic manipulator, and for identifying the right questions to come up with improved methods that

will help make DRL safer in such applications.

V. Conclusion

In this paper, we have shown how a Deep Reinforcement Learning policy can be implemented on a real-world robotic manipulator using ROS and used to manipulate objects, in this case, a lever. Hindsight Experience Replay simplifies the process of designing a reward function and is useful for a complex system such as a robotic manipulator. The agent performs very well when controlling the simulated manipulator, and performs comparably for most episodes when controlling the real-world manipulator. As stated in the discussion, the differences between the performance in the simulated and real environments likely stems from under-modeling of the lever in the simulations.

To explain the agent's decisions, we implemented the Explainable Artificial Intelligence method SHAP, which gives results that mostly agree with human intuition. The two most important states, according to SHAP, are the current lever angle and the goal lever angle, which is reasonable. However, it is unexpected that some states that seem like should be important, such as the joint variables, are not very important according to SHAP. This divergence from human intuition may stem from correlations between the states in the system, which may make the Shapley value approximation less accurate.

Acknowledgment

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

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013.

[2] 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

[3] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the Atari Human Benchmark," *arXiv:2003.13350 [cs, stat]*, Mar. 2020.

[4] 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," *arXiv:1509.02971 [cs, stat]*, Jul. 2019.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *arXiv:1801.01290 [cs, stat]*, Aug. 2018.

[6] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," *arXiv:1707.01495 [cs]*, Feb. 2018.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *arXiv:1504.00702 [cs]*, Apr. 2016.

[8] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3803–3810, May 2018.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Aug. 2017.

[10] G. C. Lopes, M. Ferreira, A. d. S. Simões, and E. L. Colombini, "Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, Nov. 2018, pp. 503–508.

[11] T. Long, X. Ma, and Q. Jia, "Bi-level proximal policy optimization for stochastic coordination of ev charging load with uncertain wind power," in *2019 IEEE Conference on Control Technology and Applications (CCTA)*, 2019, pp. 302–307.

[12] M. Totani, N. Sato, and Y. Morita, "Step climbing method for crawler type rescue robot using reinforcement learning with proximal policy optimization," in *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, 2019, pp. 154–159.

[13] H. Pang and W. Gao, "Deep deterministic policy gradient for traffic signal control of single intersection," in *2019 Chinese Control And Decision Conference (CCDC)*, 2019, pp. 5861–5866.

[14] Y. Wang, J. Tong, T. Song, and Z. Wan, "Unmanned surface vehicle course tracking control based on neural network and deep deterministic policy gradient algorithm," in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, 2018, pp. 1–5.

[15] A. Tsourdos, I. A. Dharma Permana, D. H. Budiarti, H. Shin, and C. Lee, "Developing flight control policy using deep deterministic policy gradient," in *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, 2019, pp. 1–7.

[16] A. Barredo Arrieta *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, p. 82–115, Jun 2020. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2019.12.012

[17] V. Belle and I. Papantonis, "Principles and Practice of Explainable Machine Learning," *arXiv:2009.11698 [cs, stat]*, Sep. 2020.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," *arXiv:1602.04938 [cs, stat]*, Aug. 2016.

[19] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks," *arXiv:1703.01365 [cs]*, Jun. 2017.

[20] 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.

[21] S. B. Remman, "Robotic manipulation using Deep Reinforcement Learning," Master's thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2020.

[22] Y. Wang, M. Mase, and M. Egi, "Attribution-based salience method towards interpretable reinforcement learning," *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*, 2020.

[23] A. Alvanpour, S. K. Das, C. K. Robinson, O. Nasraoui, and D. Popa, "Robot failure mode prediction with explainable machine learning," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 61–66.

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

[25] OpenAI. Part 2: Kinds of RL Algorithms — Spinning Up documentation. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[26] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[27] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *arXiv:1802.09477 [cs, stat]*, Oct. 2018.

[28] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, *Ingredients for Robotics Research*, 2018 (accessed 15- Oct- 2020). [Online]. Available: https://openai.com/blog/ingredients-for-robotics-research/

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

[30] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 2013.

[31] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger, and S. Friedler, "Problems with Shapley-value-based explanations as feature importance measures," *arXiv:2002.11097 [cs, stat]*, Jun. 2020.