DEPARTMENT OF DIGITAL SYSTEMS

NCSR DEMOKRITOS
INSTITUTE OF INFORMATICS AND
TELECOMMUNICATIONS

Explainability in Human-Artificial Intelligence Collaboration

by

ILIAS KASIOTAKIS

Submitted

in partial fulfilment of the requirements for the degree of

Master of Artificial Intelligence

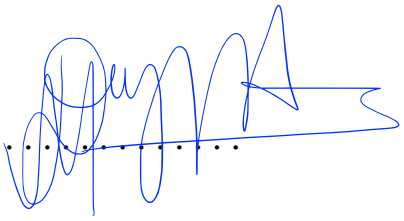
at the

UNIVERSITY OF PIRAEUS

June 2026

Author ILIAS KASIOTAKIS

II-MSc “Artificial Intelligence”
June 05, 2026

Certified by. . . . . 

Maria Dagioglou
Researcher (C) of NCSR “Demokritos”
Thesis Supervisor

Certified by. . . . . .

Christos Spatharis
Collaborating Researcher, NCSR “Demokritos”
Member of Examination Committee

Certified by. . . . . .

Georgios Vouros
Professor, University of Piraeus
Member of Examination Committee

Explainability in Human-Artificial Intelligence Collaboration

By

ILIAS KASIOTAKIS

Submitted to the II-MSc "Artificial Intelligence" on
June 05, 2026,
in partial fulfillment of the
requirements for the MSc degree

ABSTRACT

Recent advancements in Artificial Intelligence (AI) allow the development of AI methods that enable human-AI and human-robot collaboration. Here, collaboration is defined as the operation of humans and robots in a shared workspace to achieve common goals. Deep Reinforcement Learning (DRL) methods have been used recently for human-robot collaboration tasks allowing also co-learning of the task at hand. However, such methods, despite their ability to generate desirable behaviors, they cannot explain agents' behavior, i.e. why an agent decided to select a certain action over all other possible actions. The lack of explainability in AI methods has many implications, including reduced trust in human–AI interaction, as well as issues of transparency and accountability. In the present thesis, we use SHAP values as a method to explain the agent's behavior by analyzing how the features of the state space affect its selected actions. In our case, an agent is trained through a Soft Actor-Critic (SAC) DRL algorithm to learn how to collaborate with a human in order to collaboratively move a UR3 cobot towards a common spatial goal in real-time, in a simulation environment. Furthermore, we use KernelSHAP to compute feature importance within the agent's state and leverage these values to generate explanations based on data collected from an expert

human–agent team. Finally, we check whether the SHAP values align with human intuition.

ACKNOWLEDGEMENT

I thank everyone who has helped me to carry out this project to its completion.

Firstly, I would like to express my profound gratitude and deepest regards to Maria Dagioglou and Christos Spatharis who gave me the opportunity to investigate the field of Explainability in Human-Artificial Intelligence Collaboration thoroughly, as well as for their guidance and support during the conduction of this thesis.

Secondly, I would like to thank University of Peiraeus and National Centre For Scientific Research Demokritos for providing an inspiring academic environment and access to invaluable resources that greatly contributed to the completion of this work.

Furthermore, I would like to sincerely thank HELLEniQ ENERGY Holdings A.E. for providing the scholarship that supported my studies and the completion of this research.

Last but not least, I would like to thank my family and friends for their invaluable support and contributions throughout the years.

Ilias Kasiotakis

June 2026

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent technological innovations and the advent of the Fourth Industrial Revolution (Industry 4.0) have led to rapid development in the fields of the Internet of Things (IoT), automation, and analytics. Advances in automation have increased the use of robotic agents in industry and paved the way for the development of robots that collaborate with humans to accomplish common tasks, thereby improving performance and productivity while relieving users of task-related burdens.

This evolution has led to the growing importance of Human–Robot Collaboration (HRC), which studies the interaction between humans and robots working as a team to achieve a common goal. The development of Deep Reinforcement Learning (DRL) algorithms and their integration into robotic systems have further contributed to the advancement of HRC tasks in real-time scenarios. DRL enables the derivation of a robot's policy and, consequently, the modeling of its behavior during collaboration with a human. Developing optimal behavior is critical for successful task execution and for fostering a sense of trust and safety in the human user.

However, despite their impressive performance and promising results, DRL algorithms suffer from a major limitation: they lack the ability to explain their decisions and actions in a transparent manner. As a result, they are often considered "black-box" models, which makes them difficult to trust in safety-critical applications. In such contexts, an agent should be able to provide causal explanations for its decisions and reasoning

processes.

To address this limitation, eXplainable Artificial Intelligence (XAI) has emerged. XAI refers to machine learning and artificial intelligence systems that are capable of explaining their behavior in ways that are understandable to humans. Explanations facilitate effective collaboration between humans and autonomous or semi-autonomous agents by enabling users to understand why an agent succeeds or fails to achieve a goal or behaves unexpectedly, thereby ensuring transparency, explainability, and accountability.

However, the application of feature attribution methods to explain agent behavior in shared-control HRC tasks remains unexplored. Existing research has applied SHAP-based methods to single-agent navigation and manipulation tasks, but these settings lack the collaborative structure, where human partners must understand and adapt to the agent's decisions in order for the team to succeed.

In this work, we focus on generating explanations using the KernelSHAP explainer [3] for a HRC task. The objective of this study is to examine whether SHAP values, which represent the importance weights of features in the state space, align with human intuition and, consequently, produce explanations that lead to more transparent and interpretable collaboration. The collaboration is realized through a shared task that requires effective cooperation between a human and a Reinforcement Learning (RL) agent.

The structure of this thesis is organized as follows. Chapter 2 presents a solid theoretical background by reviewing the fundamentals of Machine Learning (ML) and RL, with particular emphasis on DRL and the Soft Actor–Critic (SAC) algorithm [4]. This chapter also introduces eXplainable Artificial Intelligence (XAI) methods, focusing on the computation and interpretation of SHAP values. Chapter 3 describes the approach and methodologies employed in this research, including the formulation of the DRL agent and the implementation of the KernelSHAP explainer. It also discusses the metrics and tools used to evaluate the effectiveness of SHAP values. Chapter 4 presents the experimental results, highlighting the role of SHAP in enhancing the explainability of the human–robot collaboration task. Finally, Chapter 5 presents the conclusions derived from the results and discusses future research directions.

Chapter 2

Background

In this chapter, we present the fundamental concepts of Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL), which model the behavior of an agent that interacts with its environment to perform a task. We place particular emphasis on the analysis of the SAC and SAC-Discrete algorithms. In addition, we present explainability methods of DRL systems, which aim to provide insights into the agent's behavior and its decision-making processes.

2.1 Reinforcement Learning

In this section, we analyze the fundamental concepts of RL based on the works of [1], [5] and [6]. RL is one of the four main types of Machine Learning (ML), and its goal is to train autonomous agents to develop a desired behavior in a dynamic environment. It is inspired by the idea that an entity—whether a person, an animal, or an artificially created agent—can learn how to act through interaction with its environment by receiving rewards or penalties.

2.1.1 The Formulation of Reinforcement Learning

The target in RL is to create an artificial autonomous agent that develops a desired behavior in order to achieve an objective. The agent operates within a simulated or physical environment and continuously interacts with it. By receiving information from the environment, the agent determines its current situation and acts accordingly. The way an agent decides its actions is called a policy π. After selecting an action, the agent transitions to a new environmental state and receives a reward signal that reflects the quality of its decision, either reinforcing or penalizing it. This procedure is repeated until the agent reaches its goal and achieves its objective. The series of actions taken until the goal is reached is called a path.

This procedure is mathematically modeled as a Markov Decision Process (MDP). MDP is a tuple (S, A, T, R, γ), where:

1. S: The state space, which includes every possible state in the environment.
2. A: The action space, which represents all the possible actions that can be performed by the agent.
3. T: The transition function $T : S \times A \times S \to [0, 1]$, where $T(s_{t+1}|s_t, a_t)$ shows the probability to reach state s_{t+1} after performing action a_t at state s_t.
4. r: The reward function $r : S \times A \to \mathbb{R}$, where $r(s_t, a_t)$ is the immediate reward received after taking action a_t in state s_t.
5. γ: The discount factor. Its value lies between 0 and 1 and is used to discount future rewards, reflecting the fact that future rewards are less certain and ensuring that the total accumulated reward remains bounded.

We can now formulate the process in mathematical terms. The state space is considered to be discrete, the agent acts at discrete time steps $t \in \mathbb{N}$ and it starts from an initial state $s_0 \in S$. At each timestep t the agent is in a state $s_t \in S$ and according to the policy π selects an action $a_t \in A$. Then, with a probability $T(s_t, a_t, s_{t+1})$, the agent transits from state s_t to a new state $s_{t+1} \in S$. After that, the agent receives a positive or a negative

reward $r(s_t, a_t)$ discounted by the discount factor γ. The procedure is repeated until the agent reaches its goal, encounters failure or after a predetermined number of time steps. The procedure is illustrated in Figure 2.1

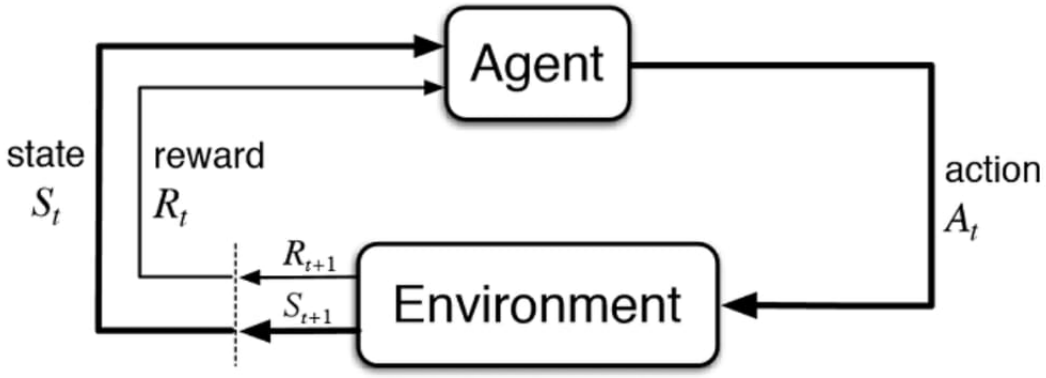


Figure 2.1: The agent–environment interaction in a MDP [1].

Policy π is a function that maps states to actions. According to the form of the function, the policy can be deterministic or stochastic. A deterministic policy $\pi : S \rightarrow A$ returns always the same action in a given state. This is achieved by the $argmax$ function, which returns the action with the hightest probability to be performed in the given state. In contrast, a stochastic policy $\pi : S \times A \rightarrow [0, 1]$ returns a probability distribution for a given state s_t and an action a_t is sampled from the distribution. Thus, the action is not necessarily the same for each given state.

After the agent completes the procedure, a sequence of states and actions, denoted by $\tau = \{s_0, a_0, s_1, a_1, ..., s_t, a_t\}$, is generated. This sequence is known as a trajectory, and the cumulative sum of rewards is then calculated: $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. In order to reach its goal with the most efficient way, the agent needs to discover the optimal policy π^*, which maximizes the cumulative sum of rewards:

$$\pi^* = \arg\max_{\pi} \sum_{t} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi}[r(s_t, a_t)] \tag{2.1}$$

where ρ_π represents the state–action distribution induced by policy π— that is, the probability distribution over state–action pairs (s_t, a_t) when the agent follows policy π in the

environment.

In order for this to be achieved, tools like state value function $V(s_t)$, action-value function $Q(s_t, a_t)$ and Bellman equations need to be employed.

State value function $V^\pi(s_t)$ indicates the expected total discounted return when starting from state s_t and following policy π and action-value function $Q^\pi(s_t, a_t)$ predicts the expected total discounted return from taking action a_t in state s_t, under policy π. Thus, the Q-value shows the effectiveness of actions in specific states. The functions are defined as follows:

$$V^\pi(s_t) = \mathbb{E}_\pi\{R_t\} = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right] \tag{2.2}$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi\{R_t\} = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right] \tag{2.3}$$

The Bellman Equation is a formula used to calculate the state value function $V(s_t)$ and the action state value function $Q(s_t, a_t)$. According to the Bellman function, the value of a state is equal to the reward received now plus the expected value of the next state. With that way the agent considers both immediate and future rewards. Thus, the state value function is also defined as:

$$V^\pi(s_t) = \mathbb{E}_\pi\left[r(s_t, a_t) + \gamma \sum_{s_{t+1}} T(s_{t+1}|s_t, a_t) V^\pi(s_{t+1})\right] \tag{2.4}$$

Similarly, the action state value function is estimated by:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi\left[r(s_t, a_t) + \gamma \sum_{s_{t+1}} T(s_{t+1}|s_t, a_t) Q^\pi(s_{t+1}, \pi(s_{t+1}))\right] \tag{2.5}$$

These Bellman equations play a crucial role in RL by breaking down the problem into smaller sub-problems, making it possible to find the optimal policy π^* and the associated optimal value functions $V^*(s_t)$ and $Q^*(s_t, a_t)$.

2.1.2 Reinforcement Learning Algorithms

RL algorithms differ in how the agent is trained, how it explores the environment, how the environment's dynamics are defined, and how the optimal policy is estimated.

There are two main RL categories according to the way the algorithm is trained: off-policy and on-policy learning.

In the case of off-policy learning, the agent has two distinct policies: a behavioral policy and a target policy. The behavioral policy is used by the agent to select actions in states and produce transitions of the form $(s_{t,t}\, a, s_{t+1})$, where s_t the current state, a_t the selected action in state s based on the behavioral policy and s_{t+1} the resulting next state. These transitions are then used to update iteratively the target policy. The target policy is trained to approximate the optimal policy. Therefore, the data collection process is distinct from policy training. A common practice with off-policy learning is to periodically update the behavioral policy with the latest target policy to maximize learning gain.

In contrast, in on-policy learning algorithms the target policy and the behavioral policy are the same. Thereafter, there is a common RL policy which is used to produce data during agent's interaction with the environment and these data are then used to iteratively refine the parameters of the same policy.

Depending on how we model the environment, there are Model-Free and Model-Based RL methods. In model-Free RL methods, the agent learns during its interactions with an environment that has not explicitly modeled dynamics. In contrast, in the case of Model-Based RL methods, the agent uses or learns a model of the environment to plan its actions.

There is also a significant challenge during data collection known as the exploration–exploitation trade-off. This dilemma arises when the agent must select an action during its interaction with the environment. The agent needs to choose between taking an action that is already known to be effective (*exploitation*) or trying an unexplored action that might lead to higher rewards (*exploration*). To discover the optimal policy, an agent must balance both exploration and exploitation effectively.

The estimation of the optimal policy can be succeeded by three main methodologies in RL:

1. Actor-Only: The Actor maximizes the expected return by optimizing the policy $\pi(s, a)$.

   A well known and widely used actor-only method is Policy Gradient. In Policy gradient the policy π is represented by a model parametrized by θ. This model is defined as π_θ. As mentioned in Chapter 2.1.1, the target is to estimate the optimal policy π^* that maximizes the expected reward $R(\tau)$ of a trajectory τ. So, the π_θ model needs to maximize the function $J(\theta) = \mathbb{E}_{\pi_\theta}[R(\tau)]$. Thus, the parameters θ are updated in the direction that increases the expected return:

   $$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.6}$$

   where α is the learning rate and the gradient $\nabla_\theta J(\theta)$ is calculated by:

   $$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) Q^{\pi_\theta}(s_t, a_t)] \tag{2.7}$$

   This procedure is called Gradient Ascent and is repeated iteratively to improve the policy.

   Even if traditional Actor-Only methods are particularly effective in environments with high-dimensional or continuous action spaces and are known for their ability to learn stochastic policies, optimization methods, such as gradient ascent, suffer from high variance in the estimates of the gradient, leading to slow learning.

2. Critic-Only: This method approximates the optimal policy by estimating the state value function $V(s_t)$ or the action-state value function $Q(s, a)$. This is achieved by a method called Temporal Difference (TD) Learning and it has lower variance in the estimate of expected returns than Actor-Only.

   In TD learning, the value function $V(s_t)$ or the action-state value function $Q(s_t, a_t)$ is estimated based on the current state and the observed immediate reward and

the estimated V-value or Q-value of the next state, respectively.

A fundamental TD method is TD(0), which estimates the $V(s_t)$ function and it is represented by the formula:

$$V(s_t) \leftarrow V(s_t) + \alpha[r + \gamma V(s_{t+1}) - V(s_t)] \tag{2.8}$$

where α is the learning rate, γ is the discount factor, r is the reward received after transitioning from state s_t to s_{t+1}, $V(s_t)$ represents the estimated value of state s_t, and $V(s_{t+1})$ is the estimated value of the next state s_{t+1}.

Another fundamental TD method is Q-learning, which estimates the $Q(s, a)$ value by the formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.9}$$

where $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ estimates the best action for the next state s_{t+1} according to the action state value function.

3. Actor-Critic: Actor-Critic methods are designed to combine the advantages of actor-only and critic-only approaches. They consist of an *Actor*, which represents the policy, and a *Critic*, which estimates the value function. The Critic uses temporal-difference (TD) learning to approximate the value function, and this estimate is then used to compute the policy gradient $\nabla_\theta J(\theta)$, which the Actor uses to update the policy π_θ. In this way, the policy is updated using a low-variance estimate of performance provided by the Critic, leading to faster and more stable learning. Actor-Critic methods generally exhibit better convergence properties than pure critic methods and have demonstrated superior performance compared to both actor-only and critic-only methods.

2.2 Deep Reinforcement Learning

Unfortunately, even if RL is a powerful tool, it faces serious challenges and limitations in tasks where the state and action spaces are not discrete but continuous. One of these challenges is scalability. In traditional RL methods each state and action needs to be enumerated. Even if continuous environments can be digitalized, this is going to increase the state and action spaces, which leads to computational challenges. Also, RL is characterized by low sample efficiency, as it demands a large number of samples to learn effective policies, and limited generalization, reducing its ability to transfer learned policies to unseen environments. These factors make it hard to apply RL methods in real world scenarios.

These limitations have led to the emergence of Deep Reinforcement Learning (DRL), which combines the decision-making framework of RL with the representational power of function approximators, such as Deep Neural Networks (DNN). Function Approximators are used in order to estimate the optimal policy π^* and the optimal state value $V^*(s)$ and action state value $Q^*(s, a)$ functions in large or continuous state spaces. Neural networks are often used in DRL for function approximation due to their ability to model complex, non-linear relationships. This integration addresses the scalability and generalization issues of traditional RL, enabling applications in more complex environments.

In this chapter, we examine several challenges in DRL and discuss how these challenges are addressed by different algorithms and techniques, including Twin Delayed Deep Deterministic Policy Gradient (TD3) [7], Deep Deterministic Policy Gradient (DDPG) [8], and Proximal Policy Optimization (PPO) [9]. In addition, we provide an in-depth explanation of the SAC [10] and SAC-Discrete [11] algorithms.

2.2.1 Challenges and Solutions in Deep Reinforcement Learning

The use of DNN introduces new challenges for DRL. One of the most prominent issues is overestimation bias. For example, in Q-learning with discrete actions, the value estimate is updated with a greedy target $y = r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$, but if the calculated target y differes from the actual target by an error ϵ, then the maximum over the approximated Q-value will be greater than the true maximum: $\mathbb{E}_\epsilon[max_{a_{t+1}}(Q(s_{t+1}, a_{t+1}) + \epsilon)] \geq max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$. Thus, the error causes a consistent overestimation bias, which is then propagated through the Bellman equation. Since the function approximation always causes an error, overestimation bias is inevitable [7].

Overestimation bias is also present and has effects in actor-critic settings where the policy is updated via gradient descent. Even a minimal overestimation bias of the value estimate, it may be significantly increased over many updates and it may lead to poor policy updates. As a result, there is a suboptimal critic which will highly rate suboptimal actions, leading to the reinforcing of the suboptimal action in the next policy update [7].

The overestimation bias was addressed during the development of TD3 algorithm [7], which is an advanced off-policy RL algorithm designed for continuous action spaces. A pair of actors $(\pi_{\phi_1}, \pi_{\phi_2})$ and critics $(Q_{\theta_1}, Q_{\theta_2})$ is used, where π_{ϕ_1} is updated with respect to Q_{θ_1} and π_{ϕ_2} is updated with respect to Q_{θ_2}:

$$y_1 = r + \gamma Q_{\theta'_2}(s_{t+1}, \pi_{\phi_1}(s_{t+1})) \tag{2.10}$$

$$y_2 = r + \gamma Q_{\theta'_1}(s_{t+1}, \pi_{\phi_2}(s_{t+1})) \tag{2.11}$$

where $\theta'_i \leftarrow \tau\theta + (1-\tau)\theta'_i$ for i=1,2. As a result, π_{ϕ_1} optimizes with respect to an independent estimate of Q_{θ_1}, which helps avoid the bias introduced by the policy update. The same holds for π_{ϕ_2}.To mitigate the overestimation bias introduced by the critics, the minimum between the two estimates Q_{θ_2} and Q_{θ_1} is taken:

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s_{t+1}, \pi_{\phi_1}(s_{t+1})) \tag{2.12}$$

Another fundamental challenge in DRL is the issue of variance, which refers to the sensitivity of value and policy estimates to stochasticity in the environment, sampling noise, and function approximation errors. High-variance estimates introduce significant noise into the policy gradient updates, which can slow down learning and degrade performance in practice [7].

In several DRL algorithms, such as DDPG, TD3, and SAC, variance is mitigated through the use of stable target networks and delayed policy updates. Target networks are copies of the actor and or the critic networks, respectively, that are updated by having them slowly track the learned networks. The slowly change of the target values ensures that the residual error produced in each update remains small and it reduces the likelihood of repeatedly updating the critic with respect to rapidly shifting targets, greatly improving the stability and smoothness of learning [7] [8].

Also, the use of DNN adds an extra challenge in DRL algorithms. DNN assume that training samples are independently and identically distributed, adding an extra challenge in DRL, but this assumption does not hold in RL, where samples are generated sequentially through interactions with the environment, leading to strong temporal correlations. In addition, efficient utilization of hardware accelerators requires learning from mini-batches [8]. To address these issues, DRL algorithms employ a replay buffer, which is a finite-sized cache R. Transitions are sampled from the environment according to the policy of the agent and the tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer. When the replay buffer is full the oldest samples are discarded and the actor and or the critic are updated by sampling a mini-batch uniformly from the buffer at each timestep [8].

2.2.2 Soft Actor Critic (SAC)

Soft Actor-Critic (SAC) is an off-policy maximum entropy DRL algorithm that provides sample-efficient learning while retaining the benefits of entropy maximization and stability. It was introduced by Haarnoja in [10] and in [4].

SAC was created to tackle two major challenges in model-free deep RL:

1. *High sample complexity*. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more.
2. *Brittleness to hyperparameters*. Learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results.

Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

SAC is based on the maximum entropy framework, which has a different learning objective from standard RL. Instead of simply finding the policy that maximizes the total reward, an additional entropy term is introduced. This entropy term represents how randomly the agent selects an action. Thus, SAC's optimal policy maximizes the expected return and its entropy at each visited state, meaning it solves the task while acting as randomly as possible. This is expressed by the following equation:

$$\pi^* = arg\ \underset{\pi}{max} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))] \tag{2.13}$$

where $\mathcal{H}(\pi(\cdot \mid s_t))$ is the entropy term of policy π given the state s_t and α is the temperature parameter that determines the relative importance of the entropy term versus the reward, and thus controls the stochasticity of the optimal policy. Also, a discounted factor γ can be introduced to ensure that the sum of the expected rewards is finite.

The maximum entropy objective has two main advantages:

1. The policy explores more widely, but in the same time it gives up on clearly unpromising avenues.
2. The policy can capture multiple modes of near-optimal behavior and when multiple actions seem equally attractive, all of them will have the same probability to be selected by the policy.

Function approximators are used to calculate both the Q-values and the policy. The function approximator employed for the modeling of the soft Q-function is a DNN,

which is the critic network. Another DNN is used to produce the mean and the covariance of the policy, which is expressed as a Gaussian, and it is the actor network. Both DNN are optimized with stochastic gradient descent. As such, there is a parameterized soft Q-function $Q_\theta(s_t, a_t)$ and a tractable policy $\pi_\phi(a_t, s_t)$. The parameters of these networks are θ and ϕ, respectively.

The soft Q-function parameters are trained to minimize the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \left(r(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}\left[V_{\bar{\theta}}(s_{t+1})\right]\right)\right)^2\right] \tag{2.14}$$

where $\mathcal{D}$ is the replay buffer. Also, $J_Q(\theta)$ can be optimized with stochastic gradients:

$$\widehat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)\left(Q_\theta(s_t, a_t) - \left(r(s_t, a_t) + \gamma\left(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha\log\left(\pi_\phi(a_{t+1}|s_{t+1})\right)\right)\right)\right). \tag{2.15}$$

A target soft Q-function with parameters $\bar{\theta}$ is used during the updates to stabilize training. $\bar{\theta}$ is expressed as an exponentially moving average of the soft Q-function weights. The policy parameters are updated in order to minimize the expected KL-divergence:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}[\mathbb{E}_{a_t\sim\pi_\phi}[\alpha log\pi_\phi(a_t \mid s_t)) - Q_\theta(s_t, a_t)]] \tag{2.16}$$

Because the action a_t is sampled from the probability density returned by the policy π, the reparameterization trick is employed to make the gradient calculation tractable. Thus, the action is calculated by a neural network transformation f parameterized by ϕ:

$$a_t = f_\phi(\epsilon_t; s_t) \tag{2.17}$$

where ϵ_t is an input noise vector, sampled from some fixed distribution, such as a spherical Gaussian. Thus, the Equation 2.16 is rewritten as:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}[\alpha log\pi_\phi(f_\phi(\epsilon_t; s_t) \mid s_t)) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))] \tag{2.18}$$

The gradient is calculated as:

$$\widehat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi\alpha\log\left(\pi_\phi(a_t|s_t)\right) + \left(\nabla_{a_t}\alpha\log\left(\pi_\phi(a_t|s_t)\right) - \nabla_{a_t}Q(s_t, a_t)\right)\nabla_\phi f_\phi(\epsilon_t; s_t) \tag{2.19}$$

where a_t is evaluated as $f_\phi(\epsilon_t; s_t)$.

So far, the temperature hyperparameter α has been considered constant. As a result, SAC can suffer from brittleness with respect to the temperature hyperparameter. To tackle this issue, an automatic gradient-based temperature tuning method is employed. The tempereture hyperparameter is updated to minimize the following objective function by computing its gradients:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t}[-\alpha log \pi_t(a_t \mid s_t) - a\bar{H}] \tag{2.20}$$

where $\bar{H}$ is a desired minimum expected entropy.

Lastly, the algorithm makes use of two soft Q-functions Q_1 and Q_2 to speed up training and mitigate positive bias in the policy improvement step. Each soft Q-function i, where $i = \{1, 2\}$, is parameterized with parameters θ_i and is trained independently to optimize $J_Q(\theta_i)$. Then, the minimum of the two soft Q-functions is used for the stochastic gradient in Equation 2.15.

Algorithm 1 Representation of the SAC algorithm

Input: Initialize parameters and an empty buffer: $\theta_1, \theta_2, \phi, \bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2, \mathcal{D} \leftarrow \emptyset$
  for each iteration **do**
    for each environment step **do**
      $a_t \sim \pi_\phi(a_t \mid s_t)$ ▷ Sample action from the policy
      $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$ ▷ Sample transition from environment
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$ ▷ Store transition to the buffer
    end for
    for each gradient step **do**
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the parameters of the actor
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update the parameters of the policy
      $\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J(\alpha)$ ▷ Update tempereture hyperparameter
      $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target networks parameters
    end for
  end for
Output: θ_1, θ_2, ϕ

2.2.3 Discrete Soft Actor-Critic (SAC)

An adaptation of the SAC algorithm was proposed by Christodoulou in [11] for discrete action spaces, knows as SAC-Discrete. That led to the modification of the equations 2.14, 2.18 and 2.20:

1. The policy a finite selects an action from a distinct set. As a result, it directly outputs the action distribution using a softmax function in the output layer, changing from $\pi : S \rightarrow \mathbb{R}^{|2A|}$ to $\pi : S \rightarrow [0, 1]^{|A|}$.

2. The soft Q-function outputs the Q-value for each possible action instead of just the input action, transitioning from $Q : S \times A \rightarrow \mathbb{R}$ to $Q : S \rightarrow \mathbb{R}^{|A|}$.

3. The action distribution is now completely determined by the softmax, so the soft state-value is calculated by:

$$V^{\pi}(s_t) = \pi(s_t)^T [Q^{\pi}(s_t) - \alpha log(\pi(s_t))] \tag{2.21}$$

   Similarly, the temperature objective is calculated by:

$$J(\alpha) = \pi_t(s_t)^T [-\alpha(log\pi_t(s_t) + \bar{H})] \tag{2.22}$$

4. The reparameterization trick is no longer needed because the action distribution is determined. As a result, a direct calculation in the policy objective is allowed:

$$J_{\pi}(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\pi_t(s_t)^T \left[\alpha \log \pi_{\phi}(s_t) - Q_{\theta}(s_t) \right] \right] \tag{2.23}$$

Algorithm 2 Soft Actor-Critic with Discrete Actions (SAC-Discrete)

Input: Initialise $Q_{\theta_i} : S \rightarrow \mathbb{R}^{|\mathcal{A}|}$, $\pi_\phi : S \rightarrow [0,1]^{|\mathcal{A}|}$ ▷ Initialise local networks
Initialise $\bar{Q}_{\theta_i} : S \rightarrow \mathbb{R}^{|\mathcal{A}|}$ ▷ Initialise target networks
$\bar{\theta}_i \leftarrow \theta_i$ ▷ Equalise target and local network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialise an empty replay buffer

for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t \mid s_t)$ ▷ Sample action from the policy
    $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$ ▷ Store the transition in the replay buffer
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J(\alpha)$ ▷ Update temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for

Output: θ_1, θ_2, ϕ

2.3 Explainability in Deep Reinforcement Learning

While the results of DRL algorithms are promising, they lack the ability to provide explanations for their decisions and reasoning. This is happening because the Artificial Neural Networks (ANN) implemented to estimate the Actor or and the Critic behave like "black boxes". This is due to the vast number of parameters and the intricate, non-linear interactions between them, making it difficult to explain exactly why a specific decision was made. That makes it difficult to trust DRL algorithms in safety-critical applications.

For this reason, eXplainable Artificial Intelligence (XAI) was developed. XAI are ML or AI systems which can generate explainations about their behavior that can be understood by humans. The integration of XAI systems into autonomous or semi-autonomous agents enables them to explain to humans why they fail to achieve a goal or unexpectedly complete a task. This is really important when a human collaborates with an agent,

because they can understand the dynamics driving the agent's actions and decide how to respond to that behavior. The ability of AI systems to provide explanations for how they produce results, make decisions, and act is particularly important in safety-critical applications in order to ensure transparency, explainability, and accountability [12].

XAI consists of two major categories, Data-driven AI and Goal-driven XAI (XGDA) [12]. Data-driven AI aims to the development of interpretable ML models. A ML system is interpretable if a human can understand its operations through explanation or introspection. This is achieved by determining how the input attributes contribute to the output predictions. In addition, Data-driven AI aims to show if the ML process can reliably replicate the same decision, given similar data and specific circumstances.

Goal-driven XAI (XGDA) is a specific domain of XAI which aims to create explainable robots or agents that can justify their behaviors to a lay user. The explanations would assist the user in creating a Theory of Mind (ToM), comprehending the agent's behavior, and contribute to greater collaboration between the user and the agent.

In this chapter, we present explainability techniques in XGDA, focusing specifically on explainability for reactive agents. Reactive agents implement simple behavioral patterns and respond to events in their environment in a stimulus–response manner, without requiring any internal model of the world. They are typically created using model-free RL. Some notable explainability works in RL include Memory-based eXplainable Reinforcement Learning (MXRL), Minimal Sufficient eXplanation (MSX) via Reward Decomposition and Reward Augmentation and Repair through Explanation (RARE)[12]. Also, we are going to present SHapley Additive exPlanation (SHAP). Even if SHAP is used mainly for Data-driven AI, it is used to explain the behaviour of reactive agents.

2.3.1 Memory Based eXplainable Reinforcement Learning (MXRL)

MXRL is an explainability strategy for reinforcement learning algorithms, in which the model uses episodic memory to store each episode — that is, the agent's record of executed state–action combinations. Then, the information obtained from memory can

be used with two main ways:

1. To calculate the agent's probability of success P_s and the number of transitions to the goal N_t [13].
2. To extract interestingness elements that help explain its behavior [2].

In the first case, the idea was based on the fact that the $Q(s_t, a_t)$ values can be used to explain why an agent favors an action in comparison with another. The $Q(s_t, a_t)$ shows the expected total future reward an agent can obtain when it is in state s_t and takes action a_t. The agent will therefore select the action with the highest Q-value for the given state. Even if this can show we an agent favors an action in a given state, this kind of explanation makes little sense for non-expert users who need to be given explanations using domain-like language in order to allow them to fully understand the agent behavior.

For this reason, the authors of [13] compute the success probability P_{s_t} and the transitions to the goal N_t consisting of an RL agent with an episodic memory. They implement a list of state-action pairs T_{List}, comprising the transactions the agent performed during its learning process. To compute the success probability P_{s_t}, they compute the total number of transitions T_t and the number of transitions involved in a success sequence T_{s_t}, which is obtained by the transactions previously saved into the list T_{List}. Each time the agent reaches its goal state, they compute the probability $P_{s_t} = \frac{T_{s_t}}{T_t}$ considering the transitions N_t involved in the path towards the goal state, which are computed when an episode is completed. Unfortunately this leads to a rapid increasement in the use of memory and computer resources, making this approach unsuitable to RL tasks where the state and action space are continuous or of high dimensionality.

For this reason, the authors in [14] created a formula to calculate the possibility of success P_{s_t} in DRL:

$$\hat{P}_{s_t} \cong \left[(1-\sigma) \times \left(\frac{1}{2}\log_{10}\left(\frac{Q^*(s_t,a_t)}{R^T}\right)+1\right)\right]_{\hat{P}_{s_t}\geq 0}^{\hat{P}_{s_t}\leq 1} \tag{2.24}$$

where R^T is the reward received upon completing the task and σ is a parameter that denotes the stochasticity of the transitions.

In the second case, after analyzing the agent's history of interaction with the environment, they extract interestingness elements, which are properties that can help identify "interesting" moments of behaviour during the interaction. The interstingness elements help humans to identify key characteristics and abilities of the agent during its interactions, hus making the RL system more explainable. The interestingness elements are calculated by statistical analysis and ML methods to data collected during the interaction. One disadvantage of this method is that it suffers from limitations with regard to the utilization of memory in broad solution spaces and it is tried only in discrete state and action spaces.

Dimension	Generated Interestingness Elements
Frequency	(in)frequent situations: *situations the agent finds very (un)common*
Execution Certainty	(un)certain executions: *hard/easy to predict situations with regards to action execution*
Transition-Value	minima and maxima: *favorable and adverse situations*
Sequence	most likely sequences to maxima: *capture the agent's learned strategy*

Figure 2.2: Overview of the generated interestingness elements in [2]

2.3.2 Minimal Sufficient eXplanation via Reward Decomposition (MSX)

In MSX the explainability is provided by recognizing the usefulness of the selected action compared to the others based on the value returned by the Q-function. The main problem is that raw Q-values cannot provide information about how each factor contributed to the agent's preferences. For this reason, Reward decomposition breakes the environment's reward into a sum of semantically meaningful reward types. Each reward type corresponds to a different factor and the agent learns a separate component Q-function for each reward type [15].

According to [15], reward decomposition is achieved in the MDP formulation by the specification of C reward types and by defining a vector-valued reward function $\vec{R}$:

$S \times A \to \mathbb{R}^{|C|}$, where $R_c(s_t, a_t)$ is the reward for type $c \in C$. The objective is to optimize the overall reward function:

$$R(s_t, a_t) = \sum_{c \in C} R_c(s_t, a_t). \tag{2.25}$$

Since the reward is expressed as a vector, Q-function is also expressed as the vector $\vec{Q}^{\pi}$, where $Q_c^{\pi}(s_t, a_t)$ gives action values that account for the reward type c. Thus, the overall Q-function is also decomposed as

$$Q^{\pi}(s_t, a_t) = \sum_{c} Q_c^{\pi}(s_t, a_t). \tag{2.26}$$

Then, greedy exploration is used to find the best action a_t by considering the overall reward R and the update is carried on each Q network branch $Q_c^{t+1}(s_t, a_t)$ separately, allowing each Q_c to converge the overall greedy policy.

Algorithm 3 Table-Based Decomposed Reward RL for drQ, HRA, and drSARSA

$s_0 \leftarrow$ Initial State
$a_0 \leftarrow \epsilon(Q^0, s_0)$
$t \leftarrow 0$
repeat
  $(s_{t+1}, \vec{r}_t) \leftarrow \text{Act}(a_t)$
  $a_{t+1} \leftarrow \epsilon_t(Q^t, s_{t+1})$ ▷ ϵ-greedy exploration
  for all $c \in \mathcal{C}$ **do**
    if drQ **then**
      $a' \leftarrow \arg\max_a \sum_c Q_c^t(s_t, a)$
    else if HRA **then**
      $a' \leftarrow \arg\max_a Q_c^t(s_t, a)$
    else if drSARSA **then**
      $a' \leftarrow a_{t+1}$
    end if
    $Q_c^{t+1}(s_t, a_t) \leftarrow Q_c^t(s_t, a_t) + \alpha_t \left[r_{t,c} + \gamma Q_c^t(s_{t+1}, a') - Q_c^t(s_t, a_t) \right]$
    $t \leftarrow t + 1$
  end for
until convergence

According to [15] and [16], reward decomposition in the hands of an RL practitioner

allowes for spotting certain “bugs” in the agent’s action values and even help identify more fundamental issues related to the interaction of the gradient optimizer and the RL loop. Also, in the reward decomposition technique, more categories of rewards can be found to have an even deeper insight.

2.3.3 Reward Augmentation and Repair through Explanation (RARE)

Reward Augmentation and Repair through Explanation (RARE) was developed by [17] and addresses the need for establishing a shared behavior mental model between an RL agent and a human collaborator using a timely update to the reward function. In [17], they make the assumption that sub-optimal collaborator behavior is a result of misinformation of the task. Robots and humans collaborate effectively when there is a shared mental model amongst the teammates. Thus, an incongruous model is a problem and there is the need of a mechanism to enable an autonomous system to detect model disparity between itself and the human collaborator and profile human-interpretable feedback to encourage model correction.

For this reason, they made an autonomous agent which infers the most likely reward function as a basis of human behaviour and it identifies the missing piece of the reward function. This is achieved by a combination of Hidden Markov Models and Partially Observable Markov Decision Process. Then, the agent communicates back to the human and provides him with info to how to update his reward function and policy. However, RARE faces several challenges: it requires an expert agent for the human’s task, the study was conducted in a discrete environment, the human participant had unlimited time to take an action, and the robot interrupted the experiment to provide the information.

2.3.4 SHapley Additive exPlanation (SHAP)

SHapley Additive exPlanation (SHAP) values is a unified framework for interpreting predictions and it can explain any machine learning model output. In many applica-

tions the understanding of how a model made a specific prediction is as important as the prediction's accuracy. Unfortunately, complex models cannot be interpreted, creating a tension between accuracy and interpretability. SHAP values are computed to determine the impact of a data point or feature on the model's predictions and they assign to each feature an importance value for a particular prediction [3].

SHAP values is an Additive Feature Attribution Method, which approximate a complex model by a simpler interpretable model, the explanation model. Let f be the original prediction model to be explained and g the explanation model. g is a local method, i.e. it explains a prediction $f(x)$ based on a single input x. Explanation models often use simplified inputs x' that map to the original inputs through a mapping function $x = h_x(x')$. Local methods try to ensure $g(z') \approx f(h_x(z'))$ whenever $z' \approx x'$.

Accoring to [3], Additive feature attribution methods are defined as an explanation model that is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2.27}$$

where $z' \in \{0,1\}^M$, M is the number of simplified input features, ϕ_0 is an effect, which is most of the times a reference input, and $\phi_i \in \mathbb{R}$ for $1 \leq i \leq M$ an effect attributed to each feature by the explanation model g. Summing the effects of all feature attributions approximates the output f(x) of the original model.

SHAP has its roots in cooperative game theory and it is based on the Shapley decomposition, introduced by Lloyd Shapley in 1953 [18]. This decomposition fairly assigns parts of the outcome of a game to the game's players based on their contribution to the game's outcome. In the context of SHAP, these "players" are the input features, and the game's outcome is the output of the ML model [19].

SHAP values are feature importances for linear models in the presence of multicollinearity [3]. In this method the model must be retrained on all feature subsets $S \subseteq F$, where F is the set of all features. The target is to compute an importance value for each feature that represents the effect on the model prediction of including that feature. To compute

this effect, a model $f_{S\cup\{i\}}$ is trained with the feature i present, and another model f_S is trained with the feature withheld. Then, predictions from the two models are compared on the current input:

$$f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S) \tag{2.28}$$

where x_S represents the values of the input features in the set S. Since the effect of withholding a feature depends on other features in the model, the preceding differences are computed for all possible subsets $S \subseteq F\backslash\{i\}$. The Shapley values are then computed and used as feature attributions. They are a weighted average of all possible differences:

$$\phi_i = \sum_{S\subseteq F\backslash\{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \left[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)\right]. \tag{2.29}$$

For Shapley regression values, h_x maps 1 or 0 to the original input space, where 1 indicates the input is included in the model, and 0 indicates exclusion from the model. In SHAP ϕ_0 is equal to the baseline value, which is the average prediction the model makes over a background dataset. According to the definition of Additive feature attribution methods by [3], the equation of SHAP values can be expressed also by:

$$\phi_i(f, x) = \sum_{z'\subseteq[M],\, i\in z'} \frac{|z'|!\,(M - |z'| - 1)!}{M!} \left(f_x(z') - f_x(z' \setminus \{i\})\right). \tag{2.30}$$

where $|z'|$ is the number of non-zero entries in z', and $z' \cup x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries on x'.

SHAP values is the only additive feature attribution method that satisfies three desirable properties:

1. Local Accuracy: The approximation produced by the explanation model g given a simplified input x' matches the output of the original model f for a specific input x, when $x = h_x(x')$.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{2.31}$$

2. Missingness: Features that are missing in the original have no attributed impact

to the result of the approximation model g:

$$x'_i = 0 \implies \phi_i = 0 \tag{2.32}$$

3. Consistency: If a model changes so that some simplified input's contribution increases or stays the same regardless of the other inputs, that input's attribution should not decrease. Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$. For any two models f and f', if

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i) \tag{2.33}$$

for all inputs $z' \in \{0, 1\}^M$, then

$$\phi_i(f', x) \geq \phi_i(f, x). \tag{2.34}$$

SHAP values are meant to explain any model but they face two major challenges [19]:

1. This formula considers all possible subsets S, making SHAP values computationally intensive to calculate exactly.
2. It is generally impossible to evaluate neural network models with missing features.

For this reason the SHAP values are approximated. SHAP values are computed using a simplified input mapping, $h_x(z') = z_S$, where z_S has missing values for features not in the set S and S is the set of non-zero indexes in z'. Since most models cannot handle arbitrary patterns of missing input values, $f(z_S)$ is approximated with $\mathbb{E}[f(z) \mid z_S]$. In addition, $\bar{S}$ is the set of features not in S. For the SHAP values approximation feature independence and model linearity are two optional assumptions simplifying the

computation of the expected values:

$$f_x(z') = f(h_x(z')) = \mathbb{E}[f(z) \mid z_S] \quad \text{SHAP explanation model simplified input mapping} \tag{2.35}$$

$$= \mathbb{E}_{z_{\bar{S}}|z_S}[f(z)] \quad \text{expectation over } z_{\bar{S}} \mid z_S \tag{2.36}$$

$$\approx \mathbb{E}_{z_{\bar{S}}}[f(z)] \quad \text{assume feature independence} \tag{2.37}$$

$$\approx f([z_S, \mathbb{E}[z_{\bar{S}}]]) \quad \text{assume model linearity} \tag{2.38}$$

There are two main methods approximating SHAP values: Kernel SHAP and DeepSHAP.

Kernel SHAP

Kernel SHAP is a model agnostic method and it assumes feature independence. The SHAP values are the weights of a linear explanation model which is optimized by minimizing the following objective function:

$$\xi = \arg\min_{g \in \mathcal{G}} L(f, g, \pi_{x'}) \tag{2.39}$$

where:

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}\,|z'|\,(M-|z'|)}, \tag{2.40}$$

$$L(f, g, \pi_x) = \sum_{z' \in Z} \left[f(h_x^{-1}(z')) - g(z')\right]^2 \pi_x(z') \tag{2.41}$$

DeepSHAP

While Kernel SHAP can be used on any model, DeepSHAP is tailored for deep learning models and leverages neural network structure to efficiently approximate SHAP values. DeepSHAP leverages extra knowledge about the compositional nature of deep networks to improve computational performance. DeepSHAP propagates DeepLIFT multipliers backward through the network (as illustrated in Figure 2.3) to obtain an approximation to SHAP values:

$$m_{x_j f_3} = \frac{\phi_j(f_3, x)}{x_j - \mathbb{E}[x_j]} \tag{2.42}$$

$$\forall j \in \{1, 2\}, \quad m_{y_j f_3} = \frac{\phi_j(f_j, y)}{y_j - \mathbb{E}[y_j]} \tag{2.43}$$

$$m_{x_j f_3} = \sum_{j=1}^{2} m_{y_j f_3} \, m_{x_j f_j} \qquad \text{(chain rule)} \tag{2.44}$$

$$\phi_j(f_3, y) \approx m_{y_j f_3} \, (y_j - \mathbb{E}[y_j]) \qquad \text{(linear approximation)} \tag{2.45}$$

where $m_{\Delta x_i \Delta_y}$ the Deeplift multipliers.

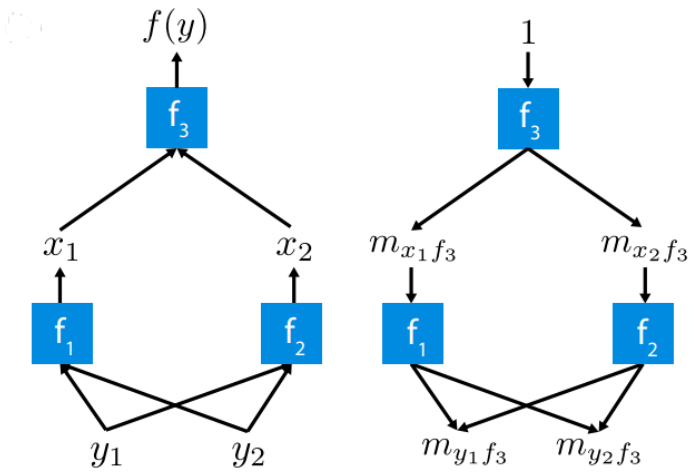


Figure 2.3: Compositional models such as DNNs are comprised of many simple components. Given analytic solutions for the Shapley values of the components, fast approximations for the full model can be made using DeepLIFT's style of back-propagation [3].

DeepLift is an additive feature attribution method developed by [20] that attributes for each input x_i a value $C_{\Delta x_i \Delta y}$ that represents the effect of that input being set to a reference value as opposed to each original value, where $\Delta x_i = x_i - r_i$, $\Delta y = y - y_r$, r is the reference input value, y is the original output and y_r is the reference output. The reference value is chosen by the user and represends a background value. It uses both Back Propagation and the Chain Rule for the effects' calculation. DeepLift multipliers are defined as:

$$m_{\Delta x_i \Delta_y} = \frac{C_{\Delta x_i \Delta y}}{\Delta x_i}$$

2.3.5 SHAP values in Deep Reinforcement Learning agents

SHAP values have been integrated to explain the behaviour of DRL agents. In [19], they employed SHAP to interpret a DRL agent's decisions, trained using the SAC algorithm for controlling a TurtleBot3 via LiDAR in a simulated environment. The SHAP explainer was used to compute feature attributions from the inputs to the policy to the mean action chosen by the policy, with SHAP values approximated using DeepSHAP method. Their results demonstrated that SHAP can provide valuable insights into the agent's behavior, revealing that many decisions were based on human-interpretable features. However, they also observed cases where the agent relied on non-intuitive information, suggesting potential overfitting to the specific training environment.

The same authors in [21] used the KernelSHAP on a DNN trained using DRL on the task of controlling a lever using a robotic manipulator. The agent was trained using the DDPG algorithm as developed in [8] together with the technique Hindsight Experience Replay (HER) [22]. They collected a dataset 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. They identified a collection of interesting events and compared the explanations generated by the two different SHAP implementations on these. They removed the episodes where these events occured and they used the resulting dataset as a background dataset.

In [23], SHAP were adopted to provide an interpretable framework for an open-source platform called Reinforcement Learning for Grid Control (RLGC), with the DeepSHAP

explainer selected for interpreting the DRL agent. The proposed SHAP-based explainable AI (XAI) framework consisted of four components: the SHAP inputs, the SHAP core algorithm, the SHAP outputs, and the SHAP tasks. The SHAP inputs comprised three elements: (i) the DRL model, including observations and actions recorded at each training step; (ii) the dataset used for explanation; and (iii) the model's network architecture. The core algorithm employed the DeepSHAP explainer to compute feature attributions. The resulting SHAP values served as the XAI output, while explanation and diagnosis constituted the primary XAI tasks. The authors demonstrated that SHAP values significantly enhance the interpretability of DRL agents, making their decision-making processes clearer and easier to understand.

2.4 Human Robot Collaboration (HRC)

According to [24], Human Robot Collaboration (HRC) is the approach that explores the interaction between a human and a robot acting like a team in reaching a common goal, both in cognitive and physical level. In HRC tasks Robots and humans share the same workspace and they jointly work. This field of robotics is called cobotics and the robots in these applications are called cobots. The target of Human Robot Collaboration is to increase performance and productivity and to relieve the user from the burden of the task.

HRC tasks are categorized into two main families according to the side of the task: cognitive and physical tasks. Cognitive tasks include collaborative assembly and object handover, while physical tasks incude collaborative manufacturing and object handling:

1. Collaborative Assembly: The robot has to assist the human to the completion of subtasks in order to assemble an object. For example, in [25] a human hits some objects with a known sequence and a robot must recognize the sequence and collaborate with the human to perform the last part of the sequence.
2. Object handover: The user and the robot has to exchange objects. The robot has to understand the users intentions and expectations in the reception of the object.

For example, in [26] a robot needed to touch a different pad from a human and avoid collisions with them.

3. Collaborative manufacturing:The robot applies permanent physical alteration to an object in collaboration with the human. For example, in [27] a human has to lift an object and a robot assists the lifting of the object.

4. Object handling: The user with the robot, jointly grab an object and move it. For example, in [28], a robot and a human collaborated to move an object from a point A to a point B.

Another important aspect of HRC is the degree of robot autonomy. According to [29], the degree of automation of a robot is defined as the amount of time a human intervented to complete the collaborative task by controling directly the robot and it ranges from full human control, where the human performs all aspects of the task, to fully autonomous robot behavior, where the robot performs all aspects of the task autonomously. Intermediate levels of automation include shared control schemes, where humans supervise or intervene during environmental sensing, planning, and action execution. As robot autonomy increases, their implementation in more realistic dynamic environments becomes more challenging. Since robots may encounter unseen situations, the likelihood of errors is increased, having a serious effect in the reliability of the system, especially if humans cannot diagnose the problem in time and react. Therefore, transparency and explainability are essential for increasing reliability and maintaining trust in HRC systems. Trust in HRC refers to the willingness of humans to cooperate with robots under uncertainty [30]. Several factors influence trust, including robot behavior, reliability, level of automation, adaptability, communication, anthropomorphism, proximity, and task complexity [30, 31]:

- Robot behaviour: To what extent can the robot adapt to the diverse needs of each individual in real time.
- Reliability: People tend to trust machines more, when they expect reliable behavior.
- The Level of automation: People trust robots more when they can interact and collaborate with them, rather than when robots are completely autonomous.

- Proximity: The physical or virtual presence of a robot.
- Adaptability: A robot teammate capable of emulating the behaviours and teamwork strategies observed in human teams.
- Anthropomorphism: People tend to trust anthropomorphic robots more, but a lifelike appearance may cause a sudden transition from empathy to repulsion.
- Communication: Confidence levels fluctuated depending on the transparency and detail included in communication between robots and humans.
- Task type: The nature and complexity of the task affect the level of trust during interaction.

DRL methods have been introduced in HRC because they enable robots to learn complex behaviors, adapt to dynamic environments, and make real-time decisions based on human actions [32, 33]. In addition, DRL can continuously improve through interaction while incorporating safety constraints into robot decision-making [34, 35]. For these reasons, both physical and virtual environments are developed that can be used to monitor and evaluate human-agent collaborations. For example in [36], they use a robotic arm to create an environment in which a human and an agent must cross certain paths to reach the goal, where the agent controls the robot's movement on one axis and the human controls the other. Another example is presented in [37], where they developed a HRC framework that includes a robotic manipulator that controls the angular motion of a platform around one axis and a human user that controls the other. The objective involves shifting a ball from a starting corner to a designated target location, while in the same time avoiding obstacles placed on the platform. In [38], they created a visual simulation of this work in order to evaluate different training approaches. Then, in the work of[39], a Transfer Learning (TL) methodology was applied, called Deep Q-learning from Demonstrations, to examine how this can enhance the HRC.

A serious disadvantage of DRL is that it does not allow the robot to provide explanations about its behavior and decision-making process, which leads to a reduction in the transperacy, trustworthiness and reliability of the system in safe-critical applications. For this reason, XAI methods are introduced into HRC in order to make the system more explainable. One example is the use of RARE in [17], where a human and a robot

play a color-based collaborative Sudoku variant on a table with a grid overlay using colored toy blocks. The robot is an expert in the game and provides the human with directions that will lead the team to achieve the maximum possible reward. Furthermore, in the work of [40] to improve the explainability of HRC tasks in the primary sector, they used SHAP values to make the recognition and classification of human actions by the robot interpretable and thus providing transparency about how robots interpret human behavior. Also, in [41] they used reward decomposition on a RL algorithm in order to understand how different factors impact the robot's actions in an automated warehouse scenario, a HRC environment where human and robots work together.

2.5 Motivation and contribution

The HRC task used in our work is a virtual adaptation, implemented in the Gazebo simulation environment, of the task originally presented in [36], where a human controls the y axis of a robot's EE and collaborates with a DRL agent that controls the x axis of the EE. The goal of the task is to move the EE from an initial position to a target position with a specific speed. Our goal was to make the agent's behavior more explainable, so that the participating human could understand how the agent makes decisions during the game and thus increase the reliability of the interaction. Since we wanted to see how the agent makes decisions based on the state of the environment in which it finds itself and how each feature of the state contributed to the action being taken, we used the SHAP values method. Since SHAP values give each feature an attributed impact on a network's prediction, we can use them as importance weights to determine the positive or negative role of the feature in the agent's decision to take or reject an action, respectively. Since we want to explain how the agent makes decisions, we calculate the SHAP values in the Actor's network. Through SHAP, we can examine whether the contribution of the features is consistent with human intuition, i.e., whether they are consistent with the rules of the experiment for satisfying the goal. Furthermore, since we have information related to both the agent and the human, SHAP values allow us to examine the degree to which each contributes during the collaboration.

Chapter 3

Methodology

This chapter describes the methodology employed to investigate whether SHAP values can produce explanations of agent behavior in a Human Robot Collaborative (HRC) setting. The methodology is organized into four interconnected stages. First, a virtual collaborative task is established in the Gazebo simulation environment, in which a human and a Deep Reinforcement Learning (DRL) agent jointly control the End-Effector (EE) of a UR3 cobot toward a target position. Second, the DRL agent is described in detail: its state space, action space, reward structure, and the Discrete Soft Actor Critic (SAC) algorithm used to derive its policy. Third, the process of training the human–agent team is described. Finally, the implementation of the *KernelSHAP explainer* is presented, through which we produce the SHAP value of each feature of the state space and, consequently, its importance weight. By applying the KernelSHAP explainer to the agent's Actor network, we can observe how each feature contributed to the acceptance or rejection of an action during the interaction and whether this influence of the feature is consistent with human intuition.

3.1 The Virtual Collaborative Task

The *collaborative task* used in this work is a virtual adaptation, implemented in the Gazebo simulation environment, of the task originally presented in [36], in which a

human collaborates with a DRL agent to move the End Effector (EE) of a Universal Robots UR3 cobot from an initial position to a target position at a controlled speed. More specifically, the EE can start from 4 possible initial positions that form a rectangular parallelogram. The human controls the y axis of the EE via a keyboard, while the DRL agent controls the x axis. The task involves running multiple games for training and testing the agent. A *game* is defined as the process during which the human and the agent interact with the robot's EE until the EE is moved from an initial position to a circle with radius $r = 0.01m$ around the target position at a speed of less than $0.05m/s$ or until 30 seconds have elapsed since the start of the game. During the execution of the task, the EE's speed could not exceed $0.2m/sec$ and its position was limited $-0.162m$ and $-0.35m$ on the x axis and $0.348m$ and $0.159m$ on the y axis. All features mentioned earlier are min–max normalized to the range $[0, 1]$. The task was implemented in Robot Operating System (ROS 1).

Based on the orientation of the task space relative to the human and the position of the EE relative to the target position, the space is divided into four quadrants: quadrant *UR (Up Right)*, where the EE is to the right and above the target position; quadrant *UL (Up Left)* where the robot is to the left and above the target position, in quadrant *LR (Low Right)* where the robot is to the right and below the target position, and in quadrant *LL (Low Left)* where the robot is to the left and below the target position. The four possible initial positions have the same names as the quartiles in which they are located. The coordinates of the target position are $(-0.252m, 0.245m)$ and their min-max normalized values are $(0.521, 0.455)$. It is worth noting that the goal state in terms of the target position is satisfied when the min-max normalized values of features x and y are within the values 0.468 to 0.574 and 0.402 to 0.508, respectively. The task space of the robotic arm is shown in detail in Figure 3.1, where the four possible initial positions are represented by four red dots, the purple dot represents the target position, and the purple circle represents the region around which the target criterion is satisfied, while the boundaries of the task space are represented by dashed black lines.

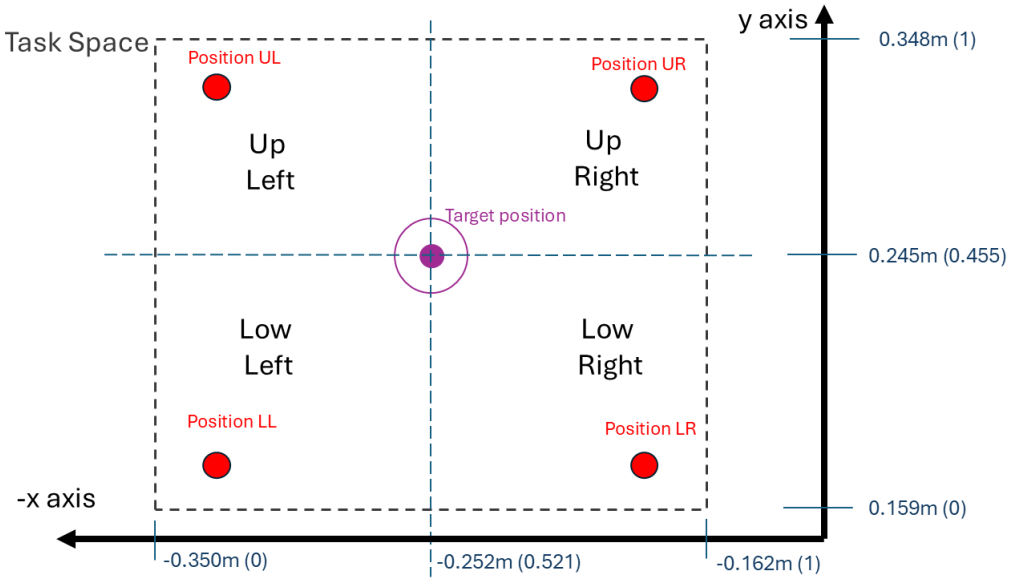


Figure 3.1: The task space of the game, including the 4 initial positions and the target position. The min-max normalized values are shown in parentheses.

One of the problems that had to be addressed during the thesis was that the Gazebo simulation was not fully functional in the *first version* of the simulation due to an error in the robot control loop. The robot control loop consisted of two different controllers, one for controlling the EE's movement during interaction and one for initializing the game, i.e., moving the EE from its current position to one of the four possible initial positions in order to start the game. In ROS1, both controllers were implemented by one controller plugin both in the physical implementation and in Gazebo, named *ur3_cartesian_velocity_controller* and *ur3_cartesian_velocity_controller_sim*, respectively. While the control strategy worked in the physical implementation of the task in the work of [36], the control related to the initialization of the game in Gazebo led the cobot arm to a singularity, where the elbow joint acquired a value of $0rad$. This resulted in the loss of one degree of freedom and the complete immobilization of the robot, as shown in Figure 3.2.

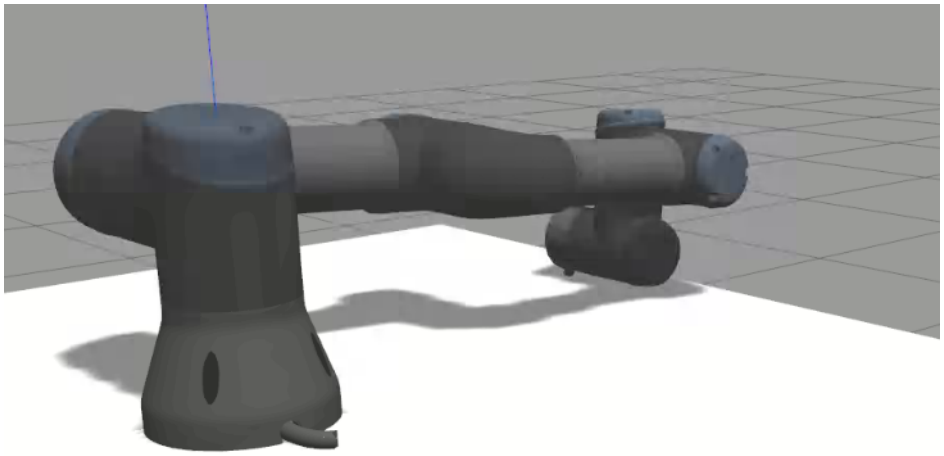


Figure 3.2: The configuration of the robot when it passed through the singularity during the initialization of the game in Gazebo *(first version)*.

For this reason, a *second version* of the simulation had to be created in which the cobot was not driven into a singularity. The problem was solved by using a different controller plugin that was available for the Gazebo simulation environment, the *joint_group_position_controller*. The *joint_group_position_controller* is a position controller implemented by the *ros_controller* package and accepts as input the angles of the joints that we want the arm to have in the desired configuration. After the necessary calibration of the physical robot, we found the angles that the joints should have when the arm is in the 4 initial positions and used them in the simulation environment as input to the *joint_group_position_controller* to perform the initialization. After initialization is complete, the control strategy of [36] during interaction is used normally through the *ur3_cartesian_velocity_controller_sim*. Switching between the two controller plugins is achieved through the switch_controller service. It is worth noting that EE control with respect to the z axis was also added to *ur3_cartesian_velocity_controller_sim*. This was because it was observed that in the simulation, when the EE velocity was equal to $0m/s$, there was a loss of EE's height, whereas it should have remained constant.

An additional error in the simulation had to be resolved, which concerned the measurement of the EE velocity components on the x and y axes, respectively. In the physical implementation of the game, the components of the EE's position and velocity were

recorded via the rostopic *ur3_cartesian_velocity_controller_sim/ee_state*. However, in the simulation environment, while the position components were recorded correctly, the velocity components were recorded incorrectly and received very small values, resulting in the velocity criterion not being correctly maintained. Therefore, the *final version* of the simulation was created, where we use the positions obtained to calculate the velocity of the EE, as the velocity is the derivative of position with respect to time:

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} \tag{3.1}$$

where u_x and u_y are the velocity components along the x and y axes, respectively, and dx and dy are the changes in the position of EE along the x and y axes, respectively, within a time interval dt.

After the resolution of the above control issues, the visual representation of the simulation was enhanced to allow the user to clearly distinguish the current position of the EE, the four possible initial positions, and the target position along with its acceptance radius. For this reason, six cylindrical markers were added. To represent the four possible initial positions, four blue cylindrical markers with a radius of $0.01m$ were used, centered on the coordinates that correspond to those positions. To represent the target position, including the radius around it, a red cylinder was used with its center at the coordinates of the target position and a radius of $0.01m$. Finally, the current position of the EE is represented by a green cylindrical marker with a radius of $0.003m$, whose center is shifted based on the coordinates of the current EE's position. The final version is shown in Figure 3.3.

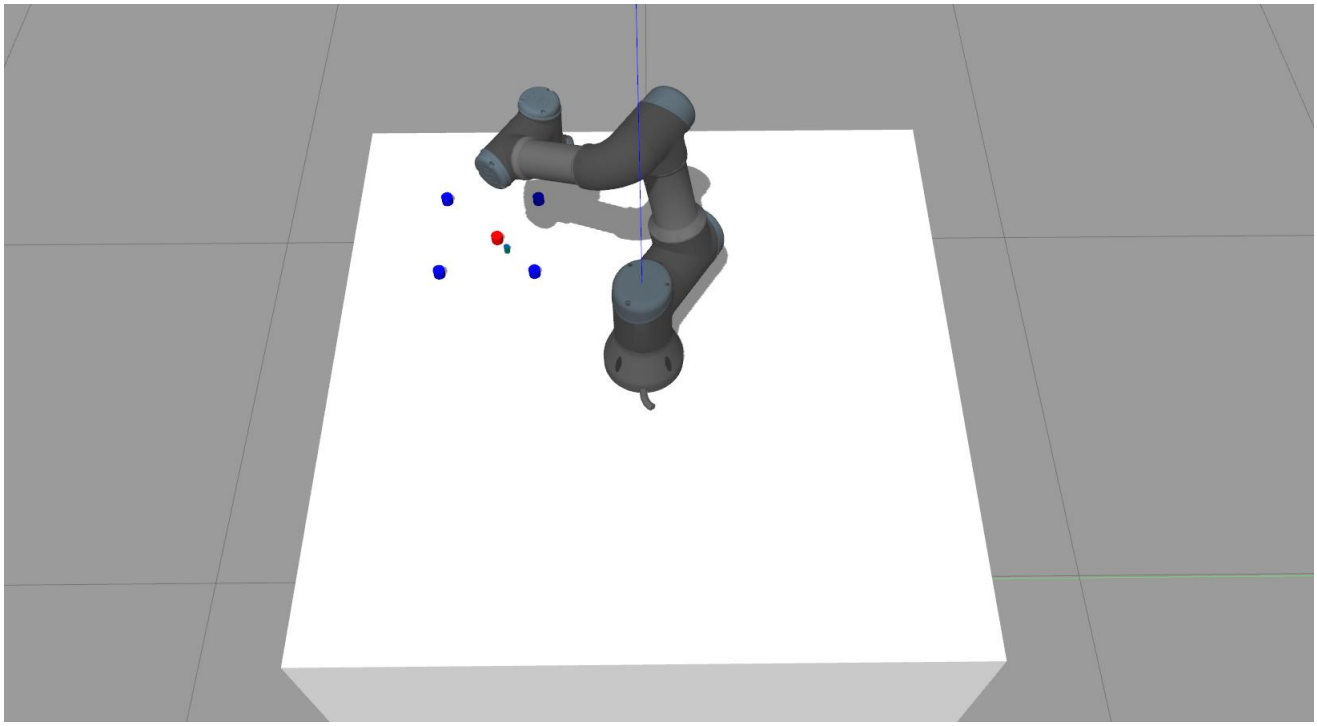


Figure 3.3: The Gazebo simulation of the collaborative task (*final version*).

3.2 The DRL agent

This chapter briefly describes how the agent controls the EE's movement along the x-axis works. The agent makes decisions using the Discrete SAC algorithm, as described in detail in Chapter 2.

As in the work of [36], during the interaction, the agent can take three possible actions at each timestep: positive ($+1$), negative (-1), and zero acceleration (0). A new action is taken every $200ms$. In addition, the agent receives a reward of 10 as soon as it reaches the target position within a certain radius around it and maintains the speed limit. Otherwise, for each timestep that it is not at the target, it receives a reward of -1 as a penalty.

As part of this thesis, it was further explored whether the agent's state space would be 4-dimensional or 2-dimensional. The state space would include the position (x_{ee}, y_{ee})

and velocity $(\dot{x}_{ee}, \dot{y}_{ee})$ of the EE or only the position, respectively. During the experiments, it was observed that if the state space consisted only of the position of the EE, there was not only low performance but also a strong sense of loss of control of the arm's movement, in contrast to the 4-dimensional state space. Therefore, the state space was maintained in its 4-dimensional form. A difference from the work of [36] is that the position and velocity of the EE undergo min-max normalization in order to be on the same scale (from 0 to $+1$) and their effect during agent training to be equal.

Table 3.1 lists in detail the hyperparameters used in the agent architecture. In the output layer, the agent uses a softmax function to generate a probability distribution that determines the selected action. Finally, the replay buffer has the ability to store up to 10^6 transitions so that it can store as much information as possible.

Hyper-parameter	**value**
Layers	2 fully connected, 1 output
Fully connected layer units	32, 32, available actions: 3
Batch size	256
Replay buffer size	1000000
Discount rate	0.99
Learning rate Actor	0.0003
Learning rate Critic	0.0003
Learning rate Alpha temperature	0.002
Optimizer	Adam
Weight initializer	Xavier initialization
Activation function	ReLU
Networks update per off-line training	12,500
Loss function	Mean square error
Entropy target	$0.36(-\log(1/\|A\|))$

Table 3.1: Discrete SAC settings

3.3 Co-Learning process and Data Collection

Once the simulation was fully functional, we moved on to the co-learning process of the Human-Agent (HA) team so that the human, who in our case is the author of this

thesis, could become an expert at the game and the team could learn to collaborate. To achieve this, the author had to perform multiple training rounds with the agent.

An experimental *run* consists of 13 *blocks*, with each block containing 8 games. In each game, the EE started randomly from one of the four possible starting positions, but each position had to be selected twice throughout the block. The first block was used as baseline testing, where the agent's networks were initialized based on Xavier initialization and the selected action was sampled from the probability distribution returned by softmax. The remaining blocks were organized into 6 experimental *batches*, each consisting of a training block (the agent's networks training process) and a testing block. The training block included the collection of data during the interaction and its storage in the buffer, where agent's selected action was sampled from the probability distribution returned by softmax. The weights of the neural networks were then updated through $12,500$ off-line gradient updates with a batch size of 256. Finally, in the testing batch, the agent selected the action with the highest probability based on softmax, and the interactions collected during the test batch were used to evaluate whether the team managed to learn to collaborate.

During the execution of each block, the following data were collected:

- Data concerning the performance in each game in the training and testing batch, such as the total reward, the total duration of the game, and the total distance traveled by the EE.
- Data collected during off-line gradient updates, such as network Q values, Q losses, Q target losses, and policy loss.
- Data related to each transition during each game, such as the human action, the agent action, the coordinates of the EE's position at the current and previous timestep, and the components of the EE's velocity at the current and previous timestep.

The evaluation of the team to determine whether it learnt to collaborate, and therefore whether the author had managed to become an expert at the game, was carried out

using objective measures were calculated from data collected during the testing blocks. For each testing block, the following objective measures were used:

- *Score*: In each game, 150 is added to the total Reward collected by the team. The score indicates how quickly the team managed to reach the goal state. The maximum possible score is 160 (this would occur if the EE started directly from the goal state at the beginning of the game and it is calculated by adding 10, the reward the agent receives when it reaches the goal state, to 150), and the minimum possible score is 0 (the team used up all the available time without the EE reaching the goal state).
- *Number of Wins*: The number of games in the block in which the team reached the goal state.
- *Normalized Travel Distance*: The total distance traveled by the EE during the game divided by the total time it took to complete it.
- *Heatmaps*: A distinct representation of the EE's area of action, showing the frequency with which the EE was found in each distinct part of the area.

The task was performed 67 times, lasting a total of 63 hours, of which the 61 (59 hours) were performed in the second version of the simulation, 1 (1 hour) was performed in the physical implementation of the game, and 5 (3 hours) were performed in the final version of the simulation. The total duration of the experiments does not include the time required to perform the off-line gradient updates. Even if the 61 times were performed in the second version of the simulation, the fact that the task was repeated several times led the author to gain valuable experience from his interactions with the agent, as he understood how the task worked and how to use the keyboard to properly control the EE's position relative to the y-axis. In addition, he learned how to guide the team in executing trajectories that led the EE to its target position.

3.4 The dataset for the KernelSHAP explainer

The main goal of this thesis was to explore explainability methods that could provide some insights about the agent's learnt policy, at the end of the co-learning process. For

this reason, the SHAP values explanation method was used, as presented in detail in Chapter 2. More specifically, the KernelSHAP method was used to approximate SHAP values. SHAP values are importance weights that indicate the extent to which each feature of the state space (EE position and velocity components) at a specific timestep discouraged or encouraged the selection of one of the three available actions. Therefore, the SHAP value returned by the KernelSHAP method for a specific timestep of the game is of dimension 4×3, and the element SHAP[i,j] for timestep t shows how much feature i contributed to the probability produced by softmax for action j. The magnitude of the SHAP value shows each feature's contribution, while its sign shows whether it encouraged (positive) or discouraged (negative) the choice of the action.

The KernelSHAP method was implemented using the KernelSHAP explainer, an object of the Python shap library. It is initialized by providing both the model that needs explanation and the background dataset (samples that represent the baseline behavior). Since our goal is to understand why the agent selected a specific action, we explained the agent's actor model. After the explainer is created, the test set (the samples that need interpretation) is fed into it. Then, the explainer uses a special weighted linear regression to compute the importance of each feature. The output consists of features' SHAP values of each sample in the test set. The Python code used to implement the explainer was based on the "Explainable Reinforcement Learning Tutorial" (as presented in Explainable Reinforcement Learning Tutorial). The implementation of the KernelSHAP explainer is available in the following GitHub repository.

After becoming an expert, the author used the final policy resulting from the last run to create a dataset consisting of 520 games. These games were used to create the KernelSHAP explainer and to interpret the agent's behavior. Since the games do not have the same duration, we had select games. A histogram was created to observe the frequency with which different durations appear in these 520 games. The histogram is shown in Figure 3.4.

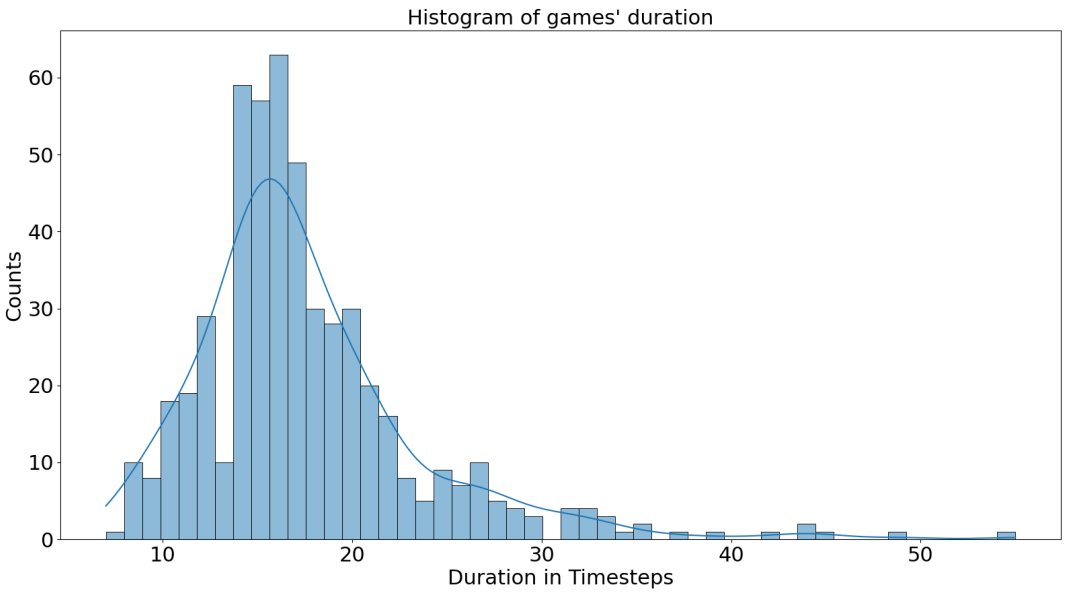


Figure 3.4: The frequency with which games of different durations appear in the 520 games collected for the development of the KenrelSHAP explainer as an explainability method.

It can be observed that most games have duration ranging from 10 to 24 timesteps. From the 520 games, we selected 420 so that the game durations ranged from 10 to 24 timesteps, to filter out the rest less optimal examples, and games with the same initial position were evenly distributed. Of the 420 games, 344 were used as a background set to create the explainer, and 76 were used to generate explanations regarding the agent's behavior. In both the 344 and 76 games, we ensured that all games starting from the same initial position were still evenly distributed. In order to compare how the agent's behavior changes over time, even though the game durations are not the same, we decided to observe how the SHAP values of the games change at intervals of 20% of their duration. Since games lasting between 10 and 24 timesteps have an integer percentage of 20% only if they last 10, 15, or 20 timesteps, only games with these durations were included in the 76. More details are presented in Chapter 4.

Finally, to confirm the generalization of the SHAP values regarding the agent's behavior, we performed K-fold cross-validation, with $K = 5$. From the 420 games, we cre-

ated 5 different test sets, each containing 76 games, and 5 corresponding background sets, each containing 344 games with uniformly distributed initial positions. From each background set, we created 5 different explainers, and for each corresponding test set, we calculated their SHAP values and compared their similarity.

3.5 SHAP values as a criterion for explainability

Since we know the SHAP values of the features for each timestep and the contribution of the features to each action, we can observe how they change over time in each game and see how they relate to the way the feature values change. Thus, we can observe how feature values affect the agent's behavior. Furthermore, we can observe when features encourage the selection of an action and draw conclusions about when one action is preferred over another. Given that SHAP values should help us produce explanations that are consistent with human intuition, we would like to make the following observations:

- When the components x and y of the EE's position differ greatly from the components of the target position, the contribution of the coordinates to the selection of the appropriate action should be high in order to bring the EE close to the target position. If the action leads to a departure from the target position, then the components discourage the selection of this action.
- When the EE approaches the target position, the contribution of speed to the selection of the action should be high in order to comply with the speed criterion. If the action leads to a deviation from the speed criterion, then the components discourage the selection of this action.
- When the EE must change direction and move to the right, the action of positive acceleration ($+1$) should be encouraged.
- When the EE must continue moving to the right, the action of positive acceleration ($+1$) or constant speed (0) is encouraged. If it needs to slow down the negative acceleration (-1) will be encouraged.

- When the EE should change direction and move to the left, the action of negative acceleration (-1) should be encouraged.
- When the EE must continue moving to the left, the action of negative acceleration (-1) or constant velocity (0) is encouraged. If it needs to slow down the negative acceleration ($+1$) will be encouraged.

Chapter 4

Results

4.1 Human-Expert behaviour in the collaborative task

An demonstration of expertise in the task is shown by the scores, wins and the normalized distances for the 5 executions of the task performed by the team. It is observed that even though the initialized agent in the baseline block differed in each execution, the team's performance was strong, achieving high scores and low normalized distances, as well as 8 out of 8 wins across all block (with the exception of the baseline), as shown in Figures 4.1, 4.3 and 4.2, respectively.

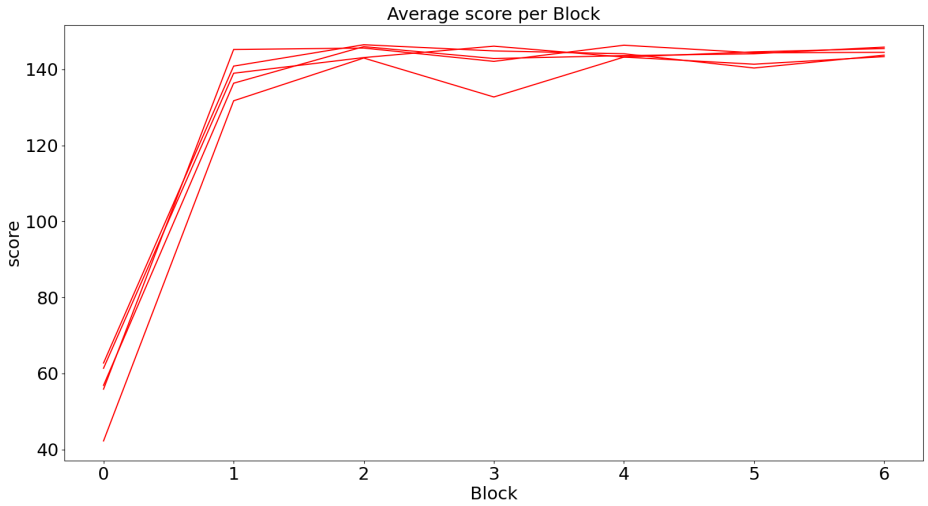


Figure 4.1: Scores per block for 5 executions of the task.

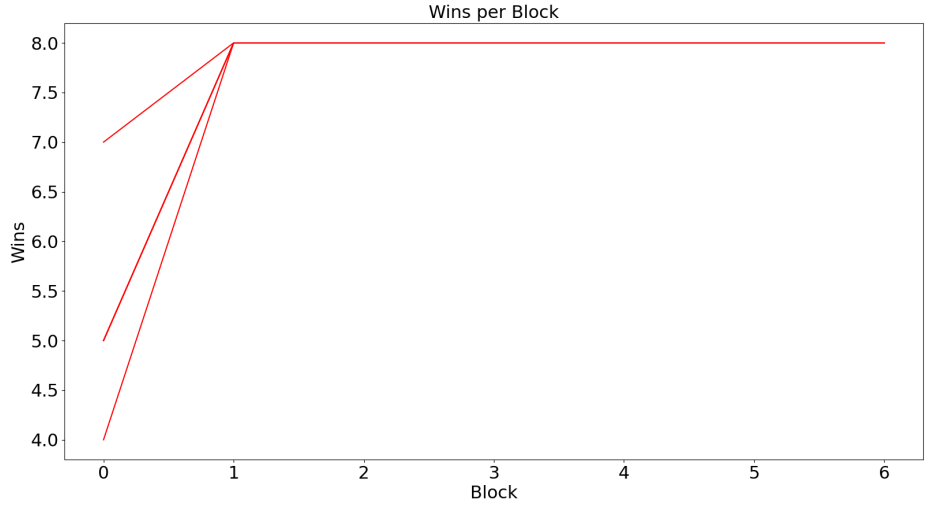


Figure 4.2: The wins per block for 5 executions of the task.

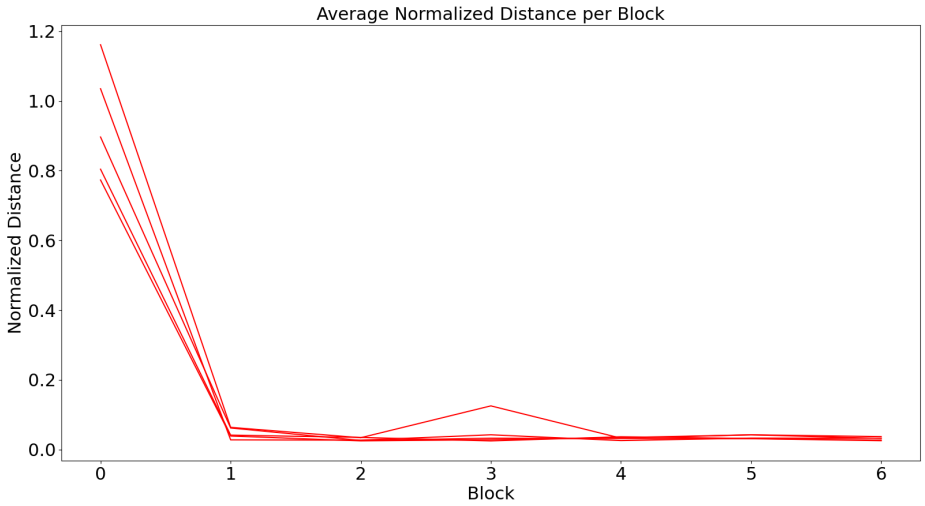
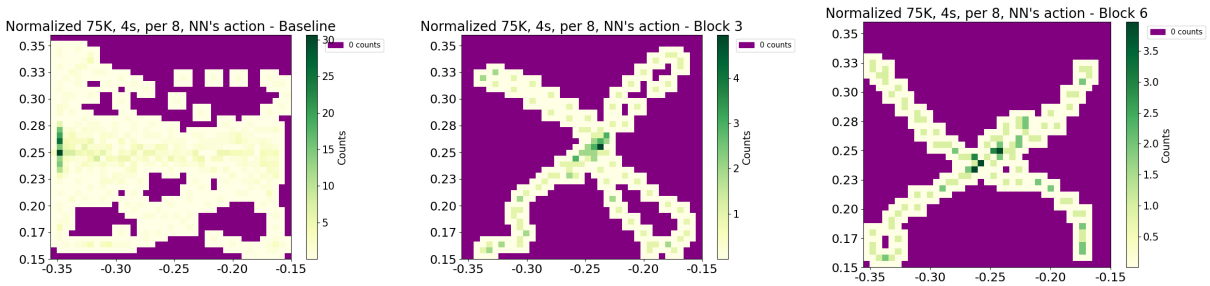


Figure 4.3: The normalized distances per block for 5 executions of the task.

Finally, the heatmaps show that in the baseline, the human-agent team tends to move the EE to the correct y coordinate, and in the last block, the trajectories tend to form an X shape, which is particularly noticeable in the fifth execution. The X-shape is an important indicator of whether the author has reached the expert level, as it demonstrates a very direct and efficient way of moving from the initial position to the target position. Because the EE follows the shortest route, the game is completed in the shortest possible time. Furthermore, the straight path demonstrates the team's ability to properly control the EE's position. In order to illustrate the evolution of the EE's trajectory from the baseline (when the agent is Xavier initialized-to the completion of the final block) when the team has achieved the best possible performance, the heatmaps for the fifth execution of the task for the baseline block, the third block and the sixth block are shown in Figures 4.4a, 4.4b and 4.4c respectively.



(a) Baseline block. (b) Third block. (c) Sixth block.

Figure 4.4: The heatmaps of the fifth execution in the baseline, third and sixth block.

Therefore, performance plateaued across the final blocks, suggesting convergence to a stable behavioral pattern, which shows that the author became an "expert". Thus, the agent's actor network that emerged at the end of the fifth execution of the task was used in the implementation of the KernelSHAP explainer.

4.2 SHAP as a criterion for explaining the agent's behavior during the games

Since SHAP values are importance weights, the explanations produced relate to how much each of the four features (the x coordinate of the EE, the y coordinate of the EE, the u_x and u_y velocities of the EE relative to the x and y axes) contributes to choosing or avoiding a decision. The explanations generated should be understandable and they could be expected to match the expected team behaviour, i.e., the way in which SHAP values contribute to the selection of a decision must be consistent with the satisfaction of the goal state.

We initially observed how the SHAP values of the features for the selected action change over time. This allows to draw conclusions about how the contribution of features changes at each stage of the game and how SHAP values change in relation to the actual values of the features. To enable comparison across games of different durations, only games lasting exactly 10, 15, or 20 timesteps were included in the test set, as these

lengths allow clean sampling at exactly 20% intervals. At each interval (20%, 40%, 60%, 80%, and 100% of game completion) the unnormalized feature values and their corresponding SHAP values were recorded. From the unnormalized feature values, we calculated the distance of the EE from the target position along the x-axis, the distance of the EE from the target position along the y-axis, and the total distance of the EE from the target position. In addition, we calculated the magnitude of the EE's speed and the magnitude of each velocity component. Also, we made comparisons between games that have the same starting position, because they have more similar trajectories and this would lead to a more accurate evaluation.

4.2.1 Overall Observations

By analyzing the games with the same initial position individually, we were able to draw the following general conclusions about all the games in the test set:

- The most important features for action selection are those that the agent can control (u_x, x), unlike the features (u_y, y) controlled by the human.
- As the EE approaches the target position, the importance of u_x increases. The importance of position x increases slightly or remains constant in most cases
- The y-position has a significant influence on the selection of actions only when the game is 20% complete for the initial positions UL and UR, since the EE is particularly far from the target position. This is an important finding because it shows that the agent is sensitive to the human axis when the EE is far from the target position. This is more evident in the UR and UL positions compared to the LR and LL positions. After this, the importance of y and u_y is constant throughout game and of low importance as mentioned above.

Our findings are supported by the graphs showing how the features' SHAP values change with respect to each game completion percentage and how the EE distance changes with respect to the target position across all games. The graphs are presented in Figures 4.5 and 4.6, respectively.

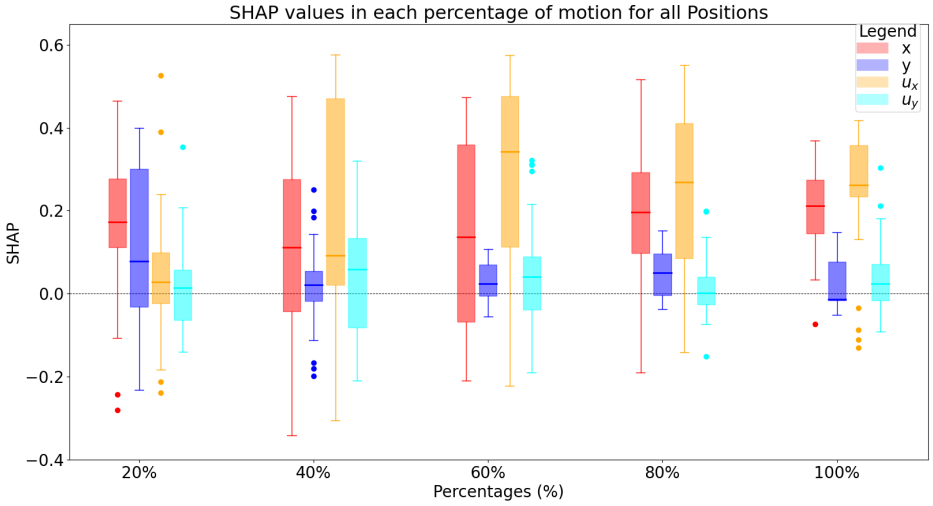


Figure 4.5: Change in the SHAP value of each feature per percentage of game completion.

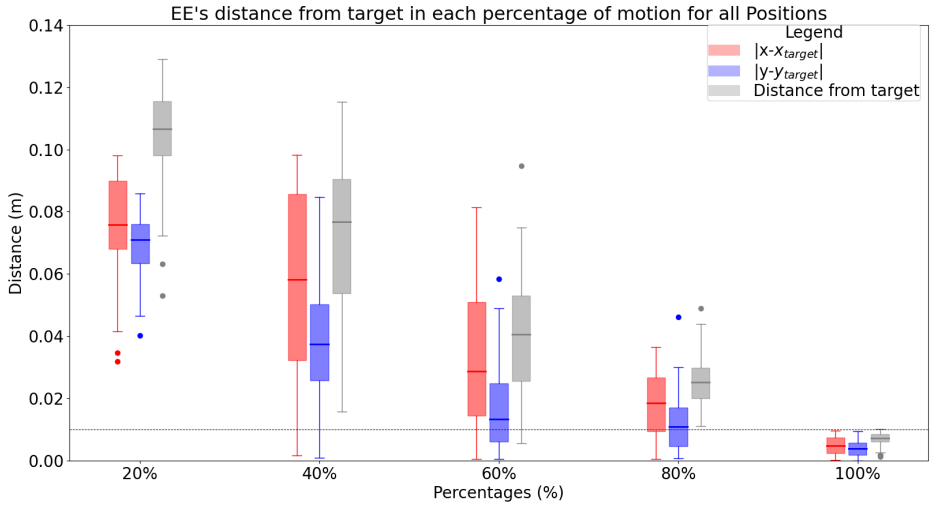


Figure 4.6: Change in the calculated distance of the EE from the target position, including the distance of each component, per percentage of game completion.

4.2.2 SHAP values' evolution for initial position UR

We present how the SHAP values evolve starting from initial position UR (Up Right). Figures 4.7, 4.8, and 4.9 show, respectively, how the SHAP values of each feature, the EE's distance from the target, and the EE's speed evolve across the five completion checkpoints.

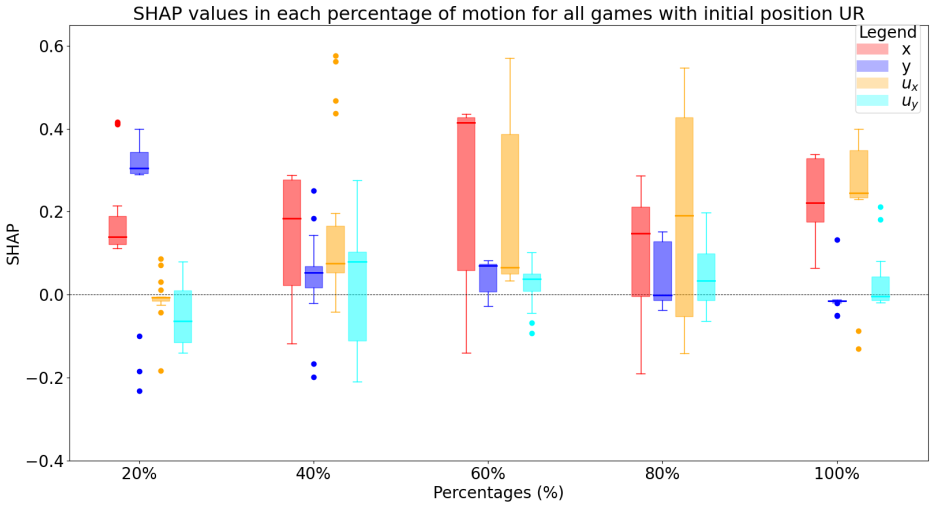


Figure 4.7: Change in the SHAP value of each feature per percentage of game completion when the initial position is *position UR*.

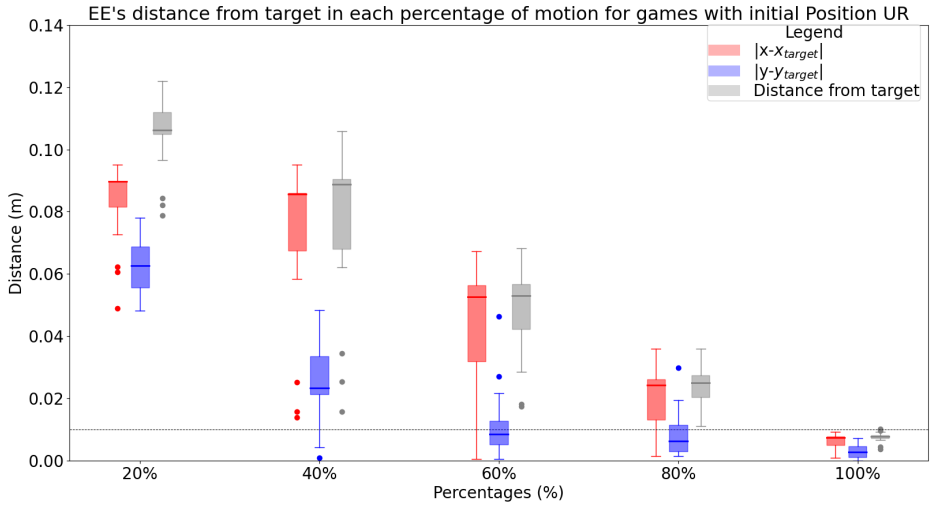


Figure 4.8: Change in the calculated distance of the EE from the target position, including the distance of each component, per percentage of game completion when the initial position is *position UR*.

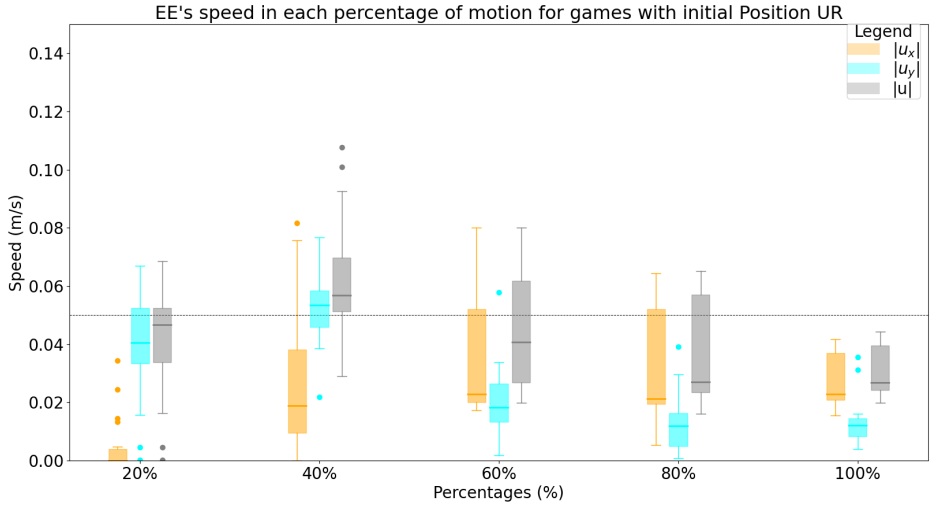


Figure 4.9: Change in the calculated speed of the EE, including the magnitudes of each component, per percentage of game completion when the initial position is *position UR*.

From the graphs above, we can draw the following conclusions for games that start at position UR:

- 20% of the game: The EE is furthest from the target position, the most important feature is the y-position of the EE (controlled by the human), followed by the x-position of the EE (controlled by the agent).
- 40% of the game: The x-position is still important and the importance of the velocity component u_x increases, and it now drives the selection of actions. In contrast, the importance of the y-position decreases. This coincides with the fact that the EE is located near the y-coordinate of the target position, as shown in Figure 4.8. The importance of the y-position after 40% completion of the game is relatively stable and is one of the least important features in action selection.
- From 60% to 80% completion of the game: Even if the EE is approaching the x-coordinate of the target position, the importance of the x-position remains significant. Together with velocity u_x, they guide the selection of actions. It is also observed that as the EE approaches the target position, the variance in the importance of the features decreases.
- 100% of the game: The most important feature in action selection is u_x, followed by the EE's x-position.
- The importance of the velocity component u_y remains relatively constant from 20% to 100% completion of the movement and is one of the least important features in action selection.

4.2.3 SHAP values' evolution for initial position UL

We examined how SHAP values evolve as a function of the percentage of movement completed for games where their initial position is UL (Up Left). The graphs showing how SHAP values, distances, and velocity magnitudes evolution are presented in Figures 4.10, 4.11, and 4.12, respectively.

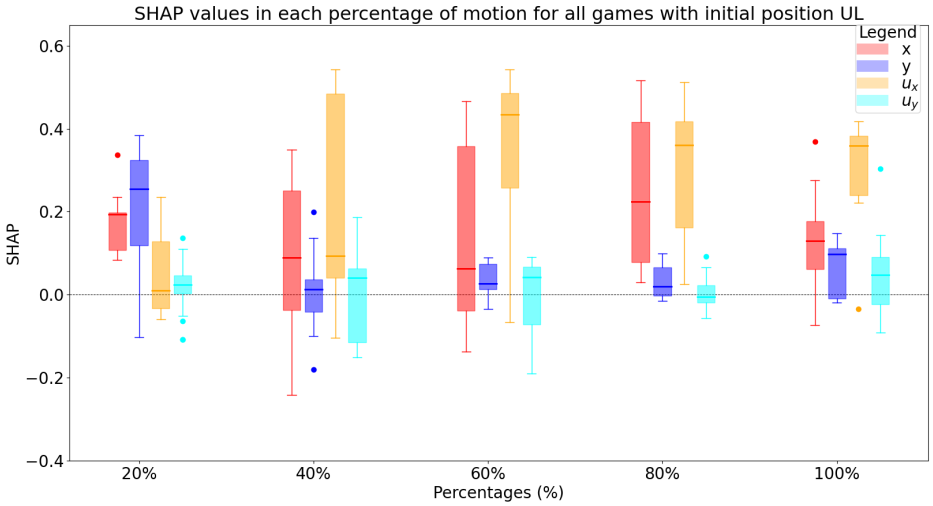


Figure 4.10: Change in the SHAP value of each feature per percentage of game completion when the initial position is *position UL*.

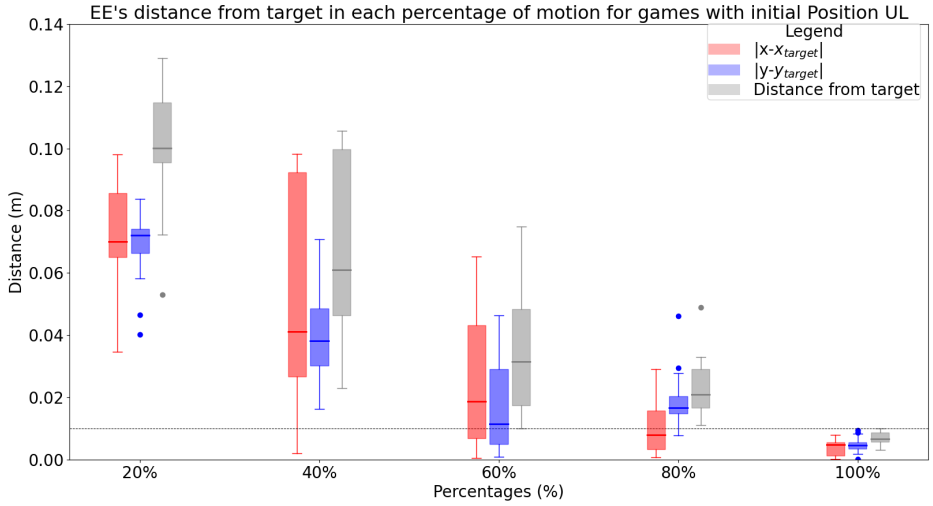


Figure 4.11: Change in the calculated distance of the EE from the target position, including the distance of each component, per percentage of game completion when the initial position is *position UL*.

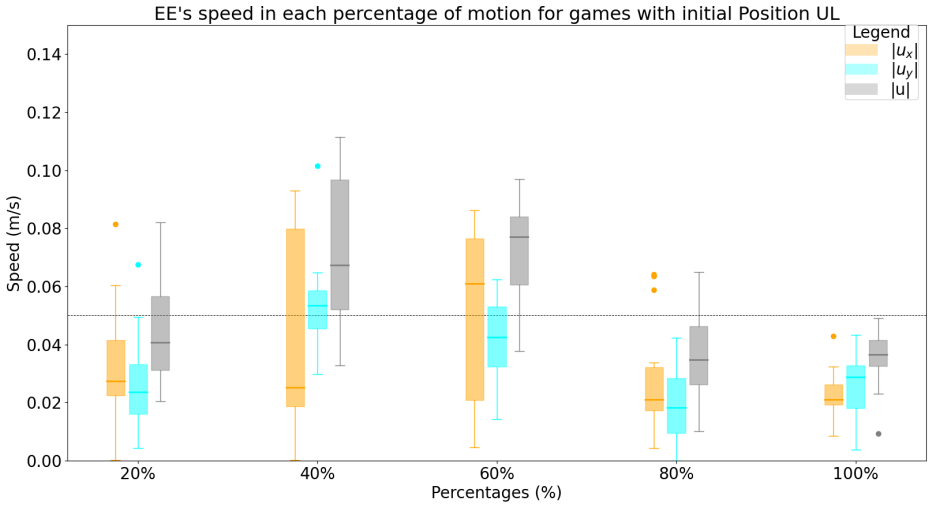


Figure 4.12: Change in the calculated speed of the EE, including the magnitudes of each component, per percentage of game completion when the initial position is *position UL*.

From the graphs above, we can draw the following conclusions for games that start at position UL:

- 20% of the game: The most important feature is the EE's y-position, followed by the EE's x-position.
- 40% of the game: The EE's x-position is still important and it guides the selection of action. In contrast, the importance of the y-position decreases. This coincides with the fact that the EE is located near the y-coordinate of the target position, as shown in Figure 4.11. The importance of the y position after 40% completion of the game is relatively stable and is one of the least important features in action selection.
- From 60% to 80% completion of the game: The EE is located near the target position, as shown in Figure 4.11, and the importance of feature u_x increases. At 60%, the importance of x-position decreases, and at 80% its importance increases again, even if the EE further approaches the x-coordinate of the target position. Thus, x-position and u_x, guide the selection of actions.

- 100% completion of the game: The most important feature in action selection is the feature u_x, followed by the x-position of the EE.
- The importance of the velocity component u_y remains relatively constant from 20% until the end of the game and is one of the least important features in action selection.

4.2.4 SHAP values' evolution for initial position LR

The following shows how SHAP values evolve as a function of the percentage of movement completed for games where their initial position is LR (Low Right). The graphs illustrating the evolution of SHAP values, distances, and velocity metrics are presented in Figures 4.13, 4.14, and 4.15.

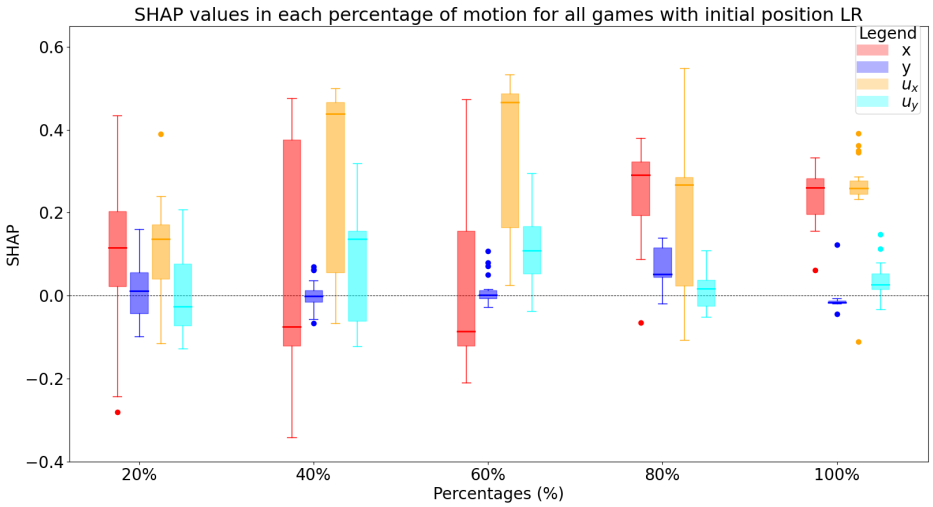


Figure 4.13: Change in the SHAP value of each feature per percentage of game completion when the initial position is *position LR*.

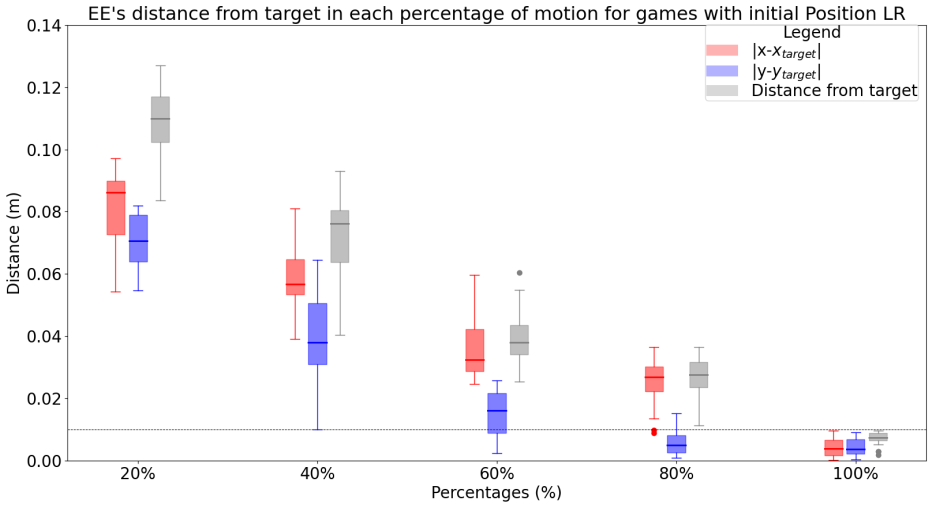


Figure 4.14: Change in the calculated distance of the EE from the target position, including the distance of each component, per percentage of game completion when the initial position is *position LR*.

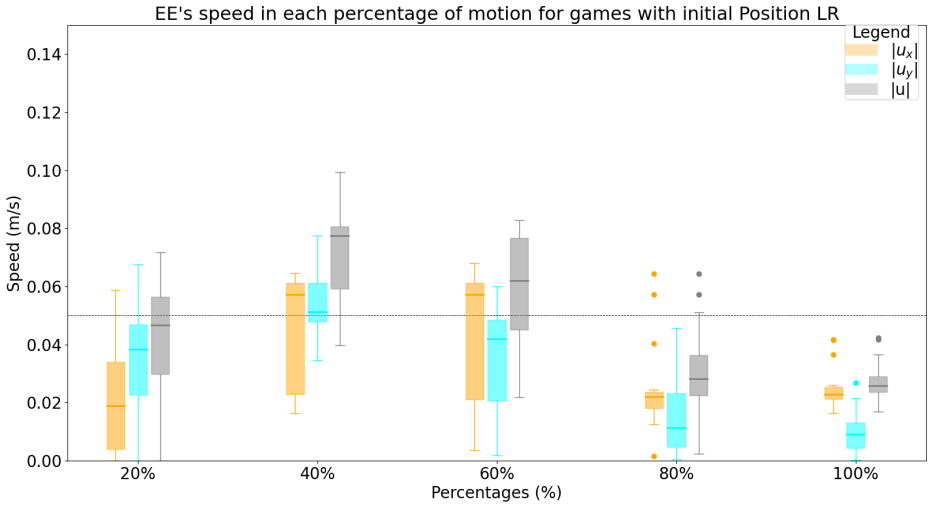


Figure 4.15: Change in the calculated speed of the EE, including the magnitudes of each component, per percentage of game completion when the initial position is position *LR*.

From the graphs above, we can draw the following conclusions regarding games that

start at position LR:

- From 20% to 40% of the game: The selection of actions is determined by the EE's x-position and u_x. The average importance of the two features increases during this interval but exhibits relatively high variance.
- 60% completion of the game: The average importance of the u_x continues to increase, while that of x-position decreases. The selection of actions is now determined primarily by u_x. Because of a slight increase in the importance of EE's u_y, u_y becomes the second most important feature in action selection, while x-position is among the least important features. Throughout the rest of the game, the importance of the u_y remained relatively constant and was one of the least important features
- 80% completion of the game: The importance of EE's x-position increases (even if the EE further approaches the x-coordinate of the target position), while the importance of feature u_x decreases. The most important features in action selection are the x-position, followed by u_x.
- 100% completion of the game: The importance of the u_x feature increases, while that of x-position remains constant. The selection of the action is determined equally by the features u_x and x-position.
- Throughout the entire movement, the y-position remained relatively stable in terms of importance and was one of the least important features in the selection of actions.

4.2.5 SHAP values' evolution for initial position LL

Finally, we examined games starting from position LL (Low Left). The graphs showing how the SHAP values, distances, and velocity metrics evolve are presented in Figures 4.16, 4.17, and 4.18.

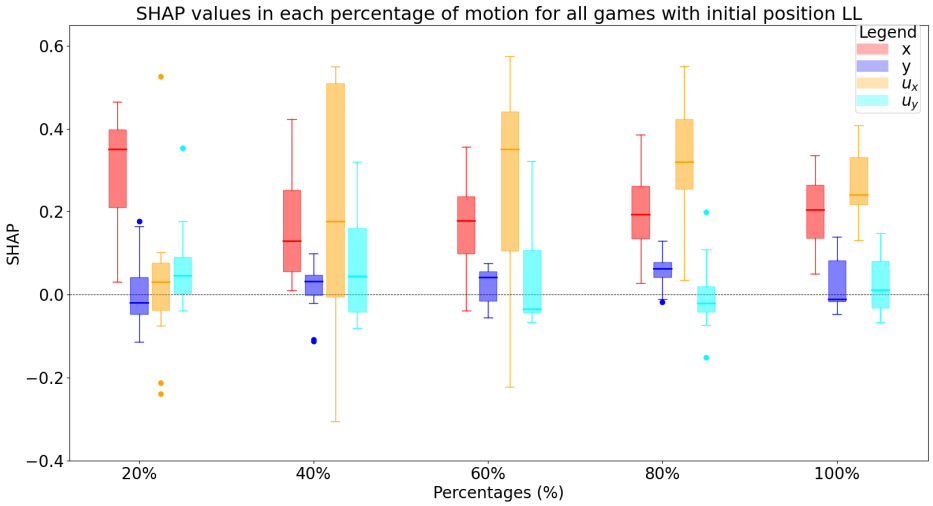


Figure 4.16: Change in the SHAP value of each feature per percentage of game completion when the initial position is *position LL*.

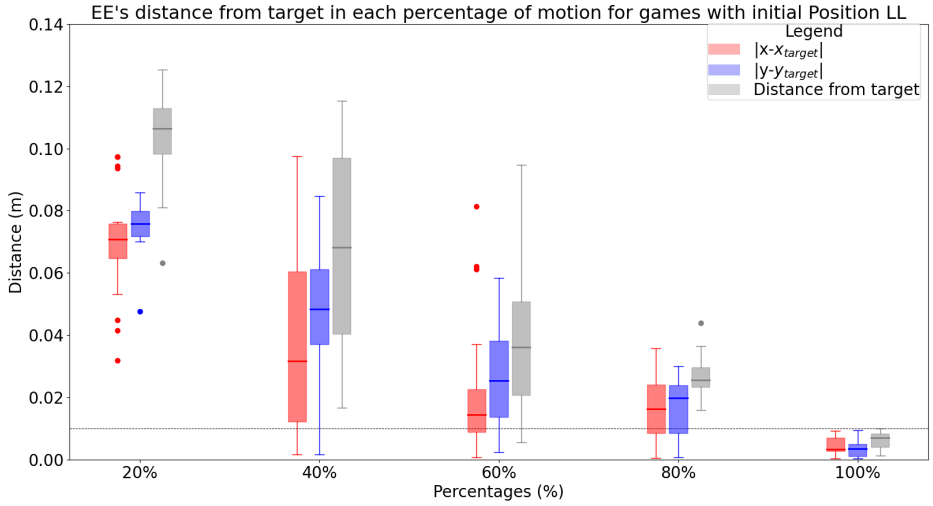


Figure 4.17: Change in the calculated distance of the EE from the target position, including the distance of each component, per percentage of game completion when the initial position is *position LL*.

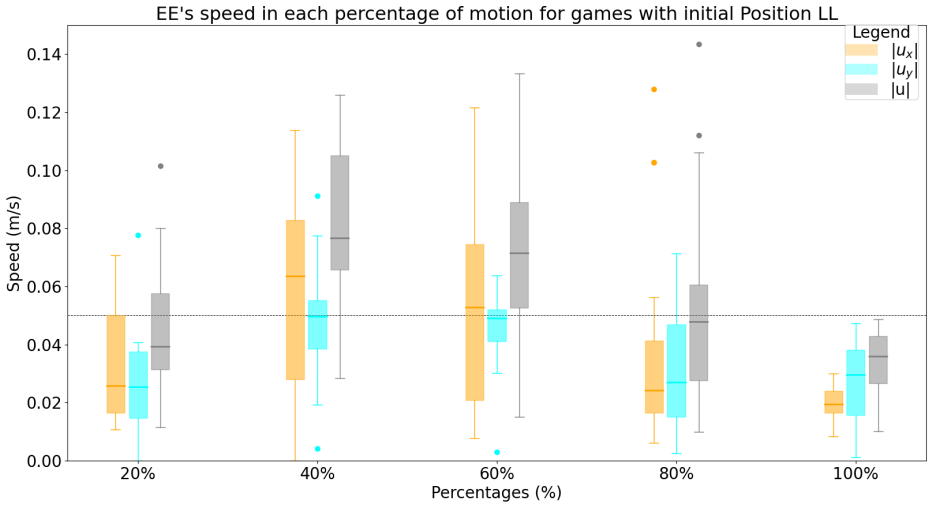


Figure 4.18: Change in the calculated speed of the EE from, including the magnitudes of each component, per percentage of game completion when the initial position is *position LL*.

From the graphs above, we can draw the following conclusions about games that start at position LL:

- 20% completion of the game: EE's x-position is the most important feature in the selection of actions. From 40% onward, it is the second most important feature.
- From 40% to 100% completion of the game: The importance of u_x increases and it becomes the most important feature for action selection, while the EE approaches the target position, as shown in Figure 4.17.
- Throughout the entire game, the importance of EE's x-position remains relatively constant.
- Throughout the entire game, the least important features for action selection were y and u_y. Furthermore, their importance remained relatively constant.

4.3 How feature values influence the selection or rejection of each action based on SHAP values

In this section we observe how each feature affects the selection or rejection of each of the agent's three available actions accross the workspace.

4.3.1 The SHAP values of EE's position

First, for each poassible action, we created a scatter plot in which the horizontal and vertical axes represent the x and y positions of the EE, respectively. Each point on the scatter plot corresponds to the EE's position at each timestep during the execution of the 76 games that make up the test set. The points are colored according to the SHAP value of feature x. The scatter plots for the 3 available actions, 0, $+1$, and -1 for the x-position are shown in Figure 4.19.

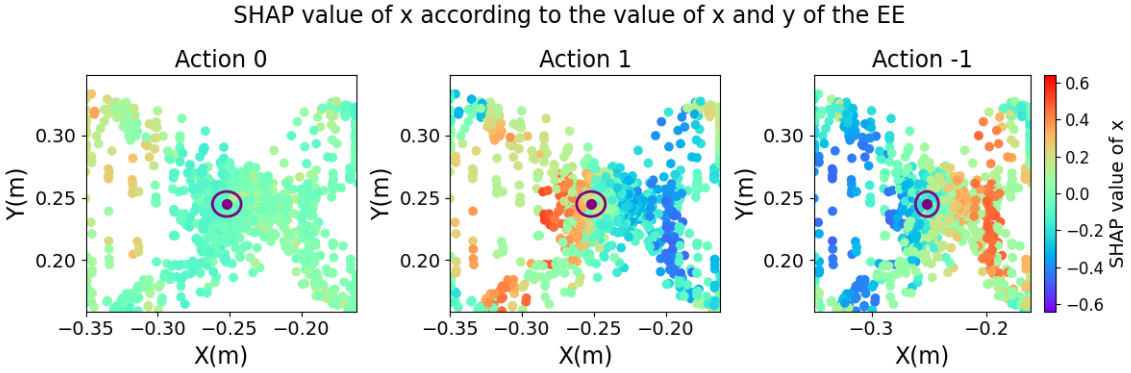


Figure 4.19: The scatter plot graphs of the 3 available actions showing the position of the EE during the execution of the games that make up the test set, with the SHAP value of feature x as the colormap. We denote the target position and the region within which the goal state is satisfied with a purple point and a purple circle, respectively

Note that overall the team seems to exhibit suboptimal performance. While it performs better in games with starting positions UR and LR, the X-shaped pattern formed by the

trajectories is more distorted in games with starting positions UL and LL.

We draw the following conclusions regarding the contribution of feature x to the selection of actions:

- Feature x favors the selection of action 0 when the EE is to the left of the target position, where $x < -0.3m$ and $y > 0.2m$. However, overall, feature x does not contribute to the selection of action 0.
- The feature x favors the selection of action $+1$ when the EE is to the left of the target position. The closer the EE gets to the target position while remaining to its left, the higher the positive SHAP value of x becomes. This means that the support for action $+1$ from feature x increases. However, when the EE is within the satisfaction region of the target position, the SHAP value relate to x decreases. When the EE is to the right of the target position, the feature x does not favor the selection of action $+1$, as its SHAP value is negative. The further to the right the EE is, the more it discourages it. The opposite is true for the selection of action-1: Action -1 is favored when action $+1$ is not favored and action -1 is not favored when action $+1$ is favored. Thus, action -1 and action $+1$ are complementary.
- Consequently, feature x behaves intuitively, as it directs the EE toward the target and penalizes overshoot.

Corresponding scatter plots were created to observe how the SHAP value of feature y evolves, and they are presented in Figure 4.20.

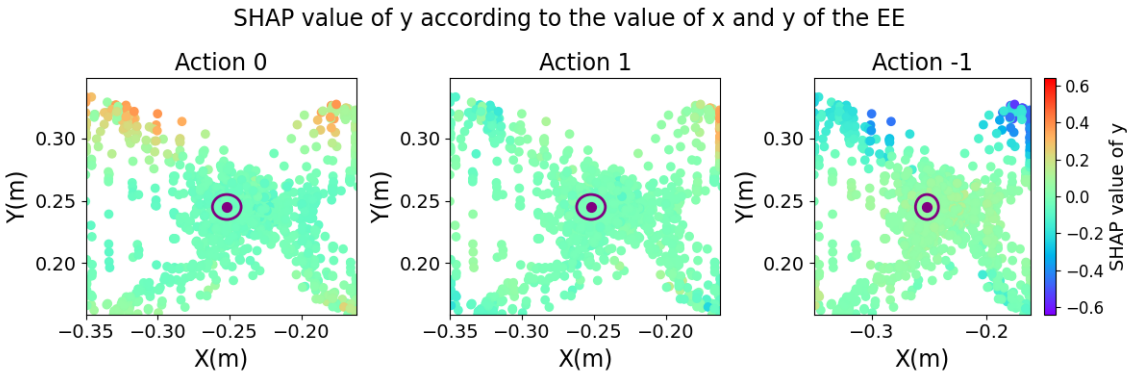


Figure 4.20: Scatter plots of the 3 available actions showing the position of the EE during the execution of the games comprising the test set, with the SHAP value of feature y as the colormap.

We observe that the discouragement or encouragement of an action by position y is not as strong as that by position x. This is in line with the findings presented in 4.2.1.

4.3.2 The SHAP values of EE's velocity

In addition, we observed how the velocity component affects decision-making. The significance of u_x depends on two factors: the position of the EE and the magnitude of the velocity. Actions should be selected so that they guide the EE toward the target position but control the velocity's magnitude to ensure compliance with the speed criterion based on the goal state.

For each action we created a scatter plot where the vertical axis represents the u_x velocity component, the horizontal axis is the position x of the EE, and the points represent all sampled game states in the test set. The color of the points corresponds to the SHAP value of u_x in each sample. Additionally, we use two vertical and two horizontal dashed lines, which indicate when the EE reaches the positions $-0.252m$ and $-0.250m$ along the x-axis and the velocities u_x of $-0.05m/s$ and $0.05m/s$, respectively. This gives us a better picture of what is happening in the region where the EE tends to or satisfies the goal state with respect to the features x and u_x. In order to observe any dependence that the EE's position y might have, we created two sets of scatter plots: one where the

EE is above the target position (Figure 4.21) and one where the EE is below the target position (Figure 4.22).

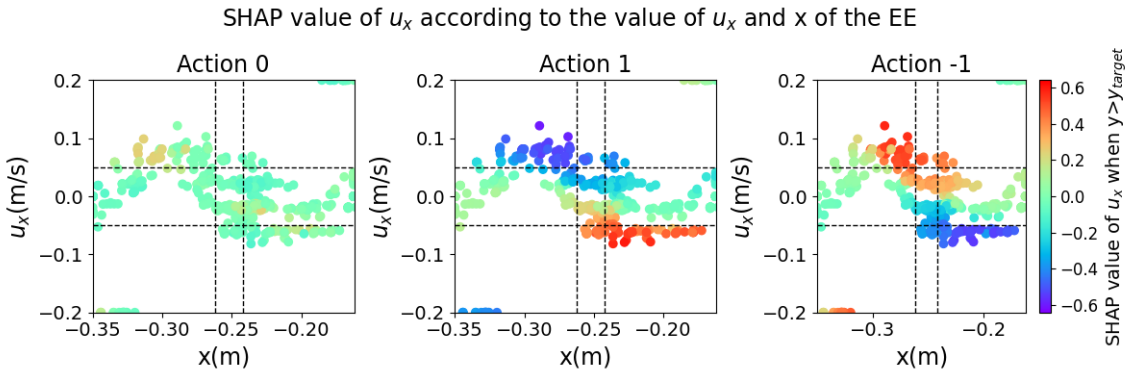


Figure 4.21: The scatter plot graphs showing the EE's u_x values at positions x for the 3 available actions across the test set, in the case where the EE is *above the target position*. The colormap is the importance of the SHAP values.

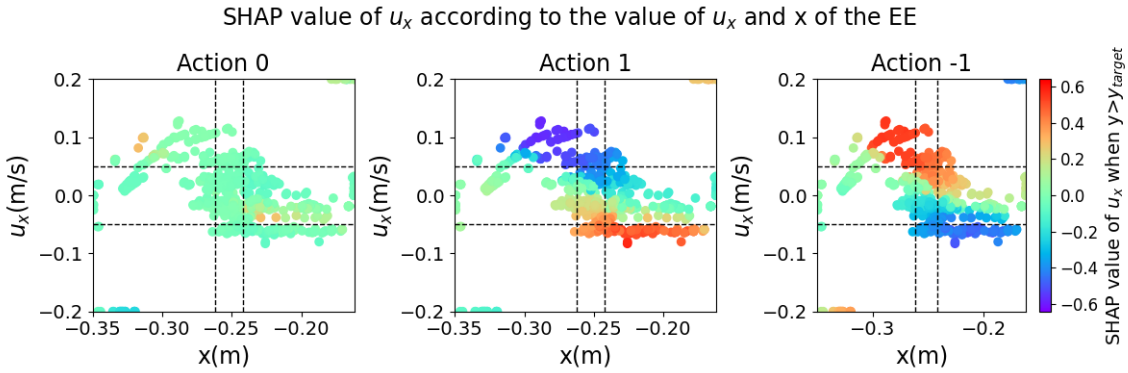


Figure 4.22: The scatter plot graphs showing the EE's u_x values at positions x for the 3 available actions across the test set, in the case where the EE is *below the target position*. The colormap is the importance of the SHAP values.

It can be observed that the pattern is similar in both cases. Thus, we can draw the following conclusions:

- It can be observed that u_x does not seem to play an important role to the choice of action 0.
- EE to the left of the target position: Action $+1$ is not favored when the value of $u_x > 0.05m/s$. Otherwise, u_x does not seem to play an important role or tends to favor action $+1$.
- EE to the right of the target position: Action $+1$ is favored when $u_x < -0.05m/s$. Otherwise, u_x does not seem to play an important role or tends to not favor action $+1$.
- EE is in the target's position x-coordinate $\pm 0.01m$: When $u_x < 0m/s$, action 1 is favored. The more negative the value of u_x is, the more it is favored. Otherwise, if $u_x > 0m/s$, the selection of action $+1$ is not favored.
- The action -1 is favored when the action $+1$ is not favored and action -1 is not favored when action $+1$ is favored. Consequently, actions $+1$ and -1 are complementary.

Similarly, we created scatter plots showing the relationship between the y-position and the u_y velocity component of the EE for each possible action, where the horizontal axis represents the y-position, the vertical axis is the value of u_y, and the color of the points corresponds to the SHAP value of u_y for each sample. As before, in order to observe any bias that the position x of the EE might introduce, we created two sets of scatter plots: one where the EE is to the right (Figure 4.23) and one where the EE is to the left of the target position (Figure 4.24).

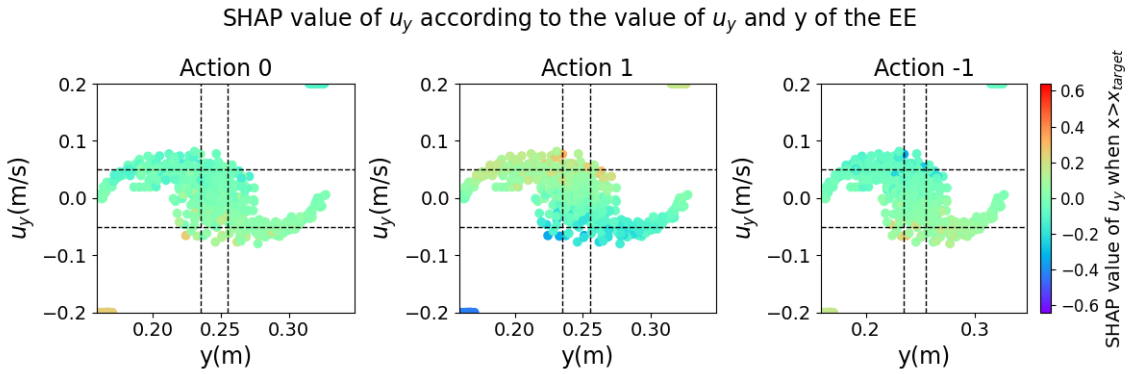


Figure 4.23: The scatter plot graphs showing the EE's u_y values at positions y for the 3 available actions across the test set, in the case where the EE is *right the target position*. The colormap is the importance of the SHAP values.

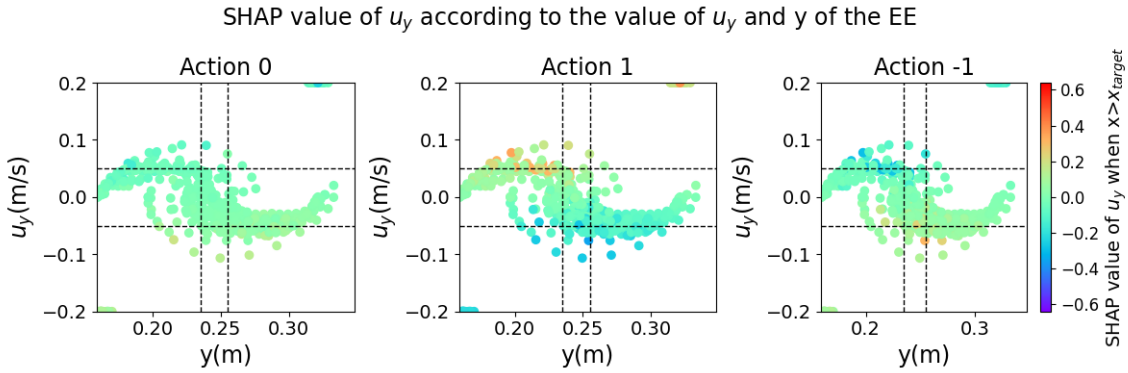


Figure 4.24: The scatter plot graphs showing the EE's u_y values at positions y for the 3 available actions across the test set, in the case where the EE is *left the target position*. The colormap is the importance of the SHAP values.

It is observed that while u_y exhibits the same pattern as u_x, it is not as important in the selection of actions, which is consistent with the findings in 4.2.1.

4.4 K-fold cross-validation

In order to confirm the generalization of our findings, we would like to see if we can obtain similar SHAP values from explainers trained on different background sets when

using different test sets. For this reason, we performed K-fold cross-validation: from the 420 games, we created 5 different test sets, each containing 76 games, and the 5 corresponding complementary background sets, each containing 344 games. From each different background set, we created a separate KernelSHAP explainer and generated SHAP values from the corresponding test set. We then compared the similarity between the different SHAP values generated by each test set.

To achieve this for each fold, we calculated the mean SHAP value of each feature for each action when the feature values fell within specific intervals. The intervals were chosen such that the mean SHAP value of each feature is representative of that interval. The scatter plots presented earlier contributed to the selection of the intervals. We then compared the calculated mean SHAP values across the folds for each action and for each feature within the respective intervals. The results from K-fold cross-validation is reported in A in Tables A.1 throught A.16.

We observed that when the EE is to the right of the target position, the average SHAP value of feature x is positive across all folds for action -1 and negative for action $+1$, whereas this pattern is reversed when the EE is to the left of the target position. This indicates that regardless of the background set on which the KernelSHAP explainer was trained, action -1 is favored when the EE is to the right of the target position; otherwise, action $+1$ is favored.

Furthermore, in all folds, the mean SHAP value of u_x when $0.2m/s > u_x > 0.05m/s$ is positive for action -1 and negative for action $+1$. The same pattern is observed when the EE is within the target position and the velocity u_x is positive. The opposite pattern is observed when $-0.2m/s < u_x < -0.05m/s$ or when the EE is within the target position and the velocity u_x is negative. This indicates that deceleration of the EE is favored when the speed criterion is not met and when the EE is within the target position. Furthermore, the magnitude of the mean SHAP value of u_x is greater than the mean value of x, suggesting that u_x plays a more significant role in action selection in this case. In addition, when the EE is to the left of the target position and the velocity criterion is satisfied, the average SHAP value of u_x across all folds is positive and indicates that action $+1$ is favored, whereas when it is to the right and the velocity criterion is satisfied, action -1 is favored. However, the magnitude of the SHAP value of u_x is smaller in this

case, indicating its lesser influence on action selection.

Finally, the average SHAP values of the features y and u_y have a smaller magnitude across all folds, indicating that they have a smaller influence on decision-making compared to the features x and u_x. However, when the EE is to the right of the target position, the feature y favors action -1 in all folds, while when it is to the left of the target position, action $+1$ is favored. Furthermore, when u_y is positive, its influence is greater compared to the other cases, and action $+1$ is favored. This indicates that the agent has learned to anticipate or respond to the human's movement.

It is worth noting that small standard deviations are observed across all SHAP values of the features (especially for the u_x feature) within each interval. This indicates that the SHAP values are stable and do not depend on the background set used for training. Therefore, the generalization of our results is confirmed.

Chapter 5

Conclusions

In this thesis, we sought to develop a method through which we could explain how a Deep Reinforcement Learning (DRL) agent makes decisions in the context of an Human-Robot Collaboration (HRC) task. To this end, we developed a virtual version of the task proposed by [36] in the Gazebo simulation environment, where a human and an agent collaborate to move the End-Effector (EE) of a robotic arm to a target position at a specific speed. Once the simulation became fully operational, a human participant (the author of this thesis) performed the task several times in order to become an expert and lead to the creation of an agent that learned a policy demonstrating that the team had learned to collaborate successfully. Specifically, the task was repeated 67 times, a process that lasted 63 hours, during which the participant gained valuable experience in understanding the task. Of the 67 executions, only the last 5 were used to evaluate the team's behavior. The assessment of whether the author became an expert at the task was carried out using objective metrics, which included the number of the team's wins, the team's score, the normalized travelled distance of the EE, and heatmaps showing the frequency with which the EE was located in a specific position in the task space. The team's expert-like behavior was confirmed primarily by the appearance of the X-shape in the heatmaps, which indicated that the team was able to guide the EE to the goal state by following the shortest path in the shortest possible time.

Next, we wanted to explain the agent's policy, specifically how the features of the state space influenced the selection or rejection of available actions. For this reason, we im-

plemented the KernelSHAP explainer method, which uses SHAP values to quantify the importance of state space features in action selection. Furthermore, the SHAP values indicate whether the feature's value had a positive or negative effect on the selection of each available action. To implement the KernelSHAP explainer, multiple games were run and collected to create a dataset. The human-agent team used to create the dataset consisted of the agent that exhibited the best consistent behavior during the repeated task executions and the author of this thesis. Because of the team's structure, the SHAP values provide explanations only about the agent's behavior that emerged at the end of the co-learning process. After collecting 420 games, they were divided into a background set (344 games) and a test set (76 games). The background set was used to implement the KernelSHAP explainer, and the test set was used to generate the SHAP values and, consequently, to explain the agent's policy.

Therefore, we were able to study how the SHAP values change during the course of the game and observe how the features influence the acceptance or rejection of an action depending on the state in which the EE is. Thus, we derived the following general rules for the agent's behavior:

- The most important features for action selection are those that the agent can control (u_x, x). On the contrary, the least important features for action selection are those that the human can control (u_y, y).
- As the agent approaches the target position, the importance of u_x increases. The importance of position x increases slightly or remains constant in most cases.
- The feature x encourages taking action 1 when the agent is to the left of the target position and action -1 when it is to the right of the target position. Thus, x encourage actions that guide EE to goal.
- The feature u_x encourages taking actions that will lead to a reduction in the velocity magnitude when the velocity criterion is not met or when the EE is near or within the target position. Consequently, u_x encourage actions that guide EE to goal.
- Action $+1$ and action -1 are complementary.

- The y-position has a significant influence on the selection of actions only when the movement is 20% complete for the initial positions UL and UR, since the agent is particularly far from the target position. This is an important finding because it shows that the agent is sensitive to the human axis when the EE is far from the target position. This is more evident in the UR and UL positions compared to the LR and LL positions. For the rest of the movement, the importance of y and uy is relatively low (as mentioned above) and constant across the game.

The generalization of our results was confirmed through K-fold cross-validation with K=5. Across five different folds we small standard deviations are observed across all SHAP values of the features (especially for the u_x feature) within each interval, which indicates that the SHAP values are stable and do not depend on the background set used for training. In addition, we observed that the influence of the features in the action selection was similar in all folds and it aligned with our general rules for the agent's behavior.

Consequently, we demonstrated that KernelSHAP can produce interpretable, human-aligned explanations of a Discrete-SAC policy in a shared control HRC task.

As next steps, we intend to further investigate the use of SHAP values as a means of explaining the decision-making policy of an autonomous agent within more complex Human-Robot Collaboration (HRC) environments involving multiple human-agent teams operating simultaneously. Extending the framework to multi-team settings would allow us to evaluate whether SHAP-based explanations remain interpretable and informative when coordination dynamics, interdependencies, and shared task objectives become more sophisticated. Also, we would like to compare the teams' results between the virtual and physical implementation of the task.

In addition, we plan to conduct a user study to investigate whether explanations generated by the KernelSHAP explainer enable non-expert participants to better understand the task and, consequently, improve their performance. We also aim to examine whether these explanations enhance users' trust in the agent's behavior, thereby fostering safer and more effective human-agent interaction. The study could be performed by presenting to the participants snapshots of other teams' behavior along with the cor-

responding explanations and they would be asked to assess whether these explanations improve the interpretability of the agent's behavior.

Finally, we would like to explore the potential of leveraging SHAP values to automatically generate explanations in natural, human-understandable language. Rather than presenting explanations solely through numerical feature attributions or visualizations, this direction would focus on translating the most influential factors behind the agent's decisions into concise textual descriptions that are intuitive for human collaborators. This could significantly improve communication between humans and autonomous agents and contribute to the development of more transparent, trustworthy, and user-friendly HRC systems.

Appendices

Appendix A

The results from K-fold cross-validation

Fold	Action 0	Action 1	Action -1
0	0.00443737	−0.09619684	0.10063421
1	−0.00157039	−0.09925749	0.10082787
2	0.00024347	−0.09343978	0.09319630
3	−0.00436281	−0.09928478	0.10364757
4	0.00008919	−0.10191363	0.10182446
$\mu \pm \sigma$	-0.00023 ± 0.00285	-0.09802 ± 0.00301	0.10003 ± 0.00359

Table A.1: Average SHAP value of x when $x \geq x_{target}$

Fold	Action 0	Action 1	Action -1
0	0.00525200	0.15346155	−0.15871352
1	−0.00698510	0.16387182	−0.15688672
2	−0.01190318	0.17324079	−0.16133762
3	0.00826698	0.15034545	−0.15861242
4	0.00934189	0.14628979	−0.15563166
$\mu \pm \sigma$	0.00079 ± 0.00880	0.15744 ± 0.01011	-0.15824 ± 0.00205

Table A.2: Average SHAP value of x when $x < x_{target}$

Fold	Action 0	Action 1	Action -1
0	0.06131800	−0.42414933	0.36283131
1	0.04223035	−0.42698008	0.38474978
2	0.04426665	−0.41774943	0.37348270
3	0.05066417	−0.41999831	0.36933408
4	0.04812090	−0.41826388	0.37014295
$\mu \pm \sigma$	0.04932 ± 0.00706	-0.42143 ± 0.00357	0.37211 ± 0.00723

Table A.3: Average SHAP value of u_x when $0.2m/s > u_x > 0.05m/s$

Fold	Action 0	Action 1	Action -1
0	−0.00722946	0.45172575	−0.44449630
1	−0.00419314	0.45491083	−0.45071771
2	−0.00836110	0.45235990	−0.44399872
3	−0.00387059	0.45029012	−0.44641956
4	−0.01149629	0.45189100	−0.44039471
$\mu \pm \sigma$	-0.00703 ± 0.00282	0.45224 ± 0.00150	-0.44521 ± 0.00337

Table A.4: Average SHAP value of u_x when $-0.2m/s < u_x < -0.05m/s$

Fold	Action 0	Action 1	Action -1
0	−0.04262721	0.02215766	0.02046957
1	−0.03901088	0.01410144	0.02490944
2	−0.03070650	0.01918994	0.01151664
3	−0.03919683	0.02692647	0.01227041
4	−0.04406216	0.03172759	0.01233463
$\mu \pm \sigma$	-0.03912 ± 0.00469	0.02282 ± 0.00644	0.01630 ± 0.00541

Table A.5: Average SHAP value of u_x when $-0.05m/s \leq u_x \leq 0.05m/s$ and $x < x_{target} - 0.01$

Fold	Action 0	Action 1	Action -1
0	0.00936039	−0.04110003	0.03173966
1	0.00656764	−0.05079867	0.04423099
2	0.00429702	−0.05633307	0.05203608
3	0.00532446	−0.05598238	0.05065793
4	0.00768809	−0.04286931	0.03518124
$\mu \pm \sigma$	0.00665 ± 0.00183	-0.04942 ± 0.00643	0.04277 ± 0.00839

Table A.6: Average SHAP value of u_x when $-0.05m/s \leq u_x \leq 0.05m/s$ and $x > x_{target} + 0.01$

Fold	Action 0	Action 1	Action -1
0	−0.03396808	−0.31126524	0.34523333
1	−0.03248296	−0.26265962	0.29514259
2	−0.03046461	−0.25546521	0.28592980
3	−0.02641039	−0.29890627	0.32531664
4	−0.03430602	−0.28743134	0.32173730
$\mu \pm \sigma$	-0.03153 ± 0.00297	-0.28315 ± 0.02323	0.31467 ± 0.02257

Table A.7: Average SHAP value of u_x when $u_x > 0m/s$ and $x_{target} - 0.01 \leq x \leq x_{target} + 0.01$

Fold	Action 0	Action 1	Action -1
0	0.00842786	0.23938835	−0.24781622
1	0.01065915	0.25145412	−0.26211330
2	0.01179083	0.24849918	−0.26028997
3	0.00768990	0.23950055	−0.24719040
4	0.01448677	0.23649955	−0.25098634
$\mu \pm \sigma$	0.01061 ± 0.00239	0.24307 ± 0.00598	-0.25368 ± 0.00641

Table A.8: Average SHAP value of u_x when $u_x < 0m/s$ and $x_{target} - 0.01 \leq x \leq x_{target} + 0.01$

Fold	Action 0	Action 1	Action -1
0	-0.01554027	0.01461948	0.00092076
1	-0.00956698	0.01433336	-0.00476638
2	-0.01210045	0.00905389	0.00304659
3	-0.01124304	0.01954465	-0.00830166
4	-0.01365538	0.00638603	0.00726938
$\mu \pm \sigma$	-0.01242 ± 0.00208	0.01279 ± 0.00445	-0.00037 ± 0.00558

Table A.9: Average SHAP value of y when $x > x_{target}$

Fold	Action 0	Action 1	Action -1
0	0.01112299	-0.01279380	0.00167076
1	0.02256563	-0.02155781	-0.00100780
2	0.02055090	-0.01763069	-0.00292015
3	0.01454332	-0.02161957	0.00707622
4	0.01470285	-0.02821529	0.01351242
$\mu \pm \sigma$	0.01670 ± 0.00474	-0.02036 ± 0.00506	0.00367 ± 0.00635

Table A.10: Average SHAP value of y when $x < x_{target}$

Fold	Action 0	Action 1	Action -1
0	-0.05926924	0.14381422	-0.08454499
1	-0.05051164	0.15056348	-0.10005187
2	-0.05810757	0.16127704	-0.10316955
3	-0.05650154	0.14274274	-0.08624121
4	-0.06166496	0.15103414	-0.08936919
$\mu \pm \sigma$	-0.05721 ± 0.00389	0.14989 ± 0.00715	-0.09268 ± 0.00787

Table A.11: Average SHAP value of u_y when $0.2m/s > u_y > 0.05m/s$

Fold	Action 0	Action 1	Action -1
0	0.06703795	-0.17427254	0.10723459
1	0.06083638	-0.15098654	0.09015020
2	0.05959960	-0.16157539	0.10197584
3	0.06668479	-0.16320719	0.09652240
4	0.06301762	-0.16580408	0.10278649
$\mu \pm \sigma$	0.06344 ± 0.00319	-0.16317 ± 0.00743	0.09973 ± 0.00591

Table A.12: Average SHAP value of u_y when $-0.2m/s < u_y < -0.05m/s$

Fold	Action 0	Action 1	Action -1
0	-0.02262292	0.05273220	-0.03010929
1	-0.02584189	0.05986288	-0.03402099
2	-0.02592911	0.05312526	-0.02719615
3	-0.03016760	0.06053133	-0.03036368
4	-0.02463720	0.04863852	-0.02400133
$\mu \pm \sigma$	-0.02584 ± 0.00250	0.05498 ± 0.00438	-0.02914 ± 0.00327

Table A.13: Average SHAP value of u_y when $-0.05m/s \leq u_y \leq 0.05m/s$ and $y < y_{target} - 0.01$

Fold	Action 0	Action 1	Action -1
0	0.02369374	-0.05551493	0.03182116
1	0.02111602	-0.04967662	0.02856057
2	0.02390680	-0.05676384	0.03285702
3	0.01949199	-0.05464896	0.03515698
4	0.01457740	-0.04232134	0.02774390
$\mu \pm \sigma$	0.02056 ± 0.00355	-0.05179 ± 0.00534	0.03123 ± 0.00266

Table A.14: Average SHAP value of u_y when $-0.05m/s \leq u_y \leq 0.05m/s$ and $y > y_{target} + 0.01$

Fold	Action 0	Action 1	Action -1
0	-0.01701565	0.05341806	-0.03640247
1	-0.01647550	0.04703455	-0.03055902
2	-0.01892045	0.05756800	-0.03864752
3	-0.01676426	0.04982496	-0.03306065
4	-0.01940511	0.06002582	-0.04062072
$\mu \pm \sigma$	-0.01772 ± 0.00115	0.05357 ± 0.00478	-0.03586 ± 0.00371

Table A.15: Average SHAP value of u_y when $u_y > 0m/s$ and $y_{target} - 0.01 \leq y \leq y_{target} + 0.01$

Fold	Action 0	Action 1	Action -1
0	0.01065000	−0.04103518	0.03038512
1	0.01261118	−0.04518994	0.03257877
2	0.01459798	−0.04412419	0.02952624
3	0.00925551	−0.04212940	0.03287383
4	0.01106209	−0.03973962	0.02867751
$\mu \pm \sigma$	0.01164 ± 0.00187	-0.04244 ± 0.00190	0.03081 ± 0.00163

Table A.16: Average SHAP value of u_y when $u_y < 0m/s$ and $y_{target} - 0.01 \leq y \leq y_{target} + 0.01$

Bibliography

[1] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[2] Pedro Sequeira and Melinda Gervasio. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. *Artificial Intelligence*, 288:103367, 2020.

[3] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[5] Dimitri Bertsekas. *Reinforcement learning and optimal control*, volume 1. Athena Scientific, 2019.

[6] Dimitri Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021.

[7] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[8] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[11] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[12] Fatai Sado, Chu Kiong Loo, Wei Shiung Liew, Matthias Kerzel, and Stefan Wermter. Explainable goal-driven agents and robots-a comprehensive review. *ACM Computing Surveys*, 55(10):1–41, 2023.

[13] Jixue Liu and James Bailey. *AI 2019: Advances in Artificial Intelligence: 32nd Australasian Joint Conference, Adelaide, SA, Australia, December 2–5, 2019, Proceedings*, volume 11919. Springer Nature, 2019.

[14] Francisco Cruz, Richard Dazeley, Peter Vamplew, and Ithan Moreira. Explainable robotic systems: Understanding goal-driven actions in a reinforcement learning scenario. *Neural Computing and Applications*, 35(25):18113–18130, 2023.

[15] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on explainable artificial intelligence*, 2019.

[16] Alessandro Iucci, Alberto Hata, Ahmad Terra, Rafia Inam, and Iolanda Leite. Explainable reinforcement learning for human-robot collaboration. In *2021 20th International Conference on Advanced Robotics (ICAR)*, pages 927–934. IEEE, 2021.

[17] Aaquib Tabrez, Shivendra Agrawal, and Bradley Hayes. Explanation-based reward coaching to improve human performance via reinforcement learning. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 249–257. IEEE, 2019.

[18] Lloyd S Shapley et al. A value for n-person games. 1953.

[19] S Remman and A Lekkas. Using shapley additive explanations to explain a deep reinforcement learning agent controlling a turtlebot3 for autonomous navigation. In *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics*, volume 1, pages 334–340, 2024.

[20] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMlR, 2017.

[21] Sindre Benjamin Remman, Inga Strümke, and Anastasios M Lekkas. Causal versus marginal shapley values for robotic lever manipulation controlled using deep reinforcement learning. In *2022 American Control Conference (ACC)*, pages 2683–2690. IEEE, 2022.

[22] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[23] Ke Zhang, Jun Zhang, Pei-Dong Xu, Tianlu Gao, and David Wenzhong Gao. Explainable ai in deep reinforcement learning models for power system emergency control. *IEEE Transactions on Computational Social Systems*, 9(2):419–427, 2021.

[24] Francesco Semeraro, Alexander Griffiths, and Angelo Cangelosi. Human–robot collaboration and machine learning: A systematic review of recent research. *Robotics and Computer-Integrated Manufacturing*, 79:102432, 2023.

[25] Thibaut Munzer, Marc Toussaint, and Manuel Lopes. Efficient behavior learning in human–robot collaboration. *Autonomous Robots*, 42(5):1103–1115, 2018.

[26] Guilherme J Maeda, Gerhard Neumann, Marco Ewerton, Rudolf Lioutikov, Oliver Kroemer, and Jan Peters. Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks. *Autonomous Robots*, 41(3):593–612, 2017.

[27] Loris Roveda, Shaghayegh Haghshenas, Marco Caimmi, Nicola Pedrocchi, and Lorenzo Molinari Tosatti. Assisting operators in heavy industrial tasks: On the

design of an optimized cooperative impedance fuzzy-controller with embedded safety rules. *Frontiers in Robotics and AI*, 6:75, 2019.

[28] Zhen Deng, Jinpeng Mi, Dong Han, Rui Huang, Xiaofeng Xiong, and Jianwei Zhang. Hierarchical robot learning for physical collaboration between humans and robots. In *2017 IEEE international conference on robotics and biomimetics (robio)*, pages 750–755. IEEE, 2017.

[29] Jenay M Beer, Arthur D Fisk, and Wendy A Rogers. Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of human-robot interaction*, 3(2):74, 2014.

[30] Erlantz Loizaga, Leire Bastida, Sara Sillaurren, Ana Moya, and Nerea Toledo. Modelling and measuring trust in human–robot collaboration. *Applied Sciences*, 14(5):1919, 2024.

[31] Gale M Lucas, Burcin Becerik-Gerber, and Shawn C Roll. Calibrating workers' trust in intelligent automated systems. *Patterns*, 5(9), 2024.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[33] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[34] Timothy Paul Lillicrap, Jonathan James Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daniel Pieter Wierstra. Continuous control with deep reinforcement learning, September 15 2020. US Patent 10,776,692.

[35] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[36] Athanasios C Tsitos and Maria Dagioglou. Enhancing team performance with transfer-learning during real-world human-robot collaboration. *arXiv preprint arXiv:2211.13070*, 2022.

[37] Ali Shafti, Jonas Tjomsland, William Dudley, and A Aldo Faisal. Real-world human-robot collaborative reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11161–11166. IEEE, 2020.

[38] Fotios Lygerakis, Maria Dagioglou, and Vangelis Karkaletsis. Accelerating human-agent collaborative reinforcement learning. In *Proceedings of the 14th PErvasive Technologies Related to Assistive Environments Conference*, pages 90–92, 2021.

[39] Dimitrios Koutrintzes. Knowledge transfer in human-artificial intelligence collaboration. Master's thesis, Πανεπιστήμιο Πειραιώς, 2023.

[40] Lefteris Benos, Dimitrios Tsaopoulos, Aristotelis C Tagarakis, Dimitrios Kateris, Patrizia Busato, and Dionysis Bochtis. Explainable ai-enhanced human activity recognition for human–robot collaboration in agriculture. *Applied Sciences*, 15(2):650, 2025.

[41] Alessandro Iucci. *Explainable Reinforcement Learning for Risk Mitigation in a human-robot collaboration scenario*. PhD thesis, Politecnico di Torino, 2021.