Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

# BOXRL: Safe Reinforcement Learning for Robotic Manipulators

**Yangtao Chen, Carter Facey-Smith, Helle Friis and Ansgar Schäfftlein**

Practical Course *Reinforcement Learning for Robotic Manipulators* SS 2023

**Advisor:** M.Sc. Jakob Thumm

**Supervisor:** Prof. Dr.-Ing. Matthias Althoff

**Submission:** 10. August 2023

# BOXRL: Safe Reinforcement Learning for Robotic Manipulators

Yangtao Chen, Carter Facey-Smith, Helle Friis and Ansgar Schäfftlein
Technische Universität München
Email: ge52feb@tum.de, carter.facey-smith@tum.de, ge62deq@mytum.de, ansgar.schaefftlein@tum.de

*Abstract*—**Over the last decade, deep reinforcement learning has been used with significant success to learn from trial and error. Examples include playing video games, object manipulation and the fine-tuning of large language models. In robotics, there is an increasing interest in using deep reinforcement learning for tasks involving interactions with humans. However, the black-box nature of deep reinforcement learning generates a need for safety precautions to prevent humans from being harmed by the robot's unpredictable behavior.**

**In this paper, we apply the SaRA safety shield approach to ensure DIN-certified safety in a scenario where the robot has to evade striking combinations executed by a human. Being more demanding than regular human-robot interactions, this novel task provides a challenging testbed. First, we train agents in the simulated Mujoco environment and then deploy them afterward on a real Schunk robot. Our results indicate that the current safety shield approach can be very restrictive for the evasion case at hand.**

## I. Introduction (C. Facey-Smith)

Advancements in robotics and artificial intelligence have led to an increased presence of robots in our everyday environments, from industrial applications to domestic settings. As the frequency of human-robot interactions (HRI) rises [1], ensuring their safety has emerged as a crucial research concern. This concern is amplified when we consider the potential for robots to cause unintentional harm due to factors such as unpredictable environments, or a lack of comprehensive understanding of human norms and behaviors [2].

Human-robot interactions involve a complex blend of physical, cognitive, and social aspects. Physically, robots need to respect human personal space and avoid collisions. Cognitively, robots should understand and predict human intentions to facilitate cooperative tasks. Socially, they should adhere to human norms and behaviors to foster comfortable and positive interactions.

The focus of this research is to investigate safe HRI within the context of physical safety. Specifically, we aim to develop a reward function for reinforcement learning (RL) based robots, enabling them to navigate around humans without causing harm. However, the design of reward functions that encourage safe behaviors presents a significant challenge due to the complex and high-dimensional task space, coupled with the need to predict spontaneous human behavior.

By exploring and improving upon various factors within the reward function, including the incentivization of goal completion, collision avoidance, and respect for human personal space, this study aims to provide insights into how RL can be used to improve safety in HRI. It is our hope that this investigation will contribute to the groundwork of safer, more reliable, and more user-friendly robots in the future.

## II. Literature review

In the following section, we introduce the concepts underlying our project. Firstly, we cover an approach to making HRI safe (section II-A). Secondly, we provide a short introduction to reinforcement learning (sections II-B to II-E), which was used extensively during the course of our project. Lastly, we discuss work related to the gamification of human-robot interactions (section II-F).

### *A. Safety shield (C. Facey-Smith, H. Friis)*

In the literature, the concept of a 'safety shield' is emphasized as a critical component for ensuring human safety when working in close proximity to robots, especially in the context of reinforcement learning within real-world human environments [3], [4]. The primary goal of the safety shield is to prevent harm to humans without unduly restricting the actions of either the robot or the human.

In one study [3], the safety shield is described as operating at high speed, allowing the robot to function at a lower frequency. This design ensures that the robot will come to a complete stop before a collision occurs, thereby enhancing safety. Furthermore, the safety shield's presence improves the performance of reinforcement learning by preventing collisions with humans during training episodes, ensuring that the episode continues uninterrupted.

Another work [4] delves into the technical aspects of safety, such as the reachability analysis of human body parts, represented by capsules or balls, depending on the specific body part. By connecting these geometric representations, the human's motion can be accurately defined by the motion capture system. This information is then used to assess the potential for collisions between the human and the robotic arm. If a collision is deemed likely, the safety shield is activated to halt the robot immediately, thereby preventing harm.

Overall, the safety shield serves as a vital tool in human-robot interactions, providing a robust mechanism to ensure human safety without limiting the functionality and learning capabilities of the robot.

### *B. Reinforcement learning (A. Schäfftlein)*

*Reinforcement learning* is a machine learning paradigm used for solving sequential decision-making problems by learning from trial and error. Over the last decade, combining this approach with deep neural networks [5] has enabled algorithms to play simple, low-resolution video games [6], complex multiplayer video games [7], and to beat top human players in the demanding board strategy game "Go" [8]. Most recently, deep reinforcement learning has been used to fine-tune large language models and align them with the developers' interests [9]. In robotics, the approach has been used for grasping [10] and in-hand manipulation [11].

A key feature of reinforcement learning is the interaction between the agent and the environment, which spans multiple time steps $t$. Based on an observation of a state $s_t$, the agent takes an action $a_t$ and receives a reward $r_t$ as well as the next state $s_{t+1}$ from the environment [12]. In the concrete example of a video game, the state $s_t$ could be a preprocessed image of the screen, the action $a_t$ could be moving in one of the four directions of a two-dimensional space or firing at enemies, and the reward could be the points awarded by the game [6]. The agent is supposed to maximize the reward [13, p. 2], which can either be a positive scalar (a positive reinforcement), a negative scalar (indicating a punishment), or zero [13, p. 374]. Maximizing the reward requires the agent to explore the environment by taking new actions as well as exploiting courses of action that have proven effective in the past, leading to the exploitation-exploration trade-off, which is not present in other learning paradigms [13, p. 3].

This concept can be formalized by a *Markov Decision Process* (MDP), which is denoted as the tuple $< \mathcal{S}, \mathcal{A}, p, r, s_0, \gamma >$ [12]. The fundamental property of a MDP is that the next state $s_{t+1}$ depends only on the current state $s_t$ and the current action $a_t$ and not the interaction history [12]. This property, which can mathematically be expressed using probabilistic notation as $p(s_{t+1}|s_t, a_t, s_{t+1}, a_{t+1}, ...) = p(s_{t+1}|s_t, a_t)$, is also referred to as the "Markov property" [12]. In reinforcement learning, it is crucial that the environment fulfills the Markov assumption because it allows the agent to compute an action using only the current state. Moreover, the formulation as a MDP enables the use of dynamic programming methods, for example based on equation 7, to solve the reward maximization problem [13, p. 73].

The components of the MDP introduced above include the set of all possible states $\mathcal{S}$, the set of all possible actions $\mathcal{A}$, the transition probabilities $p(s_{t+1}|s_t, a_t)$, a reward $r \sim R(r|s, a)$ as well as a discount factor $\gamma \in [0, 1]$ and the initial state $s_0$ [12]. The transition probabilities determine the probabilities with which the agent will end up in a subsequent state $s_{t+1}$ given a current state $s_t$ and the current action $a_t$ [12]. An environment and the reward function can be deterministic, but we present the general case here. The discount factor $\gamma$ decreases the value of rewards the further they are in the future, hence introducing a preference for rewards that will be received soon (see equations 3, 4, 6, and 7).

The goal of reinforcement learning is to maximize the expected discounted sum of future rewards over trajectories $\tau$ sampled from a reasonable distribution, which can be expressed as follows using the notation introduced above as well as the symbol $\pi$ for the policy [12]:

$$\max_{\pi} \mathbb{E}_{\tau} \sum_{t \geq 0} \gamma^t r_t \,, \quad \gamma \leq 1 \tag{1}$$

In equation 1, the sum is over the subsequent time steps. For each summand of the sum, we multiply the reward $r_t$ of the time step $t$ with the discount factor $\gamma$ to the power of the current time step, effectively reducing the value of rewards further in the future for $\gamma < 1$.

The agent is defined by its policy $\pi$, which is the probability of taking an action $a$ in a state $s$ [12]:

$$a \sim \pi(a|s) \tag{2}$$

The probabilistic component is necessary during training due to the need to explore the environment, but we use the deterministic policy during deployment. In the case of deep reinforcement learning, the policy $\pi$ can incorporate one or more deep neural networks.

The optimal policy $\pi^*$ maximizes the cumulative discounted sum of rewards over the trajectories (see equation 1). Two main deep reinforcement learning approaches for obtaining this optimal policy $\pi^*$ exist: Value iteration methods (section II-C) and policy gradient methods (section II-D). Moreover, algorithms exhibiting features of both approaches, called "actor-critic methods", exist as well (section II-E).

### *C. Value iteration methods (A. Schäfftlein)*

The fundamental concept of *value iteration methods* such as Q-learning [14] is the Q-value function $Q^{\pi}(s_t, a_t)$. It is also referred to as the "action-value function" because it represents the expected cumulative reward for taking an action $a_t$ in a state $s_t$ and then following a specific policy $\pi$ [12]. Mathematically, the Q-value function is defined as [12]:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\tau} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right] \tag{3}$$

As shown in equation 3, the value of the Q-value function $Q^{\pi}(s_t, a_t)$ depends on the current state $s_t$ and the current action $a_t$ as well as the current policy $\pi$. It is computed as the expected value of the discounted cumulative reward $\sum_{t \geq 0} \gamma^t r_t$ over the trajectories $\tau$, conditioned on the initial state $s_0$, the initial action $a_0$ and the policy $\pi$.

The optimal Q-value function $Q^{\pi *}(s_t, a_t)$ is the maximum expected cumulative reward that is obtainable from a given state-action pair [12]:

$$Q^{* \pi}(s_t, a_t) = \max_{\pi} \mathbb{E}_{\tau} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right] \tag{4}$$

Note that the optimal Q-value function maximizes the goal of reinforcement learning (equation 1) for a given initial state $s_0 = s$, an action $a_0 = a$ and a policy $\pi$. Thus, it follows that the optimal policy we want to find maximizes the optimal Q-value function $Q^{\pi *}(s_t, a_t)$ [12]:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{5}$$

Due to equation 5, we can obtain the optimal policy by finding the optimal Q-value function, which is now discussed. The cumulative discounted reward $\sum_{t\geq 0} \gamma^t r_t$ in equation 4 can be decomposed into the immediate reward $r(s,a)$ and the discounted future reward $\gamma Q^{*\pi}(s', a')$, where the $'$ indicates the next state. By rewriting the equation further, we arrive at the Bellman optimality equation (here for a deterministic reward) [12]:

$$Q^{*\pi}(s,a) = \mathbb{E}_{s' \sim p(s'|s,a)} \left[ r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^{*\pi}(s', a') \right] \tag{6}$$

It can be shown that by using the Bellman optimality equation as an iterative update rule for fixed-point iteration with a suboptimal Q-value function $Q^{\pi}(s,a)$, as shown in equation 7, the Q-value function will converge over time to the optimal Q-value function if the Q-function is implemented as tables, and all state-action pairs are updated during each iteration [12]. In this iterative update rule, the new estimate $Q_i^{\pi}(s,a)$ is computed using the old estimate $Q_{i-1}^{\pi}(s', a')$:

$$Q_i^{\pi}(s,a) = \mathbb{E}_{s' \sim p(s'|s,a)} \left[ r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_{i-1}^{\pi}(s', a') \right] \tag{7}$$

Because the max operation over the action space in equation 7 is computationally infeasible for continuous action spaces, vanilla value iteration methods are only applicable to discrete action spaces [12]. However, an approximation of the max operation using an optimal approximating policy, which is trained using gradient descent, is possible [15]. This approach is referred to as “Deep Deterministic Policy Gradients” (DDPG) [15].

In classical reinforcement learning, the Q-function is represented by a Q-table. *Deep Q-learning* introduces a deep neural network to approximate the same information as $Q^{\pi}(s,a,\theta)$, with $\theta$ being the parameters of the neural network [6], increasing generalization and scalability but removing the theoretical convergence guarantee. The possibility of divergence was one of the reasons why deep reinforcement learning initially received less attention than linear function approximators with better convergence guarantees [6].

For deep Q-learning, interactions with the environment are stored as tuples $e_t = (s_t, a_t, r_t, s_{t+1})$ in a so-called “replay buffer”, from which tuples are randomly sampled with replacement for updating the neural work that represents the Q-function [6]. The update step is computed with equation 7, the Bellman equation [6]. The ability to reuse samples makes value-based reinforcement learning algorithms such as deep Q-learning more sample-efficient than policy gradient methods [12], which are presented in the next section and do not feature the ability to reuse samples.

## *D. Policy gradient methods (A. Schäfftlein)*

*Policy gradient methods* take a different approach and maximize the expected reward directly by gradient ascent [16]. For this, the gradient $\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_\tau \left[ \sum_{t\geq 0} \gamma^t r_t \right]$ is required [17]. However, the computation of this expression requires the gradient to be inside the expected value [17]. For this reason, it is necessary to rewrite the expression, which introduces the logarithm in equation 8 [17]. Moreover, we use the fact that only the policy $\pi_\theta$ depends on the network parameters $\theta$ with respect to which we compute the gradient to arrive at [17]:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[ \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t, s_t) \right] \tag{8}$$

$r(\tau)$ is computed as $\sum_{t' \geq 0} \gamma^{t'} r_{t'}$ with $t'$ as the variable for the subsequent time steps. Unfortunately, this gradient estimator suffers from high variance [17]. This is because we use the reward $r(\tau)$ of the entire trajectory $\tau$, which means that we increase the likelihood of all actions executed during a good trajectory [17]. The variance of the gradient estimator can be reduced by augmenting the vanilla gradient descent approach, as described in the next section.

## *E. Actor-critic methods (A. Schäfftlein)*

*Actor-critic methods* reduce the variance of the gradient estimator presented in equation 8 by combining value- and gradients-based methods. The main idea behind actor-critic methods is to augment policy gradient methods with a value-based baseline function for the reward, reducing the gradient estimator's variance [17]. The variance of an estimate for the expected value $\mathbb{E}_x[f(x)]$ can be reduced by subtracting a so-called baseline $b(x)$ [18] with the following properties: Firstly, a baseline has to fulfill $\mathbb{E}_x[b(x)] = 0$ so that the expected value remains unchanged [17]. Using the linearity of expectation, we can see that subtracting a baseline does not change the expected value:

$$\mathbb{E}_x[f(x) - b(x)] = \mathbb{E}_x[f(x)] \underbrace{-\mathbb{E}_x[b(x)]}_{=0} = \mathbb{E}_x[f(x)] \tag{9}$$

Secondly, a baseline should be an approximation of the actual function to ensure optimal variance reduction. In deep reinforcement learning, a typical choice for this baseline with approximating properties is the value function $V^{\pi}(s) = \mathbb{E}_\tau[\sum_{t\geq 0} \gamma^t r_t | s_t]$, which represents the expected reward of a policy starting in state $s_t$, irrespective of the next action $a_t$ [12]. By subtracting our baseline $V^{\pi}(s)$ and rewriting the discounted cumulative reward as the Q-value function $Q^{\pi}(s,a)$, we obtain a popular gradient estimator [17]:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[ \sum_{t\geq 0} \underbrace{(Q^{\pi}(s,a) - V^{\pi}(s))}_{A^{\pi}(s,a)} \nabla_\theta \log \pi_\theta(a_t | s_t) \right] \tag{10}$$

We call the difference $Q^{\pi}(s,a) - V^{\pi}(s)$ of the Q-value function $Q^{\pi}(s,a)$ and value function $V^{\pi}(s)$ the “advantage function" $A^\pi(s,a)$ because it describes what advantage the action provides over the average action [17]. We refer to this approach as an "actor-critic" method because we train an actor (the policy) and a critic (the value function) [17]. An advantage of this approach is that, contrary to deep Q-learning, it can be applied directly to continuous action spaces such as robot control [17].

This actor-critic formulation is the concept underlying a popular algorithm called "Proximal Policy Optimization" (PPO) [19], which builds on the gradient estimator in equation 10. Additionally, PPO uses two concepts originally introduced by trust region methods (TRPO) [20]: Restricting policy updates and using a surrogate objective [19]. Instead of maximizing $\mathbb{E}_\tau[A^\pi(s,a)]$, TRPO maximizes a scaled version ("surrogate objective") $\mathbb{E}_\tau[p_t(\theta)A^\pi(s,a)]$ with $p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ [19]. $\pi_{\theta_{old}}$ denotes the policy before the update step [19]. The surrogate objective $L^{CLIP}$ used by PPO introduces additional clipping:

$$L^{CLIP} = \mathbb{E}_\tau[\min\Big(p_t(\theta)A^\pi,\, clip(p_t(\theta), 1-\epsilon, 1+\epsilon)A^\pi\Big)] \tag{11}$$

This nested structure ensures that a lower (pessimistic) bound of the unclipped objective is used, which increases the stability compared to vanilla policy gradient methods [19].

## F. Related work (H. Friis)

Many investigate, experiment and develop new designs or tools in the field of human robots. Humans can get help from the human robot in different tasks, such as lifting heavy objects. When doing tasks together it is especially important to remember to work in a safe, efficient, and successful manner [21].

A way that human robots have been used that relates to our topic is the game approach. In the field of human robot games, there is a need for decision-making. The robot has to learn new skills to communicate and make proper decisions against or with the human [22]. The game field relates to games in our physical world and not electronic games. When having real human robots and a real human the interaction is thereby physical. The human robots also have access to information digital which is needed and used when making decisions. Games have many limitations which is beneficial and at the same time limiting for the human robot. Such as a game takes place in a restricted space and with specified rules to follow which benefits the human robot. On the other hand, games are limited to a certain time period which may not be enough to fully replicate some of the real world scenarios which the games can symbolize [23]. This relates to our project as the robot in our case also has to make a decision when observing the human's action which it learns in the training with the recorded animations such as moving backwards to avoid colliding with the human.

## III. Concept Overview

*H. Friis, A. Schäfftlein*: The main contribution of this paper is an evaluation of the use of reinforcement learning algorithms in unity with a safety shield [4] in real-life evasion tasks. The safety shield stops the Schunk robot with six degrees of freedom before a collision with the human could occur [4]. In particular, we use the task of evading a human trying to box the robot, while the agent is supposed to stay in its home position (the upright position) if it is reasonably safe for it to be there.

Because the agent has to be trained in the simulation, the first task is to capture human boxing movements for use in the simulation (section III-A). These are then used to train the agent in simulation (section III-B). Afterwards, the agent is deployed on the real robot (section III-C). Both in simulation and in deployment we have observed how the agent behaved. With the observations in mind, we could start over in an attempt to create an even better reward function.

### A. Motion capturing setup (C. Facey-Smith)

Our research employs a 6-camera Vicon system for efficient tracking of motion points in a global coordinate frame. This system records data from reflective markers placed on an individual's upper body joints, tracking these points through 3D space at approximately 100 Hz with a root-mean-square error of less than one millimeter [24]. The focus on the upper body is due to the hardware limitations, the specific nature of the HRI under investigation, and the table-mounted position of the robot.

Before each new recording session, the camera system is re-calibrated to account for any environmental changes in the laboratory. The individual being recorded assumes a T-pose for the initial recording. This pose is used to align the local marker coordinate systems with the global coordinate system and to adjust the offsets of the joints [24].

Following the recording process, the captured pose data is passed through a post-processing pipeline to transform it into a simulation animation that can be used in subsequent training and deployment steps. Initially, the Vicon tracker software exports the data to a *CSV* format. The pipeline then fills in any missing values using polynomial interpolation and removes outliers. The data is also smoothed using a 4th-order Butterworth filter.

The cleaned data is then converted into Biovision Hierarchy (*BVH*) format, a popular format for representing 3D character animation. It is then transformed into a format compatible with *Mujoco*, a physics engine widely used for simulating the complex movements and interactions of robots [24].

### B. Training setup (A. Schäfftlein)

Because reinforcement learning requires many interactions with the environment, we have to train the agent in a simulated environment. For this purpose, we use the Mujoco physics engine [25]. Furthermore, we use the reinforcement learning implementations provided by stable baselines 3 [26], facilitating parallelization through simultaneous training in multiple environments. This library and our code are both implemented in Python. We use mainly the stable baselines 3 default hyperparameters. However, after preliminary tests, we reduced the network size for all algorithms to two layers of 16 neurons each and switched to the parametric ReLU activation function [27] where applicable.

We evaluate two cases: One in which the action space is discrete and one in which the action space is continuous but reduced to one joint. We provide further details on the implementation in section IV-A. For the discrete action space, we use PPO [19] as well as a deep Q-network (DQN) [6]. The DQN implementation uses gradient clipping, a target network, and the Huber loss function. In the continuous case, we use PPO [19] and optionally twin-delayed DDPG (TD3) [29].

We train the agents for 200 000 time steps. Metrics such as the cumulative reward are logged using the framework "Weights and Biases"[1]. For the DQN, we reduced the replay buffer size to 100 000 and started the training at time step 40 000. As the observation space, we use the distance to the left hand, right hand, or head, depending on which is the closest. Moreover, we used 15 recordings fitted together randomly to create the interaction sequence.

Because reinforcement learning results can depend significantly on the random initialization (i.e., the random seed) [28], we run each algorithm configuration with five different seeds (14, 32, 56, 77 and 92). These results are then used to compute the mean and standard deviation for specific metrics over the course of the training process (see section V-A).

### C. Deployment setup (Y. Chen)

Having obtained a trained agent from the previous parts of the pipeline, it can be deployed on the hardware environment of this project. Here, it has to be taken into account, that the interfaces between the agent and the environment during deployment are the same as during training. For example, if the trained agent has been trained on specific hand, end manipulator, and robot joint positions in the simulated environment, the same observations have to be established in the hardware communication, moreover, also in the chronology in which these are fed to the agent. For one to assure this, the observation keys in the configuration files of the environment have to be unchanged when deploying.

Said observation keys are read by the real time communication protocol of the hardware environment which is the UDP. This is realized within the deployment repository through a Python script, which makes a correspondence between the observation keys and the actual sensor data flow.

In a separate script, the actual agent is loaded through its run ID in the configurations. The agent then gets the observations as an input with which the computed action is returned. Depending on the implementation, additional post-processing steps can be added to the action such as clipping.

It has to be again noted, that the fed observations to the agent in simulation and deployment have to be identical in type and chronology. As customized observations can be added in the environment of the agent, the same processing steps to the initial observations have to be made in the deployment repository. This can be either done in the UDP or deployment script.

[1]www.wandb.ai

## IV. Proposed method and implementation

This section presents the proposed method and implementation for improving a reinforcement learning agent's control of a robotic arm. It discusses the use of action space wrappers, the development of reward functions, and the challenges faced during deployment.

### A. Action space wrappers (C. Facey-Smith)

In the context of this project, we utilize 'wrappers' as Python files that extend the base functionality of the *robosuite* Wrapper class. These wrappers encapsulate an instance of a *robosuite GymWrapper()* call, which includes the defined environment, variables, and functions. They then append additional functions to this base, thereby providing supplementary functionality tailored to our specific requirements.

We implemented two custom wrappers in this project: the 'skill wrapper' and the 'reduce wrapper'. The skill wrapper is designed to facilitate the application of reinforcement learning algorithms with a discrete action space, such as a deep Q-network, to the interaction between the robot and the human. It achieves this by transforming an integer action, provided by the agent, into a change in joint angles. This transformation is computed as the difference between the goal and current robot configuration in joint space. It is important to note that this wrapper is specifically designed for use with a discrete reinforcement learning algorithm.

On the other hand, the reduce wrapper is designed to streamline the action space of a continuous reinforcement learning agent by actuating a predetermined number of joints. This allows for more focused control, with the flexibility to actuate either the first joint alone or both the first and second joints. This wrapper is intended for use with a continuous reinforcement learning algorithm, such as Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), or Twin Delayed Deep Deterministic policy gradient (TD3).

These wrappers provide a mechanism to tailor the reinforcement learning approach to the specific needs of the human-robot interaction, thereby enhancing the effectiveness and efficiency of the learning process.

### B. Implementation of reward function (Y. Chen)

Being considerably the most influential factor on the final performance of the reinforcement agent, multiple approaches have been conducted to reach its desired behaviour. During the design of each reward function, the taken observations and their subsequent computation to the scalar reward have to be taken into account.

In the first stages of the project, fairly simple, sparse reward functions have been implemented. These rewarded the agent with a scalar value of one when being in the desired position, which is the robot joint configuration represented by an array with zero entries, and a negative ten when a collision of the end effector and either one of the human hands occurred.

Approaches with reward functions of this kind yielded no remarkable results as these lead to a more passive behaviour of the final agent.

A change of reward function topology was made by introducing dense rewards. With these, distance based rewards incorporated trade-offs in between favored proximity to the mentioned desired goal and the penalized proximity to the human hand positions. As a base approach, the product of the reciprocal value of the absolute euclidean distance and an either negative or positive factor have been used to compute the respective rewards, which are cumulatively added for each step.

As of the time of the report, reward functions which rewarded the agent for being in a constant difference to the human hands yielded the most responsive and suitable behaviour. In the aspect of implementation, the final part of the reward consists of computing the negative, scalar difference of the minimal human hand to end manipulator distance and a set value. With this, the highest reward is zero while deviations are penalized.

### *C. Iterative process for evaluating the reward function (H. Friis, A. Schäfftlein)*

Having written a reward function, we use it for training the agent. Then several aspects are considered to decide how well the reward function works and where there is room for further improvement. For different seeds, we observe both the physical behavior in simulation and deployment and the data logged during training in “Weight and Biases”[2].

Observing how the agent behaves in simulation is crucial because it is a more direct way of evaluating it than looking at the reward function. Looking only at the reward function is problematic because it might not produce the behavior we want in the robot.

During the evaluation, we had to consider that one seed is insufficient to evaluate the reward function’s quality because of the strong influence of random initialization. Because of this, we have looked at multiple seeds.

We have logged several metrics, such as the reward during training, in the tool “Weight and Biases”. Besides the visual observation of the robot behavior, the reward curves have been used to determine the number of time steps needed for training. We successfully improved our reward function in an iterative process until the robot’s behavior improved.

### *D. Implementation of deployment (Y. Chen)*

Difficulties were faced during the deployment of fully trained agents due to the difference in the configuration file handling when comparing the training and deployment repositories. As the latter was part of earlier projects, configuration data were managed through reading JSON files. In the newer approach, such as used in the training repository, configuration data is organized in a cascaded manner through YAML files which are read out by the HYDRA system which has been created by the third party Meta. In the deployment script, this is implemented by calling the main function through a decorator in which the directory and the name of the training configuration file is passed as inputs.

After extracting the required training configurations through accessing the respective key words in the nested library, the environment and reinforcement learning model are loaded through functions from the human robot gym training utilities. In the further process, the observations are passed to the agent’s predict function. As the observations are computed by key word comparison in the UDP script, an additional sector for UDP specific observation keys is added to the training configuration as the corresponding observations keys in simulation and UDP do not match in name.

As only discrete agents were used during this iteration of the project, the predict function of the agent returns a scalar value depending on the actual observation. These values correspond to specific joint configurations which are defined in the skill wrapper. Consecutively, its post processing of the scalar action values are added in the deployment pipeline after the predict function. Given the minimum and maximum boundaries as defined in the failsafe controller configuration, the passed actions are clipped to a specific range before being transferred to the robot.

## V. Evaluation

In the following sections, we discuss the results of our experiments. We start with the training process (section V-A) and continue analyzing the agent behavior both in simulation and in reality (section V-B). Afterward, we evaluate the effect of the safety shield (section V-C). Lastly, we present our results on the importance of the sim-to-real gap (section V-D).

### *A. Evaluation of results during training (A. Schäfftlein)*

As section III-B discussed, we trained the reinforcement learning agent in a simulated environment. In the following section, metrics obtained during this training process are discussed. The reward function switches depending on whether the human is closer than $1.5m$. For the case with the close distance, the reward is based on the distance between the desired and current positions in joint space. In the other case, the reward is the distance to the closest body part.

Reinforcement learning results can vary significantly depending on the seed used for the random initialization [28]. In Fig. 1, five different seeds for a deep Q-network together with the hardcoded baseline (black dashed line) are depicted; each seed has its color. We plot the cumulative reward (i.e., the sum of the rewards that were obtained during one episode) over the number of time steps for which the agent interacted with the environment. Furthermore, we use a window filter to smooth the curves, resulting in some missing values. All of the seeds feature a markedly different training process.

To enable a comparison between the discrete and continuous action spaces (see section IV-A), Fig. 2 depicts results obtained with PPO in both cases. PPO was chosen because it is an algorithm that can be applied to both the discrete

[2]www.wandb.ai

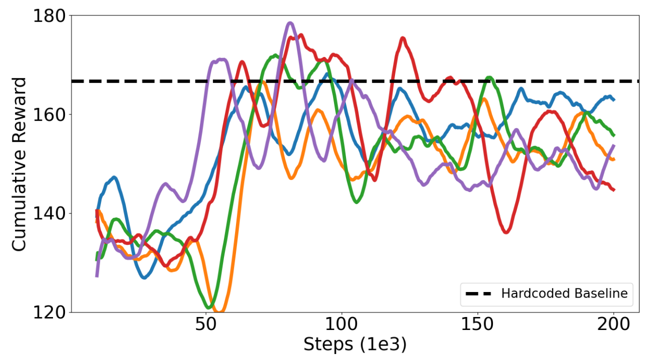

Fig. 1. Cumulative reward over number of time steps for training runs with different seeds for a deep Q-network. The different seeds, each of which has its own color, vary with respect to their performance. The hardcoded baseline is the black dashed line.

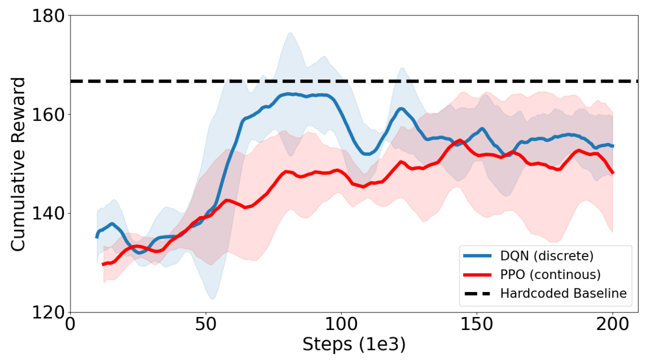

Fig. 3. Comparison of the reward of proximal policy optimization (PPO) in the case with the continuous action space (red) and a deep Q-network with a discrete action space (blue). The performance of the deep Q-network (blue) is initially better but then decays over time, presumable to the instability issues associated with the algorithm.

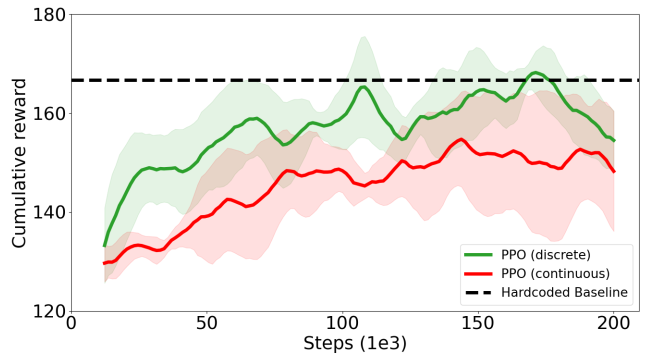

Fig. 2. Comparison of the reward of proximal policy optimization (PPO) in the case of the continuous action space with one actuated joint (red) and the discrete action space with two goal positions reached by the actuation of one joint (green). On average, the algorithm with the discrete action space outperforms the algorithm with the continuous action space.

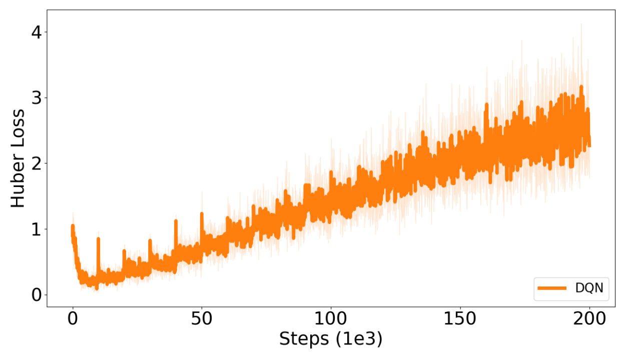

Fig. 4. Huber loss for deep Q-network with gradient clipping and target network over number of time steps. The loss increases steadily over time, which is one possible reason for the performance decrease that can be observed in Fig. 3.

and the continuous action space, allowing us to evaluate the effect of the action space without tainting the comparison by performance differences due to the different algorithm types. Nonetheless, minor differences, for example, regarding the exploration scheme, remain. On average, the version with the discrete action space outperforms the version with the continuous action space (Fig. 2). Due to the limited number of seeds we have used, we must be cautious with drawing conclusions. However, if valid, the results might come about because choosing between two separate configurations might be more straightforward than actuating a robot joint.

The comparison between the discrete and continuous action spaces is less conclusive if we compare PPO to a deep Q-network (see Fig. 3). Here, the cumulative reward over the number of time steps is plotted in blue for the DQN and red for PPO. Firstly, the mean reward curves are much closer than in the previously mentioned case, especially at the beginning and at the end of the training process. Secondly, the choice of different algorithms reduces the comparability. In particular, we can notice a decrease in the performance of the DQN after about 100 000 time steps, which is typical for value-based methods. The instability issues with value-based deep Q-learning (see section II-C) are the most likely explanation for this. As shown in Fig. 4, the loss increases steadily for the deep Q-network, which indicates divergence. The stable baselines 3 implementation we use does implement measures to alleviate this downside of value iteration methods, for example, gradient clipping and the target network. However, these methods are not effective in the case at hand.

PPO, on the other hand, shows a more steady increase in performance without significant drops (see Fig. 3). This observation is in line with the fact that policy gradient methods such as PPO possess a theoretical convergence guarantee for sufficiently small learning rates [17]. Small declines in performance indicate that the learning rate is too large for the current objective function topology, causing temporary divergence which subsides quickly. Moreover, the slower learning process for PPO, which is evident between time steps 50 000 and $100\,000$, can also be partially caused by the lower sample-efficiency of gradient-based methods in comparison to value-based methods [17].

In addition to the reward function that was used for the results that have already been presented, we have investigated a second reward function. This reward function switches the reward depending on whether the human is closer or further away than $1.5m$ like the previous one but uses a symmetric reward: For the low-distance case, we punish the agent for the distance to the evasion position and for the high-distance case we punish the agent for the distance to the home position.

In Fig. 5, the results of a DQN in the discrete case (blue), PPO in the discrete case (green), PPO in the continuous case (red), as well as TD3 in the continuous case (orange) are plotted. Even though we train for the same number of time steps, we can see much more apparent convergence with the new approach (Fig. 5) than with the old one (Fig. 2 and Fig. 3). The easier task makes the difference between the convergence properties of the DQN and PPO, which has been discussed previously, less evident. In addition, we can see an apparent increase in the performance of the deep Q-network after about $40\,000$ time steps when it starts to update. Moreover, the standard deviation, which is represented by the shaded areas, is significantly reduced with the symmetric reward function in comparison to the previous approach (Fig. 2 and Fig. 3).

As shown in Fig. 5, the two discrete algorithms, the DQN (blue) and PPO (green), once again outperform the two continuous algorithms, PPO (red) and TD3 (orange). This observation supports the previously stated hypothesis that learning the task in the discrete action space is more straightforward than in the continuous one.

We compare the reinforcement learning results to a hard-coded baseline to evaluate the qualities of learning-based approaches compared to knowledge-based approaches (Fig. 1, 2, 3, 5). If the robot behavior is supposed to switch depending on whether the human is closer or farther away than a certain distance, it is straightforward to code a baseline based on a switching condition. Our baseline moves to the home position if the human is further away than $1.5m$ and to the dodging position if the human is closer than $1.5m$, which is the same behavior that is encouraged by our reward functions.

As shown in Fig. 2, Fig. 3, and Fig. 5, none of the tested algorithms can outperform the baselines for an extended period in terms of reward. Nonetheless, single seeds are indeed able to surpass the baseline shortly. This becomes evident in Fig. 1 as well. One reason for the lower performance is that reinforcement learning methods do not always learn the exact distance to switch. This behavior could be caused by the two parts of the switching condition featuring different probabilities of occurring, leading the algorithm to exhibit a preference for one because it maximizes the expected reward over its interactions. Moreover, stability issues decrease performance. Firstly, value iteration is potentially unstable (section II-C). Secondly, a high learning rate can decrease the performance of policy gradient methods such as PPO (section II-D).

The following section discusses the agent's performance in the simulation and during deployment on the real Schunk robot. This analysis is based on visually observing the robot's behavior.

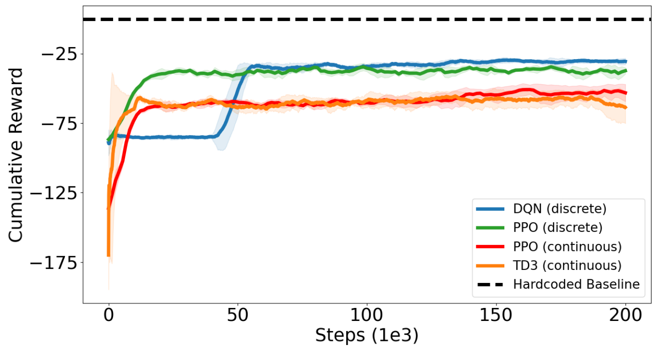

Fig. 5. Comparison of different algorithms and actions spaces for a symmetric reward function which punishes the agent for the distance to one of two joint configurations that are chosen depending on the distance of the closest body part of the human. The change in the reward function leads to much clearer convergence behavior and smaller confidence intervals. Blue: DQN with discrete action space. Green: PPO with discrete action space. Red: PPO with continuous action space. Orange: TD3 with continuous action space.

## *B. Analysis of resulting robot behavior (H. Friis)*

We made many observations when deploying the robots which we will go into detail in this section.

We changed the observation space for the agent such that instead of looking at both the right and left hands distance to the end effector it only looked at the smallest distance of these two. Meaning the closest hand for the observation space. Thereby keeping it more simple and we got more precise results with this.

We observed when deploying some of the agents that it reacted only to the left hand when trying to box. This could be because there was a limited amount of animations with the right hand trying to box and when training the agent for a long time on that limited amount of animations it overfitted to only react to the left hand. The reason for the limited amount of animations with the right hand could be that the person that was recorded trying to box was left-handed and thereby without noticing did not use the right hand as much. We solved it partially by changing the observation space as described.

The current set switching condition in the reward function is set such that sometimes we observed that the distance between the hand and the base of the robot resulted in oscillations. This means that the robot constantly moved between going back (dodging) or going to the home position as it was not sure what to do. This could be because of the limited animations such that it was not trained or had learned what to do in this scenario. Another reason could be that the switching condition was not final in this specific case thereby leading the robot to not be sure what to do either. A third reason could be because the end effector crossed the distance for the switching condition of 1.5 meters and thereby constantly getting a new action to move the opposite way.

### C. Analysis of the effect of the safety shield (H. Friis)

We were very careful when working with the robot and by having the safety shield we could make sure that we remained safe. The safety shield was set with a switching condition at 1.5 meters. This meant that when we were closer than 1.5 meters the safety shield will be activated and the robot will freeze at the position it was in. The idea was to make sure that it was triggered as few times as possible. This made it in some cases difficult to test the reward function in reality because the safety shield was too restrictive.

The safety shield is working with capsules to illustrate the volume of a body part where the sensors are placed to ensure that we work in a safe environment. We experimented with different sizes of capsules as we experienced that the safety shield was blocked too early in our deployment. The result of making the safety shield have smaller capsules while still being realistic was that the safety shield was less restrictive and the robot reacted to more of our human different actions.

### D. Sim-to-real gap observation (C. Facey-Smith, H. Friis, A. Schäfftlein)

The training of reinforcement learning agents often necessitates numerous interactions with the environment, leading to the frequent use of simulations to expedite training [30]. However, the simulated environment merely approximates components such as robot dynamics, potentially diminishing the performance achieved in the real world [30]. Given that our model was also trained in simulation, the sim-to-real gap could impact our performance.

Evaluating the sim-to-real gap presented a challenge as it required identical motion in both the real robot and the simulation. This issue was addressed by conducting motion capture during an interaction with a hard-coded agent and subsequently running the agent again in simulation on the recorded motion sequence. The real data was recorded in the UDP file, the Python interface to the real robot, and saved post-interaction. Concurrently, the interaction was recorded using the Vicon motion capture system, transformed to Mujoco format, and the simulated data was subsequently recorded (section III-A).

To ensure the exact sequence, no offsets in the x- and y-direction were used during recording. However, a z-direction offset was necessary due to shifted coordinate frames. A potential issue with this approach is the time offset between the start of the motion capture procedure and the start of the Python UDP file, which was manually estimated to align the two recordings.

To evaluate the sim-to-real gap, we compared the joint angle of the second joint of the Schunk robot, crucial joint for our evasion task, in reality and simulation during an interaction (Fig. 6). Our results indicate that there is in fact a sim-to-real-gap. The real robot results (green) are smoother than those in simulation (orange), potentially due to an underestimation of friction in the simulation, resulting in a more agile robot.

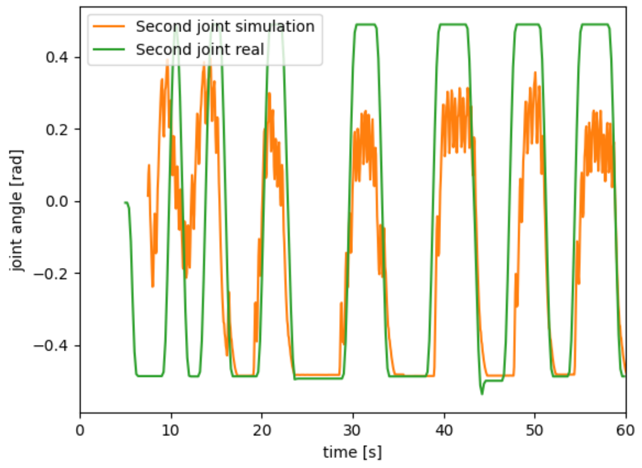

Fig. 6. Comparison of the angle of the second joint of the Schunk robot in simulation (orange) and reality (green). We plot the joint angle in radians over the time in seconds. For the simulation, we have used the motion capture recording of the real interaction.

Additionally, the simulated robot does not always move as close to the human as permitted (a joint angle of $+0.5\,rad$), unlike the real robot. This discrepancy could be due to the human being closer to the robot in the simulation than in reality, altering the robot's response.

However, shifts in the human's position might not necessarily reduce the robot's overall performance as it should be capable of handling different human positions. Random offsets during training and testing could have been applied. Compared to the sim-to-real gap encountered when using video data from simulation and reality, we do not leave the manifold of the training data due to the sim-to-real gap.

Further experiments on the effects of the sim-to-real gap are recommended to gain a more comprehensive understanding of the issue.

## VI. Future work (H. Friis)

If we had the possibility for example if given more time it would have been interesting to experiment more with the reinforcement learning environment and experiment with larger data sets.

To experiment more within the environment of reinforcement learning we could have looked further into the different parameters and which effect they have if we changed them.

To experiment with larger data sets it could be in the form of more recorded animations. Where the animations also were of different types such as a few where the human is far away from the robot, a few where the human is near the robot, and a few where the human moves between being far away and close to the robot. It could furthermore also be interesting to experiment with having different sets of data. For example, the training animations being different from the animations used in the simulation. Thereby could we potentially see more closely how the robot would behave in reality before deploying it on the real robotic arm.

## VII. Conclusion (Y. Chen)

During this work, the possibilities for safe human-robot interactions have been explored using nowadays's robotics and artificial intelligence algorithms. To lay the groundwork for more reliable and interact able robots, a reinforcement learning based approach has been conducted.

Despite all of the models showing fast convergence during training, the inner workings of a reinforcement learning agent still remains a black box, and hence the arbitrary computation of robot actions. Measures regarding human safety from this perspective are an open topic. Additionally, the early activation of the safety shield led to a restrictive behaviour of the agent during deployment. To account for these factors, the human model in simulation and the actual human during deployment were placed in a safe distance to the robot. Said distance was implemented in the final reward function by setting a switching condition at that point.

Regarding the progress in the implementation, a streamlined workflow starting from the design of the reinforcement learning environment, training, testing in simulation up to the deployment of the agent on the actual robot has been successfully achieved. Configuration parameters can be easily accessed and customized while being simultaneously used in training and deployment.

## References

[1] Guiochet, J., Machin, M. and Waeselynck, H., 2017. Safety-critical advanced robots: A survey. Robotics and Autonomous Systems, 94, pp.43-52.

[2] Lee, K., Shin, J. and Lim, J.Y., 2021. Critical hazard factors in the risk assessments of industrial robots: causal analysis and case studies. Safety and health at work, 12(4), pp.496-504.

[3] Thumm, J. and Althoff, M., 2022, May. *Provably safe deep reinforcement learning for robotic manipulation in human environments*, 3rd ed. In 2022 International Conference on Robotics and Automation (ICRA) (pp. 6344-6350). IEEE.

[4] Schepp, S.R., Thumm, J., Liu, S.B. and Althoff, M., 2022, May. Sara: A tool for safe human-robot coexistence and collaboration through reachability analysis. In 2022 International Conference on Robotics and Automation (ICRA) (pp. 4312-4317). IEEE.

[5] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986. Learning representations by back-propagating errors. nature, 323(6088), pp.533-536.

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[7] Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P. and Oh, J., 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 575(7782), pp.350-354.

[8] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), pp.484-489.

[9] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A. and Schulman, J., 2022. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, pp.27730-27744.

[10] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V. and Levine, S., 2018, October. Scalable deep reinforcement learning for vision-based robotic manipulation. In Conference on Robot Learning (pp. 651-673). PMLR.

[11] Andrychowicz, O.M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A. and Schneider, J., 2020. Learning dexterous in-hand manipulation. The International Journal of Robotics Research, 39(1), pp.3-20.

[12] Bäuml, B., 2023. Advanced Deep Learning for Robotics: Lecture 7. Lecture material provided by the Technical University of Munich.

[13] Sutton, R.S. and Barto, A.G., 2018. Reinforcement learning: An introduction. MIT press.

[14] Watkins, C.J.C.H., 1989. Learning from delayed rewards. Thesis (Ph. D.). King's College, Cambridge.

[15] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[16] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016, June. Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.

[17] Bäuml, B., 2023. Advanced Deep Learning for Robotics: Lecture 8. Lecture material provided by the Technical University of Munich.

[18] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8, pp.229-256.

[19] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[20] Schulman, J., Levine, S., Moritz, P., Jordan, M.I. and Abbeel, P., 2015. Trust Region Policy Optimization (TRPO). CoRR abs/1502.05477.

[21] Ajoudani, A., Zanchettin, A.M., Ivaldi, S., Albu-Schäffer, A., Kosuge, K. and Khatib, O., 2018. Progress and prospects of the humanârobot collaboration. Autonomous Robots, 42, pp.957-975.

[22] Lee, K.W. and Hwang, J.H., 2008. Human-robot interaction as a cooperative game. Trends in Intelligent Systems and Computer Engineering, pp.91-103.

[23] Xin, M. and Sharlin, E., 2007, September. Playing games with robots-a method for evaluating human-robot interaction. In Human Robot Interaction. IntechOpen.

[24] Dobers, S., Nathaus, C., Balletshofer, J., 2022, August. Safe Reinforcement Learning on a Real Robot.

[25] Todorov, E., Erez, T. and Tassa, Y., 2012, October. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems (pp. 5026-5033). IEEE.

[26] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. and Dormann, N., 2021. Stable-baselines3: Reliable reinforcement learning implementations. The Journal of Machine Learning Research, 22(1), pp.12348-12355.

[27] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

[28] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D., 2018, April. Deep reinforcement learning that matters. In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1).

[29] Fujimoto, S., Hoof, H. and Meger, D., 2018, July. Addressing function approximation error in actor-critic methods. In International conference on machine learning (pp. 1587-1596). PMLR.

[30] Salvato, E., Fenu, G., Medvet, E. and Pellegrino, F.A., 2021. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. IEEE Access, 9, pp.153171-153187.