KIcker: An Industrial Drive and Control Foosball System automated with Deep Reinforcement Learning

Preprint · April 2020

DOI: 10.13140/RG.2.2.19201.28003

6 authors, including:

Stefano De Blasi
Bosch Rexroth
16 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Sebastian Klöser
DXC Technology
5 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Robin Reuben
DXC Technology
5 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Fabian Sturm
Bosch Rexroth
2 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Journal of Intelligent & Robotic Systems manuscript No.
(will be inserted by the editor)

KIcker: An Industrial Drive and Control Foosball System automated with Deep Reinforcement Learning

Stefano De Blasi[*,1,3] · **Sebastian Klöser**[2] · **Arne Müller**[2] · **Robin Reuben**[2] · **Fabian Sturm**[1] · **Timo Zerrer**[2]

the date of receipt and acceptance should be inserted later

Abstract The majority of efforts in the field of sim-to-real Deep Reinforcement Learning focus on robot manipulators, which is justified by their importance for modern production plants. However, there are only a few studies for a more extensive use in manufacturing processes. In this paper, we contribute to this by automating a complex manufacturing-like process using simulation-based Deep Reinforcement Learning. The setup and workflow presented here are designed to mimic the characteristics of real manufacturing processes and proves that Deep Reinforcement Learning can be applied to physical systems built from industrial drive and control components by transferring policies learned in simulation to the real machine. Aided by domain randomization, training in a virtual environment is crucial due to the benefit of accelerated training speed and the desire for safe Reinforcement Learning. Our key contribution is to demonstrate the applicability of simulation-based Deep Reinforcement Learning in industrial automation technology. We introduce an industrial drive and control system, based on the classic pub game Foosball, from both an engineering and a simulation perspective, describing the strategies applied to increase transfer robustness. Our approach allowed us to train a self-learning agent to independently learn successful control policies for demanding Foosball tasks based on sparse reward signals. The promising results prove that state-of-the-art Deep Reinforcement Learning algorithms are able to produce models trained in simulation, which can successfully control industrial use cases without using the actual system for training beforehand.

Keywords Deep Reinforcement Learning · Industrial Automation · Real-Time System · Unity3D · Sim-to-Real

*Corresponding author
Tel.: +49 9352 18-3246
E-mail: Stefano.DeBlasi@boschrexroth.de
ORCID: 0000-0003-0459-5956

[1] Bosch Rexroth AG
97816 Lohr am Main
Germany

[2] DXC Technology
71034 Böblingen
Germany

[3] University of Applied Sciences Fulda
36037 Fulda
Germany

1 Introduction

Current innovations in automation technology aim at the seamless integration of manufacturing facilities into the production environment with real-time data exchange. The increasing data complexity opens up new opportunities to improve process quality or avoid downtime. However, as manufacturing data becomes more complex, it becomes increasingly difficult to find optimal solutions using conventional methods and human programming. This is especially critical when considering flexible production lines and ever-shorter development cycles (e.g. technical consumer goods or fashion items).

Machine Learning holds tremendous potential for automation technology and intelligent manufacturing (Zhong et al., 2017; Weichert et al., 2019). One way in which improvements are sought is by learning industrial process control strategies (Yin et al., 2014), also called policies. In the subfield of Reinforcement Learning (RL), a policy maps observations to actions and thereby determines the behavior of a so-called agent. Such agents can learn by interacting with their environment and receiving feedback on their actions in the form of a reward. The goal of RL techniques is for the agent to learn a

behavior that provides high rewards, measured e.g. by production quality in a manufacturing process. Thus, the RL problem can be formalized as optimizing the agent's policy with respect to the cumulative reward obtained in individual trials. In Deep Reinforcement Learning (DRL), performance can be enhanced compared to classical approaches by using a deep neural network as the policy. This can be necessary in particular in industrial applications since industrial control processes can be very complex. In the recent past, DRL systems have been successfully applied to various complex challenges such as learning computer and strategy games (Mnih et al., 2015; Silver et al., 2016) or robot control (Andrychowicz et al., 2020). The high degree of autonomy as well as the impressive performance of the learned policies motivate the search for real-world applications of DRL in industry and business settings. This effort is hindered by the fundamental problem of sample inefficiency of state-of-the-art DRL algorithms, which need a lot of experience to develop successful policies. This property limits the applicability of DRL directly to real-world systems. To overcome this issue, the use of simulations is often preferred over a direct application. Such simulations have several advantages thanks to which they are frequently used evaluation and comparison in RL (Brockman et al., 2016). First, simulations are easy to share and ensure identical testing conditions for developers all over the world. Second, no feature-extraction or rating functions need to be implemented because the simulation directly provides such information. Third, with virtual systems there is no risk of physical hardware damage or real-world harm. Fourth, to learn a complex task an agent might need to gather hundreds of years of experience, which is hardly feasible. Training in a simulation, however, can make it possible to counteract this problem of inefficiency by running the simulation many orders of magnitude faster than real-time. The use of simulated environments to train DRL agents with the goal of transferring them back to a real-world system is known as sim-to-real DRL. Sim-to-real projects suffer from the so-called reality gap, i.e. the discrepancy between simulation and reality. The larger this gap is, the greater the expected performance loss of the policy learned in the simulation when evaluated in reality.

Sim-to-real DRL for industry: Sim-to-real DRL projects must address the reality gap to enable the successful transfer of policies learned in simulation to the respective real physical systems in question. With regard to the reality gap in robotic manipulation tasks, Collins et al. (2019) discuss the need for precise simulations and the shortcomings of commonly available state-of-the-art simulators. A widely followed and investigated approach to cope with such shortcomings is the use of noise, which is artificially introduced into the simulation and the DRL agent's observation and action system. With such domain randomization, successes have been achieved for a wide range of tasks like quadruple locomotion (Tan et al., 2018), dexterous in-hand manipulation (Andrychowicz et al., 2020) and robot manipulation (Peng et al., 2018). More explicit studies of domain randomization showed that applying policies trained with it are about 20 times more efficient after being transferred to the physical system than without (van Baar et al., 2019). Moreover, one analysis found that, for continuing training on the real system after the sim-to-real transfer, the number of real-world samples could be reduced by a factor of 50 when domain randomization was applied in the simulation (Bousmalis et al., 2018). In the future, such approaches will be especially interesting for industrial applications. For example, a furniture assembly environment based on Mujoco has been introduced as a benchmark environment for manipulation tasks (Lee et al., 2019). Using sim-to-real training with domain randomization could help improve and accelerate the workflow leading to DRL solutions like solving complex insertion tasks for a robotic arm using visual inputs (Schoettler et al., 2019). Further examples are the combination of DRL and operational space force controllers producing high-precision control (Luo et al., 2019) or vision-based dynamic manipulation skills for gripping tasks on robot manipulators (Kalashnikov et al., 2018). It is noteworthy that the majority of efforts in this field focuses on robot manipulators, which is justified given their importance for modern production plants. However, there are few studies exploring a more extensive use of sim-to-real DRL in manufacturing processes. Also, it has been noted that, besides the Machine Learning challenge itself, it is not easy to integrate DRL solutions for complex tasks into current industrial control components (Schmidt et al., 2020). In our work, we contribute to both challenges by applying sim-to-real DRL to a gamified manufacturing-like environment with automation control and drive components.

Automation of real-world reaction games: Based on the classic pub game Foosball, we introduce the KIcker (KI is the German acronym for artificial intelligence), see Fig. 1. This complex, real-world system is automated with industrial drive and control components commonly used in automation technology. Depending on the region, Foosball is also known as table football or table soccer. As an automated robot, Foosball can be used as an interesting and versatile platform for Machine Learning approaches (Padon et al., 2003). In the past, the Foosball

game has been successfully semi-automated to enable single player play based on a decision tree developed without Machine Learning methods (Weigel, 2005). This robot was able to recognize the opponent's strength and maximize entertainment value by adapting its action velocities (Weigel et al., 2005). From a scientific point of view, the domain and thus the device is ideally suited to evaluate different learning approaches (Weigel and Nebel, 2008). For example, through the use of recorded games (Zhang and Nebel, 2007b), it has already been possible to imitate human action sequences such as slide-kicking and locking (Zhang and Nebel, 2007a). The rule-violation of spinning a rod can also automatically be detected to ensure fair play (Hornung and Zhang, 2008). To determine the states of the environment, camera-based estimation of the angles of the rods and the position of the ball can be used (Bambach and Lee, 2012). Depending on the desired quality, the camera-based ball tracking system in Foosball can be complex due to the rods (they can obstruct the view of the ball) and the game speed (Janssen et al., 2009, 2012), which has also led to the approach of placing the camera underneath a Foosball table with a transparent playing field (Weigel, 2005). Although there have been several innovations in state-tracking, little has changed in the considerable effort required to program the controller, because even if some of these Foosball robots were able to defeat advanced players, their logic was always coded by programmers. To the best of our knowledge, in the past Foosball has been rejected as a challenge for full RL due to the enormous number of states, the complex patterns of state transitions and the complexity of building a machine that could make precise contact with the ball (Campbell et al., 2009). To take automation to a new level, in this work we focus on the use of a Machine Learning approach to generate logic that reacts appropriately to given conditions in real-time. By using DRL, the Foosball table is transformed into a self-learning system. Although there have been efforts to automate other real reaction games using Machine Learning approaches, such as table tennis robots (Kober et al., 2013) with trajectory prediction (Zhao et al., 2017), air hockey striking (Taitler and Shimkin, 2017) or hockey (Chebotar et al., 2017), this has not led to many applications in the real-time sensitive industry. We also contribute to this by considering industrial circumstances and by explicitly using control architectures, which are suitable for industrial use.

Fig. 1 System overview: A professional Foosball table with semi-automated rods can be controlled by industrial drive and control components. A camera above the playing field captures images, which can be used for perception.

1.1 Own contributions

In manufacturing, sim-to-real RL research mainly focuses on robot manipulators as active objects in the specific task. In contrast to this, we automate a complex manufacturing-like process with common industrial drive and control components using simulation-based DRL. We show that state-of-the-art DRL agents and sim-to-real strategies are capable of independently learning successful policies for demanding KIcker tasks based on sparse reward signals. The Foosball environment will be presented from both an engineering and a simulation perspective, describing the strategies applied to increase transfer robustness and showing the results of the performed experiments. Since our focus lies on the sim-to-real transfer of DRL models, we automate a reduced but sufficiently complex subtask of the Foosball game: scoring goals with the striker rod. In this way, we investigate and demonstrate the potential of DRL for the manufacturing industry beyond the usual robot manipulator scenario and motivate further research in realistic manufacturing scenarios.

In contrast to the majority of the referenced literature, our intended workflow is designed to mimic the characteristics of project work on real manufacturing processes. In particular, we assume that in industry there are strict time constraints for deploying new automation, little possibility to stop production for measurements or experiments and that the software that controls the production is highly integrated into a larger

Manufacturing Execution System (MES) and therefore not readily available for simulations. Based on these constraints, our approach to apply DRL to the KIcker deliberately involves no additional adaptation or recalibration before the model is transferred to the physical system for evaluation, has a restricted training time and models the controls of the KIcker externally. To assess the transmission characteristics from simulation to reality, we evaluate and compare model performance in the simulation and on the real physical system. Finally, we summarize our promising results to hopefully motivate further applied research into the use of DRL for general manufacturing processes. The motivation of this project is not to present a superior Foosball algorithm, but rather to minimize the necessary programming effort for this manufacturing-like system through the use of Machine Learning methods. This is demonstrated via the automation of a subtask and can be applied to more complex KIcker tasks in the future. Superior performance could be achieved by using rule-based approaches. However, implementing such approaches requires high programming efforts, which will only increase for more complex tasks. In contrast, the human effort required to train a DRL model for more complex tasks will increase less strongly and provides a high degree of flexibility. A video of our results is available on youtube (`https://youtu.be/7HBgPtR1Uks`).

The paper is structured as follows: The first section covers the fundamentals and background of Reinforcement Learning, which are necessary for understanding the applied methods in the paper. Next, an overview of the physical KIcker system, the drive and control architecture with its information flow and the related specifics are presented in Sect. 3. Here, we also formulate the problem as decision process. Sect. 4 covers detailed information about the simulation and the realization of the control system. The training procedure, the randomization process, and the final transfer to the real system are described in Sect. 5, as is the training reward progress performed in the simulation. In Sect. 6, we compare the performance of the trained model in the simulation and on the real system and evaluate and the success of the sim-to-real transfer. This allows us to summarize our promising results and briefly touch upon possible next steps for the KIcker and also relevant industrial challenges.

2 Reinforcement Learning: Background and methods

This section gives an introduction to RL and describes the applied methods. Readers who are familiar with RL and policy gradient approaches may want to skip this section. Unless otherwise stated, the presentation follows the theoretical foundations from the classical textbooks (Sutton and Barto, 2018).

2.1 Markov Decision Processes

RL is a Machine Learning domain that focuses on developing algorithmic approaches to learning behavior from interactions. The active learning entity is called the agent and the surroundings it can interact with are referred to as the environment. Formally, RL problems are framed as so-called Markov Decision Processes (MDPs), consisting of sets of states S, actions A, a transition function T_a and a reward structure r_a. In this framework, the interaction between agent and environment is formalized as a discrete sequence of states, actions and rewards: At a given time step t, the agent receives a partial observation, o_t, of the state of the environment, $s_t \in S$, and decides on an action, $a_t \in A$. The dynamics of the environment are given by the probabilistic transition function T_a. For a given action a and a given state s, the transition function gives the probability of transitioning into the state s':

$$T_a(s'|s,a) = \Pr\left(s_{t+1} = s' | s_t = s, a_t = a\right). \tag{1}$$

It is worth noting that the transition function, which is responsible for the dynamics of the whole process, depends only on the current state and not on the previous states. Therefore, the states need to be designed such that they contain all the information required to infer the future state distribution using the transition function. A sequence of states and actions is called a *trajectory* and denoted by τ. The reward structure $r_a : S \times A \to \mathbb{R}$ is a function that maps states and actions to the reward, which is used as a feedback signal to the agent, rating its decisions.

MDPs serve as a framework to describe RL problems. One assumes the presence of an environment that provides states and an (unknown) transition function. In the majority of cases, the reward function is not directly given by the environment but is defined manually according to the task to be solved. The challenge for the agent is to learn to choose actions such that it maximizes the cumulative reward it receives. This challenge can be formalized by introducing the concept of a policy. The agent's policy $\pi_\theta(a|o)$ is a distribution over actions, a, for a given observation, o, from which concrete actions can be sampled. The policy can be represented and constructed in different ways: As a simple mapping table, as a formal rule or with the help of function approximation. As we will make use of function approximation, the subscript θ indicates that π is represented by a model that comes with a set of free parameters. Independently of

how the policy is constructed, the objective of the whole RL problem is to find an optimal policy that maximizes the reward accumulated over time:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \underset{a,s\sim\pi_\theta}{\mathbb{E}} \left[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) \right] \tag{2}$$

Equation (2) is formulated with respect to the policy parameters since we assume these parameters model completely determine the policy. Moreover, for simplicity, we already formulated (2) with a discount factor γ that is required for infinite time horizons (i.e. the number of state-action steps is not limited) but also helpful for finite cases. Based on these considerations, the RL process can be formalized as follows: For a given (non-optimal) policy π the agent interacts with the environment according to that policy, explores its state-space and receives rewards based on the reward function. The information gathered through this process is a set (S, A, R) of states, actions and rewards and it is used by an RL algorithm to update the policy of the agent with the objective of maximising the reward. This process is repeated until the cumulative rewards converge.

2.2 Model-free Reinforcement Learning methods

RL methods can be separated into model-based and model-free approaches. Model-based approaches make use of an explicit model of the transition function of the environment that is either provided analytically or learned with the help of Machine Learning methods. If a model is available, model-based RL can be very efficient. However, obtaining a precise enough model can be highly challenging, especially for physical environments. Because of this, we focus on model-free approaches, which do not require any information about the transition function of the environment. In model-free RL, algorithms can be split into two categories: Q-Learning and policy gradient approaches. We will make use of a modern policy gradient algorithm. However, given its importance, we will also briefly outline the basics of Q-Learning.

2.2.1 Q-Learning

Q-Learning algorithms rely in the evaluation of the *value* of a given state, s. Formally, the value of a state can be described as the expectation value of the cumulative discounted reward received when starting in the state s and choosing actions by following the current policy:

$$V^\pi(s) = \underset{a,s\sim\pi_\theta}{\mathbb{E}} \left[\sum_{k=0}^{\infty} \gamma^k r_{k+1} | s_0 = s \right]. \tag{3}$$

If a value function is given, an optimal policy can easily be inferred by implementing the decision rule of always choosing the action that will lead to the next state with the highest value. To simplify this, one often introduces the concept of the Q-function, which is very similar to the value function but formulated with respect to a state-action pair:

$$Q^\pi(s, a) = \underset{a,s\sim\pi_\theta}{\mathbb{E}} \left[\sum_{k=0}^{\infty} \gamma^k r_{k+1} | s_0 = s, a_0 = a \right]. \tag{4}$$

Given these concepts, Q-Learning algorithms aim to represent and learn a reliable Q-Function from the experience data generated by interacting with the environment. In doing so, the agent needs to balance between always choosing the action with the (currently) highest rated Q-value in a given state and to deviate from the potentially best action in order to gather experience in unseen regions of the state-space. This trade-off between gaining immediate high rewards and exploring the environment is referred to as the problem of *exploitation versus exploitation*. While Q-Learning algorithms have widespread applications, they are primarily designed for scenarios with a finite action-space (although extensions exists).

2.2.2 Policy Gradients

Since the problem we are going to tackle in this paper possesses a continuous action-space, we use policy gradient methods. In the class of policy gradient algorithms, the policy is itself directly optimized rather than being derived from a Q-function (Sutton et al., 2000). The policy $\pi_\theta(a|s)$, a probability distribution over actions a for a given state s, will be represented by a neural network, which receives the (usually normalized) state of the agent as an input, and on every node of the output layer (corresponding to the number of actions), produces the mean of a Gaussian distribution with a variable standard deviation. The individual actions for each decision step are then sampled from these distributions, which are the source of the model's stochasticity. The standard deviation, along with the weights and biases of the neural network, is also part of the model parameters to be optimized in training. Thus, for policy gradient methods, the exploration-exploitation balance is controlled indirectly through the model parameters via the standard deviation.

In the simplest case, the updates of the policy are performed with the help of the following gradient (the derivation can be found in the appendix):

$$\nabla_\theta L(\theta) = \underset{\tau\sim\pi_\theta}{\mathbb{E}} \left(\sum_{k=0}^{\infty} \nabla_\theta \pi(a_k|s_k) \right) \left(\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) \right). \tag{5}$$

This gradient is very similar to a standard maximum likelihood gradient used in supervised learning. The difference is the inclusion of the cumulative rewards, which acts as a weight term for the likelihood terms. The policy updates can be performed after each iteration:

$$\theta' \leftarrow \theta + \alpha \nabla_\theta L(\theta). \tag{6}$$

2.2.3 Advanced Policy Gradient Methods

The simple policy gradient algorithm helps to understand the basics of policy gradient RL, but for the KIcker we use more advanced algorithms. Here, the model parameters θ are learned via an actor-critic approach (Grondman et al., 2012). The main difference between this approach and the simple policy gradient algorithm is a different and more general weight factor, $\hat{A}^\pi(a, s)$, instead of just the cumulative reward:

$$\mathrm{L} = \mathop{\mathbb{E}}_{a,s\sim\pi_\theta}\left[\sum_{k=0}^{\infty} \log\, \pi_\theta(a_k|s_k)\hat{A}^\pi(a_k, s_k)\right]. \tag{7}$$

In this loss function, $\hat{A}^\pi(a, s)$ is the estimated advantage function. It is defined as the difference between the Q and the value function for a given state-action pair

$$\hat{A}^\pi(a, s) = Q^\pi(a, s) - V^\pi(s). \tag{8}$$

The reasoning for using the advantage function as a weight factor in the loss is simple: The value of a state is the expected cumulative reward obtained from that state, while the Q-function gives the cumulative reward for a chosen action in that state. If the advantage is positive the action is better than average and should be reinforced, while it should be avoided if it is negative.

In actor-critic methods, two systems are used in parallel. The actor is the policy itself and responsible for choosing actions, while the critic is used to evaluate the advantage of a given state. In modern algorithms, both systems are built from neural networks, thus actor critic approaches use two different deep neural networks that participate in the training process. Basically, the two networks involved in actor-critic approaches can be designed completely independently of each other. However, one often chooses both networks to share all but the output layers. The reason for this is the fact that both networks need to be trained from the experience gained in the environment, and that the internal representation of the states in the hidden layers that are suitable for decision making should also be a good basis for evaluating the value of the state itself, and vice versa.

The objectives of the two systems are different but interlinked. While the critic system is trained in a supervised fashion to minimize the difference between the predictions and the reward sums obtained in the sampled episodes, the actor is trained using the policy gradient loss.

It is important to note that, as all policy gradient approaches, actor-critic algorithms are online approaches, meaning that all quantities are defined with respect to a given policy. Once the policy changes due to an update, all old data points cannot be used anymore. One way of overcoming this is given by introducing importance sampling (Humayoo and Cheng, 2018) which allows one to reformulate (7) as follows:

$$L = \mathop{\mathbb{E}}_{a,s\sim\pi_\theta}\left[\sum_{k=0}^{\infty} \frac{\pi_\theta(a_k|s_k)}{\pi_{\theta_{\mathrm{old}}}(a_k|s_k)}\hat{A}^\pi(a_k, s_k)\right]. \tag{9}$$

However, using importance sampling induces some instabilities which is why we do not use it directly but rather use a more advanced algorithm for the policy parameter updates: Proximal Policy Optimization (PPO) (Schulman et al., 2017). PPO estimates the gradient for the parameter updates based on the following loss:

$$\mathrm{L}_{\mathrm{PPO}} = \mathop{\mathbb{E}}_{a,s\sim\pi_\theta} \min\left[\sum_{k=0}^{\infty} \frac{\pi_\theta(a_k|s_k)}{\pi_{\theta_{\mathrm{old}}}(a_k|s_k)}\hat{A}^\pi(a_k, s_k),\right.$$
$$\left.\sum_{k=0}^{\infty} \mathrm{clip}\left(\frac{\pi_\theta(a_k|s_k)}{\pi_{\theta_{\mathrm{old}}}(a_k|s_k)}, 1-\epsilon, 1+\epsilon\right)\hat{A}^\pi(a_k, s_k)\right]. \tag{10}$$

This loss consists of two parts. The first one is the standard actor-critic loss in the importance sampling formulation. The second part adds a clipping of the probability ratio defined by the clipping range ϵ. Taking the overall minimum results in establishing a lower bound for the final loss function, which prevents too big parameter updates.

Using this, the final loss used for training is constructed as the sum of multiple losses, the rest of which we don't derive here:

$$\mathrm{L} = \mathrm{L}_{\mathrm{PPO}} + c_{\mathrm{v}}\mathrm{L}_{\mathrm{V}} + c_{\mathrm{ent}}\mathrm{H}. \tag{11}$$

Here, H is an entropy term and c_{v} and c_{ent} are weight coefficients for the value and entropy terms. H is a measure for the randomness of chosen actions. The entropy term is added to address the trade-off between exploration and exploitation. As described above, on the one hand, the policy should be updated such that, in a given state, it chooses the action that promises the maximum reward based on the data that has been gathered (greediness of the algorithm, high exploitation). On the other hand, this can lead the algorithm to quickly converge to only a local maximum which does not represent the optimal policy. To counter this by encouraging a degree of random exploration, the entropy term is added to the

loss. Due to the fact that the standard deviation of the Gaussian distribution around the values given by the output layer of the neural network is also among the model parameters to be optimized, policy updates can widen the probability distributions from which the actions are sampled, increasing the entropy and therefore the strength of exploration. The objective of adding H is the same as using epsilon-greedy policies in Q-Learning.

3 The KIcker drive and control system

In this section, the KIcker system shown in Fig. 1 is presented from an engineering perspective. Additionally, the developed architecture of the interface for the agent's interaction with the table is described.

3.1 Construction

The KIcker system which is to be automated consists of four principal components which which are briefly described below.

Foosball table: A widely used professional Foosball table *Ullrich Sport U4P* with four rods per side is the basis of the project.

Drive and control system: One side of the Foosball table is driven by a set of motors, two for each rod. The rotation is handled by synchronous *IndraDyn S MS2N03* motors and linear *IndraDyn L MCL020* motors are installed for the translation of the rods. These motors are connected to drive converters of type *IndraDrive Cs*. The programmable logic controllers (PLC) *IndraControl XM21* and *XM22* communicate with the drives via the Sercos interface and handle the control of all electrical components. For the Python functionality (for ball-detection and inference using the trained models), an *NVIDIA Jetson TX2* is used. Since the different rods have different lateral position limits, the control actions are restricted. Referring to the lateral position of a rod by the position of its center, depending on the rod, the limits vary from $\pm 55\,\mathrm{mm}$ about the middle for the midfield rod to $180\,\mathrm{mm}$ for the defender rod. The striker rod, which is our main focus in this work, takes lateral positions $p \in [-115\,\mathrm{mm}, 115\,\mathrm{mm}]$. In addition to lateral translation, the rotation angle of each rod takes values $\alpha \in [-90°, 90°]$. These position limits as well as kinematic limits (e.g. maximum jerk, velocity and acceleration) are specified by the KIcker system and kept fixed. They are enforced before a new command is set. Furthermore, the parameters of the actuator control (cascade control) remain unchanged with default values.

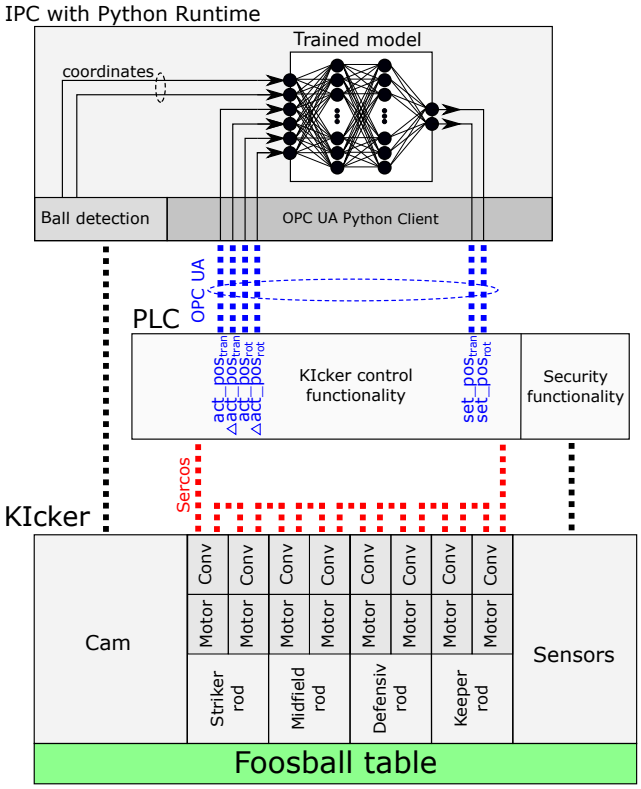

Fig. 2 Architecture of KIcker system: The drives are controlled via Sercos communication on a PLC. This PLC also takes care of the security functionalities of the system. Within the PLC, the drive data (current position and velocity) is cyclically read and can be accessed by an IPC via the OPC-UA Python client. In addition to the drive data, the IPC also detects the ball based on pictures from the camera. This data is used as input for the trained neural network and the output command (desired target points for drive positions) is then executed via the PLC.

Sensing: The visual observation of the Foosball environment is enabled by a *Logitech Brio* 4K camera with up to 60 fps which is mounted above the table. Furthermore, the drives are used as sensors to obtain the current lateral and angular positions and velocities of the motors.

Safety: Sick miniTwin4 safety light curtains are installed above the Foosball field to stop all rod movement whenever an object (e.g. a human hand) gets too close to the playing field and the moving rods.

3.2 System architecture

The architecture of the system and the information flow is illustrated in Fig. 2. The current lateral and angular position and velocity values of the drives are transferred to the PLC via Sercos and can be read by the Open Platform Communications Unified Architecture (OPC-UA) Python client within the Python script running on

the industrial personal computer (IPC). Based on the inputs given by the ball position and the drive values, the actions of the agent (setting the desired angular and lateral position) are calculated and sent back to the PLC. The movement of the industrial motors is handled by drive converters that are controlled via target waypoints transferred through Sercos. Given an overall desired (lateral + angular) position decided on by the agent, the drive controller calculates these waypoints as points on a jerk-limited trajectory. With the next generation of control units (De Blasi and Engels, 2020), there exists the possibility to simplify the architecture by using multiple software modules (e.g. model execution, ball detection, security and soft PLC functionality) on the same device.

3.3 The KIcker as a Markov Decision Process

The scope of this project is to prove the applicability of sim-to-real DRL in this manufacturing-like setting - in particular, the successful bridging of the reality gap. While from a pure Machine Learning perspective it is, of course, interesting to explore training all rods on the table (whether as a cooperative single- or multi-agent scenario), to prove that this sim-to-real transfer indeed works, it is enough to focus on a reduced, yet sufficiently complex subtask of the game. Therefore, to limit the involved project effort while maintaining all of the validity of the conclusions for the sim-to-real transfer for this setting, we reduce the scope and work to a subtask: training the striker rod (all three players) to score goals without an opposing goalie. More complex tasks do not add any further complexity to the sim-to-real transfer. In the following, we describe the selected subtask as an MDP.

Observation space: For the chosen subtask, the observations used for the MDP are six-dimensional. This includes the Cartesian ball coordinates (b_x, b_y) as well as the lateral position p, angular position d, linear velocity $\dot{p}$ and angular velocity $\dot{d}$ of the striker rod. While the drive variables (rod positions and velocities) can be read directly from the drives (via Sercos), the ball position can not. Rather, the ball position is obtained more indirectly, being extracted with a standard color-based detection algorithm (here sensitive on red) from each frame captured by the camera above the table. For illustration, such a sample camera frame is shown in Fig. 3. The center of the field with dimensions 120 cm x 68 cm is chosen as the origin of the coordinate system (the yellow cross). The unavailability of good information on the ball velocity from the camera (running at 60 fps) led us to entirely exclude it from the observations. There

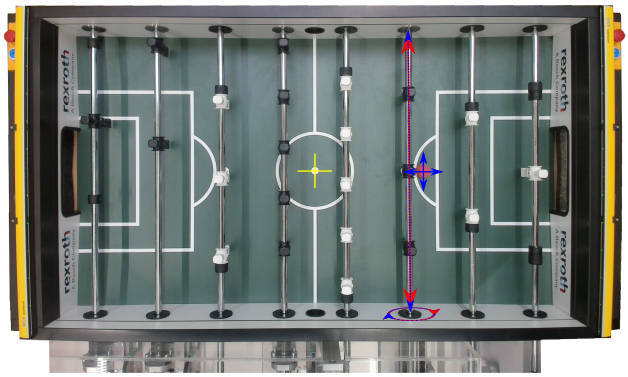

Fig. 3 Camera frame of playing field describing observations: The yellow cross indicates the origin of the coordinate system. The observations (in blue) include the ball position, the rotation angle and angular velocity of the striker rod and the translational/lateral position and velocity of the striker rod. The action space is two-dimensional and consists of setting the desired target lateral and angular positions for the striker rod.

are many possible options to extend the complexity of this system in the future, such as the expansion of the input space to include the ball velocities which can be easily calculated through the history of detected ball coordinates (would require the installation of a high-speed camera), or the visually detected positions of the human opponent's rods.

Control: The range of possible control actions for the KIcker MDP are 2D vectors with continuous entries corresponding to the lateral and angular target positions. Such vectors are sent to the PLC, which sets up the commands for the linear and the angular drives. The drives calculate the movement based on the maximum jerk, speed and position for each motor.

Rewards: The objective for the automated KIcker is to score goals. Since we want to mimic an easily-trainable manufacturing-like process, it is important that the chosen rewards are simple. This implies using a sparse reward setup which, in the case of the KIcker, means essentially only issuing rewards for scored goals. The details of the reward structure are covered in Sect. 5.1.

4 Simulation

Based on the construction and the interfaces of the KIcker described above, in this section we present the physically representative simulation environment that was developed. First, we point out the importance of a consistent simulation and specify parameters which were taken into account. Then, our modelling solution for the

jerk-limited position control of the KIcker's motors is presented.

4.1 Bridging the reality gap

As described in the introduction, simulations play a crucial role in real-world applications of RL since they allow for dramatically faster experience sampling compared to real life. However, simulation-based RL is known to suffer from the reality gap. The term describes the issues that arise from the unavoidable deviations of any simulation from the real world. If not addressed, the reality gap can nullify all benefits that are associated with simulation-based learning. There are multiple approaches to bridge this gap (Muratore et al., 2019). Based on the recent successes associated with domain randomization in sim-to-real DRL, we do two things to bridge it (Tobin et al., 2017; Andrychowicz et al., 2020) for the KIcker:

1. We ensure that the simulation in which we are conducting our training is as good a depiction of the real physical system as possible. In other words, we make sure that our digital twin is indeed a good twin. This means that it captures to a sufficient degree all the kinematical and dynamical properties of the KIcker relevant to the task which is to be learned. The measures taken to meet this demand are described in 4.2 and 4.3.
2. We apply domain randomization to those parts of the simulated physical system which are assumed to suffer from imprecision. Details for this are described in 5.2.

4.2 Passive physics of the Unity environment

The KIcker was simulated using version 2018.3.7f1 of the game engine Unity3D. Below, we list the (physical) parameters available in Unity through its Nvidia PhysX engine that were used to obtain a sufficiently accurate simulation of the KIcker.

Dimensions of the table and figurines: The dimensions of the table itself were available through the manufacturer *Ullrich* and a created CAD model that can be integrated into Unity. This was used to set and check the dimensions, providing a baseline representation of the physical system in Unity which we further customized. On the real table, the goal has a width of 20 cm. In our simulation, however, we set it to 18 cm in order to later have more margin for error for the trained policy. For the figurines, we used an openly available Unity asset which is not identical to the original Ullrich players. To achieve a sufficient correspondence, we adjusted the measurements (width, height, angle of the feet) of the asset to match the original ones.

Physic materials: Unity allows for the use of customized Physic Materials carrying the properties *bounciness, static friction* and *dynamic friction*, as well as the possibility to choose how the friction and bounciness of colliding objects are combined by the physics engine. We used four different kinds of materials: one each for the walls, the ball, the field and the figurines.

Drag: Unity allows one to set both linear drag and angular drag. Nontrivial values for these were used only for the ball - not for the rods/figurines. This is because the actuation for the rods is kinematically controlled as described in Sect. 4.3.

Masses: There are only two relevant types of dynamical objects on the table, namely the ball and the rod-figurine systems. This means that only the masses of these two must be taken into account.

Calibration of the parameters: As described in the introduction, our approach is based on real constraints as found in industry. With respect to parameter calibration, we assume that industry processes cannot be stopped for extensive measurements or even, more severely, are not yet available because the production line is still being constructed. Thus, we decided not to perform any measurements or other experiments on the KIcker itself. Instead, we adopted a more pragmatic approach for calibration, effectively arriving at the values for the Physic Materials, drags and ball mass by trial and error, sweeping the parameter-space for values which provide a realistic-looking rolling, sliding and collision behavior when compared by eye with another standard Foosball table. The initial calibration of the ball mass was aided by taking straight shots with the striker rod at the maximum shot-power (see Sect. 4.3), recording the velocity of the ball and comparing it to the velocities typically achieved by professional Foosball players. Due to the fact that we circumvent using the physics engine by controlling the rods kinematically by setting effective velocities at every time step (see Sect. 4.3), the masses of the rod-figurine systems themselves are not relevant. Rather, only the relative mass between the rod-figurine system and the ball is relevant - and this was already calibrated by calibrating the ball mass, as just described. This pragmatic approach results in inaccuracies that had to be addressed by randomization as described in Sect. 5.2.

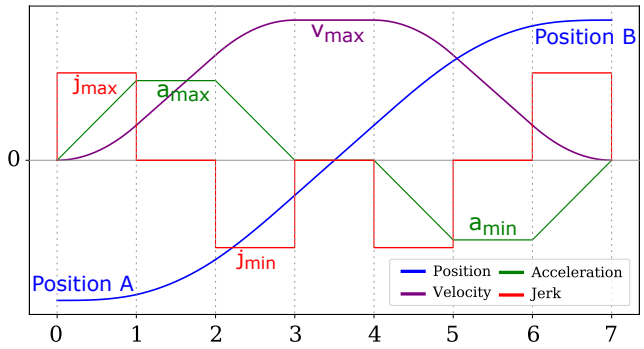

Fig. 4 Jerk-limited motion profile: Seven different phases of the trajectory from position A to position B

4.3 Control module for the motion of the rods

As described in Sect. 3.1, the rods are driven by industrial motors. They are controlled using commands for desired positions which are used to calculate and set the rods upon trajectories with jerk-limited dynamics. A typical trajectory can be seen in Fig. 4, showing motion from position A to position B along with the curves for velocity, acceleration and jerk. Once the parameters have been chosen for the desired limiting values of the velocity $(v_{\min}, v_{\max})$, acceleration $(a_{\min}, a_{\max})$ and jerk $(j_{\min}, j_{\max})$, under the constraint of time optimality the trajectory is uniquely defined. For more detailed information about trajectory-generation, we refer to the technical literature, e.g. Kröger (2010). As a result, for the KIcker one can specify a lateral and angular target and the rod will be set off on an uniquely defined trajectory towards that position until one chooses a different position causing the system to calculate and execute a new trajectory.

To have an adequate representation of the KIcker, we must mimic this mechanism of trajectory-generation and execution in our simulation. This is achieved in two steps. First, the entire trajectory is generated. Then, it is translated into actuation commands in Unity. Fig. 5 illustrates the process flow of the involved subsystems of the control module described in the following sections.

Trajectory-generation module for the motion of the rods: As formulated in the introduction, our project should follow the characteristics of industrial projects. Regarding trajectory generation, we fulfill this by using an external library instead of the software built into the motor controls. In this way, our solution does not rely on the availability of any vendor-dependent components or firmware. The external library (Beul and Behnke, 2017) enables fast (« 1 ms per axis per trajectory) and parallelizable generation of the full trajectories, which allows us to integrate it with Unity to rapidly receive and start following new trajectories when a new decision is made by the agent.

As the library is originally written in MATLAB, we compiled a .NET assembly which can be used as a trajectory-generator from within the C#-scripts that control our Unity environment. Whenever the agent decides on a new target position, the current positions, velocities and accelerations of the rods are sent to the assembly, the time-optimal trajectories are calculated and are sent back to Unity.

Kinematic control of the rods for tracking the trajectories: If a trajectory is given, it must be ensured that the rod actually follows it faithfully. Unsurprisingly, being first and foremost a game engine, Unity itself offers no built-in functionality to perform the kind of trajectory-following we require. Naively though, one could simply take the positions on the trajectory and, after every time step in Unity, set the position of the rod to the appropriate one. In fact, this will give us a perfectly good motion of the rod in position-space, but it will break the dynamics that we need when collisions occur - since, for this, one needs information on the rod's momentum. However, the principal idea of controlling purely the kinematics, entirely ignoring the mass and the complicated dynamics involved when applying forces and torque simplifies the control problem dramatically. Unity's physics engine uses velocities of Unity Rigidbodies as the principal dynamical quantity (forces are effectively implemented as incremental changes in the velocity). Given this, manually modifying the velocity of a Rigidbody in Unity lets us achieve incremental position adjustments while still allowing dynamical events to occur. Therefore, in order for our simulated rod-figurines system (one Rigidbody, just "rod" from now on) to follow a desired trajectory, in every Unity time step we manually set its velocity to the appropriate value for it to indeed follow. PhysX models the kinematics and dynamics in each time step by uniform motion. Thus, we must approximate the non-uniform trajectories with uniform increments. We did this such that, after each simulation step, the rod is at the right position as dictated by the trajectory. Note that this is a choice we make and this position-accuracy is obtained at the expense of other kinematical quantities. The approximation is obtained as follows: Let $X(t)$ be the position-space trajectory we wish to follow. At time t (a multiple of the time step Δt), we call the current position x_t^e of the rod and calculate the effective velocity v_{eff} required to reach $X(t+\Delta t)$ in the time Δt , i.e.

$$v_{\text{eff}} = \frac{X(t+\Delta t) - x_t^e}{\Delta t}. \tag{12}$$

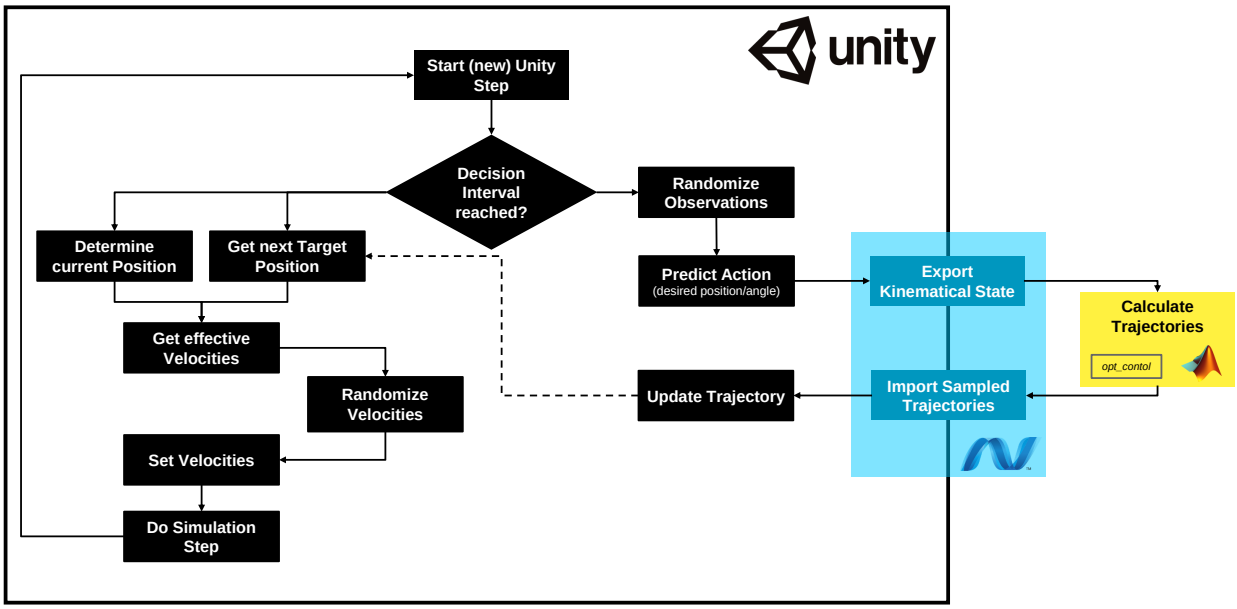

Fig. 5 Simulation Control Logic: In every Unity time step, the environment checks whether to demand a new decision from the agent. If the decision interval (which we can vary) is reached, the agent receives randomized observations based upon which it decides on new actions (this decision-loop reflects the MDP described in Sect. 2). The associated positions and angles are used to calculate the time optimal trajectories controlling the striker rod. Trajectory-following is realized by using the next target waypoint and the current rod position to calculate the effective velocity required to arrive at the target position in the next simulation time step. Before setting the velocity and doing the simulation step, it is randomized.

Note that we do not take the current position from the trajectory itself but instead call it from Unity (indicated by the label e for empirical). With this approach, we address inaccuracies like numerical errors, lags in the simulation etc. that could otherwise pile up. This is because the effective velocity is calculated in such a way as to catch up and bring us back to the trajectory-dictated position after the time Δt . Moreover, this approach allows for an easy randomization of the commanded position in every time step without having to care about the accumulation of errors.

So, the choice of this effective velocity we make here ensures maximum faithfulness in following the positions dictated by the trajectory while sacrificing some faithfulness in the velocity, as we deem position-accuracy to be more critical.

5 Training

This section covers details related to the training in the simulated environment. First, we recap and expand on the structure for the simulated KIcker as an MDP. Second, we introduce our approach of domain randomization and finally, we describe how we transferred the learned policies to the real KIcker.

5.1 Environment and MDP structure

The MDP underlying the KIcker system was introduced in Sect. 3.3. Including the aspects required for training, its features are the following:

Observation space: The observation input fed to the model is a 6D vector with continuous entries containing the lateral and angular rod positions and velocities, as well as the ball position. All are taken directly from the simulation.

Action space: The action output is a 2D vector with continuous entries corresponding to the lateral and angular target positions.

Done criterion: As is common, the MDP is split into episodes of a defined length. Rewards are accumulated based on the actions taken by the agent until an episode is done. The done criterion for an episode is fulfilled whenever MaxSteps is reached, a goal is scored, the ball crosses the middle line, or it comes to rest in an unreachable area.

Time intervals: The simulation time interval (i.e. the Unity time step and the step size of the MDP) is set to 1 ms while the decision interval is greater of equal to this, depending on the model scenario (see Sect. 5.2 for details).

Reward structure: The instantaneous reward that the agent receives from the environment is calculated after each decision step. A sparse reward structure is chosen because complex reward-engineering would be in conflict with our overall goal of evaluating DRL for manufacturing-like processes. Thus, only scoring a goal provides a meaningful reward of $r_{\text{goal}} = 5$. This is simple to set up (as is desirable for a manufacturing-like setting) but it is harder for the agent to learn from than if we were giving carefully crafted rewards that can guide it towards the right behavior. In addition to this, a time penalty of $r_{time} = -\frac{1}{\text{MaxSteps}}$ is given and an additional penalty of $r_{\text{backshot}} = -1$ if the ball crosses the middle line (meaning it has been shot backwards or rebounded and crossed into the other half of the table). So, the reward that the agent receives after every decision step is $r = r_{\text{goal}} + r_{\text{time}} + r_{\text{backshot}}$. This reward is accumulated during each episode.

5.2 Domain randomization

The applied randomization addresses the known and anticipated shortcomings of the simulation leading to improved generalization. Based on the related work described in the introduction, as many quantities as possible were randomized with relatively wide ranges for the applied noise. The actual randomization is applied on two levels:

1. *Episode level:* Multiple physical quantities were randomized from episode to episode. So, the values for these quantities vary between two episodes but are fixed within one episode. The randomized quantities, randomization type and the associated parameters can be seen in Tab. 5.2. For the quantities randomized with a normal distribution, the adjustment is performed by scaling the quantities with the random factor.
2. *Decision level:* In each time step, the observations as well as the actions are randomized around their current value. The observations are randomized by scaling the original value with a factor sampled from a normal distribution for each entry. The actions are randomized by adjusting the effective velocity v_{eff} (12), also with a factor sampled from a normal distribution. For both, the distribution has a mean of 1 and a standard deviation of 0.01. Randomizing the effective velocity results in a deviation from the desired trajectory in each time step. Nevertheless, since the effective velocities are always calculated on the actual position and not the trajectory positions, the deviations do not accumulate over time.

In addition to using domain randomization, we trained multiple models with varying decision intervals, i.e. the time interval after which the environment demands a new decision from the agent. This is done to deal with the relatively slow image processing from the camera. The selected decision intervals are 5, 10, 20 and 33 ms.

5.3 Network architecture and hyperparameters

For environments developed in Unity such as ours, the Unity ml-agents framework provides a ready-to-use implementation of PPO (Juliani et al., 2018). As it is common, the actor and the critic networks share all layers but the output. These feed-forward neural networks contain two hidden layers, each with 512 neurons, both fully connected. All neurons use a swish activation function. The chosen set of hyperparameters can be found in Tab. 2. Besides that, we constrained the training in order to meet our demand for a realistic manufacturing scenario as formulated in the introduction. We assume that flexible automation in industry requires training to be relatively quick. Therefore, regardless of training progress, we stopped the training after 48 hours in wall-clock time (running on a business laptop with an Intel Core i7-3720QM processor and 28 GB RAM, with no GPU acceleration).

5.4 Training execution

We deployed multiple models with different setups, such as variations of the decision interval or models trained with a single striker, and selected the most promising model: The model trained with all three strikers and a decision interval of 10 ms is used for the detailed evaluation. Fig. 6 shows the reward progress of the training. The performance shows clear progress in the 48 hours of training but it is also evident that no plateau was reached.

5.5 Transfer to the physical system

We transferred the trained models directly to the physical system without any additional back-and-forth adaptation or re-calibration. The applied domain randomization procedure was applied to allow the model to generalize and perform when transferred to the real environment. As we are interested in modelling the process as it would take place in an industry setting, having only one shot at transferring the model is a realistic scenario since strict time constraints for setting up automation are common and there is low tolerance for downtime of

Table 1 Randomization of physical quantities per episode

Quantity	Randomization type	Parameters
Rod Spawn Position	Uniform	low, high = -7.75cm,7.75cm
Ball Spawn Position	Uniform	low, high = -20cm,20cm
Additional Figurine Width	Normal	mean, std = 1, 0.1
Additional Friction	Normal	mean, std = 1, 0.1
Additional Ball Mass	Normal	mean, std = 1, 0.02
Additional Ball Size	Normal	mean, std = 1, 0.02

Table 2 Hyperparameters for policy training

Parameter	Value
#Layers	2
#Neurons per layer	512
ϵ	0.2
γ	0.95
λ	0.95
c_{v}	0.5
c_{ent}	0.01
Learning rate	$3 \cdot 10^{-4}$

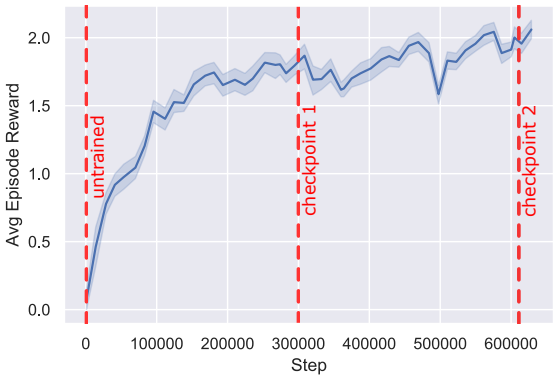

Fig. 6 Reward progress over training steps of the model with decision interval of 33 ms: Red dashed lines indicate the untrained and the first and second checkpoint of the model used for evaluation.

existing hardware, which complicates adaptation measurements.

6 Evaluation

In addition to the quantitative evaluation presented in this section, a video of the results is available on youtube (https://youtu.be/7HBgPtR1Uks).

To investigate the characteristics of the sim-to-real transfer for the KIcker, we defined a set of five evaluation shots and examined them using two quality metrics: first, the goal success rate and second, the goal scoring time as an indicator of goal quality (faster goals indicate a higher shot quality). The evaluation shots consist of investigating the quality measures for five different ball positions. These positions, reflecting both straight and diagonal shots, were chosen to lie symmetrically to the left and right of the center position, as well as at the center positions itself (i.e. lateral distances from the center of 0 cm, ±5 cm and ±10 cm). The distance of the ball to the midfield line was 26 cm - meaning that it is just slightly shifted towards the goal from under the striker rod. Due to inaccuracy in placing the ball on the KIcker for evaluation, the positions are rather position regions. We take this into account in the simulation by placing the ball randomly in a square of 5 mm around the specified position. The striker rod is always initialized in the center position with the feet pointing downwards. Note that this set of tested shots is only a small subset of the initializations seen in training, since there, both the ball and the striker rod are initialized along a large continuous range of positions (between -7.75 cm and 7.75 cm for the rod and -20 cm and 20 cm for the ball, see 5.2). To investigate the agent's behavior at different training stages, we evaluated all quantities for an untrained, half-trained and maximally trained agent. The different stages are highlighted in Fig. 6 as vertical red dashed lines. In the simulation, a total of 500 trials was run for each of the above-mentioned positions. On the real KIcker, the number of recorded trials was between 23 and 36.

6.1 Results

The results for the goal success rate in the simulation and with the model running on the real KIcker are compared in Tab. 3. In the simulation, with the untrained model the agent already has a high success rate for the trivial task where the ball is placed at the 0 cm position, directly in front of the center striker. This improves to 100 % success rate with further training progress. Even for the untrained model, this high success rate is not surprising since all that has to happen for a goal to be scored is for the center striker to even lightly tap the ball lying right in front of its feet. For the other positions, in the simulation we see that the success rate of the first checkpoint model is higher than the untrained one for all but one position. On the left side, the difference

Table 3 Goal success rate in the simulation ($n = 500$) and on the real KIcker ($n = 23...36$)

Simulation

stage	left (10 cm)	left (5 cm)	center	right (5 cm)	right (10 cm)
untrained	0.00	0.00	0.85	0.00	0.00
checkpoint 1	0.62	0.83	1.00	0.92	0.84
checkpoint 2	0.68	0.74	1.00	0.42	0.92

KIcker

stage	left (10 cm)	left (5 cm)	center	right (5 cm)	right (10 cm)
untrained	0.09	0.33	0.54	0.31	0.23
checkpoint 1	0.32	0.61	0.89	0.72	0.27
checkpoint 2	0.65	0.69	0.88	0.69	0.38

between the two trained checkpoint models is in the single digit percentage point range. The behavior for the right side is much more unexpected. Here, we find a significant decrease in performance for the 5 cm position between the trained checkpoint stages, while at the same time, the success rate for the outer position increases to 92 %. We interpret the behavior for the outer right position as an effect which reflects the agent's adaptation to specifics of the simulation, which are not present in reality (despite domain randomization). This interpretation is supported by the fact that we see a significantly lower success rate for this position on the real Kicker (see below). To further investigate what is causing the performance-drop at the 5 cm right position, we ran an additional evaluation in the simulation for balls uniformly initialized between the 4 cm right and 6 cm right positions. For the checkpoint 1 model, we observe a success rate of 0.95 - and 0.64 for the checkpoint 2 model. For the former, this reflects the rate originally achieved at the 5 cm position. The latter model, however, is more than twice as successful in this 4 cm to 6 cm range than it is at the 5 cm position alone, with a success rate more in the range of what we are seeing on the real KIcker (see below) and also consistent with the left 5 cm position. This indicates that this model is struggling specifically with the 5 cm position. As mentioned above, the five testing shots are only a few among the myriad of starting-configurations seen in training, but obviously, for testing, a selection of testing situations has to be made. We expect that effects such as this one should decrease more and more if training is continued. An additional observation is that the checkpoint 1 model generally plays the ball more softly than the checkpoint 2 model. We often see the agent tapping the ball lightly to send it off rolling into the goal (also reflected in the higher goal scoring times, see Fig. 7) which for some positions resulted in the observed higher success rate, since with this technique there is more margin for error. For the checkpoint 2 model, the shots are taken much more strongly, requiring higher accuracy.

For the real KIcker, we observe an increase in the success rate from the untrained to the first checkpoint stage for all positions. For the outer left and right position we also find a significant increase in accuracy between both trained checkpoint stages while for the inner positions, the performance is comparable between the two stages. The performance increase at the outer positions is noteworthy especially in comparison to the simulation where we do not observe this. We assume that the randomized training progress in the simulation led to a more robust behavior which proves more successful on the real system. For the half right position, we find no significant difference between the two checkpoints while the behavior on the outer right position improved by more than 10 percentage points. In comparison with the results of the simulation, this observation is interesting because the differences in the 5 cm right position between the trained models are not reflected in the real system. The same holds for the extreme success rate for the 10 cm right position which is not even reached halfway on the real physical system. However, the trend shows a clear improvement through training. The differences between the simulation and the real system at this position support our earlier argument that the observed accuracy in the simulation is caused by a too strong adaptation to the simulation specifics since this automatically leads to a performance loss on the real system. Comparing the left and the right side, we find the behavior for the 5 cm right position to be consistent with the 5 cm left position. This is as expected based on the results obtained for the 4..6 cm spawn range in the simulation.

By calculating the Pearson correlation coefficient r (resulting values of r close to 1 indicate a positive linear correlation, r = 0 is random and r close to -1 indicates a negative linear correlation) between the results of the simulation and the KIcker, for the left (5 cm, r = 0.95) and middle position (r = 1) we can conclude that there is a strong correlation. For the other positions, on the other hand, we found values between 0.76 and

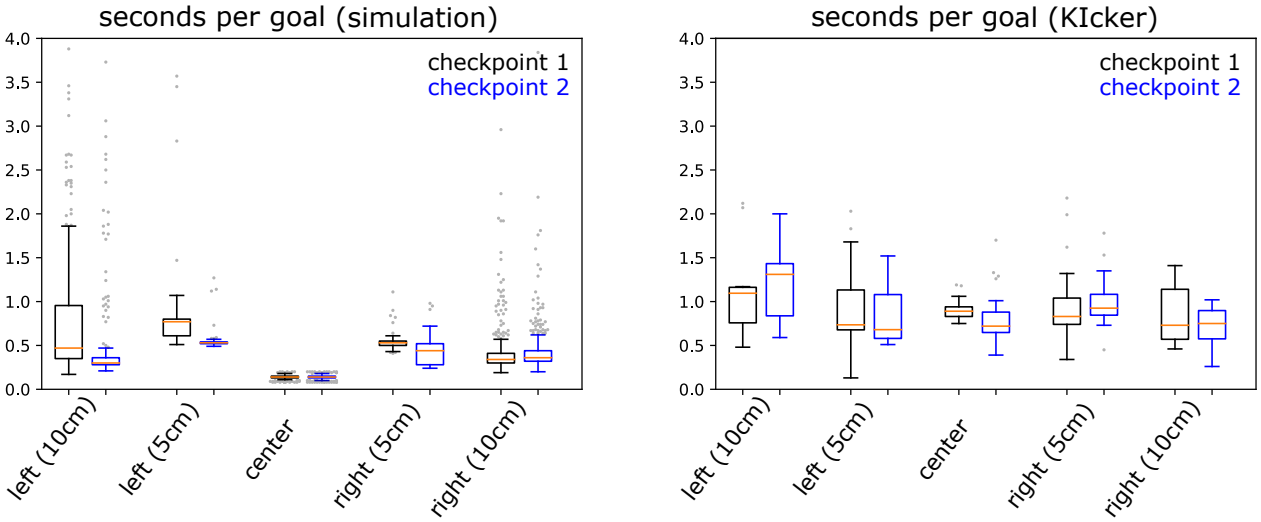

Fig. 7 Seconds per successfully scored goal: Lower values indicate a higher scoring quality

0.87, which indicate the desired positive correlation, but somewhat less strong. Even though $n = 3$ is low and the P-values correspondingly high (between 0.016 and 0.45), there is a clear indication that the simulation reflects the real physical system and is suitable as a training environment.

Since we defined a successful experiment by a goal being scored, there is no indicator of strongly or weakly scored goals. To investigate the quality of scored goals, we measured the time it took to score the goal in both the simulated and the real environment. While the values are available in the simulation, an image-based timer function is implemented for the real KIcker. The timer for this starts as soon as the camera detects the ball for the first time and a goal can be recorded as soon as the ball leaves the camera frame on the right side. The results are plotted in Fig. 7. As might be expected, the required time to score a goal is lower in the simulation than in reality. Furthermore, it is lower for the maximally trained stage than for the first checkpoint. It is noteworthy that the spread is much higher for the left outer position than it is for the right outer position. Again, this supports the assumption that the behavior in the simulation for the right outer position is affected by an adaptation to simulation details. Also, we observed that the large spread for the left outer position in the simulation is due to the fact that rebound shots are being scored, which take more time. In the real system, these are more seldomly scored which explains the lower spread of the scoring times there. For the KIcker, the scoring time is larger compared to the simulation across all positions and more or less comparable between the two training stages. On average, at all positions, the KIcker takes slightly less than a second to score a goal. This consistency can be interpreted as robust behavior due to randomization.

More qualitatively, it is remarkable that both the simulated system and the real KIcker are able to generalize to less familiar situations. While the ball was always initialized at rest during training, we also tested whether goals are scored for non-vanishing ball speeds, such as when the ball is rolled towards the striker rod from the front. This is a situation that is not explicitly set up in training, although it can occur due to the ball rebounding off the back wall and coming back towards the striker rod. We observed that the trained KIcker displays a reasonable behavior for non-vanishing ball velocities and could score goals for rolling balls from the corners (see video). Taking into consideration the absence of these situations in the training (at least explicitly) and that the observations do not even include the ball velocity, this is a gratifying result. If such situations are of interest in the future, a model could be trained to specifically take into account the ball velocity - for this purpose, however, a higher sampling rate for the camera is required.

In summary, it can be stated that the sim-to-real transfer was successful. We see that the behavior on the KIcker is robust due to domain randomization and a correlation between simulation results and real results is visible, as can be seen in the provided video. The existing deviations in behavior can be explained and it is to be expected that they can be reduced by an adapted randomization.

7 Conclusion and Outlook

We successfully applied simulation-based DRL to obtain the complex control of a manufacturing-like system with a sparse reward setup. This proves that DRL can be applied to physical systems built from industrial drive and control components by transferring policies learned in simulation to the real machine. Aided by domain randomization, training in a virtual environment is crucial due to the benefit of accelerated training speed and the desire for safe RL.

Our results demonstrate that state-of-the-art DRL algorithms are able to produce models which can successfully control manufacturing use cases. The performance obtained using sim-to-real DRL is already promising and we assume that including experiences from the real system will enhance it even further, following the instructions of previous work (Rusu et al., 2017). For us, this could mean further training the model on the KIcker, potentially also through playing against human players. Here, current research on real-time capable control architectures for the implementation of RL with industrial control components (Schmidt et al., 2020) is of interest. Even though the complexity of DLR was there not yet covered, this will be the logical next step to unleash the full potential of data-driven optimization in automation technology.

To further extend our exemplary automation problem, all rods on the table as well as state-tracking of the player rods could be included to create a fully automated Foosball table. This will make the learning task somewhat harder but the sim-to-real transfer demonstrated here remains solved.

We believe that our findings prove that further research in applied DRL for manufacturing systems is well-motivated as it holds great potential for the manufacturing of the future. It is our hope that this contribution will serve to inspire further investigations into this exciting and promising area of research.

Appendix - Derivation of a simple policy gradient loss

Starting from equation (2), we introduce a short notation for the cumulative discounted reward obtained for a given trajectory: $R(\tau) = \sum_{k=0}^{\infty} \gamma^k r(s_k, a_k)$. This allows us to easily derive a basic policy gradient algorithm using the following loss function which we want to optimize:

$$L(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} R(\tau). \tag{13}$$

To expand the expectation value, one integrates the cumulative returns over all trajectories weighted by the probability for that trajectory:

$$L(\theta) = \int_\tau p_\theta(\tau) R(\tau) d\tau. \tag{14}$$

Since only the trajectory-probability depends on θ, we can take the derivative of $J(\theta)$ with respect to θ. By using a simple log-derivative identity, this can be reformulated:

$$\begin{aligned} \nabla_\theta L(\theta) &= \int_\tau \nabla_\theta p_\theta(\tau) R(\tau) d\tau \\ &= \int_\tau p_\theta(\tau) \nabla_\theta \log\, p_\theta(\tau) R(\tau) d\tau \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \nabla_\theta \log\, p_\theta(\tau) R(\tau) \end{aligned} \tag{15}$$

The term $\nabla_\theta \log\, p_\theta(\tau)$ is evaluated by decomposing $p_\theta(\tau)$:

$$\begin{aligned} \nabla_\theta \log\, p_\theta(\tau) =& \nabla_\theta \log \left(s_0 \sum_{k=0}^{\infty} T(s_{k+1}|s_k, a_k) \pi(a_k|s_k) \right) \\ =& \nabla_\theta s_0 + \sum_{k=0}^{\infty} \nabla_\theta T(s_{k+1}|s_k, a_k) + \\ & \sum_{k=0}^{\infty} \nabla_\theta \pi(a_k|s_k) \\ =& \sum_{k=0}^{\infty} \nabla_\theta \pi(a_k|s_k) \end{aligned} \tag{16}$$

So, overall the gradient of the loss function can be formulated as:

$$\nabla_\theta L(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \left(\sum_{k=0}^{\infty} \nabla_\theta \pi(a_k|s_k) \right) \left(\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) \right) \tag{17}$$

This expression is very similar to a standard maximum likelihood gradient used in supervised learning. The only difference is the last term coming from the cumulative rewards. It acts as a weight term for the gradient likelihood terms. With this, policy updates

$$\theta' \leftarrow \theta + \alpha \nabla_\theta L(\theta) \tag{18}$$

can be performed iteratively.

Declarations

Ethical approval

Not applicable.

Consent to participate

Not applicable.

Consent to publish

Not applicable.

Authors contributions

SDB, SK, AM, RR, FS initialized and conceived the project. SK, AM, RR, TZ designed the kinematic control simulation with environment. RR ran the training of the agents. SDB, SK, RR conceived and planned the experiments. SK prepared the models for deployment. SDB deployed the model on the system, carried out the experiments and analyzed the data. SDB, SK, RR discussed the results and wrote the manuscript.

Funding

No funding was received for conducting this study.

Competing interests

There are no conflicts of interest.

Availability of data and materials

Not applicable.

References

Andrychowicz OM, Baker B, Chociej M, Józefowicz R, McGrew B, Pachocki J, Petron A, Plappert M, Powell G, Ray A, Schneider J, Sidor S, Tobin J, Welinder P, Weng L, Zaremba W (2020) Learning dexterous in-hand manipulation. The International Journal of Robotics Research 39(1):3–20, DOI 10.1177/0278364919887447

van Baar J, Sullivan A, Cordorel R, Jha D, Romeres D, Nikovski D (2019) Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. In: 2019 International Conference on Robotics and Automation (ICRA), IEEE, pp 6001–6007

Bambach S, Lee S (2012) Real-time foosball game state tracking. Tech. rep., School of Informatics and ComputingIndiana University

Beul M, Behnke S (2017) Fast full state trajectory generation for multirotors. In: Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, DOI 10.1109/ICUAS.2017.7991304, URL https://github.com/AIS-Bonn/opt_control

Bousmalis K, Irpan A, Wohlhart P, Bai Y, Kelcey M, Kalakrishnan M, Downs L, Ibarz J, Pastor P, Konolige K, et al. (2018) Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In: 2018 IEEE international conference on robotics and automation (ICRA), IEEE, pp 4243–4250

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv preprint arXiv:160601540

Campbell B, Deo P, Hanganu C, Muhlstein L (2009) Reinforcement learning in graphical games. Tech. rep., Rutgers University, New Jersey Governors School of Engineering and Technology

Chebotar Y, Hausman K, Zhang M, Sukhatme G, Schaal S, Levine S (2017) Combining model-based and model-free updates for trajectory-centric reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning, JMLR. org, vol 70, pp 703–711

Collins J, Howard D, Leitner J (2019) Quantifying the reality gap in robotic manipulation tasks. In: 2019 International Conference on Robotics and Automation (ICRA), IEEE, pp 6706–6712

De Blasi S, Engels E (2020) Next generation control units simplifying industrial machine learning. In: 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), IEEE, pp 468–473

Grondman I, Busoniu L, Lopes GAD, Babuska R (2012) A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42(6):1291–1307, DOI 10.1109/TSMCC.2012.2218595

Hornung A, Zhang D (2008) On-line detection of rule violations in table soccer. In: Annual Conference on Artificial Intelligence, Springer, pp 217–224

Humayoo M, Cheng X (2018) Relative importance sampling for off-policy actor-critic in deep reinforcement learning. CoRR abs/1810.12558, URL http://arxiv.org/abs/1810.12558, 1810.12558

Janssen R, de Best J, van de Molengraft R (2009) Real-time ball tracking in a semi-automated foosball table. In: Robot Soccer World Cup, Springer, pp 128–139

Janssen R, Verrijt M, de Best J, van de Molengraft R (2012) Ball localization and tracking in a highly dynamic table soccer environment. Mechatronics 22(4):503–514

Juliani A, Berges V, Vckay E, Gao Y, Henry H, Mattar M, Lange D (2018) Unity: A general platform for intelligent agents. CoRR URL http://arxiv.org/abs/1809.02627, 1809.02627

Kalashnikov D, Irpan A, Pastor P, Ibarz J, Herzog A, Jang E, Quillen D, Holly E, Kalakrishnan M, Van-

houcke V, Levine S (2018) QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. arXiv e-prints arXiv:1806.10293, 1806.10293

Kober J, Bagnell JA, Peters J (2013) Reinforcement learning in robotics: A survey. The International Journal of Robotics Research 32(11):1238–1274

Kröger T (2010) On-Line Trajectory Generation in Robotic Systems - Basic Concepts for Instantaneous Reactions to Unforeseen (Sensor) Events, vol 58. Springer, DOI 10.1007/978-3-642-05175-3

Lee Y, Hu ES, Yang Z, Yin A, Lim JJ (2019) IKEA Furniture Assembly Environment for Long-Horizon Complex Manipulation Tasks. arXiv e-prints arXiv:1911.07246, 1911.07246

Luo J, Solowjow E, Wen C, Ojea JA, Agogino AM, Tamar A, Abbeel P (2019) Reinforcement learning on variable impedance controller for high-precision robotic assembly. In: 2019 International Conference on Robotics and Automation (ICRA), IEEE, pp 3080–3087

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529

Muratore F, Gienger M, Peters J (2019) Assessing transferability from simulation to reality for reinforcement learning. IEEE Transactions on Pattern Analysis and Machine Intelligence DOI 10.1109/TPAMI.2019.2952353

Padon O, Zuckerman O, Kindergarten L (2003) Development of robotic foosball as a versatile platform for robotics research and contests. Tech. rep., Lifelong Kindergarten Media Lab, MIT

Peng XB, Andrychowicz M, Zaremba W, Abbeel P (2018) Sim-to-real transfer of robotic control with dynamics randomization. In: 2018 IEEE international conference on robotics and automation (ICRA), IEEE, pp 1–8

Rusu AA, Večerík M, Rothörl T, Heess N, Pascanu R, Hadsell R (2017) Sim-to-real robot learning from pixels with progressive nets. In: Conference on Robot Learning, pp 262–270

Schmidt A, Schellroth F, Riedel O (2020) Control architecture for embedding reinforcement learning frameworks on industrial control hardware. In: Proceedings of the 3rd International Conference on Applications of Intelligent Systems, pp 1–6

Schoettler G, Nair A, Luo J, Bahl S, Aparicio Ojea J, Solowjow E, Levine S (2019) Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards. arXiv e-prints arXiv:1906.05841, 1906.05841

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. CoRR URL http://arxiv.org/abs/1707.06347, 1707.06347

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484

Sutton R, Mcallester D, Singh S, Mansour Y (2000) Policy gradient methods for reinforcement learning with function approximation. Adv Neural Inf Process Syst 12

Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA

Taitler A, Shimkin N (2017) Learning control for air hockey striking using deep reinforcement learning. In: 2017 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), IEEE, pp 22–27

Tan J, Zhang T, Coumans E, Iscen A, Bai Y, Hafner D, Bohez S, Vanhoucke V (2018) Sim-to-real: Learning agile locomotion for quadruped robots. arXiv preprint arXiv:180410332 URL https://arxiv.org/pdf/1804.10332.pdf

Tobin J, Fong R, Ray A, Schneider J, Zaremba W, Abbeel P (2017) Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 23–30

Weichert D, Link P, Stoll A, Rüping S, Ihlenfeldt S, Wrobel S (2019) A review of machine learning for the optimization of production processes. The International Journal of Advanced Manufacturing Technology 104(5-8):1889–1902

Weigel T (2005) Kiro - a table soccer robot ready for the market. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp 4266–4271, DOI 10.1109/ROBOT.2005.1570776

Weigel T, Nebel B (2008) Tischfußball: Mensch versus computer. Informatik-Spektrum 31(4):323, DOI 10.1007/s00287-008-0255-z

Weigel T, Rechert K, Nebel B (2005) Behavior recognition and opponent modeling for adaptive table soccer playing. In: Furbach U (ed) KI 2005: Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 335–350

Yin S, Li X, Gao H, Kaynak O (2014) Data-based techniques focused on modern industry: An overview. IEEE Transactions on Industrial Electron-

ics 62(1):657–667
Zhang D, Nebel B (2007a) Learning a table soccer robot a new action sequence by observing and imitating. In: AIIDE, pp 61–67
Zhang D, Nebel B (2007b) Recording and segmenting table soccer games–initial results. In: Proceedings of the 1st International Symposium on Skill Science, pp 61–67
Zhao Y, Xiong R, Zhang Y (2017) Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm. Journal of Intelligent & Robotic Systems 87(3):407–423, DOI 10.1007/s10846-017-0515-8
Zhong RY, Xu X, Klotz E, Newman ST (2017) Intelligent manufacturing in the context of industry 4.0: a review. Engineering 3(5):616–630

Stefano De Blasi received his M.Eng. degree in Electrical Engineering and Information Technology with focus on Computational Neuroscience in 2018 at UAS Aschaffenburg. His current research as PhD student focuses on safe-learning machines for automation industry in cooperation with Bosch Rexroth and the PZAI HAW Hessen. This leads to strong interests in the field of Reinforcement Learning, optimization theory and industrial control architectures.

Sebastian Klöser holds a M.Sc. degree in theoretical Physics as well as a M.Ed. for higher education from the University of Hamburg. He is a solution principal at DXC Technology with a focus on applied Machine Learning. Within that role he is the head of DXC Technology's initiative for applied Deep Reinforcement Learning. He is a researching member and co-founder of the haven research association for applications of artificial intelligence (haven-association.org).

Arne Müller received his M.Sc. in Physics in 2018 at the University of Hamburg. After receiving his degree, he started to work at DXC Technology as a Technical Consultant working on Reinforcement Learning problems and containerization technologies. Current topics he is interested in are container and orchestration technologies, as well as MLOps. He is a researching member and co-founder of the haven research association for applications of artificial intelligence.

Robin Reuben Holds an M.Sc. degree in Physics from the University of Hamburg. He works on various Machine Learning topics and is part of DXC Technology's initiative for applied Deep Reinforcement Learning. He is a researching member and co-founder of the haven research association for applications of artificial intelligence.

Fabian Sturm received his M.Sc. degree in Industrial Engineering with focus on Electrical Engineering and Information Technology in 2020 at UAS Darmstadt. As a PhD student, he is currently researching the recognition of human action in industrial environments in cooperation with Bosch Rexroth and the PZAI HAW Hessen. His research interests include computer vision and edge-to-cloud infrastructures.

Timo Zerrer received his B.Sc. in Business Information Systems in 2019 at Baden-Württemberg Cooperative State University (DHBW) where he is currently pursuing the M.Sc. in Computer Science. In his role at DXC Technology within the innovation team he works on applications of Machine Learning technology. His current research interest lies in the field of applied Reinforcement Learning. He is a researching member of the haven research association for applications of artificial intelligence.