Reinforcement Learning of Action Sequences in Table Football

Jonathan Croenen[1 [0000−0002−1843−972X]], Jens Bürger[1,2 [0000−0001−8900−9666]] and Wannes Meert[1,2 [0000−0001−9560−3872]]

[1] KU Leuven, B-3000 Leuven, Belgium
[2] Leuven.AI - KU Leuven Institute for AI, B-3000 Leuven, Belgium

Abstract. This work investigates reinforcement learning (RL) for robotic control in a dynamic, stochastic, continuous, and competitive environment. Such environments require robotic agents to construct more complex representations of the environment's dynamics as well as their own actions within it, as compared to static, discrete or deterministic environments. Using a simulated table football environment, we explore the applicability of discrete control for continuous motion, curriculum learning for effectively mastering increasingly complex action sequences, and reward shaping to promote domain-specific behaviours. We show that state-of-the-art RL algorithms can accomplish multi-step lateral and forward passing sequences as well as decision-making to outperform a competitive dynamic rule-based opponent. Investigating the application of AI in a well-described game environment (akin to chess, Go, and RoboCup), and surpassing prior attempts in this particular application domain, this work contributes to research on intelligent robotic agents and their ability to handle tasks of increasing complexities.

Keywords: Reinforcement learning · Robotic control · Curriculum learning · Reward shaping

1 Introduction

Many robotics applications require interaction with physical, continuous environments that contain dynamic elements and inherent stochasticity due to, e.g., other agents, uncertainties, or noise. These environments demand that agents possess strong representational capabilities to accurately model the complex dynamics of their own actions, as well as the effects of external forces, objects, and other actors beyond their control. In such settings, agents must not only adapt to deterministic factors or sparse feedback from the environment, but also account for the unpredictability introduced by noise, partial observability, and real-world variability [7]. Conquering these challenges is essential for the development of robust robotic systems capable of performing intricate tasks in manufacturing, autonomous navigation, and human-robot collaboration, where precision and adaptability are critical.

Game environments are often selected to advance education and research in artificial intelligence (AI) and robotics (e.g., chess, Go, football). Game rules

provide clear descriptions of the complexity, while also providing a bounded domain that avoids many complexities inherent in real-world scenarios (e.g., noise, unexpected situations, etc.). For example, Mnih et al. have shown how state-of-the-art reinforcement learning (RL) algorithms can perform human-level control in Atari games [8]. However, achieving human-level performance in a real-world robotic game environment where intelligent agents act via a range of sensors and actuators is still far more complex (e.g., RoboCup). Robotic table football (foosball) presents a rich, dynamic, and stochastic environment requiring agents to integrate continuous control and strategic decision-making in a competitive setting. At the same time, foosball avoids complex human-like bi-pedal locomotion or self-localization as in the RoboCup challenge. First, robotic foosball projects investigated rule-based systems such as [16,15], or had a major focus on control and automation [4]. First work involving RL to address challenges within foosball has been shown in [2,11]. Our work is positioned to further advance education and research in this context: achieving more adaptive and varied control and decision-making via the use of RL. We take professional human game-play as a reference for learning three common action sequences used in attacking strategy – tick-tacking, dribbling, and forward-passing – each requiring fine motor control and some decision-making under uncertainty. With these tasks, we demonstrate the following contributions:

- A first contribution is the implementation of a foosball simulator that will serve as a platform for a wide range of RL and robotics challenges.
- With other robotic foosball projects taking more of a control-approach ([4]), a second contribution is the demonstration that state-of-the-art RL algorithms are capable of effectively learning a variety of realistic action sequences known from high-level human play, incl. detailed motion control and decision-making under uncertainty. The set and complexity of the action sequences demonstrated in this work exceeds what has been shown in previous work using RL, while focusing on the unique challenges in attacking strategy.
- Lastly, utilizing curriculum learning and reward shaping we demonstrate a generalizable approach that can be used to learn a wide variety of domain-specific action sequences of varying complexity similar to high-level human game play.

This paper is organized as follows: in section 2 we briefly review related work, with a focus on robotic foosball applications. Following, in section 3 we detail the simulation environment used within this work. Efforts to construct such an environment will allow further in-depth research and education on RL in intelligent, robotic applications. Subsequently, we define action and state spaces as well as the domain-specific tasks to be learned in section 4. The precise experiments and results will be presented in section 5. In section 6 we conclude by contextualizing the obtained results with respect to the related literature and what we consider the next set of challenges for RL in such an environment.

2 Related Work

In this work we will apply the Proximal Policy Optimization (PPO) algorithm [12], which has already been demonstrated on a wide range of relevant tasks, ranging from continuous control to long-term planning [9,5,10,1].

KiRo [16,15] has been the first foosball robot with a focus on the application of AI methods. It made use of a finite set of predefined and manually programmed action sequences that could be applied to various situations with names such as `KickBall`, `MoveKickBall`, `BlockBall`, and `BlockAtPos`. These action sequences are then utilized and directed through a high-level strategy built as a decision tree. From the perspective of achieving a human-type playing style, the limitation of this architecture was the purely reactive play not foreseeing to gain control over the ball and to take decisions as a result of the opponent's behavior. The Babyfoot project from EPFL is a multiyear project powered by student efforts from which reports are available in [4]. It has a focus on implementing advanced automation and control techniques and has achieved the implementation of similar action sequences as will be investigated in this work, yet via implementing explicit calculations on shot timings and positions, without the use of RL [3]. First steps in the explicit use of reinforcement learning to play foosball have recently been shown by [2,11]. Both works make use of the same physical, robotic foosball table, and corresponding simulators, and focus mostly on the challenges associated with the sim-to-real gap, of transferring an agent learned in a simulated environment to the real world. Another open question left by those works, is which control scheme, or action type is best suited for the foosball setting. The proposed system in [2] opted for actions based on continuous targets, while [11] used discrete choices at each time step, an approach that has found success in other continuous control problems [9].

In conclusion, earlier work has achieved complex multi-action sequences inspired by professional human techniques, though by relying on high reactivity and intricate manual modeling of the system's dynamics. On the other hand, attempts using learning algorithms have been limited in task complexity, with a focus on other challenges. As will be shown, our work surpasses these latter attempts by tackling complex and adversarial action sequences with RL, exploring curriculum learning and reward shaping as tools in achieving that goal. As an intermediate step, we also share our findings from an experiment making an explicit comparison between a discrete and continuous action space inspired by those used in prior works.

3 Foosball Simulation Environment

The foosball simulation environment developed for this project is built within the Unity engine [6], leveraging its 3D visualization capabilities, physics engine, and the ML-Agents package with its RL support [13].

Table Model As a foosball table model, we are using a 3D model of the Tornado T3000. This table is one of the main competition tables and ensures that our

experiments are designed in accordance to the dimensions of real tables and in line with action sequences commonly executed on such tables. To increase simulation performance, we modified the model to reduce poly-counts, removing unnecessary external features, and removing redundant vertices in flat or near flat surfaces using Blender modifiers. A render of the complete table can be seen in Figure 1.

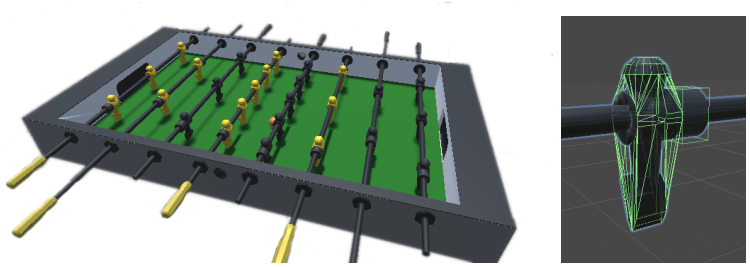


Fig. 1: Render of the simulated foosball table (left) and example of the physics collider applied to the player figures (right).

Physics Colliders Each of the elements relevant in simulating physical interactions is fitted with a collision shape. For performance reasons all primitive shapes and elements that are not involved in ball collisions are approximated with primitive collider shapes. This includes the frame (a group of box colliders), the table- and limit bumpers, and the ball. The player figures are fitted with a mesh collider to provide increased accuracy in collision calculations. An example wireframe render of the figure colliders can be seen in Figure 1. Finally, the colliders are separated into collision layers to eliminate redundant collision checks, improving physics performance. Only the table and limit bumpers, the ball and frame, and the ball and players will be considered for collision checks.

Rigid Bodies & Joints Movable physics objects in Unity require a rigid body component, hence the rods and ball are configured with one. To retain the rods position in the frame subject to forces from ball contacts and gravity, the rods translation and rotation are constrained using `configurable joints`. They are setup to only allow translation along and rotation around the lateral axis, whereby the limit bumpers limit the range of translation.

Programmatic Control The simulation exposes various information sources like coordinates of the ball, translation and rotation of the rods, current possession of the ball, etc. for use as RL observations or the implementation of reward functions. Additionally, the simulation can be controlled to set the initial states

and to control the ball and rod motion through forces. The motors controlling the rods are modelled naïvely as reversible motors that generate an acceleration or torque with a power level dial, indicating percentage of max power. The RL agents and environments that make use of these tools are implemented using the ML-Agents framework [13].

4 Target Tasks, States, Actions, and Rewards

This section details the setup and tasks we will execute in the foosball environment. It starts out by defining the components that are common between the action sequences in section 4.1 and follows with the explanation of each sequence – tick-tacking, dribbling and forward passing – and their unique configuration in sections 4.2 to 4.4.

4.1 Shared State and Action Space

States A state s_t is built up from all positional information p_t of the ball and the rods that are relevant in performing the action sequence. This includes the 2D coordinates of the ball on the field, and the lateral translation and rotation of each rod. To adhere to the Markov property, the data is combined over the three previous time steps so that $s_t = [p_{t-2}; p_{t-1}; p_t]$. This allows the calculation of velocity and acceleration estimates and distinguishes each state, which would otherwise only be a still snapshot of the simulation with no sense of motion.

We refrain from using the velocity information available in the physics engine, to simplify estimating states in a future physical twin system. This way a physical system would only have to estimate the positions of the ball and rods, likely from motor information and visual inputs, without the need to also estimate motion. Motion estimation is then incorporated in the neural network architecture and training of the model.

Actions We use one of two different action spaces for each experiment. Most use a fully discrete action space for its simplicity and efficiency in training as recognized in other robotics related works such as [9,11], and validate its applicability in increasingly complex action sequences. An action consists of two discrete, ternary outputs l_t and $r_t \in \{-1, 0, 1\}$, corresponding to lateral translation and rotation respectively. Each action value corresponds with either move inward/rotate counterclockwise, hold, or move outward/rotate clockwise. The hold action is a special action that resists motion, holding its current position even with outside forces applied. The continuous action space outputs target positions and rotations, and uses a proportional–integral–derivative (PID) motor controller, a common tool in industrial control systems. In this case it is used to determine the necessary motor inputs to reach a given target position from its current state. Formally, the agent would output two continuous action values $l_t \in [-1, 1]$ and $r_t \in [0, 1]$ at each time step, corresponding to the normalized ranges for rod translation and rotation.

4.2 Tick-tacking Task

Tick-tacking entails tapping the ball back and forth in between two adjacent player figures on the same rod, where each contact is considered as one successful tick-tack. An example sequence is illustrated in Figure 2. A trained agent should perform the sequence between any pair of figures on the midfield rod. The initial conditions are randomized with the ball being placed anywhere along the rod and the rod starting from a random position and rotation. We supply a reward of 1 for each successfully performed tick-tack i.e. each contact with alternating figures. Whenever the maximum of 500 decision steps are made, the episode is interrupted and whenever the ball leaves the reach of the midfield bar, the episode is ended.

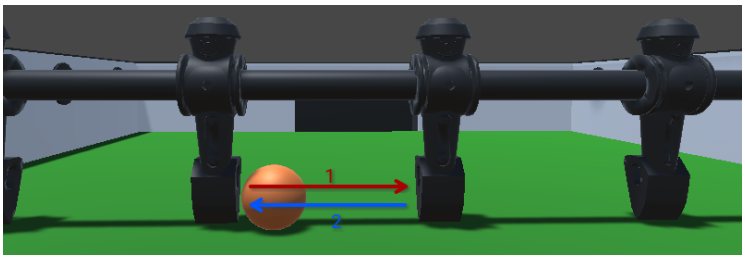


Fig. 2: An example sequence of two tick-tacks. Arrows indicate the motion of the ball between the player figures.

4.3 Dribbling Task

Dribbling is an extension of tick-tacking with the introduction of longer lateral passes along the rod, skipping past at least one figure. The agents are expected to perform tick-tacks, followed by dribbles to a new target pair of figures specified as explicit input. This choice is made to shift the focus more on the control side of the problem and remove the decision-making requirements (e.g. automatically decide when and where to dribble based on opponent movement). This simplifies the problem here and has an interesting implication in that an outside source can direct the behavior of the agent. This is useful for e.g. demo purposes, but could also have interesting implications for hierarchical RL methods, where a high-level policy could control the behavior of the lower level dribbling policy for example.

An example showing a possible sequence of ball contacts is shown in fig. 3. Dribbling demonstrates learning the integration and combination of two different action sequences. Similar to the tick-tacking sequence, a reward of 1 is given for each successful tick-tack performed, though only for those performed between the target pair of figures in this case. This means that if the agent does not perform a dribble (i.e. long lateral pass) to the target pair, the agent can not

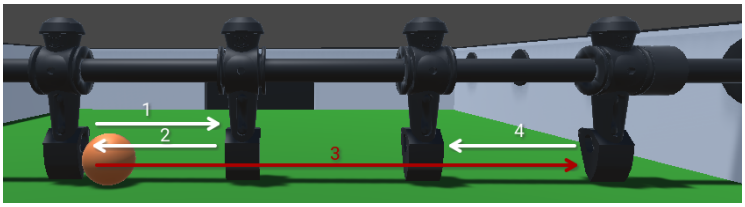


Fig. 3: An example sequence of two tick-tacks (1, 2), followed by a dribble from pair 2 to 4 (3), concluded with one more tick-tack (4). Arrows indicate the motion of the ball between the player figures, where white indicates a tick-tack and red a dribble.

gain any more tick-tacking rewards. A reward of 3 is given for the completion of a requested dribble. An episode is interrupted whenever a maximum of 800 decision steps have been made, or ended whenever the ball goes outside the reach of the rod.

4.4 Forward Passing Task

Forward passing involves a passer, (i.e. the black midfield rod) an opponent, (i.e. the yellow midfield rod) and the receiver (i.e. the black front line). An agent is expected to shoot the ball forward from its initial position, through the opponent line, catching the ball with the receiver. A catch is successful after maintaining the ball in possession for 100 simulation steps. The ball is placed in a consistent spot at the right flank in front of the passer line, which also starts pinned to the right side. Starting a passing sequence from a known setup is common in human play and allowed for faster convergence during training. Agents should be able to perform this sequence against a range of manually modeled, rule-based opponent policies, avoiding the complexities of multi-agent RL, which are outside the scope of this work. The explanation of the policies can be found below in the following section. Finally, at initialization, the receiver is positioned randomly, while the opponent is initialized based on its active policy. An example sequence of movements is shown in fig. 4.

For all action sequences we track an episode's progression with a finite state machine (FSM) that operates on high-level concepts relevant to the task at hand (e.g. ball blocked, ball caught, lost possession). We then implement our reward function on top of these FSMs. This has the effect of greatly simplifying the implementations by consolidating lengthy and tedious conditional logic into a common structure, and hiding raw simulation variables, like coordinates, behind high-level concepts. An example visualization of such an FSM can be seen in fig. 5. It shows all possible turnouts of an episode and the rewards or penalties that are given for reaching certain states. Any outcome that results in the ball being lost, indicated by a dashed state, or failing to complete the pass in 800 decision steps ends the episode.

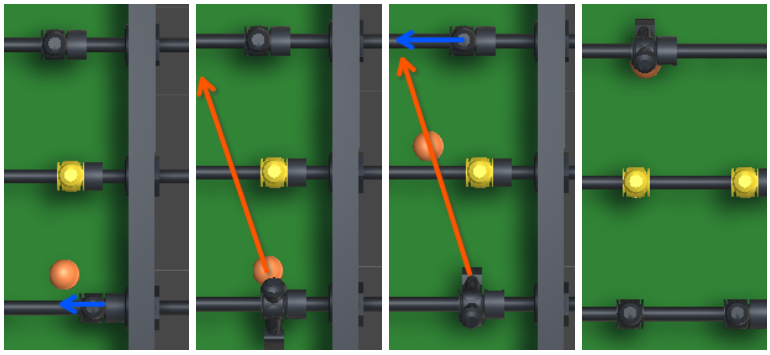


Fig. 4: An example of a possible forward pass sequence. (1) shows the passer moving into passing position, (2) depicts the pass, aimed at a gap in the opponent line, (3) shows the receiver moving to intercept, and finally (4) shows a successful catch and hold on the ball. Blue and orange arrows indicate rod and ball movement, respectively.

Opponent Policies The opponents are rule-based, where each policy's rules determine target positions at each simulation step. The opponent's motors are then controlled by a PID controller that tries to move the bar to the target. The following strategies were developed to train and benchmark against:

- **Oscillating:** The rod is initialized at a random position and moves according to a sine function with a randomly chosen frequency in $[2, 8]$ at the start of each episode, across the entire width of the field.
- **Tracking:** The rod is initialized with a random position and tracks the lateral position of the ball on the field. It tries to be in front of the ball with the closest figure at all times but is not fast enough to catch fast well placed diagonal passes.
- **Biased Random:** The bar initializes at a random position and moves to randomly sampled positions. The samples are taken from a distribution built from two Gaussians with a uniform random mean in $[-1, 1]$ (i.e. the full range of motion) and uniform random stddev. in $[0.1, 0.4]$. This opponent is made to be exploitable by an agent that could adapt to learn of the bias at inference time.
- **Deceptive Strategy:** This opponent starts out disguised as the oscillating opponent behaving similarly, in an attempt to trick an agent in expecting it to be predictable in the same manner. However, once a pass is made, the agent transitions to the tracking strategy attempting to intercept the ball. This is a common strategy in professional play, where players intentionally leave gaps to bait for a shot or pass they are actually expecting.

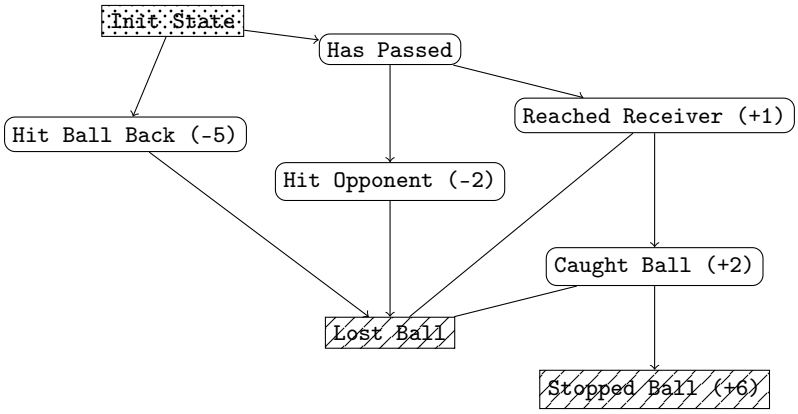


Fig. 5: Visual representation of the FSM used to track the high-level state of a forward passing episode in training. Values in parentheses represent rewards for entering a state, the dotted state is the initial state at the start of an episode, and the dashed states are terminal states.

5 Experiments and Results

5.1 Discrete Action Space

To validate the effectiveness of discrete action sequences in a continuous environment (in contrast to previous work using continuous actions [2], and similar to [11]), we train two agents for the tick-tacking task. One agent takes discrete actions and another continuous PID-based actions. We trained both agents for 15×10^6 decision steps equipped with a neural network of 4 layers, each with 256 neurons. This shape was determined through limited search to find a near optimal size for the continuous action space agent. The hyperparameters of the ML-Agents' PPO implementation are configured according to the Unity recommended ranges [14]. We then compare the policies by executing them for approximately 500 episodes (totalling 5×10^4 simulation steps), with each episode starting from a random ball position, and recording the distribution of total tick-tacks at the end of each episode.

As shown in fig. 6, our discrete action space is more consistent revealed by the greater density around its highest scoring episodes. It is also considerably faster in its tick-tacking, reaching a peak of 22 tick-tacks in 500 steps compared to only 15 for the continuous action space agent. It is important to note here, that while the discrete actions are limited to full motor engagement or holding, the PID-based variant is capable of reaching the same accelerations.

Both variants currently struggle with their initial first contact, both losing possession instantly in around 1/3 of the episodes. Many of those failures occur whenever the ball is initialized in line with a player figure on the bar caused by

the randomization of the starting positions. A reward shaping approach such as used in section 5.3 might be suitable to alleviate this failure case.

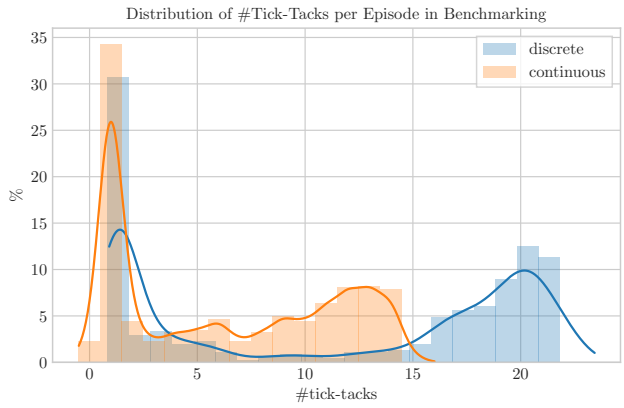


Fig. 6: Distribution of total number of tick-tacks per episode in benchmarking for an agent with discrete actions and one with PID-based continuous actions.

In conclusion, the discrete action space is easier to train, displaying better control given the same amount of training time compared to a continuous counterpart that is assisted by a motor controller, similar to one used in earlier work [2].

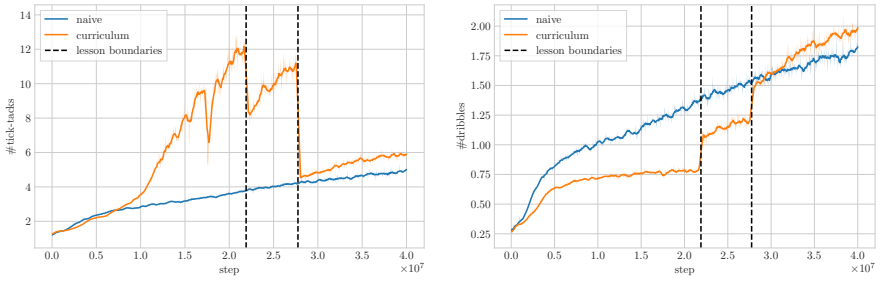
5.2 Curriculum Learning

Dribbling Task The curriculum consists of three lessons, numbered 1 through 3, where each lesson is only different in how the target figure pair is changed throughout an episode. First, in (1) the dribbling target is initialized randomly and does not change until the end of the episode, meaning the agent has to perform a single lateral pass from standstill followed by tick-tacking. Then in (2) the target is changed one time after completing the initial pass form standstill, thus requiring a second dribble, now from a moving ball, alternated with tick-tacking. Finally, in (3) the target is changed once after completion of each dribble with a small delay, thus rapid firing dribbles. The other agent is trained naïvely, directly in its most complex form, i.e. starting out in lesson (3).

Both agents are trained for 4×10^7 steps and are given a neural network with 4 layers and 256 neurons each. Again, the hyperparameters of the PPO algorithm are configured according to the Unity recommended ranges [14]. The agents are compared on their peak performance, determined by their average number of successful tick-tacks and dribbles per episode.

By modulating the way the dribbling target changes between lessons, we aim to teach tick-tacking and dribbling subskills in a more isolated manner. Figure 7

confirms this assumption. In the early episodes the initial lateral pass improves rapidly, followed by a long period where tick-tacking greatly improves. In the second lesson tick-tacking is already mostly mastered, yet dribbling improves further. Finally, in the lesson where the target pair continuously changes within short intervals, both metrics keep increasing slowly. (An important correlation between the two graphs and an explanation for the considerable drop at the third lesson in the tick-tacks graph, is that each lesson increases the number of target pair changes and thus dribbles. As more frequent dribbling results in less time tick-tacking, it explains the apparent drop in tick-tacking performance.)



(a) Average number of tick-tacks per episode.

(b) Average number of dribbles per episode.

Fig. 7: Average number of tick-tacks and dribbles per episode throughout training. Dotted lines indicate transitions between consecutive lessons.

The same figure also shows that the agent trained with CL, outperforms the naïvely trained one by achieving higher peak performance at the end of training, being able to complete more tick-tacks and dribbles on average in time limited episodes. Qualitatively[1], we can see the CL trained agent mastered both tick-tacking and dribbles, while the other appears chaotic often showing moves that seem to merge both tick-tacks and longer passes into an unfavorable hybrid.

Forward Passing Task Performing successful passes requires several subskills (e.g. passing at angles, timing a pass, catching). In the CL approach we teach these simpler sequences separately in a progression of lessons, increasing the difficulty by slowly increasing the opponent's complexity. Before training the agent on the full forward passing task against the defined opponent policies, we first teach the required subskills based on intermediate, less difficult policies.

The lessons, numbered 1 to 5, are structured as follows: (1) effectively removes the opponent, letting the agent train the control aspects of performing straight passes and catches, then (2) places the opponent directly in the way of the

[1] Short recordings to evaluate the qualitative performance are available here: project playlist.

straight pass, forcing the agent to learn angled passes and catches, (3) places the stationary opponent at a random position with a random rotation in $[0, 0.15] \cup [0.85, 1]$, (4) introduces the oscillating opponent to enforce timing of a pass, and finally (5) adds in the remaining policies from section 4.4. In the naïve training approach, we do not teach subskills in isolation and instead train against the four complex opponents directly.

Both approaches are trained for 6×10^7 steps, where each agent uses a 6 layer neural network with 256 neurons per layer. The larger size compared to previous tasks accounts for the agent's need to model the dynamics of two controllable rods, plus the opponent's rod. The PPO hyperparameters are again set according to the Unity recommendations [14]. At the end of the training time, we compare both agents in terms of successfully completed passes against each of the benchmarking opponents.

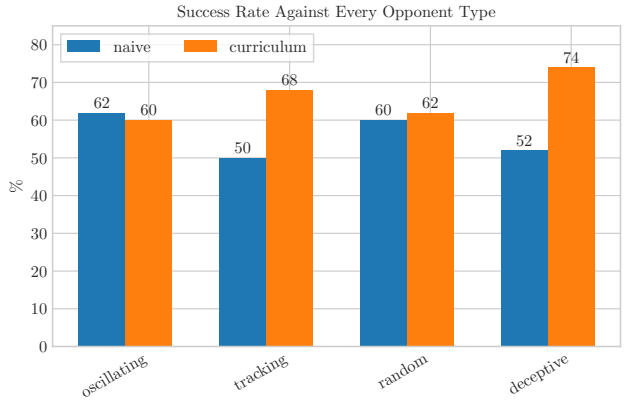


Fig. 8: Success rate of 50 trials against each opponent type of the naïve and curriculum trained agents.

The chart in Figure 8 shows the rate of successfully complete passes against each opponent type recorded over 50 trials each. There is a clear difference between the naïvely and CL trained agents, with the CL agent mostly exceeding the performance of the naïve agent, especially for the more complex opponents. Furthermore, there are differences in the mastering of some of the required subskills. The CL agent's passes are blocked in 10.5% of cases on average, while the naïve agent is blocked 15% of the time, and the CL variant misses a catch in 7% of the trials, versus 11.5% for the naïvely trained agent.

5.3 Reward Shaping

The agents trained on the forward passing task in the previous experiment showed two unwanted behaviors, moving aimlessly when no actions were needed

(e.g. passer keeps spinning after making the pass) and the receiver not maintaining control over the ball past the first touch. We shape the reward structure by introducing two simple auxiliary rewards to alleviate these problems.

A movement penalty is introduced to associate a cost with each non-hold action with a value of -9×10^{-4} for translation and -1.5×10^{-3} for rotation. These values were determined first, via a heuristic, ensuring that the maximal penalty for constant movement over a maximal length episode would be slightly smaller than the smallest reward of 1 in the original reward function, and were then tuned slightly through iteration. The second is a reward tied to the speed of the ball during every simulation step after making the catch. The reward follows the formula $r = 0.002667 * scale^2$ where $scale = \text{clamp}(1 - 100v_{ball}, 0, 1)$. The scalar values were determined so that velocities greater than the maximal recorded stopping velocity in previous agents, roughly correspond with 0, and the maximal reward would be smaller than the smallest regular reward of 1.

An agent is trained in an identical manner to the one in the previous experiment, meaning for 5×10^7 steps with a neural network of 6 layers with 256 neurons each. We reused the CL method for its effective training results and benchmark the new agent against the one from the previous CL experiment, by running each for 100 episodes and recording 2 sets of relevant metrics: average linear and angular rod velocities, and average ball velocity after a catch is made.

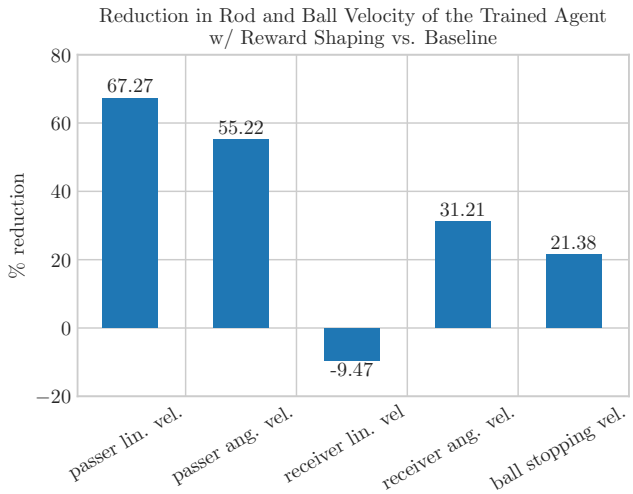


Fig. 9: Percentage improvement on average rod velocities and average ball stopping speed after catch with reward shaping.

Figure 9 shows a comparison of the recorded metrics for the two agents. As shown, there is a significant improvement on all metrics, with the exception of receiver angular velocity. This is due to the competing nature of the two shaping

rewards, where one incentivizes the policy to hold the motors more overall, while the other pushes the agent to do more work to keep the ball still after the initial catch. Importantly, the agent also did not lose any performance toward the main goal, achieving an average success rate against the opponents of within 1% of the original agent. Qualitatively, the agent showcases a new, clearly discernible behavior where it flips around over the ball to pin it from the side it typically rolls away to, and rod motion is reduced overall with little wasted work and a reduction in its sometimes jittery appearance.

6 Conclusion

This study advances the application of reinforcement learning (RL) in robotic foosball, developing an environment and agents capable of mimicking aspects of human game play. Via RL, we pushed the state-of-the-art for robotic foosball agents to execute more complex action sequences as compared to previous work. These actions require a blend of fine motor control and strategic decision-making, demonstrating that RL is capable of addressing the varied challenges of dynamic, competitive environments like foosball.

Our experiments showed that agents utilizing discrete actions, which directly control the actuators, outperform those using continuous actions controlled by an additional motor controller in the simulated environment. Discrete actions led to higher peak performance in the control-focused tick-tacking task. Furthermore, via curriculum learning we effectively increased task complexities beyond other RL-based work. By learning first simpler tasks, agents were able to progress more efficiently and achieve better overall performance compared to those trained naïvely. Lastly, reward shaping was successfully used to refine agent behavior towards expected domain behaviors. We presented two low-effort examples to improve on additional metrics related to efficiency, safety and domain expectations, retaining a desirable level of control over the trained policy akin to manual modeling as done in previous work.

These findings open up possibilities for future research in RL-driven robotics. Within the environment of robotic foosball, there are multiple next challenges that can be tackled such as further sim-to-real research, multi-agent competition, and complex sequential decision-making (e.g., hierarchical RL). In addition, the robotic foosball environment has shown a challenging yet manageable complexity such that, in combination with reinforcement learning, curriculum learning and reward shaping, future results and insights will be of direct relevance for a wider set of robotic environments that involve dynamic, competitive tasks.

References

1. Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., Mordatch, I.: Emergent Tool Use From Multi-Agent Autocurricula, http://arxiv.org/abs/1909.07528

2. De Blasi, S., Klöser, S., Müller, A., Reuben, R., Sturm, F., Zerrer, T.: KIcker: An Industrial Drive and Control Foosball System automated with Deep Reinforcement Learning **102**(1), 20 (2021). https://doi.org/10.1007/s10846-021-01389-z, https://doi.org/10.1007/s10846-021-01389-z
3. Dulon, M., Windler, N., Zemp, M.: Babyfoot: from juggling to shot on goal, https://www.epfl.ch/labs/la/wp-content/uploads/2021/07/BB_IngSim_DULON_WINDLER_ZEMP_Spring2021.pdf
4. EPFL: Babyfoot, https://www.epfl.ch/labs/la/pi/babyfoot/
5. Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S.M.A., Riedmiller, M., Silver, D.: Emergence of Locomotion Behaviours in Rich Environments, http://arxiv.org/abs/1707.02286
6. Juliani, A., Berges, V.P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D.: Unity: A General Platform for Intelligent Agents. https://doi.org/10.48550/arXiv.1809.02627, http://arxiv.org/abs/1809.02627
7. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
8. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236, https://www.nature.com/articles/nature14236
9. OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, 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.: Learning Dexterous In-Hand Manipulation, http://arxiv.org/abs/1808.00177
10. OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H.P.d.O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S.: Dota 2 with Large Scale Deep Reinforcement Learning, http://arxiv.org/abs/1912.06680
11. Rohrer, T., Samuel, L., Gashi, A., Grieser, G., Hergenröther, E.: Foosball Table Goalkeeper Automation Using Reinforcement Learning (2021), https://www.semanticscholar.org/paper/Foosball-Table-Goalkeeper-Automation-Using-Learning-Rohrer-Samuel/46fa82334baf0aadd19381fd2f5a1a189cf3861c
12. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. https://doi.org/10.48550/arXiv.1707.06347, http://arxiv.org/abs/1707.06347
13. Technologies, U.: Unity-Technologies/ml-agents, https://github.com/Unity-Technologies/ml-agents
14. Unity: Training Configuration File - Unity ML-Agents Toolkit, https://unity-technologies.github.io/ml-agents/Training-Configuration-File/
15. Weigel, T.: KiRo - A Table Soccer Robot Ready for the Market. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. pp. 4266–4271 (2005). https://doi.org/10.1109/ROBOT.2005.1570776, https://ieeexplore.ieee.org/document/1570776
16. Weigel, T., Nebel, B.: KiRo – An Autonomous Table Soccer Player. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002: Robot Soccer World Cup VI.

pp. 384–392. Lecture Notes in Computer Science, Springer (2003). https://doi.org/10.1007/978-3-540-45135-8_34