

Learning to Play Robot Soccer from Partial Observations

1st Márton Szemenyei

Dept. of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
<https://orcid.org/0000-0003-1397-6080>

2nd Patrik Reizinger

Dept. of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
<https://orcid.org/0000-0001-9861-0293>

Abstract—Reinforcement learning (RL) has undergone unprecedented evolution in the last few years, managing to surpass the human baseline in several high-profile board and computer games. Despite these successes, deep neural agents remained absent in several challenging fields, such as robot soccer, which is a rather complex task involving simultaneous cooperation and competition between multiple agents. In this paper, we investigate the feasibility of playing robot soccer via deep reinforcement learning using an environment of our own making. This environment provides the agent with imperfect and incomplete observations to simulate the errors of a real vision pipeline. To improve the agent’s performance on this task, an intrinsic reward module is proposed based on self-supervised learning. Our experiments show that the proposed extension significantly improves the agent.

Index Terms—Reinforcement Learning, multi-agent, self-supervised learning, reconstruction

I. INTRODUCTION

RoboCup is a major yearly international event aimed at advancing research in the fields of Artificial Intelligence (AI) and Robotics. The event provides numerous challenges involving robots performing specific complex tasks, such as playing soccer, providing home assistance, or participating in rescue missions. All of these challenges include several different leagues with different robot hardware capabilities and slight variations on the task.

In recent years, deep learning methods became widespread, resulting in the frequent application of deep neural networks to various perception [1, 2] problems in RoboCup. However, despite the groundbreaking advances of deep Reinforcement Learning (RL) methods in recent years [3, 4, 5, 6], there have been few applications in the context of robot soccer [7, 8].

Arguably, the two major factors enabling the rapid development of new methods are the easy access to research and open-source libraries, such as PyTorch or Keras. The other factor is the wide availability of various simulated environments, such as the ones in OpenAI Gym [9], as well as other compatible environments, such as the Duckietown [10] simulator. While most RoboCup challenges have simulation leagues (such as the RoboCup Soccer 2D and 3D leagues), these simulators have not been designed with reinforcement learning in mind.

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

In this paper, a novel simulated RL environment is presented for RoboCup soccer, designed for training multiple cooperating and competing neural agents. To fulfill the requirements arising in RL research, the simulator adheres to the following design principles:

- Compatibility with the OpenAI Gym standard.
- Focus on learning strategy and high-level goals, so problems such as bipedal robot control or object detection are abstracted over, but common failures are emulated.
- Usage of 2D physical simulation for reasonably accurate ball and robot movement, including collisions and falling. This allows robots to learn approximate physics and the inherent risks of specific actions.
- Computational efficiency, as the speed of the simulation is a significant bottleneck of the training process.

In addition to the new environment, we also present a novel neural network architecture to facilitate learning in the simulator. The architecture is capable of handling common errors, such as false positive/negative, noisy, and partial object detection, by attempting to accurately reconstruct the full state from multiple imperfect observations in a self-supervised manner. Our experiments show that by pre-training the reconstructor sub-network, the rewards achieved by trained agents can be significantly improved.

II. RELATED WORK

A. Reinforcement Learning in Robot Soccer

While the research of reinforcement learning methods in RoboCup is scarce, the first attempts go back to the earliest days of the annual competition. Stone and Sutton [11] used traditional RL methods to solve the keep-away problem, in which one team of agents is tasked with foiling the other team’s attempts to seize control of the ball. Later, Riedmiller et al. [12] used neural batch RL to learn the value function in robot soccer. Notably, RL methods were also present in other leagues, such as the 2D simulation, showing that learning in such environments is possible, although cumbersome [13].

Research using modern deep reinforcement learning methods is scarce nonetheless. Schwab, Zhu, and Veloso [7] expanded on team CMDragons’ existing Skill, Tactics, Play (STP) architecture, and replaced hand-coded skills with

learned ones using the Deep Deterministic Policy Gradient (DDPG) method. They showed that these learned skills could be integrated into high-level human-designed tactics successfully. The skills learned include walking up to the ball or kicking it in a particular direction.

Since the 3D robot soccer simulator provides a reasonably accurate dynamic model of the Nao robots used in the standard league, it can be used to learn certain control-centric behaviors, such as walking or running. Abreu, Reis, and Lau [8] used the widespread Proximal Policy Optimization (PPO) method to train a simulated Nao robot to run and stop in this environment. Their maximum achieved stable running speed was 2.5 m/s, which is almost double the previous best. Notably, in the real world, the Nao robots achieve significantly lower top speeds.

B. Curiosity-based Exploration

A significant contribution of the field to widen the range of successful applications of the reinforcement learning paradigm was the introduction of the notion of intrinsic rewards. Among other model-based formulations, such as World models [6], curiosity [14, 15] formulations were developed to incentivize the learning process of the agent, while simultaneously incentivizing exploration of the environment.

The curiosity paradigm was extended in Reizinger and Szemenyei [16] by utilizing the attention mechanism [17], which was proven to result in superior performance in different fields of deep learning. The main contribution of Reizinger and Szemenyei [16] is the so-called Rational Curiosity Module (RCM), an adaptive curiosity loss formulation, which was even utilized in multi-agent environments [18].

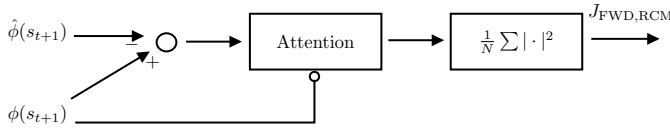


Fig. 1: Forward loss formulation of the RCM agent [16]

The main idea of RCM is - as shown in Figure 1 - to utilize the attention mechanism, conditioned on $\phi(s_{t+1})$ - where ϕ denotes the transformation of the state s_{t+1} into feature space and $\hat{\cdot}$ stands for predicted values. This way, it is possible to focus on a specific subspace of the features extracted by the neural network; and to use this weighted difference in the calculation of the loss of the forward model of the original Intrinsic Curiosity Model (ICM) [14]. That is, the attention mechanism selects the dimensions of the loss where exploration will be incentivized. The rationale behind this idea is that although the features provide a compact representation of the whole state space, it is possible that in particular steps, different subspaces are relevant to be modeled. Thus, the curiosity loss should only incentivize the agent to learn a better model of those features.

As it was shown in Szemenyei and Reizinger [18], the RCM approach is equivalent to an adaptive weighted least squares

formulation, where adaptivity is given by the attention mechanism. I.e., $J_{FWD,RCM}$ is equal to the following quadratic form:

$$J_{FWD,RCM} = \frac{1}{N} e_{t+1}^T e_{t+1} \quad (1)$$

$$e_{t+1} = \sqrt{W} \left(\phi(s_{t+1}) - \hat{\phi}(s_{t+1}) \right), \quad (2)$$

where W stands for the weights of the Attention network, as noted in [18].

III. ENVIRONMENT

In the following, we describe the proposed environment, first introduced in our previous work [18]. The OpenAI Gym-compatible [9] environment simulates robot soccer. Our intent was to implement a lightweight multi-agent scenario simulator for robot soccer while abstracting away from low-level physics (such as the control of robot joints) or expensive 3D graphics.

However, following our goal to make the environment realistic, we also attempt to emulate the failures of real computer vision pipelines, so that the agent may learn to deal with them. For this reason, the environment provides a partial observation type, in which the detected objects are returned in the agent's own coordinate system, and the visibility of the objects is limited by their distance from the agent and the field of view of the robot's camera.

The environment also provides an image-based observation type, which provides a semantically labeled image for each of the robot's two cameras, aiming to emulate the output of semantic segmentation. Both of these observation types introduce the challenge of localization of both the agent and other dynamic objects in the absolute coordinate system.

Both observation types support the emulation of detection failures, including false negatives, false positives, misclassifications, and inaccurate localization. Objects may also be lost due to occlusion. The failure emulation feature also supports two modes: in the random mode, these errors occur with a fixed, adjustable probability. In the realistic mode, however, the probability of random errors depends on other factors as well, aiming to adjust the reliability of the observations depending on the situation. These additional rules are as follows:

- 1) The magnitude of position noise and the probability of false negatives increases with the distance from the agent.
- 2) False negatives are more likely to occur when two or more objects are nearby or at the edge of the field of view.
- 3) Visually similar objects (such as balls and penalty crosses) are more likely to be confused than dissimilar ones.
- 4) False-positive sightings are more likely to occur in certain places (such as false-positive balls at the feet of other nearby robots).

The environment and the observation types are shown in Figure 2.

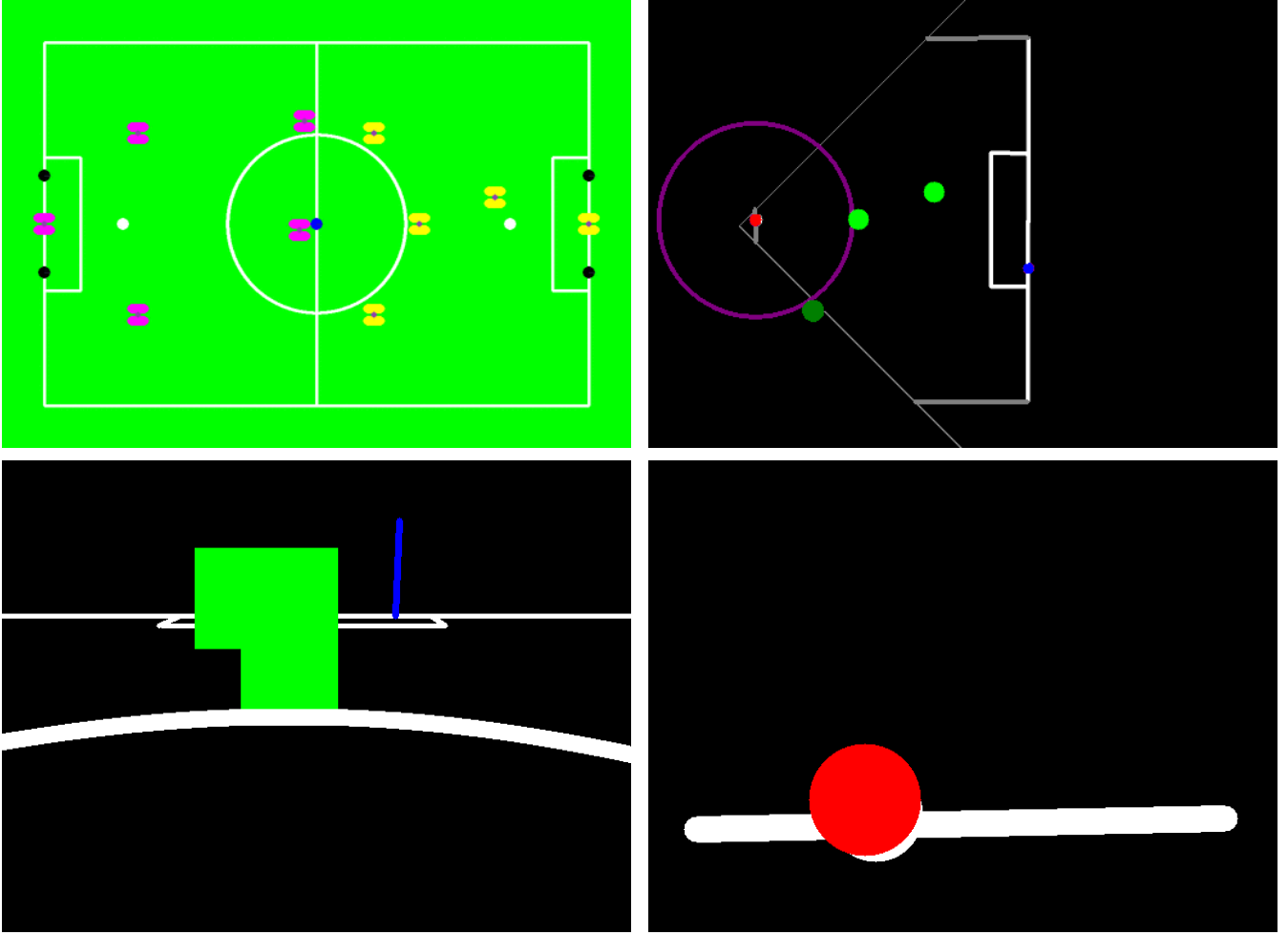


Fig. 2: The environment (top left) and the partial observations (top right). The bottom images are the top and bottom camera observations (from left to right).

The reward system implemented in both environments aims to encourage the agents to make moves towards reaching the end goal. The types of achievable rewards can be organized into three categories: **primary**, **rule-based**, and **auxiliary**. Primary rewards are the simplest to describe: the agents score for reaching or failing their goals, e.g. scoring. Rule-based rewards are usually awarded for breaking a rule, and as such, they are usually negative. These rules include pushing or walking off the field.

Auxiliary rewards are the trickiest, as they are awarded for actions that usually increase the probability of success, yet they are not necessarily the optimal actions. As such, the magnitude of these rewards is considerably smaller so that a later primary or rule-based reward could compensate them. Agents can get auxiliary rewards for getting closer to the ball, or moving the ball towards the opponent’s goal is also rewarded.

IV. MODEL

The environment introduced in the previous section presents novel challenges, necessitating the design of a custom model architecture. To this day, there are few environments [19]

that provide agent-centric partial observations, and even those only account for occlusion. Our environment, on the other hand, introduces false negatives and positives along with misclassifications, which renders traditional architectures that expect a fixed number of objects infeasible.

Moreover, the Nao robots used in real-world games are relatively clumsy, meaning that executing a single action (such as moving, turning, or kicking) takes considerable time, usually between 500 – 1000 ms. This results in the vision pipeline running at a much higher frame rate, breaking the single action-single state orthodoxy of reinforcement learning. Indeed, performing a single action in this environment results in multiple observations at different intermediate time steps.

A. Encoder

For the reasons detailed above, the proposed encoder architecture has to tackle noisy, incomplete observations and effectively combine them over multiple time steps. Moreover, it should satisfy the requirement of being able to extract information from variable-length observations (as the number of objects varies) with multiple object types. To manage this

problem, the encoder first embeds every observed object using a small Multi-Layer Perceptron (MLP) independently. Each object type has its unique embedder MLP; however, the number of outputs are equal. This allows these embeddings to be concatenated to form a varied-length sequence of embeddings for each time step.

The first step is to allow the network to learn how objects in a single observation interact. This is essential since the objects in the game interact in various ways, causing false positives/negatives or occlusion, among others. Also, the presence of particular static field objects means that other such objects should be nearby - such as observing the penalty box would mean that the goalposts are also in the neighborhood. To allow the model to learn these, a self-attention layer is applied to the sequences in every time step independently (these layers share weights, naturally). The self-attention utilized is a multihead attention layer with scaled dot-product attention, the same as used in [17], which was shown to be able to learn the inherent connections between a sequence of data.

The next step of the encoder architecture is to combine the observations at different time steps into a single sequence. Naturally, interactions between elements of these sequences are integral to this step as well. For this task, we used a new type of layer called Recurrent Temporal Attention (RTA) [18]. This module is a multi-head attention layer, where the two input sequences are the observations at two subsequent time steps. Once the output sequence is produced, it is fed back as one of the inputs, with the next observation sequence taking up the other input. The architecture is shown in Figure 3.

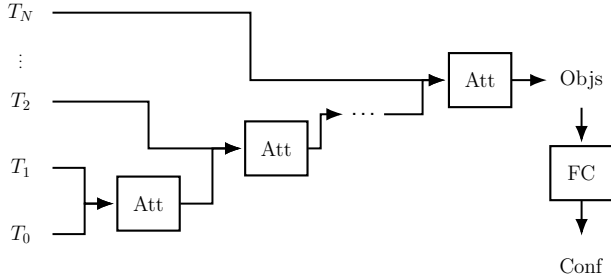


Fig. 3: The Recurrent Temporal Attention (RTA) module.

It is easy to see that this is simply an RNN cell, except that it uses a multi-head attention layer in place of a linear one. Naturally, there are a few further considerations: first, when combining the sequences of two subsequent time steps, the model must avoid losing objects due to false-negative sightings. To achieve this, padding is applied, which results in the final output of the RTA module having as many elements as its longest input.

However, this guarantees that false-positive sightings will remain in the sequence. To allow the model to filter these out, a simple linear layer with sigmoid activation is applied to the elements of the final output sequence, producing a confidence score for every object. Then, as the final step of the encoder, the objects are averaged, using the confidence scores

as relative weights, producing a single embedding vector for the observations. The encoder is shown in Figure 4.

B. Localization and reconstruction

Notably, in this environment, the game rules depend on the *absolute* position of the relevant objects, as opposed to their position relative to the agent. In other words, localization is required to avoid scoring own goals. The observations in our environment are inherently relative; however, it is easy to see how this would make learning difficult.

Two auxiliary outputs are introduced to circumvent this difficulty, aiming to help the agent to learn absolute localization and detection. The first of these outputs is called the localization output, where the agent is trained to accurately predict its own position and orientation at every step in a self-supervised manner. Note that robots can move their heads from their bodies independently, so predicting the robot pose involves determining both orientations.

In addition to the localization output, there is also a reconstruction head, aiming to accurately predict the position and other properties of dynamic objects (the ball and other robots in this case) in the environment. This output helps the agent to learn to determine the absolute position of other relevant objects. It aims to reconstruct the state vector of the environment, hence its name, while it also explicitly trains the model to reduce position noise and to account for false negatives and positives.

When predicting multiple objects of the same type, such as robots, the reconstruction head uses anchors, similarly to YOLO [20]. In this case, anchors are fixed to certain positions on the field, whereas objects are assigned to anchors to minimize the sum of the distances between the object and anchor positions via the Hungarian algorithm. Notably, during reconstruction, the agent is only trained to predict objects it has seen at least once before during the rollout.

The architecture of the proposed agent is depicted in Figure 5: it consists of two encoder modules for static and dynamic objects, respectively. The output features of both encoders are concatenated and fed into an LSTM [21] layer to capture temporal dependencies. The recurrent layer produces the features that will be used for localization, reconstruction, as well as for the A2C and RCM heads.

V. EXPERIMENTAL RESULTS

The proposed methods have been evaluated and compared against baseline algorithms to demonstrate their efficacy. The evaluation was performed for the environment, using the partial observation mode and realistic noise with a magnitude of 1.0. We used two agents per team for robot soccer, and the training was run with 8 parallel environments.

We used a relatively simple metric for evaluation: for each experiment, we computed the best average reward achieved by the agents (the average was computed using a 10 episode moving window). Two agent types were compared: the baseline agent was an A2C architecture extended with the RCM

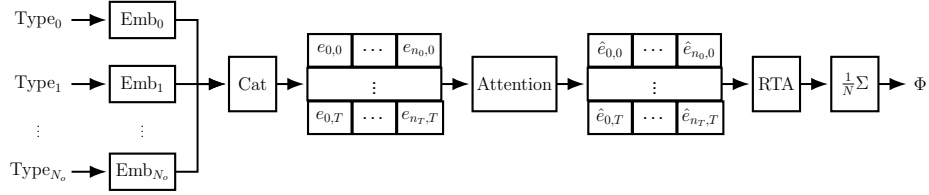


Fig. 4: The encoder architecture. Here $\hat{\cdot}$ is only used to denote that attention was applied to the embedded objects.

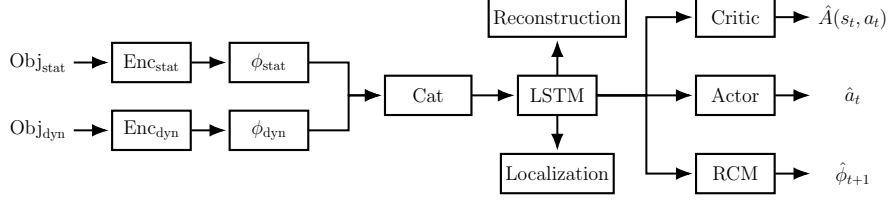


Fig. 5: The network architecture. The subscripts stat and dyn denote components (objects, encoder, and features) for static and dynamic objects, respectively. $\phi(s_t)$ stands for features, a_t for actions, and A for the advantage function $[A(s_t, a_t) = Q(s_t, a_t) - V(s_t)]$, i.e., the difference of the state and action value functions; whereas $\hat{\cdot}$ denotes predicted values.

module, while ‘our’ method included the reconstruction module.

The training was performed with 8 different random seeds, in order to make sure that the differences between the architectures were not due to a lucky random seed. Finally, a Bayesian t-test was used to determine whether the improvement provided by the proposed methods is significant. For performing the test, the independent samples version from the BEST [22] was used, since having the same random seed does not necessarily make two runs directly comparable.

Notably, the network used in the reconstruction variant was pre-trained for localization and reconstruction, using randomly generated samples from the environment. These samples were N step sequences, where N is the rollout size. To measure the effectiveness of the pre-training, the Average Precision (AP) was measured at different threshold values. Table I shows the results of pre-training for localization and reconstruction.

TABLE I: The AP achieved for localization and reconstruction.

Distance threshold	25 cm	50 cm	1 m
Localization	83%	100%	100%
Reconstruction	54%	70%	92%

Notably, localization achieves far better results than reconstruction. The reason for this is that for distant objects, even small errors in determining the robot’s orientation can result in significant errors in absolute localization. To help to put the accuracy in perspective, the soccer field is 9×6 meters in size.

To examine the effects of the localization and reconstruction module, the RoboCup environment was used with partial observations. The 95% credible Interval (CI) and the mean of the improvement in the reward, the probability of improvement compared to the baseline, and the baseline rewards are shown in Table II.

Clearly, the results show an approximately 15% improvement compared to the RCM baseline and a near 100%

TABLE II: Results of the Bayesian t-test.

Metric	95% CI	Mean	PoI	Baseline Reward
Results	[2.01, 2.44]	2.23	100%	13.25

probability that the reconstruction improves the agent’s performance. To put the average 15.48 reward achieved by our agent into context, the random policy achieves a significantly lower, 5.75 reward score, while a human player controlling a single robot (interface supporting multiple human players was not implemented) achieved 22.57 on average.

VI. CONCLUSION

In this paper, we presented a Gym-compatible simulated robot soccer environment for reinforcement learning with noise, imperfect partial observations, which - although realistic - make learning much more difficult. To alleviate this problem, a novel architecture and training method were proposed, employing self-supervised learning to provide agents with an intrinsic reward. This intrinsic reward allowed agents to learn to localize themselves and other relevant objects in the field using the imperfect observations, while also making one step forward predictions, which allow agents to learn the dynamics of their environment.

Our experimental results show that the agents are able to learn in this environment, proving the usability of the simulator. Moreover, the proposed self-supervised training method *credibly* improves the performance of the agents, by a significant margin. Still, the performance of the best agents falls short of the human baseline, requiring further research. However, it is worth noting that due to the multi-agent nature of the problem and limitations of the control UI, the human baseline is only a rough estimate. The implementation of the environment and the code for training is available at <https://github.com/szemenyeim/DynEnv>, and in the form of the DynEnv python package.

REFERENCES

- [1] Bernd Poppinga and Tim Laue. “JET-Net: Real-Time Object Detection for Mobile Robots”. In: *RoboCup 2019: Robot World Cup XXIII*. Springer International Publishing, 2019, pp. 227–240. DOI: 10.1007/978-3-030-35699-6_18.
- [2] Marton Szemenyei and Vladimir Estivill-Castro. “ROBO: Robust, Fully Neural Object Detection for Robot Soccer”. In: *RoboCup 2019: Robot World Cup XXIII*. Springer International Publishing, 2019, pp. 309–322. DOI: 10.1007/978-3-030-35699-6_24.
- [3] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (Oct. 2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.
- [4] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, 2016, pp. 1928–1937. URL: <http://dl.acm.org/citation.cfm?id=3045390.3045594>.
- [5] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. “Self-Supervised Exploration via Disagreement”. In: (June 10, 2019). arXiv: <http://arxiv.org/abs/1906.04161v1> [cs.LG].
- [6] David Ha and Jürgen Schmidhuber. “Recurrent World Models Facilitate Policy Evolution”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 2450–2462. URL: <http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf>.
- [7] Devin Schwab, Yifeng Zhu, and Manuela Veloso. “Learning Skills for Small Size League RoboCup”. In: *RoboCup 2018: Robot World Cup XXII*. Springer International Publishing, 2019, pp. 83–95. DOI: 10.1007/978-3-030-27544-0_7.
- [8] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. “Learning to Run Faster in a Humanoid Robot Soccer Environment Through Reinforcement Learning”. In: *RoboCup 2019: Robot World Cup XXIII*. Springer International Publishing, 2019, pp. 3–15. DOI: 10.1007/978-3-030-35699-6_1.
- [9] Greg Brockman et al. “OpenAI Gym”. In: (June 5, 2016). arXiv: <http://arxiv.org/abs/1606.01540v1> [cs.LG].
- [10] Maxime Chevalier-Boisvert et al. *Duckietown Environments for OpenAI Gym*. <https://github.com/duckietown/gym-duckietown>. 2018.
- [11] Peter Stone and Richard S. Sutton. “Scaling Reinforcement Learning toward RoboCup Soccer”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 537–544. ISBN: 1558607781.
- [12] Martin Riedmiller et al. “Reinforcement learning for robot soccer”. In: *Autonomous Robots* 27.1 (May 2009), pp. 55–73. DOI: 10.1007/s10514-009-9120-4.
- [13] Luiz A. Celiberto et al. “Heuristic Reinforcement Learning Applied to RoboCup Simulation Agents”. In: *RoboCup 2007: Robot Soccer World Cup XI*. Springer Berlin Heidelberg, 2008, pp. 220–227. DOI: 10.1007/978-3-540-68847-1_19.
- [14] Deepak Pathak et al. “Curiosity-Driven Exploration by Self-Supervised Prediction”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, July 2017. DOI: 10.1109/cvprw.2017.70.
- [15] Yuri Burda et al. “Large-Scale Study of Curiosity-Driven Learning”. In: (Aug. 13, 2018). arXiv: <http://arxiv.org/abs/1808.04355v1> [cs.LG].
- [16] Patrik Reizinger and Marton Szemenyei. “Attention-Based Curiosity-Driven Exploration in Deep Reinforcement Learning”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2020. DOI: 10.1109/icassp40776.2020.9054546.
- [17] Ashish Vaswani et al. “Attention Is All You Need”. In: (June 12, 2017). arXiv: <http://arxiv.org/abs/1706.03762v5> [cs.CL].
- [18] Marton Szemenyei and Patrik Reizinger. “Attention-Based Curiosity in Multi-Agent Reinforcement Learning Environments”. In: *2019 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*. IEEE, May 2019. DOI: 10.1109/iccairo47923.2019.00035.
- [19] Bowen Baker et al. “Emergent Tool Use From Multi-Agent Autocurricula”. In: (Sept. 17, 2019). arXiv: <http://arxiv.org/abs/1909.07528v1> [cs.LG].
- [20] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017. DOI: 10.1109/cvpr.2017.690.
- [21] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
- [22] John K. Kruschke. “Bayesian estimation supersedes the t test.” In: *Journal of Experimental Psychology: General* 142.2 (2013), pp. 573–603. DOI: 10.1037/a0029146.