Learning to Defend Penalties in a Humanoid Robot Soccer Environment Through Reinforcement Learning

Frederico Lopes
FEUP
up201904580@edu.up.pt

Fábio Huang
FEUP
up201806829@edu.up.pt

Luís Guimarães
FEUP
up202204188@edu.up.pt

***Abstract*—This paper details the development and results of training a simulated humanoid robot to autonomously defend penalties in a 3D football game through Deep Reinforcement Learning (DRL) algorithms in a simulated environment. Despite the complexity of the task, the robot demonstrated the ability to learn and adapt, although the performance improvement was not substantial. The paper discusses the implementation specifics and challenges of training a simulated robot for penalty defense in football. It highlights strategies and areas for further research, contributing to robotics and machine learning with a realistic case study.**

***Index Terms*—DRL, Humanoid Robot, Football, Simulation**

Fig. 1. Robocup Game

I. Introduction

Reinforcement learning is continually being expanded to new fields. Its capacity to tackle issues with large state and action spaces is gaining prominence as computational power escalates and new algorithms emerge. This trend is particularly adopted in competitive scenarios due to the high demand for innovative and enhanced solutions. One example is the RoboCup competition, where new robotic capabilities are developed annually. Ultimately, the development and optimization of high-performance behaviors could be beneficial for future real-life applications.

In the context of a 3D soccer simulation match, one of the crucial skills is goalkeeping. The team with a goalkeeper who can effectively defend the ball has a significant advantage over the opposing team, which could be decisive if consistently achieved. Even though the goalkeeper task is quite difficult, this can be made easier by training a goalkeeper to defend penalties. The knowledge and strategies acquired from this specific training can then be extrapolated and applied to other tasks within the game, enhancing the overall performance and versatility of the goalkeeper.

This paper presents a method of harnessing the Proximal Policy Optimization using the information provided by the simulator for official RoboCup matches.

This paper is structured as follows. Section II provides an overview of the related work in the RoboCup competition and DRL methods, and Section III describes the Robot's task, setup, and implementation in detail. Then, Section IV presents the experimental setup and discussion of the results and findings, and finally, Section V concludes the paper, summarizing the contributions and outlining potential future research directions.

II. Related Work

A. RoboCup

RoboCup is an annual international robotics competition that has been gathering teams from multiple countries since 1997 [6]. In 2004, the 3D Soccer Simulation League (3DSSL) was adopted, which led to a considerable evolution in low-level behaviors. Every year, every participating team releases their binary to a public repository that contains teams until 2023 [2].

Over the years, several particular skills that robots should have to be successful in this competition have been developed using RL, like learning how to run faster, how to dribble a ball, how to kick a ball, and others [3], [4], [10].

B. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines Reinforcement Learning (RL) and deep learning to solve complex problems with large state and continuous action spaces. It employs neural network models to enhance representation abilities, enabling effective decision-making in various domains like driving and action imitation. DRL's integration of RL and deep learning has attracted attention for its ability to handle challenging tasks previously beyond the capabilities of traditional RL methods. By interacting with environments, DRL agents continually learn from received rewards, refining decision-making strategies over time. This adaptive learning mechanism, combined with the neural network's ability to

approximate complex functions, enables DRL to effectively tackle challenging real-world problems in diverse domains ([11], [12]).

Value-Based DRL methods like Q-learning focus on representing and determining the optimal value functions. On the other hand, Policy-based methods optimize the policy directly instead of first calculating the optimal Q-values [12]. Actor-critic [9] are a type of Policy-Based methods that typically refer to the simultaneous learning of a policy and a value function, where the latter is used to evaluate the policy ([12]). In the next section, we will introduce Trust Region-Based Algorithms to which PPO, the algorithm used for training our agent, belongs.

1) Trust Region-Based Algorithms: Trust Region-Based Algorithms have been introduced into RL in recent years and achieved substantial performance improvements in various scenarios. TRPO ([7]), based on the conclusion of conservative policy iteration ([5]), calculates the maximum total variation divergence as the learning rate and considers the relationship between total variation divergence and Kullback-Leibler divergence so as to extend the mixed policy into a general stochastic strategy.

TRPO struggles with intricate computations or compatibility with certain architectures (e.g., noise or parameter sharing). PPO, introduced by [8], simplifies TRPO by using a clipped surrogate objective while retaining similar performance. Compared to TRPO, PPO is simpler, faster, and more sample efficient.

III. Robot Football

A. Task

As mentioned before, our agent was trained to defend penalty shots in a soccer game. These shots were taken from a fixed place at a fixed distance and pointed to the inside of the goal. However, the direction and speed of each shot were random to make the simulation more real and to make it harder for the agent to defend the ball. Because of this, the agent had to be ready for any kind of shot and quickly decide where to move to block the ball.

B. Setup

1) Simulator: For the simulator used to conduct the learning process we used SimSpark [13], the official simulator for the RoboCup 3D Soccer Simulation League. By using this simulator, we ensure that the optimization results can run seamlessly in official RoboCup matches. Adding to this, SimSpark also uses a realistic and stable physics engine, the Open Dynamics Engine (ODE). The learning agent is a version of the NAO humanoid robot [1], created by SoftBank Robotics.

Since one of the objectives of this work was to train a goalkeeper to be integrated into a competing team, the simulator was configured with the official RoboCup rules with the exception of the cheats and the real-time mode. The cheats were turned on in order for us to be able to move the ball in an efficient way. However, the agent has no access to any information that it wouldn't have during an official match. The real-time mode was disabled to allow for synchronous communication with the agent and to accelerate the learning.

Fig. 2. NAO Robot

2) Observation Space: We use two different observation spaces depending on whether we are optimizing the joints or the sequence of the poses. Starting with the optimization of the joints:

The state space is composed of 30 continuous variables represented by the single-precision (32b) floating-point data type. Some of these observations are obtained directly from the simulator, and others are computed (see Table I). There are a total of 24 joints in the robot, and the movement of all of them is being optimized in the training.

For the ball position, we calculate the ball position relative to the robot's torso. To calculate this, the robot simply uses its vision. In the case of the ball velocity, we get the velocity in each of the axes also in relation to the torso of the robot. To calculate this velocity, the last known positions of the ball are used. We can choose how many of the last positions are used to calculate the velocity. We chose to use the last 3 positions since we considered it was the best value between the tradeoff of having the most updated information about the position (last position) but being inaccurate, and having very accurate but outdated information (choosing a high number of the last observations).

The only difference in the case of the observation space of the poses is that the joints are not taken into account.

TABLE I
OBSERVATION SPACE

Parameter	Data size (x32b)	Acquisition method
Joints	24	raw
Ball position	3	computed
Ball velocity	3	computed

C. Action space

The robot is designed with a joint-actuator system, where each joint has a corresponding actuator. There are 24 actuators in total, mirroring the state space. The simulator controls these actuators using angular speeds, which are continuous variables. However, our agent doesn't directly provide these speed values. Instead, it predicts the target angles for each joint, and these angles indirectly determine the speeds of the actuators.

In terms of training, there are two distinct scenarios. When training the robot on joint usage, all the joints are utilized. However, when training the robot to optimize its poses, the possible actions change. In this case, each pose is defined by a list of rotation values that the joints should assume. Instead of optimizing these values, the training process optimizes the sequence in which these poses are adopted. Given this, the action space is only 3, throwing himself to the left or to the right and squatting.

D. Training

To train the agent, we created a gym environment using the OpenAI gym library. In this environment, we used the above-specified simulator, action spaces, and observation spaces.

At the beginning of each episode, the robot is positioned in a neutral pose in the center of the goal (-14m, 0m) and the ball is positioned 5 meters in front of it (-9m, 0m). The ball is then shot with a random speed and direction within a certain interval to ensure that it is directed to the inside of the goal. The episode resets when either the agent defends the ball and the ball stops or the ball passes the goal line.

For the rewards of the agent, we tried 2 different types of rewards, sparse and dense. For the dense rewards, the model only receives a reward of 1 if it defends the ball and receives 0 otherwise. In the dense rewards, we implemented a new reward where even if the agent does not defend the ball, it receives a reward of 1 at each step if it is closer to the ball (only considering the y and z axis, since the x axis will always get closer). In this gym, the agent receives 1000 of reward instead of 1 if it defends the ball.

As mentioned before, we also have two different types of actions for that agent. The agent is either trained to learn which joints to move and when in order to defend the ball, or it learns the best sequence of predefined poses given a certain shot direction and speed. All the models were trained with about 50 000 episodes, this being translated in a different number of steps depending on the available actions since the steps were defined in different ways. For the combination of Dense rewards with all the joints, we also did extra training, using 100 000 episodes (10M steps) instead of 50 000 (5M steps).

For the DRL algorithm, we chose to use the PPO implementation from Stable Baselines3. Based on other projects aimed at training agents for the RoboCup competition ([4]), for the optimization layer we used a multilayer perceptron with two hidden layers with 64 neurons.

IV. EVALUATION

A. Experimental Setup

For the testing of the agent, the simulator was configured in the same way as for the training. The only difference was that real-time mode was sometimes turned on to allow for a visual inspection of the behavior. To test the 3 trained models, we ran the models in the experimental setup and calculated the percentage of penalties defended by each one for 100 episodes. The results can be seen in Table II

B. Results

TABLE II
PERCENTAGE OF PENALTIES DEFENDED (100 EPISODES)

Type of reward	Action space	Percentage of penalties defended
Sparse	Joints	24%
Dense	Joints	28% / 21%
Sparse	Poses	37%
Dense	Poses	42%

C. Discussion

In Table II we can see that the best results were achieved by using the pre-defined poses of the robot. This was to be expected since the number of available actions was much smaller when compared to moving the joints. Given the smaller action space, was easier for the model to optimize which actions to map to a certain observation.

In contrast, given that the action space was much more complex for the movement of the joints, the results were not so good. The results for the dense rewards were better but still not sufficient. A possible reason for this is that since the model is not able to defend the ball most of the time, it's hard for it to understand which action leads to a better reward and consequently, a higher probability of defending the ball.

Dense rewards also result in a better model overall, even if by a small amount. This is probably because the model has more information when learning which leads to it understanding better which actions to take. Anyway, we can conclude that 5M steps were not enough for the robot to learn correctly how to defend the ball. For comparison, a model with a similar setup trained for 200M steps to learn how to run [4].

A visual analysis of the models also allowed us to conclude that all models have a high tendency to do the same movement every time, either standing straight, going left or right, regardless of the ball's direction. The ball's direction is chosen in a random way, and it should result in a normal distribution of the direction of the ball. However, there may be small biases

in the direction of the ball while training. Because of this, over the millions of timesteps of training, the models learned that it's better to lean always to a given direction instead of trying to go in the direction of the ball. The only model where this trend was not so pronounced was the one trained using dense rewards with all joints. Even though this is the model with the lowest reward, it tries to lean in the direction of the ball.

We could also suppose that a much simpler program without DRL could have been enough to achieve better results than our models trained over millions of steps with state of art DRL algorithms. For instance, a basic Rule-based Agent could be employed, designed to predict the ball's final position in the goal by analyzing its initial positions and velocity. Our agent currently makes decisions based on the three available poses: squat, fall left, and fall right. These poses, however, are insufficient for covering all potential target areas within the goal. To enhance its effectiveness, the introduction of additional poses, such as jumping, jumping towards the top right, and jumping towards the top left, would have to be considered.

Using this information, we can conclude that the best way to quickly create a robot to defend against penalties is also the simplest, by using predefined poses. However, the model using all the joints with dense rewards also showed signs of progress. Training this model with more time steps and tuning the dense rewards would probably be the best way to get an advanced model capable of adapting to the different directions in the best way possible since it offers much more flexibility than the predefined poses.

V. Conclusion

In conclusion, this paper has presented a study on the application of DRL algorithms for training a simulated humanoid robot to autonomously defend penalties in a 3D football game. Despite the inherent complexity of the task, the robot demonstrated a capacity for learning and adaptation, even though there is room for improvement.

The study revealed that models trained with dense rewards outperformed those trained with sparse rewards. The results also indicated that the use of predefined poses yielded the best outcomes due to the smaller action space and easier optimization. However, the models using all the joints with dense rewards showed promising signs of progress, suggesting that with more training steps and reward tuning, they could potentially adapt more effectively to different ball directions.

Overall, while the study has made contributions to the field of robotics and machine learning, it also highlights several areas for future research, including reward structure optimization, action space simplification, and the exploration of simpler solutions. The findings of this study serve as a step towards the development of more advanced and adaptable autonomous robots in the future.

References

[1] Nao robot. https://unitedrobotics.group/en/robots/nao. [Accessed 05-01-2024].

[2] archive robocup. https://archive.robocup.info/Soccer/Simulation/, 2023. [Accessed 05-01-2024].

[3] Miguel Abreu, Nuno Lau, Armando Sousa, and Luis Paulo Reis. Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–8. IEEE, 2019.

[4] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. Learning to run faster in a humanoid robot soccer environment through reinforcement learning. In *Robot World Cup*, pages 3–15. Springer, 2019.

[5] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[6] Itsuki Noda, Sho'ji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. Robocup-97: The first robot world cup soccer games and conferences. *AI magazine*, 19(3):49–49, 1998.

[7] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[9] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[10] Henrique Teixeira, Tiago Silva, Miguel Abreu, and Luís Paulo Reis. Humanoid robot kick in motion ability for playing robotic soccer. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 34–39. IEEE, 2020.

[11] Hao-nan Wang, Ning Liu, Yi-yun Zhang, Da-wei Feng, Feng Huang, Dong-sheng Li, and Yi-ming Zhang. Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 21(12):1726–1744, 2020.

[12] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[13] Yuan Xu and Hedayat Vatankhah. Simspark: An open source robot simulator developed by the robocup community. In *RoboCup 2013: Robot World Cup XVII 17*, pages 632–639. Springer, 2014.