RobôCIn Team Description Paper 2021

Cristiano Santos de Oliveira[1], Mateus Gonçalves Machado[1], Walber de Macedo Rodrigues[1], Thiago da Silva Araújo[1], Pedro Vitor Cunha[1], Rafael dos Reis de Labio[1], Felipe Nunes de Almeida Pereira[1] Mateus Ferreira Borges Soares,[1], Edna Natividade da Silva Barros[1], Tsang Ing Ren[1], and Paulo Salgado Gomes de Mattos Neto[1]

Universidade Federal de Pernambuco, Centro de Informática, Recife PE, Brazil
robocin@cin.ufpe.br
https://www.robocin.com.br/

Abstract. RobôCIn Soccer Simulation 2D team started in 2018 at the Universidade Federal de Pernambuco. Our first competition was at João Pessoa, Paraíba, Brazil in Latin American Robotics Competition (LARC) 2018 where we obtained the 4th place against teams from Latin America. In 2019 we participated for the first time at the RoboCup, obtained the 9th place and we also participated at the Brazil RoboCup Open 2019 (LARC) where we obtained the 2nd place. In 2020 we participated at the Brazil RoboCup Open 2020 (LARC) where we obtained the 3rd place. In this paper we describe the evolution of the approaches developed last year, the new improvements and researches we made for the simulation 2D.

Keywords: Deep reinforcement learning · Machine learning · Clustering · Classification · Statistics.

1 Introduction

RobôCIn is a robotic research team from the Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), created in 2015 to participate in competitions and research subjects related to robotics. We are currently working in four categories: Very Small Size (VSS) since 2015, Soccer Simulation 2D (SS2D), Small Size since 2018 and Line Follow since 2019.

We based our code on agent2d 3.1.1 [1,2] at the first year, since it is a well-structured base and from it we could develop our studies and approaches more quickly. To the new release, we integrated the gliders2d-v1.6 [8] to our code, since it demonstrate a better performance and it is a evolution of the agent2d-3.1.1; we integrated the MarlikBlock of the Marlirk2011 [10] to our moving behaviour and made some improvements to the function that controls the behavior selection. We also made hard-coded improvements to some game situations, in order to make our attack more aggressive, creating new attack behaviors, formations, and improving the action evaluation algorithm.

Also, we continue to improve our Deep Reinforcement Learning (DRL) for defense behavior, focusing on the ball interception, developed last year. During the last year, we also implemented attacking behaviors based on DRL algorithms, which will be explored during this TDP. To deploy these algorithms in the real game environment, we integrate the Open Neural Network Exchange (ONNX) [3] lib to our agent. With ONNX, we can use all kinds of models from a variety of frameworks, leading our agent lenient to use learning models without friction.

Rather than using a simple statistical analysis of the games, we developed a log analyzer to analytically compare the behaviours of our agents with information gathered from *.rcg* log files. The log analyzer is a desktop interface that was built from scratch using Python and it is intended to be extended to other categories, such as Small Size League (SSL), standardizing the analysis of the changes made in an agent's code.

2 Deep Reinforcement Learning

For the RoboCup2020, we developed a defense behavior applying Deep Reinforcement Learning models, more specifically we used the Deep Deterministic Policy Gradient (DDPG) [7] algorithm. In this last year, we seek to improve the defense model by applying a hybrid solution, using hard-coded behaviors and DRL behaviors.

In order to further improve research and applications of DRL algorithms, in this year, we introduced DRL to our offensive players. As the literature of DRL is vast, in this work we compared three algorithms: Deep Deterministic Policy Gradient (DDPG) [7], Sof Actor-Critic (SAC) [5] and Twin Dueling Deep Deterministic Gradient (TD3) [4].

2.1 Deep Deterministic Policy Gradient

There is a branch of Reinforcement Learning that studies the Policies and State Values themselves. There is a Critic and an Actor algorithm, see Figure 2, where the Critic tries to predict the State Value and the Actor the Policy Value, like any Q-Learning based algorithm [9]. The DDPG tries to learn an optimal deterministic policy μ^*.

2.2 Soft Actor-Critic

This algorithm extends the maximum entropy reinforcement learning framework to boost exploration. This solution is capable of learning to achieve complex tasks by generating an extensive sampling of random actions with the inclusion of the entropy measure to the reward computation. We chose this method precisely due to its exploratory approach since the environment we apply is highly dynamic.

2.3 Twin Dueling Deep Deterministic Gradient

Twin Delayed Deep Deterministic evolves from the DDPG method by incrementing some operations to avoid the overestimation of the Q-values. This kind of overestimation is a common issue for DDPG in which commonly leads to policy break.

2.4 Deep Reinforcement Learning on Defense

This year, we used a technique similar to Cyrus2019's [11], which uses DDPG [7] to select defensive actions to a player. In our approach, we introduce a new Action and State Space for defense situations using DDPG.

State Space We designed our state space with 10 general features, 3 features per teammates (T) and 2 features per opponents (O). Resulting in the following features:

1. **X position** - The agent's x-position on the field.
2. **Y position** - The agent's y-position on the field.
3. **Orientation** - The global direction that the agent is facing.
4. **Ball X** - The ball's x-position on the field.
5. **Ball Y** - The ball's y-position on the field.
6. **Able to Kick** - Boolean indicating if the agent can kick the ball.
7. **Goal Center Proximity** - Agent's proximity to the center of the goal.
8. **Goal Center Angle** - Angle from the agent to the center of the goal.
9. **Proximity to Opponent** - If an opponent is present, proximity to the closest opponent. Invalid if there are no opponents.
10. **Interceptable** - Whether the agent can intercept the ball or not.

T **Proximity from Teammate i to Opponent** - For each teammate i: the proximity from the teammate to the closest opponent. This feature is invalid if there are no opponents or if teammates are present but not detected.
$2T$ **X, Y of Teammates** - For each teammate: the x-position, y-position.
$2O$ **X, Y of Opponents** - For each opponent: the x-position, y-position.

Action Space As actions we chose: Intercept (interval: [-1, -0.68)), which is the default block behavior for the agent, Marlik Move (interval: [-0.68, 0.36)), and Marlik Block (interval: [0.36, 1]). Marlik Move is composed by *Cross Mark* and *Play_on Mark* [10].

Cross Mark - This behaviour occurs when an opponent reaches near one of the defensive corner flags, usually when $-36 > Ball_X > -53$ and $20 < |Ball_Y| < 34$. The main idea of the algorithm is to position the marker between the nearest attacker and the ball to reach the ball faster. When the number of defenders is greater than the attackers, man-to-man marking will be done and the rest of the markers will guard the goal for probable shots. When attackers have an advantage, the three midfielders (usually numbers 6, 7 and 8) will try to mark opponents depending on their stamina.

Play_on Mark - This type of marking is to avoid through passes and it is only applied on defenders, usually numbers 2, 3, 4 and 5. The idea is to disarm the attack when they are trying to break the defense line. Depending on attackers position and the defenders formation, the marker will choose one of the attackers to stay "behind" and then block a possible through pass.

Marlik Block - It has two important parts: moving to the best block point and decide how to act then. When the defender is the near from the ball, the best block point is calculated by a prediction of opponent's future dribble target. When the agent is too far from the attacker in possession, it marks the nearest opponent. Once the agent have reached the best position to block the attack, the defender will decide if it will continue doing the block, intercept the ball or push the attackers back.

2.5 Deep Reinforcement Learning on Attack

In this approach we propose to build and evaluate an Actor-Critic Architecture extended into a Parameterized Action Space (AC-PA).

The individual offensive soccer player behavior is centered on two canonical characteristics: ball possession and scoring. With these characteristics, we define three soccer policies as goals for our AC-PA architecture, which are: Go-To-Ball, Ball-To-Goal, and Ball-To-Goal-With-Opponent. For each policy we built a specific reward function and action parameterization, combining low-level action, mid-level action, and high-level action. The definition of these types of actions can be found on Half Field Offensive framework manual [6].

Go-To-Ball Policy represents the behavior which initially the robot is located in a distant position from the ball $p_0^{robot} = (x_0, y_0)$ in the instant t_0 and is capable of arriving at the position of the ball in the instant t_{0+n} subsequently.

Ball-To-Goal Policy Represents the behavior which the robot is able to arrive at the ball located with a certain distance from it and following it takes the ball to score at the goal.

Ball-To-Goal-With-Opponent Policy This approach represents a similar behavior as the Ball-To-Goal having the increment of an opponent in the field, which we define as a goalkeeper. So in this case the robot has to pass by a hand-coded goalkeeper to score

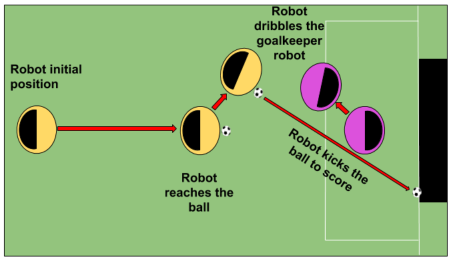
Robot dribbles the goalkeeper robot
Robot initial position
Robot reaches the ball
Robot kicks the ball to score

Fig. 1: Ball-to-goal-with-opponent behavior representation.

3 Log Analyzer

Our goal in building this tool is to increase the number of Key Performance Indicators (KPI) available during our analysis. Our previous analysis method only accounted for victories, defeats, draws, goal balance, and score per game. The log analyzer reads in a .csv file parsed from a .rcg and plots information in widgets that can be placed side by side for comparison. Among the available information are: the amount of faults committed/received, the location of these faults, the stamina through time of all players, the or any positions in any given moment, goal replays, and a heatmap for the ball or any player position's.

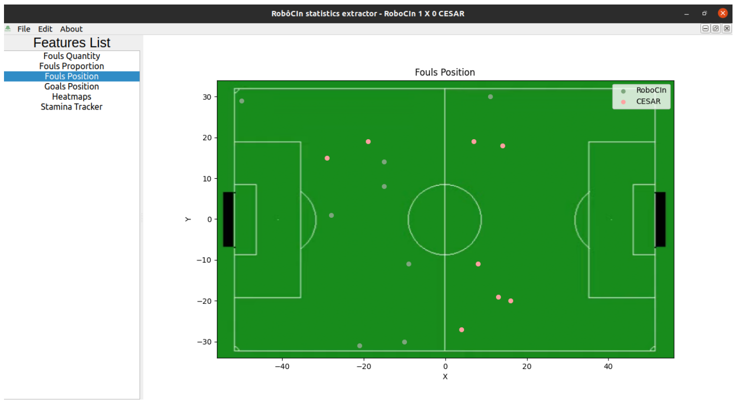

Fig. 2: Log Analyzer's graphic user interface showing the positions of fouls occurrence.

4 Experiments and Results

4.1 Deep Reinforcement Learning Defense

We decided to analyze the following attackers against our DRL Defense: Agent2d, RoboCIn2019, Fractals2019, Helios2019 and Cyrus2019. We could not perform comparisons with Fractals2019 due to issues executing it's binaries. We defined successful defenses as: ball out of pitch or any defensive player has the ball.

	RoboCin2019 Defense	DDPG Defense
Agent2d Attacker	99%	99%
RoboCin2019 Attacker	53.3%	86.7%
Helios2019 Attacker	53%	65%
Cyrus2019 Attacker	84%	93.7%

Table 1: Successful defenses analysis for 3000 episodes. DDPG Defense using RoboCIn2019's goalie.

4.2 Deep Reinforcement Learning Attack

Go-To-Ball Experiment The three Actor-Critic models, DDPG, TD3, and SAC were trained until converge to an optimal policy for the Go-To-Ball with complete dash action parameterization architecture. As shown in Figure 3, the SAC method approaches the optimal policy more slowly than the TD3 and DDPG. Also, it took more time of training: 5000 episodes.

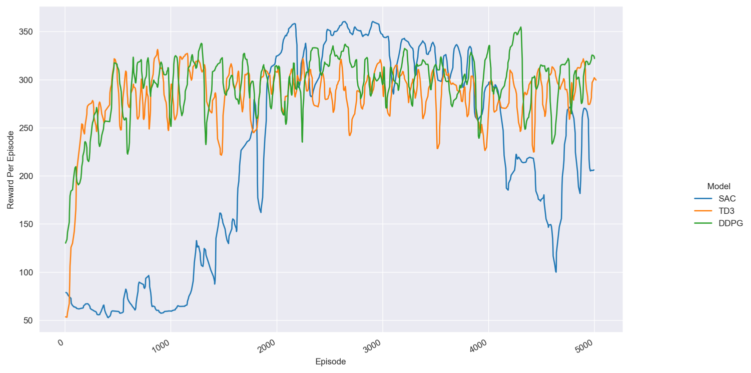

Fig. 3: Go-To-Ball with Complete Dash Action, policy training reward per episodes

Ball-To-Goal Experiment The training session of the go-to-ball policy shown in Figure 4 demonstrates that it was possible to achieve an optimal policy for SAC, meanwhile,the others TD3 and DDPG demonstrated fluctuations.

Ball-To-Goal-With-Opponent Experiment The Ball-To-Goal-With-Opponent adds a higher level of complexity compared to Ball-To-Goal soccer task, in this experiment it was included a goalkeeper to the environment, the goalkeeper has a standard hand-coded RCSS2D implementation. The training session was done until the models converged, just like in previous experiments.

The last comparison was made getting the ball-to-goal-with-opponent SAC trained model with mid-level+high-level action parameterization, playing in 1000 penalty game sessions, and completing the same 1000 session for a RCSS2D offensive agent with hand-coded implementation. The results can be seen in the Table 2.

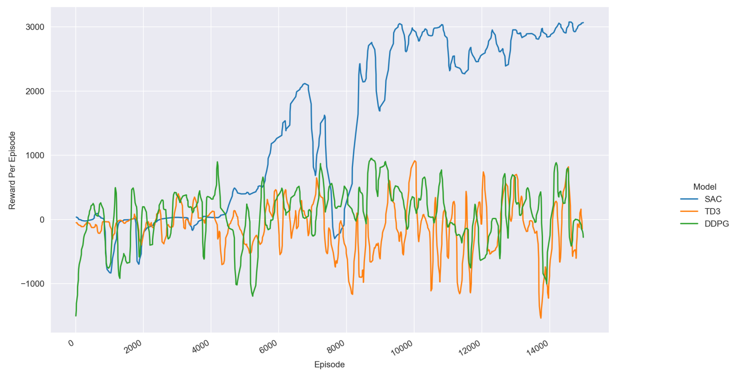

Fig. 4: Ball-To-Goal with Low-level Actions, proposed policy training reward per episodes.

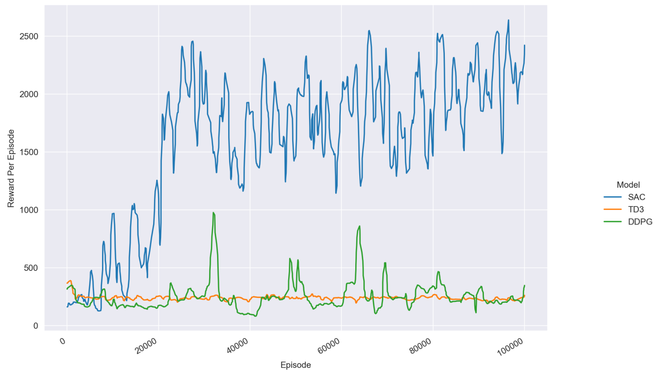

Fig. 5: Ball-To-Goal-With-Opponent with Mid/High Level Actions, policy training reward per episodes

Table 2: Goal percentage and average cycles to score resulted from the SAC Offensive Player and Hand-coded Offensive Player in test round of 1000 episodes.

Offensive Player Algorithm	Goal Percentage	Average Cycles to Score
SAC Offensive Player	68.9%	67
Hard-coded Offensive Player	97.8%	55

The SAC, despite being the best model so far, is outperformed by the hard-coded agent, thus there are further improvements and experiments to be performed. As SAC is highly grounded on exploration, hyperparameter search to the exploration constants is extremely necessary. A further improvement, is modifying the granularity of the parameterization spaces for the actions types. A better parameterization helps the algorithm's convergence into a stable policy. Additionally, we will make a precise investigation on the reward functions and include more features to the model, such as current game cycle and agent stamina.

References

1. Robocup tools agent2d. `https://pt.osdn.net/projects/rctools/`, accessed: 2019-01-01
2. Akiyama, H., Nakashima, T.: Helios base: An open source package for the robocup soccer 2d simulation (01 2014). https://doi.org/10.1007/978-3-662-44468-9$_4$6
3. Foundation, T.L.: Open neural network exchange (2019), `https://github.com/onnx/onnx`
4. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. Proceedings of Machine Learning Research, vol. 80, pp. 1587–1596. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018), `http://proceedings.mlr.press/v80/fujimoto18a.html`
5. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018), `http://arxiv.org/abs/1801.01290`, cite arxiv:1801.01290Comment: ICML 2018 Videos: sites.google.com/view/soft-actor-critic Code: github.com/haarnoja/sac
6. Hausknecht, M.: RoboCup 2D Half Field Offense (Mar 2015), `https://github.com/LARG/HFO`
7. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2015)
8. Prokopenko, M., Wang, P.: Gliders2d: Source code base for robocup 2d soccer simulation league (2018)
9. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction (2011)
10. Tavafi, A., Nozari, N., Vatani, R., Yousefi, M.R., Rahmatinia, S., Pirdeyr, P.: MarliK 2011 Soccer 2D Simulation team description paper (2011), `http://archive.robocup.info/Soccer/Simulation/2D/TDPs/RoboCup/2011/MarliK_SS2D_RC2011_TDP.pdf`
11. Zare, N., Sarvmaili, M., Mehrabian, O., Nikanjam, A., Khasteh, S.H., Sayareh, A., Amini, O., Barahimi, B., Majidi, A., Mostajeran, A.: Cyrus 2D Simulation 2019 team description paper (2019), `https://archive.robocup.info/Soccer/Simulation/2D/TDPs/RoboCup/2019/CYRUS_SS2D_RC2019_TDP.pdf`