RobôCIn Team Description Paper 2020

Cristiano Santos de Oliveira[1], Mateus Gonçalves Machado[1], Marvson Allan Pontes de Assis[1], Walber de Macedo Rodrigues[1], Thiago da Silva Araújo[1], Victor Miguel de Morais Costa[1], Lucas Grisi Oliveira de Queiroz[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 this paper we describe the evolution of the approaches developed last year and 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), @Home and Small Size categories since 2018. At the SS2D category we evolve through our first competition obtaining the 4th at LARC and the second year we obtained the 2nd place at the same competition, and also the 9th place at the RoboCup 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 [7] to our code, since it demonstrate a better performance and it is a evolution of the agent2d-3.1.1; we also integrated the MarlikBlock of the Marlirk2011 [10] to our bhvbasicmove.cpp and made some improvements at the function to be used with our switch of behaviors, depending on the situation on the defensive behavior of the agent.

With the Deep Reinforcement Learning (DRL) approach at the last year we continued the research to improve this method applied to the defense behavior

of interception and we developed new approaches for that method and corrected some of the issues that we found. For the Ball Possession Algorithm, we increased the method using Machine Learning, since our last approach was to use the pure statistic calculation on the chain action to decide the next agent actions, now we have a trained model to do this calculation.

Include learning modules to the agent code is a work with lots of frictions, depending on the framework that is used to train the model and the base language used to train, considering that all our study and tests for the models accuracy was made in Python, because of the facility and the speed that we can analyze and develop models, we have to find a way to integrate theses models to our agent code, whatever is the framework - last year we used only tensorflow framework, but this year and the next ones we what to use other frameworks to create our learning models. To do that we first included the Eigen lib [5] to our code, in which we can do high level linear algebra models and use to load trained models to the agent (we are using this approach to load our Machine Learning model to the Ball Possession action decisor), and we are also working to integrate the Open Neural Network Exchange (ONNX) [4] lib to our agent, with the ONNX we can use all kinds of models from a variety of frameworks, leading our agent lenient to use learning models without friction.

2 Machine Learning Ball Possession Algorithm (ML-BPA)

At the RoboCup 2019 we have developed the Ball Possession Algorithm (BPA), that is a statistical method to select the best action for the agents to keep the possession of the ball to our team at the game. As an improvement over this approach we have replaced by a machine learning based method using a classification approach. With a classification algorithm, our objective is to conduct a study of risk prediction in the player action space.

To keep the ball possession, we guide the evaluation function to select passes that have less chance to be missed. Rather than using statistics extracted from the passes, we made a study over a broad range of machine learning algorithms and features related to the player world perspective.

The first change over the last year is how ball possession is interpreted. Instead of checking which team has the possession 10 cycles ahead, the player who passed the ball checks which team kicked the ball after the intended pass is finished. If the opponent team kicked, the label associated with the observed state is a miss, if our team kicked, is a hit.

In our method we extract information during the agent decision making process, when the agent decides which action to execute, the information about the environment is saved. There are two types of features being logged in a match, a high-level and a low-level feature set.

Our scenario considers high and low-level as related to the level of abstraction of each feature set. The high-level feature set contains prepossessed state

information, listed in Listing 1.1, and the low-level which contains world observations such as teammates and opponent positions and velocities, target pass position and the total duration of the pass.

- **Action Duration** - Total duration of an action, in cycles.
- **Ball Position** - The ball position (x,y) on the field.
- **Target Point** - The objective position (x,y) on the field.
- **Closest Opponent to Target Point** - The distance of the closest opponent to the target point.
- **Opponent Velocity Projection** - The inner product between the closest opponent velocity and the vector of the pass ($passvector = targetposition - ballposition$).
- **Target Teammate Velocity Projection** - The inner product between the target teammate velocity and the vector of the pass.
- **Opponent Reach Cycle** - Time in cycles in which the closest opponent to the target point can reach the target.
- **Opponent Distance to Pass Line** - The distance between the closest opponent to the pass line.
- **Opponent Reach Cycle** - Time in cycles in which the closest opponent to the target point can reach the target.
- **Target Teammate Distance to Pass Line** - The distance between the target teammate to the pass line.
- **Target Teammate Reach Cycle** - Time in cycles in which the target teammate can reach the target point.

List 1.1: Set of high-level features defined to use in pass risk classification.

In our experiments we played 500 games against Robocin 2019, Helios 2018, Helios 2019, and CYRUS 2019 to generate datasets containing possession information. We evaluated two sets of features, a high-level feature set, and a low-level feature set. Additionally we compared the agent evaluation function used as guiding function and a random pass selection. The random pass selection is done by randomizing the pass evaluation value, with that we intended to observe if a random selection would increase the amount of missed passes.

3 Deep Reinforcement Learning Defense

We continued our research on DRL applied to the defense. This year we used a technique similar to Cyrus2019's [11] exploring Deep Deterministic Policy technique (DDPG) [6] and making a new approach building the action and state space for defense situations.

3.1 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 1, 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^*.

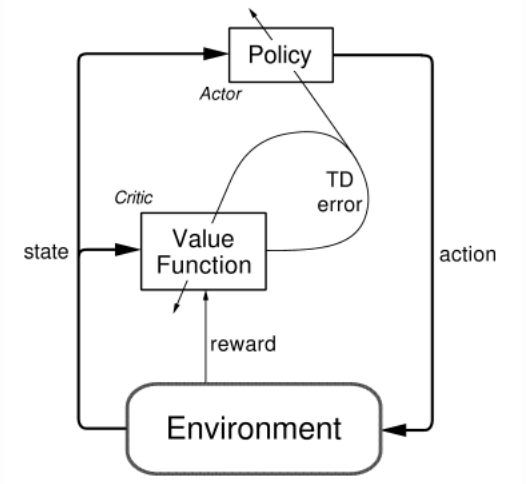

Fig. 1: Illustration of an Actor Critic system. Image from [9].

The input of the Actor is the current state and the output is a single real value representing an action chosen from a continuous action space. The output of the Critic is $V(s)$ where s is the current state.

3.2 State Space

We designed our state space as 11 + 3*teammates + 2*opponents features. The features of our state space are:

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.

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.
$+1$ **Interceptable** - Whether the agent can intercept the ball or not.

3.3 Action Space

As actions we chose: Intercept (interval: [-1, -0.68)) , 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.

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.

4 RCG to CSV Log Extractor

To analyze games in a better file format, we have built an extractor tool that gets RCG files and convert to a CSV files, extracting all the data that the RCG file can provide at the game.

This tool is open to be used at the link (`https://github.com/robocin/rcss-log-extractor`). The features that can be extracted by the RcssLogExtractor are detailed at the repository. It was made in KOTLIN language, so all that is needed is to have a JDK installed to run and it can convert lots of RCG files at the same time.

5 Results

5.1 Machine Learning Ball Possession Algorithm (ML-BPA)

As result of the ML-BPA we observe that the randomness in the pass selection does not lead to an increase in the missed passes, instead, it increases the

amount of passes, as can be seen when comparing Table 1 and Table 2. This can be explained by the pass generation routines. These routines defined by the Agent2D base focus in generating passes that have high probability of reaching the target.

Games	Missed Passes	Right Passes	Total
Robocin2020 vs Helios2018	3681(13.44%)	23708(86.56%)	27389(100.00%)
Robocin2020 vs Helios2019	3966(11.72%)	29876(88.28%)	33842(100.00%)
Robocin2020 vs Cyrus2019	2722(9.63%)	25547(90.37%)	28269(100.00%)
Robocin2020 vs Robocin2020	3698(11.81%)	27608(88.19%)	31306(100.00%)

Table 1: Distribution of missed and right passes selected by the evaluation function.

Games	Missed Passes	Right Passes	Total
Robocin2020 vs Helios2018	9452(8.64%)	99936(91.36%)	109388(100.00%)
Robocin2020 vs Helios2019	9158(8.66%)	96653(91.35%)	105811(100.00%)
Robocin2020 vs Cyrus2019	8386(4.11%)	195707(95.89%)	204093(100.00%)
Robocin2020 vs Robocin2020	9327(8.54%)	99761(91.45%)	109088(100.00%)

Table 2: Distribution of missed and right passes randomly selected.

As the quality and adherence of the generated passes is consistent with the passes on the Table 1, so we used this set to conduct our study.

First, we used Instance Hardness Threshold [8] using Random Forest as classifier to remove the pass imbalance during the training. This method removes the majority class which makes the minority class be harder do classify, improving the quantity of missed passes classified correctly. As we show in Figure 2, we projected the high-level features in a 2D space using PCA and we can see that the undersampling method cleans the dataset, allowing the classifier to create a better decision region.

Then we compared Monolithic, single classifiers, and Multiple Classifier Systems (MCS), combination or selection of classifiers [3]. The use of MCS yield better results than monolithic classifiers, we used METADES as ensemble selection with MLP as base classifier with 10 neurons in the hidden layer, resulting in an increase of 2% to 5% in accuracy and F1 when compared with a MLP with 100 and 50 neurons in two hidden layers.

Meanwhile, the complexity added in the METADES model can lead to a slower decision making, along with our approach to use deep reinforcement learning we choose to compare the Monolithic MLP approach.

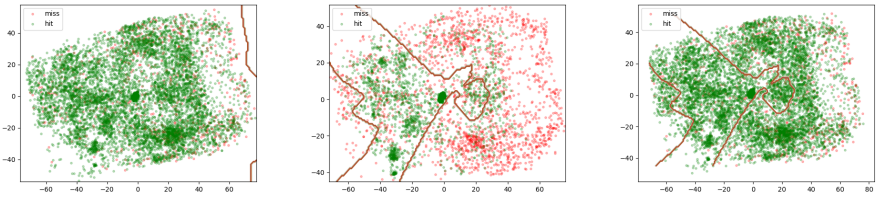

Fig. 2: At left the test instances without Instance Hardness Threshold, in the middle the training set with downsampling and in right the test set with resampling. The green dots are the passes labeled as right and the red as missed. The decision region for both downsampled and non-downsampled are drawn.

Therefore, the results for MLP with 100 and 50 neurons in two hidden layers with the high-level and low-level features is shown in Table 3. As we can see, in 3 of 4 test cases, a high-level feature set leads to better classification results, in accuracy and F1, F1 being more relevant as our classification task is unbalanced.

Our results shows that this classification task is hard to be classified, but our objective is to predict which passes have lower chance to get intercepted or missed.

Games	Accuracy	F1 Score
High-level Features		
Robocin2020 vs Helios2018	**55.29%**	**66.95%**
Robocin2020 vs Helios2019	**50.39%**	**62.20%**
Robocin2020 vs Cyrus2019	44.82%	56.94%
Robocin2020 vs Robocin2020	**52.92%**	**64.90%**
Low-level Features		
Robocin2020 vs Helios2018	52.57%	64.44%
Robocin2020 vs Helios2019	**47.23**%	**58.89**%
Robocin2020 vs Cyrus2019	50.57%	63.58%
Robocin2020 vs Robocin2020	49.58%	61.68%

Table 3: Comparison between High-level and low-level features using MLP as classifier and undersampling.

5.2 Deep Reinforcement Learning Defense

We decided to analyze the following attackers against our DRL Defense: Agent2d, RoboCIn2019, Fractals2019, Helios2019 and Cyrus2019. We could not have results with Fractals2019 due to running issues with Fractals' 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 4: Successful defenses analysis for 3000 episodes. DDPG Defense using RoboCIn2019's goalie.

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. Cruz, R.M., Sabourin, R., Cavalcanti, G.D.: Dynamic classifier selection: Recent advances and perspectives. Information Fusion **41**, 195–216 (2018)
4. Foundation, T.L.: Open neural network exchange (2019), `https://github.com/onnx/onnx`
5. Guennebaud, G., Jacob, B., et al.: Eigen v3. http://eigen.tuxfamily.org (2010)
6. 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)
7. Prokopenko, M., Wang, P.: Gliders2d: Source code base for robocup 2d soccer simulation league (2018)
8. Smith, M.R., Martinez, T., Giraud-Carrier, C.: An instance level analysis of data complexity. Machine learning **95**(2), 225–256 (2014)
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`