ITAndroids 2D Soccer Simulation Team Description 2019

Marcos R. O. A. Maximo, Felipe V. Coimbra, Henrique F. Feitosa, Raphael de V. Nascimento, and Rubens M. G. Aguiar

Aeronautics Institute of Technology,
São José dos Campos, São Paulo, Brazil
mmaximo@ita.br,{felipecoimbra97,hentt30,rapharvn,rubens.miguek}@gmail.com
itandroids-soccer2d@googlegroups.com
http://www.itandroids.com.br/

Abstract. ITAndroids 2D Soccer Simulation team is composed by undergraduate students of Aeronautics Institute of Technology. The team is currently one of the strongest teams in Brazil, having won 1st place 4 times consecutively from 2012 to 2015, and is the current Vice Champion in 2018 Latin American Competition. Moreover, the team has qualified for the last five editions of RoboCup, having participated of four. This paper describes some of our advances in 2018 and our plans for 2019.

1 Introduction

ITAndroids is a competitive robotics team from Aeronautics Institute of Technology reestablished in 2011. The group participates in the following leagues: RoboCup 2D Soccer Simulation (Soccer 2D), RoboCup 3D Soccer Simulation, RoboCup Humanoid Kid-Size, IEEE Humanoid Robot Racing, IEEE Very Small Size and RoboCup Small Size league.

Our Soccer 2D's team, ITAndroids2D, has continuously participated in Latin American Robotics Competition (LARC) and Brazilian Robotics Competition (CBR) since 2011. Moreover, ITAndroids 2D competed in RoboCup in 2012, 2013, 2015, 2016, 2017 and 2018. The team also qualified for RoboCup 2014, but unfortunately it was not able to attend to the competition. Our results in these competitions are represented in figures 1 and 2.

Lack of continuation and documentation of the project and the spreading of the team towards other field made ITAndroids2D slow down its improvements. This can be seen from the leagues results from 2015 to 2017. However, ITAndroids 2D recovered in 2017, after a complete restructuring of the project [3]. As a result, we won 9th place at RoboCup2018, our best absolute place in the competition.

In this paper, we describe our advancements during 2018: a great use case of our Possession Automaton [3] (Section 2) and the restructuring of our decision making by use of Behavior Trees (Section 3). In the Section 4 we describe what is the RoboCup Soccer Simulation Server, how is its functioning and how to use it.

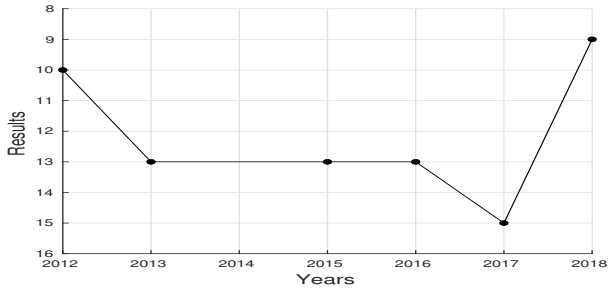

Fig. 1. Results of ITAndroids2D in past RoboCup competitions.

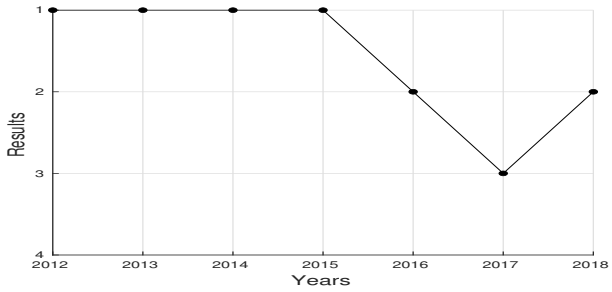

Fig. 2. Results of ITAndroids2D in past LARC competitions.

2 Use Case of the Uncertain Possession Automaton: Backpass Extinction

Using the uncertain possession automaton [3], one can easily find out which team retains possession of the ball, which is not a trivial task, considering the noises and incomplete information provided by the server. Several new strategies can be elaborated for the team if the players have accurate information about the possession of the ball, here it is described the use of the possession automaton to avoid backpasses.

To solve this problem, a new uncertainty coefficient [3] that measures the reliability of the information of the goalkeeper was created. It defines how much information should be considered for the analysis done by the goalkeeper to determine the team that keeps possession of the ball. Information containing little noise is preferable to information with higher noise.

The reason for creating this new coefficient is to distinguish the goalkeeper from other players. Since the goalkeeper is further away from the ball compared to the remaining players, information of the goalkeeper usually contains more noise and should be treated differently from other players in order to avoid false positives and false negatives in determining the possession of the ball.

This coefficient was improved with testing and causes the goalkeeper to disregard information when the ball is too far away from him and to consider when the ball is close. The net effect is more accurate information about the possession when the ball is specifically in the defensive half of the field, the situation that mostly matters to him.

Finally, to solve the problem of backpasses, it was determined that the goalkeeper should not emit catch commands if the possession of the ball is with his team. However, if the ball is going towards the goal, the goalkeeper uses another command to move the ball away from the goal, for instance, the clear ball behavior.

To measure results, 120 matches against the ITAndroids2017 team were carried out, the results are shown in Table 1.

Table 1. Number of backpasses in 120 matches

	ITAndroids2017	**ITAndroids2018**
# of backpasses	46	1

Therefore, this shows a reduction of nearly 98% in the number of backpasses, which shows a drastic reduction of the problem. Moreover, statistics were made of wins, losses and draws against the ITAndroids2017 team, before and after the possesion automaton improvement. The results are shown in Table 2, that express good team improvements.

Table 2. Performance of ITAndroids2018 against ITAndroids2017

	Before modification	**After modification**
Wins	37.5 %	45.8 %
Draws	26.7 %	34.2 %
Losses	35.8 %	20 %

3 Behavior Trees: Development Sugar for Behavioral Robotics

In behavioral robotics, the robotic agent is modeled focusing in adaptability and reactivity. Complex objectives can be accomplished by breaking in smaller subobjectives and creating Behaviors to structure how to do things with high level decisions.

There are several ways to organize these decisions. The simplest and most common is to use Finite State Machines (FSM), however, FSM do not scale with growing number of states.

Behavior trees are tree structures that permits describing generic decision making by correctly combining internal Control Flux Nodes with Action Leafs and Decision Leafs. After the tree is built, the decision making is executed by just triggering transversal from the root.

These general flow control nodes make Behavior Trees highly modularized and prompt for code reuse. Nodes and entires subtrees can be substituted to achieve high adaptability to situations. Its natural granularity permits us to create short behaviors without writing additional state transition logic, promoting easy reactivity [5].

Behavior Trees can generalize popular decision structures like FSM, HFSM, Decision Trees and Teleo Reactive programs [6] [10] but with principles appropriated for good software development practices.

We have started a simple and generic library for creating behavior trees in C++. It achieves general use structure by intense use of templates and has permitted us to reduce the project complexity. Figure 3 shows sample usage of our library to build the tree shown.

```
BTBuilder builder;
BT defenseBT, attackBT, goalieBT;

// Goalie Behaviour Tree
goalieBT
        = builder.
            compositeRoot<BT_Selector>().
                composite<BT_Sequence>(). // Defend with hands
                    leaf<BT_DecisionLeaf>(new GoalieShouldCatch(), "GoalieShouldCatch").end().
                decorate( new SuccessOnlyPolicy() ).
                    leaf<BT_BhvLeaf>(new Bhv_GoalieCatch(), "BhvGoalieCatch").end().
                end().
                composite<BT_Sequence>(). // Defend with feet
                    leaf<BT_DecisionLeaf>(new SelfIsKickable(), "SelfIsKickable").end().
                decorate( new SuccessOnlyPolicy() ).
                    leaf<BT_BhvLeaf>(new Bhv_GoalieClearBall(), "BhvGoalieClearBall").end().
                end().
                composite<BT_Selector>(). // Just move
                    composite<BT_Sequence>().
                        leaf<BT_DecisionLeaf>(new GoalieShouldChaseBall(), "GoalieShouldChaseBall").end().
                    decorate( new SuccessOnlyPolicy() ).
                        leaf<BT_BhvLeaf>(new Bhv_GoalieChaseBall(), "BhvGoalieChaseBall").end().
                    end().
                    leaf<BT_BhvLeaf>(new Bhv_GoalieBasicMove(), "BhvGoalieBasicMove").end().
                end().
            end().
        end();
```

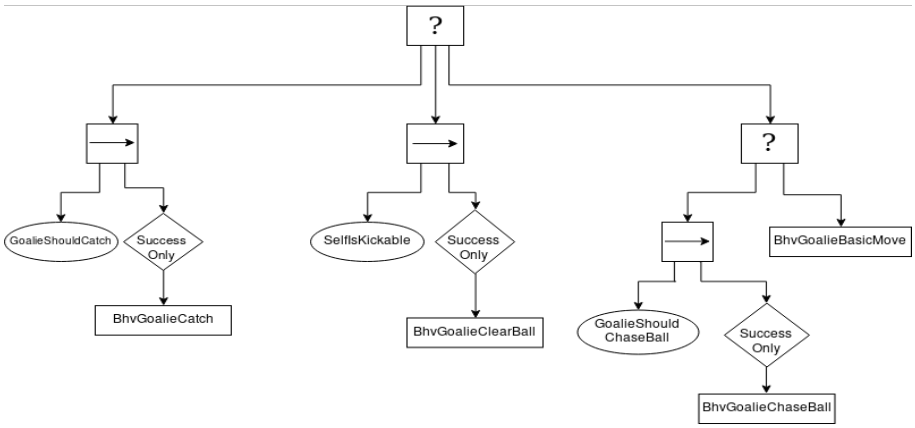

Fig. 3. Code for building our goalie behavior tree and the tree built.

4 RoboCup Soccer Simulation Server Document

The RoboCup Soccer Simulation Server (rcssserver) is a program that controls the simulation of the soccer matches based on server-client communication, where each player is a client that sends and receives information from the server.

The communications is done via UDP/IP protocol with string messages. A player/client connects with the server sending a *init* message, if the conection succeeds it receives information about match parameters and heterogenous player types. The simulation occurs in cycles, where in each cycle the player send commands, e.g, *move, kick*, and receives noisy information about the match state, e.g., player positions, ball position.

An important option that can be passed to the server is to start the game in the synchronized mode, for this you must start the server with the following command:

```
$ rcssserver server::synch_mode=true
```

In this mode, a cycle changes immediately after all the clients send and receive the messages, which makes the game faster because the players' latency is normally less than the default cycle duration of 100ms.

There are two files that contain the parameters used by the server during startup, *server.conf* and *player.conf*, which by default are located inside the *.rcssserver* folder in the user's home directory.

In these files, you can set parameters such as the number of different heterogeneous players available (*player::player_types*), number of cycles of each half of the match (*server::half_time*), and opt for also receiving information without noise with the options (*server::fullstate_l*) and (*server::fullstate_r*), among others.

Let us highlight here among all the options of the server, the possibility of running simultaneous games, for this must be passed as parameters the connection ports for the server and the coach, e.g.:

```
$ rcssserver server::port=7000 server::coach_port=7001 \
server::olcoach_port=7002
```

and make players and coaches connect to these ports too.

The server has no built-in UI to allow spectators to watch the match. However, there is also the possibility of connecting a special type of client, the monitor. In official competitions, the monitor used is the *rcssmonitor* [9].

5 Conclusions and Future Work

This paper describes some of our efforts during the year of 2018 and shares some of our knowledge about the rcssserver. Other advances made during 2018 and not mentioned in this work include making our man-marking strategy [3]

robuster to noise and reoptimizing action chain parameters with Particle Swarm Optimization [2] but higher processing power.

We are currently developing a general purpose platform of Deep Reinforcement Learning to develop Neural Behaviors. We are also exploring the combination of CMA-ES algorithms to optimize heterogeneous type selection and studying the possibility of implementing state-of-art RFS filters to improve opponent tracking as has been recently investigated in RoboCup [8].

6 Acknowledgements

We would like to acknowledge the RoboCup community for sharing their developments and ideas. Specially, we would like to acknowledge Hidehisa Akiyama for agent2d [1], librcsc [7], soccerwindow2 [11] and fedit2 [4] software. Current team members are also grateful to previous ones who have made great contributions to ITAndroids 2D. Finally, we thank our sponsors Altium, Intel Software, ITAEx, Metinjo, Micropress, Polimold, Rapid, Solidworks, ST Microeletronics and Virtual Pyxis. We also acknowledge Mathworks (MATLAB), Atlassian (Bitbucket) and JetBrains (CLion) for providing access to high quality software.

References

1. agent2d-3.1.1, 2012, online, avaliable at: http://pt.sourceforge.jp/projects/rctools/downloads/51943/agen t2d-3.1.0.tar.gz/, consulted on December 2018.
2. Mello, F., Ramos, L., Maximo, M., Ferreira, R., Moura, V.: ITAndroids 2D Team Description 2012 (2012)
3. Lema L., Coimbra F.: ITAndroids 2D Team Description 2018 (2018)
4. fedit2-0.0.1, 2017, online, avaliable at: https://osdn.net/projects/rctools/downloads/68531/fedit2-0.0.1.tar.gz/, consulted on December 2018.
5. Colledanchise, Michele, and Petter Ögren. Behavior Trees in Robotics and AI: An Introduction. (2018).
6. Michele Colledanchise, and Petter Ögren. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture and Decision Trees. In IEEE Transactions on Robotics (TRO) 2016.
7. librcsc-4.1.0, 2011, online, avaliable at: http://pt.sourceforge.jp/projects/rctools/downloads/51941/librcsc-4.1.0.tar.gz/, consulted on December 2018.
8. Cano P., Ruiz-del-Solar J. (2017) Robust Tracking of Multiple Soccer Robots Using Random Finite Sets. In: Behnke S., Sheh R., Sarıel S., Lee D. (eds) RoboCup 2016: Robot World Cup XX. RoboCup 2016. Lecture Notes in Computer Science, vol 9776. Springer, Cham.
9. rcssmonitor-15.2.1, 2017, online, available at: https://github.com/rcsoccersim/rcssmonitor/releases, consulted on December 2018.
10. Michele Colledanchise, and Petter Ögren. How Behavior Trees Generalize the Teleo-Reactive Paradigm and And-Or-Trees. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2016.
11. soccerwindow2-5.1.0, 2011, online, avaliable at: http://pt.sourceforge.jp/projects/rctools/ downloads/519 42/soccerwindow2-5.1.0.tar.gz/, consulted on December 2018.