ITAndroids 2D Soccer Simulation Team Description Paper 2022

Davi M. Vasconcelos, Gustavo F. Gottschild, Marcos R. O. A. Máximo, Nando Ferreira Farias, Vinícius F. Almeida, Vinícius P. D. Araújo, and Yuri G. Araújo

Aeronautics Institute of Technology,
São José dos Campos, São Paulo, Brazil
{davi.muniz41,gustaferraresi189,nandoffarias12,vinifreitas.d.a,viniciusdepadua6,yurigama.t23}@gmail.com,mmaximo@ita.br
itandroids-soccer2d@googlegroups.com
http://www.itandroids.com.br/

Abstract The ITAndroids 2D Soccer Simulation team is composed by undergraduate students of the Aeronautics Institute of Technology. The team is currently one of the strongest teams in Brazil, having won first place 4 times consecutively from 2012 to 2015, Vice Champion in 2018 and was the Champion of the 2019 Latin American Competition. Moreover, the team has qualified for the last nine editions of RoboCup, having participated in eight. This paper describes some of our advances in 2021 and our plans for 2022.

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, 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, ITAndroids 2D, has continuously participated in Latin American Robotics Competition (LARC) and Brazilian Robotics Competition (CBR – acronym for *Competição Brasileira de Robótica*) since 2011. Moreover, ITAndroids 2D competed in RoboCup in 2012, 2013, 2015, 2016, 2017, 2018, 2019 and 2021. The team also was qualified for RoboCup 2014, but unfortunately it was not able to attend the competition. Our results in these competitions are represented in Table 1.

Lack of continuation and documentation of the project and the spreading of the team towards other fields slowed 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 [1]. As a result, we won 9th place at RoboCup2018, our best absolute place in the competition.

During 2020 and 2021, our progress slowed down significantly due to the Covid-19 pandemic. Even then, it still can be said that we made some advancements during that troublesome year. In this paper, we describe these advancements.

Year	RoboCup	LARC
2012	10th	1st
2013	13th	1st
2014	—	1st
2015	13th	1st
2016	13th	2nd
2017	15th	3rd
2018	9th	2nd
2019	13th	1st
2020	—	4th
2021	11th	5th

Table 1: Placement of ITAndroids 2D in past RoboCup and LARC competitions.

2 Previous Works

Our code base uses agent2d [2] as base team. Since the year 2012, we have focused on improving mechanisms already present in agent2d framework. We have improved the action chain evaluator with Particle Swarm Optimization (PSO) [3]. Furthermore, we have developed heuristics [1] to increase attack and defense performance: type of formation (attack or defense) selection based on probability of scoring a goal, field evaluator selection based on opponent team, and defender optimal marking in opponent attack situations. Many early improvement ideas were inspired by Team HELIOS [4] and Team Nemesis [5].

The team proposed in 2018 a novel technique to determine the in-game ball possession [6]. The ball possession information without noise is valuable for any team, since it can be useful to create dynamic behaviors in players. We have used a Finite-State Machine called Possession Automaton that takes into account the current and the last game situations to infer the ball possession. The game situations are determined by InterceptTable simulations, the ball possession estimator in agent2d [2]. Since we do not estimate ball possession based on a single game cycle, we have obtained a classification accuracy 18% higher than the default estimator of the base team.

Lately we have experimented Deep Reinforcement Learning (DRL) techniques to improve the goalkeeper defense in penalty situations [7]. After five training experiments using Proximal Policy Optimization (PPO) algorithm, we have achieved a penalty defense rate of 40% against agent2d [2], twenty percent higher than the base team rate.

Our current efforts focused on the development of tools to perform game log analysis. An in-depth investigation of game logs brings improvement ideas and guides future work decisions. In fact, RoboCup 2D teams have presented tools to analyze team performance. Team RoboCIn has developed a graphical user interface that has a group of analyzed features for a single game log (e.g., faults position, player stamina) [8]. Furthermore, an automatic evaluation system

of soccer matches was implemented [9], which provides statistical performance indicators based on a large number of games.

3 Soccer Reporter: a stout tool for statistical analysis

Statistical analysis of match results is a pretty simple, yet highly cost-effective and sturdy way to improve the performance of any RoboCup team. Moreover, it is an utterly necessary process so that changes proposed to a team can go beyond mere theoretical models and their true impact on performance can be accurately appraised. To implement such evaluation, several techniques can be used, such as p-value, null hypothesis testing, and Bayes factors.

As our team has only a few members, focusing on creating a unified tool containing all the performance evaluation metrics for statistical analysis is the most feasible solution. Inasmuch as all match events are stored in the log files generated by the rcssserver, such application will be essentially a log analyzer. Besides, seeking to achieve the greatest long-term benefits, the solution chosen must attend the following requirements:

- **Be as easy to maintain as possible**: since only a handful of people will be maintaining this software, the application has to be modular, internally decoupled and developer-friendly. In addition, with speed of development in mind, ready-made solutions are likely to be the best alternatives and, as will be explained ahead, compiler-like front-ends should be preferred. Thus, to solve this issue it is enough to choose a well suited parser generator. Among the many options available, *ANTLR* (ANother Tool for Language Recognition), a tool used to develop compilers' front-ends, stands out because it is easy to use and produces human-readable output, as well as has active support and industrial-level language processing capabilities.
- **Be user-friendly**: This is necessary in order to speed up training within ITAndroids 2D and to avoid mistakes during match analysis. This could be done initially with an easy-to-use CLI (Command Line Interface) developed using *Click*, a state-of-the-art Python package for creating CLIs, which is preferred over other tools like *Argparse* and *Docopt* due to its clean syntax and its superior capabilities. Thereafter, a GUI (Graphical User Interface) could be implemented in order to increase usability.
- **Allow flexible feature analysis**: since statistical processing of all match parameters will not be necessary all the time, aiming the efficient use of computational power, it is clearly necessary to allow the user to pick which match features will be analyzed, and how. The most feasible ways to set these settings are: CLI options, a configuration file and a GUI. These three solutions probably would be implemented in such order due to development complexity issues. The first one is the easiest to implement, but repetitive and laborious, the second one is the the most promising of them all because it is easier to develop than the GUI alternative, as well as being the least

repetitive and the most robust choice. Finally, the last one is the most user-friendly option, but, as it is the most difficult to implement, its development could be, if necessary, left aside.
– **Be robust**: as long as protocol changes can be rather problematic, the application's front-end must be stout. That is, it has to work in as many input configurations as possible. A front-end model with the desired stoutness already exists: the one used by compilers. Therefore, building the Soccer Reporter's front-end using *ANTLR* would enable both a much faster development and a far more robust statistical tool.

Unfortunately, due to the COVID-19 pandemic, development of the project was slowed down, and the only parts finished were the grammars of the client's lexer and parser, which were written in *ANTLR*.

4 itas2d-window: A GUI to visualize 2D soccer matches

Work has begun on the itas2d-window application, which is a desktop tool that connects to rcssserver to visualize matches between 2D soccer teams, provide real-time debug information, and much more. Currently, the tool is at a very early and experimental stage — so early that there can be no meaningful demos to show pictures of the matches. Even then, there are some design choices and other considerations that might be useful knowledge to anyone interested in the topic. Some of these will be elucidated here.

4.1 The GUI library

To build a desktop application with an interactive user interface, there currently is a plethora of fully-fledged, robust GUI libraries and frameworks, such as Qt, wxwidgets, GTK+, Flutter, Electron, and many more. Some are C++-based and run natively on desktop systems(such as Qt and wxwidgets), whereas others run in a virtual machine or environment, to try and be much more platform-independent(such as Electron, which runs on an instance of Chromium, and is capable of not only running on major desktop platforms, but also on mobile). Though this is not, by any metric, an exhaustive analysis of such libraries, an attempt will be made to explain the pros and cons of each of them:

– **wxwidgets and GTK+**: Though they are different projects, it can be said that their purpose (or use-case) is very similar: making desktop apps with look, feel and functionality that is as native as possible. Also, both have their APIs written in C++ (though there are bindings for other languages, such as python). With that in mind, using either of these libraries is very advantageous if having multi-platform support (as in macOS, windows and linux) with native look-and-feel is a must (it should be noted that, technically, GTK+ fakes the operating system's style through it's theming API, though), and the developer has experience working with the C++ language which, arguably, is not the most well-suited to creating and designing UIs.

- **Flutter**: This is a relatively new UI toolkit to create native applications that can be deployed to the web, mobile or to desktop. Its main feature, as claimed by Google, is leveraging the power of the Dart language to enable fast development times, with an expressive syntax and native performance. Its main disadvantage, as of the time of writing, is its instability — for example, many dart packages break due to the rapid evolution of the language and the ever-evolving API. Also, even though it supports desktop, its main target is mobile applications.
- **Qt**: Qt is a GUI library written in C++ which has a very strong emphasis on the "Code once, deploy everywhere" philosophy. Qt apps can run on mobile, desktop, and even embedded devices. Their C++ API does not even assume features of C++11 (which, as of 2021, was approved a decade ago), such as type inference, range-based for loops, initializer lists, smart pointers etc. Instead, many macros and language extensions are provided, and Qt developers have to use qmake, which is part of the custom build system that Qt provides. Also, from Qt5 onwards, the Qt Quick API started taking space as the most supported way of developing Qt apps (at the expense of the much older Qt Widgets, which is exclusively C++-based and only targets desktop). This newer API works through a declarative language called QML to create and compose UI elements, and can have embedded javascript code to provide logic. All of this increases the technical complexity of development with the library. Finally, the documentation is thorough, but monolithic in its size and scope, which only adds to the frustration of working with Qt.
- **Electron**: This is a framework that enables the developer to create desktop applications with JavaScript, HTML, and CSS. Therefore, its main use-case is to easily make desktop applications with web technologies. From a development point of view, it can be argued that these technologies are state-of-the-art in terms of ease of use and development speed, empowering web developers to create beautiful UIs with great user experiences with simple, declarative languages mixed with imperative logic (if even standard CSS3 and vanilla JavaScript are very capable of doing these tasks, much more can be done with react, bootstrap, SASS and so on...). The main disadvantage, as mentioned before, is that this framework depends on Chromium which, in layman's terms, means that it packages an entire web browser with itself. This, evidently, adds unnecessary overhead to an otherwise sleek and modern application.

4.2 Our choice for the library

After some consideration, Qt was initially chosen as the library for development. The rationale was pretty much as follows:

- Qt is a robust and well-established library (20+ years old)
- It has already been used in other parts of the ITAndroids team
- Qt4 was used to develop our current GUI (soccerwindow2 [2]) which — given some tweaks and ajustments — would be suitable to our needs

- Qt is not just a GUI library: it also provides networking, unit testing, internationalization, multimedia and much more
- Qt runs natively on Ubuntu (our main target platform)
- It might not have a native look-and-feel (unless the user is on KDE, which is based off of Qt), but we're developing an app for debugging purposes, not for "great" or "beautiful" user experiences
- Qt has better documentation and an easier learning curve than wxwidgets or GTK+ (though that's debatable)

After a few months of learning the Qt Quick API, struggling to understand parts of the documentation, and getting frustrated with the complex macros, object annotations and unintuitive build system, we opted to go with the much simpler *dear imgui* library. In the next section, we'll talk more about why we chose this library.

4.3 Dear Imgui

Before going into the details of the dear imgui library, we will talk about the immediate-mode GUIs (*imgui*, as opposed to retained-mode GUIs).

Immediate-mode GUIs try to address the problem of complexity that is common in designing UIs with retained-mode libraries. This complexity often comes as a consequence of decentralized state, which requires widgets to hold callbacks, IDs, mouse position and all sorts of other (possibly redundant) references to the state of the UI. Also, as they are decentralized, working with these retained-mode GUIs usually means the code will be all over the place, because different logical parts affect the interface differently. ImGuis, in contrast, are stateless and have a strong binding to their respective logical parts. The key idea behind an ImGui is that everything is done within an update loop, just like with immediate-mode graphics[1].

We chose dear imgui because:

- Being a very tiny library(only a few C++ source and header files), it has very minimal overhead, even compared to the well-performant Qt library
- It enables very fast prototyping and development, without needing unnecessarily complex architectures such as those present in Qt
- It easily permits the usage and integration of more standard build tools , such as CMake
- Developing with intellisense, and enabling static code analysis tools such as clang-static-analyzer consumes less system resources than with Qt's monolithic tree of libraries
- We can fearlessly use more modern C++ features, as opposed to Qt-style C++, which is based on the extensions that the library provides to the language

[1] Incidentally, the dispute between retained and immediate-mode graphics was the idea behind the name of the ImGui paradigm, according to one of its popularizers, Casey Muratori[10].

– As part of the front-end is developed in OpenGL, any GUI library that supports it will suffice. Therefore, using either Qt or *dear imgui* is irrelevant with regards to this aspect(actually, it's easier to integrate dear imgui with OpenGL), and we can migrate to Qt later, if it is ever necessary

4.4 Design choices

Although much of what was done with Qt was eventually scrapped, some design choices for the UI still remain, such as:

– The sidebar
– The header and footer
– Minimized state for the side panels
– The timeline
– Ability to select entities and check out their properties

On Figure 1, we can see a mockup which illustrates most of these design choices for our GUI.

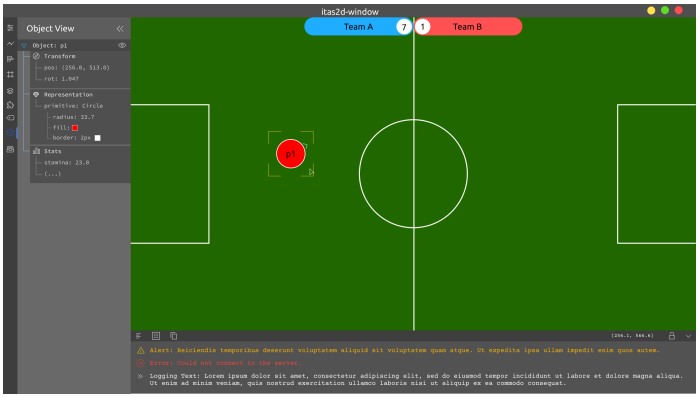

Figure 1: Mockup of the main window

5 Conclusions and Future Work

This paper presented the most recent efforts of team ITAndroids 2D. Among the several improvements that will be further made, the ones most worthy of note are the putting into practice of the plans mentioned in Section 3, and the advancement of the itas2d-window to a usable state for our team members, with increasingly better debug and statistical information, along with real-time logging. In addition to those next steps of the Soccer Reporter project, its back-end could, due to complexity issues, in a first instant, use a simpler statistical technique like Bayesian maximum *a posteriori* estimation using a conjugate gradient

method and, later, a more sophisticated one, such as a Bayesian network. Finally, the itas2d-window GUI would have to be integrated with the statistical analysis provided by the soccer reporter tool.

6 Acknowledgements

We would like to acknowledge the RoboCup community for sharing their developments. In particular, we would like to acknowledge Hidehisa Akiyama for agent2d, librcsc, soccerwindow2, and fedit2 [2]. We would also like to thank: Terrence Parr for his *ANTLR* [11] compiler-compiler, Omar Cornut for his *dear imgui* [12] library, and the Pallets Project for the *Click* [13] library. Finally, we acknowledge our sponsors: Altium, CENIC, Intel Software, ITAEx, Metinjo, Micropress, Polimold, Rapid, ST Microeletronics, and Wildlife.

References

[1] Felipe Coimbra et al. "ITAndroids 2D Team Description 2017" (2017).
[2] Hidehisa Akiyama and Tomoharu Nakashima. "HELIOS Base: An Open Source Package for the RoboCup Soccer 2D Simulation". In: *The 17th annual RoboCup International Symposium*. July 2013.
[3] Fábio Mello et al. "ITAndroids 2D Team Description 2012" (2012).
[4] Hidehisa Akiyama and Hiroki Shimora. "HELIOS2010 Team Description" (2010).
[5] Mehrab Norouzitallab et al. "Nemesis Team Description 2010" (2010).
[6] Felipe Coimbra and Lucas Lema. "ITAndroids 2D Team Description 2018" (2018).
[7] Diego Fidalgo et al. "ITAndroids 2D Soccer Simulation Team Description Paper 2020" (2020).
[8] F. N. Pereira, Mateus F. B. Soares, and Edna N. S. Barros. "A Data Analysis Graphical user Interface for RoboCup 2D Soccer Simulation League". *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)* (2020), pp. 1–6.
[9] Ryota Kuga, Yudai Suzuki, and Tomoharu Nakashima. "An Automatic Team Evaluation System for RoboCup Soccer Simulation 2D". In: *2020 Joint 11th International Conference on Soft Computing and Intelligent Systems and 21st International Symposium on Advanced Intelligent Systems (SCIS-ISIS)*. 2020.
[10] Casey Muratori. *Immediate-Mode Graphical User Interfaces*. https://caseymuratori.com/blog_0001. 2005.
[11] Terrence Parr. *antlr4-9.1*. https://github.com/antlr/antlr4/releases. 2021.
[12] Omar Cornut. *dear imgui*. https://github.com/ocornut/imgui. 2021.
[13] *click-7.1.2*. https://github.com/pallets/click/releases. 2020.