ITAndroids Soccer3D Team Description Paper 2020

Bruno Benjamim Bertucci, Dicksiano Melo, Guilherme Nascimento de Oliveira, Isabelle Ferreira, Juan Freire, Lucas José Velôso de Souza, and Marcos Máximo

Aeronautics Institute of Technology,
São José dos Campos, São Paulo, Brazil
{bertucci.b.bruno,dicksianomelo,guilhermeoliveira5001
isabelle.ferreira3000,juanfdantas,lucas.jose.veloso.de.souza
maximo.marcos}@gmail.com
http://www.itandroids.com.br/

Abstract. ITAndroids is a robotics competition group associated to the Autonomous Computational Systems Lab (LAB-SCA) at Aeronautics Institute of Technology (ITA). ITAndroids is a strong team in Latin America. Our 3D Soccer Simulation team started its activities in 2012. Currently our code is written in C++. This paper guides the reader through the most important features of our code and work tools developed, with a focus on the most recent developments.

1 Introduction

ITAndroids is a robotics research group at Aeronautics Institute of Technology. As required by a complete endeavor in robotics, the group is multidisciplinary and contains about 40 students from different undergraduate engineering courses. To motivate our research, we participate in robotics competitions. In the last 7 years, we achieved good results in competitions, especially in Latin America.

This paper describes our development efforts in the last years and points out some improvements we want to implement in the near future. Sec. 2 describes our team's code structure. In Sec. 3, our implementation of parameter monitoring is described. Sec. 4 explains our role assignment system for positioning. Sec. 5 shows our strategy and points our robot navigation method. Sec. 6 presents a framework that is able to mimic a reference motion and optimize it towards a task. Sec. 7 details our team's use of the BUMPS algorithm. Finally, Sec. 8 concludes and shares our ideas for future work.

2 Code Structure

The code has been planned and divided into several modularized parts so that each part can be separated from the others with ease. Basically, it is divided into 7 layers. Each of them will be described in this section.

2.1 Communication

It is the layer that directly connects with the server, in order to receive messages and send messages through sockets. This layer receives and sends a string as described in the server's website, following the protocol described in the Simspark documentation.

2.2 Perception

The Perception layer is responsible for turning the strings received by the Communication layer. This layer parses the string and converts it into a tree. The layer then iterates over the created tree and creates new objects (perceptor objects) from it, so that the agent can have new information each new loop. Each perceptor is as described in Simspark's website.

2.3 Modeling

Modeling basically models the world state. It executes probabilistic stochastic filters to determine where the robot is in the field, and where the other agents are. Modeling is divided in two parts, a World Model and an Agent Model.

Agent Model The Agent Model models information related to the robot itself. It computes transformation matrices which are used to transform vision observations from the camera coordinate system to a coordinate system on the ground.

World Model The World Model is responsible for modeling world states such as game state, time, and position, so that this information can be used by Decision Making. It runs the Localization algorithm in order to estimate the robot's position.

2.4 Decision Making

Decision Making is a layer that gives each agent a role. The roles assigned dictate the movements the agent should take in order to successfully follow a determined strategy. One agent cannot change its role, but, that role must be able to integrate all the possible behaviors the agent has available, e.g. a regular agent receives the AttackingRole, while a goalie receives a GoalieRole. Decision Making is also responsible for calling the Marking System to assign the navigation targets for each agent. After the Behaviors are executed, the action requests generated by them are updated.

Behaviors Behavior is a set of what the agent can do in order to change its own state. It is a set of instructions that goes from high to low level of abstraction, in order to make the agent follow its strategy.

Each behavior can use other behaviors for a more abstract level of problem solving. For example, Attack behavior can call upon NavigateToPosition so that it does not have to be reimplemented in Attack. Each behavior creates an action request, that is a communication interface with the Control layer, and it is how the agent knows which specific action to take.

The Behavior layer (it is not exactly a layer, since it is part of Decision Maker) has two parts, a part that is a data structure called BehaviorFactory, and a structure called Behavior. A BehaviorFactory stores all behaviors, and each behavior has access to all other behaviors through the Behavior Factory.

2.5 Control

Control is the layer that gets the requests from behavior and changes it into more concrete things. For example, it takes a walk request created from one of the behaviors and converts it into joints positions. It is where the movement algorithms are implemented.

2.6 Action

The Action layer is responsible for converting all information that the agent has created and that it wants to send to the server to a string in a way that the server can recognize. Then, this string is sent to the server through the Communication layer.

3 Parameter Monitoring

Taking in consideration that most of the robotics field heavily relies on empirical knowledge, in a very noisy world where robot states may deviate from what is expected at any time, and yet most of the problems do not even have a well-defined solution, but rather heuristics that can be adjusted to come as close as possible to optimal solutions, a lot of parameters in a robot's code might be constantly tuned through its development, as well as many states should be prone to easy monitoring and debugging.

Therefore, the constant changes in the robotics environment demand real-time user program interaction, which should be fast and not delay the regular behavior of code. Added to this, a good parameter monitoring system should not only store a variable value but a whole set of meta-information about it, such as a range of possible values and a description.

By means of open source collaboration, our team was able to integrate the open code from the Rhoban Input Output Library (RhIO)[7], an API that already had many features implemented, including a tree structure that stores

dynamical parameters nodes, as well as other means of input/output nodes, as leaves of directory path nodes, a shell client that provides many commands for quick access through I/O nodes, parameter setting, checking, loading or saving and even a Gnuplot binding which allows real-time values monitoring and a NCurses based interface with line sliders that can tune parameters.

As a proof of concept of the tool in the 3D Simulation domain, our team implemented a drawing selection interface for Roboviz [8] that allows choosing which of our drawings are sent to the drawing port, as well as a joystick-controlled agent, that can ease tests on agents mobility and behavior at specific field locations. There were also some additions to the shell client, such as multiagent portability and a hold command to apply multiple parameter setting commands at once.

4 Dynamic Role Assignment

4.1 Position Model

The Delaunay Triangulation algorithm calculates our player's positions and generates a formation set of agents for every single position of the ball [4]. These positions serve as reference points for an agent to where they should be if there is not an opponent in a dangerous position just according to the ball's position at that moment. The Dynamic Role Assignment [5] idea is that the agents can communicate with themselves, decide the best lineup at that moment and assign the Delaunay Triangulation positions to the team, one for each agent. And do it all dynamically, sending and receiving messages in every communication cycle.

The lineup consists of eleven positions and each agent, using the data received in the World Model, is able to determine which agent should be assigned to each Delaunay Triangulation position based on the agents' position. Each position is assigned to a number 1-11, just as the agent's uniform number. So, the agent generates an array and assigns the number of the Delaunay Triangulation position to each player in its team.

4.2 Communication between Agents

The objective is to send a vector containing each position assigned to each other agent in the team. Each agent can send and receive messages from the server in every cycle of 20 ms, with the message's size limited to 20 bytes (160 bits). The problem is that every integer occupies 4 bytes, and the team has 11 players. So, the vector would have occupied 11*4 = 44 bytes of memory, a lot more than the limit. The solution used was to use the base64 ASCII encoder [6], a compression message encoder that creates a bijection between the vector and other structures which occupy less memory. Then, the agents would receive the encoded message and, using the same bijection, be able to decode and recover the message vector. The method proved itself to be very efficient, reducing the size of the role assignment vector form the original 44 bytes to only around 110-120 bits.

4.3 Voting System

When the agents have all sent their respective role assignment vectors to each other (they may differ between agents since their world models are different), they acknowledge the position which was assigned more times to each respective agent. Then, the team's role assignment vector would be filled based on the assignments.

4.4 Marking System

The marking system is a sequential process used to mark opponent agents who are offensively dangerous during the match.

The system performs three steps: it decides the players that will be marked, defines roles to mark these players and, finally, uses the Role Assignment system to designate agents for those defined roles.

Heuristic-based marking system: To decide which opponents are to be scored, a heuristic method is used based on the following conditions about the opponent. If it is:

- Close enough to take a shot on goal.
- Not the closest opponent to the ball.
- Not too close to the ball.
- Not too far behind the ball.

Machine learning based marking system: The main objective was to add human knowledge of which opponents are in a dangerous position in a match and should be marked in order to help improve the team's defense. Qt, a framework for developing C++ graphical interfaces, was used to create a data acquisition system based on human knowledge for marking opponents. This acquired data then fed a supervised learning algorithm (a neural network implemented in Keras, a framework for the development of Python neural networks), finally training a marking model.

For the training and testing of the net, we used a dataset of 500 frames, referring to 24 matches. Of this set of frames, 300 (60% of the total) were chosen for network training, 100 (20% of the total) for cross-validation set. Meanwhile the last 100 (20% of the total) were chosen for the test set. Network training obtained a model with an accuracy of 89.15% in the training dataset, 85.57% in the cross-validation dataset, and 86.92% in the test dataset. It can be noted that there were no problems of overfitting or underfitting, being a satisfactory result in relation to accuracy.

It is still necessary in future work to apply this strategy in the code, to verify the results of these changes also in real matches against opponent teams.

After that, for both marking approaches, a set of formation roles must be selected to mark the chosen opponents. For this, marking roles positions are

calculated as the position 1.5 meters from a marked opponent along the line which connects that opponent to the center of our goal. The formation positions to be replaced are the closest to each marking position, and their selection is done by using the Hungarian algorithm, which calculates the minimum sum of distances between the previous forming positions and the marking positions.

5 Strategy and Decision Making

5.1 Set Plays

As in a soccer game, during a simulated game there are situations where the ball is stopped. When this happens, the team with the ball has a certain amount of time to make a move, while the other team cannot approach more than a certain distance from the ball. For some of these situations, we have formulated different set plays, namely: Own Goal Kick, Opponent Goal Kick, and Own Corner Kick.

5.2 Penalty State

For penalty kicks, a flag was introduced to create small changes in order to create better chances of scoring goals. The flag was introduced as a unique agent number. When this number is called through the penalty kicker's initialization script, an agent without a toe is initialized, and, exceptionally, the keyframe kick movement, which is a kind of kick implemented based on keyframe optimization [2], is set as default. This was necessary because the current neural kick (developed using Deep Reinforcement Learning, as detailed in Sections 6 and 7) makes the ball go too high, leading the ball to go over the goal most times since the penalty kicker first navigates with the ball until it is close to the goal before kicking. The keyframe movement executed with the agent with toe leads to the same issue. Therefore, a combination of the toe-less agent and the keyframe kick was necessary to ensure a penalty kick that has good chances of scoring a goal.

5.3 Walk Optimization

ITAndroids Soccer 3D started to develop optimization tasks focusing on walking's movement. The purpose of this activity is to improve the stability and velocity of the movement, based on the fact that previous work on this area reported great results [1]. The team developed three tasks and optimized Omnidirectional ZMP parameters and Navigation parameters. The tasks include the following movements for the robot: running straightforward, going to the ball and conducting it in the field, and going towards targets in the field whose positions are randomly generated.

5.4 Walk State Machine

With the optimization of walking parameters it became necessary to re-plan the movement. In this way, a new decision-making was implemented according

to the walking situation. The walking states are: sprint, when the agent walks without the ball; conduct, when the agent walks with the ball; turn, when the agent cross curve paths; and circle, when the agent circulates the ball.

5.5 Positioning

An algorithm was implemented to determine the positioning of our players when the ball is with the opposing team.

We defined that the player who is closest to the ball will go to it, while the others will go to their positions defined by this algorithm.

In this algorithm, the position of each player depends only on where the ball is on the field, that is, independent of the positions of the opponent players.

This returns us a reasonably satisfactory field positioning.

Initially, the positions of the 11 players for some ball positions in the field were defined, being strategically chosen to cover the different sectors of the field.

With these ball positions, a Delaunay triangulation was created in order to obtain a set of triangles over the field, whose acute angles were as large as possible.

Knowing that the vertices of these triangles represent ball positions whose players formation for them is already set, a linear interpolation algorithm was used on the vertices of these triangles, called Gouraud Shading, to determine the formation of the players to some ball position within that triangle that does not have a positioning initially set. The same weighted average the algorithm realized with the vertices of the triangle for the ball position in its interior is also used with the positions of each player to return an interpolated position for each of them when the ball is in that position.

5.6 Movement with Ball

For the movement with the ball, we implemented a method called Potential Fields. This robot navigation method applies to each field's point a numerical value that corresponds to the potential caused by external agents. Therefore, it is possible to know which points should be avoided and which point is the goal.

Potential is a scalar quantity, therefore we can add directly the potential caused by each object in the vector. So, our function calculates each potential and sums it all. After that, we can calculate the desired direction using this formula:

$$\theta = \arctan\left(\frac{V_y}{V_x}\right) \tag{1}$$

6 Reinforcement Learning Optimization Server

Motivated by the recent advances in Deep Reinforcement Learning, we created a framework that is able to mimic a reference motion and optimize it towards a

task. We represented the motion as a neural network, with thousands of parameters, which followed a 2-step training procedure where:

- The first step is a Supervised Learning Training, where we transfer the knowledge from a keyframe representation to a neural network; and
- The second step is a Reinforcement Learning Training, where we optimize the motion from the neural network using rewards related to a specific performance task.

We developed better policies than the initial ones for the kick motion. Using this framework, we also showed that pure Reinforcement Learning techniques by themselves will lead to suboptimal policies (due to premature convergence) and will not achieve high-performance motions [3].

7 The Bottom-Up Meta-Policy Search (BUMPS) Algorithm

In spite of the advancement and optimization we made using Reinforcement Learning, there were still many challenges in situations on which our robots need to behave in ways for which it was not trained. To minimize this problem, we used a first-order Meta-Learning algorithm called Bottom-Up Meta-Policy Search (BUMPS) [9], that works with two-phases optimization procedure:

-In the first step, it distills few expert policies to create a meta-policy capable of generalizing knowledge to unseen tasks during training, in a meta-training phase;

-In the second step, it applies a fast adaptation strategy named Policy Filtering, which evaluates few policies sampled from the meta-policy distribution and selects which best solves the task.

We conducted the algorithm in our robots, in the context of kick motion learning. We show that, given our experimental setup, BUMPS works in scenarios where simple multi-task Reinforcement Learning does not.

7.1 The First Step: The Meta-Training Phase

Bottom-Up Meta-Policy Search (BUMPS) is an algorithm that uses the knowledge of expert policies in single tasks to train a contextual meta-policy that generalizes for unseen tasks. The method is based on the idea of "Bottom-Up Learning" [11], where learning happens firstly in implicit knowledge and then in explicit knowledge (i.e, through "extracting" implicit knowledge).

The idea of contextual meta-policy comes from the fact we give the task context as input. The meta-training phase then explores the task structure to create a rich representation of shared features based on contextualization, exploiting the geometrical representation of optimal solution manifolds in parameter space. This meta-policy is more robust as it learns MDPs with other contexts, better creating such representation.

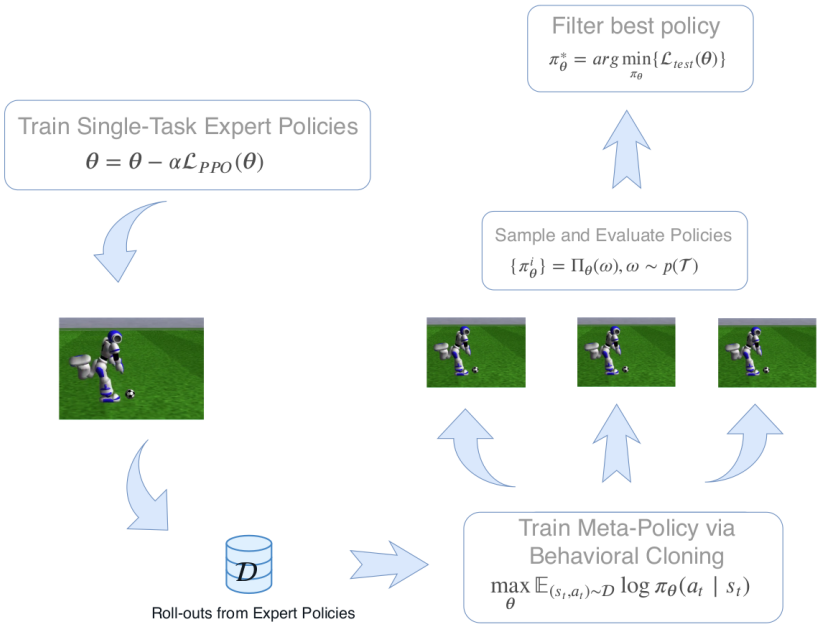

Fig. 1: Bottom-Up Meta-Policy Search.

During meta-training, we conduct supervised imitation learning, which reduces the agent-environment interaction and uses gradients with lower variance and more stable than RL gradients [10], thus improving sample efficiency.

7.2 The Second Step: The Meta-Testing Phase

The generalization comes from the fact that such parameters can also solve unseen tasks due to the proximity between optimal policies with near contexts. However, we cannot guarantee that the best policy for a task will be in the right context. To solve this problem, we apply Policy Filtering: we sample several contexts (i.e, policies) in the neighborhood of the target task context and evaluate them to check if they are inside the optimal solution manifold. When we find the context that is closer to such manifold (i.e, better solves the task), we then apply the idea of meta-gradient through context representations.

7.3 Results with Bottom-Up Meta-Policy Search Algorithm.

We address the problem of humanoid robot kick motion learning to evaluate the BUMPS algorithm. Specifically, our objective was to learn how to learn to kick: given a range of possible distances for a kick (as a distribution of tasks), we could sample a policy that precisely kicks for each task. For each task (i.e, target distance), we executed the kick motion several times to collect statistics about accuracy and target error.

• **How much meta-training is able to generalize to unseen tasks?** We confront accuracy and mean error from single-task expert policies and meta-policy, using the evaluation scenario: one hundred repetitions of each kick policy. The table below shows such values. In this experiment, we evaluate the single-task expert policies in their respective meta-training tasks, while the meta-policy in all meta-testing tasks. For meta-policy, we considered two networks with a similar number of parameters: one with 4 hidden layers of 256 neurons and the other with 11 hidden layers with 128 neurons.

Table 1: Meta-Policy Generalization Performance.

Model Type	**Statistics**			
	Accuracy		**Error** (m)	
	Mean	***Std***	***Mean***	***Std***
Single-Task Expert Policies	0.89	0.04	0.57	0.14
Meta-Policy (4, 256)	0.84	0.10	0.72	0.25
Meta-Policy (11, 128)	0.84	0.17	0.68	0.28

• **How much policy filtering improves meta-policy generalization?** Final results are presented in the table below. As we observe, there is a visible improvement from meta-policy, especially when considering ensemble and a high sample rate. In all models, BUMPS achieved a mean error of less than a half meter, which is better than initial performance for meta-training tasks.

Table 2: Final results for BUMPS algorithm

Model Type	**Statistics**			
	Error(m)		**Rel. Error (%)**	
	Mean	***Std***	***Mean***	***Std***
Policy (4, 256)	0.45	0.10	3.8	1.1
Policy (11, 256)	0.43	0.13	3.3	1.1
Ensemble	**0.40**	**0.11**	**3.3**	**1.1**
High Sample Rate (4, 256)	**0.40**	**0.09**	**3.3**	**0.9**

However, despite the lesser error, in a real play-game situation, due to the way how the robot stops to kick, the neural kick trained with BUMPS cannot be used.

8 Conclusions and Future Work

This paper showed the latest developments in ITAndroids Soccer 3D. During LARC 2019, it was noticed that our agents still cannot navigate with the ball in an effective manner, meaning that they easily lose the ball to the opponent team while navigating, usually relying instead on long-range kicks to get the ball close to the opponents' goal. It was also noticed that a tool to collect game statistics is required to effectively test tweaks in parameters and new formations, and that it is important to develop more movements for the goalie, like falling intentionally sideways when the ball is close to the goal in order to increase the area it covers.

Now aware of this, we will focus on tweaking the navigation to make it more efficient, and then modify the behaviors so that the agents put more emphasis on navigating rather than making long-range kicks. Also, we plan to look deeper into possibilities of a more dynamic goalie behavior, as well as pursue improvements in other aspects we deem relevant for our team.

Acknowledgment

We would like to acknowledge the RoboCup community for sharing their developments and ideas. Especially, we would like to acknowledge the Magma Offenburg team for sharing their code and, with that base, helping us develop our current code. We thank our sponsors Altium, ITAEx, Metinjo, Micropress, Poliedro, Poupex, Rapid, and Solidworks. We also acknowledge Mathworks (MATLAB), Atlassian (Bitbucket) and JetBrains (CLion) for providing access to high-quality software through academic licenses. We would also like to show our gratitude to Patrick MacAlpine from the UT Austin Villa team for sharing their ideas and code regarding the Soccer 3D simulation server modification. Special thanks for the cluster access provided by Intel, which raised our computing capacity to a whole new level.

References

1. P. MacAlpine, S. Barret, D. Urieli, V. Vu, P. Stone. Design and Optimization of an Omnidirectional Humanoid Walk: A Winning Approach at the RoboCup 2011 3D Simulation Competition. Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI), July 2012.
2. Francisco Muniz, Marcos Maximo and Carlos Ribeiro. Keyframe Movement Optimization for Simulated Humanoid Robot using a Parallel Optimization Framework. In: Latin American Robotics Symposium. In Proceedings of the 2016 Latin American Robotics Symposium (LARS), October 2016.
3. Muzio, A. F. V. A Deep Reinforcement Learning Method for Hu-manoid Kick Motion, 2018.
4. Pires, F. B.: Regular triangulations and applications (Doctoral dissertation). Retrieved from `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-26082008-163553/pt-br.php`.

5. Patrick MacAlpine, Francisco Barrera, and Peter Stone. Positioning to Win: A Dynamic Role Assignment and FormationPositioning System. In Xiaoping Chen, Peter Stone, Luis Enrique Sucar, and Tijn Van der Zant, editors, RoboCup-2012: Robot Soccer World Cup XVI, Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin, 2013.
6. S. Josefsson. The Base16,Base32,and Base64 Data Encodings. IETF. October 2006. Retrieved from `https://tools.ietf.org/html/rfc4648`.
7. Passault G. Hofer L. N'Guyen S. Rouxel, Q. and O. Ly. Rhoban hardware and software open source contributions. In 15th IEEE-RAS International Conference on Humanoid Robots, November 2015.
8. Visser U. Stoecker J. Roboviz: Programmable visualization for simulated soccer. In RoboCup 2011: Robot Soccer World Cup XV, 2012.
9. Bottom-Up Meta-Policy Search, Luckeciano C. Melo, Marcos R. O. A. Maximo, Adilson Marques da Cunha
10. Mohammad Norouzi, Samy Bengio, Zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. Reward augmented maximum likelihood for neural structured prediction. CoRR, abs/1609.00150, 2016.
11. Ron Sun. Bottom-Up Learning and Top-Down Learning, pages 479–481. Springer US, Boston, MA, 2012.