Very Small Size Robot Team Strategy

1st Igor Mourão Ribeiro
Computer Engineering Departament
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brazil
igormr98mr@gmail.com

2nd Paulo Marcelo Tasinaffo
Computer Engineering Departament
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brazil
tasinaffo@ita.br

3rd Marcos Ricardo O. A. Máximo
Computer Engineering Departament
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brazil
maximo.marcos@gmail.com

***Abstract*—The development of a robust and consistent strategy for a complete team of soccer in the category "Very Small Size" is fundamental to win the matches. After a research phase, it was decided to use the Behavior Tree method for making team decisions. Then, chosen three roles for the players: goalkeeper, main and assistant. Afterwards, a behavior tree for each of them, as well as a technician responsible for ensuring dynamic exchange of roles. The criterion used to evaluate the algorithm was its performance in simulated matches and in national competitions.**

***Index Terms*—robotics, strategy, decision making**

I. Introduction

In the context of robot football, decision-making poses many challenges. The strategy consists of the implementation of an algorithm that is able to use the robot's trajectory planning in the best possible way. This brings us to a higher level of abstraction challenge: deciding what the best decision each robot can take at a certain time considering limitations of possible trajectories to be followed.

In this project, we worked with the robots of the IEEE Very Small Size (VSS) competition: a fully automated robot soccer competition in which each robot has a maximum size of 7.5 cm cube. The football field is 1.5 m long and 1.3 m wide. Each team has 3 players: they can take dynamic positions, as goalkeeper and striker, during a game. A camera provides the aerial view with the positions of all elements of the match. The complete rules can be read in [3].

Fig. 1. ITAndroids Robots of the IEEE VSSS category.

The Figure 1 shows the robots used in competitions. The computational vision used by the ITAndroids team can be found in more detail in [2] and uses a camera at the top of the field that passes information to a computer processing, as seen in Figure 2.

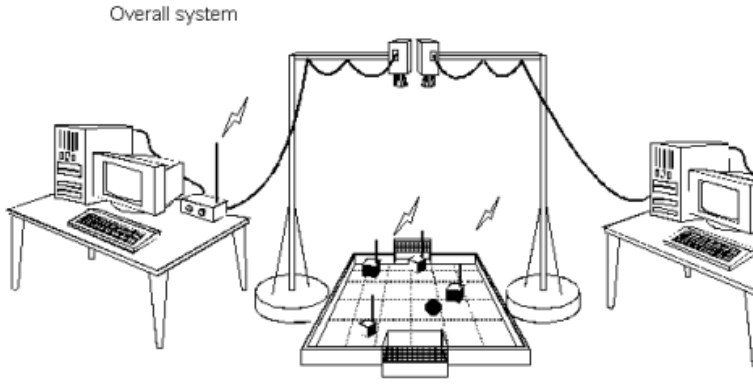

Fig. 2. Representation of the technical functioning of a VSS game.

Each robot was designed with two side wheels and two free balls at the front and back to maintain balance. This characteristic makes it a differential robot. That is, we have control over linear and angular speeds, but it is not possible to

move the robot sideways. The language used for the project was C ++ because it is a language that has a high execution speed and has good scalability for a large project.

In this context, several issues arise to be solved in order to have a winning team. In this project, the following strategy problem will be addressed: given the position and speed of each robot and ball, one must choose which position a particular robot will go with, which controller and which path planning to use. One must also put together a plan for future actions, planning a sequence of positions that the robot must go to to make a goal.

The strategy algorithm should run at run time of at most 10 ms for the entire team, which is the camera sampling time discounted from computational view processing time. Most of this time is spent on the trajectory planning used, which is the Rapidly-exploring Random Tree (RRT). The time for each execution of an RRT is 1 ms, according to [5], limiting it around a limit of twelve algorithm calls by completing the strategy, with a margin of error of thirty-three percent . The importance of not exceeding this maximum time is emphasized, since if this happens an image of the camera is lost and the operation of the game is carried out in a delayed way.

A dynamic positioning scheme must therefore be set up in which there is no fixed attacker, defender or goalkeeper, but rather that the positions depend on and adapt to the playing position, ensuring that the three players can be better positioned and free to choose their next actions. This must be done because the fixed positioning scheme does not work very well in the context of the Very Small Size category, since it greatly limits the offensive and defensive capacity of the team.

To solve this problem, it was chosen as the main algorithm for decision making the Behavior Tree. This algorithm was chosen from among options such as Finite State Machine and Decision Tree. An introduction to these issues can be found in [4] for finite state machine and [1] for Decision Tree. The choice is based on the decision of a reference team in world robot football, the Near East University team: the NEUIslanders, as described in [9].

This work brings as a contribution the implementation of the Behavior Tree method for the Very Small Size category, being the first scientific publication of this subject in the literature.

A. Behavior Tree

To build a Behavior Tree (BT), you must first understand the basics of its structure. A BT is a Tree in the sense given by graph theory, as explained in [6]. By definition, each node of this tree is called behavior.

The nodes of a Behavior Tree are differentiated into two types: the leaf nodes, which are the nodes of the tree that has no children and the compound nodes, which may have one or more children.

All nodes return, at the end of their execution, one of the following three states: TRUE, FALSE, or RUNNING, to indicate, respectively, that the node was successfully terminated, that it was terminated without success, or that the algorithm should execute that node for at least one more iteration.

1) Nós Folha: The leaf nodes are the terminal nodes of the Tree and they represent the lowest and most concrete actions of the strategy. They are divided into two types: action nodes and conditional nodes.

Action nodes: These are nodes that actually perform an action, such as moving to the ball.

Conditional nodes: These are nodes that only return a state without performing any concrete action, such as checking if the team is attacking. They are used to control which nodes will be run in conjunction with the compound nodes.

2) Nó composto: Nós compostos são os nós intermediários da árvore, que são responsáveis por escolher quais os nós que serão executados. São completamente reutilizáveis no sentido que apenas uma implementação de cada tipo de nó composto deve existir e ela poderá ser usada em diversas partes do código. Os principais tipos desses nós que foram usados para a estratégia estão listados a seguir:

Selector Behavior Nesse behavior, é selecionado um de seus filhos para ser acessado a seguir. Ele é usado quando tem-se várias maneiras de se realizar uma mesma ação e deve-se escolher a melhor delas em uma determinada situação. Ele funciona executando os filhos em uma determinada ordem e, assim que um dos filhos retorna TRUE ou RUNNING, esse nó retorna o mesmo valor. Se um filho retornar FALSE, o próximo filho será executado. Caso nenhum filho seja sucedido ou retorne que deve ser executado mais um vez, o nó retorna FALSE.

Sequence Behavior Esse nó executa cada um dos seus filhos em uma sequência bem definida e fixa. Ele é usado para fazer ações sequenciadas para completar um objetivo maior, por exemplo: chutar a bola para o gol oponente requer que o robô aliado esteja atrás da bola, o que significa que chegar atrás da bola e, em seguida, chutar a bola representam ações sequenciadas. Ele funciona de forma que, quando um filho retorna RUNNING ou FALSE, o behavior retorna o mesmo valor e, quando um filho retorna TRUE, ele avança para o próximo filho. Se todos os filhos retornarem TRUE, o behavior é bem sucedido e também retorna TRUE.

Parallel Behavior O objetivo desse nó é executar todos os seus filhos ao mesmo tempo, fato que pode ser obtido ao utilizar-se de processamento paralelo ou não. Na implementação desse trabalho foi usado a versão não paralela do algoritmo. Se um filho retorna FALSE, o behavior retorna FALSE. Se todos os filhos retornarem TRUE, o behavior retorna TRUE. Em todas as outras situações, o behavior retorna RUNNING e continua sua execução por pelo menos outra iteração.

Decorator O nome para esse behavior é inspirado no conceito de Decorator em Software Design Pattern, que pode ser estudado em [10]. No contexto da Behavior Tree, esse nó tem apenas um filho e ele modifica o comportamento do filho de alguma maneira. Um decorador pode ser diversos tipos, mas os mais usados no projeto foram:

1) AlwaysFail e AlwaysSucceed: que fazem que o filho sempre retorne FALSE ou sempre retorna TRUE respectivamente.
2) UntilFail e UntilSucceed: fazem que o fiho sempre retorne RUNNING a não ser que ele retorna FALSE ou TRUE, respectivamente.
3) Invert: muda o retorno do filho de FALSE para TRUE e TRUE para FALSE.

B. *Técnico*

Uma Behavior Tree, que embora consegue modelar comportamentos complexos com facilidade, nem sempre é a melhor escolha para determinadas situações. Por isso, a estratégia foi dividida em níveis de abstrações diferentes, em que um Técnico, representado pela classe Coach, seria responsável por escolher e administrar quais árvore cada um dos três jogadores deveria usar no momento. Com essa implementação, tem-se um agente externo controlando a função dos jogadores por meio da escolha de uma BT, que pode ser uma árvore para um goleiro, para um principal e para um auxiliar. Isso significa que, em um determinado momento, o técnico pode escolher que o time tenha 1 goleiro, 1 principal e 1 auxiliar a até mesmo 3 principais ao mesmo tempo. Os requisitos esperados para cada um desses papéis será abordado a seguir. Além disso, também é função do técnico escolher o papel de cada jogador a cada iteração do código, de modo que as posições dos jogadores não sejam fixas ao decorrer do jogo.

II. Results and Discussion

Os resultados a seguir foram obtidos estudando o comportamentos de equipes adversárias na competição, além de diversos testes com o time de robôs da ITAndroids contra si mesmo em simulações computacionais em um simulador feito pela própria equipe, como mostrado na Figura 3.

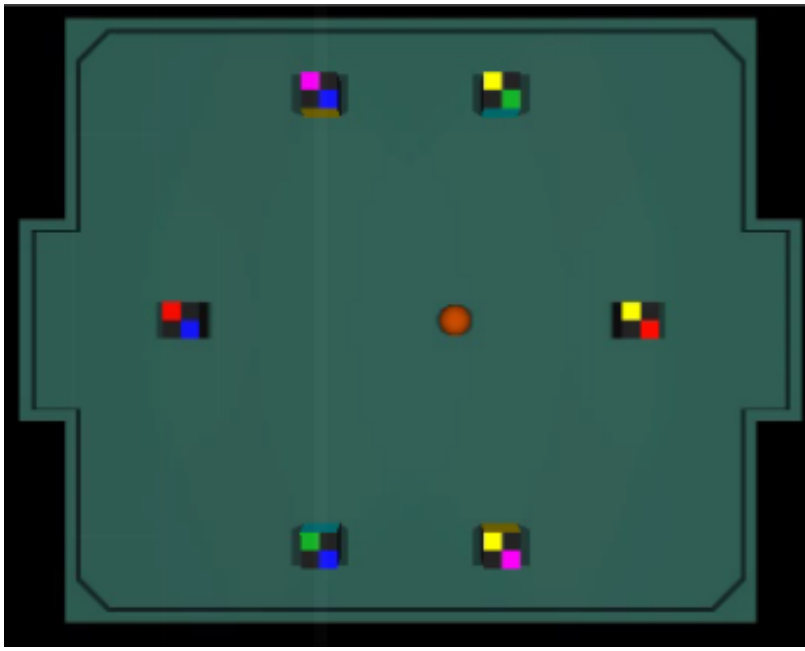

Fig. 3. Simulador da ITAndroids.

A. *Goleiro*

O goleiro é o jogador que deve ficar próximo ao gol aliado com o objetivo de proteger o gol de ataques oponentes. A BT criada para o goleiro está representada na Figura 4.

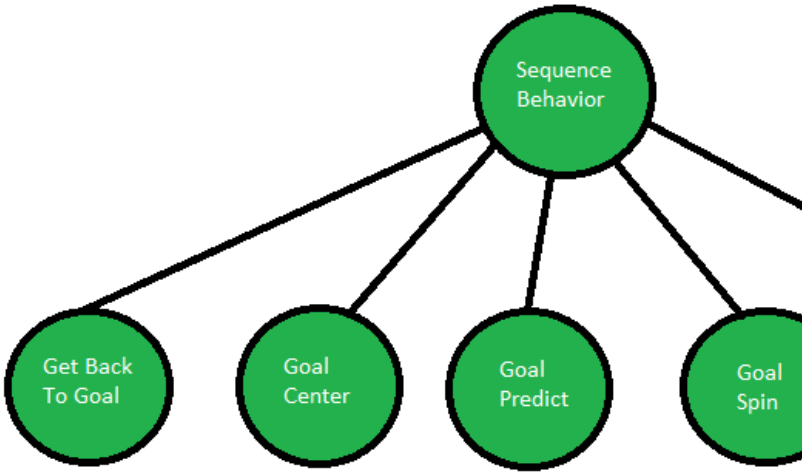

Fig. 4. Behavior Tree para o goleiro.

Como visto, ela é composta por um nó do tipo Sequence Behavior, que irá executar os seus filhos em sequência. Esse papel, então, executa as seguintes ações prioritando as primeiras a aparecerem na seguinte lista:

- **Get Back to Goal Behavior** Volta para o gol, caso que, se por algum motivo ele esteja fora do próprio.
- **Goal Center** Fica centralizado no gol quando a bola estiver longe, de forma que o jogador possa rapidamente ir para qualquer um dos lados quando a bola se aproximar.

- **Goal Predict** Prediz para onde a bola irá quando ela estiver rápida e longe do gol.
- **Goal Spin** Gira quando está perto da bola e não tem oponente perto da bola para jogá-la para longe.
- **Goal Catch** Comportamento que o goleiro deverá fazer quando não fizer nenhum outro, por isso sua última posição na sequência. Ele acompanha o movimento da bola com o goleiro sobre a linha do gol.

B. Principal

O jogador com o papel de Principal é o jogador mais ativo do time, que deve estar constantemente avançando em direção a bola. Esse papel e o goleiro são as únicas posições que efetivamente deverão tocar na bola. A árvore 5 foi desenvolvida para o principal, sendo a árvore mais complexa dentre os papéis.

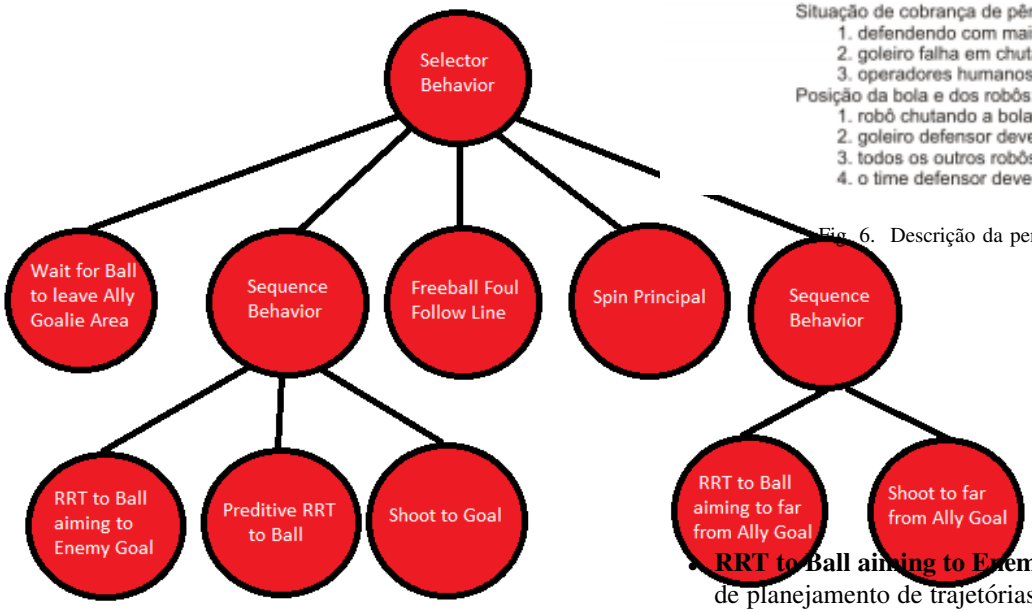

Fig. 5. Behavior Tree para o principal.

Conforme a Figura 5, a raiz da árvore do principal é um Selector Behavior que escolhe uma das ações a ser realizada. Essa BT tem mais dois outros nós compostos, que são dois Sequence Behavior usados para o posicionamento atrás da bola, seguido pelo chute. Uma descrição dos nós folha utilizados se encontra abaixo:

- **Wait for Ball to leave Ally Goalie Area** Esse Behavior utiliza a Triangução de Delaunay. Esse comportamento deve evitar que o jogador entre na área do goleiro para não sofrer penâlti, conforme descrito na Figura 6. O jogador se pocionará de acordo com posições escolhidas pelo usuário, calibradas com um arquivo de configuração .txt.

Fig. 6. Descrição da penalidade pênalti.

- **RRT to Ball aiming to Enemy Goal** Irá usar o algoritmo de planejamento de trajetórias RRT para se deslocar atrás da bola com direção apontando para o gol oponente. Usado para se aproximar da bola e, em seguida, trocar para o próximo behavior, o Predictive RRT to Ball.
- **Predictive RRT to Ball** Usa uma predição linear considerando que a bola continuará na mesma velocidade. Usado para se chegar na bola com mais precisão quando próximo a ela. Chega atrás da bola mirando para o gol.
- **Shoot to Goal** Esse behavior só é chamado quando o jogador já está alinhado com a bola e o gol oponente. Esse comportamento acelera rapidamente o robô em linha reta para chegar no gol com uma velocidade alta.
- **Free Ball Foul Follow Line** Esse behavior é um específico para situações de falta do tipo Bola Livre, conforme descrito na Figura 7. Quando for detectado uma dessas posições, o jogador deve acelerar o mais rápido possível em direção à bola para ter o controle dela antes do oponente.

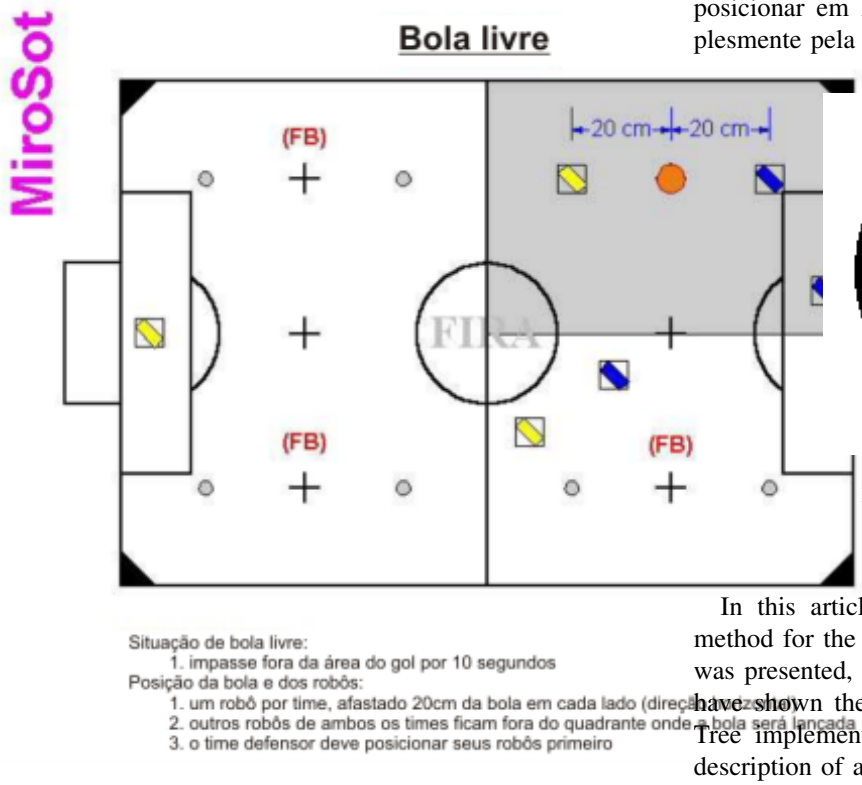

Fig. 7. Descrição da penalidade bola livre.

- **Spin Principal** Esse nó deve ser chamado quando a bola estiver em um dos quatros campos do campo. Caso seja um campo defensivo, o jogador irá girar para jogar a bola para frente; caso seja um na zona de ataque, o jogador girará para por a bola no centro do campo e continuar o ataque.
- **RRT to Ball aiming to far from Ally Goal** Irá usar o algoritmo de planejamento de trajetórias RRT para se deslocar atrás da bola com direção apontando para a zona de ataque (longe do gol aliado). Usado para se aproximar da bola e se posicionar para, em seguida, chamar o Behavior Shoot to Far from Ally Goal.
- **Shoot to Far from Ally Goal** Esse behavior só é chamado quando o jogador já está alinhado com a bola e o para frente (para longe do gol aliado). Esse comportamento acelera rapidamente o robô em linha reta para pôr a bola na zona de ataque.

C. Auxiliar

O auxiliar é um papel que tem apenas uma função: posicionar-se da melhor forma possível para manter um ataque consistente. A ideia é que, caso a situação seja oportuna, o auxiliar se torne o principal e comece a atacar a bola. Para escolher a posição que o auxiliar deverá ficar em uma determinada situação de jogo, foi utilizado a Triangulação de [7] sobre o grafo de [8]. Esse algoritmo foi aplicado similarmente a [11], relativo também à futebol de robôs.

A Behavior Tree do auxiliar, por sua vez, é muito simples, representada por apenas um nó que tem como função se posicionar em lugar ótimo. Sua árvore, é representada simplesmente pela Figura 8.

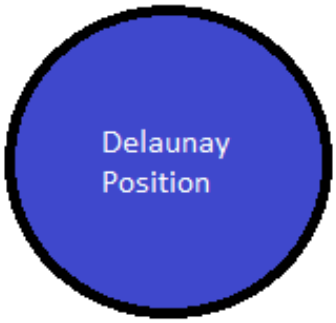

Fig. 8. Behavior Tree para o auxiliar.

III. Conclusion

In this article, an implementation of the Behavior Tree method for the category of robots of Very Small Size Soccer was presented, which was briefly described its operation. We have shown the paper scheme used as well as the Behavior Tree implemented for each of them, as well as a detailed description of all the tree nodes.

The Scientific Initiation Project presented good results, especially with respect to the ITA team's victory until the quarterfinals of CBR 2017, yielding the seventh place to the team among more than 25 participating teams. In addition to the fact that ITAndroids won the first place in the national competition of VSSS Turing Cup 2017, which occurred in late September 2017.

From a technical point of view, the use of the Behavior Tree algorithm, already consecrated and used by world-class teams, proved to be feasible and functional in real matches of competitions. However, problems such as low scalability and low reusability were shown in the developed strategy. This means that changes still have to be made in BT, especially regarding the use of more parallel-type nodes to remove these problems.

References

[1] Russel, S. and Norvig, P., 2009. Artificial Intelligence: A Modern Approach, Prentice Hall, Vol. 3, pp. 697–707. 3rd edition
[2] Zickler, S., Laue, T., Birbach, O., Wongphati, M. and Veloso, M., 2009. "Ssl-vision: The shared vision system for the robocup small size league". In Robot Soccer World Cup. Springer, pp. 425–436.
[3] CBR, 2008. Http://www.cbrobotica.org/wp-content/uploads/2014/03/VerySmall2008_en.pdf.
[4] Coelho, M.M.A., 2016. "Máquinas de estados finitos".
[5] Okuyama, I.F., 2017. TRAJECTORY PLANNING CONSIDERING ACCELERATION LIMITS FOR AN AUTONOMOUS SOCCER PLAYER. Trabalho de graduação, Instituto Tecnológico de Aeronáutica.
[6] West, D.B. et al., 2001. Introduction to graph theory, Vol. 2. Prentice hall Upper Saddle River.
[7] Delaunay, B., 1934. "Sur la sphére vide". Bulletin de l'Académie des Sciences de l'URSS, Vol. 6, pp. 793–800.
[8] Voronoi, G., 1908. "Nouvelles applications des paramètres continus à la théorie des formes quadratiques". Journal fürdie Reine und Angewandte Mathematik, Vol. 133, pp. 97–178.

[9] Abiyev, R.H., Şenol Bektas, Akkaya, N. and Aytac, E., 2017. “Neuislanders 2017 team description paper”. Robocup.
[10] Hunt, J., 2013. “Gang of four design patterns”. In Scala Design Patterns, Springer, pp. 135–136.
[11] Akiyama, H. and Noda, I., 2007. “Multi-agent positioning mechanism in the dynamic environment”. In Robot Soccer World Cup. Springer, pp. 377–384.