Otimização de parâmetros por meio de Algoritmos Evolutivos em Futebol de Robôs

Igor Mourão Ribeiro
Instituto Tecnológico de Aeronáutica
Rua H8A, 121, CTA
12228-460 - São José dos Campos/SP
Bolsista PIBIC - CNPq
igormr98mr@gmail.com

Celso Massaki Hirata
Instituto Tecnológico de Aeronáutica
Divisão de Ciência da Computação
Praça Marechal Eduardo Gomes, 50
12229-900 – São José dos Campos / SP
hirata@ita.br

Marcos Ricardo Omena de Albuquerque Máximo
Instituto Tecnológico de Aeronáutica
Divisão de Ciência da Computação
Praça Marechal Eduardo Gomes, 50
12.229-900 – São José dos Campos / SP
maximo.marcos@gmail.com

Resumo: *O desenvolvimento de uma estratégia robusta e consistente para um time completo de robôs jogadores de futebol da categoria "Very Small Size"é fundamental para ganhar as partidas. Após uma fase de pesquisa, foi decidido usar o método da Behavior Tree (árvore de comportamentos) para realizar a tomada de decisões do time. Em seguida, foi escolhido três papeis para os jogadores: goleiro, principal e auxiliar. Depois, foi desenvolvido uma árvore de comportamentos para cada um deles, além de um técnico responsável por garantir troca dinâmica de papeis. O critério usado para avaliação do algoritmo foi o seu desempenho em partidas simuladas e em competições nacionais.*
Palavras-chave: *robótica, estratégia, tomada de decisão*

1. INTRODUÇÃO

No contexto do futebol de robôs, a otimização de parâmetros traz muitos desafios. Tal problema consiste no fato de se implementar um algoritmo que consiga encontrar as melhores constantes a serem usadas no código para melhorar o desempenho do robô em uma partida real. Entretando, para encontrar esses parâmetros de forma automatizada, é necessário, além do algoritmo evolutivo em si, a existência de simulações fiéis à realidade. A Figura 1 mostra a placa eletrônica do robô utilizado.

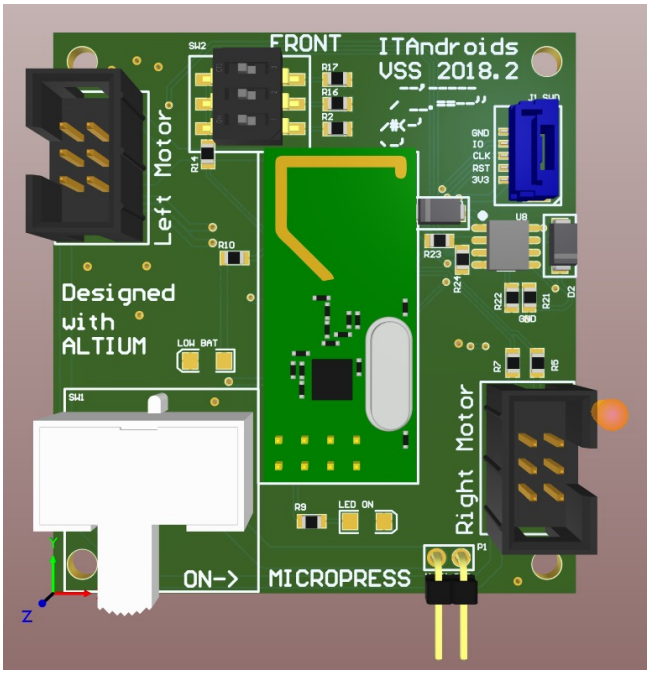

Figura 1: Placa eletrônica de um robô Very Small Size.

Nesse projeto, trabalhou-se com os robôs da competição IEEE Very Small Size (VSS): uma competição de futebol de robôs completamente automatizada em que cada robô tem como tamanho máximo um cubo de 7,5 cm de aresta. O campo de futebol possui 1,5 m de comprimento e 1,3 m de largura. Cada time tem 3 jogadores: que podem assumir posições dinâmicas, como goleiro e atacante, ao decorrer de uma partida. Uma câmera proporciona a visão aérea com as posições de todos os elementos da partida. As regras completas podem ser lidas em **?**.

Figura 2: Robôs da ITAndroids da categoria IEEE VSSS.

A Figura 2 mostra os robôs usados nas competições. A visão computacional utilizada pela equipe ITAndroids pode ser encontrada com mais detalhes em **?** e utiliza uma câmera no topo do campo que repassa informações para um computador processar, como visto na Figura 3.

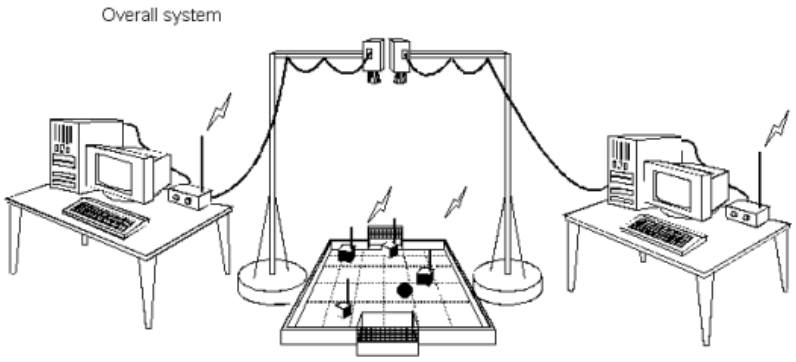

Figura 3: Representação do funcionamento técnico de um jogo de VSS.

Cada robô foi projetado com duas rodas laterais e material teflon para diminuir atrito no deslizamento de seu chassis. Essa característica o torna um robô diferencial. Ou seja, temos o controle sobre as velocidades linear e angular, mas não é possível mover o robô para os lados. A linguagem utilizada para o projeto foi C++; diferente da utilizada nesse trabalho, que é Matlab; pois é uma linguagem que tem uma velocidade de execução alta e tem boa escalabilidade para um projeto grande.

Nesse contexto, surgem várias questões a serem resolvidas para se ter um time vencedor. Nesse projeto, será abordado o seguinte problema de otimização: dado um simulador feito pela equipe que simula a física e a lógica de uma partida oficial, deve-se encontrar os melhores valores possíveis para as constantes do planejamento de trajetórias, de modo a maximizar o desempenho do time em uma partida com os robôs reais.

Sobre o assunto de planejamento de trajetórias, o utilizado pela equipe é o Univector Field **?**, que é muito eficiente e já demonstrou bons resultados em competições oficiais com algumas das melhores equipes utilizando-o. Uma das suas desvantagens, é a existência de sete parâmetros diferentes para serem otimizados e de que essas constantes não são intuitivas para uma pessoa ajustá-las sem ajuda de um computador. Isso acontece devido ao comportamento do algoritmo ser relativamente complexo quando comparado com outras alternativas como o algoritmo Campos Potenciais **?**.

Além disso, é importante ressaltar que o tempo de execução do algoritmo deve ser o mínimo possível. Nota-se que não é possível simular uma partida completa de futebol já que isso demoraria bastante e o algoritmo precisa de em torno de 100 iterações para convergir. Para isso, técnicas para executar simulações aceleradas e executar situações específicas de jogo foram desenvolvidas, com o objetivo de reduzir o tempo total da otimização.

Ademais, outro problema que será enfrentado no desenvolvimento desse projeto é a criação de interfaces de comunicação entre o algoritmo de otimização, a ser codificado em Matlab, com o código do projeto e das simulações, ambos codificados na linguagem de programação C++.

Para resolver esse problema, foi escolhido como algoritmo para a otimização de parâmetros o Covariance Matrix Adaptation Evolution Strategy, já que ele consegue otimizar problemas não lineares e não convexos, que, devido à elevada complexidade, é o caso do futebol de robôs.

1.1 Covariance Matrix Adaptation Evolution Strategy

O algoritmo CMA-ES é uma estratégia evolutiva. Um algoritmo evolutivo é baseado no princípio da evolução biológica, em que conceitos de recombinação e mutação de genes são aplicados para selecionar melhores indivíduos. Mais formalmente: em cada geração novos indivíduos são criados a partir de seus pais, com o objetivo de gerar melhores indivíduos segundo algum critério.

No contexto da computação, cada geração será considerada uma iteração. Em cada iteração existirá uma população composta de indivíduos (ou candidatos) que deverá ser alterada segundo um algoritmo para diminuir uma determinada função de custo.

Para o caso do CMA-ES, os novos indivíduos são escolhidos com base em uma distribuição gaussiana de múltiplas dimensões. A nova população será gerada a partir da média dos melhores indíviduos da população anterior, em combinação com uma perturbação gaussiana estocástica. Dependência entre diferentes parâmetros são representadas por uma matriz de covariâncias.

Nota-se que o algoritmo, brevemente descrito, não usa derivadas nem valores das funções de custo, apenas importa a comparação entre os desempenhos dos indíviduos de uma população.

1.1.1 O algoritmo

O algoritmo do CMA-ES pode ser separado em alguma etapas:

1. Escolha dos parâmetros iniciais para a otimização: o tamanho do passo, o ponto inicial, o tamanho da população

e as condições de parada do algoritmo (que podem ser número de iterações ou ter um custo menor que um certo valor).

2. Enquanto a condição de parada não for atingida:

   (a) Gera novas soluções a partir da média dos melhores indivíduos da iteração anterior. Para isso usa-se uma distribuição gaussiana normal centrada em tal média e variância inicial igual ao tamanho do passo.

   (b) Cálculo dos custos de cada novo indivíduo por meio de uma função chamada de fitness.

   (c) Ordenação dos indíviduos, com base nos seus custos.

   (d) Atualização dos valores de variáveis internas.

Não será abordado os detalhes técnicos do item 2d, pois a implementação desse algoritmo não é o foco desse trabalho. Essa decisão ocorreu porque, além de tal passo ser bastante complexo estatisticamente, esse item apenas otimiza a velocidade de convergência do algoritmo e não é necessário para o entendimento inicial da estratégia evolutiva. Uma descrição mais profunda e detalhada de como otimizar o CMA-ES se encontra em **?**.

1.2 O simulador

Testes no robô real precisam de trabalho adicional devido a problemas que podem surgir como: bateria dos robôs acaba e precisa ser trocada, problemas no hardware ou mecânica do robô, campo sujo e necessidade de reposicionamento manual dos robôs. Por isso, a equipe ITAndroids desenvolveu um simulador próprio com o objetivo de resolver essas dificuldades.

O simulador é uma aplicação separada do código principal do time, desenvolvida e idealizada pela própria equipe, que cria um ambiente com física que simula um jogo. Tal aplicação se comunica com o código principal por meio de uma biblioteca de comunicação chamada Protocol Buffers **?**. Para isso, o simulador transmite informações como as posições e velocidades dos robôs para o código principal, enquanto que esse último transmite informações de velocidade das rodas de cada robô para o simulador.

A simulação foi desenvolvida em C++ com o auxílio da biblioteca Open Dynamics Engine **?**, para fazer as computações das equações diferenciais da física e das colisões dos movimentos, e da biblioteca Ogre3D **?**, responsável por renderizar os objetos na tela em três dimensões. A Figura 6 mostra a interface visual do simulador logo antes de se começar uma partida.

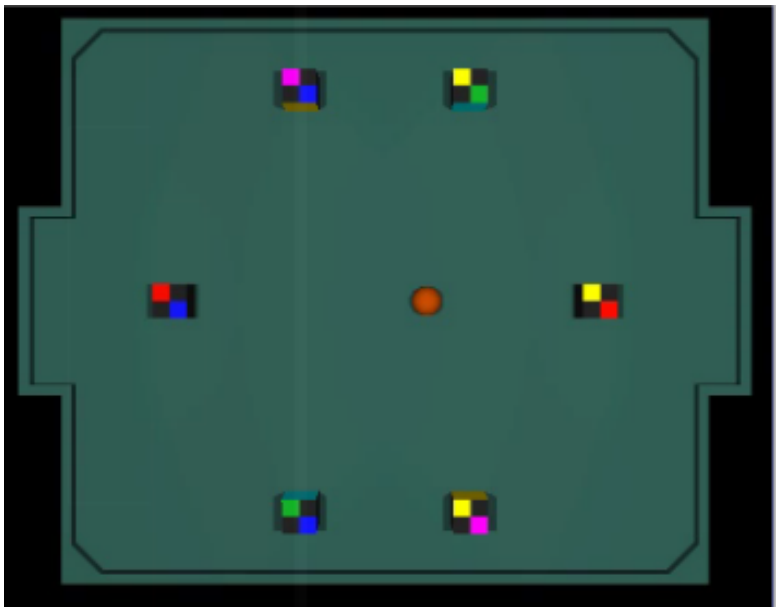

Figura 4: Simulador da ITAndroids.

Com isso, é possivel ter uma aproximação do comportamento do time em um jogo real. Dado que o simulador é desenvolvido pela própria equipe, é possível adicionar situações especiais de jogo, além de se adicionar condições de paradas específicas para o que se deseja testar. Essas funcionalidades são muito importantes para a criação de situações específicas de jogo e, consequentemente, de funções de custo mais rápidas e otimizadas, o que garante uma convergência mais rápida de determinados parâmetros.

Além disso, o simulador também tem um modo de funcionar que é o chamado: modo acelerado. Normalmente, o código tem algumas funções para esperar um determinado tempo, com o objetivo de fazer que ele execute em tempo

real (se ele não tivesse, a física seria muito rápida). Porém, no modo acelerado, esses tempos ociosos são removidos e a comunicação com o código principal funciona de forma assíncrona, o que garante uma velocidade de executação e simulação muito mais rápida, chegando a ser até dez vezes mais rápida que o modo de execução normal. Por essas razões, o modo acelerado é fundamental para garantir que a função de custo a ser desenvolvida seja mais eficiente e veloz.

1.3 Univector Field

O Univector Field é o planejamento de trajetórias usado pela equipe. O problema a ser resolvido é: como chegar de um ponto inicial à um ponto destino dentro do campo, desviando de obstáculos e considerando as limitações de controle e físicas do robô. Não será detalhado o funcionamento completo do algoritmo do Univector Field, já que isso pode ser visto em **?**.

Entretando, será descrito o que cada constante do algoritmo faz e uma descrição prática de como ela influencia na trajetória calculada, de modo a entender os resultados obtidos na otimização. Tem-se as seguinte constantes a serem otimizadas:

1. de: É o dobro de distância entre os centros de rotação do Univector Field. Na prática, quanto maior o seu valor, menos preciso o robô é em chegar em seu alvo. Se o seu valor for muito pequeno, o agente pode ficar preso em um mínimo local e nunca chegar em seu objetivo.

2. Kr: Define o quanto de rotação o campo gerado terá apontando para o seu alvo. Valores muito altos nessa constantes podem exigir que o robô faça curvas mais abruptas do que é possível e valores muito baixo farão que o robô faça curvas com raio muito grande, demorando mais a chegar no ponto desejado.

3. sigma: Parâmetro que define o tamanho do campo de repulsão gaussiano que o robô sofre de um obstáculo. Quanto maior o valor dessa constante, maior a repulsão do obstáculo.

4. dmin: A distância que o robô deve estar do obstáculo para o seu campo ser considerado infinito. Esse valor serve para garantir que o agente não colida com um obstáculo e deve ser otimizado para garantir que, se o robô se aproximar demais de um obstáculo, que ele seja repelido imediatamente e com muita intensidade.

5. K0: O parâmetro de predição de velocidade de obstáculos. É basicamente uma constante que é multiplicada pela velocidade do obstáculo para se calcular qual posição ele estará quando o agente chegar no objetivo. Único parâmetro que trata de obstáculos em movimento.

6. sigmaLine: Mesmo que o parâmetro sigma só que para as paredes do campo. As paredes são tratadas como um obstáculo especial já que, enquanto um obstáculo normal é um ponto, uma parede é uma reta.

7. dminLine: O mesmo que a constante dmin só que para as paredes.

O Univector Field é integrado com a interface de depuração da equipe, o que ajuda, qualitativamente, a verificar o quão otimizado é uma determinada trajetória gerada pelo algoritmo. Isso faz com que seja simples e rápido fazer uma checagem manual para saber se as constantes que o algoritmo convergiu não são absurdas.

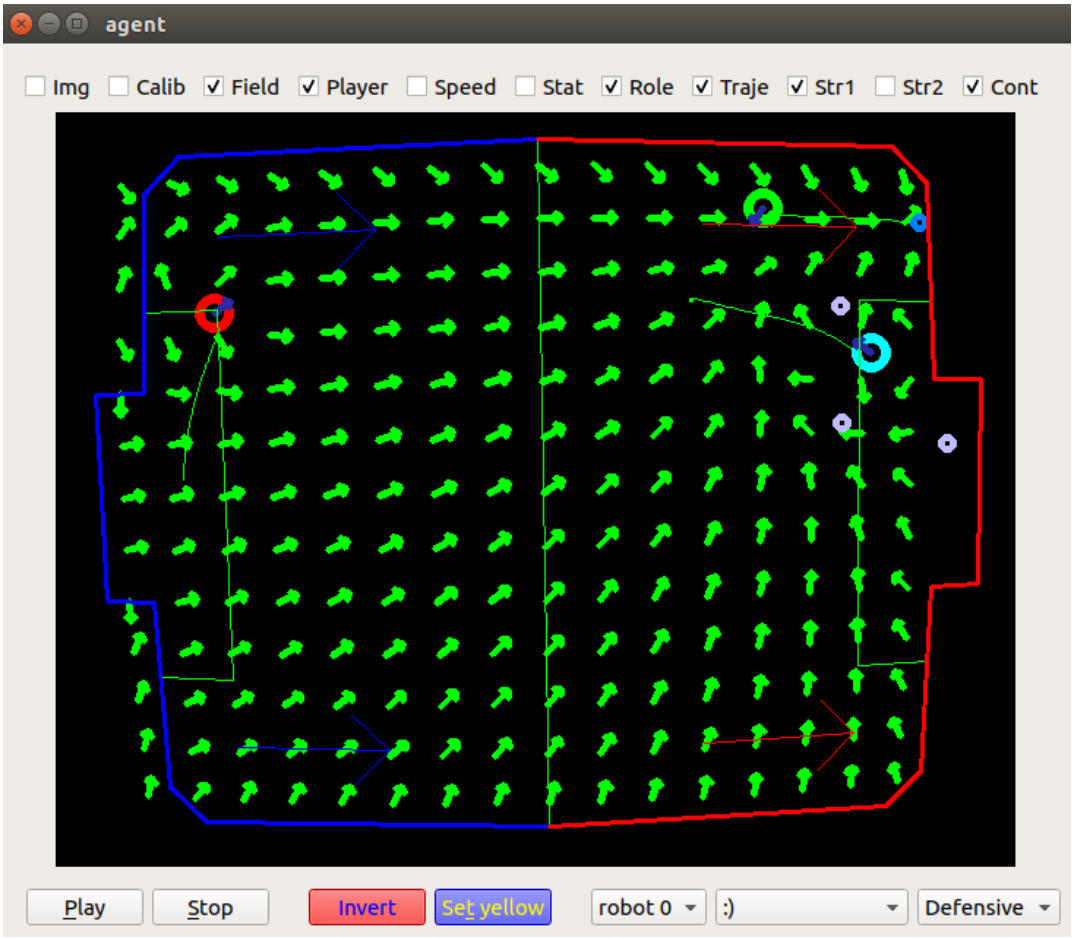

Figura 5: Interface gráfica para depuração da equipe

Na Figura 5, é possível observar, em um determinado instante, as trajetórias calculadas para cada jogador e os vetores do campo do Univector Field para o jogador verde.

2. RESULTADOS OBTIDOS

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 6.

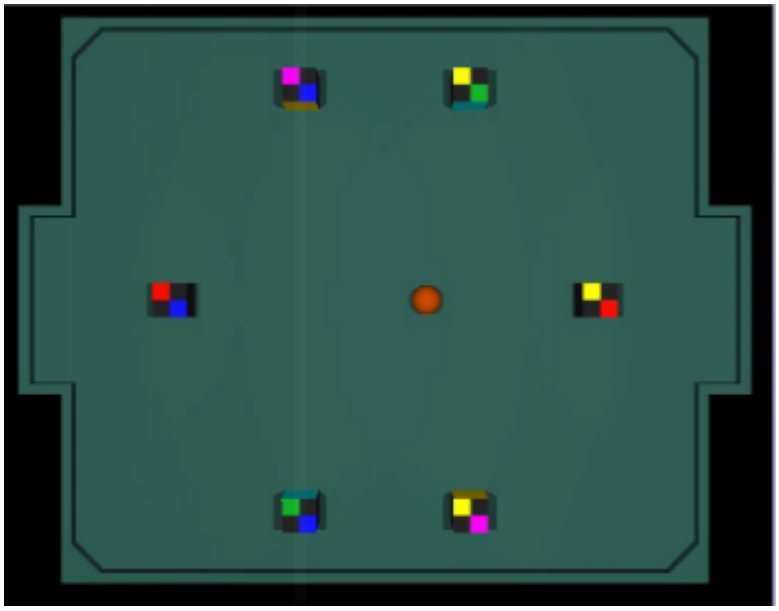

Figura 6: Simulador da ITAndroids.

2.1 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 7.

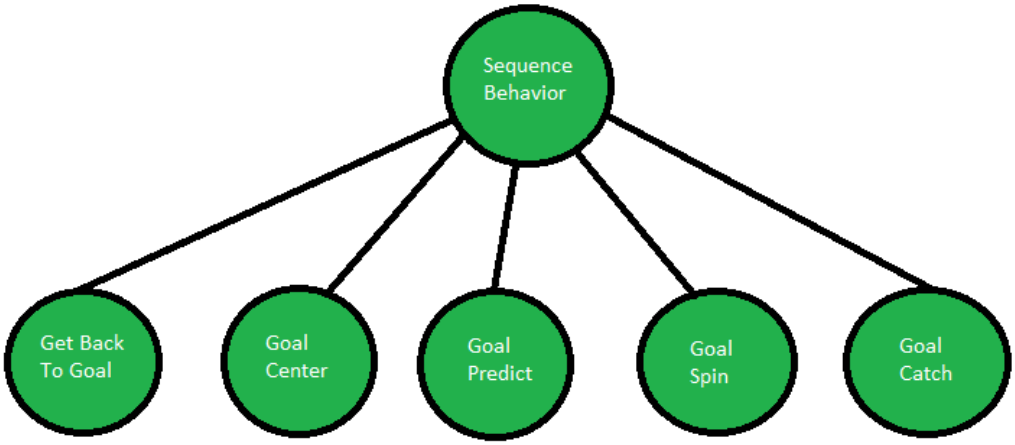

Figura 7: Behavior Tree para o goleiro.

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.

2.2 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 8 foi desenvolvida para o principal, sendo a árvore mais complexa dentre os papéis.

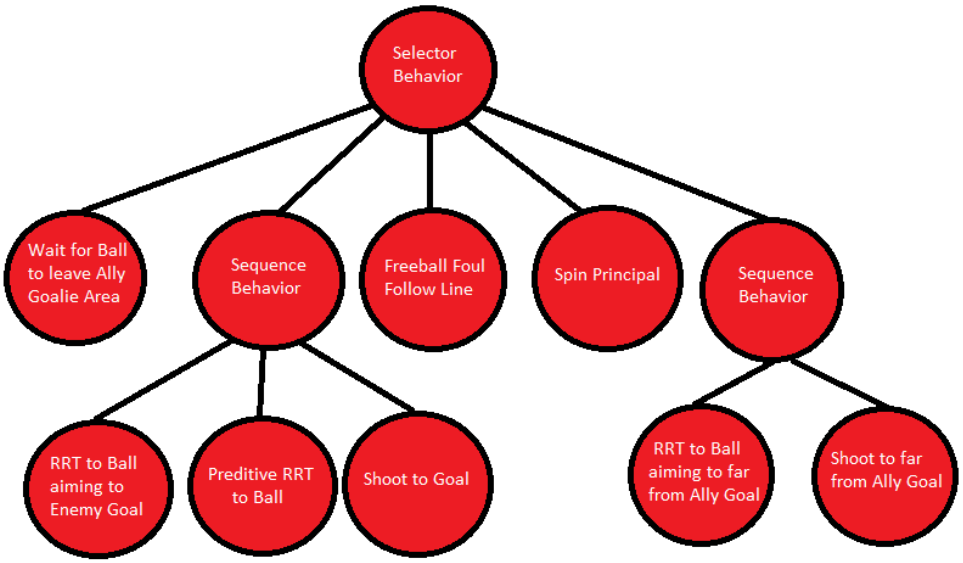

Figura 8: Behavior Tree para o principal.

Conforme a Figura 8, 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 9. O jogador se pocionará de acordo com posições escolhidas pelo usuário, calibradas com um arquivo de configuração .txt.

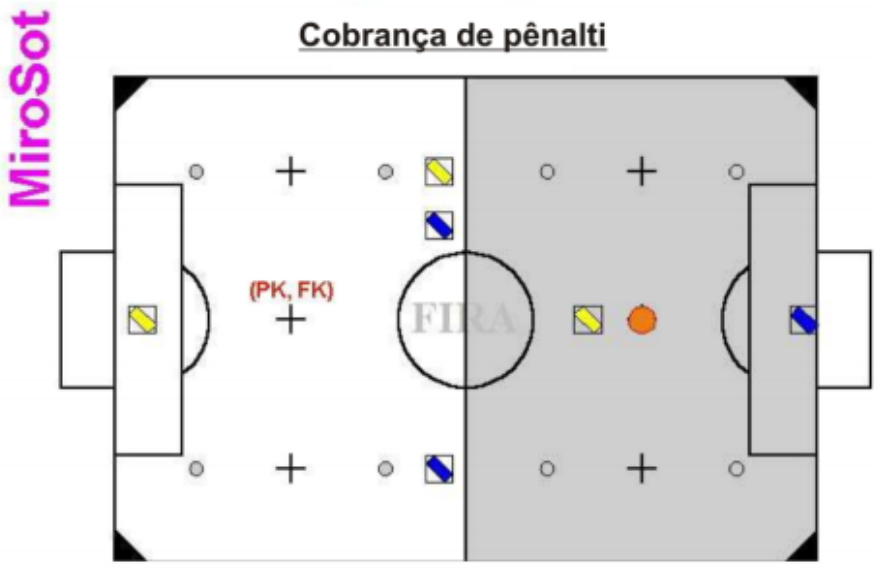

Figura 9: Descrição da penalidade pênalti.

- **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 10. 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.

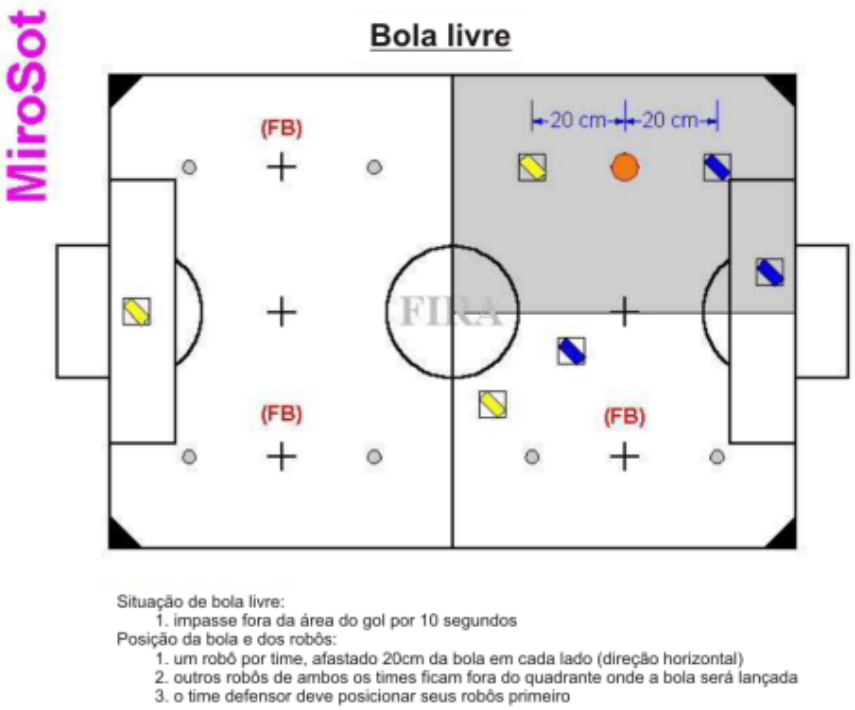

Figura 10: Descrição da penalidade bola livre.

- **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.

2.3 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 Delaunay (1934) sobre o grafo de Voronoi (1908). Esse algoritmo foi aplicado similarmente a Akiyama and Noda (2007), 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 11.

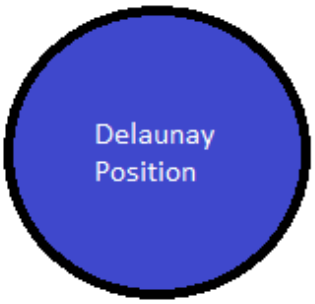

Figura 11: Behavior Tree para o auxiliar.

3. CONCLUSÕES

Nesse artigo, apresentou-se uma implementação do método Behavior Tree para a categoria de futebol de robôs Very Small Size Soccer, que foi descrita, brevemente, o seu funcionamento. Foi mostrado o esquema de papéis utilizado bem como a Árvore de Comportamentos implementada para cada um deles, além de uma descrição detalhada sobre todos os nós das árvores.

O projeto de Iniciação Científica apresentou bons resultados, especialmente quanto à vitória da equipe do ITA até às quartas de final da CBR 2017, rendendo o sétimo lugar à equipe dentre mais de 25 equipes participantes. Além do fato de que a ITAndroids conquistou o primeiro lugar na competição nacional de VSSS Copa Turing 2017, que ocorreu no final de setembro de 2017.

Do ponto de vista técnico, o uso do algoritmo da Behavior Tree, já consagrado e usado por times de nível mundial, se mostrou factível e funcional em partidas reais de competições. Contudo, problemas como pouca escalabilidade e pouca reutilizabilidade se mostraram presentes na estratégia desenvolvida. Isso significa que mudanças ainda deverão ser feitas nas BT, principalmente a respeito do uso de mais nós do tipo paralelo para remover esses problemas citados.

4. AGRADECIMENTOS

Agradeço ao CNPq, pelo apoio financeiro e motivacional.

Agradeço à ITAndroids, equipe que representa o ITA na competição da LARC/CBR, pela ideia do projeto, pela disponibilidade do robô real para implementação e oportunidade de aplicação dos métodos estudados.

Agradeço ao professor doutor Paulo Marcelo Tasinaffo, meu orientador, e ao professor doutor Marcos Ricardo de Omena Máximo, co-orientador, ambos da Divisão da Ciência da Computação do ITA, pelo apoio nos estudos e no desenvolvimento do projeto.

5. REFERÊNCIAS

Abiyev, R.H., Şenol Bektas, Akkaya, N. and Aytac, E., 2017. "Neuislanders 2017 team description paper". Robocup.

Akiyama, H. and Noda, I., 2007. "Multi-agent positioning mechanism in the dynamic environment". In *Robot Soccer World Cup*. Springer, pp. 377–384.

CBR, 2008. Http://www.cbrobotica. org/wp-content/uploads/2014/03/VerySmall2008_en.pdf.

Coelho, M.M.A., 2016. "Máquinas de estados finitos".

Delaunay, B., 1934. "Sur la sphére vide". *Bulletin de l'Académie des Sciences de l'URSS*, Vol. 6, pp. 793–800.

Hunt, J., 2013. "Gang of four design patterns". In *Scala Design Patterns*, Springer, pp. 135–136.

Okuyama, I.F., 2017. *TRAJECTORY PLANNING CONSIDERING ACCELERATION LIMITS FOR AN AUTONOMOUS SOCCER PLAYER*. Trabalho de graduação, Instituto Tecnológico de Aeronáutica.

Russel, S. and Norvig, P., 2009. *Artificial Intelligence: A Modern Approach*, Prentice Hall, Vol. 3, pp. 697–707. 3rd edition.

Voronoi, G., 1908. "Nouvelles applications des paramètres continus à la théorie des formes quadratiques". *Journal für die Reine und Angewandte Mathematik*, Vol. 133, pp. 97–178.

West, D.B. *et al.*, 2001. *Introduction to graph theory*, Vol. 2. Prentice hall Upper Saddle River.

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.

6. RESPONSABILIDADE AUTORAIS

Os autores são os únicos responsáveis pelo conteúdo deste trabalho.