Anais do XXV Encontro de Iniciação Científica e Pós-Graduação do ITA - XXV ENCITA / 2019
Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, Brasil, XXV 7 de novembro de 2019.

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.

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 CBR (2008).

Figura 1: Robôs da ITAndroids da categoria IEEE VSSS.

A Figura 1 mostra os robôs usados nas competições. A visão computacional utilizada pela equipe ITAndroids pode ser encontrada com mais detalhes em Zickler *et al.* (2009) e utiliza uma câmera no topo do campo que repassa informações para um computador processar, como visto na Figura 2.

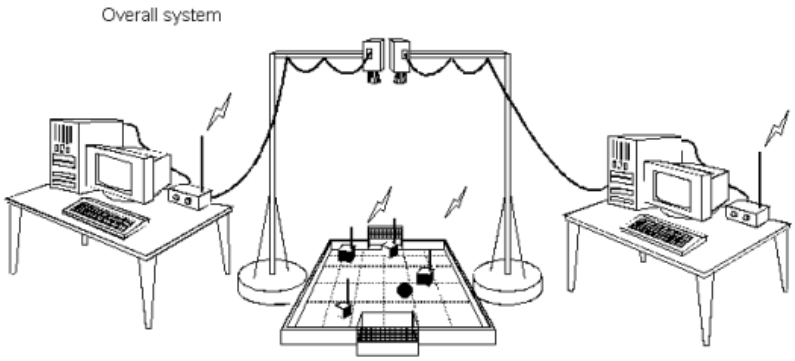

Figura 2: Representação do funcionamento técnico de um jogo de VSS.

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 Lim *et al.* (2008), 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 Koren and Borenstein (1991).

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 Hansen and Ostermeier (2001).

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 pro. 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 Ahnert and Mulansky (2011), para fazer as computações das equações diferenciais da física e das colisões dos movimentos, e da biblioteca Ogre3D Kerger (2010), responsável por renderizar os objetos na tela em três dimensões. A Figura 3 mostra a interface visual do simulador logo antes de se começar uma partida.

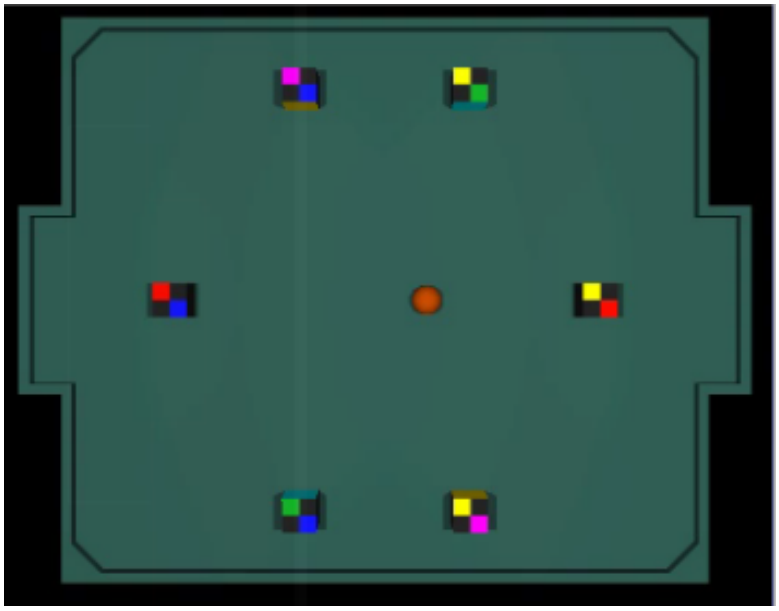

Figura 3: Simulador da ITAndroids.

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 Lim *et al.* (2008).

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.

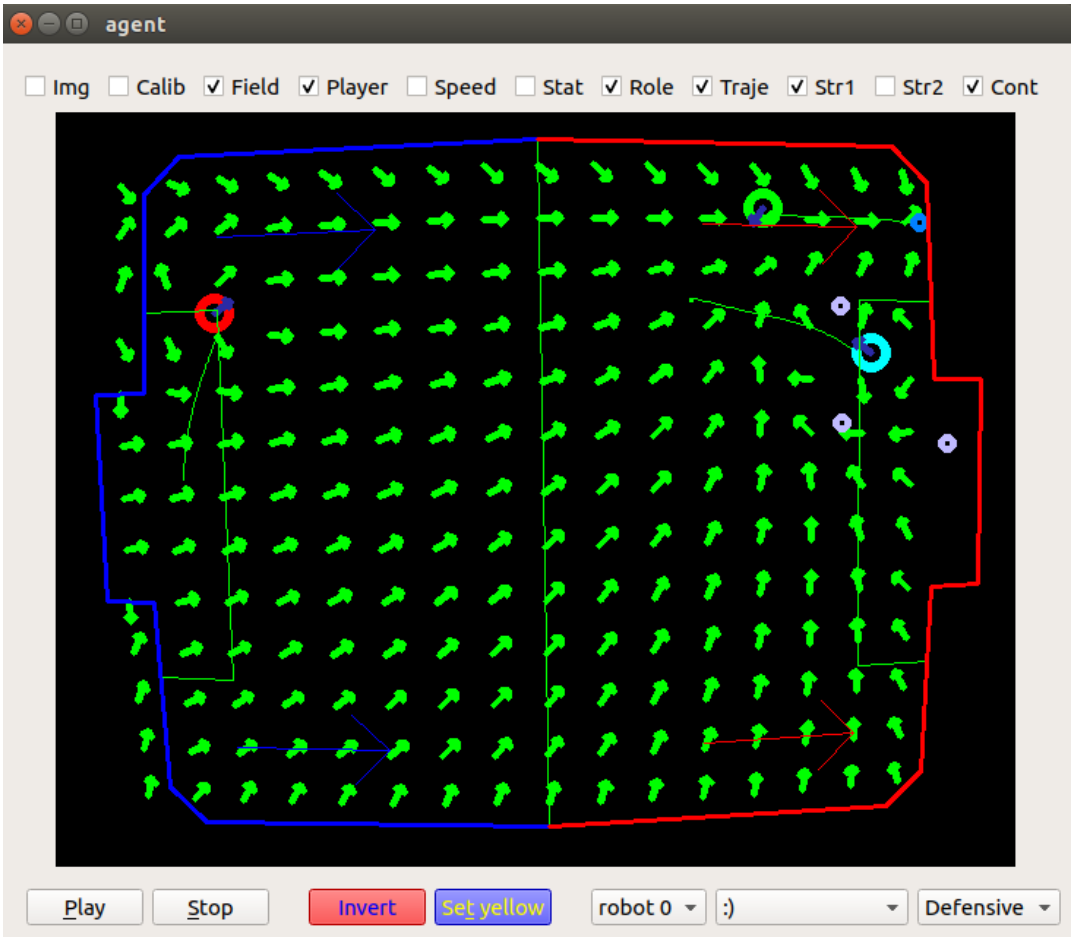

Figura 4: Interface gráfica para depuração da equipe

Na Figura 4, 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

Nessa seção, são apresentados os resultados obtidos na resolução do problema de otimização de parâmetros para robôs jogadores de futebol IEEE Very Small Size Soccer. Os resultados a seguir foram obtidos usando o simulador desenvolvido pela equipe.

2.1 Função de Custo

Aqui, será apresentado as situações de jogo escolhidas para serem simuladas para a otimização de parâmetros. É importante fixar que tais jogadas devem ser rápidas de serem executadas e realmente oferecerem resultados significantes para os parâmetros a serem otimizados. Cada situação escolhida deve ter uma função de custo associada que deve ser minimizada. Para a otimização em questão, foi desenvolvido duas situações de jogo com objetivo de otimizar variáveis diferentes. Elas são executadas sequencialmente, ou seja, a segunda situação de jogo usará os resultados obtidos da primeira situação.

2.1.1 Trajetórias Básicas

Nesse primeiro caso, será tratado o problema de um só jogador no campo com o objetivo de apenas jogar a bola para frente. Nessa situação, o foco será otimizar os parâmetros mais simples do planejamento de trajetória, que são todos não relacionados ao desvio de oponentes, mas incluindo os parâmetros relativos à repulsão de paredes.

O jogador e a bola são postos em diversas posições fixas no campo. Em seguida o jogador deve se mover até a bola e chutá-la ou conduzi-la até o final do campo do oponente. Quando a bola chega no final do campo, marca-se o tempo que demorou para esse objetivo ser atingido e começa o teste novamente para a próxima posição predeterminada. Caso o objetivo não seja alcançado em 3 minutos no tempo de jogo (a simulação é executada no modo acelerado), termina-se a simulação imediatamente.

Dessa forma, calcula-se a função de custo de acordo com a seguinte equação:

$F = tempoTotalDeTodasAsPosicoesRealizadas - numeroDePosicoesCompletadas.$

Nota-se que, de acordo com essa função fitness que deve ser minimizada, quanto menos tempo o agente completar todas a posições e quanto mais posições ele completar, melhor será o fitness calculado.

A Figura 5 mostra uma posição do jogador e da bola para essa situação de jogo.

Figura 5: Situação de jogo para otimização de trajetória simples

2.1.2 Desvio de Obstáculos

Esse caso é bastante similar à situação anterior no sentido que usará a mesma função de custo. A diferença consiste no fato de que, nessa situação, será posto um obstáculo (um jogador aliado) na frente da bola, de modo a obrigar que o agente principal desvie do obstáculo e rapidamente retorne à trajetória original para conseguir chegar na bola.

Nessa situação, será otimizado apenas as variáveis relativas ao desvio de obstáculos. Está sendo assumido que os parâmetros relativos à repulsão de paredes e os de planejamento de trajetória básicos já estão otimizados e funcionais.

A Figura 6 mostra uma posição do jogador, do obstáculo e da bola para essa situação de jogo. Percebe-se que o obstáculo está posicionado bem perto da bola, o que obriga o agente principal a fazer o desvio em sua trajetória. Nesse caso, o agente que vai tentar chegar na bola é o jogador rosa-azul, enquanto que os outros jogadores são obstáculos.

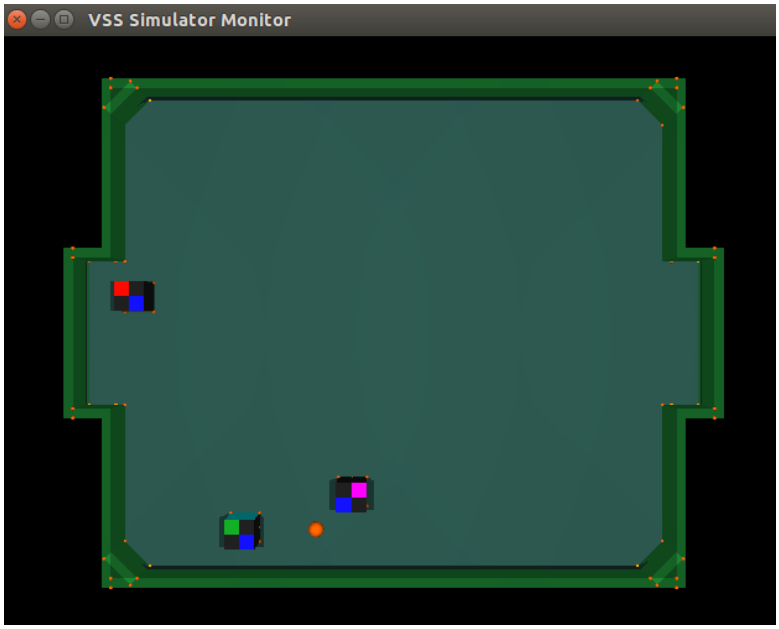

Figura 6: Situação de jogo para otimização de desvio de obstáculos

2.2 Otimização

Nessa secção, será apresentado os resultados obtidos ao executar o algoritmo de otimização CMA-ES com as funções de custo apresentadas anteriormente. Será mostrado e comentado como foi a convergência geral do algoritmo e dos parâmetros em cada caso.

2.2.1 Trajetórias Básicas

A Figura 7 mostra a evolução do custo da população com o passar das iterações. Foi realizado uma interpolação polinomial de terceira ordem para os resultados obtidos, de modo a melhor visualizar os efeitos da convergência. A interpolação pode ser visualizada pela linha vermelha, enquanto que os pontos 'X' representam o resultado de cada iteração em específico.

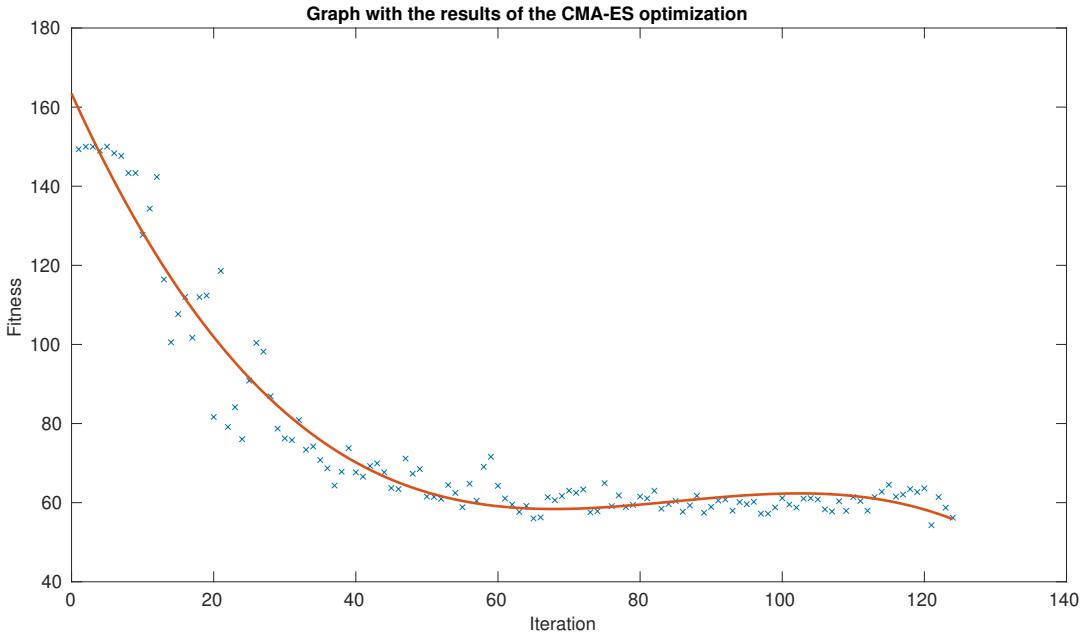

Figura 7: Fitness Cost em função do número de iterações

De acordo com a Figura 7, nota-se que, de fato, o algoritmo de otimização tende a reduzir o valor do Fitness Cost à medida que o número de iterações aumenta. Em torno de 60 iterações porém, o algoritmo já converge para um valor e

não acontece melhoras significativas a partir desse momento.

Uma normalização de dimensões foi feita em cada parâmetro separadamente e depois foi calculado a média desses resultados para cada iteração. Para a normalização, foi subtraído de cada valor a sua média e, em seguida, foi dividido o resultado obtido pela mesma média. Dessa forma, os valores dos parâmetros ficam com suas medidas mais próximas.

Na Figura 8 temos a convergência total dos parâmetros normalizada.

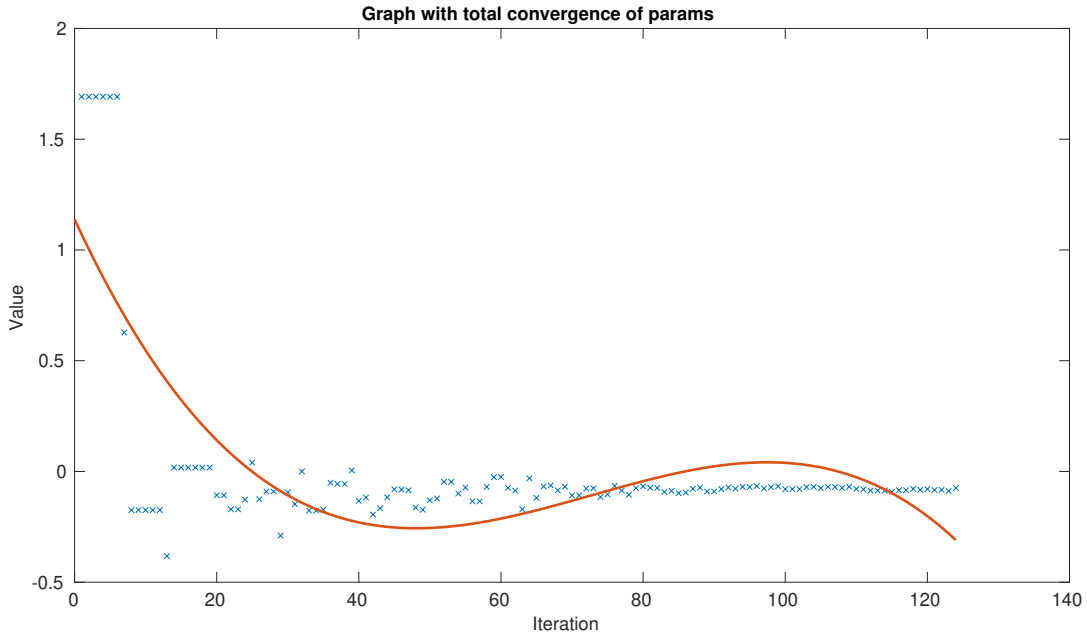

Figura 8: Convergência total dos parâmetros em função do número de iterações

Tanto a Figura 8, quanto a Figura 7 confirmam uma correta convergência do algoritmo, que converge em torno de 100 iterações.

2.2.2 Desvio de Obstáculos

A Figura 9 mostra a evolução do custo em função de uma determinada iteração do algoritmo para a otimização de desvio de obstáculos. Nota-se que os pontos nessa figura são mais esparsos que o da Figura 7, o que é esperado, já que a otimização de desvio de obstáculos é mais complexa que a de trajetórias básicas.

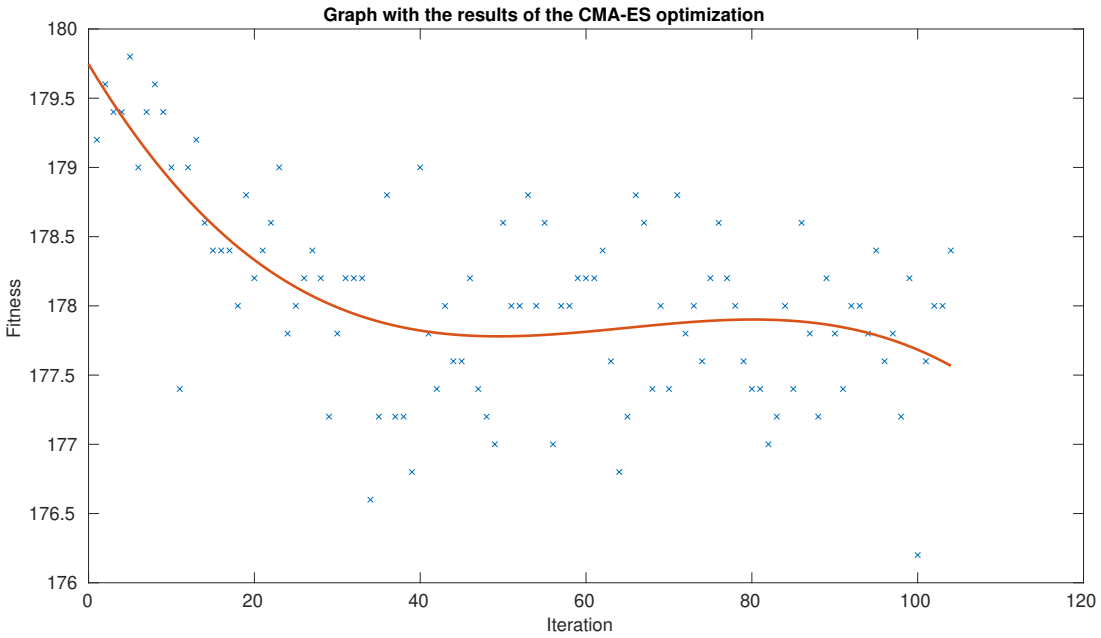

Figura 9: Fitness Cost em função do número de iterações

Na Figura 10 temos a convergência total dos parâmetros normalizada. A normalização foi feita em cada parâmetro separadamente e depois foi calculado a média desses resultados para cada iteração. Tal normalização foi realizada com o mesmo procedimento da seção Trajetória Básicas.

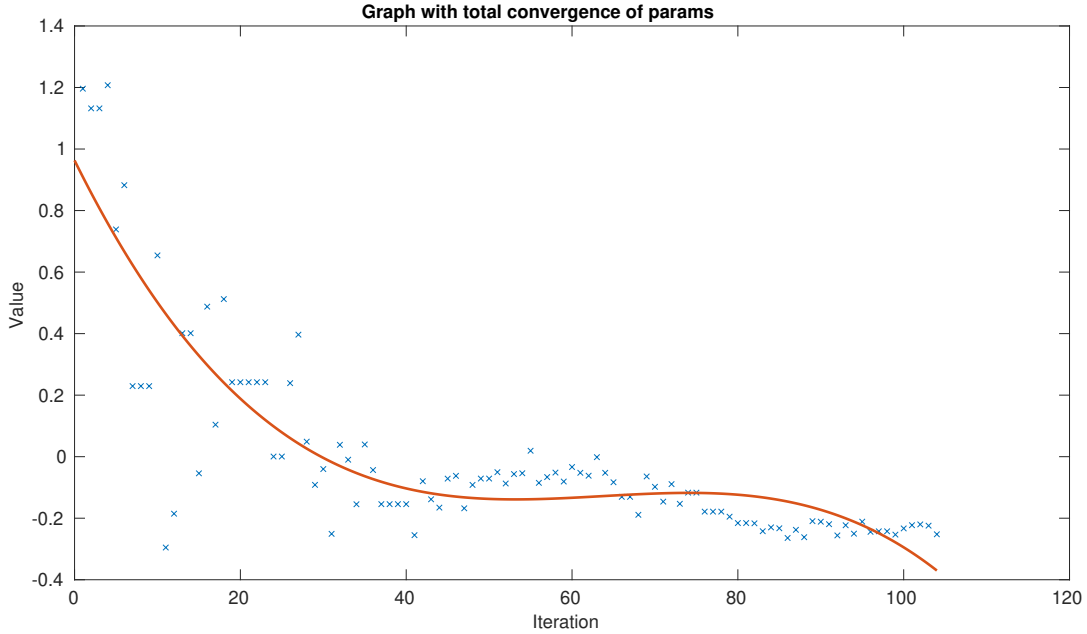

Figura 10: Convergência total dos parâmetros em função do número de iterações

Tanto a Figura 10, quanto a Figura 9 confirmam uma correta convergência do algoritmo, que converge em torno de 100 iterações.

3. CONCLUSÕES

O projeto de Iniciação Científica apresentou bons resultados, especialmente quanto à vitória da equipe do ITA na competição CBR 2018, rendendo o primeiro lugar à equipe, na primeira vez da história, dentre mais de 25 equipes participantes de toda América Latina.

Do ponto de vista técnico, o uso do algoritmo Covariance Matrix Adaptation Evolution Strategy se mostrou factível e funcional para otimização de parâmetros para o planejamento de trajetórias, com resultados demonstrados em partidas oficiais.

Com a estrutura de otimização implementada para a equipe, próximos passos incluem utilizar esse algoritmo para outras áreas do código, como otimização na tomada de decisão do time de robôs.

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 Celso Massaki Hirata, 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

, ???? Https://developers.google.com/protocol-buffers/docs/cpptutorial.

Ahnert, K. and Mulansky, M., 2011. "Odeint–solving ordinary differential equations in c++". In *AIP Conference Proceedings*. AIP, Vol. 1389, pp. 1586–1589.

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

Hansen, N. and Ostermeier, A., 2001. "Completely derandomized self-adaptation in evolution strategies". *Evolutionary computation*, Vol. 9, No. 2, pp. 159–195.

Kerger, F., 2010. *OGRE 3D 1.7 Beginner's Guide*. Packt Publishing Ltd.

Koren, Y. and Borenstein, J., 1991. "Potential field methods and their inherent limitations for mobile robot navigation". In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, pp. 1398–1404.

Lim, Y., Choi, S.H., Kim, J.H. and Kim, D.H., 2008. "Evolutionary univector field-based navigation with collision avoidance for mobile robot". In *Proc. 17th World Congress The International Federation of Automatic Control, Seoul, Korea (July 2008)*.

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.