INSTITUTO TECNOLÓGICO DE AERONÁUTICA
CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO - CNPq

CNPq
CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC

Estratégia de time de Robôs Very Small Size

Igor Mourão Ribeiro

RELATÓRIO PARCIAL DE ATIVIDADES

Orientador: Paulo Marcelo Tasinaffo

Janeiro / 2018

INSTITUTO TECNOLÓGICO DE AERONÁUTICA
CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO - CNPq

CNPq
CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC

Relatório Parcial de Atividades

Estratégia de time de Robôs Very Small Size

São José dos Campos, 03/01/2018

Nome do aluno	Igor Mourão Ribeiro
Assinatura do aluno	

Nome do orientador	Paulo Marcelo Tasinaffo
Assinatura do orientador	

INSTITUTO TECNOLÓGICO DE AERONÁUTICA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC

Formulário de Aprovação de Relatório pelo Orientador

Relatório: [X] Rel. Parcial [ ] Rel. Final

1- CONSIDERO O RELATÓRIO APROVADO COM BASE NOS SEGUINTES ASPECTOS

2- APRECIAÇÕES DO ORIENTADOR SOBRE O DESEMPENHO DO BOLSISTA NA EXECUÇÃO DO TRABALHO DE INICIAÇÃO CIENTÍFICA

Local e data:

Assinatura do Orientador:

Sumário

1 Resumo do Plano Inicial

O objetivo deste trabalho é desenvolver e implementar um algoritmo para a estratégia de um time composto por três robôs diferenciais. O contexto é uma partida de futebol segundo a Competição Brasileira de Robótica (CBR), categoria IEEE Very Small Size Soccer. A partir da posição de cada robô, incluindo robôs oponentes, e da bola, deve-se obter a melhor ação que cada robô aliado deverá executar para que faça o maior número de gols no fim de uma partida considerando limitações mecânicas e de controle. Visa-se utilizar os algoritmos aqui implementados pela equipe da ITAndroids, que representa o ITA, na competição da Latin America Robotics Competition (LARC)/CBR de 2017 em diante.

Planejamento:

- 1o Bimestre (ago / set): Estudo da literatura e revisão bibliográfica sobre o tema de tomada de decisão para robôs jogadores de futebol.
- 2o Bimestre (out / nov): Seleção de métodos a serem implementados para planejamento e codificação da implementação.
- 3o Bimestre (dez / jan): Codificação de testes automatizados para comprovação da correta implementação do algoritmo. Redação do relatório parcial.
- 4o Bimestre (fev / mar): Simulações computacionais de um partida completa para análise e adaptações do método em cenários realistas de futebol de robôs.
- 5o Bimestre (abr / mai): Análise dos algoritmos implementados em termos de carga computacional e eficácia. Integração da estratégia com o restante do código.
- 6o Bimestre (jun / jul): Redação do relatório final. Redação do artigo para o ENCITA. Teste final para o software completo no robô real, com todas as áreas do projeto operantes.

2 Resumo das Atividades Realizadas

Ao longo dos dois primeiros meses, foram realizados estudos e foi decidido que será usado o algoritmo Behavior Tree (Árvore de Comportamentos) para tomada de decisão. Esse

algoritmo foi escolhido dentre de opções como Máquina de Estados Finita e Árvore de Decisões. Uma introdução a esses assuntos pode ser encontrada em Orozco [1], para máquina de estados finita, e em Russel e Norvig [2], para Árvore de Decisões. Para uma descrição sobre a eficácia do algoritmo Behavior Tree (BT) no contexto da robótica, pode-se encontrar na referência [3]

Ao longo do segundo e terceiro bimestres, a base do algoritmo foi implementada utilizando C++, que é a linguagem utilizada pela equipe. Foi, em seguida, realizada a codificação dos testes de unidade para os componentes principais do algoritmo, os nós intermediários da árvore. A performance computacional da BT foi analisada e mostrou resultados satisfatórios e condizentes com o que foi previsto. A estrutura básica do algoritmo estava pronta.

A parte da BT que considera as limitações de um robô real foi implementada e testada no time. O método foi mostrado efetivo na competição, ocorrida entre os dias 7 a 11 de novembro de 2017. O algoritmo mostrou que realiza troca de posições entre jogadores de forma fluida, de modo que a equipe tinha o posicionamento mais dinâmico da competição. A equipe da ITAndroids conquistou o sétimo lugar de 30.

3 Descrição do Problema

No contexto do futebol de robôs, a tomada de decisão traz muitos desafios. A estratégia consiste no fato de se implementar um algoritmo que consiga utilizar o planejamento de trajetórias do robô da melhor maneira possível. Isso nos traz a um desafio de nível mais alto: decidir qual a melhor decisão que cada robô pode tomar em um certo momento considerando limitações de trajetórias possíveis de serem seguidas.

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 possuil 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 [4].

O robô foi projetado com duas rodas laterais e duas esferas livres à frente e atrás para

manter o equilíbrio. 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++, pois é uma linguagem que tem uma velocidade de execução alta e tem escalabilidade descente para um projeto grande.

Figura 1: A categoria IEEE VSSS.

Nesse contexto, surgem várias questões a serem resolvidas para se ter um time vencedor. Nesse projeto, será abordado o seguinte problema de estratégia: dada a posição e a velocidade de cada robô no campo, da bola e da direção que os nossos robores estão (não é possível descobrir a posição dos robôs oponentes, pois não sabemos a distribuição de cores que eles usam no seu topo), deve-se escolher qual posição o robô irá, com qual controlador e qual planejamento de trajetórias utilizar. Deve-se também montar um plano para ações futuras, planejando uma sequência de posições que o robô deve ir para efetuar um gol.

O algoritmo de estratégia deve executar em tempo de execução de, no máximo, 16 ms para todo o time, que é o tempo de amostragem da câmera. A maior parte desse tempo é gasto com o planejamento de trajetórias usado, que é o Rapidly-exploring Random Tree (RRT). O tempo para cada execução de um RRT é de 1 ms, de acordo com [5], limitando a em torno de um limite de doze chamadas do algoritmo por execução completa da estratégia, com uma margem de erro trinta e três porcento. Ressalta-se a importância de não ultrapassar esse tempo máximo, já que se isso acontecer perde-se uma imagem da câmera e o funcionamento do jogo é realizado de forma atrasado.

Deve-se, então, ser montado um esquema de posicionamento dinâmico em que não existam nem atacante, nem zagueiro, nem goleiro fixos, mas sim, uma ideia de que as posições dependam da posição de jogo, garantindo que os três jogadores possam estar

mais bem posicionados e tenham liberdade para escolherem suas próximas ações.

Para resolver esse problema, foi escolhido como principal algoritmo para a tomada de decisão a Behavior Tree, embasado na decisão de um time referência no futebol de robô mundial na categoria Small Size League [6], the Near East University team: the NEUIslanders[7].

3.1 Behavior tree

Para se construir uma BT, é preciso primeiro entender o básico de sua estrutura. Uma BT é uma Árvore no sentido dado pela teoria dos grafos, como explicado em [8]. Por definição, cada nó dessa árvore será chamado de behavior.

Os nós de uma Behavior Tree são diferenciados em dois tipos: os nós folhas, que são os nós da árvore que não tem filhos e os nós compostos, que podem ter um ou mais filhos.

Todos os nós retornam, no final de sua execução, um dos três estados a seguir: TRUE, FALSE ou RUNNING, para indicar, respectivamente, que o nó foi terminado com sucesso, que foi terminado sem sucesso ou que o algoritmo deve executar esse nó por pelo menos mais uma iteração.

3.1.1 Nó Folha

Os nós folha são os nós terminais da Árvore e eles que representam as ações mais baixo níveis e concretas da estratégia. São divididas em dois tipos: nós de ação e nós condicionais.

Nó de Ação São os nós que de fato realizam uma ação, como se movimentar até a bola.

Nó condicional São nós que apenas retornam um estado sem realizar nenhuma ação concreta, como checar se o time está atacando.

3.1.2 Nó composto

Nós compostos são os nós intermediários da árvore, que são resposáveis 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 estraegia 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. 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 explicado em [9]. 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 fiho 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.

3.2 Behavior Tree VSS implementation

Seja S_0 a posição inicial do jogador e v_0 sua direção inicial. Seja S_f a posição desejada e v_f a direção com que se deseja chegar a S_f. A ideia de campos potenciais consiste em modelar o campo com uma função potencial $P(x,y)$, de forma que, executando o algoritmo de descida de gradiente partindo do ponto inicial, gera-se a sequência de pontos $S'(i)$ segundo a seguinte fórmula recursiva:

$$S'(1) = S_0 + (h/2)v_0 \tag{1}$$

$$S'(i) = S'(i-1) - h\nabla P, i = 2,...,n \tag{2}$$

Para escolher uma função P adequada, foi usada a função potencial elétrico com cargas positivas nos obstáculos. Seja a equação de Poisson:

$$\varepsilon\nabla \cdot E = -\varepsilon\nabla^2 V = \rho = 0 \tag{3}$$

Pelo teorema do máximo e mínimo, funções com laplaciano zero não podem ter pontos de mínimo não degenerados no interior de um conjunto em que são analíticas. Isso faz com que o algoritmo convirja para alguma singularidade negativa ou divirja. Assim sendo, basta colocar uma carga negativa no ponto desejado, o que produz uma singularidade negativa naquele ponto.

Como o espaçamento entre os pontos é constante, pode-se obter a sequência de posições $S(i)$ e de direções $v(i)$ da seguinte maneira: A posição por onde passará o robô é o ponto médio entre dois pontos calculados $S'(i)$ e $S'(i-1)$. O módulo de $v(i)$ é unitário e sua direção e sentido coincidem com o vetor $S'(i-1)$ a $S'(i)$. $v(1) = v_0$. As equações para $S(i)$ são dadas a seguir:

$$S(1) = S_0 \tag{4}$$

$$S(i) = \frac{(S'(i) + S'(i-1))}{2}, i = 2,3,...,n \tag{5}$$

$$S(n+1) = S'(n) + \frac{h}{2}v_f \tag{6}$$

A partir destas equações, podemos concordar arcos de circunferência entre os pontos $S(i)$ e $S(i+1)$ segundo a figura 3.1.2.

Para configurar os campos de forma a que eles sejam bem condicionados e produzam uma trajetória adequada, são usados também os seguintes artifícios:

- Dipolo elétrico no alvo alinhado com a direção de chegada desejada. A carga positiva fica à frente e a negativa atrás. Para que o efeito do dipolo seja sentido em todo o campo, sua carga é da ordem de 100 vezes maior que as outras. O padrão das linhas de campo faz com que a trajetória, que as seguem aproximadamente, chegue no ângulo desejado.

- Existe um limite para o qual o campo repulsivo atua no dipolo da bola. Entre esse limite e o dobro dele, o campo é multiplicado por um fator que decai continuamente para 0. Em pontos distantes, o alvo é puramente atrativo.

- Cada parede gera um campo normal à parede que decai pela distância da mesma forma que o campo elétrico de uma carga. O cálculo é feito da seguinte maneira: a fim de calcular o campo da parede em um ponto C, projeta-se C na parede e considera-se que seja um obstáculo com uma carga positiva menor.

- Para que se mantenha uma velocidade constante no robô, pode-se definir um raio de curvatura mínimo R_{min}. Para impor esta restrição no algoritmo de campos potenciais, pode-se calcular um θ_{max} de acordo com a figura 3.1.2. A fórmula é dada a seguir:

$$\theta_{max} = arctan(\frac{4hR_{min}^2}{4R_{min}^2 - h^2}) \tag{7}$$

- Caso, durante um passo da descida gradiente, o ângulo entre os trechos $S'(i) - S'(i-1)$ e $S'(i+1) - S'(i)$ seja maior que θ_{max}, faz-se o passo com o ângulo θ_{max} de forma a que $S'(i+1)$ seja o mais próximo possível do que se fosse usado o ângulo do campo.

3.3 RRT

Este algoritmo consiste em expandir árvores em direções aleatórias a partir da posição do robô e do alvo. Durante a expansão, quando dois nós, um de cada árvore, estiverem a uma distância menor que um parâmetro h, conecta-se esses dois nós e forma-se um caminho do robô até o alvo. O fator de aleatoriedade garante uma probabilidade admissível de se achar um caminho. Para que sejam respeitadas as restrições do campo e dos obstáculos, basta que os vértices as respeitem. O algoritmo foi estudado segundo a aula [10].

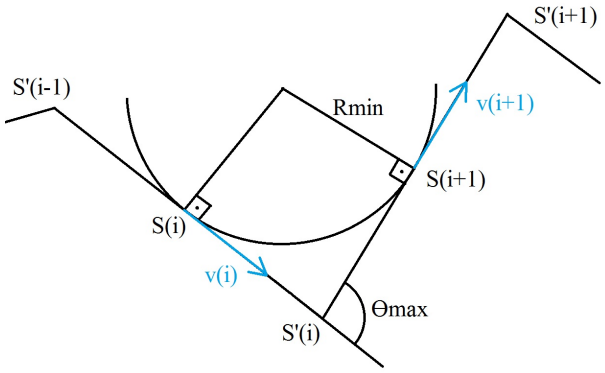


Figura 2: Circunferência concordada por dois trechos de S' e tangenciando em S

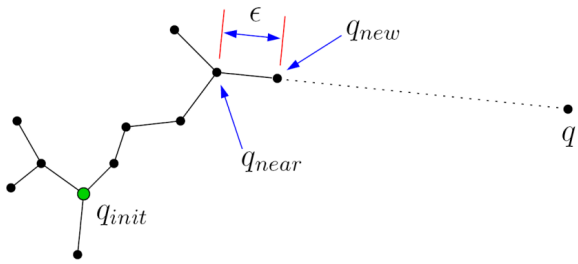


Figura 3: Exemplo de passo de extensão do RRT. Fonte: LaValle, 2000 e [10]

Para minimizar o tempo de expansão, deve-se evitar que duas arestas - os segmentos de reta no espaço que ligam dois vértices vizinhos - se interceptem. Para manter o bom condicionamento e otimizar o cálculo entre duas iterações, devem-se definir probabilidades G de se escolher como novo ponto a raiz da outra árvore e P a de se escolher um ponto da última trajetória calculada. A função PA (Ponto Aleatório) retorna tal ponto. A técnica foi proposta segundo o artigo [11]. Para isso, foram implementadas as seguintes funções:

É necessário também colocar um limite no número de nós das árvores para que, caso não haja caminho possível, o algoritmo não entre em loop infinito. Para garantir que sejam respeitadas as direções iniciais e finais, adicionaram-se obstáculos ao lado das posições finais e iniciais, de forma que as árvores possam expandir apenas se alinhando entre eles.

Para se otimizar o passo de escolher o ponto q_{near} de uma árvore mais próximo a um ponto Q, foi implementada a estrutura de dados Quadtree, que permite a inserção e busca de ponto mais próximo em $O(\log n)$, enquanto a inserção e busca normais são, respctivamente, $O(1)$ e $O(n)$. Os resultados de ambos os métodos foram comparados.

Um dos problemas identificados é a possibilidade de mal condicionamento do algoritmo: a cada passo, ele pode gerar um caminho distinto. Além do caminho atual ser

Algorithm 1 função Estender

procedure ESTENDER (ÁRVORE T, PONTO q)
  $q_{near} \leftarrow$ PontoMaisPróximoDe(T, q)
  $n \leftarrow$ direção de q_{near} a q
loop:
  if ($q_{near} + n * h$ é válido e $d(q, q_{near}) > d(q, q_{near} + n * h)$ **then**
    Adiciona $q_{near} + n * h$ *a* T.
    Liga $q_{near} + n * h$ *a* q_{near}.
    $q_{near} \leftarrow q_{near} + n * h$.
    goto *loop*.
    close;
  return q_{near}.

Algorithm 2 função CalcularTrajetória

procedure CALCULARTRAJETÓRIA (S_0, v_0, S_f, v_f)
  $T_a \leftarrow S_0 + (h/2) * v_0$
  $T_b \leftarrow S_f - (h/2) * v_f$
loop:
  if *(*$|T_a| \leq M$ *e* $|T_b| \leq M$*)* **then**
    $P_A \leftarrow$ *Estender(*T_a*, PAl).*
    $P_B \leftarrow$ *Estender(*T_b*,* P_A*).*
    if $d(P_A, P_B) \leq h$ **then**
      close *loop*.
    Troca T_a *e* T_b.
    goto *loop*.
    close;
  return *Trajetória raiz(*T_a*)-*P_A*-*P_B*-raiz(*T_b*).*

orientado para ser semelhante ao da iteração anterior, deve-se executar o algoritmo várias vezes e tomar o menor caminho de cada execução. O menor caminho converge para o mínimo possível.

Deve-se, também, suavizar a trajetória. Percorre-se a trajetória inteira calculando os ângulos entre dois trechos. Caso esse ângulo ultrapasse um limite θ_{max}, definido e calculado a partir de R_{min} da mesma forma descrita nos campos potenciais, elimina-se esse ponto da trajetória e liga-se seus pontos adjacentes. No passo de suavização por arcos de circunferência e na ponte entre as duas árvores, pode-se gerar trechos que não têm tamanho h. Basta alterar a concordância dos arcos para que eles comecem ou terminem na metade do trecho de menor comprimento. Assim, basta uma pequena alteração nas equações de concordância para que se obtenham as sequências finais S e v e as velocidades linear e angular.

3.4 Otimização com Mixed Integer-Linear Programming (MILP)

Modelar um problema qualquer como um problema de otimização significa definir uma função de erro cujo ponto de mínimo que respeita um conjunto de restrições é a solução. A partir disso, pode-se usar algoritmos computacionais para achar esse ponto.

Foi utlizado, para realizar este cálculo, a biblioteca GLPK para MATLAB. A vantagem dela é o trabalho com Programação Linear Mista-Inteira, o que permite adicionar restrições de tipo nos vetores de variáveis: eles podem ser de reais, inteiros ou binários com limites inferior e superior.

Primeiro definimos a função de custo e as restrições como

$$J(b) = \sum_{i=1}^{\log_2 N} b(i)2^{i-1} = n, \tag{8}$$

$$Av \leq B \; ou \; \sum_{j=1}^{N_{var}} A_{ij}v(j) \leqslant B_i \tag{9}$$

onde N é o número máximo de passos para a trajetória tomado sempre como potência de 2 e N_{ver} o número de veriáveis. $b(i)$ é um vetor de variáveis binárias que representa a codificação binária de n, o número de passos utilizados. Minimizar $J(b)$ significa percorrer o caminho com o mínimo de passos possíveis.

O pacote GLPK permite adicionar restrições por meio da equação (9), onde A é uma matriz inserida pelo usuário, v é a concatenação de todos os vetores de variáveis (reais, inteiras e binárias) e B é um vetor de constantes também definido pelo usuário. O algoritmo determinará v de forma a que todas as inequações contidas em (9) sejam satisfeitas e que $J(b)$ seja minimizado.

Isso significa que podemos adicionar desigualdades sobre uma determinada combinação linear de variáveis. Para preencher as matrizes A e B com os valores que representem todas as retrições do problema, foram feitas duas abordagens.

3.4.1 Navegação sobre todo o campo

Nesta abordagem, teremos como variáveis controladas pelo otimizador a parte positiva e negativa das acelerações em x (x_p e x_n, respectivamente) e em y (y_p e y_n, respectivamente). Impõem-se restrições de mínimo zero e máximo a_{max}. Sejam $u(i)$ a aceleração no i-ésimo passo, ou seja, $u(i) = (x_p(i) - x_n(i), y_p(i) - y_n(i))$, T o tempo de cada passo, $S(1) = S_0$

e $v(1) = v_0$ as posições iniciais do robô, então as equações para o movimento na i-ésima posição são:

$$v(i) = v(1) + T\sum_{j=1}^{i-1} u(j) \tag{10}$$

$$S(i) = S(1) + (i-1)Tv(1) + T^2\sum_{j=1}^{i-1}(n - j - \frac{1}{2})u(j) \tag{11}$$

Para restringir a velocidade máxima, basta adicionar desigualdades do tipo:

$$-V_{max} \leq v(i)_x, v(i)_y \leq V_{max} \tag{12}$$

Seja ϵ um erro admissível para a posição final do robô. Para se chegar na posição final desejada com a direção desejada, devemos adicionar restrições sobre a n-ésima posição calculada. Seja $bin(i,j)$ o j-ésimo algarismo de i em sua representação binária. Para implementar dessa maneira, adiciona-se um termo extra $C(i,b)$ definido da seguinte forma:

$$C(i,b) = \sum_{j=0}^{\log_2 N} c(i,j)b(j) \tag{13}$$

$$c(i,0) = \sum_{j=1}^{\log_2 N} bin(i,j) \qquad c(i,j) = \begin{cases} 1, se\ bin(i,j) = 0 \\ -1, se\ bin(i,j) = 1 \end{cases} \tag{14}$$

$C(i,b)$ é uma função que recebe a representação binária de um inteiro por meio do vetor b e um inteiro i. Ela retorna 0 caso b represente i em binário e um valor positivo caso contrário. Seja M um número suficientemente grande. Para $i = 1,...,N$, as inequações de restrição, assim, ficam da seguinte forma:

$$x_f - \epsilon - MC(i,b) \leq S(i)_x \leq x_f + \epsilon + MC(i,b) \tag{15}$$

$$y_f - \epsilon - MC(i,b) \leq S(i)_y \leq y_f + \epsilon + MC(i,b) \tag{16}$$

Dessa forma, quando o otimizador escolher um valor para n por meio de b, essas equações serão relaxadas (reduzidas a $\infty \geq x$) para passos diferentes de n e impostas (último termo eliminado) caso contrário. Da mesma forma, garantimos que a velocidade de chegada seja a desejada por meio das equações:

$$v_{fx} - \epsilon - MC(i,b) \leq v(i)_x \leq v_{fx} + \epsilon + MC(i,b) \tag{17}$$

$$v_{fy} - \epsilon - MC(i,b) \leq v(i)_y \leq v_{fy} + \epsilon + MC(i,b) \tag{18}$$

Para as restrições de campo, basta adicionar as desigualdades:

$$-X_{max} \leqslant S(i)_x \leqslant X_{max}, \qquad -Y_{max} \leqslant S(i)_y \leqslant Y_{max} \tag{19}$$

Enfim, para representar as restrições de obstáculos, modelou-se cada um como um quadrado de aresta L. Adicionaram-se duas variáveis binárias que formam números de 0 a 3 para cada obstáculo. A função $obs(i, j)$, para o i-ésimo passo e para o j-ésimo obstáculo, representa a seguinte codificação:

- $obs(i, j) = 0$: posição abaixo da linha inferior do obstáculo.
- $obs(i, j) = 1$: posição à direita da linha direita do obstáculo.
- $obs(i, j) = 2$: posição acima da linha superior do obstáculo.
- $obs(i, j) = 3$: posição à esquerda da linha esquerda do obstáculo.

Logo, para todo i e j, basta adicionar restrições do tipo:

$$x_{obs}(j) + \frac{L}{2} - \epsilon - MC(1,obs(i,j)) \leq S(i)_x \leq x_{obs}(j) - \frac{L}{2} + \epsilon + MC(3,obs(i,j)) \tag{20}$$

$$y_{obs}(j) + \frac{L}{2} - \epsilon - MC(0,obs(i,j)) \leq S(i)_y \leq y_{obs}(j) - \frac{L}{2} + \epsilon + MC(2,obs(i,j)) \tag{21}$$

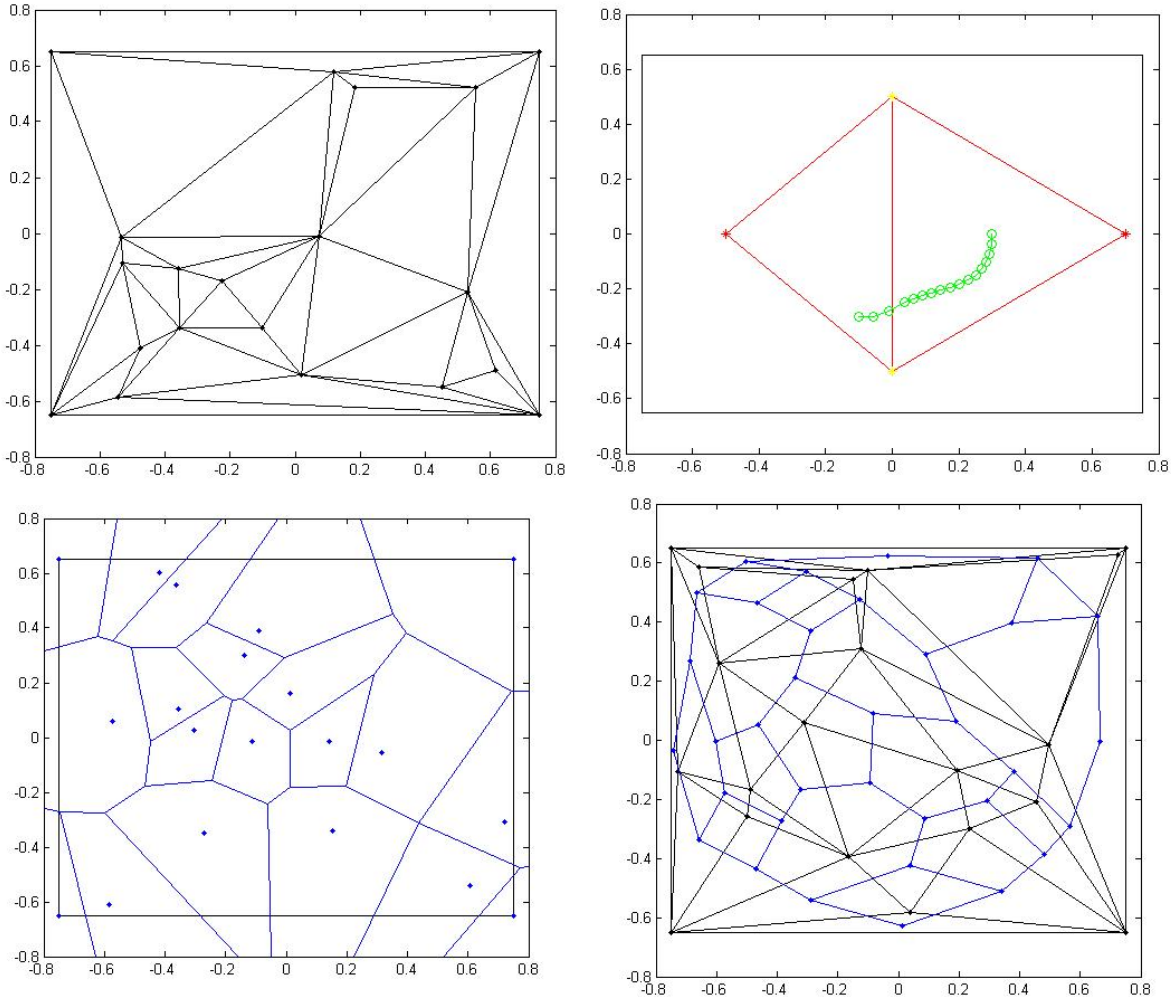
3.4.2 Navegação por uma triangulação

Outro jeito de se utilizar o GLPK é da seguinte maneira: suponha que tenhamos uma sequência de triângulos adjacentes tal que o ponto de saída está no primeiro triângulo e o de chegada, no último. Podemos usar o pacote para achar a trajetória por dentro de um triângulo até o próximo.

Para impôr as restrições de desvio de obstáculos e de parede, usa-se as quinas do campo e as pontas dos obstáculos para computar a triangulação de Delaunay [12]. Assim, podemos obter um grafo de triângulos por onde poderemos estimar o melhor caminho. Basta utilizar o algoritmo de Dijkstra [13] sobre o grafo de Voronoi [14] ou no dos centróides para estimar o melhor caminho proibindo a passagem por triângulos no interior de obstáculos.

Modelar o problema desta forma reduz a probabilidade de ele achar a trajetória ótima, mas reduz o tempo de processamento pois não são levados em consideração obstáculos pelo otimizador, diminuindo tanto o número de variáveis binárias quanto de restrições.

Sejam $P1$ e $P2$ os pontos do triângulo do lado por onde o robô quer sair. Seja $P3$ o terceiro ponto. Seja $< , >$ o produto escalar de dois vetores e $\times$ o produto vetorial. A equação 22 impõe que no n-ésimo passo o robô está do outro lado da reta que caracteriza o lado em comum entre o triângulo atual e o próximo. As equações 23 e 24 impõem que o robô deve estar entre outros dois lados do triângulo em todos os passos.



Da esquerda pra direita, de cima pra baixo: triangulação de Delaunay com as quinas do campo e com pontos aleatórios; algoritmo de otimização para sair de um triângulo pelo lado esquerdo; grafo de Voronoi, os pontos representam os obstáculos, as arestas, mediatrizes de dois obstáculos; em preto, a triangulação de Delaunay, em azul, o grafo dos centróides.

$$< (S(i) - P1) \times (P2 - P1), (P3 - P1) \times (P2 - P1) > \leq \epsilon + MC(i,b) \quad (22)$$

$$< (S(i) - P1) \times (P2 - P1), (P3 - P1) \times (P2 - P1) > \geq -\epsilon \quad (23)$$

$$< (S(i) - P2) \times (P3 - P2), (P1 - P2) \times (P3 - P2) > \geq -\epsilon \quad (24)$$

4 Resultados Obtidos

No início do trabalho, foi dedicado um mês para estudo de controle preditivo pelo livro [15], sendo adquirida uma base de conhecimento para nortear como deve ser a resposta do planejador de trajetórias. Não houve, a princípio, dificuldades na escolha dos métodos a serem utilizados. Os três algoritmos apresentados neste projeto foram escolhidos pelos seguintes critérios:

1. Campos potenciais já é largamente utilizado por equipes da competição IEEE Very Small Size. Em 2014, a ITAndroids tentou sua implementação, porém fracassou. O método foi assim escolhido por ser onipresente na competição e simples para a resolução deste problema.

2. RRT possui uma ampla gama de aplicações acadêmicas em robótica, sendo um método famoso. Em contraste com o anterior, ele é mais complexo e pode resolver o problema para casos mais genéricos: navegação por labirintos e por caminhos estreitos.

3. Otimização por MILP é um algoritmo que, em contraste com os anteriores, é menos empregado em estudos de robótica. Este método pode levar a resultados melhores que os anteriores, além de ser largamente explorado em diversos trabalhos de pesquisa recentes.

Nos meses seguintes, até outubro, com o material passado pelo orientador, nas referências [16] e [17], foi estudado o método de otimização por MILP para navegação completa pelo mapa. A implementação foi feita em MATLAB, porém seu tempo de execução

se mostrou muito longo e o método foi temporariamente descartado. Era necessária uma reformulação do algoritmo para simplificá-lo e torná-lo rápido o bastante.

Na primeira semana de outubro, foi implementado o método de campos potenciais em MATLAB e logo em seguida em C++. A fonte para os estudos foi o artigo [18]. Foi realizado o teste para o ajuste de parâmetros e para a adaptação de casos especiais.

Nas duas semanas seguintes, prosseguiu-se para o estudo pelas fontes [10], [19] e [20] e implementação do método RRT. As implementações em MATLAB e em C++ ficariam prontas ao final de outubro, sendo realizados os testes em cenário simplificado de forma que se pudesse ajustar as constantes e avaliar se o algoritmo seria aplicável ao robô real.

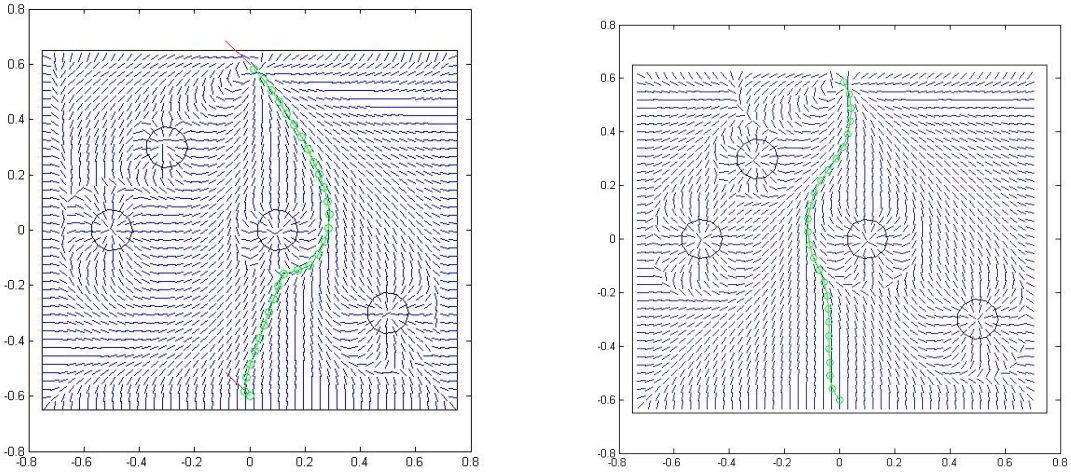
Nessa época, aproximava-se a competição da CBR 2015, sendo interrompida a série de testes do RRT. Foi escolhido o método de campos potenciais para ser utilizado pela ITAndroids. Na última semana de outubro e na primeira de novembro, ele foi testado no robô real. O ITA ganhou quarto lugar na categoria. Apesar do sucesso, foram encontrados vários erros durante os testes e algumas constantes tiveram de ser alteradas para a aplicação no evento. Após a competição, os testes do RRT em cenário simplificado foram concluídos ao longo do mês de novembro.

Durante o período de recesso (dezembro de 2015 a fevereiro de 2016), o método de otimização MILP foi reformulado com a divisão do campo na triangulação de Delaunay. Foi implementado em MATLAB, e nos dois meses seguintes (março e abril de 2016), foram realizados diversos testes em caso simplificado com o algoritmo afim de fazê-lo cumprir com os requisitos do problema. No entanto, os resultados não foram satisfatórios e, desta vez, o método foi definitivamente descartado.

No restante do tempo até a redação do relatório no mês de julho, foram realizadas simulações em cenários realistas para os dois primeiros métodos, sendo medidos os tempos e averiguando se os resultados se enquadram nos requisitos. Mais otimizações foram feitas para o RRT.

A seguir serão apresentados os resultados técnicos dos métodos de campos potenciais e de RRT para o seguinte cenário, com as simulações executadas em um computador com processador Intel Core i7 com *clock* de $2{,}4\,GHz$:

- Posição e direção iniciais, respectivamente: $(0; -0{,}6)$ e $\frac{1}{\sqrt{2}}(-1; 1)$
- Posição e direção finais, respectivamente: $(0; 0{,}6)$ e $\frac{1}{\sqrt{2}}(-1; 1)$



(a) Campos potenciais em MATLAB, trajetória não suavizada.

(b) Campos potenciais em C++, trajetória suavizada.

Figura 4: Resultados com campos potenciais.

- Obstáculos na posições: $(-0{,}3; 0{,}3)$, $(-0{,}5; 0)$, $(0{,}1; 0)$ e $(0{,}5; -0{,}3)$

4.1 Campos potenciais

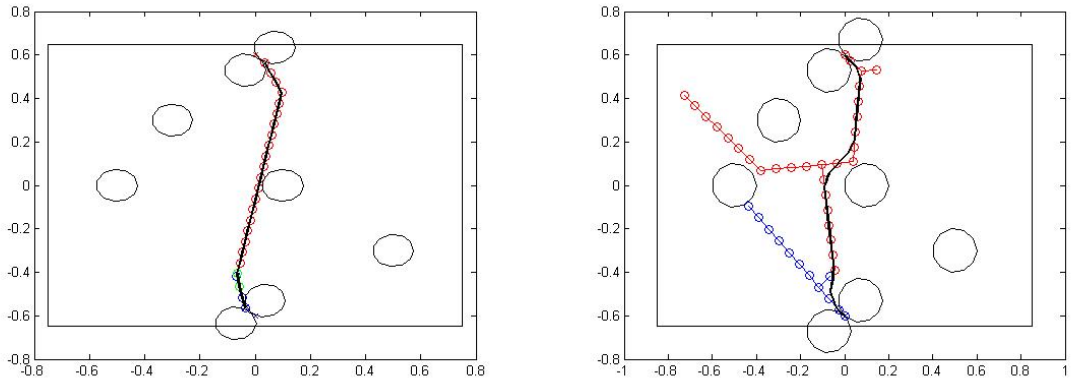
O algoritmo de campos potenciais foi testado primeiramente na plataforma MATLAB (Fig. 4(a)) para a configuração das linhas de campo. Em seguida houve o teste em C++ (Fig. 4(b)). No primeiro, não foi utilizado o artifício de fazer o campo repulsivo do alvo decair para zero a partir de um limite. Foi observado que não há necessidade de se ter e efeito do dipolo elétrico em todo o espaço, pois o código em C++ gerou, neste teste e em outros, uma trajetória mais suave para o robô.

O tempo de execução foi excelente, sendo de apenas 0,0043 segundos em MATLAB. Em C++ a medição foi de 0,00098 segundos. Isso dá uma boa margem para o requisito de tempo de 2 ms para o planejador de trajetórias, o que o torna um candidato qualificado no quesito tempo.

4.2 RRT

O método foi testado em MATLAB (Fig. 5(a)) e em C++ (Fig. 5(b)). No primeiro não foi implementada a suavização da trajetória. No segundo, sim.

Sobre usar ou não usar a Quadtree, o parâmetro h usado foi de 7 cm, fazendo com que



(a) RRT em MATLAB, trajetória não suavizada. (b) RRT em C++, trajetória suavizada.

Figura 5: Resultados com RRT.

a média n de nós das árvores seja da ordem de 50. A diferença entre o tempo $O(\log n)$ e $O(n)$ para a inserção, considerando que com o método simples não há alocação de memória e tampouco constantes altas, é pequena. A inserção $O(\log n)$ com constantes é mais lenta que $O(1)$. O tempo com a Quadtree é aproximadamente o dobro em todos os testes realizados, sendo a estrutura, portanto, descartada.

Nesse caso, devido à característica aleatória do algoritmo, para evitar a variabilidade alta entre duas execuções consecutivas, buscando realmente se aproximar do ótimo, a trajetória é tomada como a melhor (menor comprimento) de um certo número de iterações do mesmo algoritmo. Para fazer a escolha do número de iterações para o algoritmo convergir para a solução ótima, da probabilidade P e da G, foram realizados testes de forma a se obter as três tabelas de tempo, em milisegundos, nas tabelas da seção 1, respectivamente. Cada tempo é a média de 5000 testes. Deve-se ter $P + G \leq 0{,}9$, pois deve haver pelo menos 0,1 de probabilidade de se escolher um ponto aleatório no mapa.

Tabela 1: Tempos de computação com 5 iterações - média de 5000 ensaios.

$G \backslash P$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.0	0,723	0,610	0,586	0,578	0,601	0,621	0,672	0,760
0.1	0,804	0,621	0,599	0,608	0,642	0,683	0,761	0,939
0.2	0,878	0,641	0,633	0,647	0,697	0,786	0,986	1,488
0.3	0,846	0,676	0,681	0,709	0,817	1,006	1,565	-
0.4	1,046	0,758	0,788	0,854	1,030	1,697	-	-
0.5	1,059	0,809	0,866	1,041	1,624	-	-	-

Tabela 2: Tempos de computação com 10 iterações - média de 5000 ensaios.

$G \backslash P$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.0	1,605	1,128	1,098	1,106	1,144	1,207	1,329	1,526
0.1	1,530	1,179	1,155	1,174	1,243	1,380	1,575	1,969
0.2	1,740	1,206	1,235	1,283	1,389	1,553	1,935	3,049
0.3	1,784	1,278	1,300	1,443	1,626	1,977	3,124	-
0.4	2,612	1,404	1,455	1,644	2,042	3,228	-	-
0.5	2,194	1,555	1,704	2,097	3,228	-	-	-

Tabela 3: Tempos de computação com 25 iterações - média de 5000 ensaios.

$G \backslash P$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.0	3,346	2,791	2,657	2,780	2,889	2,984	3,264	3,705
0.1	3,858	2,795	2,858	2,881	3,025	3,273	3,755	4,612
0.2	4,100	2,887	2,913	3,077	3,332	3,823	4,749	7,698
0.3	4,735	3,237	3,316	3,581	4,085	5,062	8,006	-
0.4	4,788	3,552	3,730	4,210	5,231	8,177	-	-
0.5	5,405	3,985	4,400	5,395	8,326	-	-	-

A partir dos resultados das tabelas acima, foi decidido usar-se o valor de 0,1 para G e 0,2 para P. Em média, o tempo com esses valores é 30% menor em relação ao algoritmo sem essa heurística (P = G = 0).

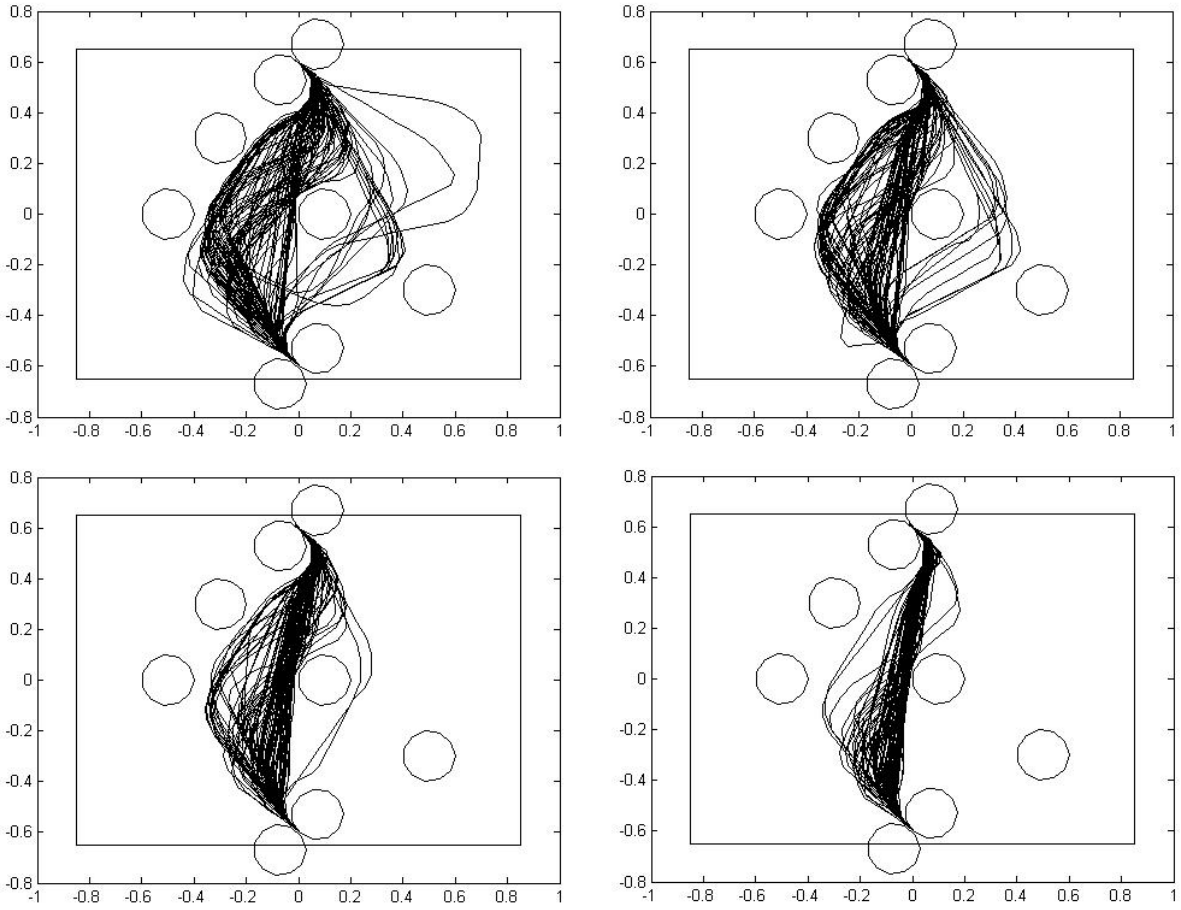
Na figura acima, pode-se observar a variança nos caminhos. De esquerda para direita, cima pra baixo: 5, 10, 25 e 100 iterações. Um número menor de iterações é mais rápido, porém existe uma probabilidade maior de se encontrar uma trajetória não ótima e distante da anterior. Diante disso, foi escolhido o número máximo de iterações que respeita o limite de tempo de 2 ms com alguma folga: 15 iterações, que fazem uma média de 1,7 ms.

4.3 Otimização com MILP

4.3.1 Navegação sobre todo o campo

O método de otimização aplicado sobre todo o campo se mostrou muito lento. O GLPK testa todas as possibilidades de combinações para as variáveis binárias, o que representa

uma complexidade de $O(2^n)$. Nessa forma, para cada obstáculo e para cada passo, existem duas variáveis binárias. Por exemplo, para um vetor b com 5 variáveis, existem 32 passos possíveis. Para 5 obstáculos (situação real), existem 10 variáveis binárias. O total de variáveis é de $5 + 32 * 10 = 325$, o que torna o algoritmo inaplicável para a complexidade dada. O método foi descartado na fase de testes em cenário simples.



4.3.2 Navegação por uma triangulação

O segundo método, por outro lado, permite o trabalho do GLPK sem levar os obstáculos em consideração, pois ao respeitar o limite dos triângulos, respeita-se ao mesmo tempo a restrição dos obstáculos. Tendo apenas 5 variáveis binárias, a complexidade $O(2^n)$ é admissível. No entando, ao se dividir o problema em triângulos e montar um algoritmo que resolve a transição por um triângulo, o algoritmo passou a não respeitar possíveis restrições do próximo triângulo enquanto soluciona o atual.

Por exemplo, foi usado para testes um gerador de situações aleatórias, que posiciona 5 obstáculos (três adversários e dois aliados) no campo. O otimizador leva o robô ao próximo

triângulo com sucesso, entrentanto ele chega com uma posição e direção desfavoráveis para a solução do próximo. Em mais da metade dos testes aleatórios, o otimizador GLPK acusou erro de equação sem solução para algum trecho.

Tentar expandir as restrições do triângulo seguinte para o atual acabou por exigir uma quantidade maior de equações de restrição, o que desacelerou o algoritmo e acabou por não conseguir resolver inteiramente o problema. Diante dessa situação, este método também foi descartado ainda na fase de testes em cenário simples.

5 Conclusões

O projeto de Iniciação Científica apresentou bons resultados, especialmete quanto à vitória da equipe do ITA até as semifinais da CBR 2015, rendendo o quarto lugar à equipe.

Do ponto de vista técnico, o uso do algoritmo já consagrado de planejamento por campos potenciais se mostrou factível em tempo real e adequado para o desvio de obstáculos. Contudo, classes mais gerais de problemas podem ser resolvidos com RRT e MILP, visto que estes podem englobar modelos do robô no planejamento, levando em conta suas limitações ao calcular a trajetória. Por esse motivo, visto as dificuldades com o método de otimização, o RRT foi escolhido para prosseguir com os testes e com a otimização de parâmetros a fim de se montar uma classe de planejamento de trajetórias funcional para a equipe da ITAndroids.

6 Agradecimentos

- Ao CNPq, pelo apoio financeiro e motivacional.
- 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.
- Ao professor Marcos Máximo da Divisão da Ciência da Computação do ITA, pelo apoio nos estudos e no desenvolvimento do projeto.

7 Bibliografia

Referências

[1] J. A. G. Orozco e E. S. de Cómputo, "Máquinas de estados finitos",

[2] S. Russel e P. Norvig, "Artificial intelligence: A modern approach", em, 3ª ed. Prentice Hall, 2009, vol. 3, pp. 697–707.

[3] M. Colledanchise, "Behavior trees in robotics", tese de doutorado, KTH Royal Institute of Technology, fev. de 2016.

[4] http://www.cbrobotica. org/wp-content/uploads/2014/03/VerySmall2008_en.pdf.

[5] I. F. Okuyama, "Trajectory planning considering acceleration limits for an autonomous soccer player", Trabalho de Graduação, Instituto Tecnológico de Aeronáutica, 2017.

[6] http://www.cbrobotica. org/wp-content/uploads/2014/03/VerySmall2008_en.pdf.

[7] R. H. Abiyev, Şenol Bektas, N. Akkaya e E. Aytac, *Neuislanders 2017 team description paper*, Robocup, 2017.

[8] D. B. West et al., *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2, cap. 2.

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

[10] S. M. LaValle, "Motion planning: The essentials", *IEEE Robotics and Automation Society Magazine*, vol. 18, nº 1, pp. 79 –89, 2011.

[11] J. Bruce e M. Veloso, "Real-time randomized path planning for robot navigation", em *Intelligent Robots and Systems, IEEE/RSJ International Conference*, 2002, pp. 2383 –2388.

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

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs.", *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

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

[15] J. M. Maciejowski, "Predictive control with constraints", em. Prentice Hall, 2002, cap. 1-3, pp. 1–107.

[16] J. Bellingham, A. Richards e J. P. How, "Receding horizon control of autonomous aerial vehicles", em *Proceedings of the American Control Conference, Anchorage, AK, USA*, 2002, pp. 3741 –3746.

[17] R. J. M. Afonso, "Fault-tolerant predictive control with trajectory planning in the presence of obstacles", Tese de Doutorado, Instituto Tecnológico de Aeronáutica, 2015.

[18] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", *The International Journal of Robotics Research*, vol. 5, nº 1, pp. 90 –98, 1986.

[19] R. J. M. Afonso, "Development of an user interface and algorithm for three-dimensional path planning for an umanned aerial vehicle", Trabalho de Graduação, Instituto Tecnológico de Aeronáutica, 2009.

[20] J. Kuffner, *Howie Choset with slides from James Kuffner*.