Trabalho Prático 1 - Implementação dos métodos de navegação: Função de Potencial, *Wavefront* e A*

Rafael Fernandes Gonçalves da Silva
Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
rafaelfgs@ufmg.br

I. Motivação

Este trabalho apresenta a aplicação de três métodos distintos para a navegação de robôs móveis: Função de Potencial, *Wavefront* e A*.

Primeiramente, para a função de potencial, foi requisitada a implementação de uma estratégia com funções atrativas e repulsivas de forma a controlar um robô entre dois pontos quaisquer em um espaço definido. Para este fim, foi determinada a utilização de um robô com acionamento diferencial e de um sensor laser, de forma a identificar os obstáculos no mapa, através do simulador *StageROS*.

A segunda aplicação envolveu a implementação do *wavefront*. Nesse método, foi determinada a utilização de um ambiente discretizado, utilizando novamente o simulador *StageROS* e um robô com acionamento diferencial.

Por fim, de forma similar, para a implementação do método A* também foi requisitada a utilização de um robô diferencial no *StageROS* com um ambiente discretizado.

II. Metodologia

O simulador *StageROS* foi utilizado para a resolução dos problemas propostos e as implementações de cada parte do problema estão descritas nas subseções seguintes.

A. Função de Pontecial

O problema consiste em navegar entre dois pontos em um ambiente com obstáculos. Para isso, deve-se utilizar um sensor *laser* e cálculos dos gradientes de forma a evitar colisão com obstáculos. O pseudocódigo a seguir demonstra a sequência de passos utilizada para a resolução do problema.

```
INITIALISE current node, subscribers and publishers;
WHILE ( forever )
  IF ( goal is not reached )
    EVALUATE attractive potential;
    EVALUATE repulsive potential;
    SET control signal as the sum of potentials;
  ELSE
    SET control signal as null;
  END IF
  EVALUATE linear and angular velocities;
  PUBLISH velocities;
END WHILE
```

As equações a seguir demostram respectivamente o cálculo dos potenciais atrativo e repulsivo para cada direção das coordenadas x e y.

$$\begin{cases} U_{att} = \zeta \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} \right) \\ \begin{bmatrix} U_{x,att} \\ U_{y,att} \end{bmatrix} = \tanh\left(U_{att}\right) \end{cases} \tag{1}$$

$$\begin{cases} D = -\begin{bmatrix} \cos(\theta + \theta_{min}) \\ \sin(\theta + \theta_{min}) \end{bmatrix} \\ U_{rep} = \eta \left(\frac{1}{Q} - \frac{1}{d_{min}} \right) \frac{1}{d_{min}^2} D \\ \begin{bmatrix} U_{x,rep} \\ U_{y,rep} \end{bmatrix} = \tanh\left(U_{rep}\right) \end{cases} \tag{2}$$

Onde $\begin{bmatrix} x & y \end{bmatrix}^T$ é a posição atual do robô, $\begin{bmatrix} x_{goal} & y_{goal} \end{bmatrix}^T$ é a posição do alvo, ζ e η são os respectivos ganhos dos efeitos atrativo e repulsivo, Q é a distância mínima para ignorar o obstáculo e d_{min} e θ_{min} são a distância e o ângulo do obstáculo mais próximo.

O sinal de controle final é dado pela soma dos potenciais atrativo e repulsivo de cada coordenada x e y:

$$\begin{bmatrix} U_x \\ U_y \end{bmatrix} = \begin{bmatrix} U_{x,att} + U_{x,rep} \\ U_{y,att} + U_{y,rep} \end{bmatrix} \tag{3}$$

Por fim, são determinados os valores das velocidades linear e angular, obtidos através do *feedback linearization* conforme a equação a seguir:

$$\begin{bmatrix} V_x \\ W_z \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \frac{-\sin(\theta)}{d} & \frac{\cos(\theta)}{d} \end{bmatrix} \begin{bmatrix} U_x \\ U_y \end{bmatrix} \tag{4}$$

onde θ é a orientação atual do robô e d é o deslocamento do ponto de controle, informado no início do programa.

B. Wave-front

Este problema consiste em aplicar o método *wavefront* em um ambiente discretizado e navegar o robô entre dois pontos através da minimização do valor da célula atual. O pseudocódigo a seguir demonstra a sequência de passos utilizada para a resolução do problema.

```
INITIALISE current node, subscribers and publishers;
SET a discretized grid using a map from a image file;
WHILE ( forever )
  IF ( goal changed )
    APPLY wavefront on grid;
  END IF
  SET next point for robot;
  IF ( goal is reached )
    SET next point as current point;
  END IF
  EVALUATE control signal;
  EVALUATE linear and angular velocities;
  PUBLISH velocities;
END WHILE
```

A discretização utilizada cria uma matriz com valores 0 e inf, formando um *grid* que determina os pontos de espaço livre e com obstáculos. Para isso, é realizada uma leitura de um arquivo de imagem, salvo em uma variável na forma matricial. Utilizando o tamanho da imagem em pixels, o tamanho do mapa, da resolução desejada (sendo estes dois últimos passados como parâmetros pelo usuário), é realizada uma conversão da imagem para um *grid* de tamanho igual à resolução desejada. Um exemplo dessa conversão está mostrada na Figura 1 com os pontos do grid em vermelho representando os obstáculos e em branco o espaço livre.

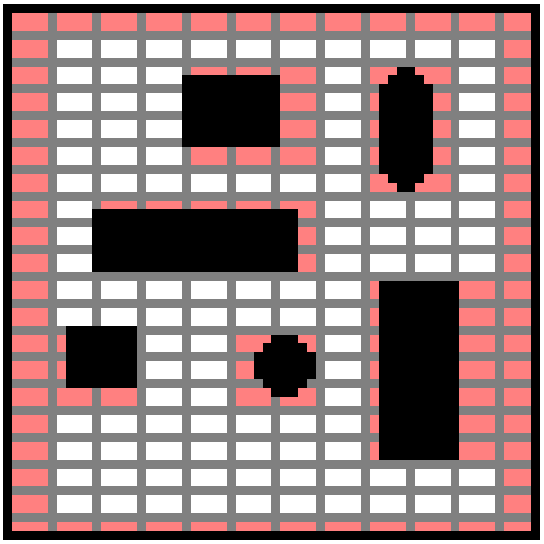

Figura 1. Exemplo de um *grid* de resolução 12x20 aplicado no mapa com obstáculos.

Em seguida, o programa entra em um *loop* infinito, permitindo o usuário movimentar o robô e o ponto alvo como desejar. Primeiramente no *loop*, caso houver alteração do ponto alvo (ou for a primeira iteração), é aplicado o método *wavefront* de forma a definir valores para os pontos do espaço livre de forma a determinar uma trajetória para cada ponto. A aplicação do método, de acordo com o mapa da Figura 1, está representada na Figura 2.

```
[[ inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf]
 [ inf  28.  27.  26.  25.  24.  23.  22.  23.  24.  25.  inf]
 [ inf  27.  26.  25.  inf  inf  inf  21.  inf  inf  24.  inf]
 [ inf  26.  25.  24.  inf  inf  inf  20.  inf  inf  23.  inf]
 [ inf  25.  24.  23.  inf  inf  inf  19.  inf  inf  22.  inf]
 [ inf  24.  23.  22.  inf  inf  inf  18.  inf  inf  21.  inf]
 [ inf  23.  22.  21.  20.  19.  18.  17.  inf  inf  20.  inf]
 [ inf  22.  inf  inf  inf  inf  inf  16.  17.  18.  19.  inf]
 [ inf  21.  inf  inf  inf  inf  inf  15.  16.  17.  18.  inf]
 [ inf  20.  inf  inf  inf  inf  inf  14.  15.  16.  17.  inf]
 [ inf  19.  18.  17.  16.  15.  14.  13.  inf  inf  inf  inf]
 [ inf  18.  17.  16.  15.  14.  13.  12.  inf  inf  inf  inf]
 [ inf  inf  inf  15.  14.  inf  inf  11.  inf  inf  inf  inf]
 [ inf  inf  inf  14.  13.  inf  inf  10.  inf  inf  inf  inf]
 [ inf  inf  inf  13.  12.  inf  inf   9.  inf  inf  inf  inf]
 [ inf  14.  13.  12.  11.  10.   9.   8.  inf  inf  inf  inf]
 [ inf  13.  12.  11.  10.   9.   8.   7.  inf  inf  inf  inf]
 [ inf  12.  11.  10.   9.   8.   7.   6.   5.   4.   3.  inf]
 [ inf  11.  10.   9.   8.   7.   6.   5.   4.   3.   2.  inf]
 [ inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf]]
```

Figura 2. Resposta do método *wavefront* para exemplo da Figura 1.

Através do grid com os valores de cada ponto, é possível então realizar um controle ponto a ponto da trajetória do robô, sendo que o próximo ponto desejado para o robô é o valor do ponto atual subtraído de um. A partir daí, são determinados os sinais de controle (mostrado na Equação (5)) e as velocidades linear e angular (mostrado na Equação (4)).

$$\begin{bmatrix} U_x \\ U_y \end{bmatrix} = \begin{bmatrix} k.(x_d - x) + \dot{x_d} \\ k.(y_d - y) + \dot{y_d} \end{bmatrix} \tag{5}$$

C. A*

Para a aplicação do método A* foi novamente necessário discretizar o ambiente de forma a navegar com o robô entre dois pontos através de uma heurística utilizando os valores das células do *grid*. Para resolver o problema, foi utilizada a sequência de passos de acordo com o pseudocódigo a seguir.

```
INITIALISE current node, subscribers and publishers;
SET a discretized grid using a map from a image file;
WHILE ( forever )
  IF ( goal changed )
    APPLY A* on grid;
  END IF
  SET next point for robot;
  IF ( goal is reached )
    SET next point as current point;
  END IF
  EVALUATE control signal;
  EVALUATE linear and angular velocities;
  PUBLISH velocities;
END WHILE
```

A discretização do ambiente foi realizada de forma similar ao método *wavefront*, com valores 0 para o espaço livre e inf para os obstáculos. Novamente o programa entre em um *loop* infinito, onde é primeiramente é implementado o método do A*, sendo determinado os pontos do caminho do robô ao alvo. Para isso, é foram utilizadas duas matrizes G e H em forma de *grid*, mostradas respectivamente nas Figuras 3 e 4.

A matriz H representa a heurística de distância (*Manhattan*), onde não são considerados os obstáculos. Devido sua simplicidade, essa matriz já é inicialmente determinada por completo.

```
[[ inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf]
 [ inf   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  inf]
 [ inf   0.   0.   0.  inf  inf  inf   0.  inf  inf   0.  inf]
 [ inf   0.   2.   3.  inf  inf  inf   0.  inf  inf   0.  inf]
 [ inf   2.   1.   2.  inf  inf  inf   0.  inf  inf   0.  inf]
 [ inf   3.   2.   3.  inf  inf  inf   9.  inf  inf   0.  inf]
 [ inf   4.   3.   4.   5.   6.   7.   8.  inf  inf  13.  inf]
 [ inf   0.  inf  inf  inf  inf  inf   9.  10.  11.  12.  inf]
 [ inf   0.  inf  inf  inf  inf  inf  10.  11.  12.  13.  inf]
 [ inf   0.  inf  inf  inf  inf  inf  11.  12.  13.  14.  inf]
 [ inf   0.   0.   0.   0.   0.  13.  12.  inf  inf  inf  inf]
 [ inf   0.   0.   0.   0.   0.  14.  13.  inf  inf  inf  inf]
 [ inf  inf  inf   0.   0.  inf  inf  14.  inf  inf  inf  inf]
 [ inf  inf  inf   0.   0.  inf  inf  15.  inf  inf  inf  inf]
 [ inf  inf  inf   0.   0.  inf  inf  16.  inf  inf  inf  inf]
 [ inf   0.   0.   0.   0.   0.  18.  17.  inf  inf  inf  inf]
 [ inf   0.   0.   0.   0.   0.  19.  18.  inf  inf  inf  inf]
 [ inf   0.   0.   0.   0.   0.  20.  19.  20.  21.  22.  inf]
 [ inf   0.   0.   0.   0.   0.  21.  20.  21.  22.  23.  inf]
 [ inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf]]
```

Figura 3. Resposta da busca G no método A* para exemplo da Figura 1.

```
[[ inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf]
 [ inf  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  inf]
 [ inf  25.  24.  23.  inf  inf  inf  19.  inf  inf  16.  inf]
 [ inf  24.  23.  22.  inf  inf  inf  18.  inf  inf  15.  inf]
 [ inf  23.  22.  21.  inf  inf  inf  17.  inf  inf  14.  inf]
 [ inf  22.  21.  20.  inf  inf  inf  16.  inf  inf  13.  inf]
 [ inf  21.  20.  19.  18.  17.  16.  15.  inf  inf  12.  inf]
 [ inf  20.  inf  inf  inf  inf  inf  14.  13.  12.  11.  inf]
 [ inf  19.  inf  inf  inf  inf  inf  13.  12.  11.  10.  inf]
 [ inf  18.  inf  inf  inf  inf  inf  12.  11.  10.   9.  inf]
 [ inf  17.  16.  15.  14.  13.  12.  11.  inf  inf  inf  inf]
 [ inf  16.  15.  14.  13.  12.  11.  10.  inf  inf  inf  inf]
 [ inf  inf  inf  13.  12.  inf  inf   9.  inf  inf  inf  inf]
 [ inf  inf  inf  12.  11.  inf  inf   8.  inf  inf  inf  inf]
 [ inf  inf  inf  11.  10.  inf  inf   7.  inf  inf  inf  inf]
 [ inf  12.  11.  10.   9.   8.   7.   6.  inf  inf  inf  inf]
 [ inf  11.  10.   9.   8.   7.   6.   5.  inf  inf  inf  inf]
 [ inf  10.   9.   8.   7.   6.   5.   4.   3.   2.   1.  inf]
 [ inf   9.   8.   7.   6.   5.   4.   3.   2.   1.   0.  inf]
 [ inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf  inf]]
```

Figura 4. Resposta da heurística H no método A* para exemplo da Figura 1.

De forma a reduzir o custo computacional, a matriz G é determinada passo a passo da seguinte forma:

- determina-se o ponto inicial $[i, j]$ como a posição atual do robô;
- encontra-se os vizinhos (4 ou 8) que ainda não possuem valor de G;
- determina-se os valores de G para os vizinhos;
- define-se os valores de F para os vizinhos, onde $F = G + H$;
- encontra-se as posições dos menores valores de F; e
- percorre-se cada uma das posições, atribuindo-as como o ponto [i,j] atual.

Nesse ponto, é também criada uma matriz que se comporta como um grafo das possíveis trajetórias. Essa matriz tem possui duas camadas onde, ao atribuir o valor de G aos vizinhos, são também atribuídos os valores de i e j da posição atual, de forma a possuir uma informação do caminho do robô ao alvo.

Em seguida, é determinado o próximo ponto do robô através da matriz descrita anteriormente. A partir do alvo, percorre-se os índices correspondentes aos vizinhos até encontrar o robô. O ponto anterior ao encontro do robô é determinado como sendo o próximo ponto desejado.

Com os valores do próximo ponto, assim como no método anterior, é realizado um controle ponto a ponto da trajetória do robô. A partir daí, são determinados os sinais de controle de acordo com a Equação (5) e as velocidades linear e angular de acordo com a Equação (4).

III. Resultados e Discussão

As simulações do código implementado foram realizadas no *StageROS*. Os resultados obtidos com os três métodos distintos estão mostrados a seguir. Para cada um, foram utilizados três diferentes mapas, com diferentes resoluções de discretização para o *wavefront* e o A*. Por padrão os valores das constantes foram fixadas da seguinte forma: $d = 0.1$, $k = 1.0$, $size = [40, 40]$ (ou $[16, 12]$ para o big), $resolution = [20, 20]$ (ou $[16, 12]$ para o big) e $tolerance = 0.01.size^2/resolution$.

A. Função de Potencial

1) Mapa com obstáculos: No mapa com obstáculos, com um ponto inicial em $[-12, 11]^T$ e um ponto alvo em $[15, -17]^T$, a função de potencial obteve o resultado demonstrado na Figura 5.

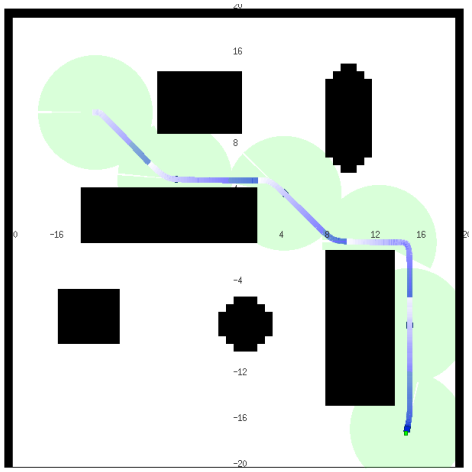

Figura 5. Resultado do método função de potencial aplicado no mapa com obstáculos.

Pode-se observar, a partir da Figura 5, que o robô alcançou o alvo desviando-se corretamente dos obstáculos no caminho.

2) Outros mapas: Utilizando agora o mapa com maior resolução em pixels e um mapa de uma sala, e partindo respectivamente do ponto $[-6.5, 4.5]^T$ e $[-0.5, 0.5]^T$ com alvos em $[4.5, 3.5]^T$ e $[12.5, 0.5]^T$, a função potencial alcançou os resultados apresentados nas Figuras 6 e 7.

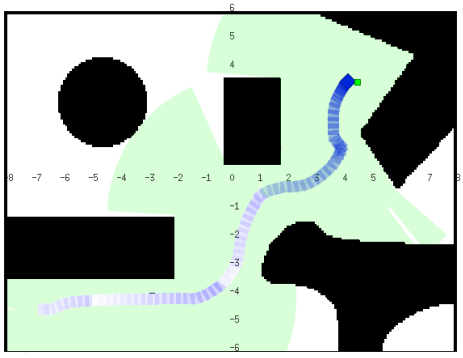

Figura 6. Resultado do método função de potencial aplicado no mapa com maior resolução.

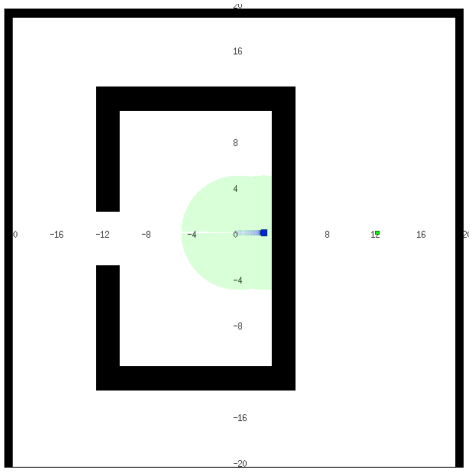

Figura 7. Resultado do método função de potencial aplicado no mapa da sala.

Na Figura 6, foi possível notar certa instabilidade, visto que o algoritmo implementado somente leva em consideração o obstáculo mais próximo e, neste mapa, há uma maior quantidade de obstáculos, o que pode atrapalhar o movimento do robô.

Na Figura 7, é nítida a presença de um mínimo local, visto que o alvo se encontra do outro lado da parede. Como o método implementado não é completo em resolução, não há garantia que o robô escape de mínimos locais, ficando assim parado em frente ao obstáculo.

B. *Wave-front*

1) Mapa com obstáculos: Iniciando no ponto $[-12, 11]^T$ e com o alvo em $[15, -17]^T$ no mapa de obstáculos, o método *wavefront* obteve o resultado mostrado na Figura 8.

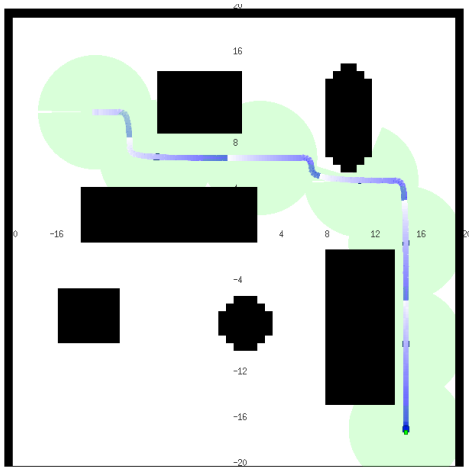

Figura 8. Resultado do método aplicado no mapa com obstáculos.

É possível observar o deslocamento em vizinhança 4 na trajetória do robô, conforme foi implementado, chegando corretamente ao alvo, sem problemas de colisão com os obstáculos.

2) Outros mapas: Nos mapas com maior resolução e da sala, os resultados obtidos estão apresentados nas Figuras 9 e 10, respectivamente. Para o primeiro o robô iniciou em $[-6.5, 4.5]^T$, com um alvo em $[4.5, 3.5]^T$, e, para o segundo, iniciou em $[-0.5, 0.5]^T$ com um alvo em $[12.5, 0.5]^T$.

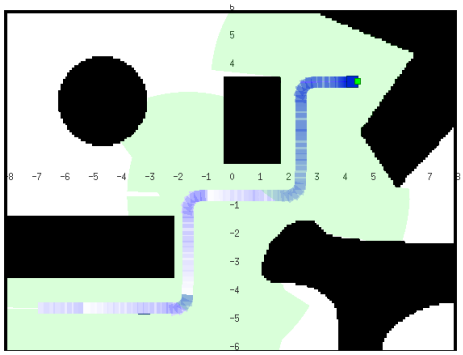

Figura 9. Resultado do método *wavefront* aplicado no mapa com maior resolução.

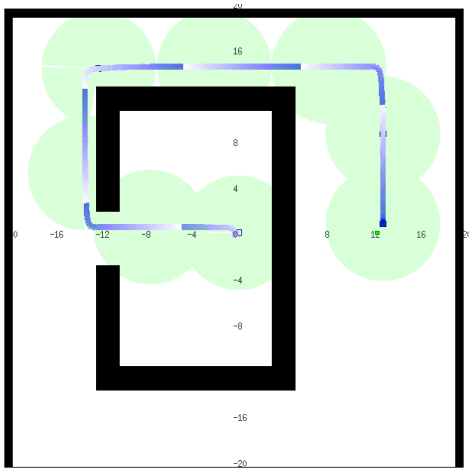

Figura 10. Resultado do método *wavefront* aplicado no mapa da sala.

Nesse método, o robô encontrou o caminho ao alvo, não havendo problemas com mínimos locais. Com esses resultados, também é possível observar a trajetória do robô na horizontal e na vertical, devido a vizinhança 4 utilizada. Na Figura 10, pode-se observar certo erro entre o ponto final do robô e o alvo. Isso se deve ao fato da baixa resolução utilizada, onde, mesmo chegando ao ponto correto do grid, não correspondia ao ponto correto no mapa contínuo.

C. *A**

1) Mapa com obstáculos: De forma similar, partindo do ponto $[-12, 11]^T$ com o alvo em $[15, -17]^T$ no mapa de obstáculos, o método A* obteve o resultado mostrado na Figura 11.

Assim como no *wavefront*, o método também possui um encontro de trajetória por vizinhança 4, obtendo assim uma resposta bastante similar ao método anterior, visto que a discretização do mapa foi exatamente a mesma.

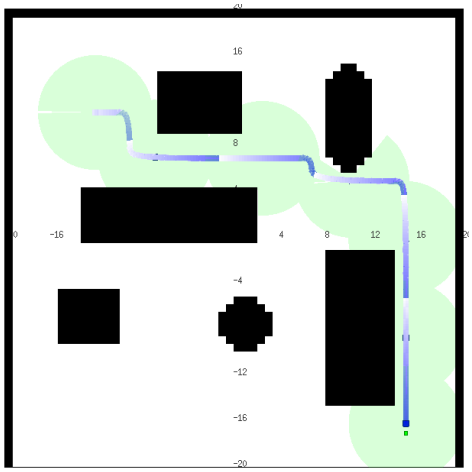

Figura 11. Resultado do método A* aplicado no mapa com obstáculos.

2) Outros mapas: Novamente iniciando do ponto $[-6.5, 4.5]^T$ com um alvo em $[4.5, 3.5]^T$ no mapa com maior resolução, e iniciando do ponto $[-0.5, 0.5]^T$ com um alvo em $[12.5, 0.5]^T$ no mapa da sala, o robô percorreu as trajetórias mostradas respectivamente pelas Figuras 12 e 13.

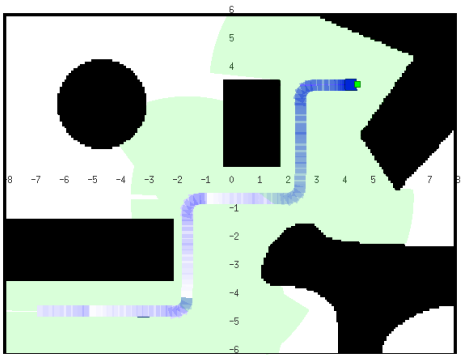

Figura 12. Resultado do método A* aplicado no mapa com maior resolução.

Nesses casos, o robô também não encontrou problemas com mínimos locais, encontrando o alvo em ambos os casos. Os resultados foram novamente semelhantes ao método *wavefront*, havendo problemas similares para o erro final em relação ao alvo devido a discretização de baixa resolução.

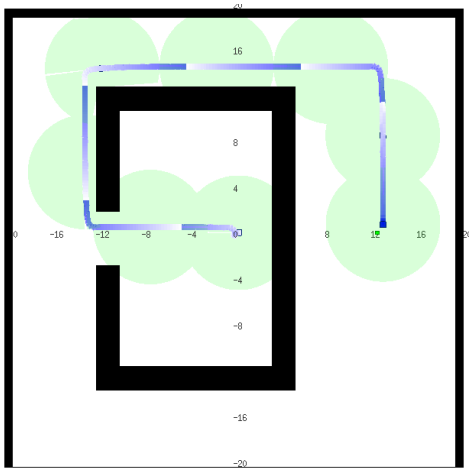

Figura 13. Resultado do método A* aplicado no mapa da sala.

IV. Conclusão

O presente trabalho apresentou implementações para três métodos distintos de navegação: Função de Pontecial, *Wavefront* e A*. Para o primeiro foram utilizadas funções de gradiente partindo do alvo e do obstáculo e, para os demais, foram utilizadas discretizações para métodos de navegação de ponto a ponto através de uma minimização de valores em um *grid* de tamanho especificado pelo usuário.

Os resultados apresentados foram satisfatórios, com problemas em algumas situações específicas. Para o método por função de potencial, a trajetória apresentou certa instabilidade para mapas com muitos obstáculos (conforme mostrado na subseção III-A2), algo que pode ser corrigido alterando a função de gradiente repulsiva para levar em consideração as menores distâncias de todos os obstáculos ao redor.

Para os métodos *wavefront* e A*, o principal problema ocorreu em relação à resolução utilizada na discretização. Esta teve de ser baixa, o que acaba gerando erros relacionados às posições, mas necessária para evitar colisões nos obstáculos, já que o robô possui um tamanho diferente de zero. Nesse caso, um aumento dos obstáculos durante a discretização evitaria colisões, independente da resolução utilizada, o que reduziria o erro de posição encontrado. Esse aumento dos obstáculos também implicaria na possibilidade de utilizar vizinhança 8 para a navegação, o que aprimoraria a trajetória percorrida e consequentemente o tempo gasto.