Universidade Federal de Minas Gerais

Escola de Engenharia

EEE935 - Planejamento de Movimento de Robôs

Trabalho Prático 1

Felipe Bartelt de Assis Pessoa 2016026841
Elnatã Mateus Evangelista Diniz 2021726872

6 de dezembro de 2021

Sumário

1 Introdução

O robô móvel é a representação de aplicações da robótica na movimentação em ambientes. De acordo com Secchi (2008), "Os robôs móveis são dispositivos de transporte automático, ou seja, são plataformas mecânicas dotadas de um sistema de locomoção capazes de navegar através de um determinado ambiente de trabalho, dotados de certo nível de autonomia para sua locomoção, portando cargas". Existem diversas aplicações para os robôs moveis, tais como, inspeção de tubulações, limpeza de tubulações, manutenção de piscinas e tanque de água, aspiradores domésticos, robô enfermeiro, empilhadeiras autônomas, robô militar entre outros, que proporcional a realização mais assertiva de tarefas na indústria e residências. Historicamente é possível descrever a evolução da robótica móvel pelos diversos modelos, alguns como Sojourner é um robô móvel usado durante missão para explorar marte em 1997,Figura 1 ou ainda, o robô Helpmate utilizado para transporte em hospitais, Figura 2.

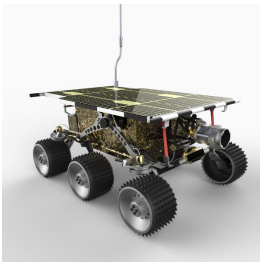


Figura 1: Robô Sojourner. Fonte: Siegwart (2004)

Figura 2: Robô Helpmate. Fonte: Siegwart (2004).

Para determinar as tarefas possíveis de execução de um robô, torna-se preciso definir o nível de autonomia, ou seja, "o grau de autonomia depende em grande medida da capacidade do robô para abstrair o entorno e converter a informação obtida em ordens, de tal modo que, aplicadas sobre os atuadores do sistema de locomoção, garanta a realização eficaz de sua tarefa" Secchi (2008), sendo assim, a recepção e raciocínio são parâmetros fundamentais para determinar o planejamento de movimentação do robô no ambiente.

As estratégias de planejamento e controle de movimento de robôs é fundamental para determinar a realização adequada das tarefas no ambiente. Uma tomada de decisão não assertiva pode proporcionar uma execução inadequada de trajetória do robô, resultado em desempenho de alto custo para o processo, por isso, é necessário o estudo e implementação de algoritmos que maximize o desempenho dos robôs, minimizando o custo operacional.

2 Simulações

O robô utilizado em todas as simulações apresenta 1440 sensores laser com alcance de 2 m, distribuídos igualmente em 360°. Utilizou-se o ROS-Noetic para as implementações em Python, assim como o simulador StageRos. Para a simulação nesse ambiente 2D, definiu-se diversos arquivo `.world`, onde são definidas as características do ambiente, tais como, obstáculos, robôs e outros objetos.

O robô utilizado é não-holonômico e, assim, utilizou-se um controlador com *feedback linearization*, que recebe o ponto a ser seguido e retorna o *twist* necessário para que esse ponto seja alcançado. O parâmetro, comumente denominado apenas por d no *feedback linearization* foi tomado como 0.1, uma vez que esse valor corresponde a 1/5 do comprimento do robô, que é quadrado.

2.1 Tangent Bug

Para compreensão do algoritmo Tangent Bug é preciso analisar o funcionamento dos algoritmos Bug 1 e Bug 2. De acordo com Choset et al. [1], "O Bug 1 e Bug 2 estão entre os primeiros e mais simples planejadores baseados em sensores para detectar obstáculos". O Bug 1,Figura 3, utiliza como princípio para o funcionamento a abordagem de movimentação direta em direção ao alvo, tendo em vista um ambiente livre de obstáculos, caso contrário ele circula o obstáculo buscando novamente o alvo como objetivo. A movimentação entorno do obstáculo não favorece

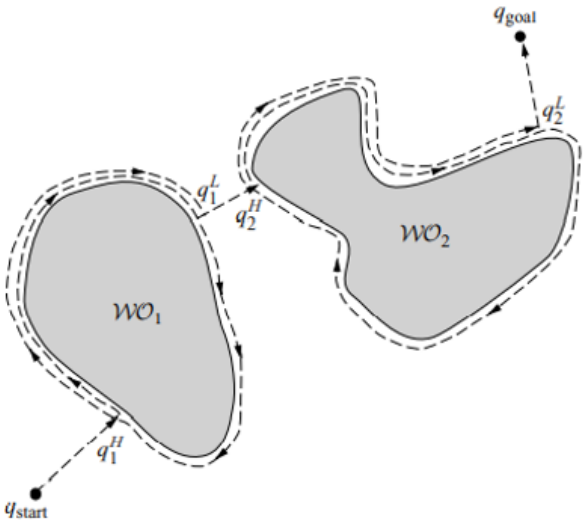


Figura 3: Algoritmo Bug1[1]

o robô ao melhor desempenho de deslocamento do ambiente, sendo assim, uma evolução dessa estratégia é aplicada no algoritmo Bug 2. No Bug 2, Figura 4, o princípio de planejamento da movimentação é o mesmo do Bug 1, ou seja, o robô movimenta rumo ao alvo até que encontre um obstáculo o qual tem que circular para prosseguir no sentido do alvo, porem para melhor desempenho nesse algoritmo "o robô não precisa circular inteiramente os obstáculos, ou seja, o robô circula o obstáculo até que alcance um novo ponto na linha mais próxima do alvo" [1].

Quando o robô tem um sensor, o algoritmo Tangent Bug é uma derivação do Bug 1 e 2 que pode utilizar essa informação de distância do obstáculo para encontrar o caminho mais curto, segundo as leituras atuais do sensor, para o alvo.

O Tangent Bug, Figura 5 apresenta um deslocamento em sentido do ponto alvo até encontra um obstáculo, onde o algoritmo apresenta um intervalo de descontinuidade do alvo, realizando então uma movimentação tangente ao obstáculo até detectar novamente o caminho de sentido ao alvo.

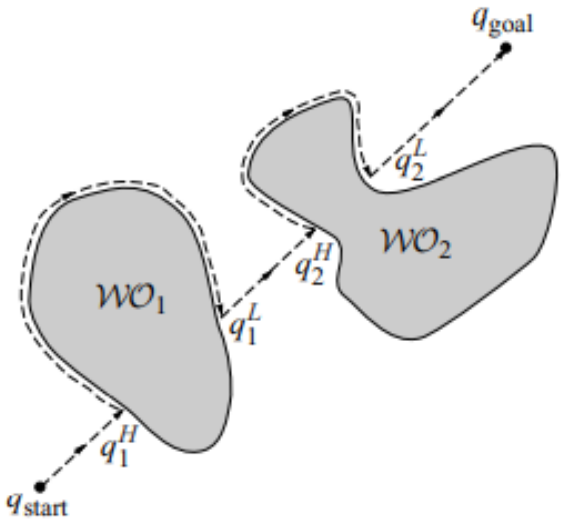


Figura 4: Algoritmo Bug2 [1]

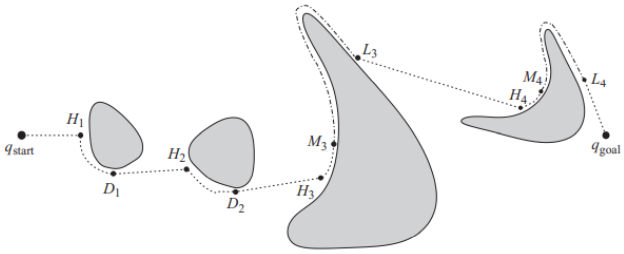


Figura 5: Tangent Bug[1]

2.1.1 Implementação e Resultados

Conforme esperado o robô realizou a movimentação continua rumo ao ponto alvo, apresentando característica adequadas de deslocamento no ambiente, ou seja, em ambiente livre de obstáculos apresentou movimentação precisa e direta para o ponto alvo. Durante a trajetória para o alvo, quando o robô localizou obstáculo desconhecido, o mesmo apresentou a execução da função de movimentação tangente, que permite deslocamento em torno do obstáculo até ocorre novamente a identificação do ambiente livre que propicia a movimentar para o ponto alvo. Inicialmente o robô apresentou colisão com os obstáculos devido a ajustes incorretos nas configurações do sensor, após correção de tais paramentos foi possível identificar uma aproximação adequada dos obstáculos e o movimento de deslocamento assertivo tangencial ao obstáculo.

O algoritmo implementado segue praticamente a mesma definição de Choset et al. [1], com algumas exceções. O algoritmo consegue identificar descontinuidades e somente as segue, caso o robô esteja no comportamento de *motion to goal* e não haja nenhum obstáculo que obstrua o objetivo dentro de seu campo de visão. Caso haja algum obstáculo, então o robô continua seu caminho até o alvo com o acréscimo de um vetor tangente ao ponto mais próximo do obstáculo e uma distância de segurança de 0.8 m. Para o comportamento de *boundary following*, o robô segue a tangente e normal ao ponto mais próximo do obstáculo, calculados da mesma forma que para o caso anterior.

Para o cálculo dos vetores tangente e normal aos obstáculos, utilizou-se a distância euclidiana suavizada [2]. Uma vez que o vetor normal pode ser aproximado pelo gradiente da distância suavizada, necessita-se apenas calcular o seu gradiente. Assim, Gonçalves [2] define o gradiente dessa distância por (1)

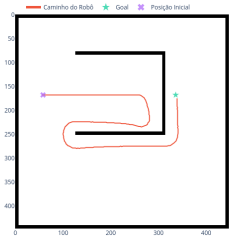
$$\nabla D_h(p) = \frac{\sum_{i=1}^{n} e^{-\frac{\|p-a_i\|^2}{2h^2}}(p-a_i)}{\sum_{i=1}^{n} e^{-\frac{\|p-a_i\|^2}{2h^2}}}, \tag{1}$$

onde p é a posição do robô, a_i são os pontos vistos dos obstáculos e h é um hiperparâmetro definido, na implementação, como 0.1.

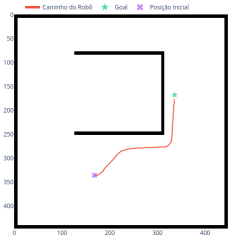
Dessa forma, para que se defina o vetor tangente, apenas se rotaciona o vetor normal em $\pm 90°$, sendo o sentido da rotação definido pelo ângulo do último movimento até o objetivo, dentro do comportamento *motion to goal*. Os pontos dos obstáculos são obtidos por meio do sensor laser, com base na distância entre o robô e cada ponto visto do obstáculo, mapeia-se por meio de matrizes de transformação homogênea, os pontos dos obstáculos, com referência ao *frame* do robô, para o referencial do mundo. Munido de uma nuvem de pontos dos obstáculos e do cálculo de vetores normais e tangentes aos mesmos, pôde-se então implementar completamente os comportamentos de *motion to goal* e *boundary following*. Uma vez que o cálculo do gradiente da distância não é calculado apenas quando o comportamento *motion to goal* não encontra um obstáculo em seu caminho, esse é o único caso no qual se alimenta o controlador com pontos a serem seguidos. Dessa forma, nos outros casos, passa-se ao controlador a prórpia velocidade a ser seguida, ou seja, a soma vetorial das componentes normal e tangente calculadas, com uma distância de segurança de 0.8 m.

Para a implementação completa do algoritmo, apenas falta tratar o caso onde não há caminho para o objetivo. Para isso, armazena-se todos os pontos por onde o robô passa quando está no comportamento de *boundary following* e, caso algum desses N pontos seja próximo de algum dos $N - 15$ pontos anteriores, o algoritmo conclui que não há caminho possível até o objetivo. Esse critério de excluir os últimos 15 pontos é arbitrário e visa garantir que o algoritmo não se encerre só porque o robô está próximo de um ponto anterior, o que é óbvio. A condição adotada infelizmente não garante o funcionamento perfeito desse tratamento de caso, uma vez que há momentos nos quais o robô circula mais de uma vez o mesmo perímetro para concluir que não há caminho e outros nos quais o robô conclui que não há caminho possível sem mesmo percorrer o obstáculo inteiro.

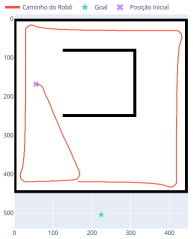
Para as simulações feitas, Figura 6, nota-se o funcionamento do algoritmo em *motion to goal* e *boundary following* perfeitamente para dois casos distintos, Figura 6a e Figura 6b, nos quais o robô segue até o alvo em linha reta até encontrar um obstáculo, no primeiro caso o comportamento *boundary following* é invocado, uma vez que a distância até o objetivo começa a aumentar, já no segundo o robô somente desvia do obstáculo enquanto converge para o objetivo. Um último teste, Figura 6c, representa o objetivo inalcançável. Nesse último caso, percebe-se que o robô segue uma linha reta até o objetivo, entretanto, ao encontrar um obstáculo, começa a circular o bordo do mesmo, até que retorna a um ponto próximo de um ponto visitado e, assim, conclui corretamente que não existe caminho até o objetivo.



(a) Robô inicialmente em $(-6, 2)$m ou $(56, 168)$px convergindo para o objetivo



(b) Robô inicialmente em $(-2, -4)$m ou $(168, 336)$px convergindo para o objetivo



(c) Robô inicialmente em $(-6, 2)$m ou $(56, 168)$px percorrendo todo o obstáculo

Figura 6: Caminhos resultantes por meio do algoritmo Tangent Bug para três configurações iniciais diferentes do robô. Nos dois primeiros casos, o objetivo está no ponto $(4, 2)$m ou $(336, 168)$px. No último, o objetivo é inalcançável e corresponde ao ponto $(0, -10)$m ou $(224, 504)$px

2.2 Path Following

2.2.1 Implementação e Resultados

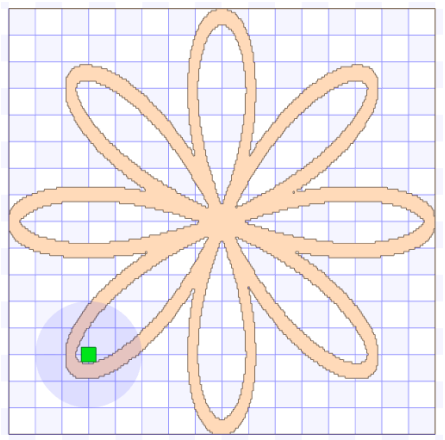
Para implementar o seguimento de caminho, adotou-se uma alimentação ponto a ponto para o controlador, isso é, o próximo ponto a ser seguido, somente é gerado quando o robô chega próximo o suficiente do ponto seguido atual. Dessa forma, o caminho não é dependente do tempo, o que é vantajoso, visto que seguimento de trajetórias nem sempre são viáveis, uma vez que qualquer atraso nos sensores ou controlador pode causar um desvio do caminho ideal durante o seu seguimento. Um simples seguimento de caminho, feito ponto a ponto, consegue garantir que, uma vez que o robô esteja no ponto inicial da curva, ele nunca irá "pular" pontos da curva e realizar um caminho fora do planejado. Portanto, a adoção do seguimento de caminho independente do tempo é justificada pelo fato de que o robô deve circular eternamente a curva definida e em um tempo infinito, há diversas oportunidades para atrasos surgirem.

Tomou-se como curva a ser seguida uma rosa polar de oito pétalas, parametrizada por (2), (3). Adotou-se $\omega = \frac{\pi}{5}$ rad/s e a discretização do tempo por um passo de $1/20$. O passo somente é acrescido no tempo discretizado atual, quando o robô está a 50 cm do ponto seguido, ou seja, somente quando essa situação é atingida, gera-se o próximo ponto na curva a ser seguido.

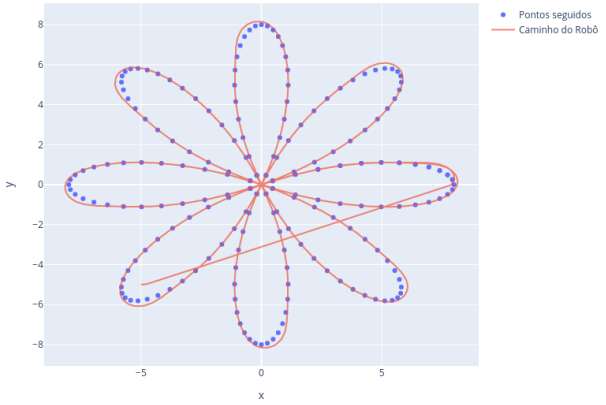
$$x = 8\cos(4\omega t)\cos(\omega t) \tag{2}$$

$$y = 8\cos(4\omega t)\sin(\omega t) \tag{3}$$

O ambiente de simulação adotado, junto da respectiva curva a ser seguida podem ser vistos na Figura 7a. O resultado das considerações adotadas, tanto para geração discretizada da curva, quanto para o caminho seguido pelo robô podem ser vistos na Figura 7b. Nota-se que a tolerância de 50 centímetros adotada, permite com que o robô saia parcialmente do trajeto ao realizar curvas mais fechadas. Entretanto, nota-se que o robô segue o caminho sem quaisquer problemas e sem pular nenhum ponto intermediário, o que poderia ocorrer num seguimento de trajetória não ideal, portanto se confirma a justificativa para a adoção de um seguimento de caminho.



(a) Mapa utilizado para simulação. Robô em verde e curva a ser seguida representada em rosa



(b) Caminho gerado a ser seguido e caminho percorrido pelo robô

Figura 7: Resultado do Path Following

2.3 Potential Function

Os algoritmos Bug apresentam algumas limitações tendo em vista sua aplicação, entre as quais estão a necessidade de utilização em configuração bidimensional simples, por isso a utilização de algoritmos como o de função potencial permite a execução em ambientes multidimensionais e não euclidianos. De acordo com Choset et al. [1], "uma função potencial é uma função de valor real diferencial. O valor de uma função potencial pode ser visto como energia e, portanto, o gradiente do potencial é a força". Neste algoritmo é necessário realizar algumas considerações iniciais, como por exemplo, o robô é uma carga positiva no espaço, já o alvo é uma carga negativa que tende a atrair o robô, por outro lado os obstáculos recebem atribuição de comportamento como cargas positivas que realizam a repulsão do robô. Com a combinação resultante de forças de atração e repulsão atuando sobre o robô, o mesmo tende a direcionar para o ponto alvo.

As funções potenciais mais simples,Figura 8, podem ser implementadas utilizado uma soma dos campos potencial atrativo e potencial repulsivos. A função permite determinar uma resultante que conduz a movimentação do robô para o alvo, enquanto se desvia dos obstáculos na trajetória. O potencial atrativo é dado em Choset et al. [1] por (4) e seu respectivo gradiente por (5), já o potencial atrativo e seu gradiente, por (6) e 7, respectivamente.

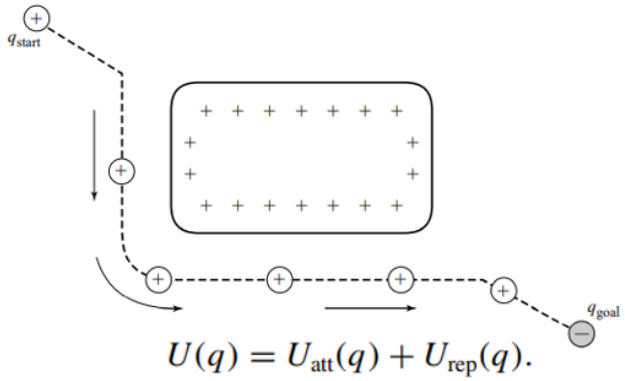


Figura 8: Representação eletromagnética das funções potenciais

$$U_{att}(q) = \begin{cases} \frac{1}{2}\zeta d^2(q, q_{goal}), & d(q, q_{goal}) \leq d^*_{goal} \\ d^*_{goal}\zeta d(q, q_{goal}) - \frac{1}{2}\zeta (d^*_{goal})^2, & c.c. \end{cases} \tag{4}$$

$$\nabla U_{att}(q) = \begin{cases} \zeta(q - q_{goal}), & d(q, q_{goal}) \leq d^*_{goal} \\ \frac{d^a st_{goal}\zeta(q-q_{goal})}{d(q,q_{goal})}, & c.c. \end{cases} \tag{5}$$

$$U_{rep}(q) = \sum_{i=1}^{N} U_{rep_i}(q), \ \nabla U_{rep}(q) = \sum_{i=1}^{N} \nabla U_{rep_i}(q)$$

$$U_{rep_i}(q) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{d_i(q)} - \frac{1}{Q^*}\right)^2, & d_i(q) \leq Q^* \\ 0, & c.c. \end{cases} \tag{6}$$

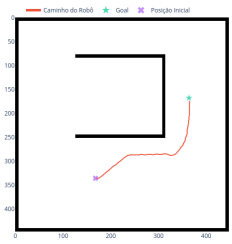
$$\nabla U_{rep_i}(q) = \begin{cases} \eta \left(\frac{1}{Q^*} - \frac{1}{d_i(q)}\right) \frac{1}{d_i^2(q)} \nabla d_i(q), & d_i(q) \leq Q^* \\ 0, & c.c. \end{cases} \tag{7}$$

2.3.1 Implementação e Resultados

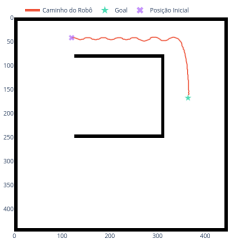
Conforme descrito em [1], implementou-se a função de potencial atrativa por meio de uma combinação de potencial quadrático e cônico,(4) e (5), garantindo convergência linear para o objetivo quando o robô está a uma distância maior que 3 m do mesmo, e uma convergência quadrática caso contrário. Como potencial repulsivo, adotou-se a soma das componentes de força repulsiva dadas por cada obstáculo observado,(6) e (7). Caso a distância do robô até o obstáculo seja maior que 1.5 m, o potencial repulsivo é nulo, caso contrário, o potencial repulsivo é calculado de forma tal que quanto mais próximo o robô está do obstáculo, mais forte é a repulsão. Dessa forma adotou-se os parâmetros $\zeta = 5, \eta = 10, d^*_{goal} = 3, Q^* = 1.5$.

Uma vez que o algoritmo calcula o gradiente da função potencial, cuja grandeza é a velocidade, alterou-se então a interação com o controlador. Ao invés de se passar o ponto a ser seguido, passa-se agora a própria velocidade, gradiente do potencial, calculada. A opção tomada para a implementação não necessita então do gradiente descendente, como descrito em [1].Caso a norma euclidiana do vetor gradiente do potencial seja menor que 0.3, o algoritmo se encerra e é concluído que o objetivo foi alcançado.

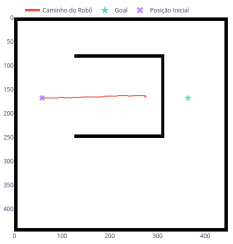
Pode-se ver na Figura 9, três casos diferentes de simulação. Para os dois primeiros, Figura 9a e Figura 9b, vê-se que o robô convergiu perfeitamente para o objetivo sem quaisquer colisões. Entretanto, na Figura 9c, nota-se que o robô converge para um mínimo local, ponto no qual o potencial repulsivo e atrativo tem a mesma intensidade. É possível evitar a convergência para mínimos locais, dando pulsos aleatórios para o controlador quando é identificado um gradiente do potencial muito baixo, sendo que o robô não está próximo do objetivo final, como descrito por Choset et al. [1], porém isso não foi implementado, o que torna a convergência para mínimos locais uma limitação da implementação.



(a) Robô inicialmente em $(-2, -4)$m ou $(168, 336)$px convergindo para o objetivo



(b) Robô inicialmente em $(-3.7, 6.5)$m ou $(120, 42)$px convergindo para o objetivo



(c) Robô inicialmente em $(-6, 2)$m ou $(56, 168)$px convergindo para um mínimo local

Figura 9: Caminhos resultantes por meio da função potencial para três configurações iniciais diferentes do robô. Em todos os casos o objetivo está no ponto $(5, 2)$m ou $(364, 168)$px

2.4 Wave-Front Planner

O mínimo local é o principal problema existente em algoritmos de gradiente decrescente, como o algoritmo da função potencial atrativa e repulsiva. Esse problema é provocado pela ação resultante de forças, onde, por exemplo, o robô é atraído para o objetivo e as paredes do obstáculo a repulsão, esse equilíbrio de forças limita a movimentação do robô no ambiente, conforme a Figura 10. Essa não é a única situação onde o robô converge para um mínimo local, uma vez que em qualquer ponto é possível que a soma das potenciais atrativa e repulsiva seja nula.

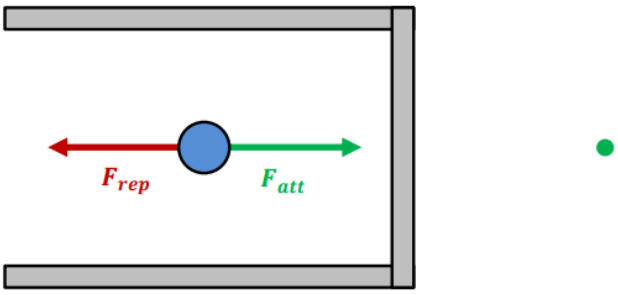


Figura 10: Exemplo de mínimo local

Uma das soluções para resolução do problema de mínimo local é aplicar forças aleatórias. O mais simples método de resolução desse problema é o algoritmo Wave-Front, Figura 11, que consiste em uma implementação no espaço com grid. Nesse algoritmo é estabelecido que o espaço livre é rotulado em zero e os obstáculos em 1, tendo em vista essa configuração o pixel do objetivo é rotulado em 2, sendo assim, os espaços começam a ser rotulados gradativamente de forma a incrementa o valor do pixel anterior. Após a rotulação do espaço é possível determinar o menor caminho de deslocamento rumo ao ponto alvo.

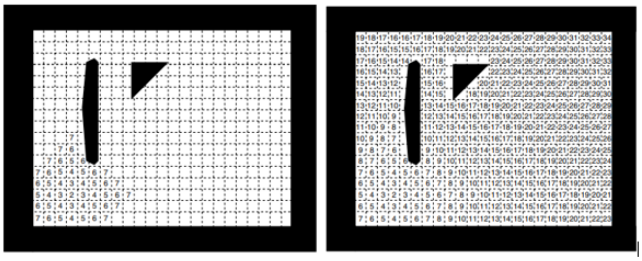


Figura 11: Algoritmo Wave-Front

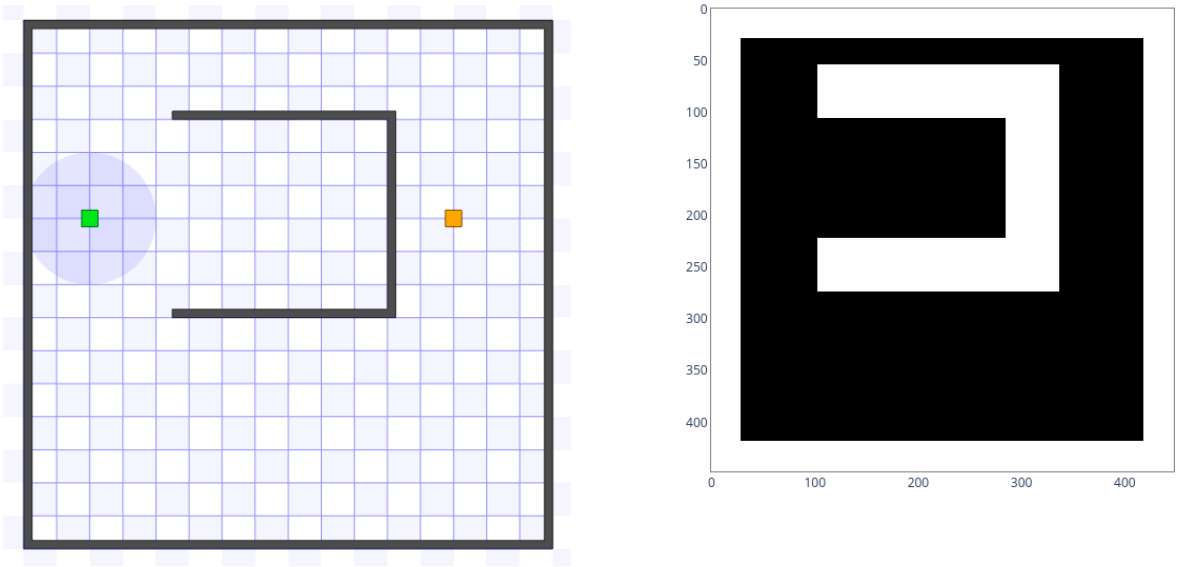
2.4.1 Implementação e Resultados

Para o algoritmo Wave-Front, tomou-se como premissa a existência de um *grid* construído a priori que representa o mapa em espaço de estados, isso é, com os obstáculos expandidos por um múltiplo das dimensões do robô. Dessa forma, é possível tratar o robô como puntiforme e então realizar o planejamento com base nos pixels e seus respectivos pesos gerados pelo algoritmo.

Implementou-se o Wave-Front Planner conforme descrito por Choset et al. [1]. Adotou-se como mapa de simulação o mapa apresentado na Figura 12a, onde o robô é representado por um quadrado verde e o objetivo por um quadrado laranja. Esse mapa tem dimensões 16×16 metros e o robô tem tamanho 0.5×0.5 metros.

Expandiu-se os obstáculos do mapa da Figura 12a por $1.2\sqrt{2}$, valor maior que a diagonal do robô para garantir que, mesmo com uma tolerância de erro, o robô não bata. Adotou-se uma resolução de 28 px/m e, ainda, representou-se os obstáculos por valores 1 e o espaço livre por valores 0. Dessa forma, obteve-se o mapa em espaço de configurações apresentado na Figura 12b.

Tomou-se 0.3 m como a tolerância para se garantir que o robô alcançou o objetivo, dessa forma, ele não precisa alcançar a posição exata, apenas uma região a 30 centímetros do objetivo.



(a) Mapa utilizado para simulação. Robô em verde e obstáculo em laranja

(b) Mapa com obstáculos expandidos (em espaço de configurações). Espaço livre em preto (0) e obstáculos em branco (1)

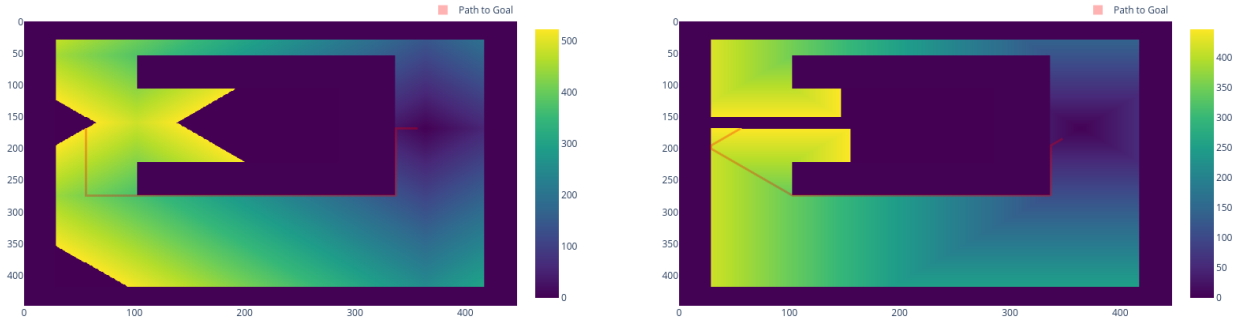
Figura 12: Mapas utilizados para o algoritmo Wave-Front

Dado que é conhecido, a priori, o mapa em espaço de configurações, a posição inicial do

robô e a posição do objetivo, o algoritmo implementado calcula o planejamento Wave-Front no início de sua execução, tendo como opções a conectividade de 4 ou 8 pontos. O algoritmo então mapeia a posição inicial do robô, em metros, para coordenadas em pixel, dessa forma, obtém-se o peso calculado para o pixel inicial w_0. Assim, o algoritmo seleciona como próximo ponto, o primeiro pixel vizinho cujo peso calculado é igual a $w_1 = w_0 - 1$, a posição desse pixel p_1 é armazenada. Passa-se ao controlador as coordenadas em metros, P_1, equivalentes ao pixel p_1. Quando o robô chega em uma região 30 cm próxima do objetivo, calcula-se então o próximo ponto. Esse próximo ponto não é mais calculado com base na posição atual do robô, visto que a tolerância imposta poderia fazer com que o robô não seguisse o planejamento calculado perfeitamente, assim, toma-se como próximo ponto o pixel vizinho de p_1 cujo peso é $w_2 = w_1 - 1$. Ou seja, somente na primeira iteração, é considerado o peso da posição atual do robô, nas demais, toma-se o peso do pixel seguido anteriormente. Assim, o algoritmo realiza esse processo pixel a pixel, até que o robô convirja para o objetivo.

O seguimento de pontos, feito pixel a pixel, garante um caminho perfeito até o objetivo, porém, de forma lenta. Por isso, adotou-se como velocidades de translação mínima 1 m/s em cada direção, o que acaba resultando em rotações indesejadas durante a simulação. Uma vez que o robô tenta seguir perfeitamente pixel a pixel, isso é, uma variação de 1/28 m, a adoção da velocidade mínima causa certas rotações no robô, já que o controlador impõe uma velocidade angular não nula em algumas situações.

Os resultados do planejamento Wave-Front para conectividades de 4 e 8 pontos, podem ser vistas nas Figura 13a e Figura 13b, respectivamente, assim como o caminho seguido pelo robô. Para ambos os resultados, nota-se em simulação, as rotações indesejadas supracitadas, principalmente durante o trecho mais próximo do objetivo.



(a) Conectividade de 4 pontos

(b) Conectividade de 8 pontos

Figura 13: Planejamentos gerados pelo algoritmo Wave-Front. Em ambos os casos de conectividade, o robô se encontra inicialmente no ponto $(-6,\ 2)$ m ou $(56,\ 168)$px. O objetivo se encontra em $(5,\ 2)$ m ou $(364,\ 168)$px. A escala de cores indica o peso de cada pixel, gerado pelo planejamento Wave-Front e o trajeto vermelho indica o caminho seguido pelo robô

3 Conclusão

Todos os algoritmos implementados apresentaram resultados satisfatórios. Alguns pequenos problemas foram percebidos e devem ser corrigidos a posteriori, como por exemplo, a melhoria da identificação de não existência de caminho até o objetivo para o algoritmo Tangent Bug. Além disso, propõe-se também melhorar o algoritmo Wave-Front para que o robô siga o caminho de uma forma a não gerar rotações desnecessárias.

Por meio da implementação de todos os métodos, foi possível unir grande parte da teoria aprendida. Pôde-se utilizar o conceito de espaço de configurações no algoritmo Wave-Front, por exemplo, permitindo com que o robô seja tratado como um ponto, facilitando o mapeamento pixel-metros, assim como o planejamento de caminho. Pôde-se observar na prática as dificuldades enfrentadas por um roboticista, assim como a importância da criação de novos métodos, principalmente ao se avaliar o comportamento de funções de potencial simples.

Em suma, foi possível implementar os algoritmos apresentados em Choset et al. [1] de forma satisfatória e, assim, compreender os pseudocódigos tão alto-níveis, da referência, de uma forma um pouco mais baixo-nível.

Referências

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005. [Online]. Available: https://www.scholars.northwestern.edu/en/publications/principles-of-robot-motion-theory-algorithms-and-implementations

[2] V. M. Gonçalves, "Artigo ainda não publicado," Jan. 2022.