Trabalho Prático 2 de Robótica Móvel

Marco Túlio P. Tristão[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, BH – Brasil

marcotuliopin@ufmg.br

Resumo. *O presente relatório descreve as atividades realizadas no Trabalho Prático 2 da disciplina de Robótica Móvel, focado no exercício de técnicas iniciais de planejamento de caminhos, navegação e controle. O objetivo deste trabalho é proporcionar o aprofundamento do conhecimento desses tópicos por meio da prática, utilizando a plataforma de simulação CoppeliaSim e a integração com scripts Python.*

1. Introdução

Neste trabalho, implementamos dois algoritmos de controle e navegação: Roadmap utilizando um robô holonômico, e Campos Potenciais utilizando um robô diferencial. No último algoritmo, usamos o controlador [De Luca and Oriolo 1994].Os programas foram criados com auxílio dos materiais da disciplina [Macharet 2024].Para cada algoritmo, utilizamos duas cenas com diferentes níveis de dificuldade para realizar experimentos.

O algoritmo de Roadmap implementado utiliza a decomposição em grid, em que o cenário é decomposto em células, que são marcadas como ocupadas ou livres. Entre as células livres, é encontrado o menor caminho entre o robô e o alvo. Já o algoritmo de Campos Potenciais utiliza da diferença entre a força de atração, gerada pelo alvo, e as forças de repulsão, geradas pelos obstáculos, para controlar a navegação do robô.

Este relatório organiza a documentação do tabalho. Cada tópico será detalhado em uma seção distinta, de acordo com o enunciado. Também serão apresentadas outras informações julgadas relevantes, como descrição de partes da implementação e como executar o código. As duas últimas seções compõe os comentários finais e as referências bibliográficas.

2. Implementação

2.1. Campos Potenciais

O algoritmo de Campos Potenciais se baseia em encontrar a força resultante da soma da força de atração gerada pelo alvo com as forças de repulsão geradas pelos obstáculos. As forças de atração de repulsão são dadas por:

$$f_{\text{attr}} = k_{\text{att}} \cdot (p_{\text{goal}} - p_{\text{curr}})$$

Onde k_{att} é o peso da força de atração, p_{goal} é a posição do alvo e p_{curr} é a posição atual do robô.

Para cada obstáculo, somamos a força de repulsão gerada por ele, para calcular assim a força de repulsão total. Fazemos isso da seguinte forma:

$$
\begin{aligned}
dv &= p_{\text{curr}} - p_{\text{obs}} \\
d &= ||dv|| \\
f_{\text{rep}} &+= k_{\text{rep}} \cdot \left(\frac{1}{d^2}\right) \cdot \left(\frac{1}{d} - \frac{1}{R}\right) \cdot \left(\frac{dv}{d}\right)
\end{aligned}
$$

Onde k_{rep} é o peso da força de repulsão, p_{obs} é a posição do obstáculo (no referencial global, assim como os demais) e R é o alcance máximo de influência de ums obstáculo.

À soma dessas forças, acrescentamos uma pequena componente aleatória, responsável por ajudar o robô a sair de mínimos locais (um problema crônico desse algoritmo, como mostrado em [Hwang and Ahuja 1992]), mas na prática essa medida não foi uma solução para o problema, apesar de amenizá-lo em casos simples.

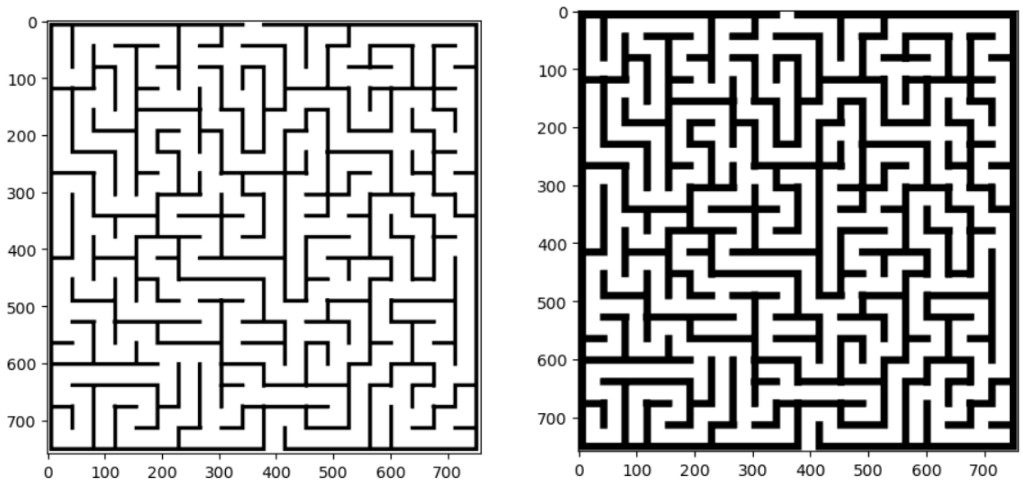
Calculada a força total, usamos o controlador [De Luca and Oriolo 1994] para enviar as informações para o robô diferencial. Apesar de funcionar para o propósito (alcançar o alvo), o controlador não se mostrou eficiente na interação do controle do robô com as forças de repulsão, no sentido em que, em espaços apertados (obstáculos muito próximos uns dos outros), o robô realiza muitas movimentações para deslocar uma distância curta.

Esse algoritmo conseguiu criar uma solução para ambientes simples, com obstáculos não-simétricos e esparsos. No entanto, em ambientes apertados, o controlador não performou bem, pois ele induz ao robô realizar manobras grandes para se locomover. Como as forças de repulsão em locais apertados impedem essas manobras, o robô não conseguia avançar. Além disso, cenários com obstáculos simétricos (por exemplo uma parede reta na frente do obstáculo e perpendicular à trajetória do robô) interrompiam a locomoção do robô, uma vez que as forças de repulsão e de atração se anulavam e o robô se prendia em um mínimo local.

2.2. Roadmap

Inicialmente, carregamos o mapa do cenário. Ao escolher o caminho que o robô irá percorrer, corremos o risco de não considerarmos o diâmetro do robô. Para resolver esse problema, dilatamos os obstáculos do mapa original. Assim, garantimos que o caminho escolhido não fará o robô se chocar com obstáculos.

Com a posse do mapa do cenário, criamos um grafo em que cada nó corresponde a um quadrante. Por exemplo, para um cenário de tamanho 50x50, podemos criar um grafo bidimensional de 50x50 nós, em que cada nó corresponde a uma área 1x1 do cenário original. Para cada mapa, escolhemos um tamanho de quadrante distinto (chamado de *cell_size* no código) para se adequar melhor às especificidades do cenário, como por exemplo a proximidade entre os obstáculos.



(a) Mapa original. (b) Mapa com obstáculos dilatados.

Figure 1. Resultado da dilatação dos obstáculos na cena *square_maze.ttt*.

Depois disso, removemos do grafo os nós que correspondem a quadrantes tomados por obstáculos. Um quadrante é tomado por obstáculo se alguma área do mesmo for ocupada por um obstáculo. Para finalizar a construção do grafo, conectamos os nós vizinhos. Nós diagonalmente vizinhos são conectados somente se os vizinhos comuns aos dois existirem no grafo (forem posições livres).

Utilizamos um algoritmo de menor caminho para projetar a trajetória do robô no grafo. Com isso, podemos descobrir a trajetória real convertendo as coordenadas no grafo para as coordenadas na cena utilizando as equações abaixo.

A transformação das coordenadas da cena para as coordenadas do grafo é dada por:

$$\text{arr}_x(x) = \left\lfloor \frac{x_{\text{dim}} + x}{cell\_size} \right\rfloor$$
$$\text{arr}_y(y) = \left\lfloor \frac{-y + y_{\text{dim}}}{cell\_size} \right\rfloor$$

A transformação das coordenadas do grafo para as coordenadas da cena é dada por:

$$\text{scene}_x(x) = (x \cdot cell\_size) - x_{\text{dim}}$$
$$\text{scene}_y(y) = y_{\text{dim}} - (y \cdot cell\_size)$$

Em que x_{dim} e y_{dim} são a metades das dimensões correspondentes da cena. Por exemplo, em uma cena 10x20, $x_{dim} = 5$.

Uma explicação detalhada do controlador que faz o robô executar o trajeto calculado é dada a seguir:

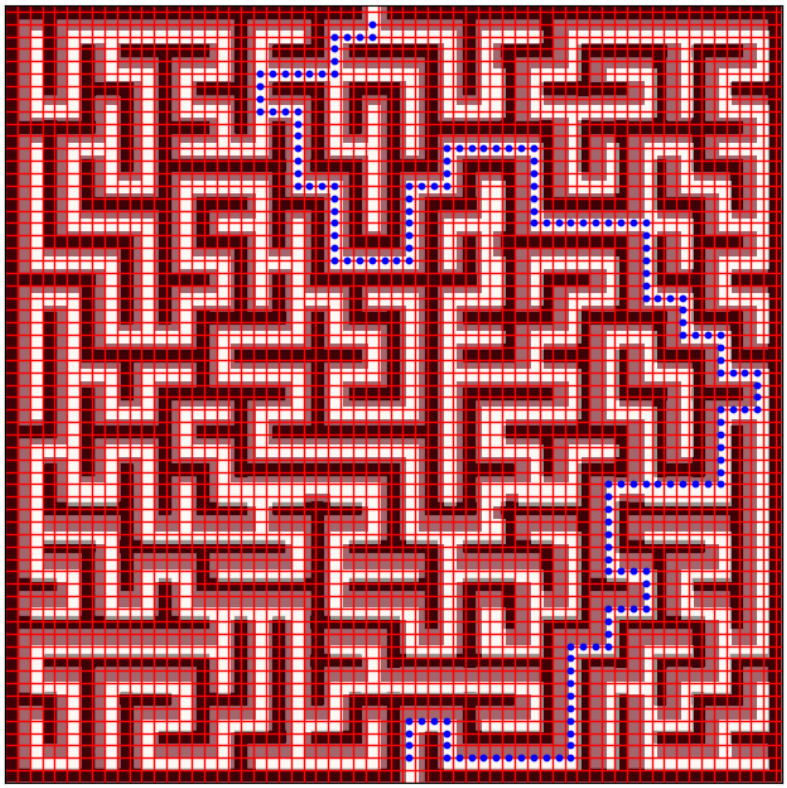


Figure 2. Caminho do robô ao longo dos quadrantes.

1. A cinemática direta configura a matriz de transformação que relaciona as velocidades das rodas com as velocidades lineares e angulares do robô em seu quadro de referência. A matriz de ganhos é usada posteriormente no controlador.
2. O loop principal itera sobre cada posição alvo do trajeto. Para cada uma, ele converte a posição de coordenadas do gráfico para coordenadas da cena usando as funções $scene_x()$ e $scene_y()$.
3. Entramos em um loop interno. O erro entre a posição alvo e a posição atual é calculado. Se esse erro for menor que um certo limite (0.05 neste caso), considera-se que o robô atingiu o alvo e o loop interno é interrompido.
4. Ainda dentro do loop interno, a velocidade desejada do robô é calculada multiplicando a matriz de ganho pelo vetor de erro normalizado. Este é um controlador proporcional simples, onde a ação de controle é proporcional ao erro.
5. Também dentro do loop interno, realizamos a cinemática inversa: as velocidades desejadas das rodas são calculadas multiplicando o inverso da matriz de cinemática direta transformada pela velocidade desejada do robô. A matriz de transformação é rotacionada pela orientação atual do robô para levar em conta sua direção.
6. Finalmente, as velocidades desejadas das rodas são definidas.

Um grande desafio na implementação desse algoritmo foi controlar o robô em ambientes apertados. Um grid com células pequenas permitia a passagem por locais pequenos facilmente, mas poderia fazer o robô se deslocar muito próximo à obstaculos e eventualmente se chocar. Outro desafio foi controlar o robô ao redor de obstáculos curvos de maneira eficiente. Era necessário conectar vizinhos diagonais, a fim de permitir uma

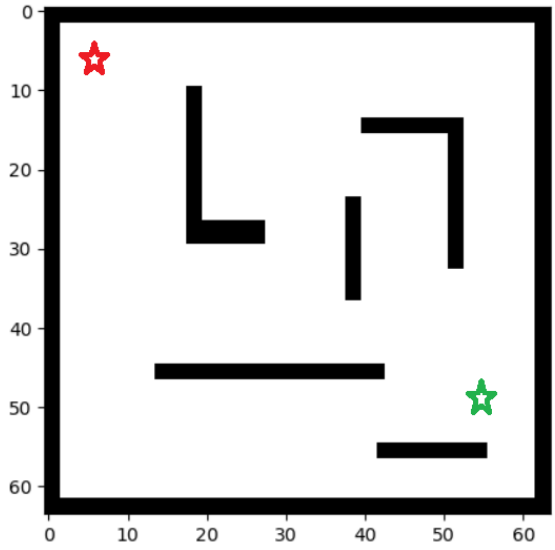


Figure 3. Mapa da cena *walls.ttt*.

locomoção mais eficiente, mas evitando obstáculos que poderiam estar na quina entre os nós. Além disso, a imprecisão da posição do robô dentro da célula se mostrou um problema. Nesse método podemos controlar em qual quadrante o robô estará, mas não sua posição dentro do mesmo.

3. Testes

Testamos os métodos em diferentes ambientes: *walls.ttt, cave.ttt e square_maze.ttt* (mostrado na figura 2.2). Abaixo temos os resultados comparativos do tempo levado por cada método para partir da posição inicial (ponto verde) e alcançar o objetivo (ponto vermelho):

1. ***walls.ttt*** - 40 segundos para o Campos Potenciais e 12 segundos para Roadmap. O último, além de mais rápido, foi mais eficiente no número de movimentações que o robô fez.
2. ***cave.ttt*** - como pode ser visto na figura 3, o objetivo está na borda superior de uma parede extensa. Quando iniciado o robô na posição inferior direita, o algoritmo de Campos Potenciais cai em um mínimo local na parte inferior dessa parede, enquanto o Roadmap demora 33 segundos. Já quando iniciado na parte de cima do mapa, com Campos Potenciais o robô alcança o objetivo em 30 segundos, enquanto o Roadmap demora 25 segundos.
3. ***square_maze.ttt*** - o algoritmo de Campos Potenciais, além de cair em mínimos locais, fica preso em caminhos sem saída no labirinto. Como em Roadmap há um planejamento prévio do caminho, o robô consegue atravessar o labirinto e encontrar o objetivo (a outra saída do labirinto).

4. Conclusão

Nesse trabalho, implementamos dois algoritmos de navegação e controle de robôs. Como os dois métodos são iniciais e servem somente de base para métodos mais avançados e

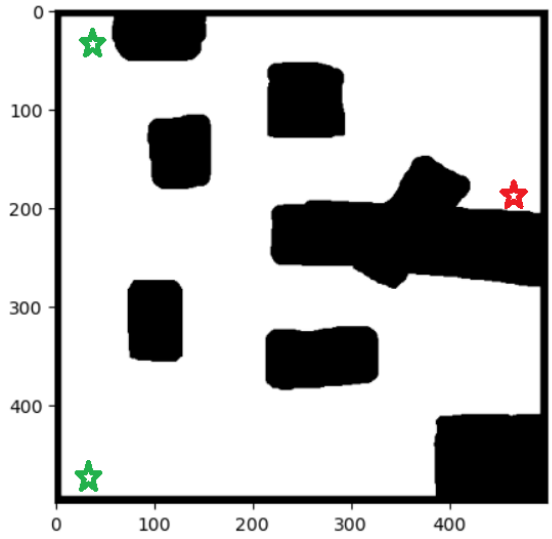


Figure 4. Mapa da cena *cave.ttt*.

com aplicabilidade real, é de se esperar que a performance dos mesmos não seja excepcional em casos variados. Percebe-se que principalmente o Campos Potenciais, especialmente com o controlador utilizado, não se saiu bem nos testes realizados. Esse método, apesar de muito valioso como base teórica para vários outros, possui muitas fraqueza (não é completo, por exemplo) e pouca aplicabilidade prática. Já o Roadmaps se saiu bem melhor em casos mais complexos e nos mesmos casos testados com Campos Potenciais. Uma grande dificuldade foi relacionar as coordenadas do nó para as coordenadas da cena. A imprecisão da transformação de coordenadas ocasionou por vezes em comandos inválidos para o robô, como dirigí-lo para um obstáculo.

Apesar dessas dificuldades para se atingir resultados satisfatórios nos testes (até mesmo pelas fraquezas inerentes aos algoritmos), foi possível atingir um grande conhecimento das técnicas e teorias empenhadas, além de maior familiarização com o simulador CoppeliaSim.

5. Instruções

Para executar o notebook do Roadmaps, é necessário baixar o OpenCV (rode a primeira célula do notebook caso seja necessário), pois ele é usado na expansão dos obstáculos. Além disso, selecione a cena correta que você deseja rodar. Essa escolha é feita na célula indicada no notebook.

References

De Luca, A. and Oriolo, G. (1994). Local incremental planning for nonholonomic mobile robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 104–110. IEEE.

Hwang, Y. K. and Ahuja, N. (1992). A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32.

Macharet, D. (2024). Robótica móvel. Material didático da disciplina de Robótica Móvel ministrada pelo professor Douglas Macharet no curso de Ciência da Computação da Universidade Federal de Minas Gerais.