Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Engenharia Elétrica

Robôs Móveis Autônomos

Projeto de Implementação

Planejamento de trajetória

Prof. Dr. Kelen Teixeira Vivaldini
Professor(es) Responsável(is)

Integrantes do Grupo

Vilson Bueno da Silva Junior RA: 769803

João Carlos Tonon Campi RA: 769723

23 de março de 2022

Sumário

Lista de Figuras

1. Introdução

1.1. Objetivo

Esse projeto tem como objetivo implementar um algoritmo de planejamento de trajetória, obtendo um caminho de um ponto inicial de origem (X_0, Y_0) até um ponto de destino (X_f, Y_f). É necessário que o robô consiga trafegar no ambiente sem nenhuma colisão com obstáculos ou paredes.

Para a solução, é necessário tratar os aspectos de exploração do mapa, desvio de obstáculos e técnica de planejamento de trajetória.

O ponto de partida e o final estão representados na figura abaixo:

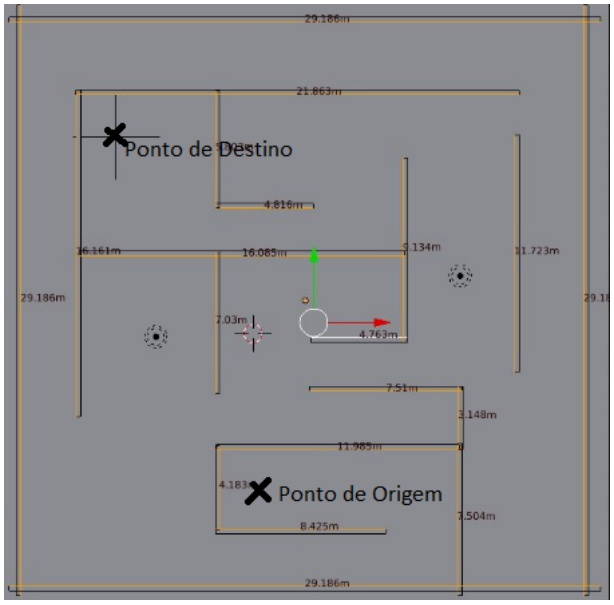

Figura 1: Mapa e objetivo

1.2. Ambiente e plataformas:

O ambiente utilizado nas simulações é o Gazebo, no qual é executado com o mapa do labirinto, como mostrado abaixo:

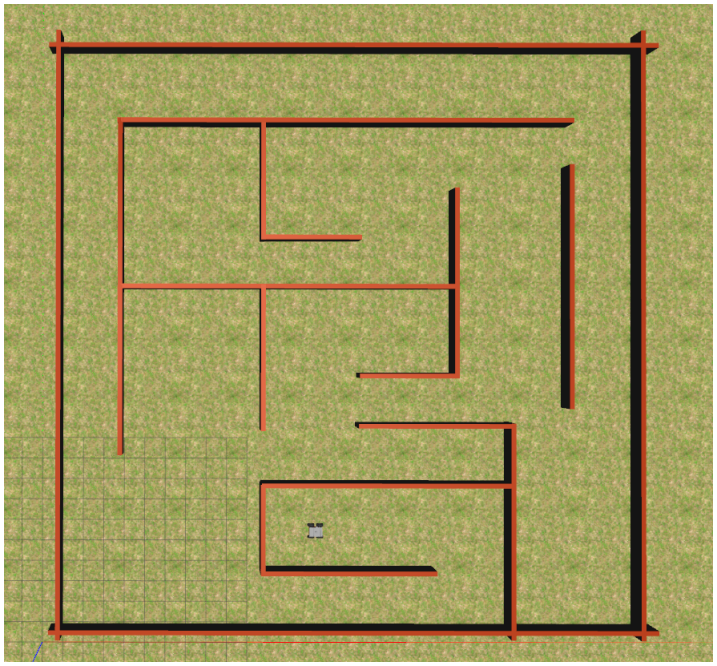

Figura 2: Mapa labirinto no Gazebo

A linguagem de programação utilizada foi o Python integrada ao sistema ROS, sendo realizado o mapeamento a partir do controle manual do robô. A partir do mapa, foram utilzados métodos de processamento de imagens para realizar o planejamento de trajetória com o algoritmo *Wavefront* e, assim, definindo o caminho necessário para o robô chegar ao ponto final.

Todos os tópicos aqui descritos serão detalhados no decorrer do relatório.

2. Mapeamento

2.1. Técnica

A forma utilizada de mapeamento é o nó do ROS *slam_gmapping*, que a partir dos dados do sensor *Laser Scan*.

Para mapear uma região com o *slam_gmapping* é necessário controlar o robô de forma manual e andar por toda a região, assim, ele salva os dados do sensor em uma frequência, pegando todos esses dados e criando um imagem de formato *ymal* do mapa.

O controle do robô é dado pela utilização do *teleop operation*, onde é possível controlar o robô com o teclado. É necessário utilizar uma velocidade baixa e não colidir o robô, de modo a não perder a odometria, o que pode comprometar os dados obtidos pelo sensor.

2.2. Obtenção do mapa:

Para realizar o mapeamento utiliza-se o seguinte comando ao estar com o Gazebo aberto:

```
$rosrun gmapping slam_gmapping scan:=rslidar_points
```

Depois de andar toda a região do mapa, salva-se a imagem do mapa. A imagem do mapa, recortada apenas com o labirinto, se encontra abaixo:

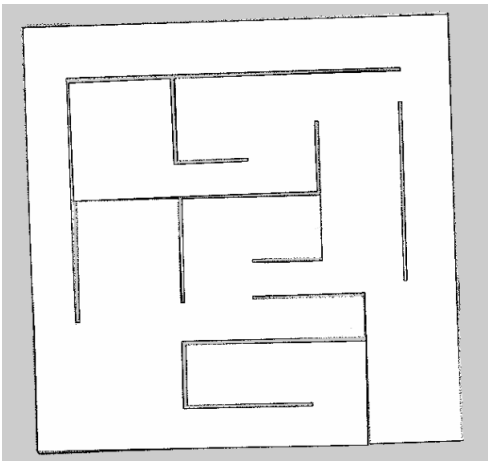

Figura 3: Imagem do mapa

Agora, realiza-se técnicas de processamento de imagem com o mapa pronto.

3. Processamento de imagem

3.1. Dimensionamento

Primeiro, é necessário realizar o dimensionamento da imagem gerada, ou seja, remover a parte cinza do mapa gerado e rotacionar de maneira a ficar reto. A maneira utilizada para rotação, foi pegar dois pontos que deveriam estar retos, pegando a posição X e Y de cada um e realizando o arco tangente da razão entre a diferença das posições, assim obtendo o ângulo para rotação.

A imagem dimensionada se encontra abaixo:

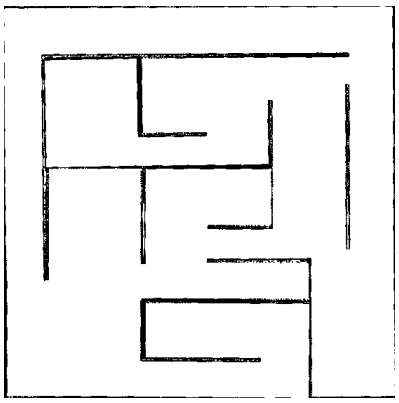

Figura 4: Mapa dimensionado

3.2. Threshold (Segmentação) e Dilatação

Utilizando o algoritmo de *Threshold* para segmentar o mapa de forma que se a cor (em tons de cinza) do pixel da imagem for menor que um certo valor é igual a 0, representando preto e caso contrário é igual a 255, representando branco. Os valores estão representados abaixo:

$$\begin{cases} \text{Se } f(x,y) < 206, \ f(x,y) = 0 \\ \text{Caso contrário}, \ f(x,y) = 255 \end{cases}$$

Agora, realiza-se o engrossamento das paredes utilizando a técnica de dilatação de processamento de imagens, com um núcleo de tamanho 36x36. Isso é feito para gerar uma faixa de segurança para o robô, onde ele evitará se movimentar próximo da parede e gerar uma colisão.

A imagem do mapa depois de processada se encontra abaixo:

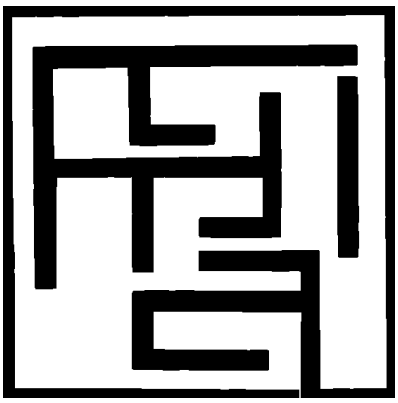

Figura 5: Mapa processado

4. Algoritmo de Trajetória

4.1. Escolha do Algoritmo

Existem diversos algoritmos para criação de trajetórias, porém como já possuímos a imagem do mapa e queremos que sempre encontre o caminho mais otimizado sempre, evitamos a utilização de algoritmos que utilizam sistemas aleatórios, como o *Random Tree*.

Sendo assim, foi utilizado o algoritmo *Wavefront*. A explicação do algoritmo de forma detalhada está descrita a seguir.

4.2. Algoritmo *Wavefront*

A execução do algoritmo *wavefront* é dada pela propagação de "ondas"do ponto final, de modo a alcançar o ponto inicial (X_0, Y_0). Para isso, o algoritmo segue os seguintes passos:

1. (Para a primeira iteração) Localiza o ponto final (X_f, Y_f)
2. Realiza a vizinhança do(s) ponto(s), com o *8-way connection*:
   $(X_i + 1, Y_i)$; $(X_i - 1, Y_i)$; $(X_i, Y_i + 1)$; $(X_i, Y_i - 1)$;
   $(X_i + 1, Y_i + 1)$; $(X_i - 1, Y_i + 1)$; $(X_i + 1, Y_i - 1)$; $(X_i - 1, Y_i - 1)$;
3. Analisa se a vizinhança é caminho ou obstáculo, e se for caminho, coloca o "custo"ds posição como i.
4. Seleciona os pixels com valor i e volta no passso 2, somando 1 ao valor de i.

5. O algoritmo para ao realizar todo o mapeamento da imagem.

Após possuir a matriz de pixels com o custo de todas as posições a partir do final, realiza-se o seguinte algorítmo para obter o caminho necessário:

1. Localiza o ponto inicial (X_0, Y_0).
2. A partir do ponto, pega a posição vizinha com menor custo e atualiza a nova posição para essa.
3. Salva-se em um vetor a nova posição.
4. Realiza-se os passos até que a posição dada seja a mesma que a posição final dada

   Com o vetor de caminhos, temos a trajetória otimizada do caminho.

4.3. Caminho gerado

Após realizar a implementação do algoritmo no mapa gerado, obtemos o seguinte caminho em azul na figura abaixo:

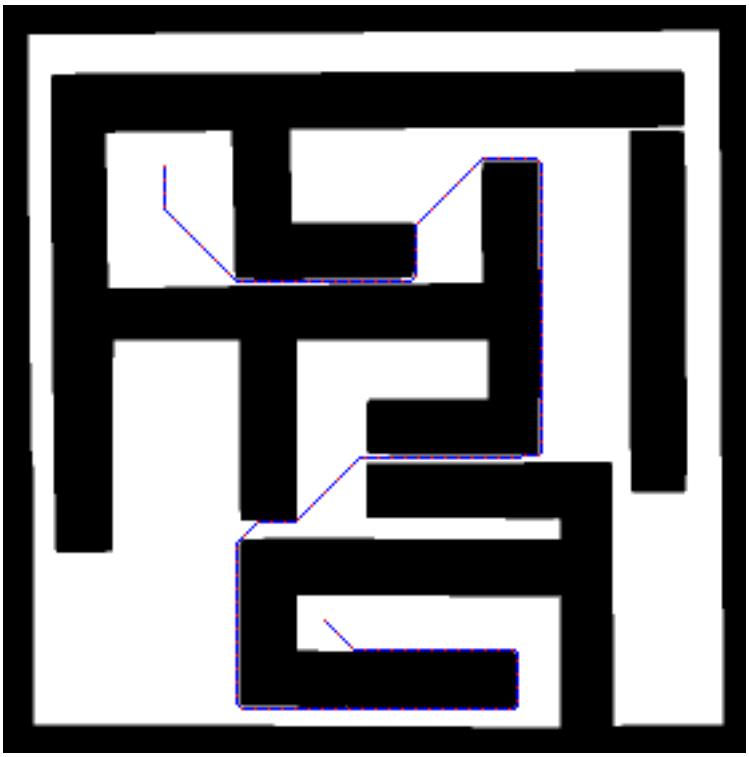

Figura 6: Caminho pelo algoritmo

5. Execução da trajetória

5.1. Conversão de medidas

Após a obtenção do caminho a partir dos pixels da imagem do mapa, é necessário realizar a conversão de cada posição no vetor de caminho a conversão para a posição do mapa no Gazebo. Utilizando as informações de medidas do mapa, de resolução do mapeamento e estimando a coordenada do vértice inferior esquerdo como (1, 1), obtemos a seguinte relação:

$$\begin{cases} X_{gazebo} = X_{pixel} * 0.1 \ + \ 1 \\ Y_{gazebo} = 29.186 - Y_{pixel} * 0.1 \ + \ 1 \end{cases}$$

5.2. Trajetória para o robô

A posição do robô é dada pelo pacote *amcl*, utilizando dados do sensor do *Laser Scan* com um filtro de partículas. Logo, é necessário ajustar os parâmetros do controlador e não realizar movimentos bruscos ou colisões para que o movimento não prejudique a odometria.

Após realizar a conversão de todos as posições do vetor de caminho para o Gazebo, cria-se um novo vetor, pegando uma ponto a cada 5 do inteiro, onde são essas posições que são mandadas para o controlador do robô. Essa discretização ocorre para possibilitar movimentos mais contínuos ao robô sem prejudicar a trajetória.

O robô irá se mover baseado em um controlador proporcional de posição. O erro a ser minimizado pelo controlador é baseado na diferença entre o ângulo do robô e o ângulo formando entre a posição do robô e a posição alvo. As seguintes equações descrevem o controlador:

$$\theta_t = \arctan2(y_t - y_r, x_t - x_r)$$

$$e = \theta_r - \theta_t$$

sendo θ_t o ângulo entre o robô e o alvo, θ_r o ângulo do robô, (x_r, y_r) a posição do robô e (x_r, y_r) a posição do alvo.

A velocidade linear v é definida em função do erro da forma:

$$\begin{cases} v = 0.5, \text{ se } e < 5^\circ \\ v = 0, \text{ se } e > 5^\circ \end{cases}$$

Já a velocidade angular ω é definida em função do erro da forma:

$$\begin{cases} \omega = 0, \text{ se } e < 5^\circ \\ \omega = K_p * \sin(e), \text{ se } e > 5^\circ \end{cases}$$

A troca de alvo para o robô chegar acontece acerca desse vetor criado. O fluxo de informações desse vetor para o robô é dado da seguinte maneira:

1. Adquira a i-ésima posição alvo do vetor do caminho.
2. Ao chegar numa distância menor que um limiar α (por exemplo, $\alpha = 0.2$) dessa posição, i é atualizado para i+1.

6. Conclusão

A partir do controle do movimento do robô, foi gerado um log de informações, mostrando a posição atual do robô, a posição do ponto a ser atingido, o erro angular e a distância restante, como mostrado abaixo:

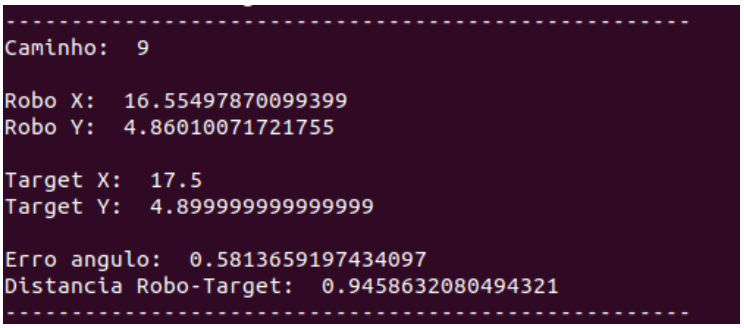

Figura 7: Terminal da execução

Ao executar o programa no Gazebo, o robô já inicia em direção ao primeiro ponto, e realizando diversos testes. O controlador foi satisfatório e robô consegue atingir a posição final sem atingir nenhum obstáculo. A figura abaixo mostra o robô no meio do caminho:

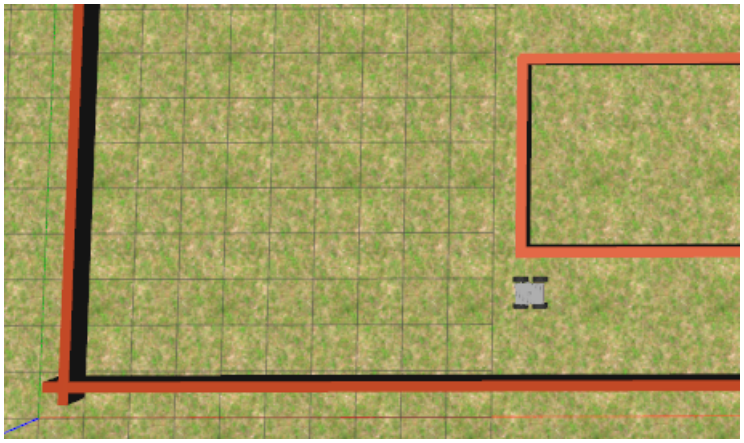

Figura 8: Robô realizando a trajetória

Para acessar o vídeo do robô realizando a trajetória, clique aqui.

Portanto, foi possível concluir que é possível realizar um planejamento de trajetória em função de um mapeamento pré-realizado. Uma das principais dificuldades possíveis de citar é a preocupação com a postura do robô, tendo em vista que uma odometria imprecisa gera resultados de controle insatisfatórios e o robô não consegue seguir fielmente sua trajetória definida. Com a adição do Filtro de Particulas, a odometria foi melhorada considerávelmente e a tarefa de localização foi facilmente resolvida.