Trabalho prático 2-Robótica Móvel

Glaucus Miranda de Almeida e João Vitor Tavares de Almeida

Matrículas: 2021032986, 2017114124

Introdução:

Dois dos conceitos fundamentais em robótica móvel são o planejamento de caminho e controle do robô. O controle refere-se a fornecer as velocidades a um robô de forma que ele encontre um caminho determinado. O planejamento de caminhos refere-se a tarefa de encontrar um caminho que leve um robô de um ponto inicial a uma posição final desejada. Além disso, o controlador pode ser realimentado com as informações já coletadas do ambiente, o que chamamos de malha fechada. Neste trabalho foi pedido que implementarmos duas formas de planejamento de caminho, sendo uma um roadmap a escolha para um robô holonômico e a outra forma o algoritmo de campos potenciais para um robô diferencial.

1.1 Roadmaps e Robôs holonômicos

A pose do robô é definida como o grupo de variáveis que definem a orientação e posição dele no espaço. O número de variáveis que compõem a pose é chamado de grau de liberdade(DF-Degrees of freedom em inglês). Um robô que consegue atuar em todas as variáveis de sua pose de maneira independente entre si, ele é chamado de Holonômico. Neste trabalho foi escolhido o roadmap de decomposição em células(Grid) para o planejamento de caminhos de robôs holonômicos. O método funciona da seguinte maneira:

- **Decomposição em células(Grid)**
  São criadas células fixas para o espaço independente dos obstáculos.
  Uma célula que tenha pelo menos parte de um obstáculo é considerada totalmente ocupada. Um robô não ocupa o espaço de uma célula ocupada.

1.2 Campos potenciais e Robôs diferenciais

Ao contrário dos robôs holonômicos, os robôs diferenciais não conseguem atuar em todos os elementos da sua pose de forma independente. Por isso, para esse trabalho adota-se outra forma de planejamento de caminhos, e de controle. O controlador escolhido para realizar o planejamento de campos potenciais foi o seguinte:

$$v = k_\rho(\dot{x}\cos\theta + \dot{y}\sin\theta)$$
$$\omega = k_\theta(\text{atan2}(\dot{y}, \dot{x}) - \theta)$$

Onde os valores v e **ω** são as velocidades lineares. Onde **ẋ** e **ẏ** são velocidades lineares em x, y são as velocidades lineares calculadas pelo algoritmo de campos potenciais. O algoritmo pode ser comparado à física e o que ocorre com partículas de sinais opostos, de magnetismo, por exemplo. Dessa forma, o goal atua exercendo uma força de atração no

robô, que por sua vez sofre repulsão pela força gerada pelos obstáculos ao seu redor. Essas forças geram uma resultante que guia o robô no ambiente. O algoritmo é o seguinte:

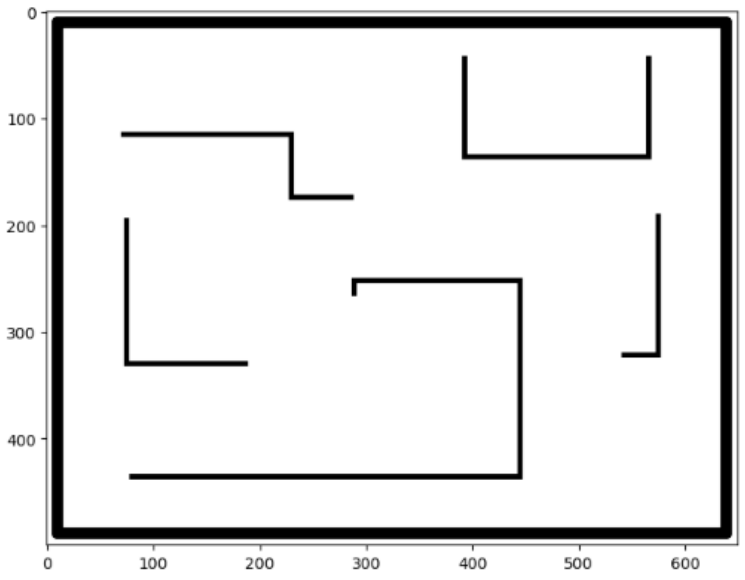
- **Algoritmo de campos potenciais**
  Defini-se a posição do robô
  Enquanto o robô não atingiu o alvo:
  *Calcula-se a força de atração ao goal
  *Calcula-se as forças de repulsão dos obstáculos ao redor do robô.
  *Calcula-se a força resultante das forças de atração e repulsão.
  *Separa-se as componentes fx e fy da força resultante que atuam no eixo x e y.(Variáveis **ẋ** e **ẏ)**
  ***fornece-se ao robô suas velocidades com base nessas forças

Implementação

Os algoritmos foram implementados utilizando a linguagem python e o programa CoppeliaSim. Em ambos os algoritmos, utilizamos funções de controle disponibilizadas nas aulas com modificações para que se seguisse o planejamento desejado.

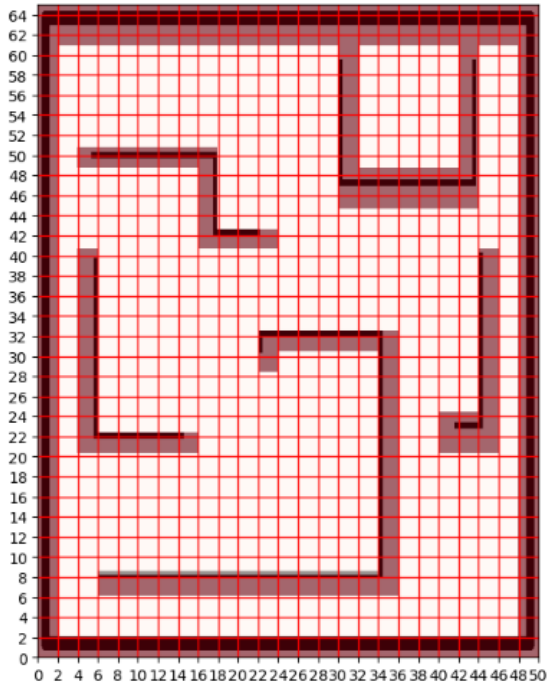
2.1 Roadmaps

Primeiramente fazemos um pequeno código para plotarmos nosso mapa, carregando uma imagem e transformando ela em valores binários para preto e branco para facilitar futuras etapas. visto abaixo com o mapa paredes:

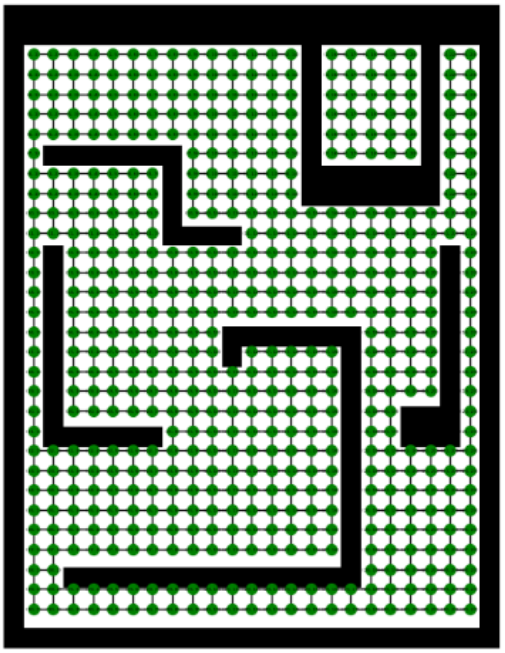


Após termos o plot feito do mapa, o dividimos em células, considerando uma célula parcialmente ocupada como totalmente ocupada, para decidir se uma célula está ocupada é

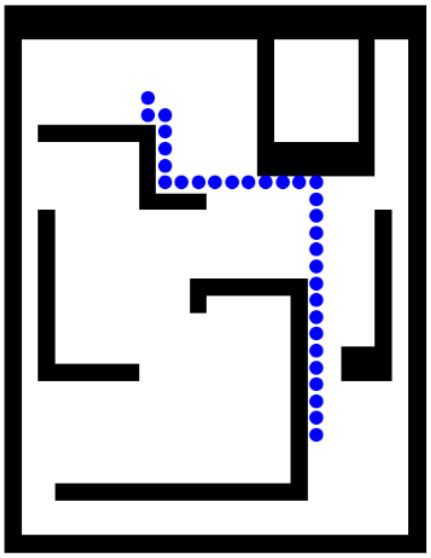
feito o somatório dos valores dos píxeis, caso seja maior do que o limite que colocamos como 0.5, ela é considerada ocupada, caso contrário livre.



Após isso é gerado um grafo com os pontos do grid que não foram preenchidos para podermos ditar onde o nosso robô pode andar pelo mapa, isso é feito ao preencher o mapa com o grafo em cada vértice das células e depois subtrair as que foram ditas como ocupadas.



Usando a função nx.shortest path, podemos encontrar o menor caminho entre o ponto que estará nosso robô, que pode ser pega pelo mapa em si e o ponto onde estará o nosso ponto final. O resultado no nosso caso ficou o seguinte:



Após isso, além de usarmos o programa para controle do robotino disponibilizado durante as aulas, utilizamos, a lógica de que a cada ponto do grafo, andaríamos 2 metros na próxima direção, por este ser o tamanho do pixel para a simulação, considerando que nosso robô é holonômico, ou seja consegue atuar em qualquer um de seus eixos de liberdade.

```
# Com a config atual, andar 2m a cada nodo que irei passar, considerando liberdade de movimento em 4 eixos N/E/S/W
# if path[0] increases, robot walks South, otherwise North
# if path[1] increases, robot walks West, otherwise East
lastNode = (path[0][0], path[0][1])
for y, x in path:
    currNode = (y, x)
    direction = tuple(map(lambda i, j: i - j, currNode, lastNode))
    print(direction)
    lastNode = (y, x)
```

2.2 Campos potenciais

Como se trata de um programa reativo e não deliberativo, não é feito um plano de caminho, mas apenas alimentado uma lógica para o robô seguir. Nesse caso o robô que estamos utilizando é o Pioneer, que é um robô diferencial, ou seja não consegue atuar em todos os seus eixos de liberdade. A lógica de campos potenciais é a de que obstáculos geram uma força de repulsão e o goal uma força de atração. Essas forças geram uma resultante que irá ditar para onde nosso robô irá se mover em direção para.

Para calcularmos essas forças foram feitas duas funções:

```
def att_force(q, goal, katt=.2):
    return katt*(goal - q)

def rep_force(q, hokuyo, laser_data, max_sensor_range=5, R=2, krep=.01):
    repAdd = 0
    d = 1

    rotationWL = Rz(hokuyo[3])
    translationWL = [hokuyo[0], hokuyo[1], hokuyo[2]]
    transformWL = np.column_stack((rotationWL, translationWL))
    aux = np.array([0, 0, 0, 1])
    transformWL = np.row_stack((transformWL, aux))

    for i in range(len(laser_data)):
        ang, dist = laser_data[i]
        if (max_sensor_range - dist) > 0.1:
            x = dist * np.cos(ang)
            y = dist * np.sin(ang)

            objCoordsT = transformWL @ np.array([x, y, 0, 1])

            v = q[:2] - [objCoordsT[0], objCoordsT[1]]
            d = np.linalg.norm(v, axis=0)
            d = d.reshape((len(v)-1, 1))
            rep = (1/d**2)*((1/d)-(1/R))*(v/d)
            rep_x = rep[:, 0]
            rep_y = rep[:, 1]

#             fig = plt.figure(figsize=(8,8), dpi=100)
#             ax = fig.add_subplot(111, aspect='equal')
#             ax.plot(q[0], q[1],  'k>', markersize=10)
#             ax.quiver(q[0]+dist, q[1]+dist, rep_x, rep_y, color='g')
            repAdd += rep

#     invalid = np.squeeze(d > R)
#     repAdd[invalid, :] = 0

    return krep*repAdd
```

Para podermos perceber os objetos foi utilizado o sensor Fast Hokuyo acoplado ao Pioneer. A partir da configuração do robô é calculada a força de atração, nós utilizamos o sensor laser para que seja possível calcular a força de repulsão a partir da configuração do sensor. Tendo esses valores, nós podemos fazer o cálculo da resultante, o que nos dá a direção para qual o robô deve ir, com isso podemos dar as velocidades com as quais ele deve atuar em cada roda nesse caso.

```
# Gets attractive force, now needs to get laser data and calculate
# Repulsive force and add the two
Fatt = att_force(robotConfig[:2], pgoal)
Fatt_x = Fatt[0]
Fatt_y = Fatt[1]

raw_range_data, raw_angle_data = readSensorData(clientID, laser_range_data, laser_angle_data)
laser_data = np.array([raw_angle_data, raw_range_data]).T

Frep = rep_force(robotConfig, hokuyoConfig, laser_data)

Ft = Fatt + Frep

Ft_x = Ft[:,0]
Ft_y = Ft[:,1]
```

Testes

Os testes feitos são mostrados no vídeo, foram feitos dois testes para cada metodologia, cada um em um mapa diferente. Podemos perceber que por conta de o robô andar nodo

por nodo do grafo, o processo de roadmaps é um pouco demorado em comparação ao de campos potenciais, mas nossa implementação de campos potenciais é imprecisa no seu caminho até o goal, mostrando que ambos têm suas vantagens e desvantagens. Apesar das dificuldades, em todos os testes o goal foi encontrado.
Teste 1 de Roadmaps: Duração: ~1 minuto
Teste 2 de Roadmaps: Duração: ~1 minuto e 12 segundos
Teste 1 de Campos Potenciais: Duração: ~25 segundos
Teste 2 de Campos Potenciais: Duração: ~31 segundos

Conclusão:

Neste trabalho foram implementados dois algoritmos de planejamento de caminhos: Roadmap de grid de decomposição em células para robôs holonômicos e Campos Potenciais para robôs diferenciais. Além do planejamento de caminho foi necessária a implementação do controle para que conseguissem navegar nos caminhos gerados.

Uma de nossas maiores dificuldades foi conseguir a força de repulsão de maneira correta, pois para conseguirmos ela é necessário levar em conta as configurações de nosso sensor, além das configurações de nosso robô.

Também tivemos uma pequena dificuldade pelo fato de a escala do mapa no programa em python e no CoppeliaSim nem sempre serem iguais.

De maneira geral, o trabalho consegue nos mostrar como diferentes formas de implementação de caminhos funcionam e que elas têm suas vantagens e desvantagens, como por exemplo o Roadmap necessitando que se conheça o mapa por completo e o algoritmo de campos potenciais não, mas o Roadmap sempre encontra o caminho até o goal, enquanto o algoritmo de campos potenciais pode sofrer com o problema de mínimos locais.