Relatório MicroRato

António Domingues[89007] & Henrique Silva[88857]

Universidade de Aveiro
Departamento de Eletrónica Telecomunicações e Informática

Abstract. O presente relatório tem como objetivo descrever a metodologia utilizada na resolução dos challenges propostos na cadeira de Robótica Móvel e Inteligente, no âmbito do projeto Micro Rato. O relatório está dividido em três capítulos principais, correspondentes aos três challenges. Cada capítulo possui subseções que explicam em maior detalhe o código implementado, complementadas com excertos do código. No fim de cada capítulo são tiradas conclusões dos resultados obtidos. Após o último capítulo é ainda apresentada a divisão de percentagens por elemento do grupo.

1 Desafio C1

1.1 Implementação

Como descrito no desafio proposto, o objetivo do C1, é desenvolver um agente que tenha a capacidade de percorrer um determinado labirinto, outrora desconhecido, no menor tempo e com o menor número de colisões possíveis.

Para tal, surgiu a necessidade de desenvolver e implementar o programa *mainrobc1.py*, a fim de movimentar o robot ao longo do percurso, sem que o mesmo batesse nas paredes. Com isso em mente foram efetuados cálculos prévios, visto que os sensores laterais se encontram a +-60º do sensor da frente do agente. Como a distância entre paredes horizontais é de cerca de 1.8 diâmetros, sabe-se que se o robot estiver no centro da célula, a distância dos sensores laterais (caso estejam a 90º), é de cerca 0.4 diâmetros. Com o uso trigonometria para descobrir a distância a que os sensores laterais se encontram das paredes, chegou-se à seguinte conclusão: 0.9-sin60º=0.46. Como o sensor mede 1/distância, foi concluído que a distância proveniente do robot à parede é 2.17.

Fazendo uma abordagem mais técnica em termos de código, o robot segue em frente sempre que não se encontra demasiado perto de uma parede esquerda ou direita, assim como de uma parede à sua frente. O código referente a esse movimento é apresentado na figura abaixo.

```
if self.measures.irSensor[left_id]>= 2.9:
    self.driveMotors(0.15, 0.097)
elif self.measures.irSensor[right_id]>= 2.9:
    self.driveMotors(0.097, 0.15)
else:
    self.driveMotors(0.15, 0.15)
    print("vai em frente")

if self.measures.irSensor[center_id]>=1.16:

    print("stop")
```

Para o caso de se encontrar a uma distância inferior a 1/2.9 diâmetros de uma parede esquerda, é aplicada uma potência superior na roda esquerda, para ajustar a sua posição. Após esse ajuste, o robot desloca-se para o sentido oposto ao da parede, evitando colisões laterais. O mesmo se aplica para o caso de se encontrar demasiado perto de uma parede direita. A diferença é que nesse caso é necessário aplicar mais potência no motor da roda direita. O código referente a este movimento é apresentado de seguida:

```
if self.measures.irSensor[left_id]>= 2.9:
        self.driveMotors(0.15, 0.097)
elif self.measures.irSensor[right_id]>= 2.9:
        self.driveMotors(0.097, 0.15)
```

No caso de surgir uma parede frontal, o agente começa a fazer sucessivas rotações para corrigir a sua posição até que lhe seja permitido ir novamente em frente.

Das expressões dadas relativas ao movimento do objeto, pode concluir-se a seguinte expressão; rot = R-L/D em que R= potência aplicada na roda direita; L = potência aplicada na roda esquerda e D=diâmetro do robot. Assim sabe-se que dada uma determinada potência às rodas, o robot faz uma rotação de radianos(rot).

1.2 Conclusões e aspetos a melhorar

Os objetivos neste desafio foram cumpridos na sua totalidade pois o agente faz quase todo o percurso na máxima velocidade. No entanto as 10 voltas propostas no enunciado não são feitas e isto deve-se ao facto de o agente não cortar as curvas e não as fazer num movimento constante. Ao invés, ele trava antes das curvas, o que acabou por diminuir o número de voltas dadas para aproximadamente 8.

2 Desafio C2

2.1 Movimentação no mapa

Neste desafio são disponibilizadas mais algumas medidas, nomeadamente o GPS e as coordenadas da bússola. Desta forma, o movimento do agente passa a ser controlado por estes valores e não pelos valores dos sensores. Assim, o robot apenas necessita de utilizar os sensores, para fazer o mapeamento das paredes do mapa. Consequentemente os sensores laterais foram colocados a 90º para determinar com maior precisão a existência de paredes.

A abordagem escolhida foi parar de célula a célula e só aí fazer o mapeamento do que se encontra em redor do robot. Para se implementar este deslocamento, foram salvaguardadas as coordenadas da posição da célula anterior através da variável *previous_gps*.

```
self.previousGps = [self.measures.x - self.firstPosX, self.measures.y - self.firstPosY]
```

Cada vez que o módulo da diferença entre as coordenadas atuais (em x para o deslocamento horizontal, em y para o deslocamento vertical) for igual a 2 (centro de célula em célula é igual a 2 diâmetros) o robot tem de parar para tirar as conclusões.

Nesta situação a inércia para o robot parar revelou-se um obstáculo. Visto que a velocidade linear é igual a 0.15 (lin = (L+R)/2), para o robot parar exatamente no centro da célula é necessário que o lin seja = a -0.15 (L=-0.15, R=-0.15).

Para fazer rotações de 90º à esquerda ou à direita, aplicam-se as seguintes expressoẽs:

- No máximo o robot vira rad(-0.15- 0.15) (pelas expressões de rotação), aproximadamente 17.8º por ciclo.

- Para rodar os 90º, são necessários pelo menos 5,05 ciclos, ou seja, 6 ciclos. Para rodar Π/2 * 0.5; lin = Π/2 * 0.% = 0.131. Ou seja, para virar para um dos lados 90º, é necessário aplicar 0.131 de potência num dos motores das rodas e -0.131 no outro motor durante os 6 ciclos (foi tido em conta a inércia nas questões de código).

- Se este se encontrar num beco (tiver que fazer inversão de marcha) a abordagem é semelhante. Apenas difere de ter de rodar durante 16 ciclos para fazer uma rotação de 180º.

Visto que as rodas sofrem ruído, o agente acerta a sua orientação conforme a bússola. Se faz pequenos desvios, (compara a sua orientação com a bússola) e se esses diferem em 2º, aplica potência nas rodas corrigindo a sua orientação. Caso contrário, segue o seu caminho.

O robot tenta ao máximo mover-se para uma célula que ainda não tenha sido visitada, com o propósito de mapear o mapa a 100%. Quando o mesmo não tem nenhuma posição que ainda não tenha sido visitada à sua volta, vagueia, até encontrar uma posição não visitada.

```
if (self.measures.compass >= 86 and self.measures.compass <90) or (self.measures.compass<-90 and self.measures.compass>=-94):
    self.driveMotors(0.13, 0.15)
elif (self.measures.compass <=94 and self.measures.compass > 90) or (self.measures.compass<= -86 and self.measures.compass>-90):
    self.driveMotors(0.15, 0.13)
else:
    self.driveMotors(0.15, 0.15)
```

2.2 Armazenamento das coordenadas

Depois de feito o mapeamento do mapa, realizou-se o armazenamento das coordenadas das células por onde o agente passa. Foi criado um array bidimensional com 55 colunas e 27 linhas, e todas as posições do array inicializadas a " ".

```
        self.coordinates = [[self.foo for x in range(55)] for y in range(27)]
```

A posição onde o agente se encontra corresponde a uma célula vazia e portanto o seu valor é sempre colocado a "X" no array bidimensional.

```
        self.coordinates[round(self.posY)][round(self.posX)] = "X"
```

Sempre que o agente pára e faz o mapeamento do seu redor, avalia a direção para onde se encontra (norte, sul, este ou oeste). Isto é feito de acordo com os valores da bússola. De seguida, são usados os valores dos sensores para sondar as posições à volta do robot e saber se são paredes ou espaços vazios. De acordo com os valores dos sensores e com a direção para onde está virado o robot, as células adjacentes ao mesmo são mapeadas e é colocado, na posição do array onde o robot se encontra, o símbolo correspondente à informação adquirida: parede vertical (|), horizontal (-) ou célula vazia (x). Este processo é feito através do uso de condições semelhantes à seguinte:

```
if self.measures.compass <= 10 and self.measures.compass >= -10:
        if self.measures.irSensor[center_id] > 1.2:
             self.coordinates[round(self.posY)][round(self.posX+1)] = "|"
```

No caso apresentado acima, o robot encontra-se na horizontal virado para a direita (Norte) pois os valores da bússola estão entre 10 e -10. A segunda condição

permite avaliar que o agente tem parede à frente e portanto a posição seguinte à que o robot se encontra (no eixo X, por estar na horizontal) fica preenchida no array com um "|".

O array bidimensional é impresso para o ficheiro *map.out* através do seguinte trecho de código, onde são criados dois ciclos que percorrem as colunas de cada linha e imprimem a posição correspondente no ficheiro.

```
with open('map.out', 'w') as outfile:
    for i in range(27):
        for j in range(55):
            outfile.write(self.coordinates[i][j])
            #print(self.coordinates[i][j])
        outfile.write("\n")
```

2.3 Conclusões e aspetos a melhorar

O resultado obtido foi positivo, pois na grande maioria das vezes em que o código é executado, o agente é capaz de mapear todo o percurso. Mesmo quando o mapa não é percorrido na sua totalidade, a maior parte das células é visitada.

No entanto, o resultado podia ter sido otimizado com a implementação de um algoritmo de path finding. A otimização passaria por uma redução do tempo necessário para visitar todas as células e pela garantia de que todas as posições seriam sempre mapeadas.

3 Desafio C3

3.1 Modelo de movimento

Sendo o objetivo deste desafio, descobrir *checkpoints* e o caminho mais curto até eles, o movimento base é o mesmo em relação ao desafio anterior. Neste caso o essencial é mapear as posições já visitadas, para ter em posse todas as coordenadas das paredes do mapa em questão.

3.2 Descoberta de beacons e caminhos mais curtos

Para começar, todas as posições do array *self.coordinates* são inicializadas como paredes (-). Por conseguinte, cada vez que o robot marca um "X" de posição visitada no array, o array referido vai ganhando a forma de espaços visitados, assim como os muros. Este passo é crucial, pois é usado um mecanismo de pesquisa, que necessita de todos os muros do mapa. A criação deste array é feita da seguinte forma:

```
for i in range(len(self.coordinates)):#LINHAS
    for j in range(len(self.coordinates[i])):
        if str(self.coordinates[i][j]) == "-":
            self.walls.append((j, i))
```

A atualização deste array que possui as coordenadas de cada parede, é feita a cada 10 deslocamentos do robot (1 deslocamento igual a deslocamento do centro de célula a célula). Posteriormente, foi usado um mecanismo de pesquisa, denominado de A*, que dada uma posição inicial e uma posição final, assim como o array de paredes, calcula o caminho mais curto entre esses dois pontos. O código foi implementado para que cada vez que vai conhecendo mais posições, seja atualizado o array de paredes e assim a probabilidade de encontrar um caminho mais curto entre *beacons* aumente. Cada beacon é guardado num array (*"*self.objetivosPassados= [(27, 13)]*"*), que é incrementado com a posição de cada beacon, assim que o *self.measures.ground*, detete que está sobre um *checkpoint*.

Por fim, é invocado o A* tantas vezes como o número de objetivos, ou seja: inicio-->beacon1-->beacon2--->inicio.

```
#executa o A* e imprime no ficheiro
for b in range(len(self.objetivosPassados)):
    if b< len(self.objetivosPassados)-1:

        a = pf.AStar()
        a.init_grid(56, 28, self.walls, self.objetivosPassados[b], self.objetivosPassados[b+1])
        path = a.solve()
        solucao.append(path)
        #print(path)

    else: #para imprimir da ultima posiçaõ adicionada aos checkpoints até ao inicio
        a = pf.AStar()
        a.init_grid(56, 28, self.walls, self.objetivosPassados[b], (27, 13))
        path = a.solve()
        solucao.append(path)

print("Solução" + str(solucao))
```

Após isso é impresso num ficheiro o path encontrado pelas diversas pesquisas, garantindo que, com o caminho visitado, o percurso mais curto entre esses 3 *checkpoints* é o apresentado no ficheiro.

O código fonte para utilização do A* segue em anexo na bibliografia.

3.3 Conclusões e aspetos a melhorar

O trabalho realizado para o challenge 3 teve desde o início um objetivo bem definido: conseguir encontrar o caminho mais curto até um ponto, através do uso do algoritmo de pesquisa A*. A estratégia para alcançar esse objetivo foi pensada e aplicada, mas por motivos que acabaram por não ser compreendidos, apenas resultou no cálculo do primeiro caminho (ponto de partida - beacon).

Desta forma, apesar do esforço investido o objetivo principal acabou por não ser alcançado na sua totalidade.

Percentagem de trabalho:

Ambos os elementos do grupo cumpriram com as tarefas individuais e ajudaram nas coletivas. Deste modo a percentagem de trabalho foi de:

António Domingues -> 50%

Henrique Silva -> 50%

Bibliografia

1. Código fonte do algoritmo A*
   https://github.com/laurentluce/python-algorithms/blob/master/algorithms/a_star_path_finding.py