

RMI - Assignment 2

João Génio (88771) and Ruben Menino (89185)

¹ Departamento de Eletrónica, Telecomunicações e Informática

² Universidade de Aveiro

joaogenio@ua.pt | ruben.menino@ua.pt

³ Robótica Móvel e Inteligente

1 Introdução

Neste relatório são apresentadas as abordagens para a resolução das três tarefas propostas, assim como os resultados obtidos, as conclusões tiradas e algumas melhorias/trabalho futuro que poderão ser realizados.

Inicialmente foi-nos proposto controlar o movimento de um robô e conseguir localizar-se através de um circuito fechado desconhecido tentando não colidir com as paredes. Foi necessário para isso utilizar as fórmulas do movimento, a distância aos obstáculos a partir dos sensores e também a bússola. Nestas tarefas, o GPS não estava acessível, e tanto os motores, como as distância dos sensores e a bússola têm ruído.

Para a segunda tarefa, foi necessário mapear o circuito todo (todas as células navegáveis), passando por todas as células, e à medida que passava pelas células conseguir localizar os beacons localizados no circuito. Foi implementada uma máquina de estados, utilizando também o algoritmo de Dijkstra que irá ser explicado posteriormente no decorrer no relatório. Após completar o mapeamento de todas as células navegáveis, o robô dirige-se para a posição inicial, que pertence à lista de beacons (índice 0).

Como última tarefa foi utilizado também o algoritmo de Dijkstra para conseguir descobrir o melhor caminho para percorrer os beacons. As tarefas no geral foram cumpridas, conseguindo o que era necessário para cada um deles.

Obteve-se assim para o mapa default uma pontuação de 3130 do mapeamento do mapa, e um score de 1 quanto ao planeamento.

Para completar todo o mapa, fazendo o mapeamento de todas as células navegáveis e o seu planeamento o agente demorou cerca de 9000 ticks.

1.1 Mapa e Robô

O ambiente de simulação CiberRato vai ser utilizado para avaliar os vários agentes desenvolvidos nas três tarefas.

Este robô é composto por 2 motores (esquerda e direita), 3 leds (visitar, regressar e finalizar), GPS, bússola, 1 sensor de beacons, 1 sensor de colisão e 1 sensor do solo.

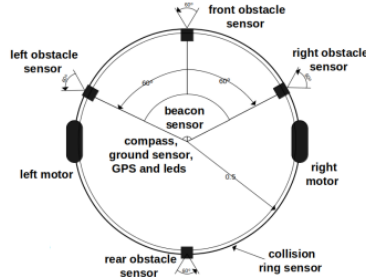


Fig. 1. Robô

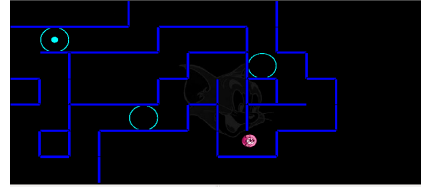


Fig. 2. Mapa

O mapa é visto como uma matriz bidimensional de células de tamanho fixo. Cada célula é um quadrado com comprimento lateral duas vezes o diâmetro do robô. As células podem ser referenciadas pela sua posição no circuito onde a célula (0,0) é a origem.

2 Localização

2.1 Cálculo da estimativa de posição

Nesta tarefa, como não temos acesso ao sensor GPS, o robô não sabe onde está posicionado no circuito. Por isso para o movimento do robô, foram utilizadas várias fórmulas de movimento, a distância dos sensores aos obstáculos e também a bússola. Visto que a pose do robô é dada como (x, y, θ) onde x e y definem a posição do robô e θ a orientação dele, são utilizadas várias fórmulas para conseguir obter uma estimativa da nova posição do robô no próximo tick. Inicialmente, para atualizar as variáveis de localização e orientação, e tendo as variáveis

`in_l` (motor left) e `in_r` (motor right) com o valor do tick atual, atualiza-se esses mesmos valores com as seguintes fórmulas.

$$self.out_l = \frac{self.in_l + self.out_l}{2}$$

$$self.out_r = \frac{self.in_r + self.out_r}{2}$$

Estes são os valores (estimados) da potência aplicada em cada motor. Após calcular esses valores, a nova posição do agente, com a sua nova orientação, dá-se como:

$$self.xt = self.xt + self.lin * \mathit{math.cos}(self.theta2) \quad (1)$$

$$self.yt = self.yt + self.lin * \mathit{math.sin}(self.theta2) \quad (2)$$

onde:

$$self.lin = \frac{self.out_l + self.out_r}{2} \quad (3)$$

De seguida, para a rotação do robô é atualizado também o valor da rotação, com os valores `self.out_l` e `self.out_r` atualizados

$$self.theta2 = \frac{self.measures.compass * \pi}{180} \quad (4)$$

2.2 Calibração da estimativa

Usando os sensores do robô, podemos estar mais seguros quanto à nossa estimativa de posição.

Quando o robô se encontra no centro de uma célula, analisa os sensores à sua volta para mapear as paredes. Se existir uma parede à sua frente, o valor medido no sensor é usado para ajustar a sua posição relativa ao eixo dos x ou dos y. Por exemplo, se o robô chega à célula (estimada) 34,14 deslocando-se da esquerda para a direita, e verifica que existe uma parede à sua frente, então ele sabe que o seu "x" estimado é 35 ("x" da parede) - 0.1 (metade da largura da parede) - 0.5 (metade do diâmetro do robô).

De forma semelhante, quando o robô está a navegar de célula para célula, verifica a existência de paredes nas suas laterais. Se encontrar apenas uma parede, usa o mesmo método apresentado em cima. Achamos este método satisfatório, porém, decidimos implementar uma lógica mais complexa para atingir melhores resultados de calibração.

Quando temos duas paredes nas laterais, temos uma oportunidade de usar dois sensores para calibrar a nossa posição. O primeiro passo é obter as distâncias medidas para ambas as paredes. Num cenário em que os sensores não têm erro, a soma das distâncias terá de ser obrigatoriamente 0.8 (caso nenhuma das paredes esteja na fronteira do mapa). Na prática isso raramente acontece, então nós normalizamos as medições para atingir uma soma de 0.8.

Ora se obtivermos 0.5 nas duas medições, a soma será 1, ou seja, reduzimos as medições para 0.4. Vejamos um caso em que os sensores medem valores diferentes. Se o sensor da esquerda medir uma distância de 0.41 e o da direita 0.33, eles serão ajustados para 0.44 e 0.36, respetivamente. Esta lógica assume que, para o sensor da esquerda, a medição 0.41 é menos confiável que a 0.44, pois a segunda é influenciada por outro sensor.

A partir daqui apenas temos de escolher um dos dois valores obtidos para calcular o nosso deslocamento em relação a uma das paredes. Optámos por escolher o valor ajustado do sensor da esquerda e usar a parede da esquerda para obter a nossa estimativa de posição (esta estimativa segue a mesma lógica apresentada anteriormente).

2.3 Movimento

O movimento baseia-se em 3 principais decisões: Se a célula objetivo estiver a mais ou igual a 0.5 diâmetros, tentamos navegar à velocidade máxima. Se o centro da célula estiver a mais de 2° de diferença da nossa direção atual, ajustamos o ângulo de direção ligeiramente (potência baixa). Se estiver a mais de 30° , rodamos o robô completamente (potência alta) até estarmos próximos do ângulo pretendido. Isto garante que o robô navega em direção ao centro de todas as células.

A rotação baseia-se na mesma lógica de ajuste durante o movimento. Normalmente o ângulo objetivo está a cerca de 90° de diferença (quando o objetivo encontra-se à esquerda ou direita do robô), o que significa que temos de rodar completamente até estarmos dentro dos 30° de diferença, para rodar mais devagar até atingir uma diferença de 2° .

Fases principais do movimento

Info \rightarrow Go \rightarrow Try Right or Try Front or Try Left or Turn Around \rightarrow Go

Esta figura indica todos os cenários possíveis para a localização de uma célula objetivo, em relação à atual (4 direções, nunca na diagonal). Calculando um simples ângulo alfa, que irá ser usado no cálculo da direção pretendida. Suponhamos que o nosso objetivo se encontra a sul do robô (esquerda, na orientação do mapa. Exemplo no canto superior direito da figura). Seria intuitivo assumir que temos de rodar o nosso robô até estar perto dos

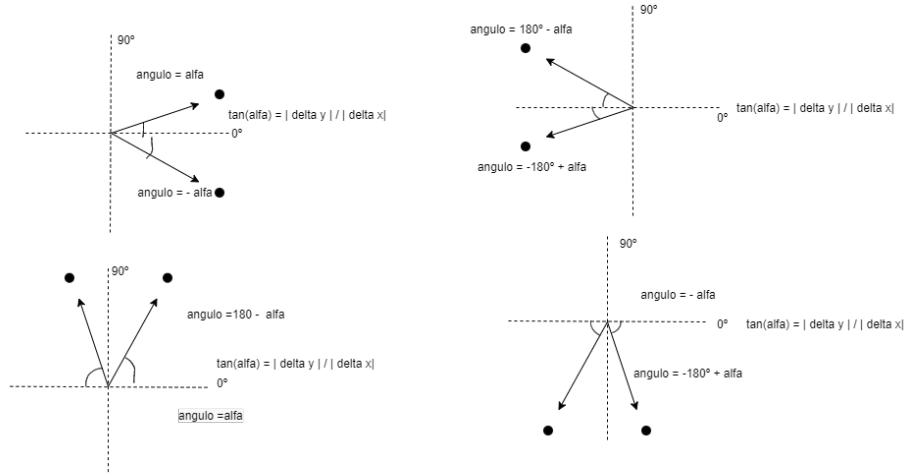


Fig. 3. Cálculo do ângulo de direção pretendido em função da localização da célula objetivo

$180^\circ/-180^\circ$, porém, nem sempre nos encontramos exatamente no centro da célula, devido ao erro introduzido pelos motores. Usando alpha, conseguimos sempre encontrar o ângulo de navegação (bússola) pretendido.

Em adição, quando o robô chega ao centro de uma célula, ajusta a sua orientação para ficar alinhado com um dos dois eixos. Isto evita que o robô, após chegar ao centro de uma célula com uma orientação muito diferente de 0, 90, 180, -180 ou -90, medir incorretamente as distâncias para as paredes. Consequências desse erro seria o mapeamento errado das paredes do desafio.

3 Mapeamento

Para abordar este desafio, decidimos usar uma navegação célula-a-célula. Isto significa que tomamos decisões de mapeamento assim que chegamos ao centro de uma nova célula, e não a cada tick.

Seguindo esta lógica, conseguimos definir dois estados principais: "Info", para indicar o robô que pode efetuar recolhas de informação, e "Go", para indicar o robô que pode avançar para a próxima célula.

Info é a primeira fase que o robô executa. Esta fase consiste em verificar se a célula atual já foi visitada (sub-fase "seen") e decidir a lógica que irá calcular a próxima célula. Caso a célula não tenha sido visitada previamente, verificamos se os sensores laterais (posicionados a 90 e -90°) e o central estão a ler um

valor superior a 1. Se sim, isso significa que estão a detetar uma parede na sua direção. Sabendo que não existe uma parede, podemos concluir que a próxima célula nessa direção é visitável.

Após recolher informação sobre as redondezas, temos de decidir para onde navegar a seguir. O nosso método de decisão inicial é tentar ir para a direita, frente, esquerda ou trás (nesta ordem de preferência). Esta decisão (chamada "RE" no código, mas que iremos referir nesta secção do relatório como "Navegação Standard") é a base que o robô usa para continuar a navegar.

Enquanto o robô está a executar uma lógica de Navegação Standard, guardamos as células percorridas numa lista chamada "circuito". Sempre que o robô encontra uma célula que nunca visitou, adiciona-a ao circuito atual.

Assim que ele visita uma célula que pertence a um circuito anterior, decidimos que o circuito atual está completo e inicializamos um novo (Para evitar uma série de circuitos vazios, apenas criamos um novo circuito, quando o mais recente não está vazio). Vejamos esta lógica em ação através de um exemplo.

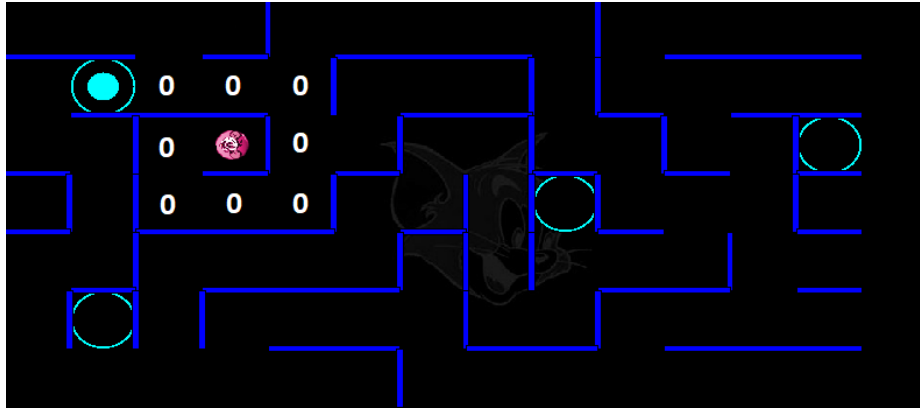


Fig. 4. Exemplo de um circuito a ser navegado

Quando um circuito fica completo, abandonamos a Navegação Standard, e iniciamos uma pesquisa de células visitáveis mas que ainda não foram visitadas (fase de mapeamento "Search"). A estas células chamamos de "candidatos", e, para cada candidato, invocamos uma implementação do algoritmo de Dijkstra para encontrar o caminho mais curto. O candidato que estiver mais perto segundo esse algoritmo, será o nosso objetivo de navegação e indica, obrigatoriamente, que existe um circuito novo por descobrir.

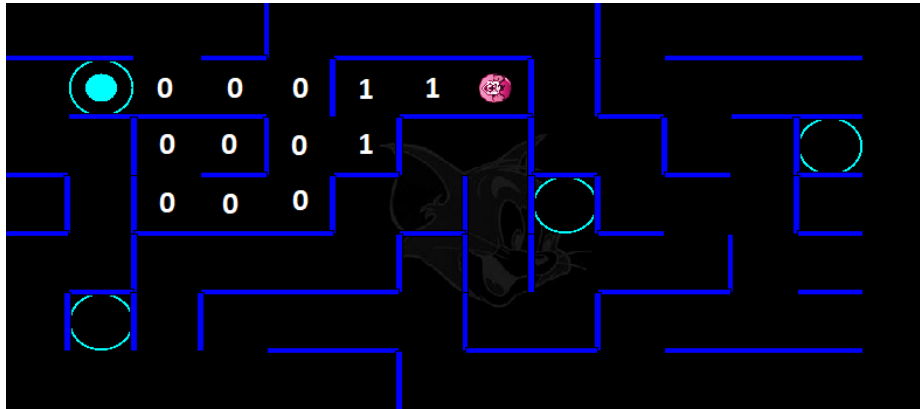


Fig. 5. Exemplo de um segundo circuito a ser navegado

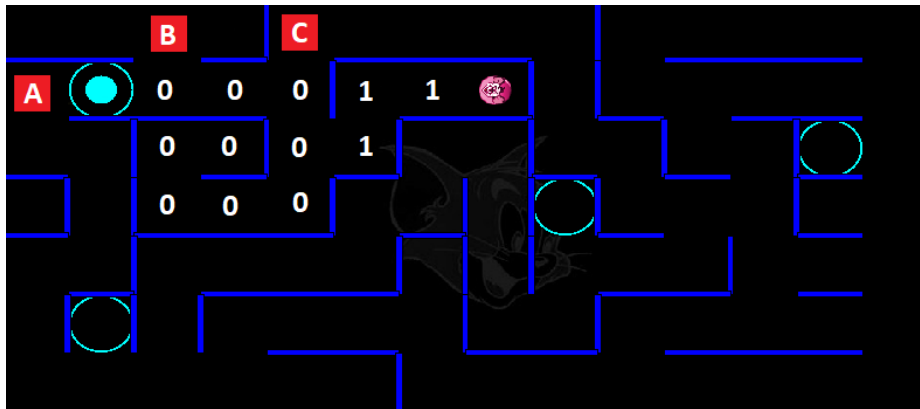


Fig. 6. Exemplo de posições candidatas para a fase de "Follow"

Segue-se a fase de mapeamento "Follow", em que o robô utiliza os seus mecanismos de navegação célula-a-célula para esvaziar uma lista de células, que representa o caminho para o novo circuito. Quando o robô chega ao novo circuito, regressa à Navegação Standard, e vai preenchendo o circuito até visitar uma célula pela segunda vez (ou superior). Quando já não é possível encontrar candidatos, então temos a certeza que o mapa foi todo percorrido. Quando isto é observado, calculamos o caminho para a célula de origem e invocamos a fase "Follow" para navegar até lá e terminar o programa.

3.1 Resultados e Conclusões

Com esta lógica de mapeamento, conseguimos garantir que qualquer desafio será totalmente mapeado. Sabemos que existem limitações nesta abordagem, porém, conseguimos ficar satisfeito com a performance do robô na maior parte

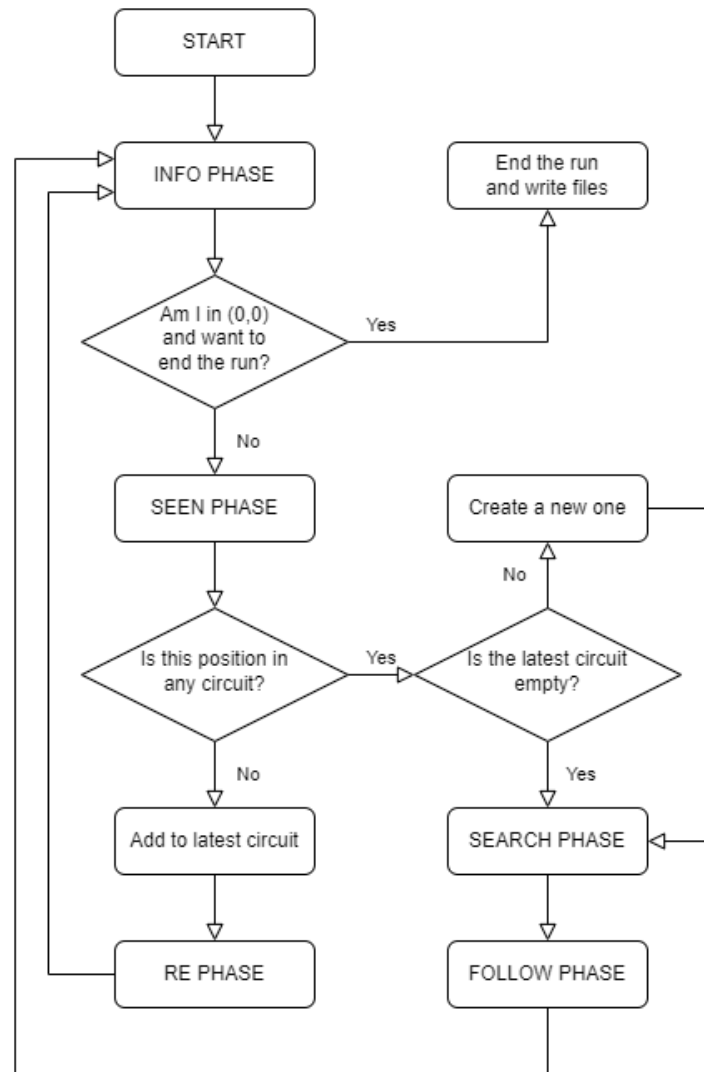


Fig. 7. Flowchart da lógica de navegação por circuitos

do mapa, nomeadamente nas que ele percorre poucas vezes. Algumas melhorias que gostaríamos de ter implementado são a verificação de becos sem saída e priorizar a sua navegação para o robô não ter de regressar a esse local mais tarde, e a verificação de circuitos ainda não navegados, através de informação recolhida à sua volta.

4 Planeamento

Para o desenvolvimento desta tarefa é necessário o mapa estar completamente mapeado pois é necessário saber as posições em que se encontram os beacons, para posteriormente conseguir obter o melhor caminho de menor custo.

No fim do mapeamento, criamos um grafo que contém todas as células navegáveis e invocamos um algoritmo recursivo que explora todas as possíveis combinações de caminhos mais rápidos entre beacons. Começando no primeiro nó (a partida), calculamos o tamanho do caminho mais curto para cada um dos outros beacons e invocamos a mesma função, recebendo esse custo e uma sublista de beacons (beacon 0 itera o beacon 1 e envia uma lista com todos os outros beacons. se iterar o beacon 2, envia uma lista com o beacon 1 e 3+...). Quando essa sublista estiver vazia, calculamos o custo do caminho que regressa à partida, e adicionamos o caminho percorrido, assim como o custo, a uma lista global de caminhos.

Por fim, escolhemos o primeiro caminho com menor custo guardado, como podemos observar na imagem.

À semelhança da nossa lógica de pesquisa na fase de mapeamento, usamos uma implementação do algoritmo de Dijkstra

4.1 Resultados e Conclusões

Estamos satisfeitos com a nossa solução visto que analisa todas as possibilidades. Uma possível melhoria seria prever quantas rotações teríamos de fazer, em adição às células que temos de percorrer. Metade dos caminhos possíveis têm o seu simétrico, mas um desses caminhos poderá (ou não) obrigar o robô a fazer meia-volta logo no início, o que iria prejudicá-lo.

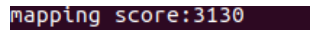
```
allpaths
[78, 0, 1, 2, 3, 0]
[62, 0, 1, 3, 2, 0]
[62, 0, 2, 1, 3, 0]
[62, 0, 2, 3, 1, 0]
[62, 0, 3, 1, 2, 0]
[78, 0, 3, 2, 1, 0]
bestpath [62, 0, 1, 3, 2, 0]
```

Fig. 8. Custo seguido do caminho, em todas as possibilidades de planeamento

5 Resultados

Para observar se o mapa está correto, utilizá-mos o script `mapping_score.awk`, para conseguir calcular o score previsto, executando:

```
$ awk -f mapping_score.awk planning.out.out myrob.map
```

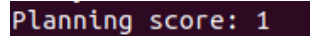


```
mapping score:3130
```

Fig. 9. Score do mapeamento

Posteriormente para observar se o caminho estava correto, utilizá-mos o script `planning_score.awk`, para conseguir calcular o score previsto (0, 1), executando:

```
$ awk -f planning_score.awk planning.out.out myrob.path
```



```
Planning score: 1
```

Fig. 10. Score do planeamento

6 Conclusão

No final do projeto, pensamos que foi conseguida a realização de um bom trabalho, conseguindo terminar e implementar o agente necessário. De referir que foi possível perceber que os erros dos sensores e da bússola dificultaram um pouco o desenvolvimento das tarefas, porém foi ultrapassada. Como trabalho futuro seria possível tentar diminuir o número de ticks a realizar o circuito na totalidade, por exemplo não navegando célula a célula, numa reta de células navegáveis, quando estas já foram mapeadas, concluindo assim num menor número de ticks.

References

1. [MicroRato04] Página oficial do Micro-Rato. (2004)
2. [Ciber04] Regras e especificações Técnicas da Modalidade Ciber. (2004)
3. <http://sweet.ua.pt/an/publications/ciber3.pdf>
4. Implementação do algoritmo de Dijkstra <https://morioh.com/p/ec5873a4c9f5>