Robótica Móvel e Inteligente - Assignment 2

Dário Matos (89288), Pedro Almeida (89205)

Universidade de Aveiro

1 Introdução

Neste *assignment*, o desafio proposto é desenvolver um agente robótico capaz de se localizar, orientar e mapear um mapa desconhecido. Assim, para completar o desafio na totalidade, pode-se dividir em três tarefas: localização, mapeamento e planeamento.

Para a localização do agente é necessário que seja capaz de se movimentar e localizar num mapa desconhecido. Uma vez que o uso de GPS não está disponível, foi utilizado um modelo de movimento, a distância a obstáculos e a bússola. Deve-se salientar que os sensores dispovíveis, como os sensores de distância e a bússola, tinham ruído e que a colisão com obstáculos é penalizada.

Para o mapeamento, o agente tem de ser capaz de explorar na totalidade um mapa desconhecido, fazendo o registo dos caminhos existentes. Ao mesmo tempo, tem de localizar um número variável de *targets* espalhados pelo mapa desconhecido. Depois da exploração do mapa, o agente deverá retornar à sua posição inicial.

No que toca ao planeamento, o agente deverá ser capaz de calcular o circuito fechado mais curto que permita passar por todos os *targets* encontrados no mapa desconhecido, começando e acabando na posição inicial que coincide sempre com um *target*.

Na realização deste *assignment* foi utilizado o ambiente de simulação CiberRato. O robo simulado está esquipado com dois motores (esquerdo e direito), três leds (explorar, retornar, fim) e sete sensores (bússola, quatros sensores de obstáculos, chão e de colisão). O mapa pode ser visto como um array bi-dimensional com uma tamanho máximo de 7x14 células. As células são um quadrado com tamanho fixo. O lado do quadrado é igual ao dobro do diâmetro do robo. As paredes entre células do mapa têm uma espessura de 0.2 diâmetros do robo.

O agente foi desenvolvido na linguagem de programação *Python*, devido à familiarização com a linguagem e ao rápido desenvolvimento de código.

2 Desenvolvimento

O trabalho desenvolvido no *assignment* anterior foi reaproveitado, tendo sido alterado e implementadas novas funcionalidades de modo a cumprir o novo desafio.

2.1 Máquina de estados

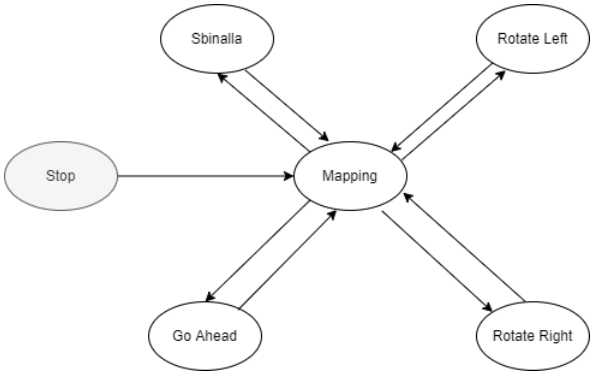

Figura 1. Máquina de estados implementada

- **go ahead** - Quando o agente se encontra neste estado vai deslocar-se num movimento linear até que atinja uma determinada posição objetivo (próxima célula, 2 unidades de diâmetro)
- **rotate left** - Estado de movimento de rotação (+90º)
- **rotate right** - Estado de movimento de rotação (-90º)
- **sbinalla** - Estado de movimento de inversão de sentido (+180º)
- **mapping** - Estado mais importante, é o estado inicial e o estado de decisão. É aqui que ocorre o cálculo do próximo movimento, o mapeamento do mapa, o planeamento do ciruito fechado mais curto, a correção da localização e a criação dos ficheiros de *output*

No fim de cada estado, quando o objetivo é alcançado, regressam ao estado *mapping*.

2.2 Movimento linear

Quando o agente se encontra no estado *go ahead* tem de andar em frente, independentemente da orientação, até alcançar uma posição objetivo pré-calculada. O deslocamento é feito através da seguinte fórmula:

```
rot = k * (m - r)

r_power = lin - (rot / 2)
l_power = lin + (rot / 2)
```

onde k é o fator de correção, m valor variável e r o valor que se quer ter. Assim, é passado o valor da bússola e a orientação do movimento pretendido de modo a que o agente avance num movimento linear. Esta solução funciona bem caso o agente esteja centrado nos corredores do mapa, contudo, por vezes, o agente inicializava este movimento linear muito perto de uma parede lateral e não era capaz de se desviar ou de se centrar. Consequentemente, com o ruído da bússola era possível haver colisões. Deste modo, foi desenvolvida lógica para o agente verificar os sensores de proximidade laterais e, caso esteja mais perto que um certo limiar (encontrado por um processo de tentativa-erro), desloca-se ligeiramente na direção oposta. Assim que baixar do limiar de proximidade, volta a usar a fórmula de deslocamento explicitada anteriormente.

2.3 Movimento de rotação

Consoante o estado de rotação em que o agente se encontra, é calculada a orientação objetivo em relação à orientação atual, isto é, caso esteja no estado *rotate left*, a orientação objetivo será somar noventa graus. Há que ter atenção aos casos especiais de cada estado devido ao sensor da bússola possuir valores entre -180 e 180 graus. Novamente no caso do estado *rotate left*, o caso especial é quando o agente se encontra numa orientação 180 graus, em que somar 90 iria resultar em 270, mas tendo em conta as limitações de valores da bússula, este valor tem de ser representado como -90 graus. Depois de calculada a orientação objetivo, o agente roda sobre si mesmo. Isto é alcançado dando uma potência inversa em cada roda. Enquanto a diferença da orientação e da orientação objetivo for maior que trinta graus, o agente roda com uma velocidade maior. Quando esta diferença baixa dos trinta graus, reduz-se a potência aplicada em cada roda de modo a não ultrapassar a orientação objetivo num *tick*. É concluído que o agente atingiu a orientação objetivo quando a orientação deste se encontra dentro de um intervalo de 10 graus (5 para cada lado).

2.4 Cálculo da próxima posição objetivo

O agente desloca-se sempre para uma posição objetivo que é uma das possíveis células adjacentes à sua célula atual. Esta posição objetivo está, assim, sempre a 2 unidades de diâmetro do robo de distância. Sempre que o agente está numa posição que ainda não visitou, analisa o seu redor e através dos sensores de proximidade conclui em que direções tem caminho livre para se deslocar. Das possíveis direções livres encontradas escolhe uma como próximo objetivo com as seguintes prioridades: frente, direita, esquerda e só depois atrás. As restantes direções livres são registadas como posições livres a visitar no futuro. Quando o agente está numa posição já visitada mas que ainda tem uma célula adjacente que não

foi explorada é essa selecionada como próximo objetivo. Quando o agente se encontra numa posição em que todo o seu redor são posições já visitadas, é calculado o caminho mais curto através do algoritmo *astar* para todas as posições não visitadas conhecidas e é selecionado como próximo objetivo a posição com o menor custo para lá chegar. Ao entrar no estado *mapping*, caso haja já um caminho calculado, o próximo objetivo é sempre o próximo "passo"desse caminho. Do mesmo modo, caso já haja uma posição objetivo essa é mantida.

Tendo uma posição objetivo definida, é então necessário verificar qual o próximo estado do agente para que seja possível alcançar essa mesma posição. Para isso, e tendo em conta a orientação do robo, é feita a diferença entre a posição atual do robo e a posição objetivo e verifica-se se é necessário uma rotação e qual a rotação a fazer (estado *rotate left*, *rotate right*, *sbinalla*) ou andar em frente (estado *go ahead*).

2.5 Verificação de chegada à posição objetivo

Para verificar se o agente chegou à posição objetivo calculada anteriormente, é tida em conta a sua orientação e a sua posição. Consoante a orientação, é verificado se a posição atual do robo está dentro de um limiar de diferença da posição objetivo. Esta verificação é apenas feita com a abcissa ou a ordenada da posição atual do robo. Caso a orientação seja 90^{o} (norte) ou -90^{o} (sul), é verificado a abcissa, caso a orientação seja 0^{o} (este) ou 180^{o}/-180^{o} (oeste) é verificado a ordenada. Se a verificação fosse feita com as coordenadas da posição do robo e da posição objetivo, o robo teria de alcançar exatamente a posição objetivo e andar sempre centrado. Contudo, rapidamente se notou que quando o robo de facto parava na posição objetivo, na realidade, já tinha ultrapassado essa posição. Isto acontecia derivado à inércia do movimento do robo. Assim, assume-se que se a posição do robo estiver dentro de um certo limiar de diferença da posição objetivo, já chegou ao objetivo. Por exemplo, se a posição objetivo for, por exemplo, (2,0) e a posição do robo for (2, 0.2) é assumido que chegou ao objetivo. Este limiar teve de ser afinado ao longo de vários testes para que a posição final do robo estivesse correta.

2.6 Localização

Uma vez que neste *challenge* o agente não tem acesso ao GPS, o primeiro passo para o desenvolvimento deste foi adaptar o agente para ser capaz de se localizar no mapa. De seguida, apresenta-se o modelo de movimento.

Modelo de movimento É tomada como referência uma posição (x, y, θ), onde x e y definem a posição do robo e θ a orientação do robo. Quando o comando DriveMotors(in_t^l, in_t^r) é enviado para o simulador no instante t as seguintes fórmulas permitem calcular a nova posição do robo:

$$out_t = \frac{in_i + out_{t-1}}{2} * N(1, \sigma^2)$$

onde out_t é a potência aplicada no instante t, out_{t-1} é a potência aplicada no instante $t-1$ e $N(1,\sigma^2)$ o filtro/ruído Guassiano de média 1 e desvio padrão σ.

Seguidamente, o movimento é dividido numa componente de translação, tendo em conta a orientação atual do robo

$$x_t = x_{t-1} + lin * \cos(\theta_{t-1})$$

$$y_t = y_{t-1} + lin * \sin(\theta_{t-1})$$

$$lin = \frac{out_t^l + out_t^r}{2}$$

e numa componente de rotação

$$\theta_t = \theta_{t-1} + rot$$

$$rot = \frac{out_t^l - out_t^r}{D}$$

Se ocorrer alguma colisão apenas a componente de rotação é aplicada no simulador.

Cálculo do GPS teórico O modelo de movimento apresentado anteriormente é usado para o cálculo do GPS teórico, isto é, calcular a posição do agente no mapa desconhecido. Do modelo de movimento apresentado, apenas foi implementado o cálculo da componente da translação do movimento sem o filtro/ruído Guassiano. A componente de rotação não foi calculada uma vez que o ruído da bússola não era significativo para o comportamento do agente, não havendo assim necessidade de tentar corrigir a orientação retornada pelo sensor. Este cálculo é feito no final de todas as iterações, sendo assim necessário guardar a potência que de facto foi enviada para os motores. Nos estados de rotação foi assumido que o robo não se deslocava (apenas roda sobre si mesmo) e a potência enviada para os motores é zero. O desempenho deste GPS teórico foi comparado com um sensor GPS "real". O GPS teórico consegue calcular a posição do agente de forma aproximada. Contudo, tem tendência a andar um pouco à frente da real posição do agente. A acumulação de erros pode levar a que o agente "pense"que está numa célula adiantada àquela que realmente está e fazer mal o mapeamento e planeamento ou até mesmo provocar colisões. É então necessário haver uma correção do GPS teórico. Na secção seguinte abordam-se as soluções encontradas e implementadas.

Correção da localização Como apresentado anteriormente, o modelo de movimento implementado calcula a posição do agente no mapa. Contudo este não é capaz de ser completamente preciso e os erros vão acumulando, podendo deixar o agente desorientado, falhando o mapeamento e planeamento ou até provocar colisões. Assim, foi necessário arranjar uma forma de anular/corrigir os erros acumulados durante o cálculo da nova posição do agente. Uma primeira abordagem foi alterar ligeiramente as fórmulas do modelo de movimento, reduzindo a variável *lin* quando a última potência enviada para os motores é zero, tentando assim eliminar a tendência do GPS teórico andar à frente da posição real. Apesar desta solução eliminar esta tendência, não é suficiente para uma correção eficaz e precisa. Decidiu-se então usar as posições das paredes por estas serem a única certeza durante a exploração do mapa. As paredes ao estarem entre as células do mapa têm sempre coordenadas ímpares. Deste modo, se o agente se encontra, por exemplo, na célula (2,0) orientado para Este e através dos sensores de proximidade deduz que tem uma parede do seu lado direito, sabe que essa parede tem coordenadas (2, -1). Com o auxílio dos sensores de proximidade, o agente consegue também deduzir a que distância se encontra da parede e assim calcular a sua real posição.

```
1     self.my_y = wall_position[1] + self.dist_from_sensor(right_sensor)
2                     + robot_radius + wall_thickness
```

Apesar do ruído dos sensores de proximidade estar presente, este não é significativo para desorientar o agente. O erro na posição real pode ser então ignorado. O raio do robo também entra no cálculo da posição uma vez que os sensores de proximidade estão colocados nas extremidades do robo, mas é desejada a posição do robo a partir do seu centro.

Esta correção é então feita sempre que o agente chega a uma posição objetivo e são usadas para tal todas as paredes que o robo detetar com os seus sensores de proximidade. Contudo, apenas corrige uma vez para cada abcissa e ordenada, isto é, se o robo estiver orientado para norte e detetar paredes tanto do lado direito como do esquerdo, apenas corrige a abcissa com uma das paredes. Isto acontece uma vez que o resultado da correção, em teoria, será igual com qualquer uma delas e a correção com a segunda parede iria fazer *overwrite* da primeira correção.

2.7 Mapeamento

Para fazer o mapeamento do mapa desconhecido, o agente precisa de ir guardando informação sobre o mapa enquanto o explora. Essa informação é guardada num dicionário em que a chave é a célula do mapa e o valor será "X", "ou -"caso seja uma posição livre, uma parede vertical ou uma parede horizontal, respetivamente. Quando o agente se encontra numa dada posição do mapa pela primeira vez, este mapeia a sua posição atual como válida (devido a ter lá chegado); seguidamente, verifica as posições em seu redor através da análise das distâncias medidas pelos sensores de proximidade, concluindo se são paredes ou caminho

livre e mapeia-as com o símbolo correspondente. Depois de o agente explorar o mapa na totalidade, acrescenta-se a posição inicial com o valor "I"e atualiza-se o valor das posições com *targets* para o número que identifica o mesmo. Posteriormente faz-se a escrita do mapa num ficheiro de *output*. Todas as posições são guardadas com um de offset de mais 28 na abcissa e subtraídas a 14 na ordenada, de forma a suportar o movimento de duas em duas células por parte do agente. Assim, o mapa terá o dobro do tamanho, e as coordenadas negativas existentes no programa serão retratadas corretas relativamente ao resto do mapa, ao escrever num ficheiro. Inicialmente a escrita do ficheiro de *output* era feita apenas depois do agente ter explorado o mapa desconhecido na totalidade mas alterou-se para ser feito sempre que o agente chega a uma posição nova. Isto permite ter algum *feedback* caso algo corra diferente do esperado no resto da execução do programa. De salientar que se a execução for abortada não é feita a renomeação do ficheiro de *output*, ficando com o nome "map.out".

2.8 Planeamento

Durante a exploração do mapa desconhecido, sempre que o agente passa por um *target*, guarda num dicionário o número que identifica o *target* e a sua posição. Estando a exploração do mapa concluída e, consequentemente, descobertos todos os *targets*, é calculado o caminho de menor custo através do algoritmo $A*$ de cada um dos *targets* para todos os outros, guardadando os custos associados (número de posições). É construído um grafo com os custos entre todos os *targets* que depois é passado para um algoritmo que permite resolver problemas semelhantes ao problema do caixeiro-viajente. Este algoritmo calcula todas as permutações de possíveis circuitos fechados que passa por todos os *targets* e retorna a ordem pela qual os *targets* devem ser percorridos. Finalmemte, é escrito um ficheiro de *output* com o circuito fechado de menor custo que passa por todos os *targets*. A escrita deste ficheiro de *output* ocorre apenas no final da exploração do mapa desconhecido. Ao contrário do ficheiro de *output* do mapeamento, este necessita que a exploração do mapa seja feita com sucesso (encontrar todos os *targets*) para que seja útil.

3 Testes e resultados

Uma parte essencial para o desenvolvimento deste desafio foi a testagem. Só assim era possível verificar se o comportamento do agente era o esperado, podendo encontrar falhas e situações que não tinham sido previamente planeadas. Neste sentido, o comportamento e desempenho do agente foi testado em vários mapas, uns com muitos obstáculos e vários pontos sem saída; outros com menos obstáculos, com mais retas e espaços vazios; e até um mapa completamente vazio, isto é, sem qualquer obstáculo para além dos limites do mapa.

Com os testes efetuados pode-se concluir que a limitação do agente desenvolvido é em mapas com poucos obstáculos, andar sempre em frente uma vez que o erro do cálculo da posição do robo vai acumulando e pode, no limite, fazer

com que o agente fique "desorientado"e faça o mapeamento e planeamento do mapa de forma errada. Para a atual dimensão do mapa este problema não se põe, no entanto o mesmo não pode ser dito caso o tamanho do mapa aumente. Pelo contrário, mapas com vários obstáculos revelaram que o comportamento do agente é mais estável pois os obstáculos permitem com que o cálculo da posição do robo seja corrigido com mais frequência.

No final do desenvolvimento do agente, este foi capaz de mapear e planear com sucesso todos os mapas usados em testes, podendo dizer-se que o desafio foi completado com sucesso.