Assignment 1 - RMI

Diogo Mendes (88801) e Raquel Pinto (92948)

Universidade de Aveiro

1 Introdução

Neste projeto, foram colocado vários desafios com diferentes objetivos. No desafio um o objetivo é desenvolver um agente adequado para controlar o movimento de um robô através de um circuito fechado desconhecido o mais rápido possível e sem colidir com as paredes circundantes. Para isso usou-se como base o código fornecido pelos professores e alterou-se apenas o valor dado às rodas tendo em conta os valores dados pelos sensores. Assim, conseguiu-se mover o robô de forma rápida e sem colisões, alcançando uma boa pontuação final.

No desafio dois, foi pedido para que o agente consiga explorar um mapa desconhecido e no final imprimir num ficheiro o mapa que descobriu. Deste modo desenvolveu-se uma maquina de estados que indica ao robô o próximo passo. Quando o robô chega ao meio da próxima célula, é calculado a próxima posição que o robô deve atingir e é guardada a vizinhança dele para mais tarde ser impresso o mapa. Assim obteve-se um ficheiro com o mapa que o robô criou após correr o desafio 2.

Por ultimo, no desafio três, o objectivo é explorar um labirinto desconhecido a fim de localizar um número de *targets* e calcular um melhor caminho fechado que permita visitar esses pontos começando e terminando no ponto de partida. Para isto, adaptou-se o código do desafio dois, pois ambos os desafios usam o algoritmo de pesquisa $A*$, a diferença está nas coordenadas de origem e nas coordenadas de chegada e no número de vezes que o algoritmo de pesquisa é corrido.

2 C1 – Control challenge

Neste desafio usou-se o código modelo fornecido pelos docentes tendo sido alterada apenas a função wander.

Nesta função, um primeiro *if*, verifica se o valor do sensor da frente é maior do que 1, o que corresponde a 1 unidade de distância. Caso seja, verifica-se qual o sensor com maior valor, esquerda ou direita, e consoante isso curva-se para o lado oposto. Os valores de velocidade foram atribuídos por tentativa e erro.

Caso o sensor da frente seja menor ou igual a 1, então existem dois *else ifs* que verificam se os valores dos sensores da esquerda e da direita, respectivamente, são maiores do que 2.1, o que corresponde a 0.47 unidades de distância, que considerando que podem existir erros nos sensores é um valor para a distância a que deve estar o sensor da parede se o robô se encontrar no centro das células. Caso o valor seja maior, e por isso a distância menor, através dos valores da velocidade obtidos por tentativa e erro, o robô vai curvar no sentido oposto à parede de que está mais próximo.

Finalmente, caso nenhum destes casos aconteça é porque o robô está perfeitamente no meio da célula e então pode simplesmente andar com velocidade máxima em ambas as rodas.

Após este código ter sido testado dez vezes, obteve-se em média uma pontuação de 3040 pontos, com zero colisões.

3 C2 – Mapping challenge

Neste desafio, o objetivo é que o robô consiga explorar um mapa desconhecido e no final imprimir num ficheiro o mapa que descobriu. A primeira diferença para o desafio anterior é que neste foram alterados os ângulos dos sensores laterais para 90^{o}.

Usou-se uma máquina de estados com 5 estados (*GA*, *RL*, *RR*, *INV*, *END*) que diz ao robô qual é o próximo passo. Se o robô estiver no estado *GA* ele anda em frente, se estiver no estado *RL* ou *RR* ele vira para a esquerda e para a direita, respetivamente, se estiver no estado *INV* significa que o robô terá de inverter e finalmente quando o robô está no estado *END* termina o desafio.

A cada ciclo, antes de entrar nos *case* da máquina de estados é feita uma análise do melhor estado para o robô (*função Estados()*) que através do valor da bússola atual (*compass*) e do valor da bússola objetivo (*compass_goal*) consegue definir o próximo estado. Note-se que o valor objetivo é definido noutra função. Desta forma, esta função consegue analisar se o robô está na direção correta para andar em frente ou se precisa de rodar mais para um lado.

Nos estados *GA*, *RL* e *RR*, é avaliado se o robô atingiu o seu objetivo (deslocar-se até ao meio da próxima célula) através da função *targetReached()* dando sempre uma margem de mais ou menos 0,15 em relação ao centro. Também é avaliado se o robô se encontra numa situação de beco, o que vai ser explicado mais à frente.

Se atingiu o objetivo é dado as rodas a velocidade (0,0) e, de seguida, chamada a função *mappingDecode()*, se ele ainda não tiver atingido o objetivo é chamada uma das funções *goAhead()*, *goLeft()*, *goRight()* consoante o estado onde ele se encontra (*GA*, *RL* e *RR*, respetivamente) que lhe permite andar em frente, rodar

para a esquerda ou para a direita. Estas funções definem a velocidade a aplicar em cada roda, utilizando o método abordado na aula de controlo, *PID*. No caso destas funções, a abordagem utilizada é apenas proporcional, ou seja, multiplica-se uma constante pelo erro, em x ou em y, consoante o eixo do movimento, para obter a velocidade de cada roda. Os valores para constantes foram obtidos por tentativa e erro.

Quanto ao *mappingDecode()*, esta é uma das principais funções. Para compreender esta função é importante perceber que ela apenas é chamada quando o robô se encontra no meio de uma célula!

A função começa por guardar o valor das coordenadas atuais na *LinkedList coordsAntigas*, que contém todas as coordenadas por onde já passou. E retira o vetor atual da *LinkedList visitaveis*.

Posteriormente, utilizando os valores obtidos pelos sensores, faz um mapeamento da sua vizinhança e guarda num array de 2 dimensões (*coords*) os dados obtidos (se é parede ou espaço livre) para no fim do programa escrever o mapa num ficheiro. Usando também esta informação a função adiciona a um *Set* temporário, as coordenadas para vizinhas de movimento possível (sem parede) e nas quais o robô ainda não esteve.

Caso, este Set seja vazio, ou seja, não existem movimentos possíveis para posições na vizinhança imediata onde ele ainda não esteve é chamada uma função *setCaminho* que vai criar um caminho, usando o algoritmo de pesquisa A estrela, para as coordenadas mais próximas marcadas na lista visitáveis.

Caso o Set contenha valores, vai ser calculada a próxima posição e o próximo valor de bússola objetivo (*compass_goal*) do robô.

A função *setCaminho()* corre o código A estrela explicado abaixo com o vetor atual como raiz e com um dos vetores da lista visitaveis. Após calcular esse caminho que é guardado na *LinkedList caminho* é chamada a função *runCaminho()* que vai definir o primeiro valor do caminho como *next* e alterar a bússola do objetivo (*compass_goal*) de acordo com isto. Sendo que no fim de cada execução é eliminado o primeiro valor do caminho.

Enquanto existirem valores no caminho a função *MappingDecode* vai continuar a chamar a *runCaminho()*.

Relativamente ao algoritmo de busca, o seguinte link foi utilizado como base para a classe *Node*, para a classe *aStar* e para as funções a elas associadas: https://stackabuse.com/graphs-in-java-a-star-algorithm/

Naturalmente, que o código foi apenas usado como base, tendo sido bastante alterado para adaptar-se ao nosso desafio. Algumas das principais alterações relacionam-se com a adição de um atributo para guardar o vetor correspondente ao nó, a adição de uma forma de *deep copy* dos filhos de um vetor quando se abre o nó, pois estavam a ser perdidos valores.

A função *printPath()* apesar do nome, foi alterada para devolver uma *LinkedList* com o caminho calculado.

Para além, destas funções principais, temos várias funções e classes auxiliares, entre as quais foi criada uma classe vetor (com seus atributos o X, o Y e os filhos, que neste caso são os vizinhos para onde o robô pode ir a partir desse vetor), e várias listas.

Criou-se um vetor chamado *next* que armazena as coordenadas seguintes, ou seja, as coordenadas para onde o robô se deve deslocar. Criou-se também uma lista chamada *coordsAntigas* que guarda as coordenadas dos centros das células onde o robô passou, uma lista chamada *visitaveis* que guarda as coordenadas dos centros das células que já foram marcadas como passagens ('X') e ainda não foram visitadas, e uma lista chamada *caminho* que contém o caminho em situações de beco.

Neste desafio tivemos de analisar quatro casos:

O primeiro caso, ocorre quando o robô tem paredes à esquerda e à direita, como regra geral, ele não anda em recuo as coordenas atrás dele serão parte do *coordsAntigas*, pelo que o *next* corresponde às coordenadas diretamente à frente do robô e o estado, normalmente, é o *GA*.

No segundo caso, o robô chega a uma curva, em que tem parede à frente e num dos lados o que significa que o seu *next* vai ser num dos seus lados e por isso terá de alterar o eixo do movimento. Na função *MappingDecode* não só vão ser calculadas as coordenadas seguintes como o *compass_goal* é alterado para um valor correto. Assim, a máquina de estados entrará no estado *RL* ou *RR* conforme tiver que rodar à esquerda ou à direita, respetivamente. E estes estados, com as funções de movimento correspondentes, farão o robô rodar até o *compass* ser igual ao *compass_goal*. Finalmente, depois da bússola alinhar, o estado *GA* fará com que ele ande em frente até ao objetivo.

O terceiro caso é semelhante, em parte, ao segundo, mas só com uma parede a ser detetada, ou sem parede de todo. Ou seja, quando o robô alcança um cruzamento, definiu-se cruzamento como a situação em que o robô pode ir para 2 ou 3 posições onde ainda não esteve. É de salientar que, caso o robô tenha estado previamente numa das posições, ou seja, uma das posições possíveis sem voltar para trás já foi visitada anteriormente, será considerado como uma parede. No caso de, só sobrar uma posição possível tratar-se-á de um dos casos anteriores consoante essa posição for frontal ou lateral.

Neste caso de cruzamento, a função *mappingDecode()* seleciona a primeira coordenada que guardou no Set de posições com movimentos possíveis como a posição objetivo, *next*, e guarda as outras na lista *visitaveis*.

Por último, no quarto caso o robô está num beco, esta situação foi definida como uma posição sem movimentos possíveis para coordenadas à sua volta não visitadas. Tal situação pode ocorrer quando o robô chega a um cruzamento onde todas as coordenadas à sua volta estão na lista *coordsAntigas*. Um exemplo simples, ocorre quando, o robô tem parede à frente, à esquerda e à direita (um

beco sem saída) em que a única posição para onde se pode mover é aquela de onde veio, ou seja, já foi visitada.

Neste caso, a próxima posição que o robô quer atingir (*next*) vai ser dada pelo último vetor inserido na lista *visitaveis*. Para o robô conseguir atingir esse ponto, utilizou-se uma pesquisa em A estrela que devolve o caminho ótimo (de menor custo) entre a posição atual e a posição *next* (último vetor inserido na lista *visitaveis*). As coordenadas deste caminho são inseridas na lista caminho e a função *runCaminho()* é chamada para movimentar o robô ao longo desse caminho.

Este desafio termina quando o robô tiver percorrido o mapa todo (ou seja, quando a lista visitaveis estiver vazia) ou quando o tempo do desafio terminar.

Finalmente, o estado *END* ocorre em duas situações. A primeira, quando o robô tiver percorrido o mapa todo. A segunda, quando o número de ciclos for maior que 4990, isto serve como precaução caso ocorra algum erro no código que impeça o mapeamento total. Neste estado, é escrito o mapa guardado no *array coords* num ficheiro de texto e quer o simulador quer o programa são terminados.

4 C3 – Planning challenge

Neste desafio, o objetivo é explorar um labirinto desconhecido a fim de localizar um número pré-definido de *targets* e calcular um dos melhores caminhos fechado que permita visitar esses pontos-alvo começando e terminando no ponto de partida.

Para este desafio, aproveitou-se o código anterior, do desafio 2, mas com algumas alterações.

Foi adicionada uma *LinkedList* que guarda os vetores das posições alvo. Como não foi encontrado um método que devolvesse o número de alvos na linguagem java, decidiu-se que o mais seguro era o robô mapear o mapa todo e depois consoante o número de alvos encontrados usar o A estrela para fazer o caminho de um para o outro.

Finalmente, basta juntar todos os caminhos num único e escrever este caminho num ficheiro de texto.

5 Conclusão

As principais conclusões sobre este trabalho são que, o desafio 1 apenas usa sensores para o movimento verificando proximidade às paredes para se desviar. O desafio 2 é exponencialmente mais complexo e usa principalmente as coordenadas de GPS para se movimentar, implementa funções de controlo do movimento, algoritmos de pesquisa, funções de mapeamento e escrita, etc. Sendo que, o desafio 3 se resume a apenas uma pequena alteração do 2 devido à forma como se implementou o desafio 2.