Assignment 1 – Robotic challenge solver using the CiberRato simulation environment

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 trabalho foram-nos propostos três desafios à qual vão ser explicados no decorrer do relatório. Com o primeiro desafio foi-nos proposto controlar o movimento de um robô através de um circuito fechado desconhecido tentando não colidir com as paredes e conseguir uma maior pontuação possível. Neste desafio utilizamos somente os sensores controlando a distância às paredes do circuito para controlar o movimento do robô no circuito. Num segundo desafio, foi necessário mapear o circuito inteiro, passando por todas as células. Foi implementada uma máquina de estados, utilizando também o algoritmo de Dijkstra que irá ser explicada posteriormente no decorrer no relatório. Como último desafio foi necessário localizar o número de beacons do circuito, e voltar ao lugar inicial, foi utilizado também o algoritmo de Dijkstra para conseguir descobrir o melhor caminho para o fazer. Os desafios no geral foram cumpridos, conseguindo o que era necessário para cada um deles.

1.1 Mapa e Robô

O ambiente de simulação CiberRato vai ser utilizado para avaliar os vários agendes desenvolvidos nos três desafios.

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.

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) está localizada no canto inferior esquerdo e a célula (13,6) está localizada no canto superior direito.

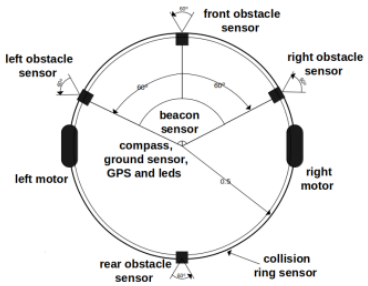

Fig. 1. Robô

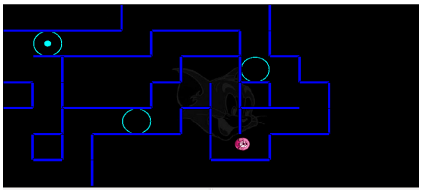

Fig. 2. Mapa

2 C1 - Desafio de Controlo

Neste desafio, como não temos acesso ao sensor GPS nem ao compasso, o robô não sabe onde está posicionado nem orientado no circuito. Neste desafio consideramos três tipos de distâncias do agente relativamente às paredes ("parede bastante perto", "parede ligeiramente perto" e "parede não está perto"). Para todos os três casos é definido um valor mínimo para entrar no if correspondente, onde em casa uns dos casos vão ocorrer velocidades de rotação diferentes.

No caso de a parede se encontrar bastante perto, e caso a closestWall seja a do sensor do centro, é feita então uma comparação entre qual dos restantes sensores (esquerda ou direita) se encontra mais próximo de uma parede, e é então definida uma velocidade de rotação para cada um dos motores, rodando para a direita se o valor do sensor da esquerda for superior ao valor do sensor da direita e vice-versa. com valores testados e que consigam um melhor desempenho.

Foi conseguida uma pontuação por volta dos 2570 pontos nos 5000 ticks.

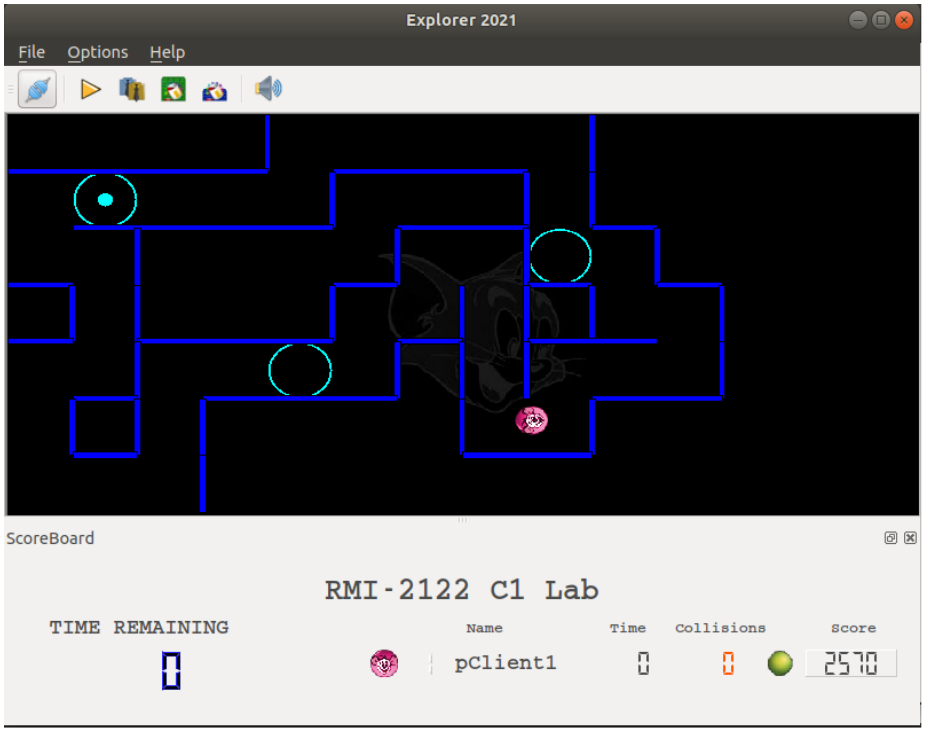

Fig. 3. Pontuação final challenge 1

3 C2 - Desafio de mapeamento

Para abordar este desafio, decidimos abandonar toda a lógica de navegação. Uma vez que temos acesso ao GPS e bússola, ambos sem ruído, decidimos usar uma navegação célula-a-célula. Isto significa que tomamos decisões assim que chegamos a uma nova célula, e não "on-the-fly", isto é, em todos os instantes.

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 -90º) 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 esquerda, frente, direita ou trás (nesta ordem de preferência). Decidimos dar o nome de "RE" a esta lógica, pois o robô prefere sempre rodar para a esquerda para continuar a sua navegação. Enquanto o robô está a executar uma lógica de navegação "RE", 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. Este último passo é feito no início do programa: Criamos um circuito composto apenas pela origem, inicializamos outro circuito, navegamos usando a lógica "RE", e assim que o robô visitar a origem, o circuito atual fica completo. Quando um circuito fica completo, abandonamos o "RE", 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.

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 objetivos, que simboliza o caminho para o novo circuito. Quando o robô chega ao novo circuito, regressa à lógica "RE", completando o ciclo. O ciclo fecha quando não existem candidatos, isto é, quando já não podem existir novos circuitos.

O movimento baseia-se em 3 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 $2^{\underline{o}}$ de diferença da nossa direção atual, ajustamos o ângulo de direção ligeiramente. Se estiver a mais de $30^{\underline{o}}$, rodamos o robô completamente 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 $90^{\underline{o}}$ 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 $30^{\underline{o}}$ de diferença, para rodar mais devagar até atingir uma diferença de $2^{\underline{o}}$.

Fases principais

$$Info \rightarrow Seen \rightarrow Go \rightarrow TryLeft \rightarrow TryFront \rightarrow TryRight \rightarrow TurnAround \rightarrow Go$$

Fases de mapeamento

$$Search \rightarrow Follow \rightarrow Go$$

!

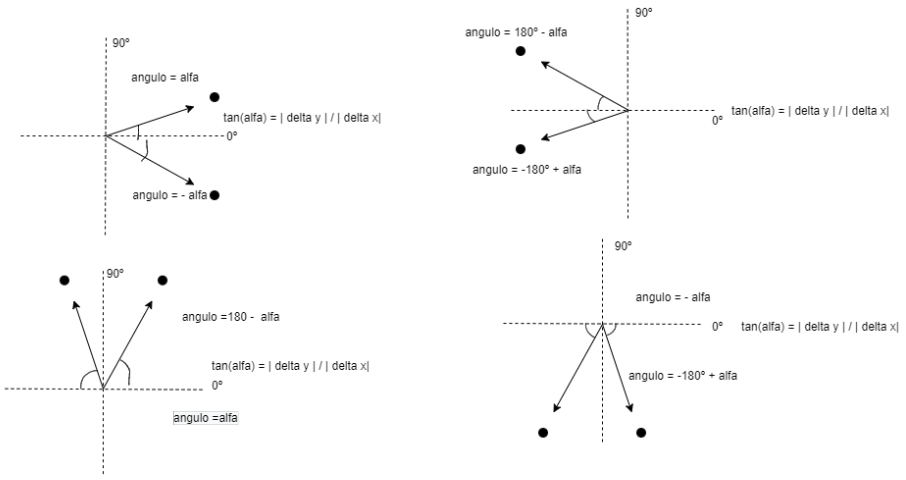

Fig. 4. Cálculo do ângulo de direção pretendido em função da localização da célula objetivo

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 180^{o}/-180^{o}, 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.

4 C3 - Desafio de planeamento

Para o desenvolvimento deste agente foi utilizada como base o código desenvolvido no challenge 2, pois foi necessário mapear o mapa para saber onde se encontravam as posições dos beacons, para posteriormente conseguir obter o melhor caminho de menor custo.

No fim do mapeamento (ou no fim do tempo disponível), criamos o grafo que contém todas as células navegáveis e invocamos um algoritmo recursivo que explora todas as possíveis combinações de circuitos que o challenge permite planear. 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 "circuitos". Por fim, guardamos o primeiro caminho com menor custo guardado.

5 Conclusão

No final do projeto e da realização dos 3 agentes para os diferentes challenges, pensamos que foi conseguida a realização de um bom trabalho, conseguindo terminar e implementar os agentes necessários. De referir que foi possivel perceber que os erros dos sensores dificultaram um pouco o desenvolvimento dos agentes, porém foi ultrapassada.

6 Trabalho futuro

A possibilidade de testar mais mapas, por exemplo com paredes diagonais, e outras características diferentes seria interesssante de resolver e implementar.

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