Path Finder Robot - Relatório

Rafael Morgado 104277

Universidade de Aveiro
DEM - Departamento de Engenharia Mecânica
Robótica Móvel

Abstract. Este relatório descreve o desenvolvimento e programação de um robô com tração diferencial, com o objetivo de seguir uma linha preta sobre um fundo branco, identificar interseções e otimizar o caminho até um destino final. Utilizando sensores infravermelhos para detecção da linha e encoders para controlo de movimento, o robô foi programado em C com o microcontrolador PIC32. A implementação inclui o controlo de movimento, detecção da linha final e uma estratégia para otimização do caminho em interseções. Os testes mostraram que o robô consegue seguir a linha, identificar interseções e parar ao atingir o destino.

1 Introdução

A robótica móvel tem se destacado como uma área importante de pesquisa, com aplicações em diversos campos, como automação, transporte, busca e resgate. O objetivo deste trabalho é programar um robô com tração diferencial, com a tarefa de seguir uma linha preta num fundo branco, identificar interseções e otimizar o caminho até um destino final. O robô é equipado com sensores infravermelhos (IR) que permitem detectar a linha e encoders nos motores para o controlo preciso do movimento.

O primeiro objetivo é construir um agente capaz de seguir a linha preta sobre o fundo branco. Quando houver várias opções, ou seja, quando a linha preta fornecer uma interseção, o robô deve virar à esquerda. O segundo objetivo é desenvolver um agente que pare automaticamente na posição final, que é marcada por uma linha preta mais larga.

Além disso, o terceiro objetivo é otimizar o caminho até a posição final, permitindo ao robô explorar diferentes opções de caminhos nas interseções e melhorar a eficiência do percurso. A solução proposta envolve a implementação de um algoritmo de controlo baseado nas leituras dos sensores, com o robô ajustando sua velocidade para manter-se alinhado à linha.

Este relatório descreve o desenvolvimento do robô, incluindo a implementação do controlo de movimento, detecção de interseções e otimização de caminho, além dos resultados obtidos nos testes realizados.

2 Metodologia

O desenvolvimento do agente robótico foi realizado utilizando o microcontrolador PIC32, com programação em C. O código implementado permite ao robô

seguir uma linha preta sobre um fundo branco, detectar interseções, otimizar o caminho até um destino final e parar automaticamente ao encontrá-lo. Abaixo estão descritas as principais funcionalidades e a estrutura do código utilizado.

2.1 Estrutura do Robô e Sensores

O robô é uma plataforma de tração diferencial equipada com sensores infravermelhos (IR) para detecção de linha e encoders nos motores para o controlo de movimento. O sistema de controlo foi desenvolvido utilizando a biblioteca `rm-mr32`, que contém funções para leitura dos sensores, controlo de PWM (modulação por largura de pulso) dos motores e controlo do servomotor.

2.2 Ambiente de Teste (Mapa)

O ambiente de teste foi projetado para simular um percurso no qual o robô deve seguir uma linha preta sobre um fundo branco, encontrar interseções e otimizar o caminho até o destino final. O mapa foi composto por uma série de linhas pretas no chão, formando um percurso com possíveis interseções e curvas mas não tendo ciclos, o que facilitou a nossa implementação.

O robô é posicionado numa posção aleatória do percurso, onde uma linha preta é detectada pelos sensores de linha. À medida que o robô segue o caminho, ele encontra interseções representadas por pontos onde a linha se divide, e o comportamento esperado é que o robô sempre siga para a esquerda, caso haja múltiplas opções. O ponto final do percurso é marcado por uma linha mais larga, indicando o destino do robô.

O mapa foi desenvolvido de forma a testar as funcionalidades de detecção de linha, interseção e finalização do trajeto, garantindo que o robô pudesse realizar a navegação de forma eficiente. A seguir, a Figura 1 ilustra o ambiente de teste utilizado no desenvolvimento e validação do agente robótico.

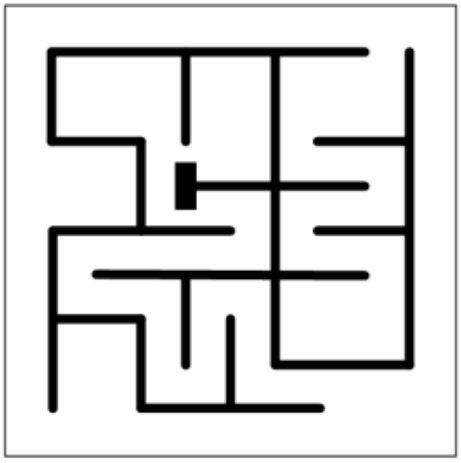

Fig. 1. Exemplo do mapa de teste utilizado no ambiente de navegação do robô.

2.3 controlo de Movimento e Leitura dos Sensores

A leitura dos sensores de linha é feita por meio da função `readLineSensors`, que ajusta o tempo de carga dos capacitores nos sensores para garantir leituras precisas. As leituras dos sensores de linha determinam a velocidade dos motores, com ajustes realizados em tempo real para manter o robô alinhado com a linha. A Figura 2 mostra-mos a distriuição desses sensores pelo nosso robô. A função `setVel2` é utilizada para configurar a velocidade dos motores, sendo que o controlo de movimento pode operar em loop fechado, utilizando a função `closedLoopControl` para ajustar as velocidades dos motores com base no feedback dos sensores.

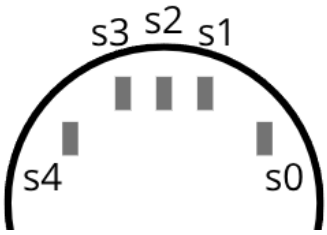

Fig. 2. Distribuição dos sensores infravermelhos (Parte de baixo do robô).

2.4 Algoritmo de Seguimento de Linha

O algoritmo de seguimento de linha foi projetado para controlar o movimento do robô com base nas leituras dos sensores de linha, permitindo que ele siga

uma linha preta sobre um fundo branco. O robô utiliza cinco sensores de linha dispostos de forma estratégica para detectar a posição da linha em relação ao seu centro. A partir dessas leituras, o algoritmo ajusta as velocidades dos motores para manter o robô alinhado à linha e lidar com interseções e curvas. Sempre que o robô procede para fazer uma curva é muito importante centraliza-lo bem para que faça a curva de forma suave e fique alinhado para retomar o seu caminho sem grandes dificuldades. Para tal ele antes de fazer a curva avança à mesma velocidade em linha reta cerca de 360ms, o que lhe dá tempo para colocar o centro do seu corpo no ápice de curva e ainda fazer pequenos ajustes de modo a ficar corretamente alinhado com a linha.

A principal lógica do algoritmo é a seguinte:

- **Seguir em frente**: Quando o sensor central (sensor 2) detecta a linha, o robô segue em frente, caso os sensores 1 e 3 detetem também a linha o robô realiza pequenos ajustes para se centralizar. As velocidades dos motores são assim ajustadas para que o robô mantenha um movimento reto e estável.
- **Curva à esquerda**: Caso o sensor 4 detete a linha, o robô roda sobre si 90^{o} no sentido anti-horário (à esquerda). Isso é indicado como uma prioridade do algoritmo, onde o robô sempre prefere virar à esquerda em situações de interseção.
- **Seguir em frente**: Caso o robô detecte as opções de virar à direita ou de seguir em frente ele irá seguir em frente ignorando a curva.
- **Curva à direita**: Caso apenas o sensor s0 detete uma linha ele irá por fim rodar sobre si próprio 90^{o} no sentido horário, irá virar à dirieta visto que é o único percurso possível.
- **Inversão de marcha**: Se o robô chegar a um beco sem saida (nenhum sensor deteta a linha), ele realizará uma manobra de 180^{o} e volta para trás.

Como foi dito, o algoritmo inclui ajustes contínuos na velocidade dos motores para compensar pequenas variações e manter o robô alinhado. Isso é feito por meio da correção proporcional, onde a diferença entre as leituras dos sensores é utilizada para ajustar as velocidades dos motores esquerdo e direito, garantindo que o robô se mantenha dentro da linha.

A prioridade das direções que o robô segue é muito importante pois caso ela fosse quebrada e o robô virasse à direita quando não era extremamente necessário ele iria entrar num loop e não avançaria na exploração do mapa.

Por fim, o robô é programado para detectar a linha final (uma linha mais larga que indica o destino) e parar automaticamente quando atingir essa linha. A função `checkForFinalLine` monitora os sensores e, quando todos indicam a linha final, o robô interrompe sua movimentação.

2.5 Detecção da Linha Final

A detecção da linha final é uma parte crucial para garantir que o robô pare corretamente ao atingir o destino. A linha final é identificada como uma linha mais larga e de cor preta, que o robô deve ser capaz de detectar utilizando seus sensores de linha.

Para evitar falsos positivos e garantir que o robô realmente esteja sobre a linha final, a verificação é feita de forma robusta. A função `checkForFinalLine` monitora as leituras dos sensores de linha durante um período de 200 ms (7 ciclos de 20 ms). Durante esse tempo, a função verifica se todos os sensores de linha detectam a linha preta ao mesmo tempo.

Caso algum sensor não detecte a linha, o robô continuará a procurar, retornando "falso" e permitindo que ele continue sua movimentação. Porém, se todos os sensores indicarem que estão sobre a linha final, após o tempo de espera de 200 ms, a função retorna "verdadeiro", os 4 leds irão acender indicando que o robô parou com sucesso na linha final.

2.6 Otimização do Caminho

A otimização do caminho foi implementada para melhorar a eficiência do percurso, minimizando o número de tentativas em interseções e evitando ciclos desnecessários. O robô explora o ambiente de forma inteligente, registrando os caminhos que escolhe em cada interseção e calculando qual o melhor a seguir, com base no destino dessa interseção, e quais caminhos evitar, que poderiam levá-lo de volta à mesma interseção.

Durante a fase de exploração, o robô utiliza uma estrutura de dados do tipo **pilha** (stack) para armazenar os caminhos que escolhe em cada interseção. Sempre que o robô chega a uma interseção, ele registra o caminho que decidiu seguir, atribuindo um número a esse caminho. Esse número é armazenado na pilha, o que permite que, após a exploração, o robô tenha um histórico completo dos caminhos tomados.

Quando o robô encontra uma interseção, ele verifica se o caminho a seguir leva a uma nova interseção ou se leva de volta à interseção onde esteve. Caso o caminho leve à mesma interseção, ele é evitado, e o robô opta por explorar outra opção. Caso o caminho leve a uma nova interseção, o robô seleciona-o e armazena essa escolha na pilha. Mas no caso de nenhum caminho levar a uma nova interseção ele roda $180^{\underline{o}}$ graus e anda para trás procurando novos caminhos. Isto acontece pois cada interseção tem um número máximo de caminhos possiveis `m_tries` e se o número de caminhos explorados `tries` exceder o número máximo de caminhos significa que para a frente não há solução, tendo que andar para trás.

Após a fase de exploração, a pilha contém o histórico completo dos caminhos tomados, que será utilizado na fase de otimização como demonstrado na Figura 3. O processo de otimização permite ao robô tomar decisões baseadas nas escolhas anteriores, evitando caminhos repetitivos e buscando a rota mais eficiente para alcançar o destino final.

```
Interseção: 1 caminho escolhido: 1
Interseção: 2 caminho escolhido: 3
Interseção: 3 caminho escolhido: 1
Interseção: 4 caminho escolhido: 1
Interseção: 5 caminho escolhido: 2
```

Fig. 3. Exemplo de stack após exploração.

Na fase de otimização, o robô detecta o tipo de interseção em que se encontra e consulta a pilha para determinar o próximo caminho a seguir. Quando o robô chega à primeira interseção, ele consulta a pilha e segue o caminho correspondente ao número registrado. Ao chegar à próxima interseção, o robô incrementa o valor na pilha, garantindo que o próximo caminho a ser seguido seja o correto.

Resumidamente o algoritmo de otimização segue as seguintes etapas:

- Durante a exploração, o robô, é defrontado com várias interseções e precisa de saber como agir para conseguir percorrer eficientemente o caminho que o leva ao destino final.
- Portanto quando o robô se encontra numa interseção, ele calcula qual caminho leva a outra interseção (e deve ser seguido) e qual caminho leva de volta à mesma interseção (e deve ser evitado).
- Após a exploração, a pilha é consultada para determinar o próximo caminho a seguir. O robô consulta o número do caminho na pilha correspondente à sua posição atual e segue o caminho indicado.
- À medida que o robô avança, ele incrementa o valor na pilha para garantir que, quando encontrar a próxima interseção, o próximo valor correspondente à interseção seguinte será consultado.

Com isto o robô na fase de otimização irá fazer um percurso muito mais eficiente e irá chegar ao seu destino em menos tempo, como demonstrado na Figura 4.

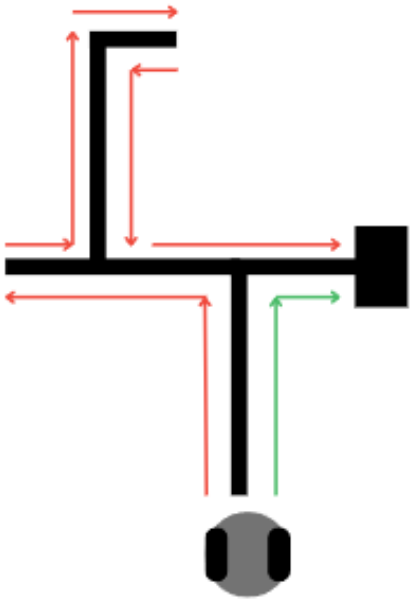

Fig. 4. Comparação entre o caminho explorado (linha vermelha) e o caminho otimizado (linha verde).

2.7 controlo PID

O controlador PID foi utilizado para garantir que o robô mantenha uma trajetória estável. A função `pid` é chamada a cada interrupção do timer, calculando os erros nas leituras dos encoders e ajustando os comandos do PWM para os motores de forma proporcional e integral. Isso ajuda a manter a precisão do movimento, corrigindo desvios e evitando que o robô saia do caminho.

2.8 Interrupções e Leitura dos Encoders

As interrupções são usadas para ler os encoders dos motores, atualizando a posição do robô com a função `updateLocalization`. O cálculo da distância percorrida pelos motores é feito com base nas leituras dos encoders e nos parâmetros do robô, como o diâmetro das rodas e a distância entre elas.

2.9 Bateria e Segurança

O sistema também monitoriza a voltagem da bateria do robô. Caso a voltagem caia abaixo de um valor crítico, o robô é programado para parar imediatamente e exibir um sinal de alerta com os LEDs.

3 Resultados

O objetivo deste trabalho foi desenvolver um robô capaz de seguir uma linha preta sobre um fundo branco, identificar interseções e otimizar o caminho até o destino final. O robô foi capaz de cumprir os dois primeiros objetivos de forma coerente, mas o terceiro objetivo, relacionado com a otimização do caminho e a navegação baseada na pilha de interseções, apresentou alguns desafios.

3.1 Cumprimento do Primeiro Objetivo

O primeiro objetivo, que era fazer o robô seguir a linha de forma precisa, foi alcançado com sucesso. O robô foi capaz de detectar a linha preta utilizando os sensores de linha e ajustar sua trajetória em tempo real. Durante os testes, o robô manteve-se alinhado com a linha, realizando correções suaves nas velocidades dos motores sempre que necessário. O algoritmo de seguimento de linha foi eficaz em manter o robô sobre o caminho, mesmo em curvas suaves e pequenas variações na trajetória.

3.2 Cumprimento do Segundo Objetivo

O segundo objetivo, que era fazer o robô parar automaticamente ao alcançar o destino final, foi alcançado com sucesso. O robô foi capaz de detectar a linha final, mais larga que as demais, e parou quando todos os sensores de linha indicaram que ele estava sobre essa linha. A função de detecção e paragem funcionou corretamente durante os testes, permitindo que o robô concluísse o percurso com precisão.

3.3 Desafios no Terceiro Objetivo: Otimização do Caminho

O terceiro objetivo, relacionado com a otimização do caminho e a navegação baseada na pilha de interseções, foi parcialmente alcançado. Embora o robô tenha sido capaz de percorrer o caminho otimizado em várias situações, houve alguns desafios notáveis durante os testes:

- **Falhas no armazenamento na pilha**: Em algumas interseções, o robô não conseguiu registrar corretamente o caminho a seguir na pilha. Isso ocorreu principalmente em situações de transição entre interseções, onde o robô não conseguiu armazenar o número do caminho de forma precisa, levando a falhas na navegação posterior.
- **Falhas no seguimento do caminho**: O robô apresentou algumas dificuldades em seguir o caminho otimizado em algumas situações. Embora tenha seguido corretamente os caminhos registrados na pilha, em certos pontos o robô desviou-se do percurso devido a erros na leitura dos sensores ou no cálculo da trajetória. Isso resultou em perda de alinhamento e no abandono do caminho otimizado.

– **Problemas no próprio robô**: Em alguns testes, o robô não foi capaz de completar o percurso devido a problemas internos, como queda na carga da bateria ou sobrecarga no sistema de controlo, o que fez diminuiar a sua capacidade de processamento e por sua vez a sua performance geral.

Esses problemas indicam que, embora o robô tenha sido capaz de otimizar o caminho em certas situações, a execução prática foi prejudicada por limitações técnicas, como a falha no armazenamento e leitura da pilha, além de problemas com a gestão de energia e a capacidade de navegação do próprio robô.

4 Conclusões

Em resumo, o robô foi bem-sucedido no cumprimento dos dois primeiros objetivos. Ele demonstrou capacidade em seguir a linha e parar automaticamente ao alcançar o destino final, como esperado. Além disso, o robô foi capaz de identificar interseções e explorar o ambiente de forma inteligente. No entanto, o terceiro objetivo, relacionado à otimização do caminho, apresentou desafios significativos. Durante os testes, foram identificadas falhas no armazenamento da pilha de caminhos, dificuldades no seguimento preciso do caminho otimizado e limitações no desempenho do robô, especialmente devido a questões de energia e sobrecarga.

Esses problemas indicam que, para garantir uma navegação eficiente e otimizada, serão necessárias melhorias no controlo de energia, no armazenamento das interseções e no próprio algoritmo de otimização do caminho. Apesar desses obstáculos, o robô demonstrou grande potencial para navegação autônoma e otimização de caminho. As futuras melhorias do projeto deverão focar em ajustes na precisão do movimento e na implementação de uma abordagem mais robusta para a otimização do caminho, a fim de melhorar a confiabilidade e a eficiência geral do sistema.