OTIMIZAÇÃO DE ALGORITMO DE RESOLUÇÃO DE LABIRINTO PARA ROBÔS MICROMOUSE

Francisco Marcolino Rodrigues Filho*, Darieldon de Brito Medeiros*, Otacílio da Mota Almeida*

*Universidade Federal do Piauí
Departamento de Engenharia Elétrica
Teresina, Piauí, Brasil

Emails: fmarcolino@live.com, darieldonbm99@outlook.com, otacilio@ufpi.edu.br

Abstract— Micromouse is a robotic technology that began in the 1970s and has been gaining ground as we move into high-density integration technologies, sensors, and intelligent control theory. The challenge consists of solving a labyrinth with the use of mini mobile robot (Micromouse). There are several algorithms for solving the labyrinths with the mobile robots, however, the most commonly used is Flood Fill. Based on this algorithm, this work proposes a new strategy for the implementation of intelligent algorithm of solving a labyrinth. The idea is to use a data structure to store both the robot's way of going and the way back. To calculate the performance of the algorithm of several simulations were performed and compared with the classic Flood Fill, resulting in a substantial saving of RAM that favors a real-time implementation of the system embedded in the Micromouse. Therefore, this work has as objective the development of intelligent algorithm of solving a labyrinth, in C language, optimized for microcontrollers at low cost, important step for the Micromouse project in question.

Keywords— Micromouse, Intelligent algorithm, Maze, Flood Fill, Robotics.

Resumo— Micromouse é uma competição robótica que teve início na década de 1970 e vem ganhando espaço à medida que as tecnologias de integração em alta densidade, sensores e teoria de controle inteligente avançam. O desafio consiste na resolução de labirinto com a utilização de mini robô móvel (Micromouse). Existem diversos algoritmos de resolução de labirinto com robôs móveis, porém, o mais utilizado é o *Flood Fill*. Com base neste algoritmo, este trabalho propõe uma nova estratégia para implementação de algoritmo inteligente de resolução de labirinto. A ideia é de utilizar uma estrutura de dados para armazenar tanto o caminho de ida do robô como o caminho de volta. Para avaliar o desempenho do algoritmo várias simulações foram realizadas e comparadas com o *Flood Fill* clássico resultando em substancial economia de RAM que favorece a implementação em tempo real do sistema embarcado no Micromouse. Portanto, este trabalho tem como objetivo o desenvolvimento do algoritmo inteligente de resolução de labirinto, em linguagem C, otimizado para microcontroladores de baixo custo, passo importante para o projeto Micromouse em questão.

Palavras-chave— Micromouse, Algoritmo inteligente, Labirinto, Flood Fill, Robótica.

1 Introdução

Micromouse é uma competição antiga. Iniciada na década de 1970, hoje existem várias competições da modalidade em todo o mundo (Li et al., 2010). A competição envolve robôs móveis autônomos para resolver um labirinto composto de 16x16 células de 180 mm^2 cada. A padronização da competição foi introduzida pela revista do IEEE *Spectrum* em 1977. O robô tem um tempo limitado para resolver um labirinto composto de 256 células, cuja configuração é estabelecida no início da competição (Li et al., 2010).

À medida que a tecnologia avançou, os robôs se mordenizaram tornando-se dispositivos compactados com tecnologia SMD, alta densidade de integração que aliado a avanços na tecnologia de construção, tornou os robôs mais compactos e velozes. Os avanços nas teorias de controle conferiram maiores capacidades de movimento ao robôs, como por exemplo o movimento controlado em diagonais ao invés de zigue-zague em células quadradas de 180 mm de largura.

No Japão, a competição avançou de tal maneira que uma modalidade de labirinto de 32x32 células foi introduzida (Su et al., 2013). Na Figura 1, apresenta-se uma competição naquele país.

Figura 1: Torneio Micromouse no Japão, em 2013.

Como ferramenta de estímulo ao ensino e aprendizado, segundo Silva et al. (2015), o projeto Micromouse é considerado mais uma ferramenta para estimular o interesse em alunos na área das Ciências e Engenharia por promover competências consideradas importantes para estas áreas do conhecimento. Lopez et al. (2015) acrescentaram

também que o mini robô móvel autônomo, Micromouse, é ideal para o estudante expandir seus conhecimentos em diversas áreas correlatas. A Figura 2 apresenta uma foto do micromouse utilizado.

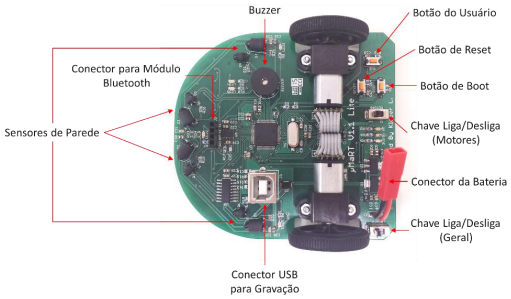

Figura 2: Composição do Micromouse utilizado.

O robô tem como objetivo sair de um dos cantos do labirinto e alcançar o centro no menor tempo possível. Isto se chama corrida. O tempo da corrida do centro à célula de partida é desconsiderado. O automato é pontuado de acordo com três parâmetros: velocidade, eficiência em resolver labirinto, e confiabilidade do Micromouse (Li et al., 2010).

Além disso, existe um tempo de conhecimento do labirinto (primeira corrida). Em algumas modalidades, Micromouse tem até 15 minutos para fechar a *corrida*. Dentro da janela de tempo o Micromouse poderá tentar várias vezes a corrida com o objetivo de diminuir seu tempo (Li et al., 2010).

O robô deve ser totalmente autônomo. Caso necessite de alguma assistência no meio da corrida, o mesmo é penalizado por toque. O Micromouse deve apresentar as seguintes características para resolver com eficiência o labirinto (Dai et al., 2015): estabilidade Mecâninca e Elétrica; velocidade mecânica e de processamento de informação; precisão nos movimentos e sistema de sensores; capacidade de armazenamento eficiente do caminho mais curto; habilidade de resolver, utilizando inteligência computacional, qualquer labirinto regulamentar.

Uma das primeiras tentativas de dotar o *Flood Fill* tradicional de inteligência pode ser encontrada no trabalho de (Cai et al., 2012), com algoritmo *Flood Fill* com expectativa de paredes.

Neste trabalho um algoritmo com baixo custo de armazenamento é proposto, para o algoritmo clássico. A ideia é de utilizar uma estrutura de dados para armazenar tanto o caminho de ida do robô como o caminho de volta. Esta estratégia promove uma substancial economia de RAM favorecendo a implementação em tempo real do sistema embarcado do Micromouse.

A seção I trata-se de um resumo geral do Micromouse. A seção II relata alguns algoritmos para resolução de labirinto. A interpretação física e lógica do *Flood Fill*, bem como o seu funcionamento básico e como é obtida a otimização de memória, é mostrada nas seções III, IV e V. Os principais resultados são mostrados na seção VI, e a última seção mostra as considerações finais.

2 Algoritmos de resolução e aprendizado

Neste trabalho, o *Micro Mouse Maze Editor and Simulator* (Solver, 2013) foi utilizado como uma ferramenta em *Java* para simular o comportamento do robô em diversos labirintos. Quatro algoritmos são comumente utilizados neste simulador, a saber:

1. `Left Wall Follower`: Seguidor de paredes à esquerda.
2. `Right Wall Follower`: Seguidor de paredes à direita.
3. `Tremaux`: Algoritmo força bruta - visita todas as células do labirinto.
4. `Flood Fill`: O mais utilizado - simples e eficiente para este tipo de competição.

Os dois primeiros algoritmos usam o princípio de seguidores de parede. Os problemas encontrados nos algoritmos seguidores de paredes, em não lograr êxito em labirintos complexos, são resolvidos com os algoritmos surgidos a partir da Teoria dos Grafos (Sadik et al., 2010). Destes algoritmos o quarto algoritmo *Flood Fill* foi utilizado por ser o que permite, com relativa simplicidade, a utilização de inteligência computacional e atende os requisitos especificados para a corrida do Micromouse. Devido à grande utilização e recomendação dele em diversos trabalhos, o tal algoritmo serviu como base para a construção do algoritmo proposto otimizado em linguagem C, porque atende as melhores expectativas pelo fato de minimizar a área de exploração do labirinto.

3 Interpretação Física e lógica do *Flood Fill*

O *Flood Fill* é um algoritmo que cria uma matriz pré-programada com as dimensões do labirinto, sendo que os números são colocados de tal forma que o labirinto se organize em curvas de níveis, o início tem a numeração mais alta, e o final (chegada) é numerado com zero. A ideia desse algoritmo é percorrer as células do maior para o menor valor.

O *Flood Fill* clássico, considerado como um dos mais eficientes para resolver labirintos (Silva et al., 2015) é baseado em tornar a superfície do tablado em um relevo com curvas de níveis, dando números para cada célula. A partir disso, o robô tende a descer esta superfície até o ponto mais baixo. O destino sempre tem a numeração 0; uma célula com valor 1 está a um passo do alvo. Se

tiver valor 4, está a quatro passos para o destino (Silva et al., 2015).

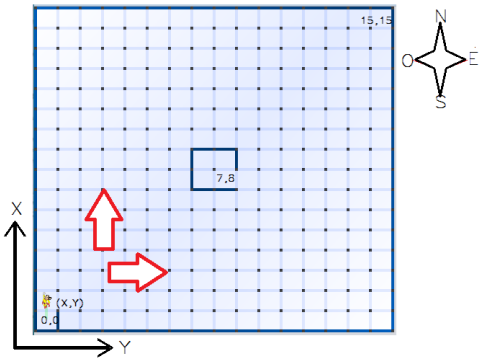

Figura 3: Interpretação do labirinto para o *Flood Fill*

A interpretação lógica que abstrai a constituição física do labirinto, pode ser concebida no algoritmo *Flood Fill* como um plano cartesiano, como mostra a Figura 3. Cada coordenada é responsável por acessar uma estrutura de dados. A coordenada X representa a linha e a Y representa a coluna da célula. As setas mostram o sentido de crescimento dos números das coordenadas. Também é padronizada a referência de orientação. Ou seja, se o robô está se movimentando da célula 0,0 para a célula 1,0, então ele está andando para o Norte.

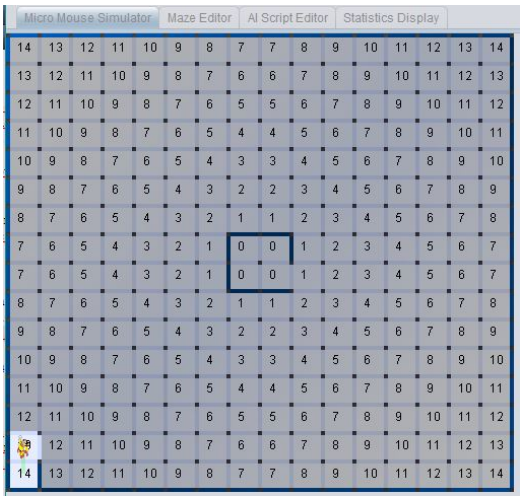

Figura 4: Numeração das distâncias sobre o labirinto

Cada célula possui informações das paredes. Por exemplo, na coordenada 15,15 (Figura 3), há parede ao norte e ao leste. Então, estas informações podem ser acessadas diretamente da célula, ou seja, as variáveis célula(15,15).wall[N] e célula(15,15).wall[E] retornarão verdadeiras, enquanto as variáveis célula(15,15).wall[O] e célula(15,15).wall[S] retornarão falsas. Então, à medida que o robô vá percorrendo o labirinto, ele vai atualizando as informações das paredes das células, nas estruturas de dados do microcontrolador, além de atualizar também a variável Checado.

Então, o algoritmo, inicialmente, considera o labirinto como se não existissem paredes e enumera as células inicialmente conforme mostrado na Figura 4. Estes valores são o número mínimo de passos até então para chegar ao alvo (valor zero). Um passo moverá o robô de sua célula atual para a célula com menor distância da célula destino. A princípio, o robô sempre procurará adentrar para a célula que tem o menor valor de distância até encontrar a célula de destino (contém distância zero). A Figura 5 mostra um algoritmo básico de tomada de decisão.

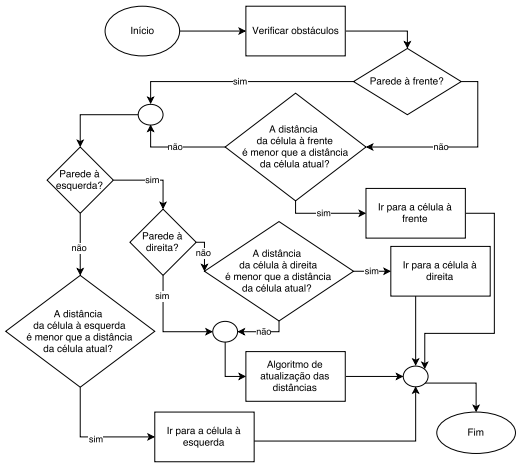

Figura 5: Fluxograma básico de tomada de decisão

Como foi mencionado anteriormente, o número da distância em cada célula é levado em conta de forma que o robô interpreta as áreas desconhecidas como aberto, ou seja, sem paredes. Segundo as regras da competiçao, as quatro células centrais tem a distância dada como zero. Então a partir destas células, é aplicada a regra das distâncias das células vizinhas para determinar os valores de cada célula.

4 Algoritmo *Flood Fill* não recursivo com otimização de memória

Neste trabalho o algoritmo *Flood Fill* é utilizado como base para desenvolver o esquema de otimização de memória proposto e que tornou mais adequado o seu uso em um hardware embarcado e que tenha restrição de memória. O algoritmo *Flood Fill* básico não recursivo é resumido no fluxograma de Figura 6.

O algoritmo de atualização das distâncias não recursivo utiliza pilhas para armazenamento do endereço das células empilhadas. Desta forma, não é necessária a recursividade, que aumentaria o consumo de memória e números de instruções em sistemas embarcados, os deixando lentos.

Os passos necessários para implementar esta parte do algoritmo de atualização é dado como mostra a Figura 7.

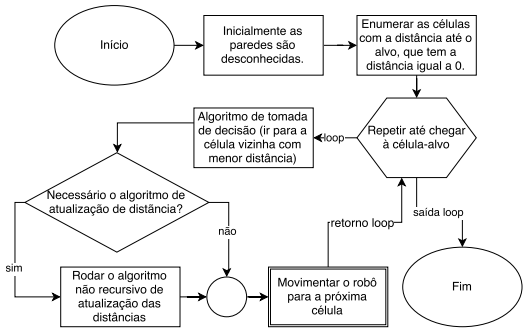

Figura 6: Fluxograma básico do *Flood Fill*

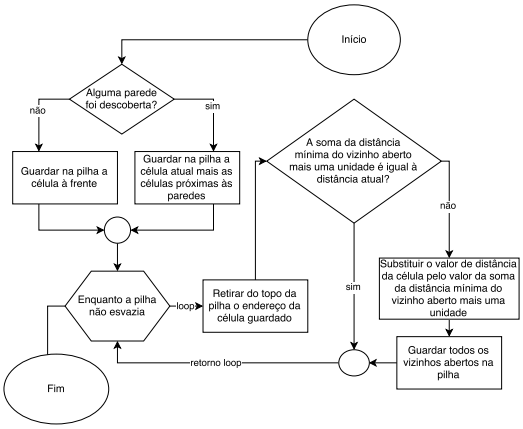

Figura 7: Fluxograma básico do algoritmo de atualização das distâncias não recursivo

Para entender a implementação do algoritmo proposto conforme os passos da Figura 7, as seguintes definições são importantes e deve-se considerar que cada célula contenha estrutura de dados para armazenar as seguintes informações:

- `distância`: Esta variável, do tipo inteiro de 16 bits, contém o número de passos para o robô chegar ao alvo.
- `paredes`: Esta variável, na verdade é um vetor do tipo booleano de quatro posições. Cada índice, de 0 a 3, armazena, respectivamente, a informação das paredes descobertas pelo robô ao norte, leste, sul e oeste. Exemplo: paredes[0] é 1 quando tem parede ao norte ou 0 quando não tem parede ao norte.
- `Checado`: Esta variável do tipo booleana é alterada para verdadeiro, caso o robô tenha visitado esta célula. Importante para evitar consumo de energia dos sensores de distância.

5 Resultados e Discussões

O algoritmo de otimização proposto, implementado em C, utiliza um conjunto de simbolos que permite a análise do percurso de rota do robô no labirinto. Os símbolos são úteis em um processo de depuração de corridas. Abaixo são mostrados alguns símbolos básicos essenciais para depurações.

```
+----+----+----+
|*  5 *  4 V  3|
+----+----+    +
|   0    1    2|
+----+----+----+
```

O símbolo `'+'` foi utilizado para as quinas. O `'---'`, para as paredes Norte/Sul. Já o símbolo `'|'`, para indicar as paredes Leste/Oeste. E o asterisco, para indicar que a célula já foi visitada. Foram utilizados também os símbolos `<`, `>`, `^` , `V` para mostrar a orientação do Micromouse.

No algoritmo proposto, a cada corrida (tanto da IDA como da VOLTA) as distâncias das células são reinicializadas como se o labirinto não tivesse nenhuma parede. Com esta estratégia, utilizando a mesma estrutura de memória, economiza uma variável de dois bytes para cada célula, um total de $16 \times 16 \times 2$ *bytes* de RAM, ou seja, 512 *bytes*, cerca de 1/4 do espaço utilizado por todo o algoritmo.

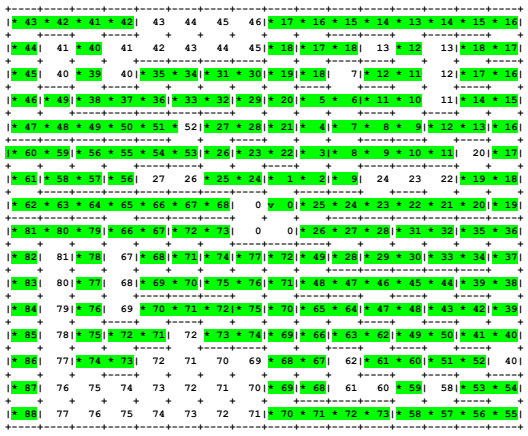

Figura 8: Percurso do robô no labirinto *Seoul* do algoritmo criado, na primeira corrida real

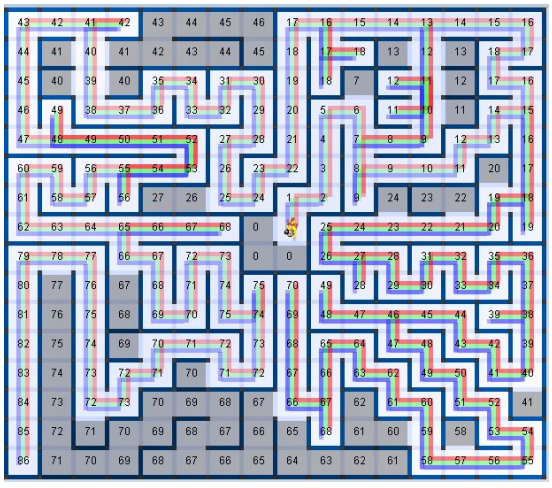

Figura 9: Percurso do robô no labirinto *Seoul* com algoritmo do *Micro Mouse Maze Editor and Simulator*

A Figura 8 mostra a ida do robô. Em compa-

ração com o percurso do algoritmo *Flood Fill* do simulador *Micro Mouse Maze Editor and Simulator* da Figura 9, são bastante idênticos ambos os percursos. Algumas diferenças são devido à tomada de decisão para vizinhos abertos de mesma numeração.

Ao final de uma corrida, as distâncias das células são redefinidas e inicia-se a *varredura virtual* do robô. A condição de parada é obtida quando o número de passos da corrida anterior se torna idêntica ao número de passos da última tentativa, em um processo de aproximação sucessiva. Assim, o robô poderá sair da célula central à celula de partida seguramente na menor distância possível para voltar em seguida.

Para o labirinto em questão, foram necessárias 5 corridas virtuais para este *primeiro run*. A tendência é que, conforme vá aumentando o número de corridas *reais*, o número de corridas virtuais caia. A Figura 10 mostra a redefinição das distâncias para permitir a volta do robô. Já a figura 11 mostra a última corrida virtual, com detalhe no número de passos. Conforme vá utilizando o algoritmo *Flood Fill* e o algoritmo de atualização das distâncias para corridas *virtuais*, o robô terá o seu menor caminho para chegar à célula de partida de fato.

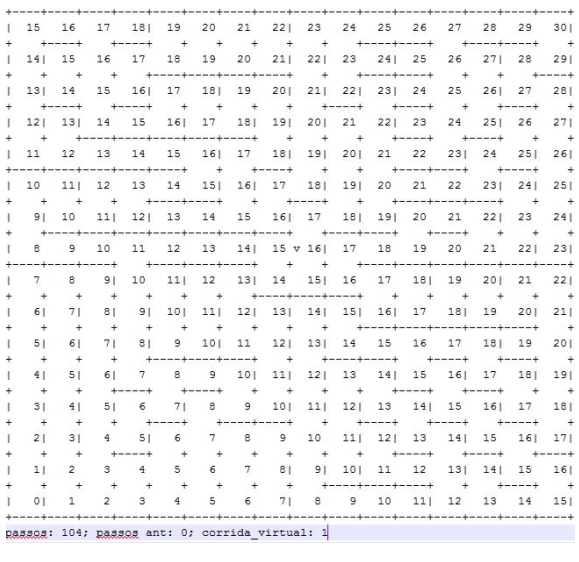

Figura 10: Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 1

O processo se repete na preparação da IDA do robô à célula de destino. Desta forma, conseguiu-se usar somente uma estrutura de dados que armazena tanto o caminho de ida como também o caminho de volta.

A eficiência do algoritmo permanece intacta. Após 3 corridas *reais*, tanto o algoritmo proposto como o algoritmo do simulador *Micro Mouse Maze Editor and Simulator* apresenta a mesma distribuição dos números das células, como mostram as Figuras 12 e 13.

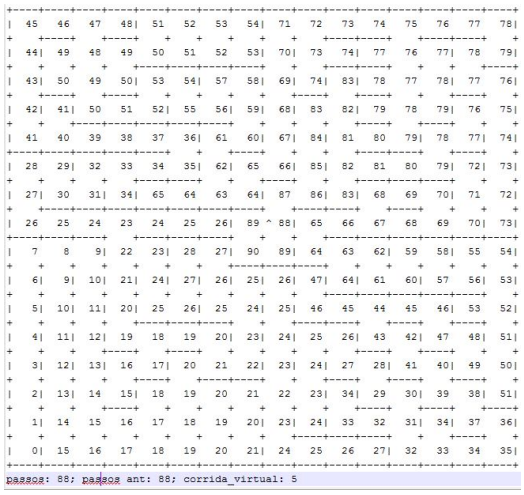

Figura 11: Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 5

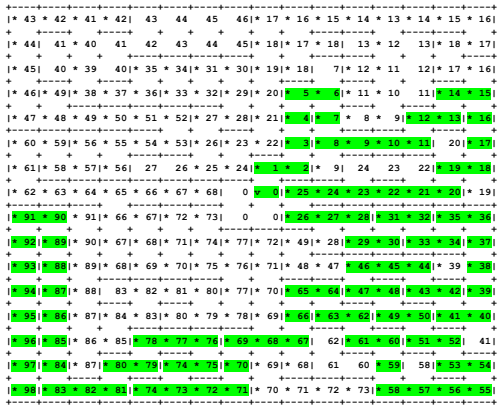

Figura 12: Labirinto *Seoul* após 3 corridas reais

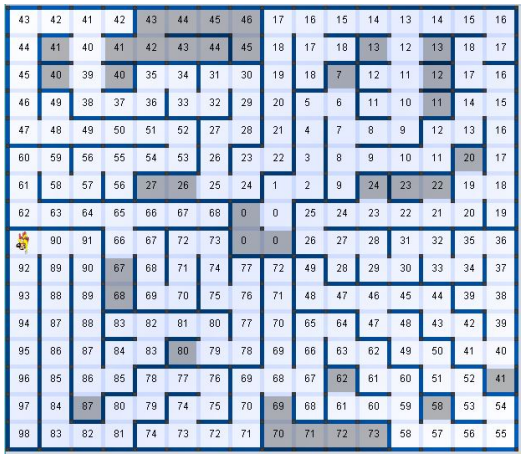

Figura 13: Labirinto *Seoul* após 3 corridas reais no *Micro Mouse Maze Editor and Simulator*

Algumas estatísticas foram feitas para os algoritmos *Flood Fill*, expostas na Tabela 1, onde

pode-se perceber que há algumas diferenças, para o mesmo labirinto. No entanto, o número de células visitadas para a melhor corrida permanece o mesmo. Estas diferenças podem ser explicadas. Para decisões de sentido para vizinhanças de mesmo número, a prioridade para o sentido é diferente.

Tabela 1: Desempenho dos algoritmos no labirinto Seoul

Estatísticas	Algoritmo proposto	*Micro Mouse*
Únicas células atravessadas	224	218
Cél. p/ chegar ao centro (1ª vez)	293	292
Curvas (na primeira corrida)	164	164
Células visit. na melhor corrida	98	98
Curvas (melhor caminho)	60	60
Cél. atrav. p/ comp. a melhor corrida	695	694
Curvas (p/ completar a melhor corrida)	388	398

6 Conclusões

Este trabalho teve como objetivo principal propor um novo algoritmo para resolver um labirinto com um robô móvel denominado Micromouse e que utilize a menor quantidade possível de memória RAM mantendo a eficiência do mesmo quando comparado ao algoritmo *Flood Fill* com expectativa de penetração de parede. O algoritmo desenvolvido com este fim foi denominado de *Flood Fill* com expectativa de penetração de parede não recursivo com otimização de memória. A estratégia otimiza o esquema de pilhas de memória para armazenar em somente uma estrutura de dados somente os caminhos de ida como de volta em uma corrida típica do Micromouse.

Por fim a eficiência do algoritmo proposto foi comparada à eficiência do *Flood Fill* com expectativa de penetração de parede não recursivo e demonstrou-se que, os algoritmos apresentaram eficiência equivalentes sendo que a estratégia de otimização de memória proposta deixa em vantagem os algoritmos proposto. A equivalência de eficiência entre os algoritmos pode ser vista a partir do caminho traçado por cada algoritmo em várias simulações propostas.

No algoritmo proposto, no traçado do caminho de volta, como já foi visto, as distâncias são redefinidas, uma vez que o novo alvo é a célula inicial (coordenadas 0,0). Porém, as informações das paredes descobertas ainda permanecem na memória, bastando apenas realizar varreduras para atualizar as distâncias. Isto é feito enquanto o robô permanece parado na célula de destino. Assim, quando as atualizações se completam, o robô poderá deslocar-se para as células de menor distância. O processo se repete quando o robô chega à célula de distância zero. Haverá redução de uso de memória RAM e um custo computacional maior, porém isto acontece enquanto o robô permanece parado e fora do tempo de corrida, não prejudicando o desempenho do robô em uma eventual competição. O algoritmo proposto é interessante para aplicações em sistemas embarcados tais como o Micromouse.

Agradecimentos

Agradecemos ao PET POTÊNCIA do curso de Engenharia Elétrica da UFPI.

Referências

Cai, Z., Ye, L. and Yang, A. (2012). Floodfill maze solving with expected toll of penetrating unknown walls for micromouse, *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pp. 1428–1433.

Dai, S., Zhao, B., Li, Z. and Dai, M. (2015). Design and practice from the micromouse competition to the undergraduate curriculum, *2015 IEEE Frontiers in Education Conference (FIE)*, pp. 1–5.

Li, X., Jia, X., Xu, X., Xiao, J. and Li, H. (2010). An improved algorithm of the exploring process in micromouse competition, *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, Vol. 2, pp. 324–328.

Lopez, G., Ramos, D., Rivera, K., del Valle, K., Rodriguez, A. and Rivera, E. I. O. (2015). Micromouse: An autonomous robot vehicle interdisciplinary attraction to education and research, *2015 IEEE Frontiers in Education Conference (FIE)*.

Sadik, A. M. J., Dhali, M. A., Farid, H. M. A. B., Rashid, T. U. and Syeed, A. (2010). A comprehensive and comparative study of maze-solving techniques by implementing graph theory, *2010 International Conference on Artificial Intelligence and Computational Intelligence*, Vol. 1, pp. 52–56.

Silva, S., Soares, S., Valente, A., Barradas, R. and Bartolomeu, P. (2015). A iniciativa micromouse e o estímulo ao saber tecnológico no ensino pré-universitário.

Solver (2013). *Maze-Solve*. Disponível em: <`http://code.google.com/p/maze-solver/`>. Acesso em 6 de julho de 2017.

Su, J. H., Huang, H. H. and Lee, C. S. (2013). Behaveior model simulations of micromouse and its application in intelligent mobile robot education, *2013 CACS International Automatic Control Conference (CACS)*, pp. 511–515.