



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE TECNOLOGIA
BACHARELADO EM ENGENHARIA ELÉTRICA

FRANCISCO MARCOLINO RODRIGUES FILHO

**ESTUDO E IMPLEMENTAÇÃO DE ALGORITMO INTELIGENTE
PARA RESOLUÇÃO DE LABIRINTO COM ROBÔ AUTÔNOMO
DIFERENCIAL MICROMOUSE**

TERESINA - PIAUÍ, AGOSTO/2017

FRANCISCO MARCOLINO RODRIGUES FILHO

**ESTUDO E IMPLEMENTAÇÃO DE ALGORITMO
INTELIGENTE PARA RESOLUÇÃO DE LABIRINTO COM
ROBÔ AUTÔNOMO DIFERENCIAL MICROMOUSE**

Trabalho de conclusão de curso apresentado
como requisito parcial para a obtenção do título
de Bacharel em Engenharia Elétrica, pelo curso
de Engenharia Elétrica da Universidade Federal
do Piauí - UFPI.

Orientador: Prof. Dr. Otacílio da Mota Almeida

TERESINA - PIAUÍ, AGOSTO/2017

R696e Rodrigues Filho, Francisco Marcolino.

Estudo e implementação de algoritmo inteligente
para resolução de labirinto com robô autônomo dife-
rencial Micromouse / Francisco Marcolino Rodrigues
Filho. – 2017.

105 f.

Monografia (Graduação) – Bacharelado em Engenharia
Elétrica, Universidade Federal do Piauí, 2017.
"Orientador: Otacílio da Mota Almeida"

1. Micromouse (Robótica). 2. Algoritmo Inteligente.
3. Labirinto. Flood Fill. 4. Robótica didática. I. Título.
CDD 629.892

FRANCISCO MARCOLINO RODRIGUES FILHO

**ESTUDO E IMPLEMENTAÇÃO DE ALGORITMO
INTELIGENTE PARA RESOLUÇÃO DE LABIRINTO COM
ROBÔ AUTÔNOMO DIFERENCIAL MICROMOUSE**

Trabalho de conclusão de curso apresentado
como requisito parcial para a obtenção do título
de Bacharel em Engenharia Elétrica, pelo curso
de Engenharia Elétrica da Universidade Federal
do Piauí - UFPI.

Orientador: Prof. Dr. Otacílio da Mota Almeida

Aprovado em 7 de julho de 2017.

BANCA EXAMINADORA

Otacílio d Mota Almeida

Prof. Otacílio da Mota Almeida, Dr.

José Maria Pires de Menezes Júnior

Prof. José Maria Pires de Menezes Júnior, Dr.

Fabiola Maria A. Linard

Prof". Fabiola Maria Alexandre Linard, Msc.

Agradecimentos

A Deus pelos caminhos que até aqui me trouxeram.

À minha família pelo apoio incondicional, pelo suporte sempre prestado e pelo amor a mim dedicado. Meus pais, mesmo à distância, não mediram esforços para me apoiar. Agradeço a eles pela confiança prestada.

Aos professores do curso que me ajudaram e guiaram nos momentos de incerteza.

Agradeço em especial ao professor orientador Otacílio da Mota Almeida pela ajuda nesta caminhada, que desde o início do curso me apoiou fortemente. Agradeço a ele, como Tutor, por ter me aceitado como membro do PET - Programa de Educação Tutorial, do curso de Engenharia Elétrica da UFPI, e por ter financiado o Micromouse por conta própria, que possibilitou este trabalho. Afirmo que minha graduação não seria a mesma sem ele.

Aos amigos do PET, à sala do PET, aos trabalhos do PET.

Aos colegas e amigos Fábio de Arruda Leda e José Genilson Sousa Carvalho, irmãos que ganhei durante a graduação. Sem vocês esta página estaria em branco. Agradeço aos colegas do curso pela amizade criada, e às mães dos meus colegas, que me adotaram nesta caminhada.

Resumo

Micromouse é uma competição robótica que teve início nos anos 70 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 partindo de um dos cantos até o seu centro, com a utilização de mini robô móvel (Micromouse), no menor tempo possível. O micromouse é um robô com sensores de distância e *encoders* conectados a um microcontrolador, permitindo que o robô se locomova numa direção predefinida por um algoritmo. Dentro do microcontrolador, o controle de trajetória, memória das paredes do labirinto e se situar no mesmo para saber se já chegou ao alvo são executados logicamente. 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 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. A implementação do algoritmo ocorreu da forma esperada, sendo que o percurso do robô num labirinto-teste é o mesmo do simulador. Portanto, este trabalho tem como objetivo a implementação do algoritmo inteligente de resolução de labirinto e do algoritmo de controle de trajetórias em robô micromouse, com características e requisitos necessários para participação em competições de desafios Micromouse.

Palavras-chave: Micromouse. Algoritmo inteligente. Labirinto. Flood Fill. Robótica didática.

Abstract

Micromouse is a robotic competition that began in the 1970s and is gaining momentum as high density integration technologies, sensors and intelligent control theory advance. The challenge is to solve a labyrinth starting from one of the corners to its center, using a mini mobile robot (Micromouse) in the shortest possible time. The micromouse is a robot with distance sensors and encoders connected to a microcontroller, allowing the robot to move in a predefined direction by an algorithm. Inside the microcontroller, the control of trajectory, memory of the walls of the labyrinth and to be in the same to know if already reached the target are executed logically. There are several labyrinth resolution algorithms with mobile robots, however, the most commonly used is textit Flood Fill. Based on this algorithm, this work proposes a new strategy for implementation of intelligent algorithm of labyrinth resolution. The idea is to use a data structure to store both the robot's way of going and the way back. In order to evaluate the performance of the algorithm, several simulations were performed and compared with the classic emph classic Flood Fill, resulting in a substantial saving of emph RAM that favors the real time implementation of the system embedded in the micromouse. The implementation of the algorithm occurred as expected, and the robot's path in a maze-test is the same as that of the simulator. Therefore, this work has as objective the implementation of the intelligent algorithm of labyrinth resolution and of the algorithm of control of trajectories in robot micromouse, with characteristics and requisites necessary for participation in competitions of challenges Micromouse.

Keywords: Micromouse. Intelligent algorithm. Maze. Flood Fill. Robotics.

Listas de ilustrações

Figura 1 – Robô <i>Moonlight Special</i> ficou em quarto lugar no campeonato de 1979, na cidade de Nova York.	3
Figura 2 – Torneio internacional Micromouse <i>half-size</i> no Japão	4
Figura 3 – Arranjo físico dos componentes principais do Micromouse	6
Figura 4 – <i>Flood Fill Expected Toll</i> - Com expectativa de penetração	10
Figura 5 – Esquema trajetórias <i>Flood Fill</i> clássico e diagonal	10
Figura 6 – Percurso em curva de 90 graus.	11
Figura 7 – Controlador Digital realimentado para robôs autônomos diferenciais	12
Figura 8 – Símbolos utilizados para impressões	13
Figura 9 – Diagrama de blocos do sistema robótico Micromouse	16
Figura 10 – Micromouses open source prontos para uso	17
Figura 11 – Trajetória do algoritmo Seguidor de parede	18
Figura 12 – Trajetória do algoritmo <i>Flood Fill</i>	19
Figura 13 – Vista superior do robô <i>uMaRT Lite Plus</i>	20
Figura 14 – Robô móvel diferencial - características	23
Figura 15 – Robô móvel diferencial - cinemática direta	24
Figura 16 – Interpretação do labirinto para o <i>Flood Fill</i>	27
Figura 17 – Numeração das distâncias sobre o labirinto	27
Figura 18 – Fluxograma básico de tomada de decisão	28
Figura 19 – Fluxograma básico do <i>Flood Fill</i>	29
Figura 20 – Fluxograma básico do algoritmo de atualização das distâncias não recursivo	30
Figura 21 – Exemplo de atualização da distância da célula	31
Figura 22 – Posição do robô na célula e seus estados	34
Figura 23 – Diagrama de blocos do controle PID em malha fechada	39
Figura 24 – Relé na malha realimentada	40
Figura 25 – Sistema Motor- <i>encoder</i> em malha aberta	41
Figura 26 – Relé - elemento não linear	42
Figura 27 – Diagrama de blocos do sistema para autossintonia dos parâmetros do Controlador PID - SISO	44
Figura 28 – Relé com histerese para $\varepsilon = 1$	45
Figura 29 – Curva de <i>Nyquist</i> para o sistema SISO do Micromouse	45
Figura 30 – Saída de velocidade utilizando os parâmetros PID do método Ziegler-Nichols	46
Figura 31 – Diagrama de blocos do sistema MIMO 2x2	47

Figura 32 – Diagrama de blocos do sistema MIMO proposto para o Micromouse com dois blocos PID em malha fechada	50
Figura 33 – Velocidades das rodas em uma curva	51
Figura 34 – Projeto de curvas	52
Figura 35 – Velocidade angular ideal x atual	53
Figura 36 – Visão geral do perfil de curva proposto	53
Figura 37 – O giro do robô em seu próprio eixo	54
Figura 38 – Visão geral dos perfis de curva para os quatro estados	56
Figura 39 – Os erros de posicionamento do Micromouse no Labirinto	57
Figura 40 – Sensores como feedback para correção da posição	58
Figura 41 – Parede à frente	59
Figura 42 – Retorno da função <code>getSensoresParede()</code> com obstáculos à frente	60
Figura 43 – EXCEL como ferramenta para modelagem de labirinto	62
Figura 44 – Modelagem do labirinto <i>Seoul</i> para simulação	63
Figura 45 – Acompanhamento e simulação do algoritmo proposto	64
Figura 46 – Percurso do robô no labirinto <i>Seoul</i> - primeira corrida <i>real</i>	65
Figura 47 – Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 1	66
Figura 48 – Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 3	66
Figura 49 – Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 5	67
Figura 50 – Labirinto <i>Seoul</i> após 3 corridas <i>reais</i>	67
Figura 51 – Diagrama de blocos do Método do relé sequencial para o sistema MIMO proposto	69
Figura 52 – Relé na malha de velocidade linear	70
Figura 53 – Relé na malha de velocidade angular	70
Figura 54 – Resultado da simulação do controle MIMO proposto	71
Figura 55 – Diagrama de blocos do controle implementado no <i>Simulink</i>	72
Figura 56 – Perfis de velocidades projetados	73
Figura 57 – Simulação da trajetória do robô utilizando os perfis de velocidade como referência para o controlador	74
Figura 58 – Perfis de velocidade e sinais de saída do controlador proposto	74
Figura 59 – Resultado do controle da malha de velocidade linear implementado no Micromouse	76
Figura 60 – Resultado do controle da malha de velocidade angular implementado no Micromouse	77
Figura 61 – Foto do primeiro teste do Micromouse sobre o tablado	78
Figura 62 – Projeto para contrução de labirinto	79

Figura 63 – Medidas de corte para as peças	79
Figura 64 – Labirinto-teste de 8x8 células desenhado no <i>EXCEL</i>	80
Figura 65 – Labirinto-teste de 8×8 simulado	80
Figura 66 – Labirinto montado para testes de 8×8 células	81
Figura 67 – Trajetória final do robô sobre o labirinto de 8x8 células	81
Figura 68 – Labirinto-teste de 16x16 células desenhado no <i>EXCEL</i>	82
Figura 69 – Trajetória final do robô sobre o labirinto-teste real de 16x16 células	83
Figura 70 – Simulação do percurso do robô no labirinto-teste proposto de 16x16 células	84

Listas de tabelas

Tabela 1 – Escolha do tempo de amostragem para sistemas de controle	41
Tabela 2 – Regras de ajustes estabelecidos por Ziegler-Nichols	44
Tabela 3 – Margem de ganho K_u e frequência crítica para relé com diversos ε . O valor de d do relé é 250.	45
Tabela 4 – Parâmetros PID para a malha de velocidade baseados pela tabela do Ziegler-Nichols	46
Tabela 5 – Parâmetros da função de transferência estimados para os motores (N = 200)	50
Tabela 6 – Máquinas de estado para geração dos perfis de reta e de curva . .	56
Tabela 7 – Controlador proporcional para o <i>feedback</i> dos sensores IR	58
Tabela 8 – Valor mínimo de tensão para detecção das paredes	61
Tabela 9 – Desempenho dos algoritmos no labirinto <i>Seoul</i>	68
Tabela 10 – Dados experimentais das saídas dos processos. Foram adotados $\varepsilon = 10$ e $d = 250$ para ambos os relés.	71
Tabela 11 – Parâmetros dos blocos PID estimados pelo Método do relé sequen- cial em simulação	71
Tabela 12 – Vetores utilizados para a simulação	73
Tabela 13 – Desempenho dos algoritmos no labirinto proposto	83

Lista de siglas e abreviaturas

CPU	<i>Central Processing Unit</i>
DC	<i>Direct Current</i>
DFS	<i>Depth First Search Algorithm</i>
IR	<i>Infrared</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
LED	<i>Light Emitting Diode</i>
Li-Po	Polímero de Lítio
LIFO	<i>Last Input First Output</i>
MDF	<i>Medium-Density Fiberboard</i>
MIMO	<i>Multi-Input, Multi-Output</i>
MQNR	Mínimos Quadrados Não Recursivos
PID	Proporcional, Integral e Derivativo
PWM	<i>Pulse Width Module</i>
QEI	<i>Quadrature Encoder Interface</i>
RAM	<i>Random Access Memory</i>
PCI	Placa de Circuito Impresso
PPR	Pulsos Por Revolução
RPM	Rotações Por Minuto
SISO	<i>Single Input, Single Output</i>
SMD	<i>Surface-Mount Device</i>
USB	<i>Universal Serial Bus</i>

Sumário

1	INTRODUÇÃO	1
1.1	Relevância e motivação	1
1.2	Descrição do trabalho	2
1.3	Revisão da literatura	2
1.3.1	História do Micromouse	2
1.3.2	As regras da competição Micromouse	4
1.3.3	Partes do projeto Micromouse	5
1.3.3.1	<i>Hardware</i>	5
1.3.3.2	Algoritmo	7
1.3.3.3	Controle	11
1.4	Metodologia	12
1.5	Objetivos	14
1.5.1	Objetivo geral	14
1.5.2	Objetivos específicos	14
1.6	Estrutura do trabalho	14
2	PROJETO HARDWARE E SOFTWARE DO MICROMOUSE	16
2.1	Introdução	16
2.2	Hardware - o robô Micromouse	19
2.2.1	Componentes do robô	19
2.2.2	Programação	21
2.2.3	Gravação	22
2.2.4	Comunicação	23
2.2.5	Cinemática do robô móvel diferencial	23
2.3	Algoritmo	25
2.3.1	Interpretação física e lógica do <i>Flood Fill</i>	25
2.3.2	Algoritmo <i>Flood Fill</i> não recursivo com otimização de memória	29
2.3.3	Construindo o algoritmo em linguagem C	32
2.3.3.1	Máquina de estados para o algoritmo	32
2.3.3.2	Funções auxiliares	34
2.3.3.3	Função para atualizar as paredes na estrutura de dados	35
2.3.3.4	Inicializando as distâncias das células para o alvo no centro do labirinto	36
2.3.3.5	Reinicializando as distâncias das células quando o alvo é realocado para a célula de partida	36
2.3.3.6	Algoritmo <i>Flood Fill</i> em C	36

2.4	Controle	38
2.4.1	Histórico	38
2.4.2	Determinando o tempo de amostragem do processo	40
2.4.3	Sintonia via Método do Relé	41
2.4.3.1	Método do Relé para sintonia do sistema SISO	44
2.4.3.2	Método do Relé Sequencial para sintonia de sistemas MIMO	46
2.4.3.3	Identificação dos motores via Mínimos Quadrados Não-Recursivos	48
2.4.3.4	Sistema realimentado proposto	50
2.4.4	Projeto de perfis de velocidade	51
2.4.4.1	Curva desejada	51
2.4.4.2	Determinando o perfil de curva de 90 graus	52
2.4.4.3	Determinando o perfil de curva para giro de 180 graus em seu próprio eixo	54
2.4.4.4	Implementando os perfis no Micromouse	55
2.4.5	Controle dos erros de posicionamento no labirinto	57
2.5	Observações finais	58
3	RESULTADOS EXPERIMENTAIS OBTIDOS NA SOLUÇÃO DO LABIRINTO	59
3.1	Hardware	59
3.1.1	Sensores de distância infravermelhos	59
3.1.2	<i>Encoders</i>	61
3.2	Algoritmo	61
3.2.1	Modelando um labirinto	61
3.2.2	Simulando o algoritmo no labirinto	63
3.3	Controle em malha fechada do Micromouse	68
3.3.1	Simulação do processo motor-encoder	68
3.3.1.1	Método do relé sequencial para sintonia do controlador PID	69
3.3.1.2	Simulação das trajetórias	72
3.3.2	Aplicando o controle no mini-robô Micromouse	75
3.4	Sistema completo	77
3.4.1	Construção e montagem do labirinto	78
3.4.2	Testes com labirinto de 8 x 8 células	79
3.4.3	Testes com labirinto de 16 x 16 células	82
4	CONSIDERAÇÕES FINAIS	86
4.1	Conclusão	86
4.2	Trabalhos Futuros	88
	REFERÊNCIAS	89

1 Introdução

1.1 Relevância e motivação

A ideia de criar um robô capaz de resolver labirinto surgiu por volta de 1950, construído por Claude Elwood Shannon, sendo que ele que construiu e apresentou pela primeira vez o labirinto original (SHANNON, 2016 apud BORGES, 2013, p. -7). O Micromouse, desde o início, foi planejado para ser um pequeno robô autônomo para percorrer um labirinto. Somente na década de 1970 que iniciou-se a primeira competição, quando o desempenho de vários robôs foi testado para mesma configuração de labirinto (LI et al., 2010).

Segundo Su, Huang e Lee (2013), estudantes de engenharia em geral (Elétrica, Computação, Mecânica, entre outros) se empenharam em participar destes campeonatos, uma vez que necessitam-se de pessoas competentes em programação, design de placas, projetos de *hardware*, entre outros. Nos países da Europa (Reino Unido, Portugal), Estados Unidos, Singapura e Japão estas competições são bastante populares com tais estudantes (SU; HUANG; LEE, 2013). O estudante de engenharia elétrica aprende, ao longo dos períodos, várias disciplinas, e acaba sendo desestimulado ao se deparar com a falta de prática, ou seja, o que está sendo aprendido em sala de aula às vezes não se acha aplicação útil. Às vezes os alunos preferem aulas mais interativas e querem saber por que o material fornecido na sala de aula é útil em sua carreira. O Micromouse contém a motivação interessante, uma vez que, segundo Silva et al. (2015), o projeto tem caráter multidisciplinar. O estudante aprende a importância de trabalhar com estudantes de outras concentrações de engenharia, o que lhe permite experimentar como uma carreira em futuro trabalho como engenheiro, servindo como experiência em coordenar projetos (LOPEZ et al., 2015a).

Portanto, a maior motivação de utilizar a modalidade robótica Micromouse é o seu caráter multidisciplinar da Engenharia Elétrica. Considerada ferramenta de estímulo ao ensino e aprendizado, segundo Silva et al. (2015), Lopez et al. (2015b), o projeto Micromouse é considerado mais que uma ferramenta para motivar o interesse em alunos na Automação e Controle por promover competências consideradas importantes para estas áreas do conhecimento. Lopez et al. (2015a) acrescentaram também que o mini robô móvel autônomo, Micromouse, é ideal para o estudante expandir seus conhecimentos em diversas áreas correlatas, além de relacionar trabalho em equipe.

1.2 Descrição do trabalho

O trabalho desenvolvido consistiu na implementação de algoritmo de resolução de labirinto *Flood Fill* e de controle de trajetórias em Micromouse, com objetivo de penetrar-se em qualquer labirinto real de quase $9m^2$ de área. A divisão da construção deste projeto será listado a seguir, e, brevemente, comentados:

- a) *hardware*: o chassi da plataforma robótica é uma placa de circuito impresso. Além de acomodar toda a parte elétrica com circuito SMD, comporta-se também os motores e sensores. É responsável por movimentar-se fisicamente pelo labirinto de forma inteligente;
- b) algoritmo: responsável por guardar na memória do robô as informações das células e o caminho otimizado, decidir e indicar a orientação da trajetória para a célula seguinte;
- c) controle: esta parte projeta o controle digital para os motores DC, de modo que as velocidades linear e angular sigam as referências pré-estabelecidas pelos perfis de curvas de velocidade.

1.3 Revisão da literatura

Esta seção apresenta uma breve revisão de pontos importantes para a compreensão do trabalho realizado. São tratados, de forma superficial, a história do Micromouse, as regras da competição, a compreensão física e lógica do algoritmo de resolução *Flood Fill*, e o *hardware* do robô. No fim, é apresentado o diagrama do sistema fechado do sistema rodas-encoder.

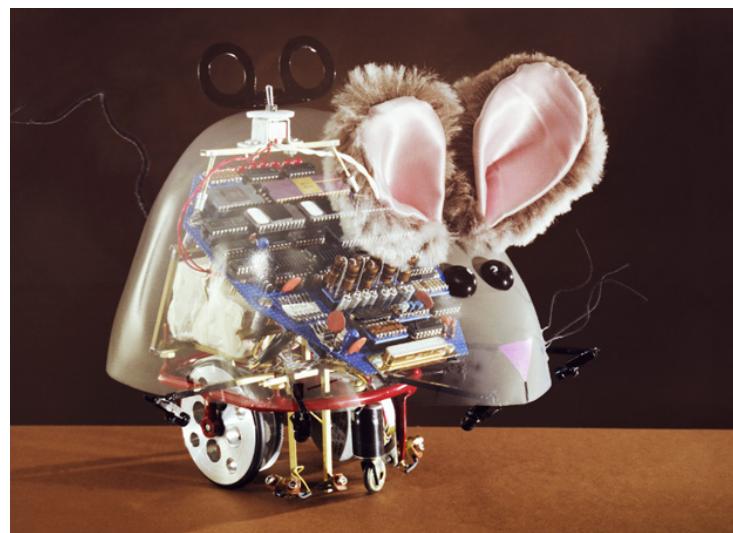
1.3.1 História do Micromouse

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 de largura, 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).

Há mais de 30 anos, Donald Christiansen, chefe da revista *IEEE Spectrum* à época, desafiou os seus leitores a projetar e construir um Micromouse para resolver labirinto. O robô teria sua própria lógica e memória autônomas e exploraria o labirinto que os próprios engenheiros editores do *Spectrum* desenhariam. A configuração do

labirinto seria mantida em segredo até o momento da competição. Cada Micromouse teria a oportunidade de investigar o labirinto nas corridas de teste, aprender com seus erros e, assim, poder melhorar seu tempo na corrida. Ele chamou o evento de *Amazing MicroMouse Maze Contest*. Cinco competidores se inscreveram, mas apenas dois atravessaram o labirinto. No entanto, em 1979, quinze Micromouses, dentre os quais o quarto colocado *Moonlight Special*, Figura 1, competiram com sucesso nas finais do *Spectrum* em 1979 *National Computer Conference* (CHRISTIANSEN, 2014).

Figura 1 – Robô *Moonlight Special* ficou em quarto lugar no campeonato de 1979, na cidade de Nova York.



Fonte: adaptado de Christiansen (2014)

As finais foram cobertas pela televisão CBS, NBC e ABC e foram relatadas nos noticiários noturnos de Walter Cronkite, John Chancellor e David Brinkley. Recortes de imprensa empilhados de uma ampla gama de jornais, do *International Herald Tribune* ao *Booneville Daily News*, do Missouri. Um Micromouse vencedor foi retratado na primeira página do *The Wall Street Journal* (CHRISTIANSEN, 2014, p. -8).

Após alguns anos, o evento tornou-se mundial. Em 1980, a primeira competição europeia foi realizada em Londres, seguida de um ano depois em Paris. O Japão anunciou o primeiro *World Micromouse Contest*, em 1985. Christiansen (2014) foi convidado a julgar a primeira competição Micromouse em Singapura, em 1987. Neste mesmo ano, o Instituto de Engenharia e Tecnologia organizou uma competição internacional em Londres (CHRISTIANSEN, 2014).

Os Micromouses pioneiros eram desengonçados em aparência. Os centros de gravidade eram elevados, com pouca estabilidade. Alguns eram tão altos que tombavam nas curvas. Outros se pareciam mais com aves que ratos. Segundo Christiansen

(2014), *Monty Mouse*, da Grã-Bretanha, parecia um sanduíche de alumínio sobre rodas. As regras da competição da época previam limites de largura e altura além das larguras das células (CHRISTIANSEN, 2014).

À medida que a tecnologia avançou, os robôs se modernizaram 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 (CHRISTIANSEN, 2014). 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.

Hoje estima-se que são realizados cerca de 100 concursos todo ano. Muitos são patrocinados por universidades e por regiões do IEEE. No Japão, em março de 2014, foi realizada a 35^a edição do concurso *All-Japan Micromouse Robot*. Outras competições tradicionais continuam em Mumbai e em Birmingham, Inglaterra (CHRISTIANSEN, 2014).

Considerando as características acima, no Japão, a competição avançou de tal maneira que uma modalidade de labirinto de 32x32 células foi introduzida (SU; HUANG; LEE, 2013). Na Figura 2, é mostrada a competição naquele país.

Figura 2 – Torneio internacional Micromouse *half-size* no Japão



Fonte: adaptado de Su, Huang e Lee (2013)

1.3.2 As regras da competição Micromouse

A função básica do Micromouse é sair de um dos cantos do labirinto de 16x16 células, de 324 cm² cada, e chegar ao centro do mesmo. Isto se chama corrida. É contado este tempo. O tempo da corrida do centro do labirinto à célula de partida é

desconsiderado. O Micromouse 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 para a primeira corrida. Em algumas modalidades, Micromouse tem até 15 minutos para fechá-la. Dentro da janela de tempo o Micromouse poderá tentar várias vezes a corrida com o objetivo de diminuir seu tempo (LI et al., 2010). A pontuação é baseada tanto na execução mais rápida quanto no tempo total acumulado para todas as corridas. Os donos do Micromouse, de forma alguma, não podem ter nenhum tipo de comunicação com os robôs (CHRISTIANSEN, 2014).

O Micromouse 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):

- a) estabilidade Mecânica e Elétrica;
- b) velocidade mecânica e de processamento de informação;
- c) precisão nos movimentos e sistema de sensores;
- d) capacidade de armazenamento eficiente do caminho mais curto;
- e) habilidade de resolver, utilizando inteligência computacional, qualquer labirinto regulamentar.

1.3.3 Partes do projeto Micromouse

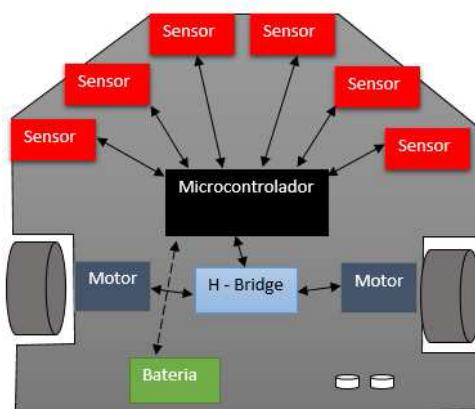
A seguir são apresentadas as partes primordiais para o projeto de um Micromouse. A harmonia perfeita destes sistemas deixa o mini robô móvel pronto para competições desta modalidade. Eles são responsáveis por dar autonomia ao robô de realizar o trajeto de menor caminho no labirinto de forma inteligente.

1.3.3.1 *Hardware*

A Figura 3 mostra o esquemático geral das conexões dos componentes do Micromouse. Todos os projetos de *hardware* e *design* dos robôs pesquisados na literatura para Micromouse são baseados no arranjo mostrado. Basicamente, em geral, um robô possui microcontrolador, sensores de distância, *encoders*, minimotores de corrente contínua, rodas e bateria para alimentação. O cérebro do robô, o microcontrolador, será capaz de verificar obstáculos à frente com os sensores de distância, tomar decisões e atuar ajustando as velocidades dos motores. A autonomia é garantida pela bateria. Análogo ao sistema humano, o cérebro processa as informações vistas

pelos sensores (olhos), processa-as e toma alguma atitude (motores). Então, o robô recebe informações dos sensores e processa-as de forma a comandar corretamente os motores (YADAV; VERMA; MAHANTA, 2014).

Figura 3 – Arranjo físico dos componentes principais do Micromouse



Fonte: adaptado de Borges (2013)

O Micromouse deve ser uma plataforma autônoma e inteligente, capaz de se locomover em ambientes estreitos de forma rápida. Por isto, segundo Torres e Ramirez (2016), Caicedo e Trujillo (2016), a tendência de construção seguida para os robôs tentam ser as seguintes:

- a) o chassi do robô, normalmente, é a própria placa de circuito impresso, que, por sua vez, tem altura baixa para garantir um centro de gravidade baixo e, consequentemente, maior estabilidade nas curvas;
- b) rodas pneumáticas em cada lado, de forma a aumentar a aderência no piso do labirinto;
- c) pneus de borracha;
- d) motores elétricos, são preferidos os motores DC a motores de passo. No entanto, nos últimos anos os motores sem núcleo ganharam popularidade;
- e) *encoderes* para cada eixo dos motores, com resolução menor que 1 grau por pulso;
- f) sensores de distância, onde os mais populares são os sensores ópticos infravermelho (IR);
- g) em geral, se usam quatro ou mais sensores ópticos IR em um robô;

- h) baterias Li-PO estão em moda, tendo melhor relação entre a capacidade de carga armazenada, peso e tamanho. São bastante usados em drones por terem estas características;
- i) os microcontroladores mais utilizados são os de 32 bits, com frequência de CPU elevada. Os favoritos são das séries STM32F4 da ST;
- j) sensores como acelerômetros, giroscópios ou bússolas garantem giros precisos de 45, 90 ou 180 graus.

1.3.3.2 Algoritmo

O algoritmo é a parte mais importante para o projeto Micromouse. As direções necessárias para o robô percorrer o menor caminho são obtidas através do algoritmo. Por isso que simpatizantes e fãs deste tipo de robô tentam criar algoritmos mais eficientes a fim de ganhar o campeonato.

A Teoria dos Grafos aparece como uma ferramenta eficiente ao projetar técnicas de resolução de labirinto proficientes. Ao incorporar um procedimento inteligente com os algoritmos de teoria gráfica existentes, alguns algoritmos para Micromouse foram desenvolvidos (SADIK et al., 2010).

O projeto proposto do algoritmo tem como objetivo resolver qualquer labirinto desconhecido, além de ter características otimizadas de consumo de memória, uma vez que circuitos embarcados não possuem grandes quantidades de *RAM*.

Ao buscar referências bibliográficas e arquivos que contenham tais algoritmos inteligentes para resolução de labirinto, foi encontrado um simulador com vários algoritmos já implementados. No *Micro Mouse Maze Editor and Simulator* (SOLVER, 2013), uma ferramenta em Java que mostra como o robô se comporta em diversos labirintos - inclusive alguns já foram labirinto oficial de competição, são encontrados quatro algoritmos. São eles:

- a) **Left Wall Follower**: seguidor de paredes à esquerda;
- b) **Right Wall Follower**: seguidor de paredes à direita;
- c) **Treumax**: algoritmo força bruta - visita todas as células do labirinto;
- d) **Flood Fill**: o mais utilizado - simples e eficiente para este tipo de competição.

Os seguidores de parede são os algoritmos mais simples. Pelo fato de existir vários caminhos, ele segue somente o caminho que foi programado, seja à esquerda

ou à direita. Segundo Yadav, Verma e Mahanta (2012), algoritmos deste naipe não conseguem lograr êxito em labirintos complexos.

O seguidor de parede é das regras mais conhecidas, também conhecida como a regra da mão esquerda ou direita. Se o labirinto é simplesmente conexo, isto é, todas as suas paredes estão ligadas entre si ou com limite exterior do labirinto, o algoritmo processa-se mantendo uma mão em contato com uma das paredes do labirinto, onde o jogador está a garantir que não se perde e vai chegar ao centro caso haja uma ligação da parede exterior ao centro. Caso contrário, vai voltar para a entrada após percorrido os corredores do labirinto. Se as paredes estão ligadas, o Micromouse pode ser enganado e fazer um loop ou círculo, não chegando assim ao objetivo. (MISHRA; BANDE, 2008 apud BORGES, 2013, p. -21–22)

O algoritmo Tremaux, na literatura conhecido como *DFS - Depth First Search Algorithm*, é um algoritmo que é eficaz, pois resolve muito bem o labirinto, mas o inconveniente dele é o fato de ele percorrer todas as células do labirinto (YADAV; VERMA; MAHANTA, 2014). Como o próprio nome sugere, é um algoritmo de pesquisa profunda. É também considerado um algoritmo de Teoria dos Grafos, como o *Flood Fill*.

No DFS, a partir da raiz do Grafo, o robô desbrava o labirinto até encontrar um beco sem saída, ou seja, até a região mais profunda da árvore ele tenta seguir. Algoritmos começam a procurar a partir de um vértice específico e, em seguida, navega no grafo, ramificando os vértices correspondentes até chegar ao ponto final ou de destino. Este algoritmo de busca pode ser eficientemente usado na resolução de labirintos Micromouse.

O labirinto inteiro é mapeado como um gráfo onde os nós ou vértices são considerados células de labirinto. Começando a corrida da célula inicial à célula destino, os robôs visitam todas as células, visitando cada entrada aberta das células uma vez em cada direção. Os robôs continuam a pesquisar as células até chegarem à célula de destino. Em vez de marcar cada célula, ele acompanha as paredes celulares. Na corrida inicial, todas as portas da célula original são desmarcadas.

O algoritmo Trémaux, inventada por Charles Trémaux, é um método eficiente para encontrar a saída de um labirinto que requer o desenho de linhas no chão para marcar um caminho e, é garantido que funcione para todos os labirintos que têm passagens bem definidas. Cada célula do caminho terá um estado de não visitado, marcado uma vez ou marcado duas vezes. Cada vez que um sentido é escolhido, é marcado pelo desenho de uma linha no chão. No início é escolhida uma direção aleatória, caso haja mais do que um. Ao chegar a um cruzamento que não foi visto antes, ou seja sem marcas, escolhe-se uma direção aleatória marcando o caminho. Ao chegar a um cruzamento marcado e se o caminho atual é marcado apenas uma vez, de seguida, vira-se e caminha de volta marcando o caminho uma segunda vez. Se este não

é o caso, escolhe a direção com o menor número de marcas. Quando finalmente chegar ao centro, os caminhos marcados exatamente uma vez vão indicar um caminho direto de volta para o início. Se não houver saída, este método irá levá-lo de volta ao começo, sendo que neste caso, cada caminho estará marcado exatamente duas vezes, uma vez em cada direção (SNAPP, 2010 apud BORGES, 2013, p. -22).

O trabalho de Sadik et al. (2010) realizou uma comparação entre Tremaux e *Flood Fill*, dois algoritmos surgidos a partir da Teoria dos Grafos, onde mostrou a vantagem do algoritmo de imundação em dois dos três labirintos em descobrir o melhor caminho, explicando o motivo de os competidores preferirem o *Flood Fill* por ele ter a característica de poder encontrar o menor caminho atravessando o número mínimo de células. Afirma-se também que tanto o Tremaux quanto o *Flood Fill* requer grande poder de processamento. O espaço de memória necessário também é muito alto em *Flood Fill*.

A ideia do *Flood Fill* é simples. O *Flood Fill* é um algoritmo que cria uma matriz pré-programada com as dimensões do labirinto, sendo que os números são colocados de uma forma que o labirinto se pareça com um redemoinho, o início tem a numeração mais alta, e o final (chegada) é numerado com zero. O algoritmo somente tenta distribuir as distâncias para cada célula indicando o número de passos para o destino.

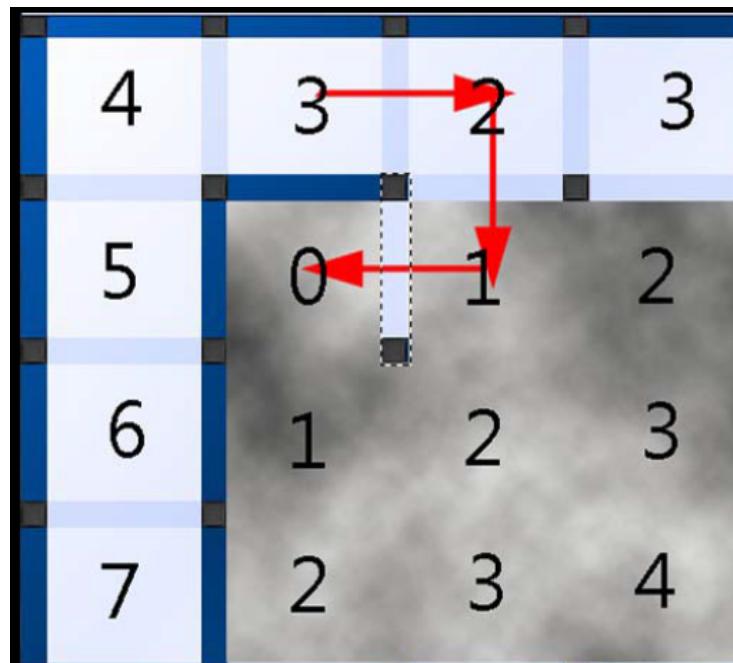
Quando a configuração do labirinto é desconhecido, o mesmo tenta seguir um *atälho* considerando-o como se não existissem paredes, ou seja, sempre tenta seguir um percurso otimista quando não há informações suficientemente. Se existe um caminho mais curto do que o feito, ele já tentou passá-lo. No entanto, às vezes o percurso o leva a caminhos mais longos que o ótimo por ele ter se enganado *nestes caminhos curtos*, e paredes descobertas pelos sensores fazem com que o caminho escolhido na verdade não é o ótimo (CAI et al., 2010). A ideia desse algoritmo é percorrer as células do maior para o menor valor, uma vez que cada célula tem a distância para o destino (YADAV; VERMA; MAHANTA, 2012).

Dos algoritmos apresentados, segundo Silva et al. (2015), o mais utilizado e um dos mais eficientes é o *Flood Fill* clássico. Ele é baseado em tornar a superfície do tablado em *relevo*, dando números para cada célula. A partir disso, o robô tende a descer esta *superfície* até o ponto mais baixo, em forma de redemoinhos, como faz a água no rio em seu curso natural. 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).

Já Cai, Ye e Yang (2012) modificou o algoritmo *Flood Fill* para ter melhor performance do que o tradicional. Segundo ele, o algoritmo *Flood Fill Expected Toll of Penetrating*, assim designado, ele já possui uma inteligência a mais. Ele tenta prever se nas próximas células desconhecidas se há possibilidade de ter uma possível parede,

e assim, ele não terá que ir até a célula para verificar isso (Figura 4). Esta inteligência reduziu os tempos de curvas e números de visitas de células do labirinto (CAI; YE; YANG, 2012).

Figura 4 – Flood Fill Expected Toll - Com expectiva de penetração

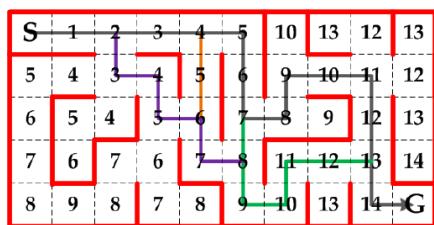


Fonte: adaptado de Cai, Ye e Yang (2012)

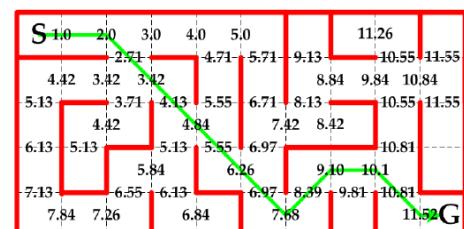
Porém, nas competições mais atuais, ao realizar corrida, os robôs percorrem em *diagonal* ao invés de realizar contornos em forma de zigue-zague, como mostra a Figura 5. Com isto, os robôs evitam perder velocidade nas curvas, o que melhorou e muito o tempo de corrida (SU; HUANG; LEE, 2013). Este algoritmo é o *diagonal Flood Fill*, considerado melhor, porém não será abordado neste trabalho.

Figura 5 – Esquema trajetórias Flood Fill

(a) *Flood Fill Clássico*



(b) *Flood Fill Diagonal*



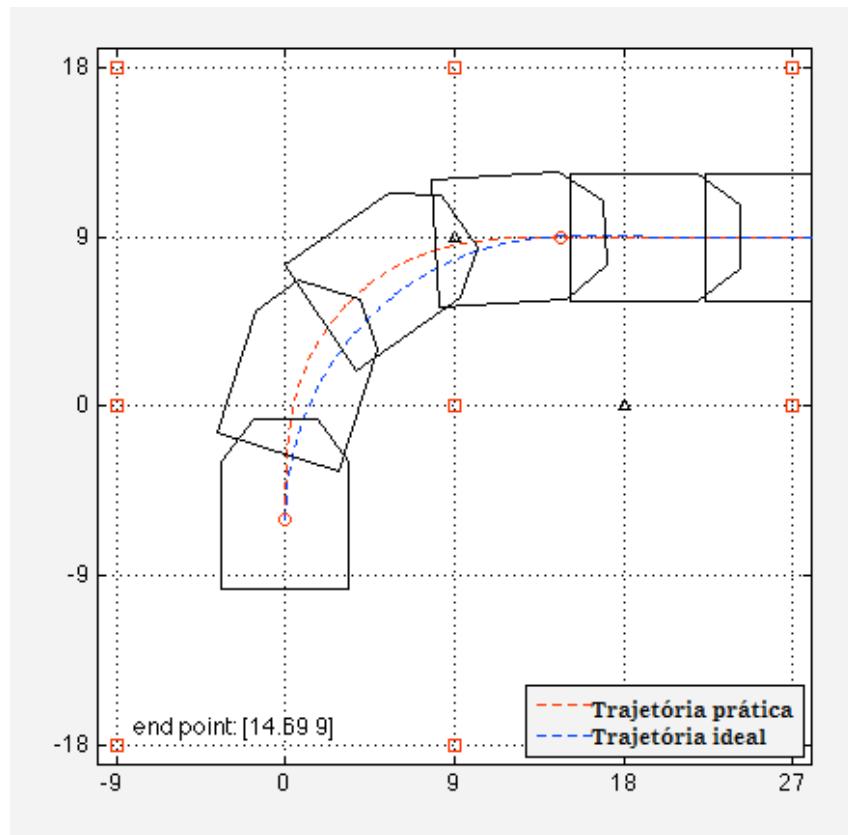
Fonte: adaptado de Su, Huang e Lee (2013)

1.3.3.3 Controle

Além da eficiência do algoritmo em localizar o melhor caminho, é necessário controle total do seu percurso no labirinto. Ambos devem andar em sintonia perfeita na sua execução.

Como o robô geralmente é móvel diferencial, pode-se prever sua trajetória através das fórmulas cinemáticas (1.1) e (1.2), onde v_R é a velocidade da roda direita e v_L é a velocidade da roda esquerda e L é a distância entre as rodas. As velocidades das rodas estão diretamente ligadas com as velocidades linear (v_C) e angular (ω_C) global do robô. Então, as informações digitais dos *encoders* são usadas para controlar o movimento da trajetória do Micromouse (SU; HUANG; LEE, 2013).

Figura 6 – Percurso em curva de 90 graus.



Fonte: adaptado de Su, Huang e Lee (2013)

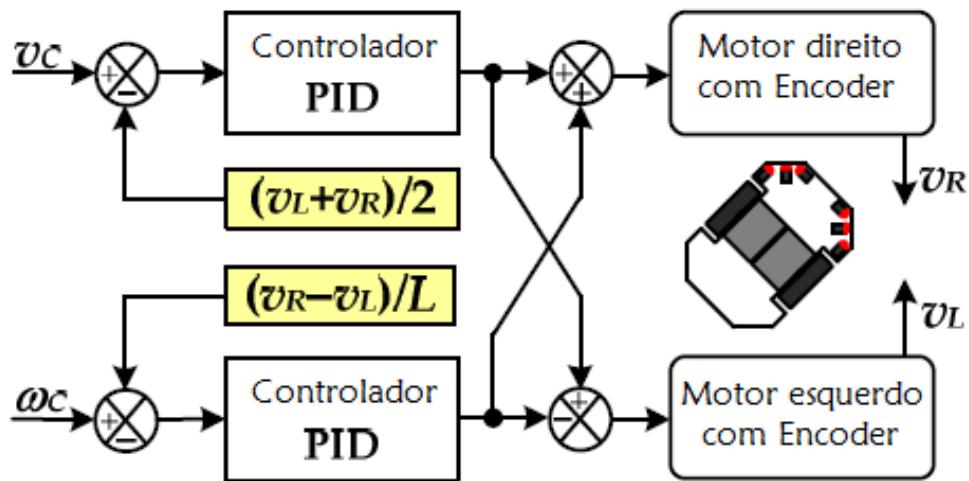
$$v_C = (v_R + v_L)/2 \quad (1.1)$$

$$\omega_C = (v_R - v_L)/L \quad (1.2)$$

Com o raio da curva é conhecido, e predefinindo valores de velocidade linear, e que o robô realizará uma curva de 90 graus em uma célula, pode-se traçar um perfil de curva. Um exemplo é mostrado na Figura 6.

Su, Huang e Lee (2013) sugerem criar um controlador digital (Figura 7) com dois *set points*: velocidade linear e angular. Quem os alimenta são os *speed profiles*, que podem ser simulados no *EXCEL* ou *MATLAB*. O controlador será responsável por aplicar as velocidades das rodas de maneira tal que o robô tenha realmente ambas as velocidades do *speed profile*. O sistema será realimentado através das informações dos *encoders* e os controladores PID (Proporcional, Integral e Derivativo) darão os *duty cycle* dos sinais PWM aos motores de corrente contínua (DC). A Figura 7 mostra melhor o esquema de controle. Assim, o robô obedecerá a trajetória de Figura 6.

Figura 7 – Controlador Digital realimentado para robôs autônomos diferenciais



Fonte: adaptado de Su, Huang e Lee (2013)

1.4 Metodologia

O estudo realizado consistiu no projeto das três partes necessárias, utilizando como ponto de partida as observações feitas em algoritmos de resolução de labirinto *Flood Fill*, que foi baseado nos trabalhos de Yadav, Verma e Mahanta (2012) e Cai, Ye e Yang (2012).

O algoritmo de otimização proposto, implementado em C, utiliza um conjunto de símbolos que permite a análise do percurso de rota do robô no labirinto. Os símbolos são úteis em um processo de depuração de corridas, a fim de validar a eficácia do algoritmo. A seguir são mostrados alguns símbolos básicos essenciais para depura-

ções. Os símbolos | e --- significam as paredes nos vértices das células, sendo que o caractere + simboliza as quinas das células. Os números em cada célula significam o número de passos para o destino. Na célula onde o robô se encontra, os símbolos V, ^, > e < indicam o sentido, e o símbolo * nas células mostram onde o robô já visitou. A Figura 8 mostra um exemplo para utilização dos símbolos.

Figura 8 – Símbolos utilizados para impressões

```
/*
O mouse e o labirinto serão mostrados desta forma:

+---+---+---+
|*5  *4  V3 |
+---+---+   +
| 0  1  2 |
+---+---+---+

'+' para demarcar os vértices da célula.
'---' para demarcar as paredes Norte/Sul.
'|' para demarcar as paredes Leste/Oeste.
'*' para células visitadas.
Usa-se <, >, ^, V para a direção do Micromouse.

*/
```

Fonte: autor (2017)

Já as constantes do controle PID do robô foram extraídas a partir do *Método do Relé Sequencial*, tendo como prioridade a maximização do tempo de assentamento, visto que quanto mais rápido chegar nas velocidades de referência, a trajetória em curvas se torna próxima à trajetória ideal (Figura 6).

O desempenho do controlador para as rodas do Micromouse foi verificado no ambiente Simulink do *MATLAB*. Após a realização da identificação do sistema robô-rodas do Micromouse, através do Método dos Mínimos Quadrados Não Recursivos (MQNR), foi construído um modelo de controle das velocidades das rodas com base no esquema de controle proposto por Su, Huang e Lee (2013) e pode-se verificar as trajetórias em curvas e retas ao variar as referências de velocidades linear e angular.

Após a verificação das partes de forma individual, foram realizados testes com todos as partes em conjunto em um labirinto de testes, contruído a partir de placas de

MDF, a fim de verificar o comportamento do mini robô em um labirinto real.

1.5 Objetivos

A seguir são apresentados os objetivos desejados com o estudo realizado. É apresentado, inicialmente, o objetivo geral deste trabalho e, em seguida, listam-se objetivos específicos que se desejam alcançar.

1.5.1 Objetivo geral

Estudar, desenvolver, simular e implementar, juntamente com o controle de velocidades, o algoritmo inteligente de resolução de labirinto em mini robô Micromouse. Desenvolver o projeto do mini robô autônomo diferencial Micromouse.

1.5.2 Objetivos específicos

Os objetivos específicos que se deseja alcançar com o trabalho são os seguintes:

- a) Aplicar os conhecimentos adquiridos em programação no curso de Engenharia Elétrica para desenvolver algoritmo de resolução de labirinto, na linguagem C, para facilitar a inclusão do mesmo em quaisquer microcontroladores;
- b) Desenvolver o controle digital de velocidade linear e angular de forma que o robô tenha facilidade em andar em linha reta e em curvas.
- c) Tornar o Micromouse autônomo e inteligente o suficiente a descobrir o menor caminho em labirinto desconhecido real.

1.6 Estrutura do trabalho

Este trabalho é estruturado em quatro capítulos, conforme as descrições que seguem:

- a) **Capítulo 01:** introdução acerca do tema tratado, motivação do estudo e contextualização do tema na sociedade atual;
- b) **Capítulo 02:** neste capítulo são realizadas análises específicas sobre cada uma das partes implementadas e são apresentadas as informações presentes na literatura sobre eles;

- c) **Capítulo 03:** apresenta as partes projetadas e os resultados observados em suas simulações e em labirinto, bem como observações pertinentes acerca desses resultados;
- d) **Capítulo 04:** são apresentadas, sucintamente, as principais conclusões observadas no trabalho, além de sugestões para futuros trabalhos nessa linha.

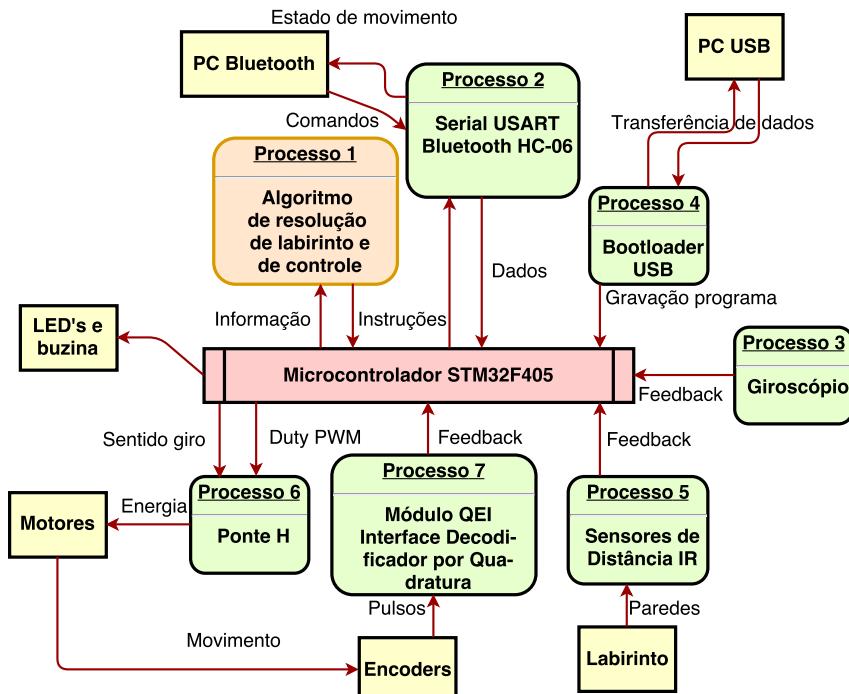
2 Projeto *hardware* e *software* do Micromouse

2.1 Introdução

O projeto de uma solução para atender todo o requisito de uma competição Micromouse exige atenção tanto no projeto *hardware* como no *software*: ambos em sintonia perfeita. Para tanto, no quesito *hardware*, é necessário que o cérebro microcontrolador do Micromouse tenha boa velocidade na execução do algoritmo, boa RAM (ao menos 4 KB de RAM), com módulos específicos QEI para decodificação de pulsos por quadratura dos *encoders* por *hardware*, e baixo consumo de energia. Já no quesito *software*, o algoritmo deve ser otimizado para o consumo mínimo de RAM, uma vez que o mesmo depende bastante de estruturas de dados e pilhas para executá-lo.

Na Figura 9 é apresentado o diagrama do projeto proposto. Pode-se perceber que todos os processos são controlados pelo microcontrolador, o cérebro do robô. No diagrama, são observados módulos de *hardware* e de *software* essenciais.

Figura 9 – Diagrama de blocos do sistema robótico Micromouse



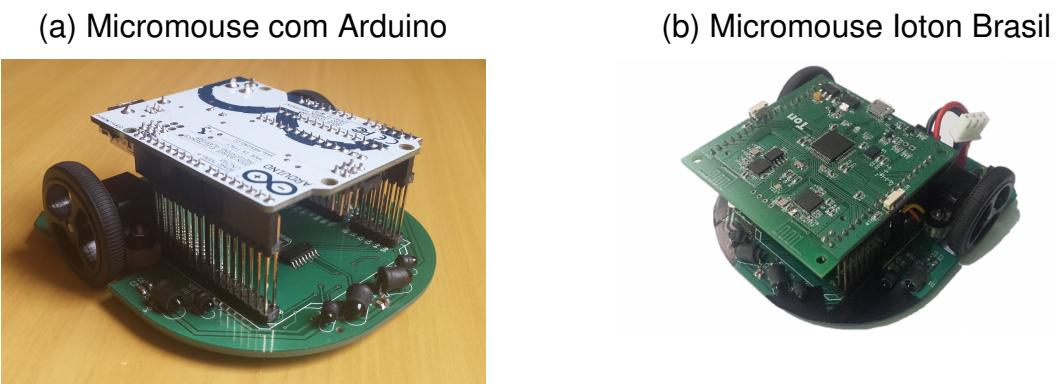
Fonte: autor (2017)

Todos os processos mostrados na figura, (exceto o processo 1), fazem parte do microcontrolador do robô e do chassi do mesmo, sendo estes módulos projetos de *hardware*. A implementação das partes de algoritmo para resolução de labirinto e para controle de trajetórias está representada como instruções de programa (processo 1). A requisição de dados de *feedback* dos *ecoders* (processo 7), do giroscópio e das paredes (processos 3 e 5), envio de dados por *bluetooth* (processo 2), e controle dos Motores (processo 6) são controlados pelo processo 1. As setas mostram os fluxos de dados entre os módulos e o microcontrolador.

O projeto foi dividido em três partes: *hardware*, algoritmo e controle. Em todos os robôs de Kibler et al. (2011), Yadav, Verma e Mahanta (2012), Lopez et al. (2015a), Su, Huang e Lee (2014), Lin (2010), Dai et al. (2015) utilizaram-se destas partes. No entanto, embora a parte de *hardware* esteja como parte do projeto proposto, somente são citados os componentes do robô adquirido, e não foi construído por este trabalho.

Existem diversos projetos *open source* prontos para uso para competições deste tipo e para competições para seguidores de linha. Hoje, os fãs da competição não precisam mais construir os seus robôs a partir do zero. Peças, kits e projetos *open source* completos estão disponíveis no mercado (CHRISTIANSEN, 2014). Por exemplo, a Figura 10 mostra o chassi Micromouse para uso com placas Arduino e é comum de se achar no mercado. Para este trabalho, foi adquirido o projeto *uMaRT Lite Plus* de Figura 13.

Figura 10 – Micromouses didáticos disponíveis no mercado

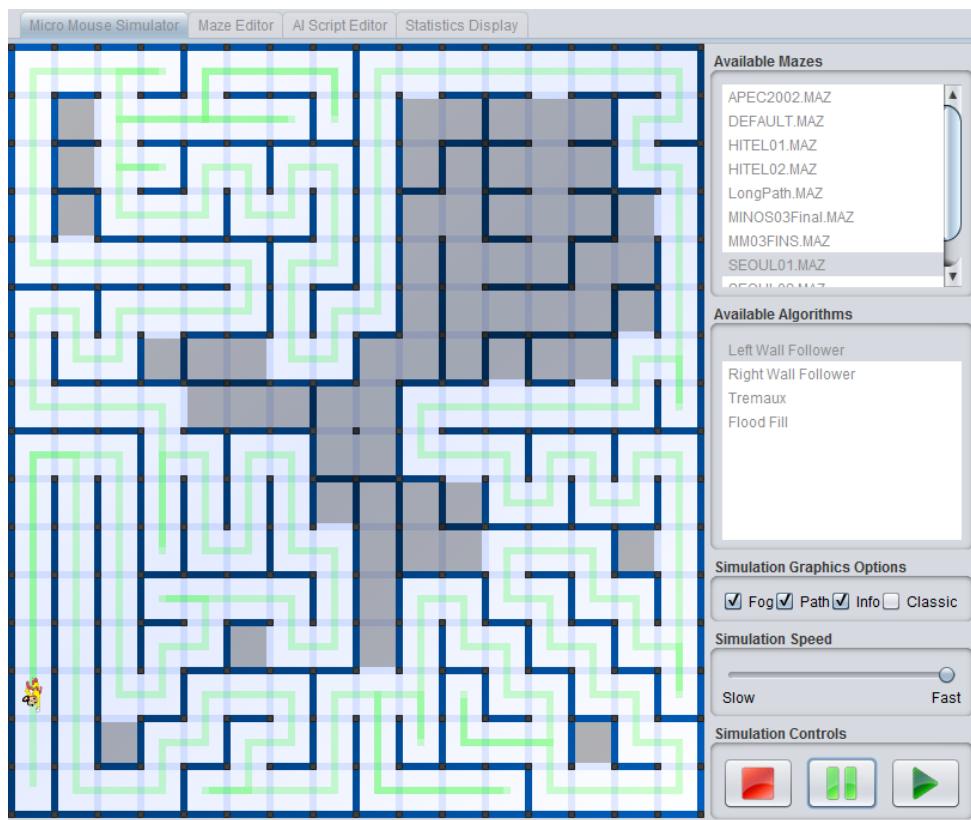


Fonte: autor (2017)

Quanto ao algoritmo de resolução de labirintos (Segunda parte), na literatura, é comum o uso do algoritmo de imundação *Flood Fill*. Dentre os três algoritmos mais estudados (Seguidores de parede, Tréumax e *Flood Fill*), o algoritmo preferido por todos consegue resolver perfeitamente qualquer que seja o *design* do labirinto, enquanto que os algoritmos mais simples não o fazem, conforme pode se observar nas Figuras

11 e 12 a comparação entre ambos os algoritmos para labirinto que não tem paredes ligadas entre si até o centro (as células em cinza não foram visitadas - o seguidor de parede não resolve o labirinto). Portanto, a justificativa para o desenvolvimento, simulação e implementação do algoritmo *Flood Fill* para este projeto é a eficiência na resolução de qualquer configuração de labirinto em uma eventual competição.

Figura 11 – Trajetória do algoritmo Seguidor de parede



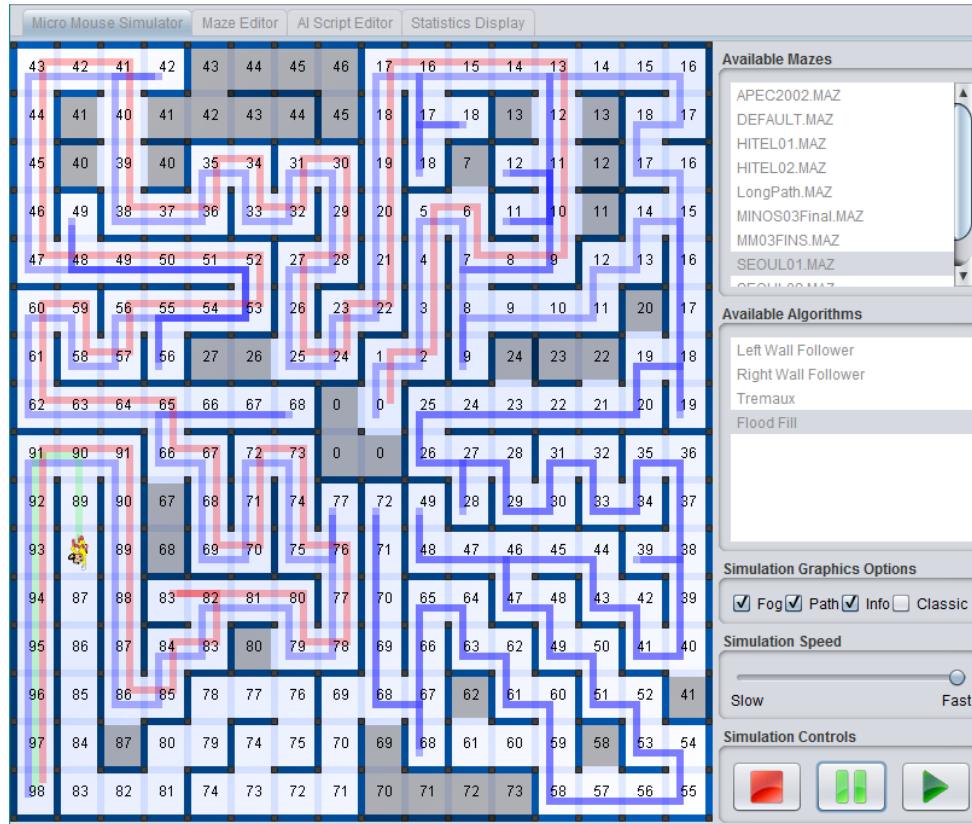
Fonte: autor (2017)

Já a terceira parte é o projeto de controlador de velocidades das rodas, em malha fechada. Como o sistema não é ideal, o robô pode sofrer interferências externas, como ruído, deslizamento das rodas, entre outros, de forma que a trajetória projetada para os perfis de reta e curva se altere em *malha aberta*, ou seja, o robô poderá sair da trajetória e perder sua referência no labirinto. Uma das formas para correção deste problema, outro controle realimentado foi construído, sendo que a sua realimentação é a informação do giroscópio e/ou dos sensores de distância. A construção de perfis de curvas, e integração dos sensores de distância e *encoders*, é de fundamental importância, a fim de atuar corretamente os motores de forma que siga a trajetória pré determinada pelo algoritmo.

Portanto, o sucesso em uma eventual competição desta modalidade depende

da construção das partes citadas.

Figura 12 – Trajetória do algoritmo Flood Fill



Fonte: autor (2017)

2.2 Hardware - o robô Micromouse

O robô tem como objetivo encontrar o centro do labirinto, onde estão as quatro células-alvo. Após esta corrida, o robô deve mapear o melhor caminho para melhorar seu tempo de corrida no sentido de alcançar o menor tempo possível, sendo que o ganhador é o robô que fez o menor tempo de corrida da célula de partida até as células-alvo.

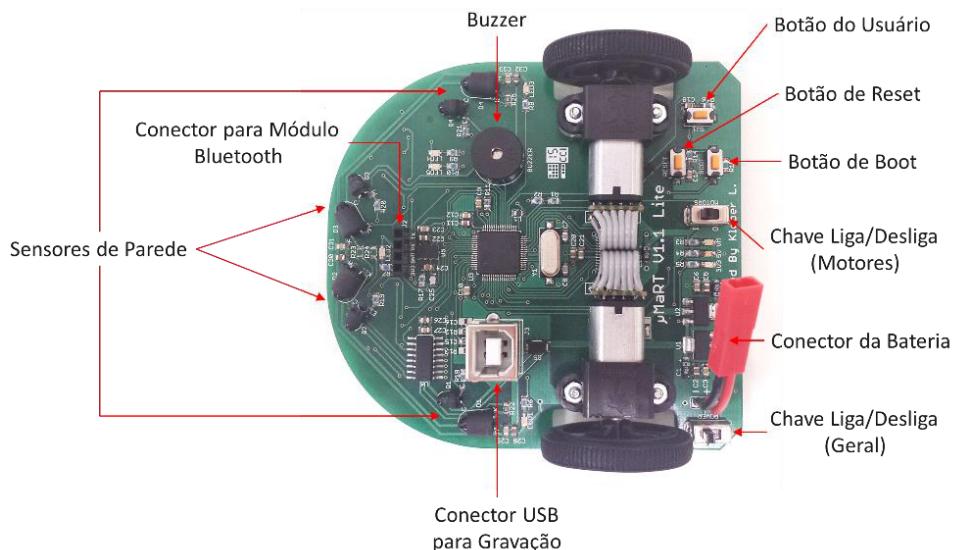
2.2.1 Componentes do robô

A perspectiva de todo o sistema pode ser visto pela Figura 3, onde mostra todos os componentes necessários para o robô operar de forma satisfatória num labirinto. O robô tem um microcontrolador conectado com os periféricos sensoriais e motores. O cérebro do robô, o microcontrolador, será capaz de verificar obstáculos à frente com os sensores de distância, tomar decisões e atuar ajustando as velocidades dos motores.

A autonomia é garantida pela bateria. Então, o robô recebe informações dos sensores e processa-os de forma a comandar corretamente os motores.

A plataforma robótica utilizada é ideal para desenvolvimento de projetos de solucionadores de labirinto. O robô denominado *uMaRT Lite Plus*, de Figura 13, é um kit educacional de robótica destinado ao aprendizado e utiliza o microcontrolador da ARM STM32F405.

Figura 13 – Vista superior do robô *uMaRT Lite Plus*



Fonte: adaptado de Silva (2015b)

As informações técnicas são as seguintes:

- a) dimensões (comprimento, largura, altura): 95x89x32 mm;
- b) peso (sem bateria): 71 g;
- c) bateria: Li-Po 300 mAh de duas células;
- d) microcontrolador: STM32F405RG6, com 192 kB de RAM e 168 MHz;
- e) tensão de entrada (bateria): 6 - 9 V;
- f) motores: micromotor Pololu com caixa de redução de metal 30:1 HP (1000 RPM);
- g) rodas: 32x7mm (plástico e borracha);
- h) encoders: magnético em quadratura (360 PPR);
- i) sensores de linha: QRE1113GR;

- j) sensores de parede: SRH4545 (LED)/TEFT4300 (fototransistor);
- k) giroscópio: LY3200ALH;
- l) ponte H dos motores: TB6612FNG.

Esta plataforma robótica é completa e de alto desempenho. Ela conta com dois micromotores com caixa de redução de metal, *encoders* independentes de 360 pulsos por revolução (PPR), oito sensores de linha, quatro sensores de parede, buzzer, medição da tensão da bateria, giroscópio, conector para depuração ou controle sem fio via módulo *bluetooth*, um botão e cinco LEDs para interação com o usuário. Capaz de atingir velocidades de até 2,0m/s, o uMaRT Lite Plus é um ótimo kit para o aprendizado e para utilizá-lo em competições de robótica.

Este robô contém dois módulos QEI (*Quadrature Encoder Interface*), incomum em microcontroladores baratos. Este módulo permite, quando ligado a um *encoder* mecânico ou óptico, dar informação acerca da direção da rotação e do número de pulsos realizados. Os *encoders* magnéticos emitem dois sinais: Fase A e Fase B. Se a fase A está em avanço em relação à fase B, o sentido de rotação é positivo e, em caso contrário, o sentido de rotação é considerado negativo.

2.2.2 Programação

Umas das maneiras de programar microcontroladores da plataforma ARM é por meio do software *ECLIPSE IDE*. No site Micromouse Brasil (em Arquivos) são disponibilizados tutoriais de configuração e criação de projeto. Também serão fornecidos alguns exemplos básicos de código em repositórios no GitHub: github.com/Micromousebrasil. Os códigos de programação são construídos em linguagem C.

No projeto teste fornecido pelo fabricante do robô, constam algumas funções básicas implementadas, prontas para uso, funções estas que retornam valores importantes para uso em algoritmos de controle como realimentadores do processo em malha fechada e para detecções de obstáculos durante uma corrida em labirinto. As seguintes funções foram utilizadas em conjunto com os algoritmos de controle e de resolução de labirinto.

- a) `getSensoresParede()`: função que retorna, em binário, se existem paredes à frente do robô. São três *bits* de informações, sendo que o *bit* mais significativo representa a existência da parede à esquerda, o segundo *bit*, parede à frente, e, por último, o *bit* menos significativo identifica a existência da parede à direita;

- b) `getSensoresLinha()`: utilizado para ler informações dos sensores leitores de linha. Função não utilizada e retirada do programa;
- c) `getGyro()`: utilizado para retornar o valor de grandeza proporcional ao valor de velocidade angular global do robô;
- d) `getTensao()`: função que retorna o valor de tensão da bateria. Utilizado para detectar o momento certo de carregar a bateria;
- e) `setMotores(int16_t pwm_esquerda, int16_t pwm_direita)`: função utilizada pelo controlador para atualizar o sinal de controle das rodas;
- f) `setLED(Led_TypeDef led, GPIO_PinState estado)`: utilizado para acender ou apagar algum LED do Micromouse. No robô existem 5 LEDs;
- g) `toggleLED(Led_TypeDef led)`: utilizado para alternar o estado do LED;
- h) `allLEDs(GPIO_PinState estado)`: utilizado para acender ou apagar todos os LEDs do Micromouse. Geralmente utilizado no final de uma corrida. É uma forma de alerta de êxito;
- i) `getEncoderEsquerda(void)`: utilizado para obter a quantidade de pulsos gerados pelo *encoder* conectado ao eixo do motor esquerdo. Útil para realimentar o controle de velocidade das rodas;
- j) `resetEncoderEsquerda(void)`: utilizado para zerar a quantidade de pulsos armazenados pelo contador do módulo QEI;
- k) `getEncoderDireita(void)`: utilizado para obter a quantidade de pulsos gerados pelo *encoder* conectado ao eixo do motor direito. Útil para realimentar o controle de velocidade das rodas;
- l) `resetEncoderDireita(void)`: utilizado para zerar a quantidade de pulsos armazenados pelo contador do módulo QEI.

2.2.3 Gravação

O ARM STM32F405 do robô uMaRT Lite Plus pode ser gravado por Bootloader em dois modos, USART 1 (conector J2) ou USB-DFU (conector J3). Para que o microcontrolador entre no modo de boot é necessário resetar o microcontrolador com o botão de BOOT pressionado, logo após este procedimento o botão de BOOT já pode ser liberado e o microcontrolador já está no modo de gravação via Bootloader.

- a) *software* de gravação via USART: STMFlashLoader;
- b) *software* de gravação via USB: DfuSe USB.

2.2.4 Comunicação

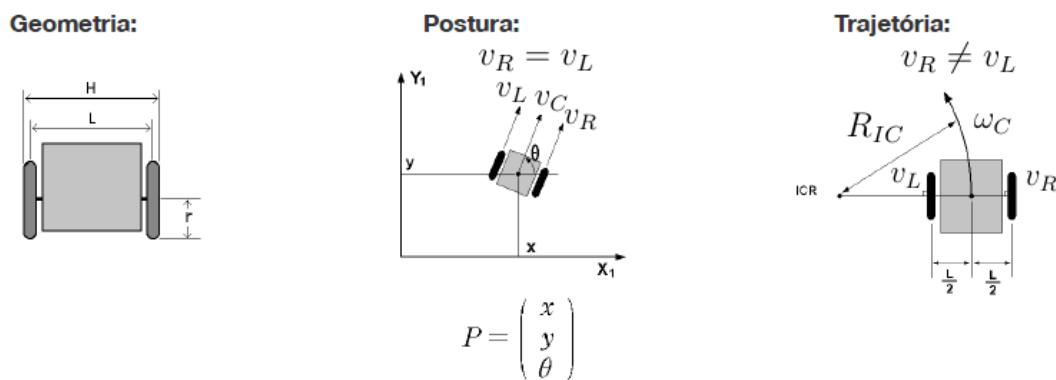
O STM32F405 possui conectividade USB 2.0 *full-speed* e *high-speed* que pode ser usada para comunicação com um computador. Outro tipo de comunicação com um computador ou com um smartphone é por meio de um módulo *bluetooth* HC-06 conectado ao conector J2.

2.2.5 Cinemática do robô móvel diferencial

O robô Micromouse, Figura 13, é um robô móvel com tração diferencial, isto é, as rodas são acionadas independentemente. A tração diferencial tem as seguintes características:

- a) rodas co-axiais;
- b) tracionadas independentemente;
- c) duas dimensões;
- d) restrito holonomicamente.

Figura 14 – Robô móvel diferencial - características



Fonte: autor (2017)

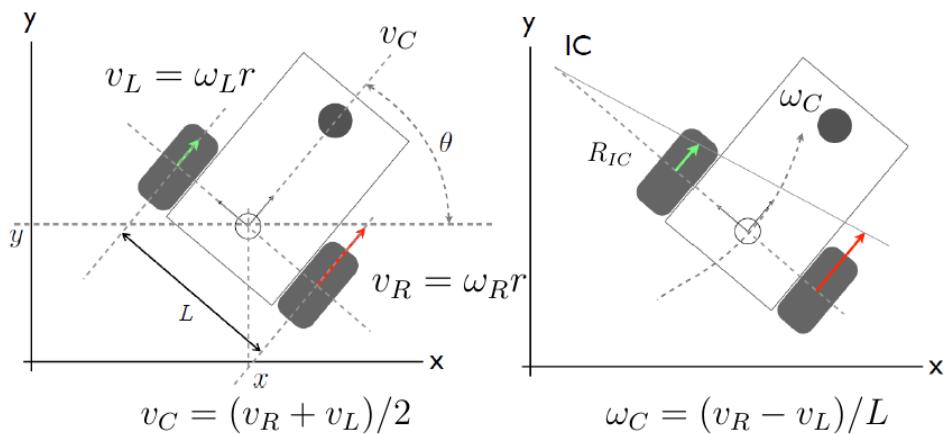
Como pode-se perceber na Figura 14, o robô diferencial depende de sua geometria (distância entre as rodas e o raio das mesmas) para o cálculo de postura (x , y e θ) e de previsão de trajetória.

Quando as duas rodas tem suas velocidades iguais, o robô realiza uma linha reta ($v_R = v_L$, onde v_R e v_L são, respectivamente, as velocidades da roda direita e da roda esquerda). Quando as velocidades são diferentes ($v_R \neq v_L$), a sua trajetória forma um arco. E para realizar um giro em torno do seu eixo, as velocidades são iguais, mas

com os sentidos trocados ($v_R = -v_L$). A restrição holonômica é quando sua posição é limitada a um subconjunto do espaço de configurações através de uma função igual a zero.

A cinemática direta de um robô móvel diferencial é caracterizada através das Equações (1.1) e (1.2). Dada a geometria do robô e a velocidade das rodas, determinam-se as velocidades linear e angular global, além do ângulo de direção num plano bidirecional, conforme é mostrado na Figura 15, onde r é o raio das rodas, L é a largura do eixo, v_R é a velocidade angular da roda direita e v_L é a velocidade angular da roda esquerda.

Figura 15 – Robô móvel diferencial - cinemática direta



Fonte: autor (2017)

Seguindo pelas equações cinemáticas, pode-se calcular a velocidade angular para velocidade linear e raio de curvatura fixas ($\omega_C = v_C/R_{IC}$), onde R_{IC} é o raio do centro instantâneo da trajetória circular. O controlador proposto ajusta as velocidades das rodas de tal maneira a fixar as velocidades calculadas e o robô seguirá a trajetória circular. As velocidades das rodas são calculadas através das Equações (2.1) e (2.2), onde ω_R , ω_L e r são, respectivamente, velocidade angular da roda direita, velocidade angular da roda esquerda e raio da roda.

$$v_R = \omega_R r \quad (2.1)$$

$$v_L = \omega_L r \quad (2.2)$$

2.3 Algoritmo

O objetivo do algoritmo de pesquisa é encontrar um caminho desde a origem até ao destino pretendido do labirinto, que no nosso caso é o centro do labirinto. A forma mais simples de resolver um labirinto é com o algoritmo aleatório, que anda sempre em frente no labirinto até encontrar uma parede. Embora existam muitos algoritmos diferentes os algoritmos mais utilizados para a resolução do labirinto são o Left Wall Follower, o Right Wall Follower, o Trémaux e o *Flood Fill* (SHARMA, 2009 apud BORGES, 2013, p. - 21).

A literatura no que diz respeito a algoritmos de resolução de labirintos possui um já consolidado algoritmo e amplamente utilizado. Ele é chamado de algoritmo de imundação, ou *Flood Fill*, que atende as melhores expectativas pelo fato de minimizar a área de exploração do labirinto. Ele foi largamente utilizado e recomendado em diversos trabalhos (Yadav, Verma e Mahanta (2012), Su, Huang e Lee (2013), Cai, Ye e Yang (2012), Cai et al. (2010), Mishra e Bande (2008), Sadik et al. (2010)). Os problemas encontrados nos algoritmos *Seguidor de parede à esquerda* e *Seguidor de parede à direita*, em não lograr êxito em labirintos complexos, são resolvidos com os algoritmos surgidos a partir da Teoria dos Grafos. Segundo Mishra e Bande (2008), Cai, Ye e Yang (2012), o algoritmo *Flood Fill* consegue resolver o labirinto em menor número de movimentos, voltas e tempo de corrida.

A ideia desse algoritmo é percorrer as células do maior para o menor valor (YADAV; VERMA; MAHANTA, 2012). O *Flood Fill* clássico, considerado como um dos mais eficientes para resolver labirintos 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).

2.3.1 Interpretação física e lógica do *Flood Fill*

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

- a) dist: esta variável, do tipo inteiro de 16 bits, contém o número de passos para o robô chegar ao alvo;
- b) wall[4]: 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:

`wall[0]` é 1 quando tem parede ao norte ou 0 quando não tem parede ao norte;

- c) `checked`: 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 e acelerar o robô.

Estas variáveis são acessadas através do identificador da estrutura, de nome `celula`, matriz de 16×16 . Cada par de índice i,j da matriz acessa a estrutura correspondente à célula de coordenada (i,j) do labirinto real. As informações de orientação do robô e suas coordenadas dentro do labirinto também estão armazenadas dentro de uma estrutura de dados, cujo nome é `robo`. As seguintes informações são armazenadas:

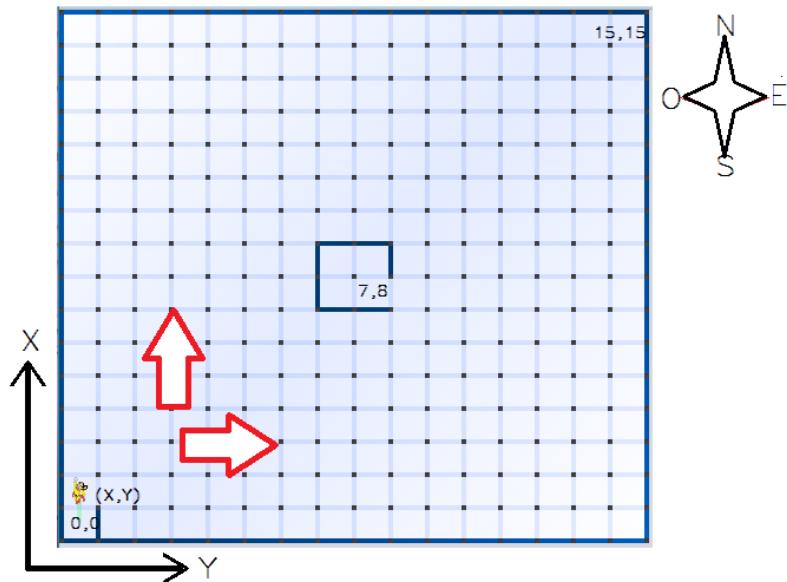
- a) `run`: esta variável terá informação do *status* da corrida. Exemplo: Fazendo uma corrida física ou preparando a volta/ida;
- b) `orientacao`: guarda a orientação do robô - NORTE, SUL, LESTE ou OESTE;
- c) `sentido_robo`: esta variável armazena o sentido de movimento no momento. Esta informação é dada pelo algoritmo *Flood Fill* sempre antes de adentrar na célula seguinte. Os sentidos são: FRENTE, ESQUERDA, DIREITA e VOLTA;
- d) `x`: esta variável armazena a coordenada X em que o robô se posiciona no labirinto;
- e) `y`: esta variável armazena a coordenada Y em que o robô se posiciona no labirinto;

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 16. 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.

Cada célula possui, além da distância, informações das paredes. Por exemplo, na coordenada $(15,15)$ (Figura 16), há parede ao norte e ao leste. Então, estas informações podem ser acessadas diretamente da estrutura de dados da célula, ou seja, durante a criação do algoritmo uma estrutura de dados com o nome `celula[16][16]` foi criado, e desta forma, ao acessar `celula[15][15].wall[NORTE]` e `celula[15][15].wall[LESTE]`, ambos retornarão verdadeiras, enquanto as variáveis

`celula[15][15].wall[OESTE]` e `celula[15][15].wall[SUL]` retornarão falsas. Portanto, à medida que o robô vá percorrendo o labirinto, ele vai atualizando as informações das paredes das células, além de atualizar também a variável `checked`, uma variável binária para informação de célula já visitada.

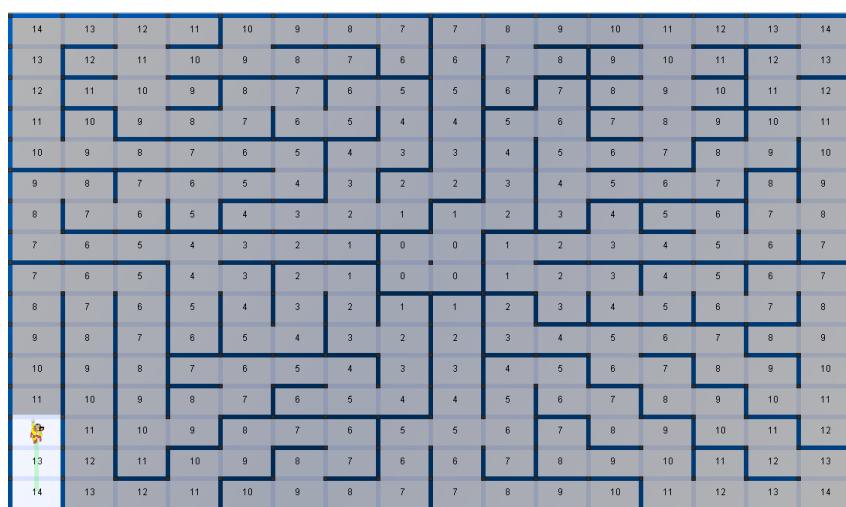
Figura 16 – Interpretação do labirinto para o Flood Fill



Fonte: autor (2017)

O algoritmo enfrenta o labirinto como se ele não tivesse paredes e enumera as células inicialmente conforme mostrado na Figura 17.

Figura 17 – Numeração das distâncias sobre o labirinto

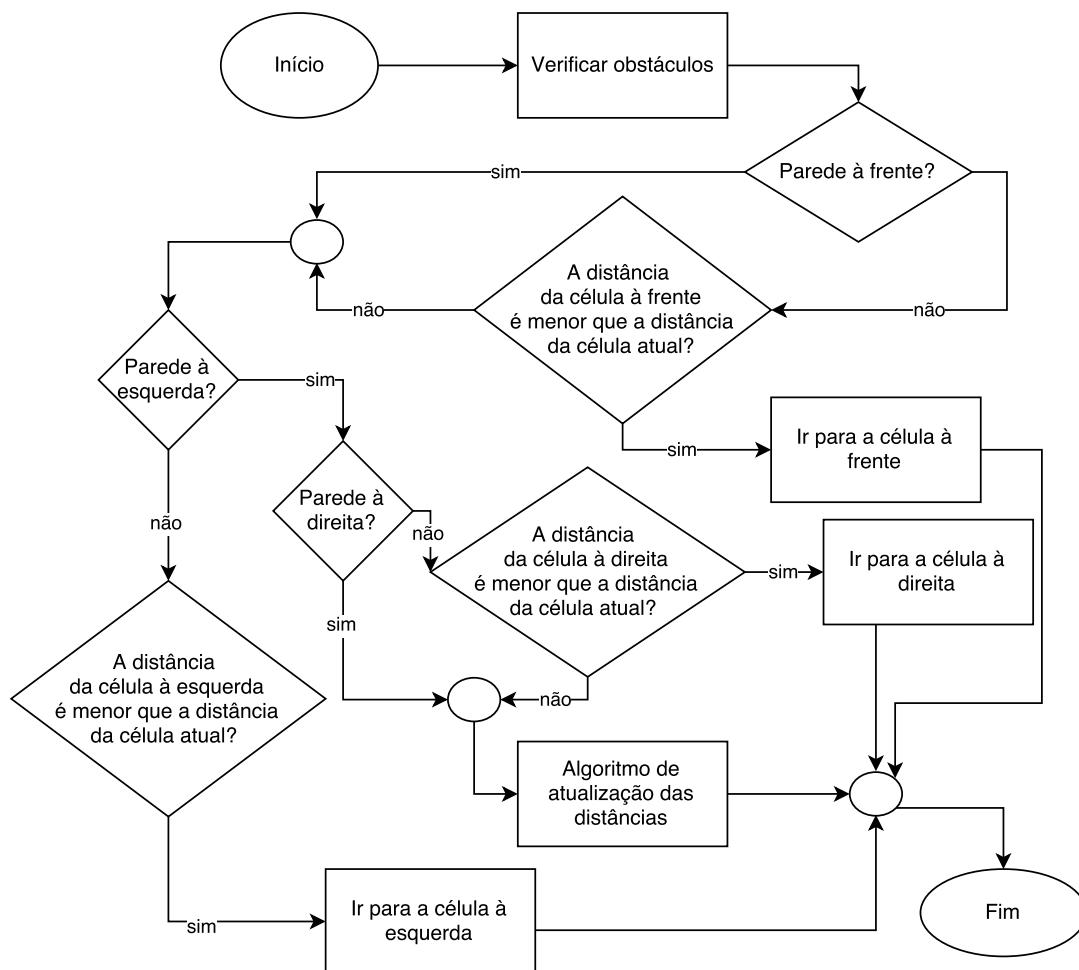


Fonte: autor (2017)

Estes valores são o número mínimo de passos até então para chegar ao alvo (valor zero). Na Figura, a parte desconhecida do labirinto pelo robô é mostrado em sombra. 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 18 mostra um algoritmo básico de tomada de decisão. Caso o robô tenha mais de uma opção de percurso, e se um destes percursos envolver a célula à frente e de distância menor que a distância da célula atual, o algoritmo sempre escolherá seguir em frente, visto que para realizar uma curva o robô perde em velocidade, ou seja, o algoritmo tenta ganhar tempo em retas com velocidades maiores.

Figura 18 – Fluxograma básico de tomada de decisão



Fonte: autor (2017)

Segundo as regras da competição, 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.

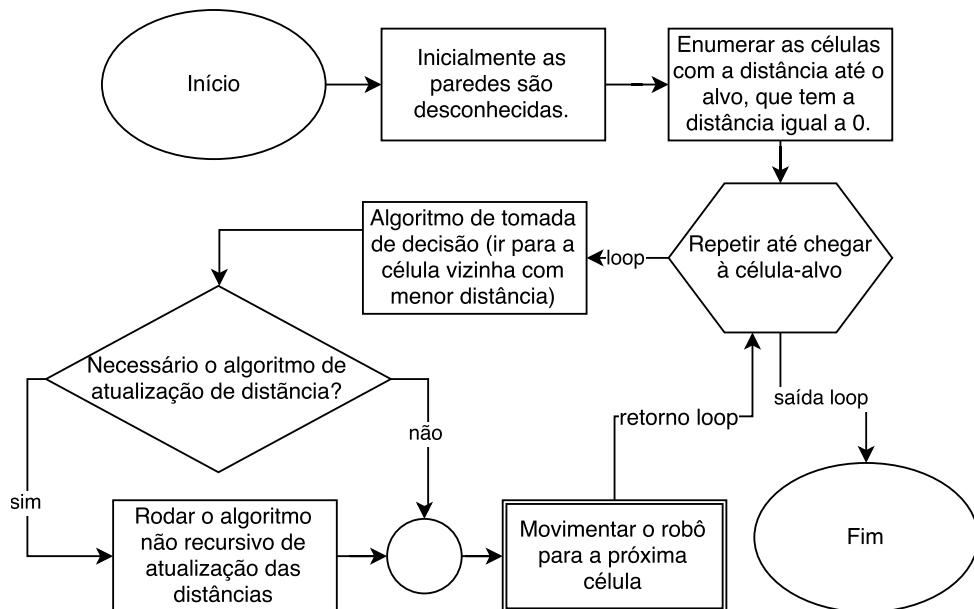
2.3.2 Algoritmo *Flood Fill* não recursivo com otimização de memória

Neste trabalho o algoritmo *Flood Fill* não recursivo 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. À frente mostram-se as discussões sobre a economia de RAM proposto. A seguir é listado o pseudocódigo do algoritmo:

- a) inicialmente o labirinto é enfrentado como se não existissem paredes;
- b) enumerar as células com a distância até o alvo, que tem a distância igual a 0;
- c) repetir até que se chegue à célula-alvo:
 - ir para a célula vizinha com menor distância;
 - rodar o algoritmo de atualização das distâncias.

O algoritmo *Flood Fill* básico não recursivo é resumido no fluxograma de Figura 19.

Figura 19 – Fluxograma básico do *Flood Fill*



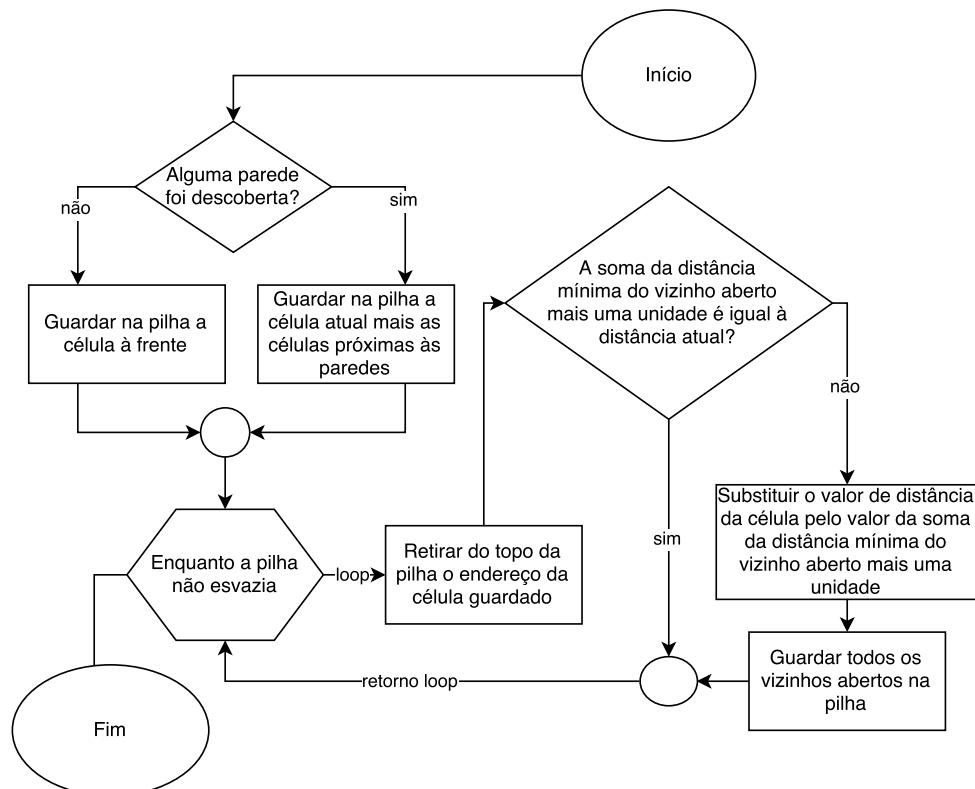
Fonte: autor (2017)

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, resumida na Figura 20, estão listados a seguir:

- a) verificar a pilha se ela está vazia;
- b) se alguma parede foi descoberta:
 - guardar na pilha a célula atual e as células próximas às paredes.
- c) enquanto a pilha não esvazia:
 - retirar do topo da pilha o endereço da célula guardado
 - verificar se a distância mínima do vizinho aberto àquela célula + 1 é a distância da célula atual
 - se isto não é verdade, deve-se modificar o valor da célula atual para o valor mínimo da distância das células vizinhas + 1 e guardar todos os vizinhos abertos para a pilha.

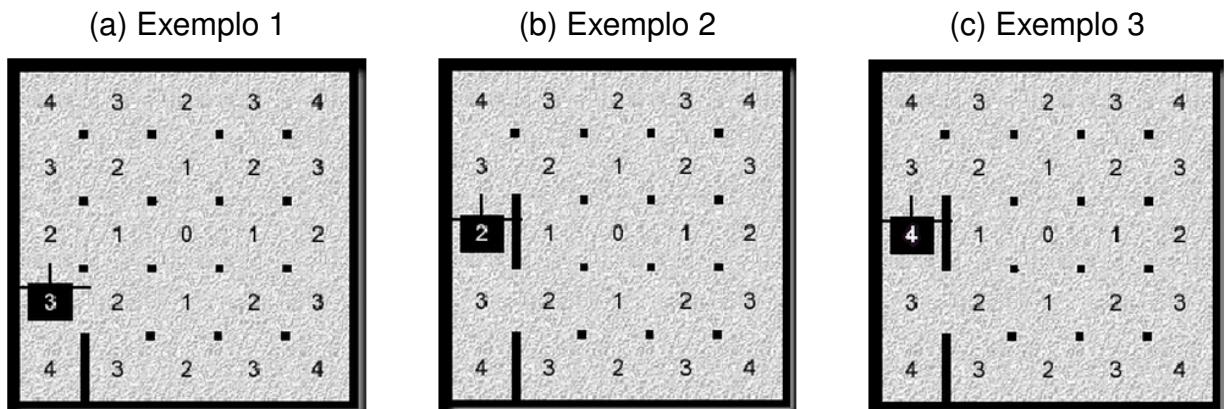
Figura 20 – Fluxograma básico do algoritmo de atualização das distâncias não recursivo



Fonte: autor (2017)

O funcionamento do algoritmo inteligente de resolução de labirinto e do algoritmo de atualização das distâncias é melhor entendido através dos exemplos (a), (b) e (c) de Figura 21.

Figura 21 – Exemplo de atualização da distância da célula



Fonte: autor (2017)

A Figura 21(a) mostra a posição inicial do robô, marcado em preto. O labirinto, inicialmente, é desconhecido. O robô tem dois caminhos, mas o robô segue em frente. Ao verificar os obstáculos à frente, na célula seguinte (Figura 21), o robô atualiza as paredes à leste e à oeste da célula à frente. O próximo passo do algoritmo *Flood Fill* é guardar na pilha as células vizinhas à parede descoberta, ou seja, as células de coordenadas (2,0) e (2,1). Então, até o momento, contém duas células na pilha. Logo em seguida, o algoritmo tira da pilha uma célula e realiza uma comparação da distância entre as células vizinhas à célula recém retirada do topo. Quando existe pelo menos um vizinho aberto à célula retirada do topo, com a distância menor que a distância da célula retirada da pilha, o algoritmo de atualização não realiza alteração da distância e retira uma outra célula da pilha, se houver. No caso da Figura 21(b), como não há vizinhos abertos com distância menor que 2, então há alteração da distância para o menor valor do vizinho aberto *mais 1*. O valor é alterado para $3 + 1 = 4$, como pode ser visto na Figura 21(c). Em outras palavras, o robô acaba de descobrir que o caminho não é o ótimo, e altera a distância, de forma a restringir a sua ida pela segunda vez.

Desta forma, o robô consegue chegar sempre à célula de destino, de distância zero. Então, como ele volta para a célula de partida e recomeçar uma corrida? A maneira tradicional é a criação de duas estruturas de dados. Uma armazena as distâncias para o robô andar da célula de partida até a célula-destino enquanto a outra estrutura armazena as distâncias calculadas pelo algoritmo para o robô voltar à célula de partida.

Por outro lado, a otimização de RAM é obtida utilizando uma única estrutura

de dados para armazenamento das informações das distâncias das células. Após o término da corrida, a célula de distância nula é realocada, ora para a célula de destino, ora para a célula de partida. Então, a partir da célula-alvo de distância nula, é aplicada a regra das distâncias das células vizinhas para determinar os valores de cada célula. Apesar do custo computacional maior em rodar o algoritmo em modo de *varredura* para atualizar as distâncias, nada atrapalha, uma vez que o robô estará parado na célula de valor nulo e o tempo de corrida já fora contabilizado.

2.3.3 Construindo o algoritmo em linguagem C

O algoritmo *Flood Fill* é um inteligente solucionador de labirinto. Seu pseudocódigo contém poucas linhas, porém, sua construção é complexa.

A linguagem escolhida para programar foi a linguagem C, pelo fato de o Micromouse aceitar algoritmos de tal linguagem. Alguns autores programam em outras linguagens e utilizam programas tradutores para linguagem C. Porém, isto não é aconselhável para sistemas embarcados com limitação de memória, uma vez que o programa final torna-se inchado, comprometendo o espaço de memória.

A fim de deixar o código enxuto, utilizou-se do ambiente de desenvolvimento *FALCON C++ IDE*, um programa comum para estudantes de engenharia para desenvolver programas em linguagem C. O programa utiliza como saída de vídeo a janela do *Prompt do Windows* para realizar as impressões através do comando `printf()`.

Frequentemente são utilizados para impressões os símbolos mostados na Figura 8.

2.3.3.1 Máquina de estados para o algoritmo

A programação torna-se fácil quando se implementa uma *máquina de estados* para códigos extensos. Para tanto, foi criada uma máquina de estados para o algoritmo, com a utilização da função `enum`, função pré-definida em linguagem C para realizar máquinas de estado.

Logo a seguir constam os estados criados com suas breves funções:

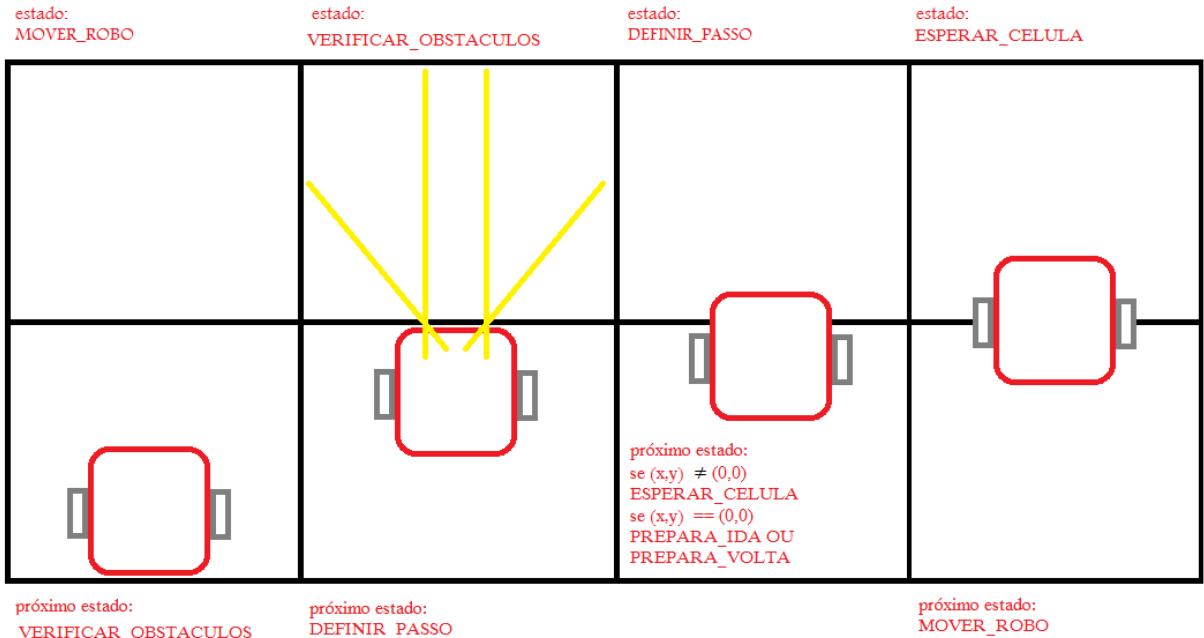
- a) **INICIO**: estado responsável para inicializar todas as numerações das células, retirar todas as informações das paredes, dar uma posição, orientação, número da corrida e sentido iniciais para o robô, zerar as variáveis para uso em controle. Após todas as configurações feitas, o algoritmo passa para o próximo estado: **SINAL_PARTIDA**;
- b) **SINAL_PARTIDA**: aguarda o sinal do usuário para sair para a célula-alvo. Lê as informações dos sensores de distância para configurar o próximo estado:

MOVER_ROBO;

- c) MOVER_ROBO: este estado aguarda o robô chegar à entrada da próxima célula. A configuração do próximo estado VERIFICAR_OBSTACULOS é feita após o robô andar uma distância mínima necessária para disparar a leitura das paredes à frente;
- d) VERIFICAR_OBSTACULOS: neste estado o robô dispara a leitura dos obstáculos à frente com os sensores IR utilizando a função `getSensoresParede()` do robô. Este estado é importante e os sensores devem estar calibrados. Uma identificação incorreta das paredes causa falha para definir o próximo sentido e orientação do robô. O algoritmo inteligente de resolução de labirinto *Flood Fill*, representado pelo estado seguinte DEFINIR_PASSO, só é executado após o término da leitura das paredes da próxima célula;
- e) DEFINIR_PASSO: este é o estado que dispara o algoritmo *Flood Fill*, que indicará o próximo sentido e orientação momentos anteriores à célula frontal. Uma vez definidos, o algoritmo configura o próximo estado: IMPRIMIR_MAZE;
- f) IMPRIMIR_MAZE: estado responsável por imprimir o labirinto, utilizando os símbolos, e também as coordenadas, estado atual, orientação, sentido, entre outros. O próximo estado é ESPERAR_CELULA;
- g) PREPARA_VOLTA: este é o estado o qual redefine todas as distâncias das células de tal forma que o robô possa aplicar a regra do *Flood Fill* e retornar à célula de partida. A economia de RAM proposto é feito realizando uma *varredura virtual*, isto é, enquanto o robô aguarda parado o caminho de volta, o algoritmo realiza o percurso virtual de volta. As paredes já conhecidas são utilizadas para atualizar as distâncias das células. Este estado é ativado quando a próxima célula é a *célula de destino*;
- h) PREPARA_IDA: este é o estado o qual redefine todas as distâncias das células de tal forma que o robô possa aplicar a regra do *Flood Fill* e retornar à célula-alvo. A metodologia para recriar o caminho é a mesma do estado PREPARA_VOLTA. Este estado é ativado quando a próxima célula é a *célula de partida*;
- i) ESPERAR_CELULA: este estado aguarda o robô chegar à próxima célula. O ciclo recomeça no estado MOVER_ROBO.

A Figura 22 mostra os momentos em que há transição de estados durante o percurso do Micromouse no labirinto.

Figura 22 – Posição do robô na célula e seus estados



Fonte: autor (2017)

2.3.3.2 Funções auxiliares

O robô em uma célula (x,y) pode tomar quatro direções, a depender dos obstáculos e da tomada de direção do algoritmo Flood Fill. No momento da atualização das paredes em uma célula (x,y) , a posição é importante. Algumas funções auxiliares foram criadas para automatizar o processo de atualização e tornar independente da orientação.

- meia_volta(int i): função recebe a orientação atual do robô e retorna a orientação inversa (Ex: Norte - Sul, Leste - Oeste);
- add_x(int orientacao,int x): função que ou adiciona uma unidade ao valor x, caso a orientação seja Norte, ou retira uma unidade à coordenada x, caso a orientação seja Sul, ou retorna o mesmo valor x, caso a orientação seja Leste ou Oeste;
- add_y(int orientacao,int y): função que adiciona ou retira uma unidade à coordenada y, caso a orientação seja Leste ou Oeste, e retorna o mesmo valor y, caso a orientação seja Norte ou Sul;
- letra_a_esquerda(int i): função que recebe a orientação atual do robô e retorna o sentido à esquerda. (Ex: O sentido à esquerda da orientação Leste é Norte; Norte - Oeste; Sul - Leste; Oeste - Sul);

- e) letra_a_direita(int i): função que recebe a orientação atual do robô e retorna o sentido à direita. (Ex: Norte - Leste; Leste - Sul; entre outros);
- f) definir_sentido(int orient_ant , int orient_atual): esta função é executada após o algoritmo *Flood Fill* e recebe tanto a orientação anterior como a nova orientação. A função compara as duas orientações e retorna um dos quatro estados do movimento do robô: FRENTE, ESQUERDA, DIREITA ou VOLTA. Caso os dois sentidos fossem iguais, o robô continuaria a seguir em frente; Caso uma orientação tenha o sentido contrário da outra, o robô deverá retornar em 180 graus. Se o sentido atual está à esquerda ou à direita em 90 graus de diferença, o robô deverá realizar uma curva de 90 graus. Um novo sentido de movimento é feito próximo à entrada da próxima célula, após a verificação dos obstáculos e execução do algoritmo de resolução *Flood Fill*;
- g) prox_celula_existe(int orientacao, int x, int y): esta função recebe como argumentos a orientação atual do robô, e as coordenadas (x,y) e a mesma retorna se a célula existe. Forma simples de verificação quando o robô se encontra nos limites do labirinto, onde não existe na estrutura de dados as células vizinhas.

2.3.3.3 Função para atualizar as paredes na estrutura de dados

A função responsável por ler as informações dos sensores infravermelhos e salvar na estrutura da célula, independente da orientação, foi criada. A informação da leitura das paredes é redundante. Ou seja, uma detecção de parede ao norte da célula (1,1), por exemplo, esta informação também deverá ser armazenada na parede Sul da célula (2,1), isto porque a parede divide as duas células. A Figura 22 mostra o momento do disparo da leitura dos obstáculos, numa posição um pouco à frente da metade da célula. O robô na orientação \circ e posicionado na coordenada (x,y), recebe e armazena na struct celula corretamente os três bits da função getSensoresParede() conforme os passos a seguir:

- a) o bit mais significativo, representando a leitura do sensor à esquerda do robô, é armazenado na variável wall[i] da célula à frente, onde i é a orientação à oeste da orientação \circ do robô. Também, na célula oposta à parede recém descoberta, é armazenado o bit, se a célula oposta existir;
- b) o segundo bit, representando a leitura dos dois sensores frontais do robô, é armazenado na variável wall[o] da célula à frente, onde \circ é a própria orientação do robô. Também na célula oposta à parede recém descoberta, é armazenado o bit, se a célula oposta existir;

- c) o bit menos significativo, representando a leitura do sensor à direita do robô, é armazenado na variável `wall[i]` da célula à frente, onde `i` é a orientação à leste da orientação `o` do robô. Também, na célula oposta à parede recém descoberta, é armazenado o bit, se a célula oposta existir;
- d) as *funções auxiliares* ajudam a automatizar todo o processo anteriormente, para qualquer que seja a orientação do robô e posição do mesmo no labirinto;
- e) se a célula já foi visitada, ou seja, se `celula[add_x(o, x)][add_y(o, y)].checked` é **verdadeiro**, os sensores não são utilizados.

2.3.3.4 Inicializando as distâncias das células para o alvo no centro do labirinto

A enumeração inicial das distâncias, conforme é mostrada na Figura 17, é feita pelo trecho do código da função `inicializa_lab()`. Nesta mesma função, o *check* de cada célula é zerada. Este trecho também faz parte da função `a_todo_vapor()`, que só é executado no estado `PREPARA_IDA`, que por sua vez, é responsável por reiniciar as distâncias e executar o *Flood Fill* no *modo de varredura* utilizando somente uma estrutura de dados para armazenamento das distâncias das células.

2.3.3.5 Reinicializando as distâncias das células quando o alvo é realocado para a célula de partida

A automação da enumeração das distâncias, para permitir ao robô voltar à célula de partida, é feita pelo trecho do código da função `a_volta_do_robo()`. Esta função é executada somente no estado `PREPARA_VOLTA`, o qual realiza a preparação da volta após êxito na corrida de ida. No mesmo código, as paredes-vértices das quatro células nulas também são consideradas, atualizando os bits das paredes correspondentes nas estruturas das células, exceto para a célula de chegada (x,y). Segundo as regras para competição, há somente uma entrada para as células centrais. Da mesma forma para o estado `PREPARA_IDA`, o procedimento é feito para o estado `PREPARA_VOLTA`, que por sua vez, é responsável por reiniciar as distâncias e executar o *Flood Fill* no *modo de varredura* utilizando somente uma estrutura de dados para armazenamento das distâncias das células.

2.3.3.6 Algoritmo Flood Fill em C

Esta é a parte mais importante de todas. O algoritmo *Flood Fill* é a parte inteligente, porque quando ele é executado (no final de cada célula), o mesmo retorna uma nova orientação e sentido para o robô, sendo o guia virtual do robô. Esta parte do programa integra o algoritmo de tomada de decisão (fluxograma na Figura 18), e

o algoritmo de atualização das distâncias (fluxograma na Figura 19). Uma função de nome `floodfill()` foi criada para realizar todo o processo necessário.

A estrutura de dados bastante utilizada nesta função é a Pilha, que na literatura é conhecido como LIFO (último a entrar e primeiro a sair). Ela tem como função ser memória temporária para dados, registradores ou tarefas, onde os itens são incluídos e retirados da mesma extremidade da lista. O algoritmo *Flood Fill* armazena em estrutura de dados do tipo PILHA os endereços das células. As seguintes funções para acesso à Pilha foram criadas:

- a) `Pilha_Construtor()`: responsável por zerar a contagem de empilhamentos e configurar o topo como NULO. Esta função é executada no estado INICIO;
- b) `Pilha_Tamanho()`: responsável por retornar o número de empilhamentos;
- c) `Pilha_Vazia()`: retorna *verdadeiro*, caso não tenha nenhum endereço na pilha;
- d) `Pilha_Push(CELULA *point)`: recebe o endereço da célula via argumento, aloca um espaço na RAM dinamicamente e guarda o valor do endereço no topo da pilha;
- e) `*Pilha_Pop()`: retira do topo da pilha o endereço da célula empilhada e retorna o endereço para um ponteiro do tipo struct.

O algoritmo é executado após a verificação dos obstáculos à frente. De acordo com o fluxograma do algoritmo, Figura 19, são empilhadas somente as células próximas às paredes recém descobertas.

Após isto, o topo da pilha é retirado e representa a célula a entrar. O acesso às variáveis da estrutura da célula é feito por meio do *ponteiro*. Um ponteiro C, do mesmo tipo da estrutura de dados struct, é criado para receber o endereço. São feitas as análises de vizinhança para tomada de decisão a partir da célula de endereço armazenado no ponteiro.

Se há um vizinho aberto, uma variável auxiliar k não será contabilizada. Sendo nula, o algoritmo de atualização das distâncias é ignorado e o programa retira um novo endereço da pilha, caso a função `Pilha_vazia()` retorne falso. Caso contrário, o algoritmo *Flood Fill* é encerrado.

Após todo o processo *Flood Fill*, são definidos uma nova orientação e um novo sentido, isto para as células de distância não nula. O trecho do código, que realiza a escolha da orientação e sentido de forma automática, está logo a seguir:

- a) se a próxima célula não é a célula-destino, e se não houver parede e sua distância é menor que a da célula atual, o robô continuará com o mesmo sentido;
- b) caso contrário, procurar vizinhanças abertas com distância menor que a distância da célula atual, sendo que a nova orientação será o mesmo sentido do vizinho aberto;
- c) por último, o algoritmo configura as coordenadas do robô para as da próxima célula.

Portanto, o algoritmo proposto tomou forma e seus resultados são comentados no próximo capítulo.

2.4 Controle

Para o correto entendimento da implementação do controle digital no Micromouse, um pequeno histórico dos controladores de processos é mostrado à frente.

2.4.1 Histórico

Com o rápido desenvolvimento dos computadores digitais e sua larga aplicação na engenharia, a identificação de modelos discretos foi amplamente desenvolvida e empregada em diversas áreas do conhecimento (ARAUJO; JUNIOR, 2012). O Método dos Mínimos Quadrados estima os parâmetros de um modelo de equação discreta para qualquer processo, e as equações podem ser usadas para simulações de controladores diversos.

Karl Friedrich Gauss formulou o *Princípio dos Mínimos Quadrados* ao final do século 18 para prever as trajetórias de planetas e cometas a partir das observações realizadas. K. F. Gauss estabeleceu que os parâmetros desconhecidos de um modelo matemático deveriam ser selecionados de modo que:

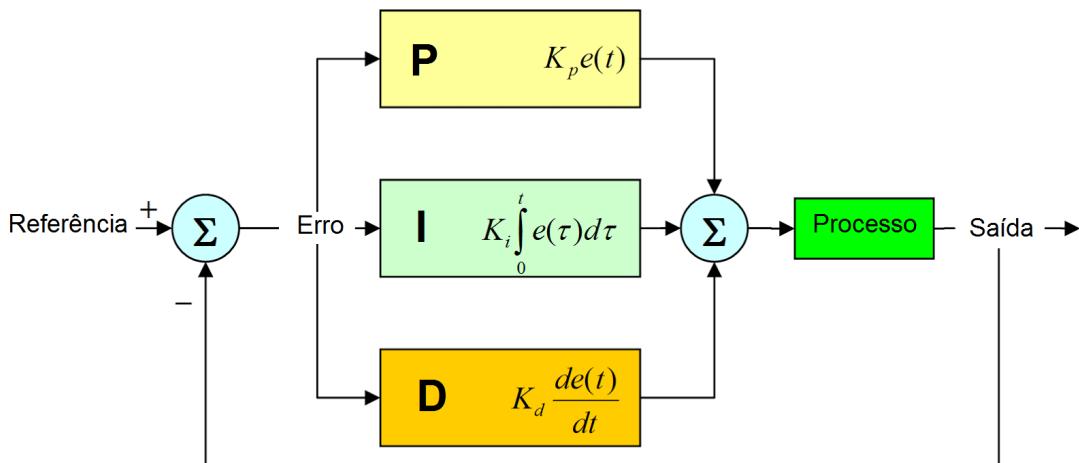
o valor mais provável das grandezas desconhecidas é a quantidade que minimiza a soma dos quadrados da diferença entre os valores atualmente observados e os valores calculados multiplicados por números que medem o grau de precisão, onde quanto mais precisa a medida, maior sua ponderação. (LJUNG; SÖDERSTRÖM, 1983 apud COELHO; COELHO, 2015, p.-123).

Em 1920, Foxboro introduziu um controlador baseado em ar comprimido com ação integral. Em um dado momento as empresas de sensores introduziram instrumentos e controladores que poderiam implementar controladores PID. Embora Maxwell

e Routh tenham desenvolvido uma base matemática para assegurar a estabilidade de um sistema realimentado, o projeto de controladores era baseado na experiência do projetista e em tentativa e erro. Dispositivo PID é composto por: um termo Proporcional para fechar a malha de realimentação, um termo Integral para garantir erro nulo à referência constante e às entradas de perturbação, e um termo Derivativo para melhorar a estabilidade e a boa resposta dinâmica. Como parte da evolução do projeto do controlador PID, um passo importante para sintonizar controladores deste tipo foi dado em 1942, quando Ziegler e Nichols, que trabalhavam para Taylor Instruments, publicaram o seu método de sintonia com base em dados experimentais (FRANKLIN; POWELL; EMANI-NAEINI, 2013).

Partindo do controle proporcional realimentado, os primeiros engenheiros descobriram a ação de controle integral como forma de eliminar o erro em regime permanente. Entretanto, encontravam, em muitos casos, uma resposta dinâmica pobre; assim, um termo de "anticipação" baseado na derivada foi adicionado. (FRANKLIN; POWELL; EMANI-NAEINI, 2013, p. -160)

Figura 23 – Diagrama de blocos do controle PID em malha fechada



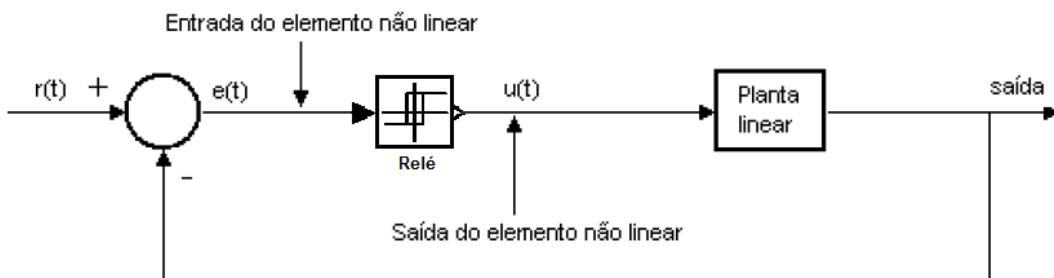
Fonte: autor (2017)

O resultado destes três controladores tem nome: controlador de três termos, ou PID, e tem a função de transferência (2.3), sendo k_P o termo proporcional, k_I o termo integral e k_D o termo derivativo. A Figura 23 mostra o diagrama de blocos do sistema em malha fechada.

$$G_c(s) = k_p + k_I/s + (k_D)s \quad (2.3)$$

Os experimentos com relé na malha de realimentação (Figura 24), com o propósito de identificação de processos, tornaram-se populares a partir do trabalho de Åström e Hägglund (1984). Este método foi utilizado para determinar o ganho crítico e a frequência crítica e, por consequência automatizar o método de oscilação de projeto do controlador PID proposto por Ziegler e Nichols (1942). A abordagem baseia-se na modelagem da não linearidade através de sua função descriptiva e na interpretação em termos do diagrama de Nyquist para a obtenção de informação em frequência do processo. A identificação do processo é feita a partir da estimativa em frequência da função de transferência do processo num experimento de malha fechada (COELHO; COELHO, 2015).

Figura 24 – Relé na malha realimentada



Fonte: adaptado de Coelho (2004)

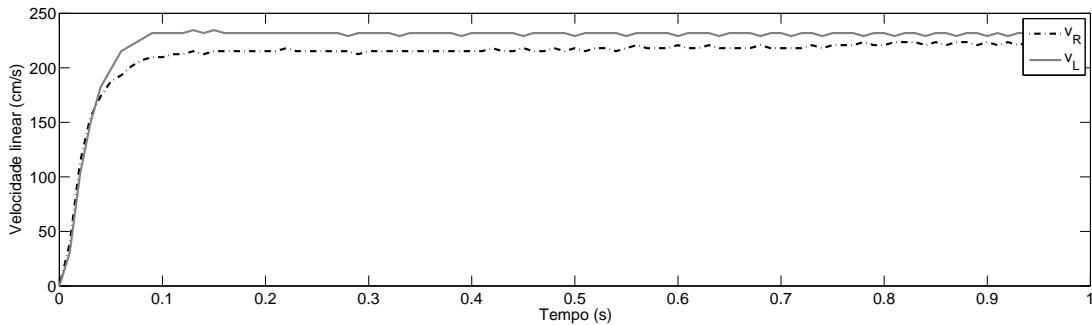
2.4.2 Determinando o tempo de amostragem do processo

Em qualquer processo é importante o conhecimento dos limites da planta em *malha aberta* ou em *malha fechada* para determinar o tempo de amostragem do sistema. A técnica para a escolha do tempo de amostragem para sistemas de controle digitais, segundo Landau e Zito (2006), é o cálculo a partir da largura de banda do sistema em malha fechada ou estimá-la a partir da resposta ao degrau. A regra usada para a escolha do tempo de amostragem é mostrado em (2.4), onde f_s é a frequência de amostragem e f_B^{CL} é a largura de banda do sistema de malha fechada. A regra da Equação (2.4) é usada igualmente em malha aberta, realizando a estimativa da largura de banda do processo e substituindo-a por f_B^{CL} . Desta forma, em malha aberta, o tempo de amostragem pode ser calculada de 1/5 a 1/10 do tempo de assentamento da resposta ao degrau.

$$f_s = (6 \text{ a } 25)f_B^{CL} \quad (2.4)$$

Como a entrada do sistema *encoder-rodas* é um sinal PWM com *duty* entre 0 e 100 %, para este teste, programou-se um *duty cycle* de 100% para um tempo de amostragem inicial de 1 ms e a saída do processo estabilizou em $T_{assent} = 100\text{ ms}$, como pode ser observado na Figura 25.

Figura 25 – Sistema Motor-encoder em malha aberta



Fonte: autor (2017)

Landau e Zito (2006) disponibilizou alguns períodos de amostragem de diferentes processos para controle digital. A Tabela 1 mostra para cada tipo de planta faixas de tempos de amostragem ideais. Como o processo do Micromouse é composto basicamente por motores DC, a escolha de 10ms para o tempo de amostragem torna-se adequada. Portanto, o tempo de amostragem adotado para o sistema de controle digital do Micromouse é de $T_s = 10\text{ms}$, considerando $T_{assent}/10$.

Tabela 1 – Escolha do tempo de amostragem para sistemas de controle

Tipo de variável (ou planta)	Tempo de Amostragem (s)
Vazão	1 - 3
Nível	5 - 10
Pressão	1 - 5
Temperatura	10 - 180
Destilação	10 - 180
Motores DC	0,001 - 0,05
Catalytic reactors	10 - 45

Fonte: adaptado de Landau e Zito (2006, p. 31–32)

2.4.3 Sintonia via Método do Relé

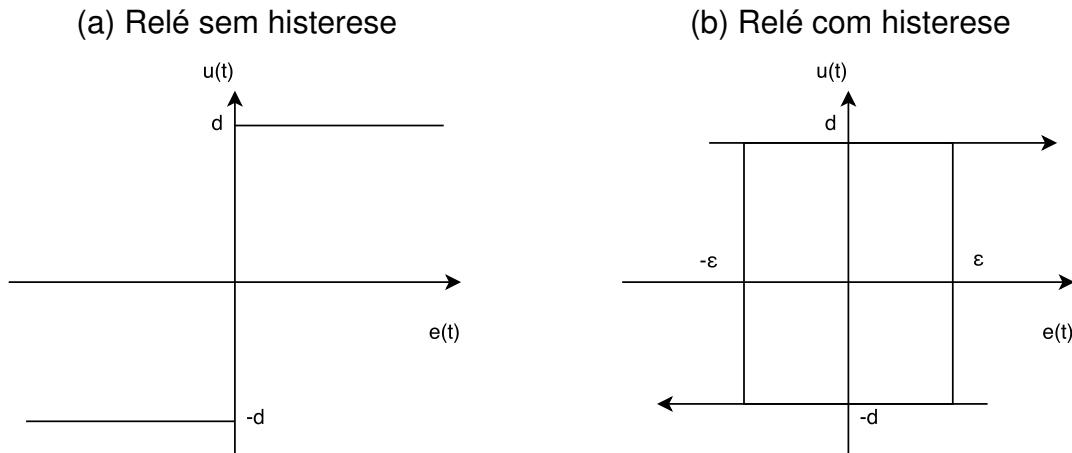
O controle PID geralmente é conhecido como um controle clássico para processos SISO (única entrada e única saída). Portanto, existem muitas pesquisas sobre métodos de ajuste do controle PID para o sistema SISO. O controlador adaptativo com autossintonia, do tipo *auto-tunning*, insere um relé no contexto da estimativa de

modelos matemáticos de ordem reduzida a partir da resposta em frequência. Segundo Coelho e Coelho (2015), na atualidade, é inevitável uma breve descrição da metodologia do relé no contexto da estimativa de modelos matemáticos de ordem reduzida a partir da resposta em frequência. O método do relé na malha de realimentação, como já foi dito anteriormente, tem o propósito de descobrir o ganho crítico e a frequência crítica.

Controladores PID (proporcional, integral e derivativo) são os controladores lineares mais utilizados na prática. O ajuste de seus parâmetros é um dos problemas de controle mais considerados. Para o caso de plantas de modelo desconhecido, Åström apresentou um método de auto-ajuste de parâmetros de PID utilizando um elemento não linear. Tendo dado ênfase ao uso do relé ideal, Åström sugeriu como extensão à possibilidade do uso do relé com histerese com o objetivo de melhorar a relação sinal-ruído, embora não tenha apresentado um algoritmo de auto-ajuste correspondente.

Åström propôs um sistema de sintonia automática de controle PID. Este método utiliza o método da linearização harmônica com base no conceito da função descriptiva para estimar parâmetros do sistema.

Figura 26 – Relé - elemento não linear



Fonte: autor (2017)

O relé sem histerese, Figura 26(a), pode ser modelado no domínio do tempo por:

Se $e(t) > 0$ então $u(t) = +d$;

Se $e(t) < 0$ então $u(t) = -d$;

Já o relé com histerese de Figura 26(b) pode ser modelado no domínio do tempo

por simples regras linguísticas descrevendo o comportamento da histerese conforme mostra a seguir, onde ε é igual a eps .

Se $|e(t)| > \text{eps}$ e $e(t) > 0$ então $u(t) = +d$;
 Se $|e(t)| > \text{eps}$ e $e(t) < 0$ então $u(t) = -d$;
 Se $|e(t)| < \text{eps}$ e $u(t-1) = +d$ então $u(t) = +d$;
 Se $|e(t)| < \text{eps}$ e $u(t-1) = -d$ então $u(t) = -d$;

Considerando o relé sem histerese e com histerese, conforme mostra a Figura 26, têm-se as equações (2.5) relatando as funções descritivas.

$$\begin{aligned} -1/N(a) &= -\pi a / 4d && (\text{sem histerese}) \\ -1/N(a) &= -\pi \sqrt{a^2 - \varepsilon^2} / 4d - j\pi\varepsilon / 4d && (\text{com histerese}) \end{aligned} \quad (2.5)$$

A partir da modelagem do relé por função descritiva e da operação do sistema sob o controle do relé, pode-se determinar a função de transferência do processo por (2.6), onde a é a amplitude de oscilação do sinal na saída do processo e ω é a frequência de oscilação medida.

$$\begin{aligned} G_p(j\omega) &= -1/N(a) \\ G_p(j\omega) &= -\pi a / 4d && (\text{sem histerese}) \\ G_p(j\omega) &= -\pi \sqrt{a^2 - \varepsilon^2} / 4d - j\pi\varepsilon / 4d && (\text{com histerese}) \end{aligned} \quad (2.6)$$

Pela Equação (2.6), pode-se estimar a função de transferência do processo na frequência de cruzamento utilizando-se o relé sem histerese. A equação (2.6) permite também estimar a função de transferência do processo em diferentes frequências utilizando-se um relé com histerese e ajustando-se diferentes valores para o parâmetro ε .

As regras de sintonia propostas por Åström consistem no seguinte:

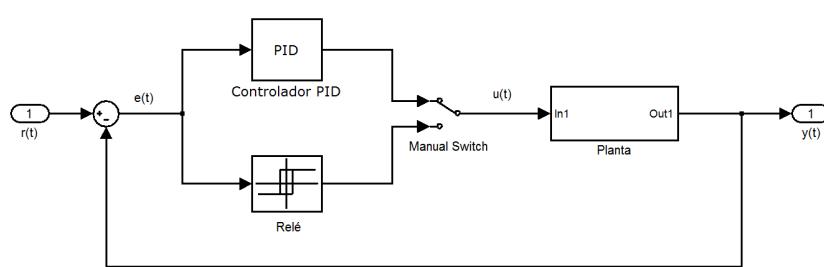
- sob o controle a relé, espera-se o sistema realimentado entrar em oscilação;
- utilizando a expressão da função descritiva do relé, estima-se o ganho crítico e o período da oscilação que o sistema teria sob controle proporcional. Åström considerou, que nestas condições, o ganho $N(A)$ de Equação (2.5) do relé representa um valor bem aproximado do ganho crítico K_{cr} . Por este motivo K_{cr} é feito igual a $N(A)$;
- encontrada a sintonia através da Tabela 2, o sistema passa a controle PID, conforme mostra a Figura 27.

Tabela 2 – Regras de ajustes estabelecidos por Ziegler-Nichols

Tipo de Controlador	K_p	T_i	T_d
P	$0,5K_{cr}$	Infinito	Zero
PI	$0,45K_{cr}$	$0,83T_{cr}$	Zero
PID	$0,6K_{cr}$	$0,5T_{cr}$	$0,125T_{cr}$

Fonte: Coelho e Coelho (2015)

A técnica de realimentação do relé é utilizada para substituir o procedimento manual de tentativa e erro na sintonia de controladores PID. A Figura 27 ilustra o princípio de implementação da técnica adaptativa *auto-tunning*. Quando a malha de controle está em oscilação com o relé, os parâmetros PID são calculados e armazenados automaticamente e o sistema troca o bloco do relé para o do PID.

Figura 27 – Diagrama de blocos do sistema para autossintonia dos parâmetros do Controlador PID - SISO

Fonte: adaptado de Barçante (2011)

2.4.3.1 Método do Relé para sintonia do sistema SISO

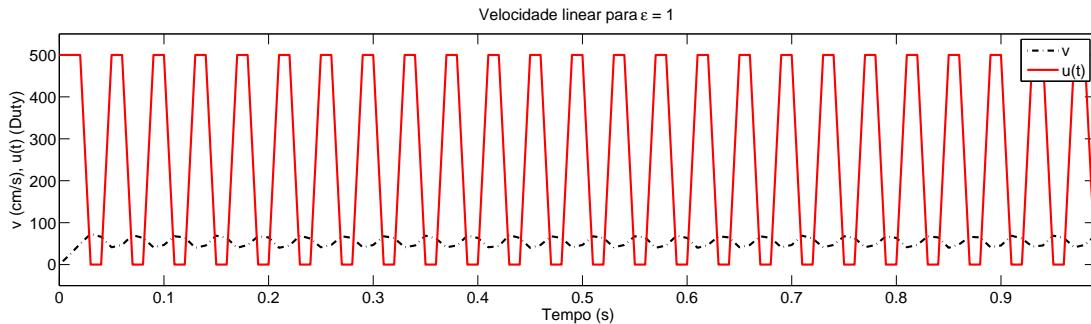
Para exemplificar o método do relé, para o sistema motor-*encoder* (SISO), foi determinado o ponto de cruzamento de ganho ou ponto crítico utilizando um relé (com e sem histerese) para a malha das rodas do Micromouse, e também traçado o diagrama de Nyquist para a malha de velocidade do robô para pelo menos 3 pontos distintos.

O procedimento do método do relé para análise do processo SISO motor-*encoder* foi feito a fim de verificar a curva de *Nyquist*. No robô foi implementado um programa com o relé no domínio do tempo. A Figura 28 mostra o gráfico de entrada do processo e o gráfico em resposta ao relé para ε de 1.

Através da amplitude e período do sinal de saída v , para seus respectivos ε , foram encontrados os valores críticos, tanto de ganho como de período. Os valores de ganho crítico e período crítico para cada ponto do relé estão expostos na Tabela 3

e calculados conforme (2.5), onde a margem de ganho K_u é o inverso do módulo de $G(j\omega)$.

Figura 28 – Relé com histerese para $\varepsilon = 1$



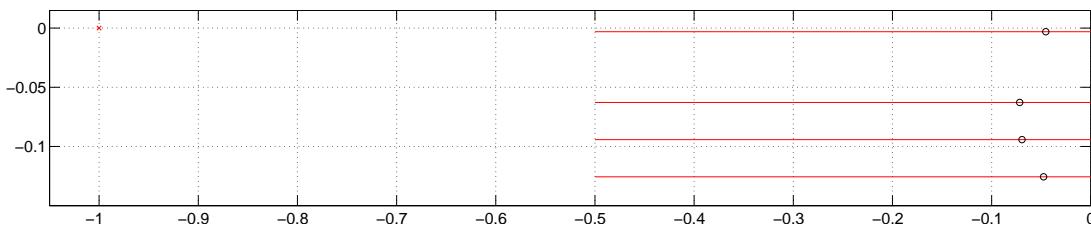
Fonte: autor (2017)

Tabela 3 – Margem de ganho K_u e frequência crítica para relé com diversos ε . O valor de d do relé é 250.

ε	a	$T_u(s)$	K_u	$\omega_u(\text{rad/s})$	$K_p K_u$
1	14.4893	0.04	22.0212	157.0796	4.8316
20	30.3585	0.07	13.9369	89.7598	3.2758
30	37.2581	0.09	14.4067	69.8132	3.1411
40	42.7779	0.11	20.9912	57.1199	4.5825

Fonte: autor (2017)

Figura 29 – Curva de Nyquist para o sistema SISO do Micromouse



Fonte: autor (2017)

Pode-se observar que o processo é muito estável porque sua curva de Nyquist passa longe do ponto de instabilidade, como mostra a Figura 29. Sua margem de ganho K_u é de 22. As marcações em círculo são os pontos real e imaginário da Equação (2.6). As retas paralelas em vermelho representam a histerese para ε de 1, 20, 30 e 40, respectivamente. Quanto maior a histerese, a reta paralela mais se desloca para baixo.

Por fim, calculou-se os parâmetros PID k_P , k_I e k_D , da Equação (2.3), utilizando a Tabela 2 para os valores obtidos para relé com histerese de $\varepsilon = 1$ da Tabela 3, que resultou na Tabela 4.

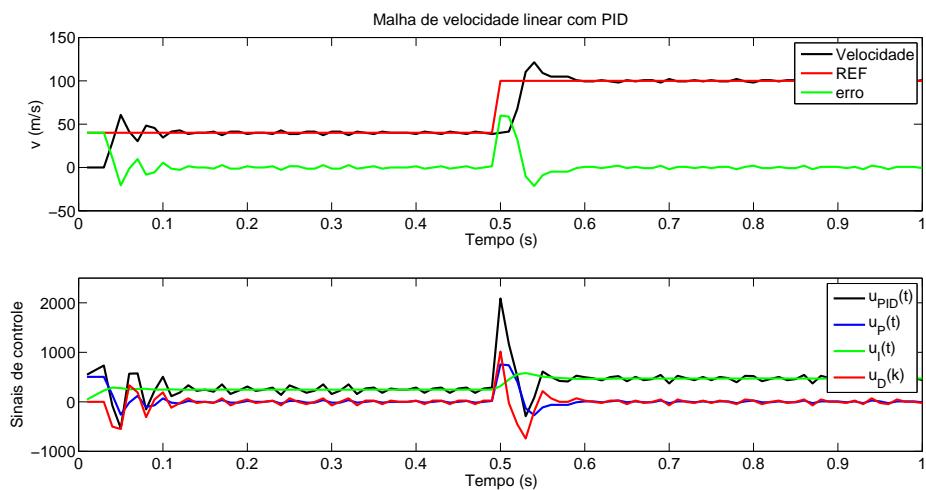
Tabela 4 – Parâmetros PID para a malha de velocidade baseados pela tabela do Ziegler-Nichols

Tipo de Controlador	k_P	k_I	k_D
PID	13.2127	660.6347	0.0661

Fonte: autor (2017)

Estes valores foram utilizados no controlador discreto do Micromouse, e a robustez pode ser observada na Figura 30. A saída de velocidade segue a referência corretamente, com *overshoots* esperados para entradas em degrau (mudança brusca de referência), e erro em regime permanente nulo com a inserção de um integrador através do PID. Os sinais de controle também estão enfatizados.

Figura 30 – Saída de velocidade utilizando o os parâmetros PID do método Ziegler-Nichols



Fonte: autor (2017)

2.4.3.2 Método do Relé Sequencial para sintonia de sistemas MIMO

Segundo Tamura e Ohmori (2006), muitas vezes, é difícil aplicar o método de autossintonia ao sistema MIMO (múltiplas entradas e múltiplas saídas), como é o do Micromouse, porque não é tão bem pesquisado como os sistemas SISO. Embora existam várias pesquisas do controle PID para o sistema MIMO, elas são restritas ao sistema de fase estável e/ou mínima em muitos casos. Os sistemas multivariáveis

MIMO podem ser representados por matriz de funções de transferência (KATSUHIKO, 2011). Nesta representação, um sistema multivariável com j entradas u_1, u_2, \dots, u_j e i saídas y_1, y_2, \dots, y_i , que definem os vetores Y de saídas e U de entradas, são dadas por (2.7).

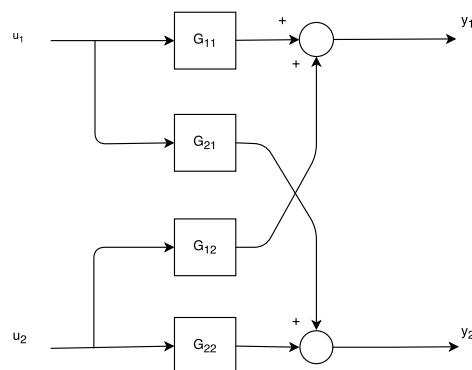
$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \end{bmatrix}; U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_j \end{bmatrix} \quad (2.7)$$

Na forma matricial, considerando um sistema linear, controlável e observável, a matriz de transferência é dado por (2.8).

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \\ \vdots \\ Y_i(s) \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} & \cdots & G_{1j} \\ G_{21} & G_{22} & \cdots & G_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ G_{i1} & G_{i2} & \cdots & G_{ij} \end{bmatrix} \times \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_j \end{bmatrix} \quad (2.8)$$

A Equação (2.8) fornece como resultado uma matriz de transferência de ordem ixj e cada elemento individual $G_{ij}(s)$ de $G(s)$ representa a função de transferência da malha respectiva de controle $y_i - u_j$, que, por sua vez, relaciona a variável manipulada u_j à variável controlada y_i . A Figura 31 apresenta o diagrama de blocos para esta representação de um sistema 2x2. $G_{21}(s)$ e $G_{12}(s)$ representam as interações entre as malhas do sistema.

Figura 31 – Diagrama de blocos do sistema MIMO 2x2



Fonte: adaptado de Barçante (2011)

2.4.3.3 Identificação dos motores via Mínimos Quadrados Não-Recursivos

Considerando que o sistema motor-*encoder* do robô seja caracterizado por uma entrada, $u(t)$, uma saída, $y(t)$, uma perturbação, $e(t)$, e com função de transferência discreta linear na forma (2.9), cuja representação por uma equação a diferença pode ser vista em (2.10).

$$A(z^{-1})y(t) = z^{-d}B(z^{-1})u(t) + e(t), \text{ onde}$$

$$\begin{aligned} A(z^{-1}) &= 1 + a_1z^{-1} + \dots + a_{na}z^{-na} \\ B(z^{-1}) &= b_0 + b_1z^{-1} + \dots + b_{nb}z^{-nb} \end{aligned} \quad (2.9)$$

$$\begin{aligned} y(t) &= -a_1y(t-1) - a_2y(t-2) - \dots - a_{na}y(t-na) + b_0u(t-d) + b_1u(t-d-1) + \\ &\dots + b_{nb}u(t-d-nb) + e(t) \end{aligned} \quad (2.10)$$

Definindo o *vetor de medidas*, $\phi(t)$, de dimensão $(na+nb+1) \times 1$, Equação (2.11), e o *vetor de parâmetros*, $\theta(t)$, Equação (2.12), de dimensão $(na + nb + 1) \times 1$, pode-se reescrever a Equação (2.10) como mostra a Equação (2.13), que é denominado *modelo de regressão linear* (COELHO; COELHO, 2015).

$$\varphi^T(t) = [-y(t-1) - y(t-2) \dots - y(t-na)u(t-d)\dots u(t-d-nb)] \quad (2.11)$$

$$\theta^T(t) = [a_1 \ a_2 \ \dots \ a_{na} \ b_0 \ b_1 \ \dots \ b_{nb}] \quad (2.12)$$

$$y(t) = \varphi^T(t)\theta(t) + e(t) \quad (2.13)$$

Supondo que são realizadas N medidas, suficientes para determinar os parâmetros a_i e b_i , então tem-se as matrizes (2.14).

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} \varphi^T(0) \\ \varphi^T(1) \\ \vdots \\ \varphi^T(N-1) \end{bmatrix} \times \theta(t) + \begin{bmatrix} e(0) \\ e(1) \\ \vdots \\ e(N-1) \end{bmatrix} \quad (2.14)$$

$$G(z^{-1}) = \frac{z^{-1}(b_1 + z^{-1}b_2)}{1 + z^{-1}a_1 + z^{-2}a_2} \quad (2.15)$$

A maioria dos processos reais pode ser modelado para ordem 2. Considerando (2.9) com $na = nb = 2$ e $d = 1$, o modelo de equação discreto $G(z)$ de segunda ordem para os motores pode ser representado por (2.15). A representação matricial de (2.14) está na Equação (2.16), onde a matriz observação está em (2.18), para sistema de segunda ordem, e o vetor de saída é dado por (2.17), onde a matriz ϕ não é quadrada, isto é, tem mais linhas do que colunas.

$$Y = \phi^T(t)\theta(t) + E \quad (2.16)$$

$$Y^T = [y(0)y(1)y(2) \cdots y(N-1)] \quad (2.17)$$

$$\phi = \begin{bmatrix} -y(-1) & -y(-2) & u(-d) & u(d-1) \\ -y(0) & -y(-1) & u(1-d) & u(-d) \\ -y(1) & -y(0) & u(2-d) & u(-d) \\ \dots & \dots & \dots & \dots \\ -y(N-2) & -y(N-3) & u(N-d-1) & u(N-d-2) \end{bmatrix} \quad (2.18)$$

A estimativa do vetor de parâmetros, $\hat{\theta}$, pode ser obtida pelo procedimento dos mínimos quadrados, que resulta na Equação (2.19). O estimador dos mínimos quadrados (2.19) é uma transformação linear sobre Y (função linear das medidas) e, assim, é denominado *estimador linear* (COELHO; COELHO, 2015).

$$\hat{\theta} = [\phi^T \phi]^{-1} \phi^T Y \quad (2.19)$$

Na aplicação do estimador dos mínimos quadrados, todas as medidas foram coletadas e posteriormente analisadas, técnica de identificação *offline*. Uma entrada aleatória é gerada através da função `rand()`, em C, e a cada período de amostragem, o microcontrolador configura um novo valor aleatório para a entrada do processo. Os valores de entrada e saída são enviados para o *MATLAB* e acumulados em vetor. Após feita a coleta, a matriz observação (2.18) é gerada. Ao aplicar (2.19), os coeficientes de (2.15) foram obtidos para os motores. Seus valores estão na Tabela 5.

Tabela 5 – Parâmetros da função de transferência estimados para os motores (N = 200)

$G_L(z)$	Motor Esquerdo	$G_R(z)$	Motor Direito
a_1	-0,5915	a_1	-0,6096
a_2	-0,0767	a_2	-0,0654
b_1	0,0149	b_1	0,0149
b_2	0,0142	b_2	0,0131

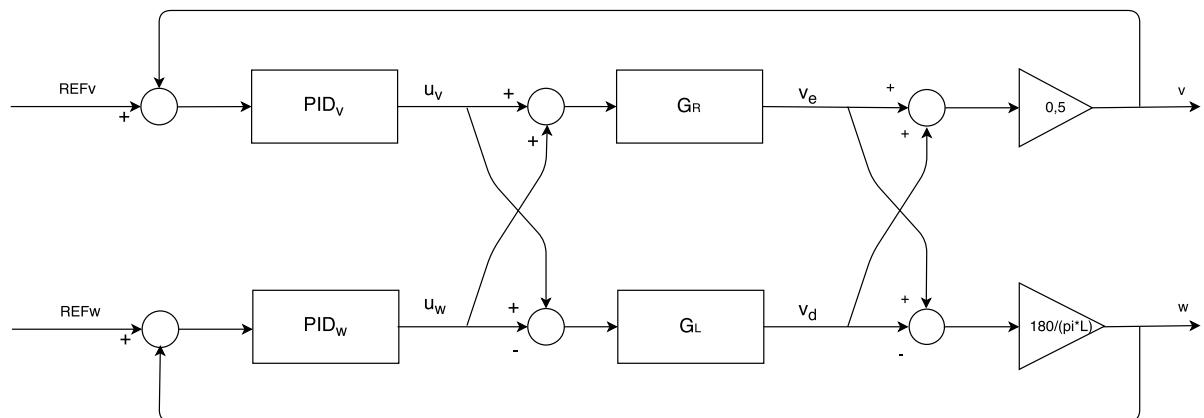
Fonte: autor (2017)

2.4.3.4 Sistema realimentado proposto

O sistema de controle proposto para o Micromouse é mostrado na Figura 32. O processo contém dois sistemas MIMO, em cascata, e dois blocos PID (um para a malha de velocidade linear e outro para a malha de velocidade angular), sugeridos por Su, Huang e Lee (2013).

Com a identificação da função de transferência dos motores, foi possível a simulação de toda a malha de controle no Simulink.

Figura 32 – Diagrama de blocos do sistema MIMO proposto para o Micromouse com dois blocos PID em malha fechada



Fonte: autor (2017)

Para identificação do processo e sintonia dos dois controladores, foram utilizados dois relés para sintonia em sequência, conforme sugere o trabalho de Barçante (2011), para sistemas deste tipo.

A seguir, é apresentado o projeto dos perfis de curva e de velocidade, que são as entradas de referência do sistema do controle em *malha fechada*.

2.4.4 Projeto de perfis de velocidade

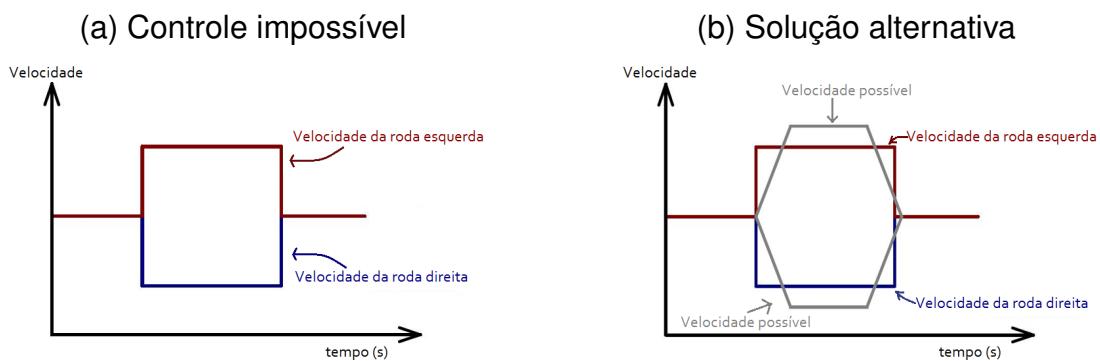
O controlador proposto implementado é responsável por ajustar as saídas do processo através dos valores de referência do sistema MIMO. O robô poderá realizar qualquer tipo de trajetória, desde que se crie perfis de reta e de curva.

Porém, os seguintes detalhes são importantes ao construir um perfil de curva:

- se a referência de velocidade linear é constante, e a referência de velocidade angular é nula, então o robô seguirá em frente com a mesma velocidade linear;
- em um sistema ideal, com aceleração infinita, se o robô deseja realizar uma curva com um raio R , e já com as velocidades de referência constante $v_0 \neq 0 \text{ m/s}$ e $\omega_0 = 0 \text{ rad/s}$, a velocidade angular de referência deverá sair de ω_0 imediatamente para $\omega = (v_0/R) \text{ rad/s}$ no instante da curva;
- ao realizar certo ângulo θ , a velocidade angular deverá sair imediatamente de ω para ω_0 , cessando o período de curva.

Todavia, num sistema real, a velocidade angular não muda abruptamente, nem tampouco com controlador mais eficiente, como mostra a Figura 33(a). Uma solução alternativa e prática é acelerar uma roda e desacelerar a outra, de forma gradativa, como é mostrado na Figura 33(b).

Figura 33 – Velocidades das rodas em uma curva



Fonte: autor (2017)

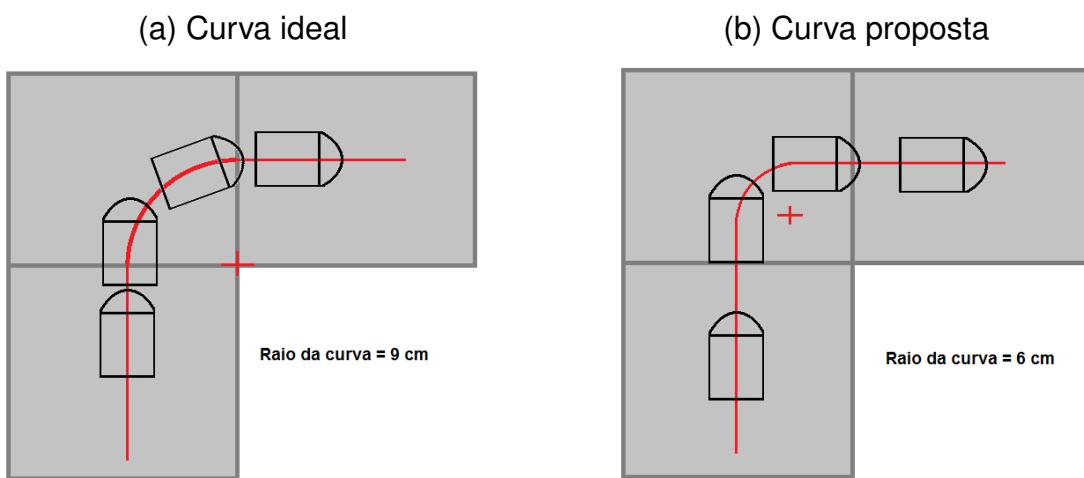
2.4.4.1 Curva desejada

O algoritmo utilizado é o *Flood Fill tradicional*, que indica as direções FRENTE, ESQUERDA, DIREITA e VOLTA. O sentido sempre é dado antes do robô entrar na próxima

célula, como mostra a Figura 22. Portanto, o robô só poderá realizar curvas de 90 graus, para curvas à esquerda e à direita, e um giro de 180 graus em seu próprio eixo quando o robô precisa retornar.

Porém, como o robô precisa se posicionar de frente para a próxima célula para obter os obstáculos corretamente através dos sensores IR, após a curva, a trajetória semicircular, com o ponto de curvatura sobre o canto da célula, de raio igual à metade da largura da célula, conforme é mostrado na Figura 34(a), deverá ser alterada para uma curva *mais fechada*, conforme mostra a Figura 34(b). Desta forma, o robô, ao terminar a curva, poderá ler os obstáculos ainda na célula anterior, com 3 cm de distância para a próxima célula.

Figura 34 – projeto de curvas

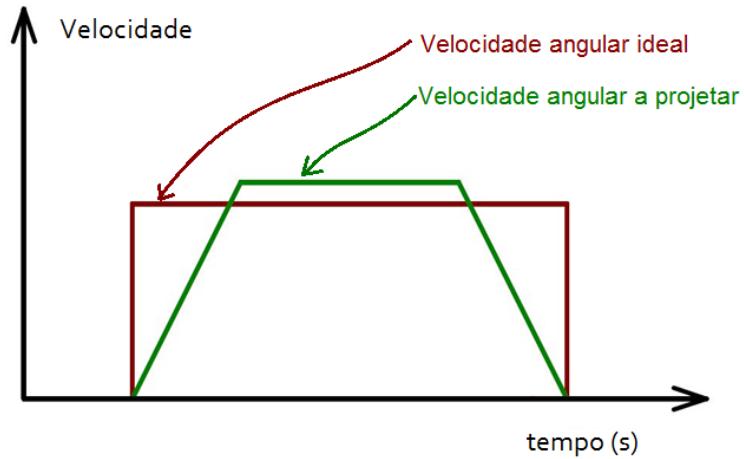


Fonte: autor (2017)

2.4.4.2 Determinando o perfil de curva de 90 graus

O robô sempre se moverá no labirinto com velocidade linear fixa, portanto, esta será a velocidade tangencial do movimento circular nos momentos de curva. Como o sistema não pode alterar abruptamente a velocidade angular, a solução viável é a aceleração em rampa, conforme mostra o gráfico de velocidade a projetar da Figura 35. O ângulo θ feito pelo robô será o mesmo da trajetória ideal quando a área dos gráficos são iguais.

Diante dos fatos, os cálculos para determinar o tempo de curva e velocidade angular ideal são mostrados em (2.20). O tempo de curva é baseado no gráfico de velocidade angular ideal e deve ser igual ao ângulo de curva desejado de 90 graus.

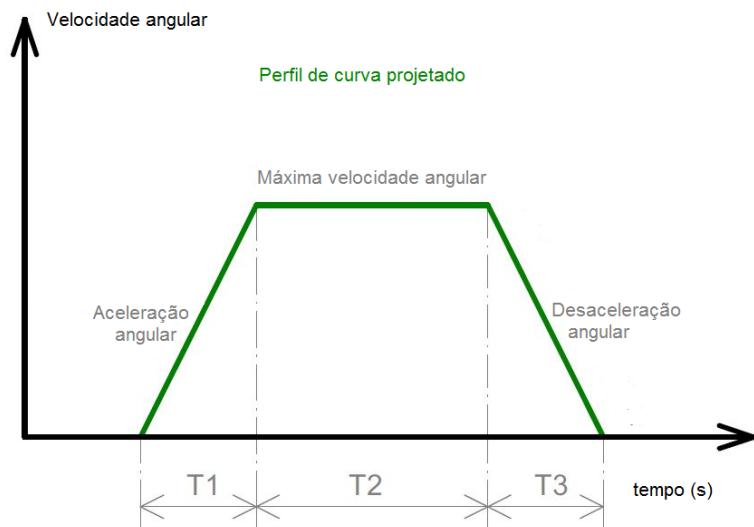
Figura 35 – Velocidade angular ideal x atual

Fonte: autor (2017)

$$\omega_{ideal} = (180 \times v_0) / (R \times \pi)$$

$$\theta_{curva} = 90^\circ \quad (2.20)$$

$$\theta_{curva} = \omega_{ideal} \times t_{curva} \Rightarrow t_{curva} = 90^\circ / \omega_{ideal}$$

Figura 36 – Visão geral do perfil de curva proposto

Fonte: autor (2017)

Com o tempo de curva, pode-se determinar a velocidade angular máxima do gráfico da Figura 36, considerando o tempo T_1 igual a T_3 e sendo 20% do tempo de

curva. Realizando a integral do gráfico de figura e igualando a 90 graus, tem-se ω_{max} , conforme mostra (2.21).

$$t_{curva} = T_1 + T_2 + T_3$$

$$T_1 = T_3 = 0.2 \times t_{curva}$$

$$T_2 = t_{curva} - 2 \times T_1 \Rightarrow T_2 = 0.6 \times t_{curva}$$

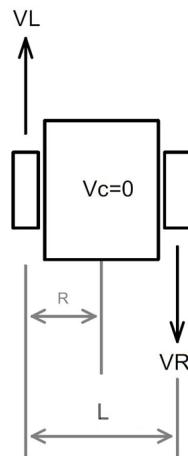
$$\omega_{max} = 90/(T_1 + T_2)^o/s$$

Portanto, para os tempos T_1 e T_3 , a velocidade angular deverá ser incrementada e decrementada à taxa de $\omega_{max}T_s/T_1$, onde T_s é a taxa de amostragem do sistema de controle, para o robô realizar uma curva de 90 graus. A diferença entre a curva à esquerda e a curva à direita é somente o sinal do ω_{max} .

2.4.4.3 Determinando o perfil de curva para giro de 180 graus em seu próprio eixo

O giro em seu próprio eixo não precisa de um tempo fixo, como é necessário para a curva de 90 graus. A velocidade linear para giro é nula. Desta forma, as velocidades das rodas são forçadas a ter sinais trocados, conforme mostra a Figura 37.

Figura 37 – O giro do robô em seu próprio eixo



Fonte: autor (2017)

Portanto, para o projeto Micromouse, o tempo para curva foi baseado no tempo T_2 do gráfico de Figura 36. Foi determinado que o robô deve girar 150 graus durante o tempo T_2 e 30 graus durante a aceleração e desaceleração da velocidade angular. As

fórmulas para determinação dos tempos T_1 , T_2 e T_3 para giro de 180 graus, bem como o tempo de curva, estão em (2.22).

$$T_2 = 150/\omega_{max}$$

$$T_1 = T_3 = (15 \times 2)/\omega_{max} \quad (2.22)$$

$$t_{curva} = T_1 + T_2 + T_3$$

2.4.4.4 Implementando os perfis no Micromouse

Uma função chamada `mover_robo()` foi criada para executar os perfis de curva. Ela é sempre executada após o algoritmo de controle. Uma terceira máquina de estados foi criada, sendo que os estados são mudados após os tempos de rampa e de curva cessarem. Logo a seguir, são listados os estados e suas funções:

- a) RAMPA_VEL: neste estado, o robô realizará uma aceleração constante de **velocidade linear**. Uma função `rampa_vel()` foi criada para realizar o incremento de velocidade de acordo com o tempo de rampa escolhido;
- b) DESCIDA_VEL: neste estado, o robô realizará uma rampa de desaceleração constante da velocidade linear. A mesma função para a rampa de aceleração é usada, porém, para decrementar o valor de velocidade por um tempo pré-determinado;
- c) RAMPA_ANG: o robô realizará uma rampa de **velocidade angular**. Uma função `rampa_ang()` foi criada para realizar o incremento de velocidade de acordo com o tempo de rampa escolhido;
- d) DESCIDA_ANG: neste estado, o robô realizará uma rampa de desaceleração constante da velocidade angular. A mesma função para a rampa de aceleração é usada, porém, para decrementar o valor de velocidade por um tempo pré-determinado;
- e) MANTER_VEL: este estado mantém o valor da velocidade linear por um tempo pré-determinado;
- f) MANTER_ANG: este estado mantém o valor da velocidade angular por um tempo pré-determinado.

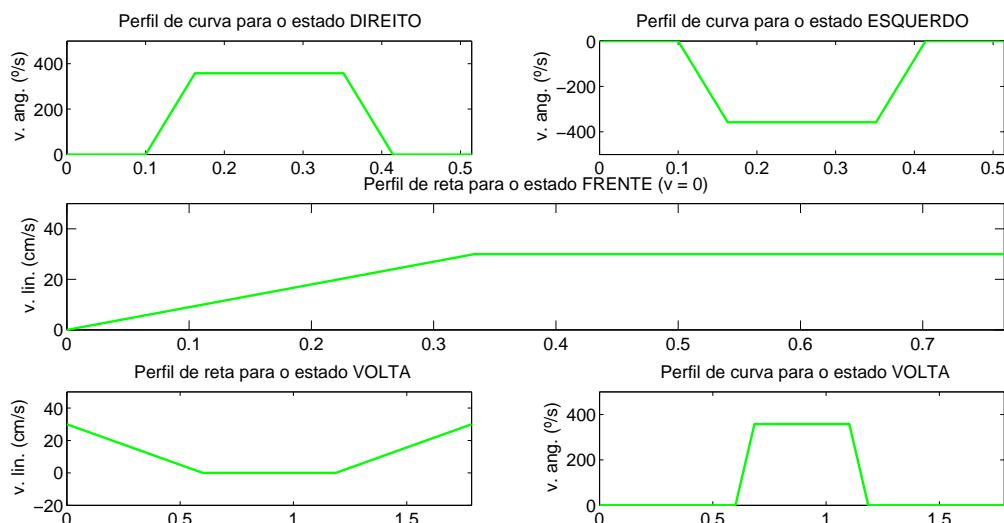
A Tabela 6 mostra a máquina de estado principal, controlado pelo algoritmo *Flood Fill*, e como ocorre a transição dos estados da submáquina de estados para criação dos perfis de curva e de reta.

Tabela 6 – Máquinas de estado para geração dos perfis de reta e de curva

Máquina de estado		Submáquina de estado	
estados	estado inicial	estado	próximo estado
FRENTE	Se $v = 0$, RAMPA_VEL	RAMPA_VEL	MANTER_VEL
•	Se $v \neq 0$, MANTER_VEL	MANTER_VEL	FLOODFILL
ESQUERDA	MANTER_VEL	RAMPA_ANG	MANTER_ANG
•	•	DESCIDA_ANG	FLOODFILL
•	•	MANTER_VEL	RAMPA_ANG
•	•	MANTER_ANG	DESCIDA_ANG
DIREITA	MANTER_VEL	RAMPA_ANG	MANTER_ANG
•	•	DESCIDA_ANG	FLOODFILL
•	•	MANTER_VEL	RAMPA_ANG
•	•	MANTER_ANG	DESCIDA_ANG
VOLTA	DESCIDA_VEL	RAMPA_VEL	FLOOD FILL
•	•	RAMPA_ANG	MANTER_ANG
•	•	DESCIDA_VEL	MANTER_VEL
•	•	DESCIDA_ANG	RAMPA_VEL
•	•	MANTER_VEL	RAMPA_ANG

Fonte: autor (2017)

Por exemplo, considerando uma velocidade linear v de 30 cm/s, e raio de curva R de 6 cm, e utilizando as fórmulas (2.20) a (2.22), tem-se o perfil de curva dos estados da máquina de estados da Tabela 6 ao longo do domínio do tempo, e a Figura 38 mostra a visão geral do perfil de curva.

Figura 38 – Visão geral dos perfis de curva para os quatro estados

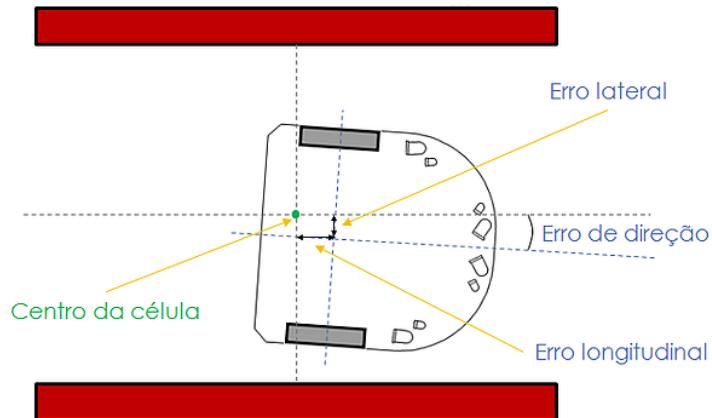
Fonte: autor (2017)

Então, cada estado da máquina de estados será responsável por modificar as entradas de referências do controle MIMO proposto, sendo que cada estado tem um tempo pré-determinado.

2.4.5 Controle dos erros de posicionamento no labirinto

Numa trajetória real, os controladores estão expostos a ruídos iminentes do sistema. Com isto, em *malha aberta*, o robô poderá sair da sua trajetória, acumulando erros de posicionamento ao longo do caminho. Derrapagens das rodas também causam erros deste tipo. A Figura 39 mostra os dois erros de posicionamento do Micromouse, que devem ser corrigidos.

Figura 39 – Os erros de posicionamento do Micromouse no Labirinto



Fonte: adaptado de Silva (2015b)

Os sensores IR poderão servir como *feedback*. O robô, ao longo do percurso, sempre encontrará paredes. Com base nos valores das distâncias do robô às paredes, um sinal de controle entra como referência na entrada de velocidade angular, corrigindo, desta forma, a sua posição.

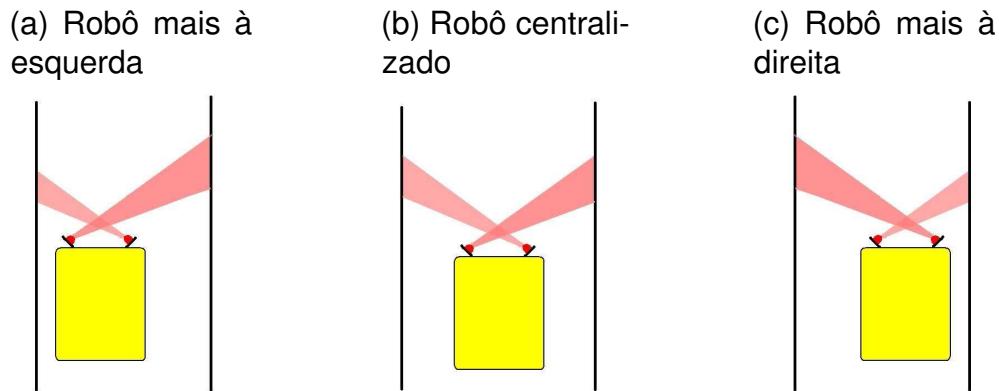
Os sensores tem os seguintes valores de tensão (em mV), quando o robô está centralizado:

- o sensor **I**, diagonal esquerdo, tem valor de tensão de 700 mV;
- o sensor **r**, diagonal direito, tem valor de tensão de 1150 mV.

Se o robô estiver mais à esquerda (Figura 40(a)), a velocidade angular deverá ser positiva para o robô desviar para o centro. Caso esteja mais à direita (Figura 40(c)),

o sinal de velocidade angular é negativo. O sinal de controle pelos sensores IR deverá ser nulo quando na posição correta (Figura 40(b)).

Figura 40 – Sensores como feedback para correção da posição no labirinto



Fonte: adaptado de Borges (2013)

A partir desta informação, caso tenha paredes laterais no percurso, os sensores poderão alimentar a referência através de um controlador proporcional, como mostra a tabela 7.

Tabela 7 – Controlador proporcional para o *feedback* dos sensores IR

Condição	Controle
Parede à esquerda	$u_{IR} = (l - 700) \times K_P;$ $REF_w = u_{IR};$
Parede à direita	$u_{IR} = (1148 - r) \times K_P;$ $REF_w = u_{IR};$

Fonte: autor (2017)

2.5 Observações finais

Neste capítulo foram abordadas as características de projeto das partes principais que compõem um projeto Micromouse. O Capítulo 3 segue com os resultados observados nas simulações do algoritmo e controle implementados e, em seguida, da união deles com o robô. Discussões acerca desses resultados são feitas, bem como análises pertinentes a respeito dos algoritmos executados durante o percurso real em labirinto.

3 Resultados Experimentais obtidos na solução do labirinto

Este capítulo apresenta os resultados observados e as análises realizadas a partir dos testes nas partes implementadas. A divisão das seções é a mesma anteriormente apresentada para a apresentação dos resultados individualmente. Finalmente, é realizada uma análise geral do sistema como um todo.

3.1 Hardware

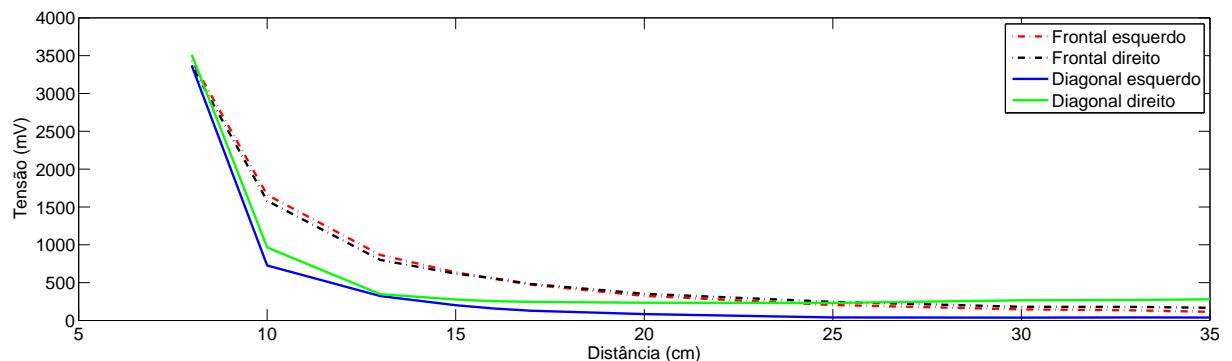
Os primeiros testes foram realizados com o robô, a fim de determinar algumas informações importantes, tais como número de pulsos por revolução gerados, verificação das relações de tensão por distância dos sensores de distância e comunicação via *bluetooth*. Estas informações são usadas nos algoritmos implementados.

Para validação do número de pulsos gerados por volta completa do eixo do motor, um programa básico foi utilizado. O módulo QEI do microcontrolador analisa os dois canais do *encoder* e incrementa um contador.

3.1.1 Sensores de distância infravermelhos

Testes foram feitos, com os sensores IR, e foi constatado que o valor de tensão aumenta à medida que o robô se aproxima do obstáculo, conforme pode ser observado na Figura 41. As medidas de distância foram realizadas do eixo das rodas até o obstáculo à frente.

Figura 41 – Parede à frente

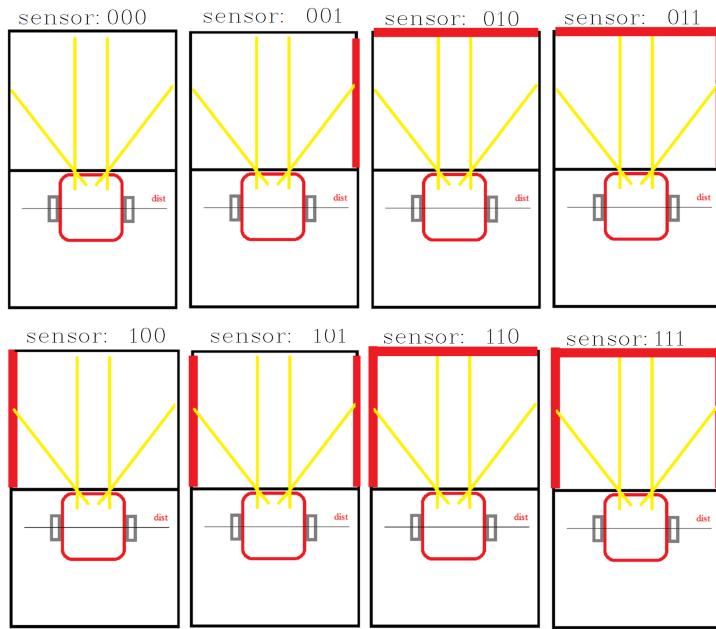


Fonte: autor (2017)

As medidas dos sensores frontais se diferenciaram dos sensores diagonais, porque os obstáculos se localizaram de frente com o robô.

Com os gráficos, pode-se determinar limiares de detecção de parede, que, através deles, são definidos os três bits da função de leitura dos sensores. A Figura 42 mostra os três bits para todas as possibilidades de obstáculos à frente.

Figura 42 – Retorno da função getSensoresParede() com obstáculos à frente



Fonte: autor (2017)

Os limiares para detecção de paredes foram determinados da seguinte forma, considerando sempre a pior posição para a medida das distâncias:

- o robô é posicionado sempre na célula anterior, para detectar as paredes da célula seguinte a tempo do algoritmo de resolução indicar a nova orientação e sentido;
- para detecção da parede à frente, a distância do eixo das rodas até o vértice da próxima célula é de 6 cm, consequentemente, a 24 cm do obstáculo;
- para detecção da parede lateral, o robô era posicionado no canto oposto;
- realizou-se a leitura dos sensores para cada condição mostrada na Figura 42;
- a medida mínima realizada para cada sensor era considerada para limiar de detecção das paredes.

A Tabela 8 mostra os limiares obtidos experimentalmente.

Tabela 8 – Valor mínimo de tensão para detecção das paredes

Sensor	Tensão (mV)
Frontal	220
Diagonal esquerdo	600
Diagonal direito	240

Fonte: autor (2017)

3.1.2 Encoders

Um teste foi feito para determinar a quantidade de pulsos, com o objetivo de conhecer o valor certo de número de pulsos por volta. Em cada roda, uma marcação de referência foi realizada. Um programa simples para este teste foi gravado no micro-mouse e a cada segundo era retornado o valor do registrador. Em cada roda, uma volta completa foi girada com a mão. Os *encoders* geraram 358 pulsos por volta, ao invés de 360 pulsos.

3.2 Algoritmo

O resultado da construção do algoritmo no *FALCON C++ IDE* pode ser observado visualmente, através dos símbolos, no próprio programa compilado. Mais à frente, é mostrado o algoritmo teste real, que será feito em conjunto com o controle em malha fechada do Micromouse.

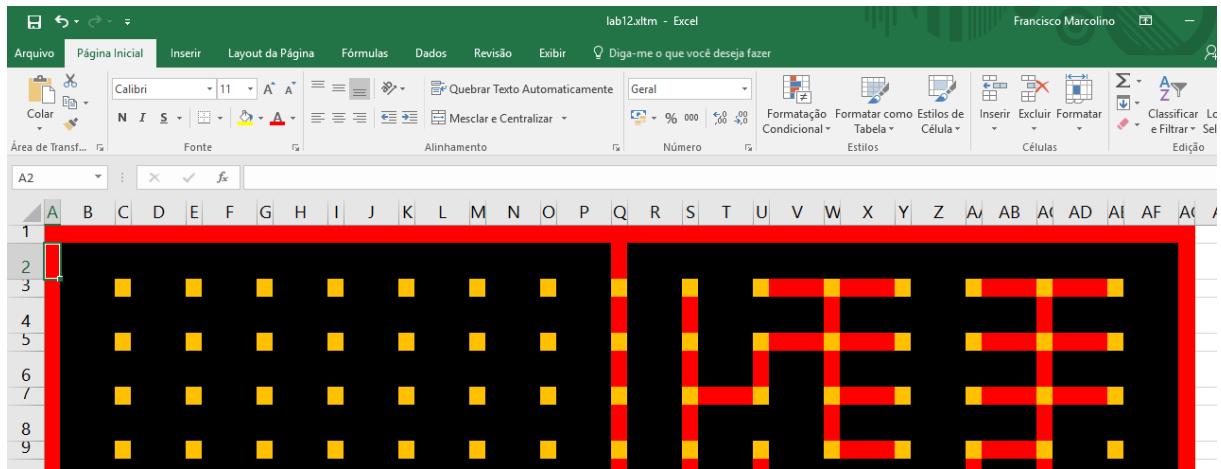
3.2.1 Modelando um labirinto

Os resultados observados do algoritmo desenvolvido no *FALCON C++ IDE* dependem da modelagem do labirinto. Para tanto, foi criada mais uma variável dentro da estrutura de dados da célula, que, por sua vez, pode ser acessada através de `celula[x][y].parede[z]`. Desta forma, os bits das paredes da célula à frente podem ser utilizados pelo algoritmo, quando solicitado, para emular os sensores de distância do Micromouse.

No algoritmo, uma função de inicialização realiza a leitura de dois arquivos de extensão `.txt` (arquivos que contém os bits das paredes horizontal e vertical) e atualiza na estrutura de dados todas as paredes do labirinto modelado. Desta forma, pode-se simular qualquer labirinto e certificar-se do correto funcionamento do algoritmo criado, até mesmo comparar os resultados com o simulador *Micro Mouse Maze Editor and Simulator*.

Através do *EXCEL*, foi criada uma plataforma conversora de labirinto. Visualmente pode-se configurar as paredes, e o mesmo é encarregado de gerar informações binárias necessárias para executar o algoritmo criado no *FALCON C++ IDE*. As colunas e linhas são comprimidas e preenchidas com cores para dar forma às paredes, conforme pode ser observado com detalhes na Figura 43.

Figura 43 – EXCEL como ferramenta para modelagem de labirinto



Fonte: autor (2017)

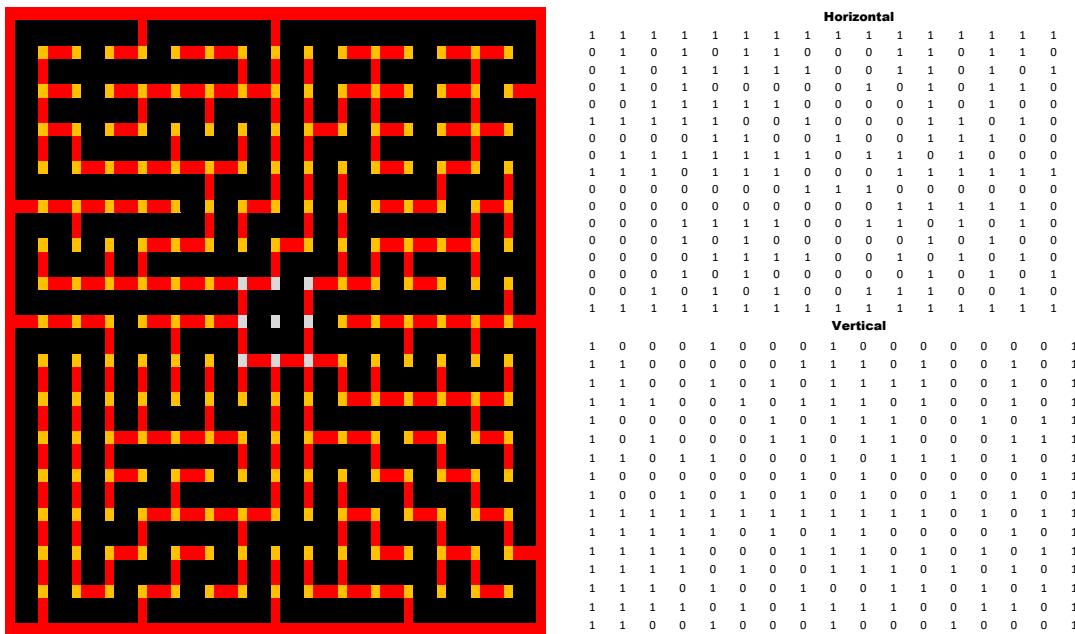
Os *bits* são formados através das fórmulas, que são inseridas nas células. As fórmulas são equações que podem executar cálculos, retornar informações, manipular o conteúdo de outras células, testar condições e mais. Uma fórmula sempre começa com um sinal de igual (=). Uma função macro do *EXCEL* foi utilizada para detectar cores de preenchimento, e ela retorna o número da cor. Desta forma, se a cor da célula é vermelha (número 255), então significa que existe uma parede e o *bit* 1 será selecionado pela função lógica condicional "SE", do *EXCEL*. Caso contrário, o valor gerado é 0. Logo a seguir é mostrada a fórmula para gerar o *bit* na célula B1, por exemplo:

```
=SE(gfCelColorName(B1)="255";"1";"0")
```

Na Figura 43, o valor retornado pela fórmula é 1, porque a célula B1 do *EXCEL* está preenchida com a cor vermelha. Desta forma, as matrizes de *bits* das paredes horizontais e verticais são geradas.

A Figura 44 mostra a modelagem de um labirinto *Seoul 01* no *EXCEL*, design de labirinto que exige esforço do algoritmo, ideal para testes. Logo ao lado da figura contém as matrizes binárias prontas para exportar para arquivos *.txt*.

Figura 44 – Modelagem do labirinto *Seoul* para simulação



Fonte: autor (2017)

3.2.2 Simulando o algoritmo no labirinto

Para validação do algoritmo construído, foi utilizado o desenho do labirinto *Seoul 01*, Figura 44, disponibilizado para testes no simulador *Micro Mouse Maze Editor and Simulator*. Segundo as estatísticas para o labirinto, cerca de 694 células devem ser atravessadas para completar a melhor corrida, e pela dificuldade, ele foi adotado para simulação.

Após modelar o labirinto escolhido, os arquivos *.txt* são atualizados através das matrizes de bits gerados pelo *EXCEL*. E no programa *FALCON C++ IDE*, o algoritmo é compilado e executado, de forma que apareça na janela de *Prompt de Comandos* do *Windows* os símbolos para modelagem do labirinto.

A cada estado, o programa imprime uma mensagem com o nome do estado em que o algoritmo se encontra, como pode ser visto na Figura 45. No estado *IMPRIMIR MAZE*, o algoritmo imprime de fato todo o labirinto visto pelo robô, ou seja, as paredes descobertas por ele, incluindo as marcações nas células visitadas e também o símbolo de direção e sentido na célula a qual o robô se encontra. Quando o robô chega no estado *ESPERAR_CELULA*, o programa espera o pressionar do *ENTER* para o algoritmo voltar para o estado *MOVER_ROBO* e prosseguir com o percurso de forma virtual. Desta forma, uma lista impressa é gerada e o percurso do robô pode ser acompanhado.

Também é possível o acompanhamento da atualização das distâncias das células, quando é preciso, a cada passo dado.

Figura 45 – Acompanhamento e simulação do algoritmo proposto

```

estado: MOVER ROBO...
robo.x 5 robo.y 7 orientacao N
estado: VERIFICAR OBSTACULOS...
estado: DEFINIR PASSO...
estado: IMPRIMIR MAZE...

Labirinto modelado:
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 14| 13| 12| 11| 10| 9| 8| 7| 7| 8| 9| 10| 11| 12| 13| 14|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 13| 12| 11| 10| 9| 8| 7| 6| 6| 7| 8| 9| 10| 11| 12| 13|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 12| 11| 10| 9| 8| 7| 6| 5| 5| 5| 6| 7| 8| 9| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 11| 10| 9| 8| 7| 6| 5| 4| 4| 4| 5| 6| 7| 8| 9| 10|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 10| 9| 8| 7| 6| 5| 4| 3| 3| 3| 4| 5| 6| 7| 8| 9|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 9| 8| 7| 6| 5| 4| 3| 2| 2| 3| 4| 5| 6| 7| 8| 9|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 8| 7| 6| 5| 4| 3| 2| 1| 1| 2| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 7| 6| 5| 4| 3| 2| 1| 0| 0| 1| 2| 3| 4| 5| 6| 7|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 18 * 18 * 17| 4| 3| 2| 1| 0| 0| 1| 2| 3| 4| 5| 6| 7|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 20| 18|* 16| 5| 4| 3| 2| 1| 2| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 21| 17|* 15| 6| 5| 4| 3| 2| 1| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 22| 16|* 14| 7| 8| 9| 8| 8| 5| 5| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 23| 15|* 13|* 10| 9| 8| 7| 6| 6| 4| 5| 6| 7| 8| 9|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 24| 14|* 12|* 11| 9| 8| 7| 6| 5| 6| 7| 8| 9| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 25| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 1|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 26| 14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
Pressione qualquer tecla para continuar. . .

```

Labirinto visto pelo robo

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 14| 13| 12| 11| 10| 9| 8| 7| 7| 8| 9| 10| 11| 12| 13| 14|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 13| 12| 11| 10| 9| 8| 7| 6| 6| 6| 7| 8| 9| 10| 11| 12| 13|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 12| 11| 10| 9| 8| 7| 6| 5| 5| 5| 6| 7| 8| 9| 10| 11| 12|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 11| 10| 9| 8| 7| 6| 5| 4| 4| 4| 5| 6| 7| 8| 9| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 10| 9| 8| 7| 6| 5| 4| 3| 3| 3| 4| 5| 6| 7| 8| 9| 10|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 9| 8| 7| 6| 5| 4| 3| 2| 2| 3| 4| 5| 6| 7| 8| 9|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 8| 7| 6| 5| 4| 3| 2| 1| 1| 2| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 7| 6| 5| 4| 3| 2| 1| 0| 0| 1| 2| 3| 4| 5| 6| 7|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 18 * 17| 4| 3| 2| 1| 0| 0| 1| 2| 3| 4| 5| 6| 7|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 20| 18|* 16| 5| 4| 3| 2| 1| 2| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 21| 17|* 15| 6| 5| 4| 3| 2| 1| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 22| 16|* 14| 7| 8| 9| 8| 8| 5| 5| 3| 4| 5| 6| 7| 8|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 23| 15|* 13|* 10| 9| 8| 7| 6| 6| 4| 5| 6| 7| 8| 9|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 24| 14|* 12|* 11| 9| 8| 7| 6| 5| 6| 7| 8| 9| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 25| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 1|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|* 26| 14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
Pressione qualquer tecla para continuar. . .

```

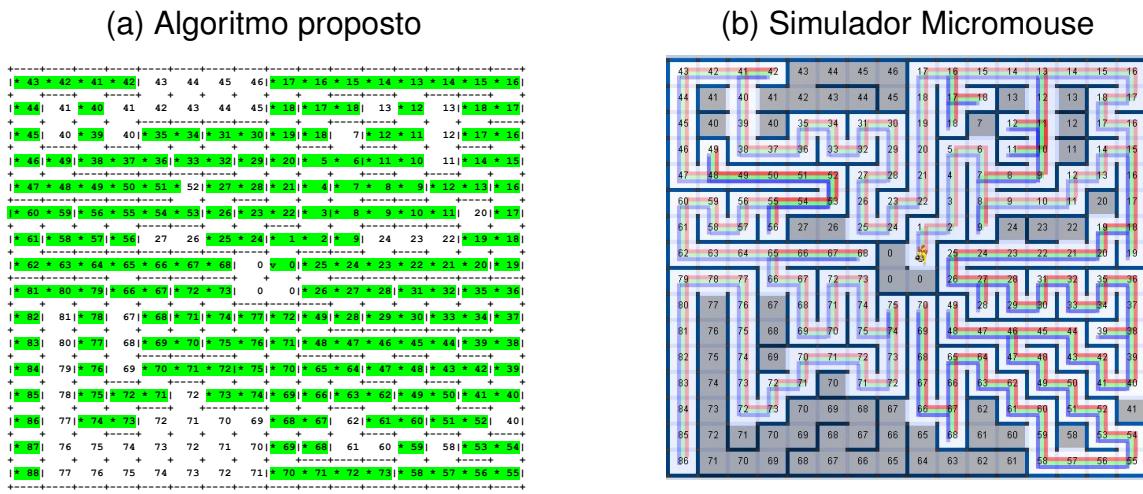
Fonte: autor (2017)

Para validação do algoritmo, foram realizadas comparações das distâncias das células a cada percurso virtual dos simuladores.

A Figura 46 (a) mostra a distribuição das distâncias ao final de uma corrida, com detalhe na marcação das células visitadas com asterisco (em verde). Em comparação com o percurso do algoritmo do simulador *Micro Mouse Maze Editor and Simulator* da Figura 46 (b), são bastante idênticos ambos os percursos. Prioridade para escolha da orientação entre vizinhanças de mesmo número, nos algoritmos de tomada de deci-

são, causam estas pequenas diferenças entre os percursos. Porém, nas simulações, observou-se que o algoritmo proposto converge sempre para o mesmo caminho do algoritmo do simulador *Micro Mouse Maze Editor and Simulator*, ao aumentar o número de corridas.

Figura 46 – Percurso do robô no labirinto Seoul - primeira corrida real



Fonte: autor (2017)

No algoritmo proposto, a cada corrida (tanto da IDA como da VOLTA) as distâncias das células são reinicializadas, porém, as informações das paredes descobertas permanecem em RAM. Com esta estratégia, utilizando a mesma estrutura de memória, resulta em uma economia de 50% de RAM neste quesito.

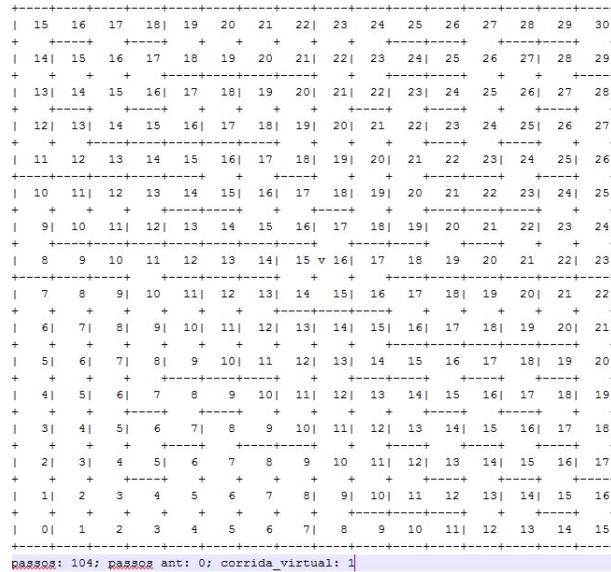
Então, ao final de uma corrida, as distâncias das células são redefinidas e inicia-se a *varredura virtual* do robô. A condição de parada é obtida quando o número de passos da corrida anterior se torna idêntico ao número de passos da última tentativa, em um processo de aproximação sucessiva. Assim, o 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 47 mostra a redefinição das distâncias para permitir a volta do robô e, depois, irá começar sua corrida virtual. A Figura 48 mostra sua terceira corrida virtual. Já a Figura 49 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.

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

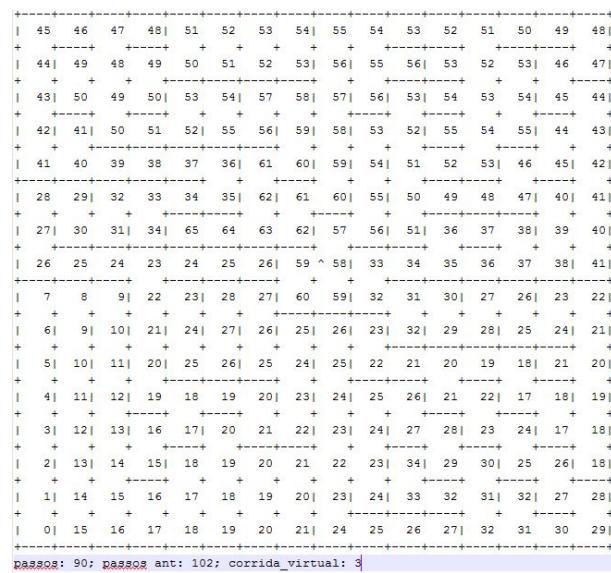
caminho de ida como também o caminho de volta.

Figura 47 – Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 1



Fonte: autor (2017)

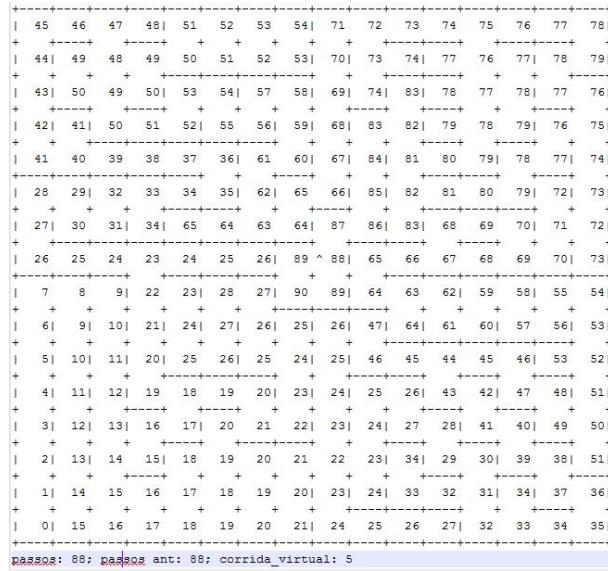
Figura 48 – Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 3



Fonte: autor (2017)

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 50 (a) e 50 (b). O percurso otimizado está em verde.

Figura 49 – Atualização das distâncias das células para o robô voltar à celula de partida - corrida virtual 5



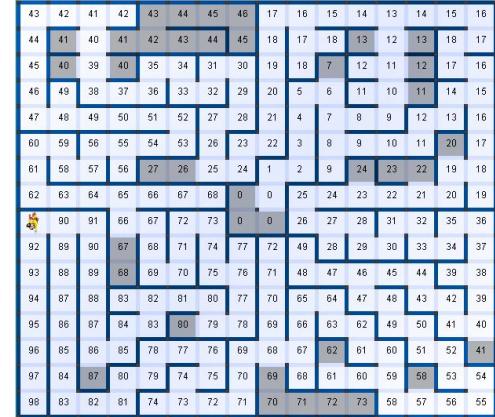
Fonte: autor (2017)

Figura 50 – Labirinto Seoul após 3 corridas reais

(a) Algoritmo proposto



(b) Simulador Micromouse



Fonte: autor (2017)

Algumas estatísticas foram feitas para os algoritmos *Flood Fill*, expostas na Tabela 9, 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 9 – Desempenho dos algoritmos no labirinto Seoul

Estatísticas	Algoritmo proposto	Micro Mouse
Únicas células atravessadas	224	218
Cél. p/ chegar ao centro (1^a 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

Fonte: autor (2017)

3.3 Controle em malha fechada do Micromouse

O controle proposto por Su, Huang e Lee (2013) foi implementado na plataforma Simulink, utilizando as funções de transferência $G(z)$ dos motores estimados pelo *Método dos Mínimos Quadrados Não Recursivos* e os parâmetros PID estimados pelo *Método sequencial do relé*.

Como referência para os controladores, um bloco chamado *Repeating Sequence Interpolated* permite a inserção de vetores de saída e vetores com valores de tempo; os perfis de curva e de reta poderão ser modelados por este bloco. A determinação destes vetores será mostrado adiante. Portanto, pode-se verificar a robustez dos controladores e, ao mesmo tempo, o percurso do micromouse num plano cartesiano.

Após as simulações, o controlador digital realimentado proposto foi implementado no Micromouse, com os mesmos parâmetros PID determinados pelo método do relé em simulação. Outra máquina de estados foi criada, com o objetivo de fornecer sinais de referência para os controladores. Enquanto o robô percorria o labirinto, as referências geradas de velocidades linear e angular para os controladores, as variáveis de controle e de saída eram enviadas para o *MATLAB*, de forma *online*, por conexão *bluetooth*, e gráficos foram gerados para validação do sistema de controle como todo.

3.3.1 Simulação do processo motor-encoder

A simulação do controle de Figura 32 foi construído no Simulink, com objetivo de testar os perfis de curva e de velocidade. Os controladores de velocidades devem ser bons rastreadores de referência, de forma a estabilizar as saídas para entradas de referência em degrau e em rampa.

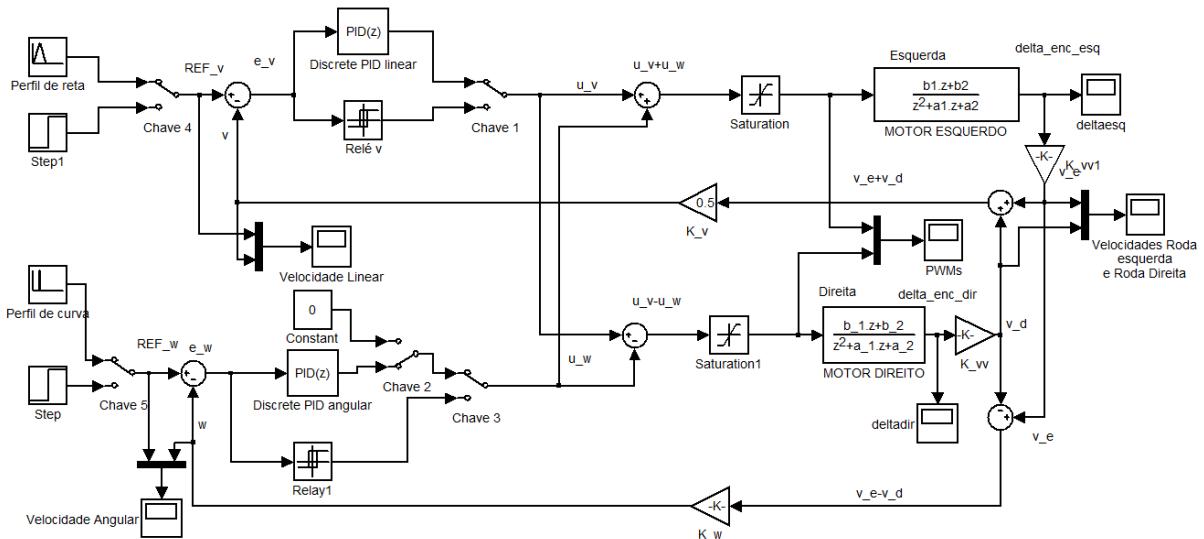
Os coeficientes estimados pelo Método dos Mínimos Quadrados foram postos via bloco de função de transferência discreta para emular os motores do Micromouse.

Os parâmetros do bloco PID foram extraídos a partir do Método do relé sequencial em simulação, e, posteriormente, foi incorporada uma parte que mostra a trajetória do ponto do centro geométrico do robô num plano XY.

3.3.1.1 Método do relé sequencial para sintonia do controlador PID

O Método do relé sequencial é um ótimo sintonizador. A técnica para sintonia do sistema MIMO é bem prática e simples. A Figura 51 mostra o esquema feito para sintonia do Método do relé sequencial. Nas duas malhas, os relés ficam disponíveis para oscilar o sistema, caso as chaves os selecionam. Para ambos os relés, a histerese escolhida é de $\varepsilon = 10$, e $d = 250$.

Figura 51 – Diagrama de blocos do Método do relé sequencial para o sistema MIMO proposto



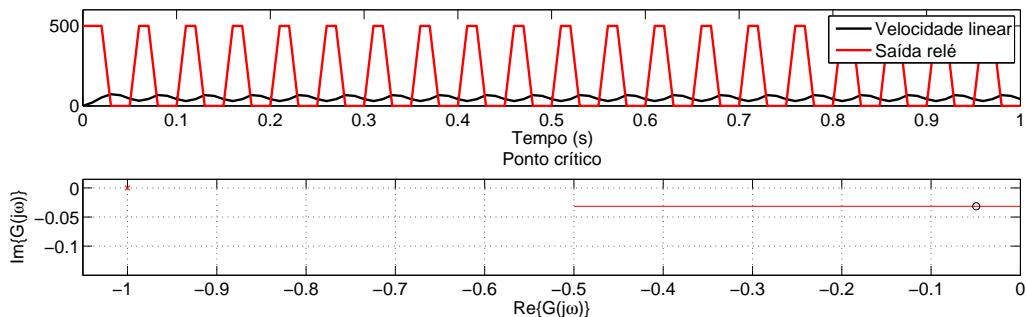
Fonte: autor (2017)

Então, para a primeira sequência, a chave 1 da Figura 51 selecionou o relé para a malha de velocidade linear, e a malha de velocidade angular ficou em aberto (Chave 2 e Chave 3 selecionam o bloco de constante zero). As chaves 4 e 5 ativaram os blocos *step* configurados para valor final de 50 cm/s e 100 °/s, respectivamente, para as malhas de velocidade linear e angular. Calculou-se os parâmetros PID com o método do relé SISO, através dos parâmetros do relé e do sinal de velocidade linear. Os sinais do relé e de saída, além do ponto do ganho crítico $G_v(j\omega)$ de Nyquist para a malha de velocidade, podem ser observados na Figura 52. Após, a Chave 1 seleciona o bloco PID com os parâmetros calculados pela tabela do Ziegler-Nichols.

Seguindo a sequência, configurando a chave 3, o relé da malha de velocidade angular é selecionado, de forma que o sinal fornecido por ele interfira em ambas as

malhas. Calcula-se os parâmetros PID do bloco, a partir do sinal de velocidade angular de saída, e o seleciona para controlar a malha da velocidade angular.

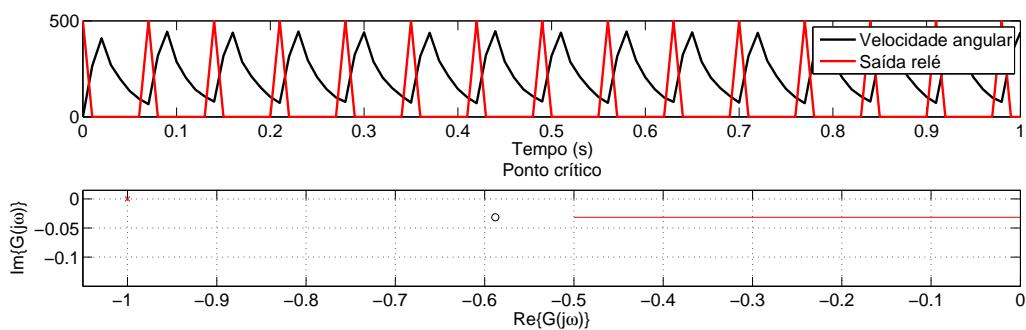
Figura 52 – Relé na malha de velocidade linear



Fonte: autor (2017)

Os sinais do relé e de saída de velocidade angular, além do ponto do ganho crítico $G_\omega(j\omega)$ de Nyquist para esta malha, podem ser observados na Figura 53.

Figura 53 – Relé na malha de velocidade angular



Fonte: autor (2017)

Este processo foi repetido até que não houvesse mais mudanças nos valores parâmetros dos controladores. Os processos, cujo K_pK_u estando entre 2 e 20, podem ser controlados pelos blocos PID sintonizados a partir da tabela do Ziegler-Nichols (COELHO; COELHO, 2015). A Tabela 10 mostra algumas grandezas críticas obtidas a partir do método. A Tabela 11 mostra os valores dos parâmetros PID para ambos os controladores do sistema MIMO. Comparando os pontos de ganho crítico $G_v(j\omega)$ e $G_\omega(j\omega)$ das malhas, a malha de velocidade é relativamente mais estável, em malha fechada, que a malha de velocidade angular.

Tabela 10 – Dados experimentais das saídas dos processos. Foram adotados $\varepsilon = 10$ e $d = 250$ para ambos os relés.

Sequência	Malha	a	$T_u(s)$	K_u	$\omega_u(rad/s)$	K_pK_u
1	V. Linear	18.6303	0.05	20.2499	125.6637	4.9157
2	V. Angular	183.2216	0.07	1.7399	89.7598	5.5154
3	V. Linear	18.6337	0.0500	20.2447	125.6637	4.9211
4	V. Angular	183.2216	0.0700	1.7399	89.7598	5.5154
5	V. Linear	18.6337	0.0500	20.2447	125.6637	4.9211

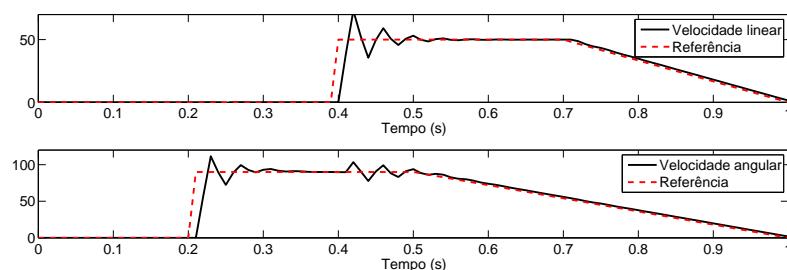
Fonte: autor (2017)

Tabela 11 – Parâmetros dos blocos PID estimados pelo Método do relé sequencial em simulação

Sequência	Malha	k_P	k_I	k_D
1	V. Linear	12.15	485.9985	0.0759
2	V. Angular	1.0439	29.8267	0.0091
3	V. Linear	12.1468	485.8737	0.0759
4	V. Angular	1.0439	29.8267	0.0091
5	V. Linear	12.1468	485.8737	0.0759

Fonte: autor (2017)

Figura 54 – Resultado da simulação do controle MIMO proposto



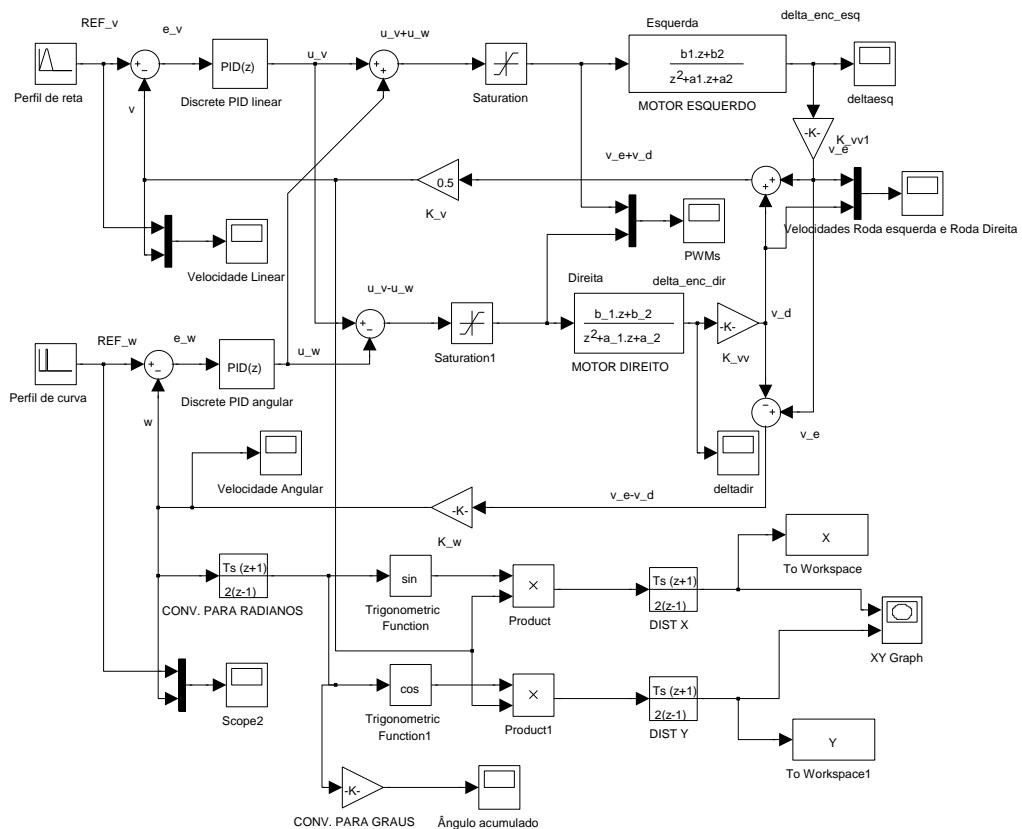
Fonte: autor (2017)

O resultado do controle de velocidade proposto já com os parâmetros PID calculados através do Método do relé sequencial pode ser observado na Figura 54. As saídas seguiram corretamente as referências dos controladores, para entradas em degrau e em rampa. Pode-se perceber também que quando há uma mudança brusca de referência de velocidade angular, há uma perturbação na saída da outra malha. Os *overshoots* são esperados para entradas em degrau, quando a sintonia é feita pelo método do Ziegler-Nichols. Porém, os perfis de curva e de reta são rampas de velocidade, e, portanto, os *overshoots* e as interferências são mínimos, quase eliminados.

3.3.1.2 Simulação das trajetórias

Para o desenvolvimento da simulação do sistema proposto, foi implementado o sistema motor-encoder do Micromouse no *Simulink*, do *MATLAB*. O esquema está na Figura 55. O modelo construído tenta aproximar-se ao sistema do Micromouse real. Os controladores utilizam os parâmetros PID da Tabela 11.

Figura 55 – Diagrama de blocos do controle implementado no *Simulink*



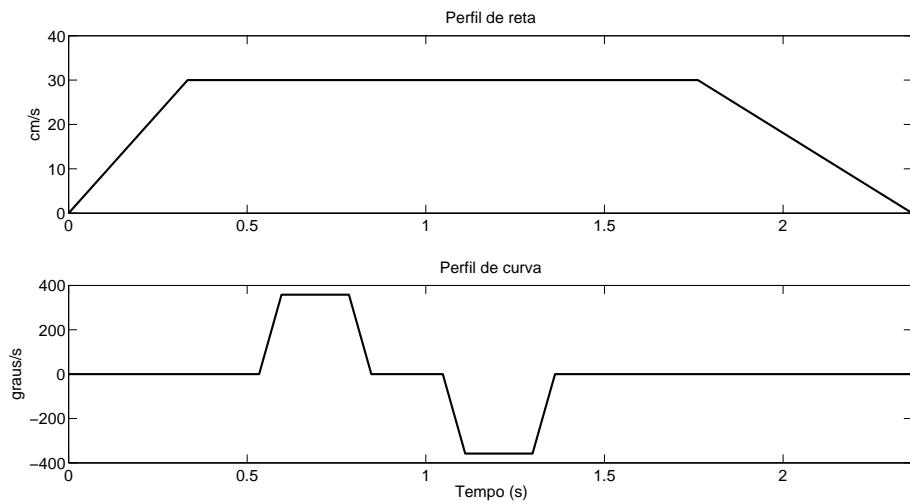
Fonte: autor (2017)

A trajetória pode ser construída a partir da integração da velocidade linear do robô (utilizando o método trapezoidal) e do ângulo θ formado no plano cartesiano. A coordenada do Micromouse no plano XY é a própria decomposição do vetor-distância percorrida no tempo de amostragem. Em (3.1) é mostrado o cálculo para decomposição do vetor-distância, num instante k , considerando θ em *graus*.

$$\begin{aligned} distX(k) &= (0.5 \times T_s \times (v(k) + v(k-1))) \times \cos(\theta) + distX(k-1); \\ distY(k) &= (0.5 \times T_s \times (v(k) + v(k-1))) \times \sin(\theta) + distY(k-1); \end{aligned} \quad (3.1)$$

Os vetores dos perfis de reta e de curva utilizados na simulação foram calculados conforme (2.20) e (2.21). O trajeto projetado faz o robô acelerar uma célula à frente, realiza uma curva para a direita e logo em seguida realiza uma curva para a esquerda e finaliza o trajeto desacelerando o robô seguindo em frente. A Figura 56 mostra os perfis de curva e de velocidade projetados para a simulação.

Figura 56 – Perfis de velocidades projetados



Fonte: autor (2017)

Dentro dos blocos que modelam os perfis de curva, foram postos os vetores de amplitude e de tempo, cujos valores estão na Tabela 12.

Tabela 12 – Vetores utilizados para a simulação

vetor	índice							
	1	2	3	4	5	6	7	8
v	0	30	30	30	30	30	30	30
w	0	0	0	0	358	358	0	0
tempo	0	0.333	0.7667	0.8667	0.9295	1.118	1.180	1.2808
vetor	índice							
	9	10	11	12	13	14	15	
v	30	30	30	30	30	30	0	
w	0	-358	-358	0	0	0	0	
tempo	1.3808	1.4437	1.6322	1.695	1.795	2.095	2.696	

Fonte: autor (2017)

O resultado da simulação ocorreu conforme o esperado. A trajetória desenhada perfeita das curvas e retas, como mostra a Figura 57, prova que o sistema de controle e os perfis de velocidade trabalham satisfatoriamente.

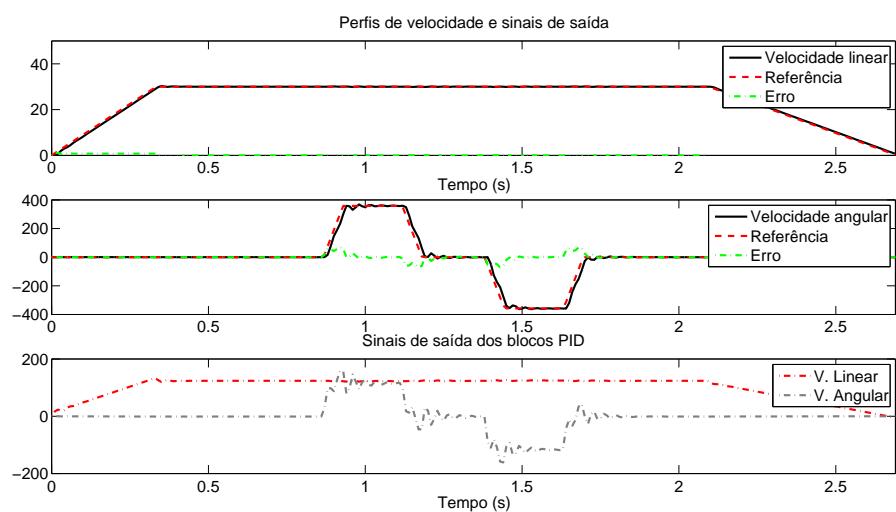
Figura 57 – Simulação da trajetória do robô utilizando os perfis de velocidade como referência para o controlador



Fonte: autor (2017)

Então, o robô pôde ser automatizado por completo utilizando o algoritmo *Flood Fill* e os perfis de curva e de reta projetados para os quatro estados do algoritmo.

Figura 58 – Perfis de velocidade e sinais de saída do controlador proposto



Fonte: autor (2017)

O erro de rastreamento, como pode ser observado na Figura 58, é praticamente

nulo. Como as entradas são rampas, os *overshoots* são praticamente eliminados. Os sinais de controle gerados pelos controladores PID sintonizados pelo método sequencial do relé também é mostrado. Mesmo com a variação da referência de velocidade angular, a velocidade linear praticamente não foi afetada, mostrando a estabilidade da malha de velocidade linear.

3.3.2 Aplicando o controle no mini-robô Micromouse

A discretização de um controlador PID analógico (2.3) pode ser feita de diversas maneiras. Um dos métodos de discretização é a já conhecida aplicação da transformada Z à função analógica discretizada. Para o Micromouse em questão, foi utilizada a discretização por meio da integração trapezoidal. O PID discretizado está em (3.2). Os parâmetros PID utilizados foram os da Tabela 11.

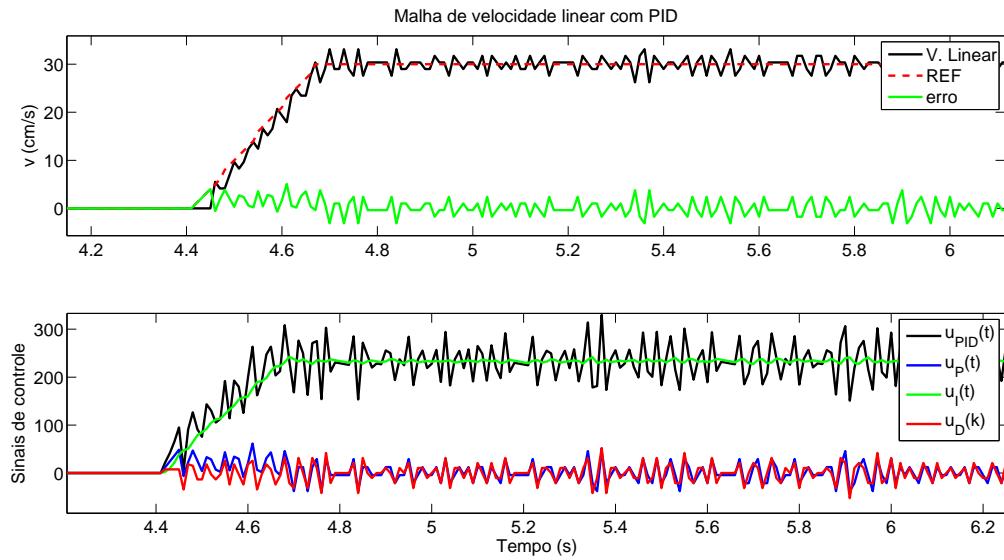
$$\begin{aligned} e(t) &= REF - y(t) \\ u_P(t) &= K_P \times e(t) \\ u_I(t) &= K_I \times T_s \times 0.5 \times (e(t) + e(t-1)) + u_I(t-1) \\ u_D(t) &= K_D \times (e(t) - e(t-1))/T_s \\ u_{PID}(t) &= u_P(t) + u_I(t) + u_D(t) \end{aligned} \quad (3.2)$$

Porém, como o Micromouse carece de RAM, utilizou-se somente uma variável para cada termo do PID ao invés de um *array*. Desta forma, os dois controladores PID foram implementados. O trecho do controle MIMO implementado mostra a equação dos dois controladores de forma otimizada:

```
e_v = REF_v - v;
u_P = K_P * e_v;
u_I = K_I * T_s * 0.5 * (e_v + e_v_ant) + u_I;
u_v = u_P + u_I + u_D;
e_w = REF_w - w;
u_Pw = K_Pw * e_w;
u_Iw = K_Iw * T_s * 0.5 * (e_w + e_w_ant) + u_Iw;
u_Dw = K_Dw * (e_w - e_w_ant)/T_s;
u_w = u_Pw + u_Iw + u_Dw;
e_v_ant = e_v;
e_w_ant = e_w;
```

O resultado da implementação do controlador PID na malha de velocidade linear está na Figura 59.

Figura 59 – Resultado do controle da malha de velocidade linear implementado no Micromouse



Fonte: autor (2017)

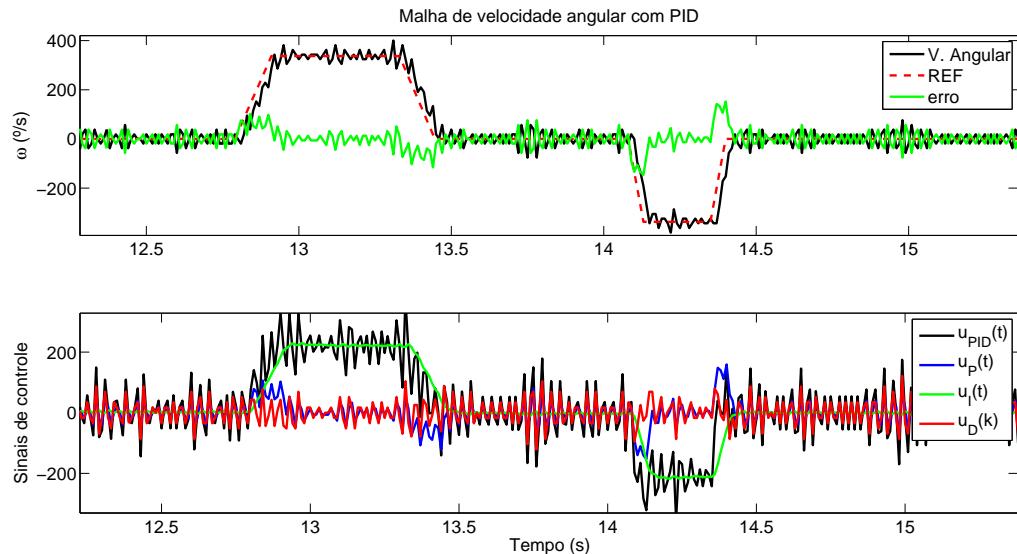
Os ruídos comuns em sistemas reais pertubam o sinal de saída, e os controladores *proporcional* e *derivativo* potencializam, conforme mostra a figura, porém a média do sinal segue a referência fortemente. As entradas do tipo rampa se comportaram perfeitamente, com pouco erro de regime permanente. A curva do erro é praticamente zero. Quase não há interferência da malha de velocidade angular, mesmo com entradas em rampa. Não existiu *Overshoot* característico, como previsto na simulação, para entradas em rampa.

Já o resultado da implementação do controlador PID na malha de velocidade angular pode ser observado na Figura 60. As oscilações comuns devido ao ruído, potencializados pelos controladores proporcional e derivativo, como pode ser observado o nível do sinal dos mesmos na figura, acabam deixando o sinal sujo, porém, a média do sinal segue firme a referência. O erro é baixo para a malha de velocidade linear, com erro máximo absoluto de 5 cm/s, porém, para velocidade angular, o erro máximo absoluto atingiu o valor de 100 °/s, sendo ω_{max} igual a 358 °/s. Portanto, a sintonia de processos MIMO pelo Método do relé sequencial logrou êxito, para velocidades baixas. Para maiores velocidades tangenciais, o erro absoluto torna-se maior para a velocidade angular, isto porque, para mesmo raio de curvatura, a velocidade angular máxima ω_{max} aumenta proporcionalmente, forçando uma aceleração mais abrupta.

O robô poderá utilizar os parâmetros sintonizados, porém, para diminuir as oscilações devido ao ruído iminente do sistema motor-rodas, um *ajuste fino* foi feito

no ganho derivativo dos controlador da malha de velocidade angular, e seu valor foi diminuído para 70% do valor original, o que melhorou um pouco a sua saída.

Figura 60 – Resultado do controle da malha de velocidade angular implementado no Micromouse



Fonte: autor (2017)

Portanto, a sintonia via método do relé é simples e funcional. À frente, é mostrado resultado da união das partes implementadas num labirinto-teste.

3.4 Sistema completo

Todas as partes de *hardware* e de *software* são importantes para um bom desempenho do Micromouse. Através do *ECLIPSE IDE*, foi gravado o programa *Flood Fill*, já em linguagem C, em conjunto com o sistema de controle para velocidades, de forma que o robô possa se locomover pelo labirinto de forma autônoma e inteligente. Portanto, os primeiros resultados foram obtidos para labirinto de 8×8 e, posteriormente, para o labirinto completo.

Após as implementações das partes e simulações, foram feitas as junções das mesmas. O programa do algoritmo *Flood Fill*, criado já na linguagem padrão de gravação do microcontrolador, foi repassado para o programa *ECLIPSE IDE* quase sem modificação. A parte do algoritmo que atualizava as paredes para modelagem do labirinto e emulava os sensores das paredes foi retirado.

Primeiro, realizou-se o teste do programa *Flood Fill* no Micromouse a fim de verificar o funcionamento do algoritmo embarcado no microcontrolador. O robô não

se movimentava fisicamente. O passo era garantido com o aperto do botão. A cada aperto, o robô dava um passo e imprimia o labirinto visto pelo robô, via *bluetooth*, para um terminal de porta serial.

Também, foi possível avaliar a função de detecção das paredes. Posteriormente à implementação de todas as partes no código do Micromouse, no estado IMPRIMIR_MAZE, a função responsável por exibir o labirinto visto pelo robô foi desativado e a validação dos testes passou a ser feita de forma visual, sobre o labirinto.

Os primeiros testes foram feitos no labirinto com poucas paredes, como é mostrada na Figura 61. O objetivo era testar os perfis de curva e a função para detecção das paredes, isto já com o algoritmo *Flood Fill* para indicar o sentido.

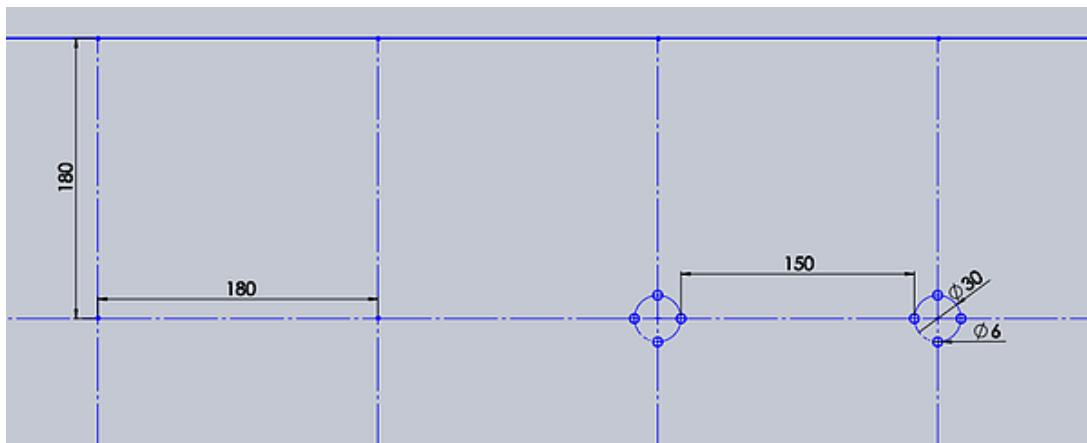
Figura 61 – Foto do primeiro teste do Micromouse sobre o tablado



Fonte: autor (2017)

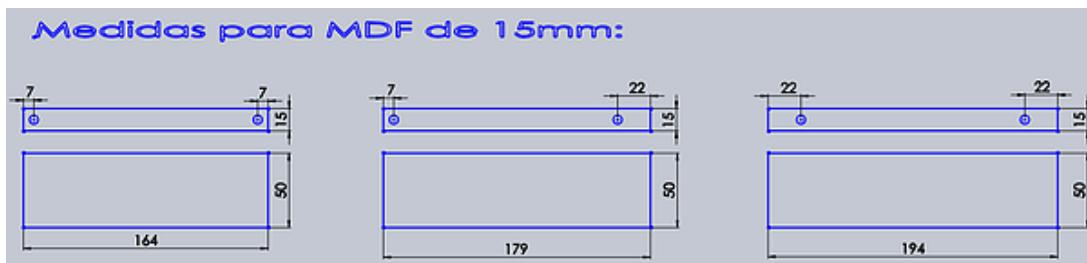
3.4.1 Construção e montagem do labirinto

Para construção das peças e do tablado, foram utilizadas placas de MDF de 15 mm de espessura. Na internet encontram-se fácil esquemas e projetos para montagem de labirinto para testes. Na Figura 62 é apresentada uma maneira simples de construir um labirinto de uma forma que as paredes podem ser trocadas, permitindo diversas configurações para testar o Micromouse.

Figura 62 – Projeto para contrução de labirinto

Fonte: adaptado de Silva (2015a)

As células de um labirinto oficial medem $180 \times 180\text{ mm}$ (distância entre os centros das paredes). Neste projeto, em cada canto de uma célula (exceto nos limites do labirinto), tem um conjunto de quatro furos de 6 mm de diâmetro circunscritos em um círculo com 30 mm de diâmetro, conforme mostra a Figura 62.

Figura 63 – Medidas de corte para as peças

Fonte: adaptado de Silva (2015a)

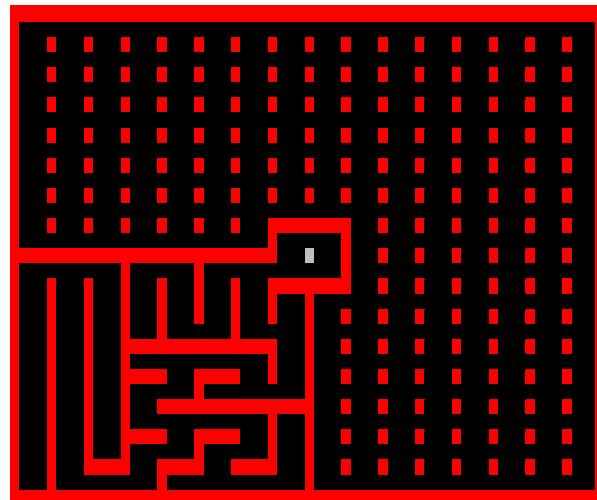
Neste projeto, são três peças diferentes, conforme pode se observar na Figura 63. Isto possibilita montar qualquer configuração de labirinto. Os furos nestas peças servem para colar cavilhas de madeira (acessório para montagem de móveis) que serão encaixadas nos furos do piso. As paredes do labirinto foram feitas de MDF com as duas faces pintadas de branco, e o piso, pintado de preto fosco.

3.4.2 Testes com labirinto de 8×8 células

O algoritmo é capaz de descobrir o melhor caminho em labirintos de 8×8 células, desde que os seus limites estejam fechadas, exceto a célula-alvo. Através do EXCEL,

foi modelado um labirinto para testes e seu desenho pode ser visto na Figura 64.

Figura 64 – Labirinto-teste de 8x8 células desenhado no EXCEL



Fonte: autor (2017)

As distâncias das células para o melhor caminho podem ser vistas na Figura 65, onde o percurso otimizado está grifado.

Figura 65 – Labirinto-teste de 8×8 simulado

Fonte: autor (2017)

Então, com as peças disponíveis para montagem, o labirinto-teste foi montado, conforme mostra a Figura 66.

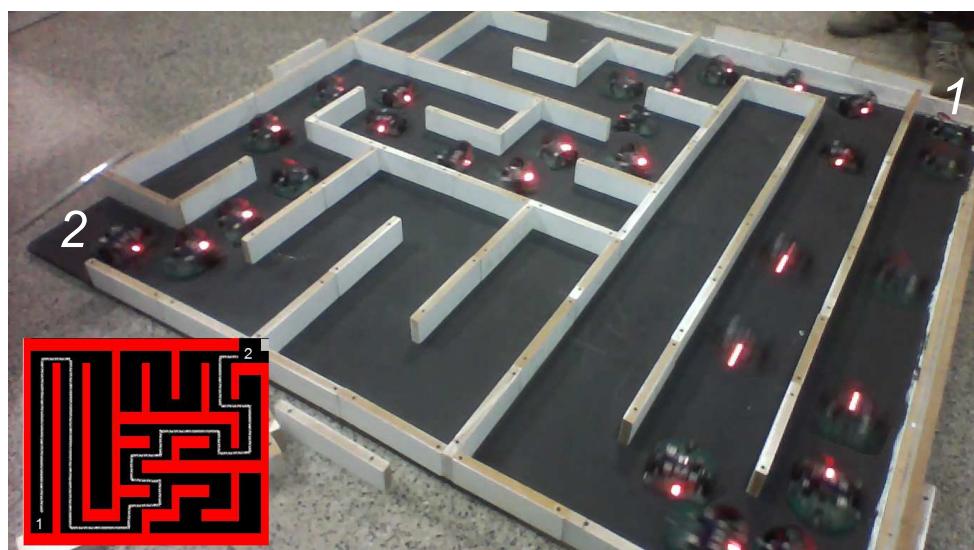
Figura 66 – Labirinto montado para testes de 8×8 células



Fonte: autor (2017)

O sucesso da união das partes propostas construídas e implementadas no robô foi provado pelo êxito do robô em realizar diversas corridas e definir sempre o melhor caminho. A Figura 67 mostra o percurso otimizado do robô para o labirinto-teste montado.

Figura 67 – Trajetória final do robô sobre o labirinto de 8×8 células



Fonte: autor (2017)

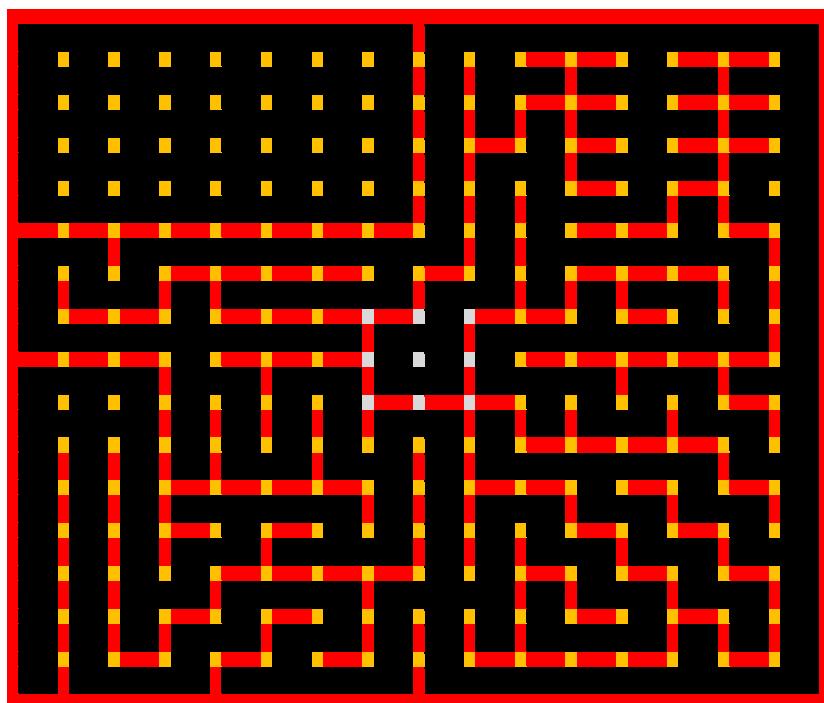
O robô conseguiu resolver o labirinto-teste corretamente e seguiu o mesmo percurso da simulação, tanto na ida como na volta, com velocidade de testes de 30 cm/s em curvas, e 60 cm/s para retas as quais o robô já visitou.

A correção do erro de posição é feita de forma satisfatória, mantendo o robô paralelamente às paredes, quando elas existem. Os perfis de velocidade, implementados para controlar o movimento, funcionaram como o planejado. O robô realiza as curvas de forma fechada, e lê as informações das paredes antes do início da próxima célula. As acelerações e desacelerações ocorreram de forma satisfatória.

3.4.3 Testes com labirinto de 16 x 16 células

Por fim, o conjunto implementado foi testado e validado num labirinto de 16x16 células. Foram repetidos os procedimentos feitos para o teste do labirinto de 8x8 células.

Figura 68 – Labirinto-teste de 16x16 células desenhado no EXCEL



Fonte: autor (2017)

Novamente, através do *EXCEL*, um labirinto foi desenhado de tal forma que as paredes não sejam totalmente conexas. Desta forma, somente os algoritmos inteligentes baseados na Teoria dos Grafos têm capacidade de resolvê-lo. Também, para validação do robô em *malha aberta*, algumas paredes foram retiradas. A Figura 68 mostra o *design* do labirinto proposto.

Para o labirinto-teste de Figura 68, foram realizadas estatísticas para comparação de desempenho dos algoritmos. A Tabela 13 mostra em detalhes.

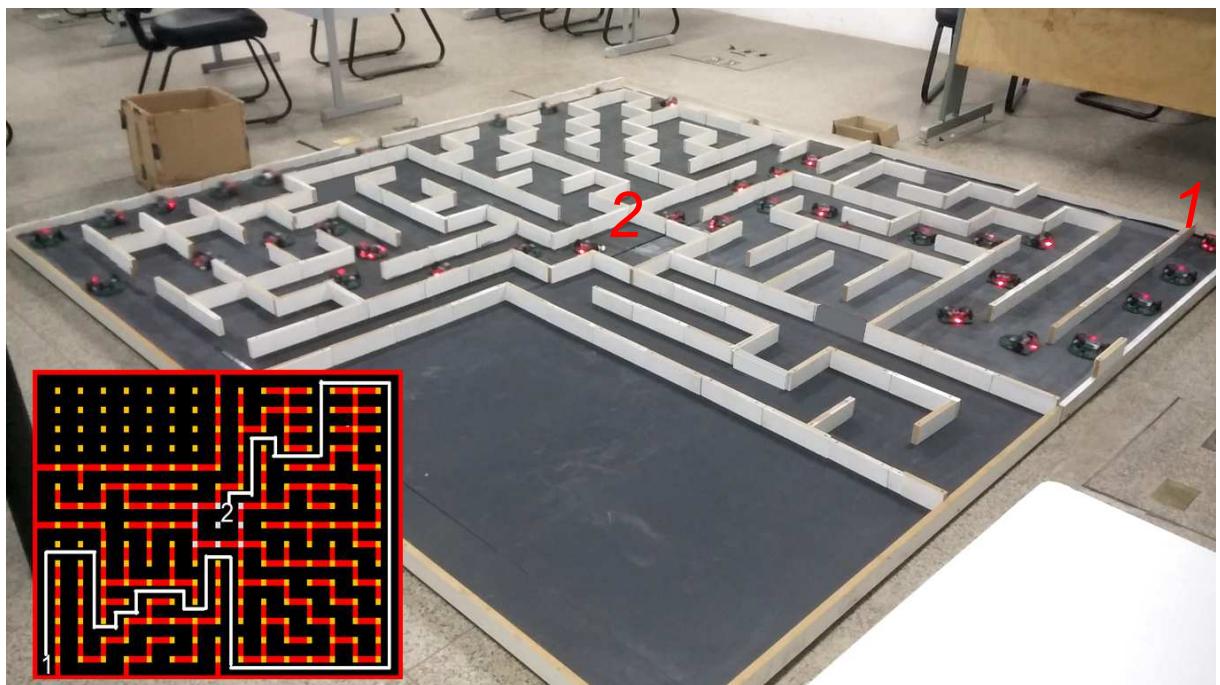
Tabela 13 – Desempenho dos algoritmos no labirinto proposto

Estatísticas	Algoritmo proposto	Micro Mouse
Únicas células atrav.	176	177
Cél. p/ chegar ao centro (1 ^a vez)	149	150
Curvas (na primeira corrida)	72	72
Células visit. na melhor corrida	68	68
Curvas (melhor caminho)	22	22
Cél. atrav. p/ comp. a melhor corrida	441	578
Curvas (p/ completar a melhor corrida)	192	228

Fonte: autor (2017)

O percurso otimizado do Micromouse, para solucionar o labirinto montado, pode ser observado através da Figura 69.

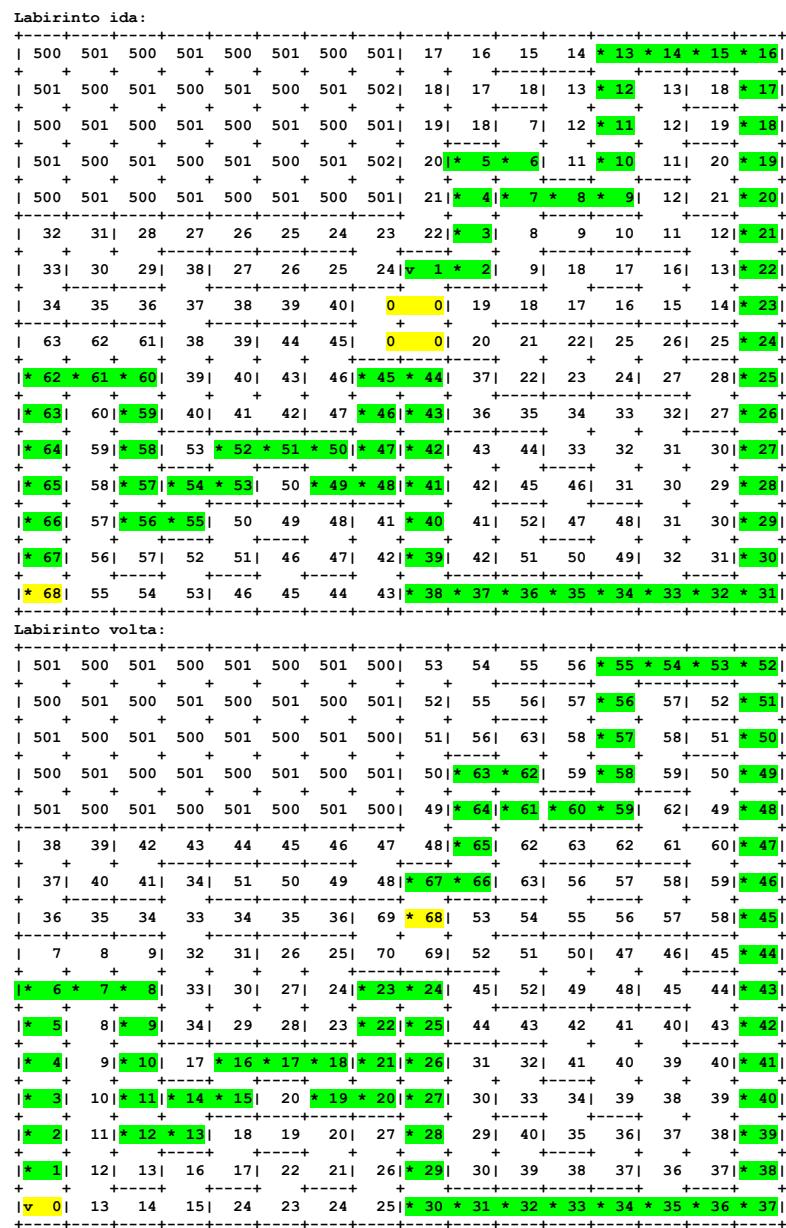
Figura 69 – Trajetória final do robô sobre o labirinto-teste real de 16x16 células



Fonte: autor (2017)

Utilizando o *FALCON C++ IDE* para simulação do percurso do robô, os percursos otimizados para o caminho de ida e de volta foram os mesmos do percursos formados pelo robô no labirinto-teste. Os resultados da simulação podem ser vistos na Figura 70. O menor caminho está marcado com asterisco.

Figura 70 – Simulação do percurso do robô no labirinto-teste proposto de 16x16 células



Fonte: autor (2017)

Portanto, no labirinto-teste, o algoritmo proposto já apresenta desempenho superior ao do *Micro Mouse Simulator*. O número de células atravessadas e o número de curvas, para completar a melhor corrida, são inferiores. Isto demonstra que o algoritmo pode apresentar desempenho até superior, a depender do labirinto. E a quantidade de células atravessadas, no percurso de melhor corrida, sempre será igual em ambos os algoritmos implementados.

Para testes em velocidades mais altas, as curvas não são realizadas conforme o esperado. As referências, para tais velocidades, não são seguidas corretamente, visto que, para maiores velocidades tangenciais, são requeridas, também, maiores velocidades angulares para mesmo raio de curva, e, portanto, há um erro em entradas do tipo rampa. O controle clássico implementado somente é adequado para velocidades baixas.

4 Considerações Finais

4.1 Conclusão

A robótica, e em particular, a robótica móvel, teve um salto de desenvolvimento muito grande nestas duas últimas décadas com aplicações em diferentes áreas. O desenvolvimento da inteligência artificial, ao longo do tempo, proporcionou a autonomia de vários processos, de forma a interagir com os sinais do ambiente. Atualmente eles estão sendo desenvolvidos para trabalhar na medicina, na agricultura, na indústria, para as residências, e até mesmo para diversas competições de robótica existentes em todo o mundo, em especial para competições Micromouse. Os robôs Micromouse analisam o meio e tomam decisões de forma inteligente para solucionar labirintos.

A competição Micromouse ganhou o mundo a partir da década de 1970. As regras da competição foram fixadas pelo IEEE. O desafio consiste em resolver um labirinto de 16x16 células, disponibilizado somente na hora da competição. O robô deverá sair de um dos cantos e chegar ao centro no menor tempo possível. O ganhador é o Micromouse que possui o menor tempo de corrida. Os robôs geralmente possuem um microcontrolador, motores com *encoders*, sensores de distância montados sobre PCI.

Alguns algoritmos foram estudados. Os mais comuns, para resolver o labirinto, são os seguintes: Seguidor de paredes, *Treumax* e *Flood Fill*. O seguidor de parede não consegue resolver labirintos com paredes não conexas ao centro, e, desta forma, há muito tempo, eles deixaram de ser utilizados. Os algoritmos *Treumax* e *Flood Fill*, dois algoritmos surgidos a partir da *Teoria dos Grafos*, foram desenvolvidos para resolver este problema. Segundo estudo, os dois algoritmos tem desempenhos satisfatórios, requerem bom processamento e memória, porém, a conclusão é que o algoritmo *Flood Fill* resolve qualquer labirinto sempre com o menor número de células atravessadas, visto que ele sempre tenta seguir um percurso otimista quando não há informações suficientes do labirinto.

Este trabalho desenvolveu e implementou, em robô Micromouse, um novo algoritmo, baseado no *Flood Fill*, que utilize a menor quantidade possível de memória RAM, mantendo a eficiência do mesmo quando comparado aos algoritmos implementados na literatura. A estratégia otimiza o esquema de memória para armazenar, em somente uma estrutura de dados, os caminhos de ida e de volta em uma corrida típica do Micromouse. Além do projeto do algoritmo, um bom projeto de *hardware* do robô e de controle de velocidades dos motores foram necessários, e o conjunto deve andar

em sintonia perfeita durante as corridas num labirinto desconhecido.

A eficiência do algoritmo proposto foi comparada à eficiência do algoritmo *Flood Fill* do simulador *Micro Mouse Maze Editor and Simulator* e demonstrou-se que os algoritmos apresentaram eficiência equivalentes, sendo que a estratégia de otimização de memória proposta deixa em vantagem o algoritmo 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. Algumas estatísticas demonstraram desempenho superior do algoritmo proposto, em relação ao número de células atravessadas para encontrar o melhor caminho.

Entretanto, para sistemas MIMO, uma extensão do método do relé, denominado *método do relé sequencial*, conseguiu oscilar as malhas em seguência. A sintonia dos controladores também é baseada na tabela do Ziegler-Nichols. Portanto, este método, utilizado para sintonia dos controladores PID do sistema MIMO proposto, teve êxito, tanto na simulação como também na implementação do controle no robô.

As saídas do sistema MIMO seguem perfeitamente os *perfis de velocidade*, que, quando gerados, dão ao robô as referências necessárias para a realização da trajetória. Quando isto ocorre, a probabilidade do robô seguir a trajetória projetada é grande. São mínimos os esforços do controle de correção do posicionamento do robô através do *feedback* dos sensores de distância. As saídas se comportaram bem para as entradas em rampa, com erro em regime permanente quase que imperceptível, para velocidades baixas.

Num sistema robótico real, os ruídos fazem parte do sistema. Eles podem gerar uma perturbação no sistema de controle, e, com isso, acúmulo de erros de posicionamento são inevitáveis, num sistema em *malha aberta*. Uma realimentação baseada nos sensores de distância IR tornou imprescindível para o Micromouse se locomover com erros mínimos de posicionamento. Os sensores trabalharam corretamente a fornecer *feedback* para correção dos erros de posicionamento do Micromouse, sendo que quanto maior o erro, maior é a sua ação sobre a velocidade angular, responsável

por ajustar o ângulo do robô e manter o mesmo no centro do labirinto.

O sucesso da união de todas as partes construídas pôde ser visto durante os testes em labirinto real. O algoritmo *Flood Fill* indicou corretamente o melhor caminho. As máquinas de estado responsáveis pelas gerações dos perfis de velocidade garantiram as referências das trajetórias de curvas e de retas. Os obstáculos foram detectados e armazenados corretamente nas estruturas.

4.2 Trabalhos Futuros

Para trabalhos futuros, sugerem-se os itens a seguir:

- a) utilizar redes neurais artificiais ou sistemas nebulosos para controlar os perfis de reta e de curva;
- b) implementar o algoritmo *Flood Fill diagonal*;
- c) implementar o algoritmo de controle robusto PID preditivo, adaptativo e inteligente;
- d) replicar o micromouse para incentivar competições locais e estaduais da categoria.

Referências

- ARAUJO, R. de B.; JUNIOR, J. d. N. A. S. Modelo matemático discreto de uma planta térmica pelo método das diferenças (backward difference) e uma estimativa recursiva de mínimos quadrados dos seus parâmetros. 2012. Citado na página 38.
- BARÇANTE, G. M. *Controle PID multivariável descentralizado: sintonia e aplicação prática*. Tese (Doutorado), 2011. Citado 3 vezes nas páginas 44, 47 e 50.
- BORGES, D. J. D. *Desenvolvimento de um robo para o concurso Micromouse*. Tese (Doutorado) — TRAS-OS-MONTES E ALTO DOURADO, 2013., 2013. Citado 6 vezes nas páginas 1, 6, 8, 9, 25 e 58.
- CAI, J. et al. An algorithm of micromouse maze solving. In: *2010 10th IEEE International Conference on Computer and Information Technology*. [S.l.: s.n.], 2010. p. 1995–2000. Citado 2 vezes nas páginas 9 e 25.
- CAI, Z.; YE, L.; YANG, A. Floodfill maze solving with expected toll of penetrating unknown walls for micromouse. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*. [S.l.: s.n.], 2012. p. 1428–1433. Citado 4 vezes nas páginas 9, 10, 12 e 25.
- CAICEDO, D. A. R.; TRUJILLO, L. E. T. *Diseño e Implementación de dos Robots de Competencia (Seguidor de Línea Especialidad Velocista, Laberinto)*. Tese (Doutorado) — Quito, 2016., 2016. Citado na página 6.
- CHRISTIANSEN, D. The amazing micromouse roars on [spectral lines]. *IEEE Spectrum*, v. 51, n. 5, p. 8–8, May 2014. ISSN 0018-9235. Citado 4 vezes nas páginas 3, 4, 5 e 17.
- COELHO, A. A. R.; COELHO, L. dos S. *Identificação de sistemas dinâmicos lineares*. [S.I.]: UFSC, 2015. Citado 7 vezes nas páginas 38, 40, 42, 44, 48, 49 e 70.
- COELHO, C. A. de S. Auto-ajuste de controladores pid usando o método da linearização harmônica. 2004. Citado na página 40.
- DAI, S. et al. Design and practice from the micromouse competition to the undergraduate curriculum. In: *2015 IEEE Frontiers in Education Conference (FIE)*. [S.I.: s.n.], 2015. p. 1–5. Citado 2 vezes nas páginas 5 e 17.
- FRANKLIN, G. F.; POWELL, J. D.; EMANI-NAEINI, A. *Sistemas de Controle para Engenharia*. [S.I.]: Bookman, 2013. Citado na página 39.
- KATSUHIKO, O. Engenharia de controle moderno. *KATSUHIKO Ogata, 5th Ed. 801p*, 2011. Citado na página 47.
- KIBLER, S. G. et al. Ieee micromouse for mechatronics research and education. In: *2011 IEEE International Conference on Mechatronics*. [S.I.: s.n.], 2011. p. 887–892. Citado na página 17.

- LANDAU, I. D.; ZITO, G. *Digital Control Systems*. [S.I.]: Springer, 2006. Citado 2 vezes nas páginas 40 e 41.
- LI, X. et al. An improved algorithm of the exploring process in micromouse competition. In: *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*. [S.I.: s.n.], 2010. v. 2, p. 324–328. Citado 3 vezes nas páginas 1, 2 e 5.
- LIN, G. Design and implementation of a micromouse [d]. Taiwan: Lunghwa University of Science and Technology, 2010. Citado na página 17.
- LJUNG, L.; SÖDERSTRÖM. *Theory and Practice of Recursive Identification*. [S.I.]: MIT Press, 1983. Citado na página 38.
- LOPEZ, G. et al. Micromouse: An autonomous robot vehicle interdisciplinary attraction to education and research. In: *2015 IEEE Frontiers in Education Conference (FIE)*. [S.I.: s.n.], 2015. Citado 2 vezes nas páginas 1 e 17.
- LOPEZ, G. et al. Micromouse: An autonomous robot vehicle interdisciplinary attraction to education and research. In: *2015 IEEE Frontiers in Education Conference (FIE)*. [S.I.: s.n.], 2015. p. 1–5. Citado na página 1.
- MISHRA, S.; BANDE, P. Maze solving algorithms for micro mouse. In: . [S.I.: s.n.], 2008. p. 86–93. Citado 2 vezes nas páginas 8 e 25.
- SADIK, A. M. J. et al. A comprehensive and comparative study of maze-solving techniques by implementing graph theory. In: *2010 International Conference on Artificial Intelligence and Computational Intelligence*. [S.I.: s.n.], 2010. v. 1, p. 52–56. Citado 3 vezes nas páginas 7, 9 e 25.
- SHANNON, C. E. *Biography*. [S.I.], 2016. Disponível em: <<http://spectrum.ieee.org/computing/software/clause-shannon-tinkerer-prankster-and-father-of-information-theory>>. Citado na página 1.
- SHARMA, M. Algorithms for micro-mouse. p. 581–585, 2009. Citado na página 25.
- SILVA, K. L. da. *Construa seu próprio labirinto para testes*. [S.I.], 2015. Disponível em: <<https://kleberufu.wixsite.com/micromousebrasil/single-post/2015/08/04/Construa-seu-próprio-maze-para-testes>>. Citado na página 79.
- SILVA, K. L. da. *Manual do umart Lite Plus*. [S.I.], 2015. Disponível em: <<http://kleberufu.wixsite.com/micromousebrasil>>. Citado 2 vezes nas páginas 20 e 57.
- SILVA, S. et al. A iniciativa micromouse e o estímulo ao saber tecnológico no ensino pré-universitário. p. 59–67, 2015. Citado 3 vezes nas páginas 1, 9 e 25.
- SNAPP, R. R. *Threading mazes*. [S.I.], 2010. Disponível em: <<http://www.cems.uvm.edu/~rsnapp/teaching/cs32/lectures/tremaux.pdf>>. Citado na página 9.
- SOLVER. *maze-solve*. [S.I.], 2013. Disponível em: <<http://code.google.com/p/maze-solver/>>. Citado na página 7.

- SU, J. H.; HUANG, H. H.; LEE, C. S. Behavior model simulations of micromouse and its application in intelligent mobile robot education. In: *2013 CACS International Automatic Control Conference (CACS)*. [S.I.: s.n.], 2013. p. 511–515. Citado 9 vezes nas páginas 1, 4, 10, 11, 12, 13, 25, 50 e 68.
- SU, J. H.; HUANG, H. H.; LEE, C. S. A simple and efficient diagonal maze-solver for micromouse contests and intelligent mobile robot education. In: *The 26th Chinese Control and Decision Conference (2014 CCDC)*. [S.I.: s.n.], 2014. p. 2407–2411. ISSN 1948-9439. Citado na página 17.
- TAMURA, K.; OHMORI, H. Auto-tuning method of expanded pid control for mimo systems. In: *2006 SICE-ICASE International Joint Conference*. [S.I.: s.n.], 2006. p. 3270–3275. Citado na página 46.
- TORRES, M. J. N.; RAMIREZ, H. R. Diseño y construcción de micromouse de alto desempeño. 2016. Citado na página 6.
- YADAV, S.; VERMA, K. K.; MAHANTA, S. The maze problem solved by micro mouse. In: *2012 IEEE Frontiers in Education Conference (FIE)*. [S.I.: s.n.], 2012. Citado 5 vezes nas páginas 8, 9, 12, 17 e 25.
- YADAV, S.; VERMA, K. K.; MAHANTA, S. The maze problem solved by micro mouse. *International Journal of Engineering and Advanced Technology (IJEAT)*, v. 1, n. 4, p. 157–162, April 2014. ISSN 2249 – 8958. Citado 2 vezes nas páginas 6 e 8.