

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

MARCELO GERVAZONI CARBONERA

**NAVEGAÇÃO EM ROBÔS MÓVEIS POR ARBITRAGEM E FUSÃO  
EM ARQUITETURAS COMPORTAMENTAIS**

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO  
2021

**MARCELO GERVAZONI CARBONERA**

**NAVEGAÇÃO EM ROBÔS MÓVEIS POR ARBITRAGEM E FUSÃO  
EM ARQUITETURAS COMPORTAMENTAIS**

Trabalho de Conclusão de Curso apresentado na disciplina de TCC1 como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientadora: Dra. Kathya Silvia Collazos Linares

**PATO BRANCO**

**2021**

Dedico este trabalho a meus familiares, pelo apoio.

Aos amigos que fiz nessa trajetória, por compartilhar todas as lutas e pelas conversas capazes de fortalecer o espírito.

E por fim, dedico também àqueles que não estavam esperando grande coisa.

## **AGRADECIMENTOS**

Agradeço à professora Kathya, minha orientadora, pela paciência.

Aos familiares, por me aguentar.

A Isaac Asimov, pelos livros que me motivaram a concluir a graduação.

E por último e não menos importante, agradeço a mim, pela teimosia de não mudar de curso ou de universidade.

*Robots of the world! The power of man has fallen! A new world has arisen: the Rule of the Robots! March! (Radius, Rossum Universal Robots, ato III)*

*He move in space with minimum waste and maximum joy (Sade, Smooth Operator)*

## **RESUMO**

CARBONERA, Marcelo G. NAVEGAÇÃO EM ROBÔS MÓVEIS POR ARBITRAGEM E FUSÃO EM ARQUITETURAS COMPORTAMENTAIS. 88 f. Trabalho de Conclusão de Curso – Departamento acadêmico de informática, Universidade Tecnológica Federal do Paraná. Pato Branco, 2021.

O presente trabalho teve por objetivo explorar o uso da arquitetura baseada em comportamento no campo da robótica móvel. Nesse sentido, foram utilizadas as duas estratégias existentes na literatura: arbitragem e fusão de comportamentos. A primeira foi implementada utilizando controle híbrido (sistemas a eventos discretos com controladores de variáveis contínuas), enquanto a segunda utilizou controladores *Fuzzy* para sua execução.

**Palavras-chave:** Robótica móvel, Robô de acionamento diferencial, Modelo Uniciclo, Robótica Baseada em Comportamento

## ABSTRACT

CARBONERA, Marcelo G. NAVIGATION IN MOBILE ROBOTS BY ARBITRATION AND FUSION IN BEHAVIORAL ARCHITECTURES. 88 f. Trabalho de Conclusão de Curso – Departamento acadêmico de informática, Universidade Tecnológica Federal do Paraná. Pato Branco, 2021.

The present work aimed to explore the use of behavior-based architecture in the field of mobile robotics. In this sense, the two existing strategies found in the literature were used: behavior arbitration and fusion. The first was implemented using hybrid control (discrete event systems with continuous variables controllers), while the second used *Fuzzy* controllers for its execution.

**Keywords:** Mobile Robotics, Differential Drive Robot, Unicycle Model, Behavior Based Robotics

## LISTA DE FIGURAS

FIGURA 1	– Arbitragem e fusão de comportamentos .....	16
FIGURA 2	– Comportamentos definidos por campos potenciais .....	17
FIGURA 3	– Modelos uniciclo e diferencial .....	18
FIGURA 4	– Diagrama de Transição de Estado .....	22
FIGURA 5	– SED em malha fechada .....	22
FIGURA 6	– Exemplos de diagramas de transição para autômatos temporizados com guarda .....	23
FIGURA 7	– Controlador Híbrido .....	25
FIGURA 8	– Modo deslizante e diagramas de transição .....	26
FIGURA 9	– Exemplo do modo deslizante em robótica móvel .....	26
FIGURA 10	– Representação gráfica da ação de um controle PID .....	27
FIGURA 11	– Diagrama de blocos para um sistema com controlador <i>Fuzzy</i> .....	28
FIGURA 12	– Conjuntos <i>Fuzzy</i> possíveis para a variável “altura” .....	29
FIGURA 13	– Operações união e intersecção em conjuntos <i>Fuzzy</i> .....	31
FIGURA 14	– Inferência em Lógica <i>Fuzzy</i> .....	32
FIGURA 15	– Defuzzificação COG .....	33
FIGURA 16	– Sinais de saída dos canais dos <i>encoders</i> para sentido de rotação .....	34
FIGURA 17	– Materiais e robô após montagem .....	37
FIGURA 18	– Robô após montagem .....	38
FIGURA 19	– Resposta do sensor infravermelho .....	39
FIGURA 20	– Curva do motor sem carga .....	39
FIGURA 21	– Curva do motor com carga .....	40
FIGURA 22	– Regressão Linear para a curva do motor .....	40
FIGURA 23	– Robôs Khepera 3 e QuickBot .....	41
FIGURA 24	– Robôs Khepera 3 e QuickBot em simulação .....	41
FIGURA 25	– Comportamento Ir para Objetivo .....	45
FIGURA 26	– Comportamento Evitar Obstáculo .....	47
FIGURA 27	– Tipos de obstáculos .....	50
FIGURA 28	– Comportamento Seguir Parede .....	51
FIGURA 29	– Recomendação vetorial .....	52
FIGURA 30	– Autômato Híbrido que soluciona o problema de navegação .....	57
FIGURA 31	– Diagrama do sistema <i>Fuzzy</i> “Evitar Obstáculo” .....	59
FIGURA 32	– Conjuntos <i>Fuzzy</i> para as entradas dos sensores .....	60
FIGURA 33	– Conjuntos <i>Fuzzy</i> para as saídas vetoriais .....	60
FIGURA 34	– Conjuntos <i>Fuzzy</i> para a saída de velocidade .....	61
FIGURA 35	– Curva de Ativação para comportamento Seguir Parede .....	62
FIGURA 36	– Diagrama do sistema <i>Fuzzy</i> “Seguir Parede” .....	62
FIGURA 37	– Conjuntos <i>Fuzzy</i> para as saídas vetoriais .....	65
FIGURA 38	– Diagrama do sistema <i>Fuzzy</i> “Seguir Recomendação” .....	67
FIGURA 39	– Conjuntos <i>Fuzzy</i> para as entradas vetoriais .....	67
FIGURA 40	– Conjuntos <i>Fuzzy</i> para a entrada de velocidade .....	68
FIGURA 41	– Conjuntos <i>Fuzzy</i> para as velocidades angulares de saída .....	68

FIGURA 42 – Esquema lógico do circuito .....	71
FIGURA 43 – Placa projetada .....	74
FIGURA 44 – Resultado em simulação para o controlador híbrido .....	76
FIGURA 45 – Resultado em simulação para o controlador <i>fuzzy</i> .....	77
FIGURA 46 – Resultado do comportamento “Ir Para Objetivo” .....	78
FIGURA 47 – Resultado do comportamento “Evitar Obstáculo” .....	78
FIGURA 48 – Resultado do comportamento “Mesclado” .....	79
FIGURA 49 – Resultado do comportamento “Seguir Parede” .....	80
FIGURA 50 – Resultado implementado para o controlador híbrido .....	81
FIGURA 51 – Resultado implementado para o controlador <i>fuzzy</i> .....	82

## **LISTA DE TABELAS**

TABELA 1	– Tabela de custos .....	36
TABELA 2	– Estados do autômato .....	54
TABELA 3	– Condições utilizadas nas transições do autômato .....	54
TABELA 4	– Regras <i>Fuzzy</i> para sistema “Evitar Obstáculo” .....	61
TABELA 5	– Regras <i>Fuzzy</i> para sistema “Seguir Parede” .....	66
TABELA 6	– Recomendações para velocidades .....	69
TABELA 7	– Comandos disponíveis na interface por linha de comando via <i>bluetooth</i> ...	75

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>12</b>
1.1 CONSIDERAÇÕES INICIAIS .....	12
1.1.1 Aspectos qualitativos em robótica móvel .....	13
1.2 OBJETIVOS .....	14
1.2.1 Objetivo Geral .....	14
1.2.2 Objetivos Específicos .....	14
1.3 MOTIVAÇÃO E JUSTIFICATIVA .....	15
1.4 ESTRUTURA .....	15
<b>2 REFERENCIAL TEÓRICO .....</b>	<b>16</b>
2.1 ARQUITETURA BASEADA EM COMPORTAMENTO .....	16
2.1.1 Comportamentos como campos potenciais .....	16
2.2 MODELAGEM .....	18
2.2.1 Modelo cinemático do uniciclo .....	18
2.2.2 Modelo cinemático do robô móvel de acionamento diferencial .....	19
2.3 CONSIDERAÇÕES PARA O CONTROLE DE ROBÔS MÓVEIS .....	19
2.4 SISTEMAS A EVENTOS DISCRETOS .....	21
2.4.1 Autômatos temporizados com guardas .....	23
2.5 SISTEMAS HÍBRIDOS .....	24
2.6 CONTROLADOR PID .....	26
2.7 LÓGICA FUZZY EM CONTROLE .....	28
2.7.1 Conjuntos Fuzzy .....	29
2.7.2 Componentes de um Sistema <i>Fuzzy</i> .....	30
2.7.2.1 Fuzzificação .....	31
2.7.2.2 Mecanismo de Inferência .....	31
2.7.2.3 Defuzzificação .....	32
2.8 ODOMETRIA .....	33
<b>3 MATERIAIS .....</b>	<b>36</b>
3.1 COMPONENTES FÍSICOS E ELETRÔNICOS .....	36
3.2 MONTAGEM FÍSICA .....	36
3.2.1 Especificações .....	37
3.2.2 Curvas dos motores .....	38
3.3 O SIMULADOR SIMIAM .....	40
3.3.1 Pacote “ui” .....	42
3.3.2 Pacote “simulator” .....	42
3.3.3 Pacote “robot” .....	42
3.3.4 Pacote “controller” .....	43
<b>4 DESENVOLVIMENTO .....</b>	<b>44</b>
4.1 DESENVOLVIMENTO DOS CONTROLADORES .....	44
4.1.1 Controle de sistema híbrido .....	44
4.1.1.1 Comportamento ”Ir Para Objetivo” .....	45
4.1.1.2 Comportamento ”Evitar Obstáculo” .....	47

4.1.1.3 Comportamento mesclado "Ir Para Objetivo e Evitar Obstáculo" .....	49
4.1.1.4 Mínimos locais .....	50
4.1.1.5 Comportamento "Seguir parede" .....	51
4.1.1.6 O autômato para arbitragem de comportamentos .....	53
4.1.2 Controle <i>Fuzzy</i> .....	57
4.1.2.1 Comportamento "Ir Para Objetivo" .....	58
4.1.2.2 Comportamento "Evitar Obstáculo" .....	58
4.1.2.3 Comportamento "Seguir parede" .....	60
4.1.2.4 A estratégia para fusão de comportamentos .....	66
4.1.2.5 Controlador <i>Fuzzy</i> para "Seguir Recomendação" .....	66
4.2 IMPLEMENTAÇÃO DO CIRCUITO E SISTEMA EMBARCADO .....	70
4.2.1 Considerações práticas, limitações do modelo e da simulação .....	70
4.2.2 Circuito Elétrico .....	71
4.2.3 Configuração de portas no microcontrolador .....	72
4.2.4 Projeto da placa .....	73
4.2.5 Esquema lógico do sistema embarcado .....	73
4.3 SIMULAÇÃO .....	75
4.3.1 Utilizando controlador Híbrido .....	75
4.3.2 Utilizando controlador <i>Fuzzy</i> .....	76
4.4 RESULTADO E DISCUSSÃO .....	76
4.4.1 Resultado para controlador híbrido .....	77
4.4.1.1 Comportamento Ir Para Objetivo .....	77
4.4.1.2 Comportamento Evitar Obstáculo .....	77
4.4.1.3 Comportamento Mesclado .....	78
4.4.1.4 Comportamento Seguir Parede .....	79
4.4.1.5 Controlador Final com arbitragem .....	79
4.4.2 Resultado para controlador <i>fuzzy</i> .....	81
<b>5 CONCLUSÕES E TRABALHOS FUTUROS .....</b>	<b>83</b>
<b>REFERÊNCIAS .....</b>	<b>85</b>

## 1 INTRODUÇÃO

Este Capítulo apresenta a motivação e a justificativa para o desenvolvimento do presente trabalho.

### 1.1 CONSIDERAÇÕES INICIAIS

A robótica móvel mescla aspectos das engenharias mecânica, elétrica e eletrônica, além das ciências cognitiva, social e de computação. Os objetivos da robótica móvel são voltados para aplicações diversas, como sensoriamento remoto, exploração ou operação em ambientes inóspitos, carros autônomos, além de relacionamento com humanos, em tarefas assistivas (SIEGWART; NOURBAKHS, 2004).

Na UTFPR câmpus Pato Branco, entre os trabalhos que tratam de robôs móveis, destacam-se:

- a) Bieseck (2016), que desenvolveu um robô para estacionamento automático.
- b) Petry (2016), que desenvolveu um robô seguidor de linha, comparando uma arquitetura de controle híbrida (sistema dinâmico combinado a um sistema a eventos discretos) com o controle por lógica *Fuzzy*.
- c) Lopes (2017), que desenvolveu um robô seguidor de linha, também de arquitetura híbrida.
- d) Marinho (2017), cujo trabalho aprimorou um robô de sumô (robô de competição) previamente disponível.

A execução de tarefas por robôs autônomos envolve raciocínio automatizado, percepção e controle, que incluem problemas como o de planejamento de trajetórias. De uma forma geral, esse problema consiste em encontrar um trajeto que transporte o robô de uma configuração (posição e direção) inicial até uma configuração final. A navegação é, portanto, uma importante área de estudo em robôs autônomos (MARCHI, 2001; NETO, 2007; OTTONI, 2000).

No intuito de aumentar robustez na tarefa de navegação, pode-se usar *lógica fuzzy*, que permite lidar com incertezas do mundo real (YAHMEDI; FATMI, 2011), já que ruídos podem provocar comportamentos erráticos.

A lógica *fuzzy* é um mapeamento não linear de dados de entrada em saídas escalares que permite traduzir conhecimento qualitativo (linguístico) em uma solução numérica, de modo relativamente direto (MENDEL, 1995).

### 1.1.1 ASPECTOS QUALITATIVOS EM ROBÓTICA MÓVEL

Conforme Mataric (2014), controle de robôs móveis pode ser dividido em controle deliberativo, reativo, híbrido (no sentido de combinar os aspectos deliberativo e reativo) e baseado em comportamento.

Na arquitetura deliberativa, deve-se planejar antes de executar qualquer ação. Essa abordagem remonta aos primeiros estudos voltados à aplicação de conceitos da Inteligência Artificial (IA) clássica ao campo da robótica. O foco desta abordagem é atender metas de longo prazo. Porém existe um alto custo computacional, já que é necessário manter uma representação interna do ambiente e operar buscas neste mapa, o que traz dificuldades em alcançar requisitos de curto prazo, dada a possibilidade iminente (imediata) de colisão. O armazenamento de mapa é uma representação do mundo, uma forma de armazenar informações do passado. Podem ser utilizadas matrizes, grafos. Quanto mais detalhe se armazena, maior o custo.

A arquitetura reativa, por sua vez, tem como foco uma curta escala de tempo, que pode ser obtida pelo estabelecimento de um mapeamento direto entre entradas e saídas, sem representação interna do ambiente. Pode ser visto como reflexo ou instinto.

Na arquitetura híbrida, no sentido encontrado no livro de Mataric (2014), o controle deliberativo (“inteligente, porém lento”) e reativo (“rápido, porém inflexível”) são combinados de modo a extrair os pontos fortes de cada abordagem. Há a necessidade de implementar uma camada intermediária entre os dois tipos, a fim de balancear objetivos conflitantes (como escala de tempo). No restante deste texto, o termo “arquitetura híbrida” será usado apenas para designar sistemas que mesclam aspectos de controle contínuo e a eventos discretos.

Por fim, a arquitetura baseada em comportamento surgiu do controle reativo, diferindo deste pelo nível de abstração. Enquanto comportamentos descrevem funções como “seguir objeto”, “encontrar objeto”, “seguir parede”, reações são simples ações como “andar reto” ou “virar”. Há também uma distinção na escala de tempo das ações, já que comportamentos não são instantâneos e operam ao longo do tempo.

O protótipo desenvolvido neste trabalho utiliza a arquitetura comportamental. A utilização de comportamentos pode acontecer por meio de duas estratégias distintas: arbitragem e fusão de comportamentos. Neste trabalho, o protótipo desenvolvido explora essa arquitetura

por completo, utilizando um controlador híbrido (sistemas contínuos com sistemas a eventos discretos) para a arbitragem de comportamentos e controladores *fuzzy* para implementar fusão de comportamentos.

Sobre as arquiteturas reativas e comportamentais, é importante citar a arquitetura de subsunção, que organiza comportamentos em camadas, de modo a estabelecer uma hierarquia na qual módulos de maior importância podem inibir módulos de baixa importância. Para exemplificar, um comportamento responsável por parar o robô deve inibir módulos responsáveis por seu movimento (MATARIC, 2014).

## 1.2 OBJETIVOS

O presente trabalho permitiu a assimilação de aspectos relativos ao campo da robótica móvel. A seguir, pretende-se delimitar quais objetivos devem ser atendidos.

### 1.2.1 OBJETIVO GERAL

Construir um robô móvel de acionamento diferencial autônomo, sem armazenamento de mapa, capaz de se locomover entre duas referências, desviando de obstáculos, se necessário.

### 1.2.2 OBJETIVOS ESPECÍFICOS

- Levantar o modelo matemático para o robô móvel de acionamento diferencial.
- Desenvolver o projeto de controle por arbitragem de comportamentos usando controlador híbrido, junção entre Sistemas Dinâmicos de Variável Contínua (SDVC) e Sistemas a Eventos Discretos (SED).
- Desenvolver o projeto de controle por fusão de comportamentos usando Lógica *Fuzzy*.
- Construção física do robô.
- Implementar os controladores no sistema embarcado.
- Realizar testes de desempenho.
- Comparar qualitativamente as abordagens de arbitragem e fusão de comportamentos. A comparação, portanto, não será em nível de tecnologia (controlador híbrido e controlador *fuzzy*), mas em nível de arquitetura, o que faz mais sentido, já que o intuito deste trabalho é explorar o campo da robótica baseada em comportamentos.

### 1.3 MOTIVAÇÃO E JUSTIFICATIVA

Com os objetivos estipulados, exploram-se no campo da robótica móvel dois aspectos cruciais: navegação e inteligência.

A utilização de lógica fuzzy tem por intuito simplificar o processo de projeto, já que a introdução de variáveis linguísticas permite resolver o problema em questão utilizando predominantemente conhecimentos qualitativos. Deseja-se investigar tal abordagem de forma crítica, verificando qualidades e deficiências frente à contraparte híbrida, que utiliza controle baseado em comportamento associado a sistemas a eventos discretos.

### 1.4 ESTRUTURA

Este trabalho está organizado em cinco capítulos: Introdução, Referencial Teórico, Materiais e Método e Desenvolvimento.

As etapas do trabalho referentes a teoria de controle e lógica *fuzzy* serão discutidas no Capítulo 2 e desenvolvidas para o robô móvel no Capítulo 4.

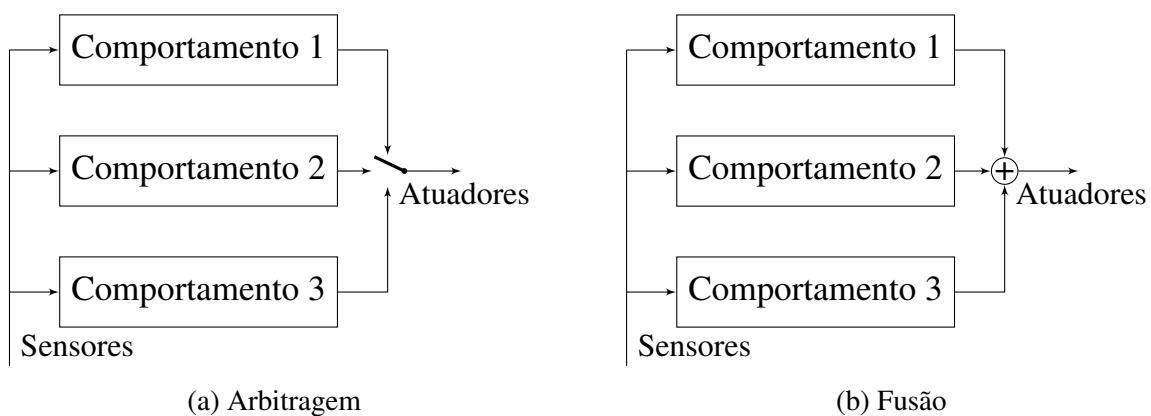
## 2 REFERENCIAL TEÓRICO

Este Capítulo busca explanar os principais conceitos relacionados ao tema.

### 2.1 ARQUITETURA BASEADA EM COMPORTAMENTO

Um dos desafios que surgem em arquiteturas comportamentais é justamente a seleção de ações tendo como base um conjunto de comportamentos (MATARIC, 2014). Duas maneiras tradicionais de fazer isso são: arbitragem e fusão. Na arbitragem, seleciona-se apenas um dos comportamentos por vez, enquanto que na fusão, os comportamentos operam em paralelo e suas saídas são combinadas. Um esquema dos dois modelos pode ser visto na Figura 1.

**Figura 1 – Arbitragem e fusão de comportamentos**



**Fonte:** baseado em Mataric (2014), p. 167.

#### 2.1.1 COMPORTAMENTOS COMO CAMPOS POTENCIAIS

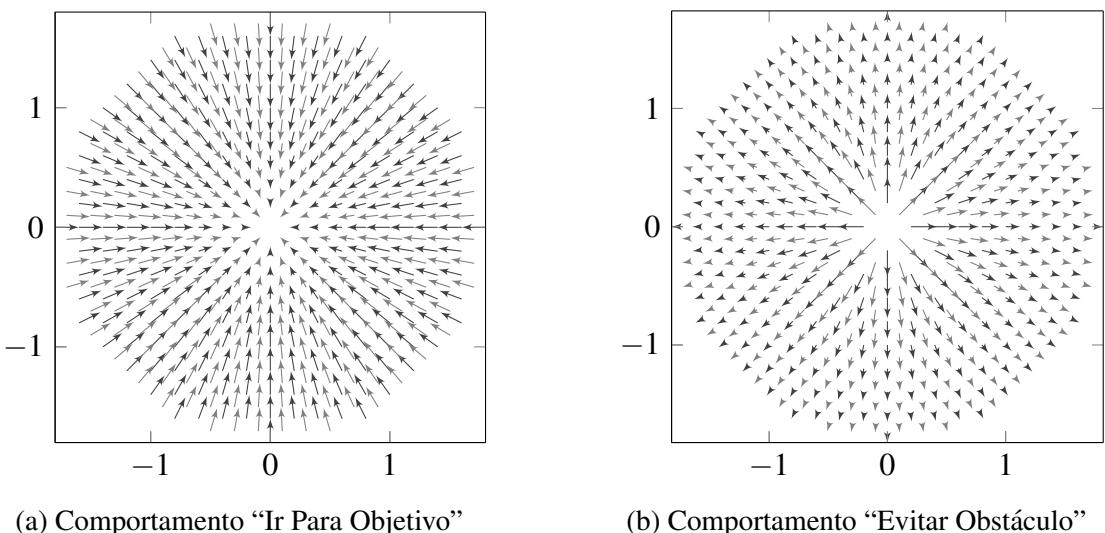
Uma forma de definir comportamentos é por meio de campos potenciais, onde o vetor gradiente determina uma direção de referência para o sistema de controle (ARKIN, 1998; YUN; TAN, 1997). Como exemplo, uma possível escolha para um comportamento de “Ir Para Objetivo” é dado na Equação 1, retratada na Figura 2.a ( $R = 1$ ). Já o comportamento “Evitar Obstáculo” (pontual) pode ser descrito pela Equação 2, retratada na Figura 2.b ( $R = 0$  e  $S = 1$ ). Nas equações,  $S$  é uma esfera de influência,  $R$  é uma distância crítica,  $\hat{\mathbf{u}}_{ipo}$  e  $\hat{\mathbf{u}}_{eo}$  são vetores

unitários que apontam, respectivamente, para o objetivo e na direção oposta ao obstáculo.

$$V(x, y) = \begin{cases} \hat{\mathbf{u}}_{ipo} & , \text{para } d \geq R \\ \frac{d}{R} \hat{\mathbf{u}}_{ipo} & , \text{para } d < R \end{cases} \quad (1)$$

$$V(x, y) = \begin{cases} [0 \ 0]^T & , \text{para } d > S \\ \frac{S-d}{S-R} \hat{\mathbf{u}}_{eo} & , \text{para } R < d \leq S \\ \hat{\mathbf{u}}_{eo} & , \text{para } d \leq R \end{cases} \quad (2)$$

**Figura 2 – Comportamentos definidos por campos potenciais**



**Fonte:** baseado em Arkin (1998), p. 146 e 148

Em um robô real, cada sensor é associado a uma distância até um obstáculo (mesmo que infinitamente distante) e, consequentemente, a um vetor definido pelo comportamento “Evitar Obstáculo”. Uma combinação linear entre estes vetores cria um obstáculo virtual pontual a partir de obstáculos não pontuais (YUN; TAN, 1997).

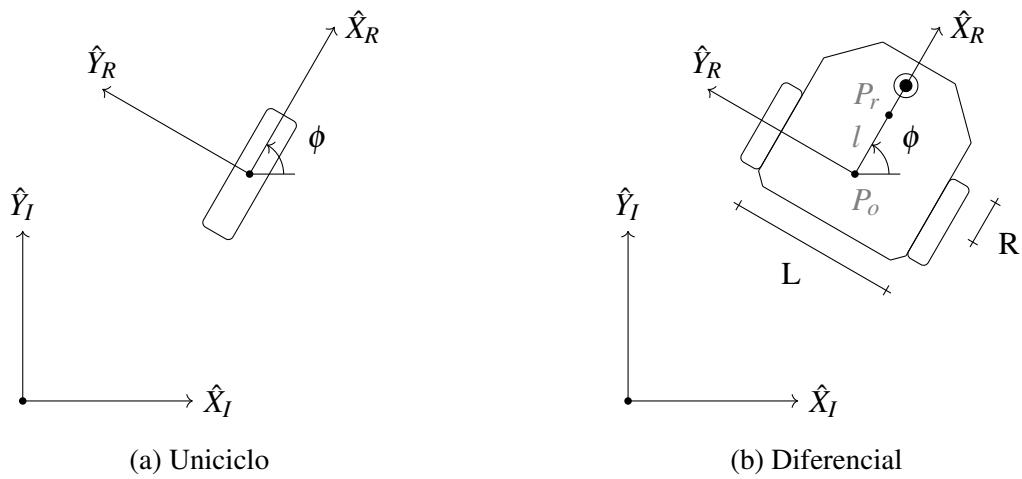
Vetores associados a comportamentos distintos podem ser combinados (linearmente) a fim de obter uma ação intermediária (Fusão, retratada na Figura 1.b), ou suas magnitudes são comparadas de modo que o “vencedor” assume controle pleno (Arbitragem, retratada na Figura 1.a).

A estratégia de utilizar campos potenciais tem suas particularidades, já que podem existir mínimos locais, o que leva o robô a parar antes de chegar ao objetivo. Para resolver este problema, deve-se, ou estabelecer um campo potencial sem mínimos locais, ou desenvolver métodos para escapar deles (YUN; TAN, 1997). Yun e Tan (1997) apresentam uma solução do segundo tipo, ao definir um comportamento “Seguidor de Parede”.

## 2.2 MODELAGEM

Nesta seção serão levantados os modelos cinemáticos para o robô móvel de acionamento diferencial e uniciclo. A seguir, será feita uma relação entre os dois, que permitirá realizar o controle do robô diferencial a partir do uniciclo, tornando mais simples o projeto. Os esquemas dos dois modelos pode ser visto na Figura 3.

**Figura 3 – Modelos uniciclo e diferencial**



**Fonte:** baseado em Siegwart e Nourbakhsh (2004), p. 49 e 54.

### 2.2.1 MODELO CINEMÁTICO DO UNICICLO

O modelo uniciclo representa uma única roda que se movimenta em uma superfície sem deslizamento. Considera-se que existe acesso direto às velocidades linear e angular.

O modelo cinemático pode ser visto na Equação 3 (LAVALLE, 2006). A Equação 4 representa o uniciclo na forma matricial (SIEGWART; NOURBAKHSH, 2004). Nestas equações, as variáveis  $\dot{x}$  e  $\dot{y}$  representam as taxas de variação para as coordenadas do robô,  $\dot{\phi}$  é a taxa de variação para o ângulo do robô em relação ao eixo  $x$  e as variáveis  $v$  e  $\omega$  representam as velocidades linear e angular do robô, respectivamente.

$$\begin{cases} \dot{x} = v \cos \phi \\ \dot{y} = v \sin \phi \\ \dot{\phi} = \omega \end{cases} \quad (3)$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos \phi & 0 \\ \sin \phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (4)$$

### 2.2.2 MODELO CINEMÁTICO DO ROBÔ MÓVEL DE ACIONAMENTO DIFERENCIAL

Neste modelo (Equação 5), são consideradas como entradas as velocidades angulares das rodas esquerda e direita (LAVALLE, 2006). As variáveis  $\dot{x}$ ,  $\dot{y}$  e  $\dot{\phi}$  são as mesmas já descritas no modelo uniciclo. Já  $\omega_l$  e  $\omega_r$  são as velocidades angulares para os motores esquerdo e direito, respectivamente. A constante R é o raio dos pneus e a constante L é a distância entre pneus.

$$\begin{cases} \dot{x} = \frac{R}{2}(\omega_l + \omega_r) \cos \phi \\ \dot{y} = \frac{R}{2}(\omega_l + \omega_r) \sin \phi \\ \dot{\phi} = \frac{R}{L}(\omega_r - \omega_l) \end{cases} \quad (5)$$

O robô móvel de acionamento diferencial, arquitetura usada neste trabalho, é cinematicamente equivalente ao uniciclo (DIB, 2011). Isso pode ser mostrado igualando  $\dot{x}$ ,  $\dot{y}$  e  $\dot{\phi}$  das Equações 3 e 5. Resolvendo para  $\omega_l$  e  $\omega_r$ , tem-se a igualdade da Equação 6. Resolvendo para  $v$  e  $\omega$ , tem-se a igualdade da Equação 7. Tem-se, portanto, as duas expressões necessárias para conversão entre modelos.

$$\begin{aligned} \omega_l &= \frac{2v - L\omega}{2R} \\ \omega_r &= \frac{2v + L\omega}{2R} \end{aligned} \quad (6)$$

$$\begin{aligned} v &= \frac{R}{2}(\omega_l + \omega_r) \\ \omega &= \frac{R}{L}(\omega_r - \omega_l) \end{aligned} \quad (7)$$

Essa equivalência entre modelos é possível pois eles não consideram a dinâmica do sistema (LAVALLE, 2006).

### 2.3 CONSIDERAÇÕES PARA O CONTROLE DE ROBÔS MÓVEIS

O sistema uniciclo tem duas restrições ditas não holonômicas, o que significa que não podem ser integradas para obter uma restrição geométrica (ORIOLO, 2013). Na Equação 8, a primeira se refere à ausência de deslizamento lateral, enquanto a segunda é relativa à condição de rotação pura (ausência de deslizamento no sentido do vetor velocidade) (JAZAR, 2011, pg. 955).

$$\begin{aligned} \dot{x} \sin \phi - \dot{y} \cos \phi &= 0 \\ \dot{x} \cos \phi + \dot{y} \sin \phi &= v \end{aligned} \quad (8)$$

Dois tipos de problemas em controle de robôs móveis podem ser explicitados: estabilização em um determinado estado (configuração) e seguir trajetória (*tracking*). Pela própria natureza do sistema não holonômico, o primeiro problema é considerado mais difícil

(DIB, 2011).

No caso da estabilização, o teorema de Brockett afirma que, para sistemas subatuados, com menos entradas que variáveis de estado, o sistema não pode ser estabilizado assintoticamente (exponencialmente) usando leis de controle por realimentação de estado contínuas e invariantes no tempo (ASTOLFI, 1995; BANAVAR; SANKARANARAYANAN, 2006). Para solucionar esse problema, técnicas de controle não linear têm sido exploradas, envolvendo controladores descontínuos e/ou variantes no tempo (DIB, 2011).

O fato do robô móvel ser de estado completamente controlável apesar de subatuado é devido à presença de restrições não holonômicas (ORIOLO, 2013).

Para o problema de seguir trajetória (*tracking*), o objetivo é seguir uma curva parametrizada em função do tempo (CARONA et al., 2008).

Em Glotfelter e Egerstedt (2018), uma técnica é explorada a fim de tratar o modelo diferencial pelo modelo de integrador-único (ponto controlado por velocidade, ou,  $\dot{\mathbf{x}} = \mathbf{u}$ ). Os autores utilizam como referência um ponto deslocado em relação ao ponto no centro entre as duas rodas. A Figura 3.b indica a posição deste ponto ( $P_r$ ). O valor  $l$  (minúsculo) também pode ser visto na Figura e é justamente a distância entre a origem e  $P_r$ . A Equação 9 mostra os valores de  $x$  e  $y$  para a nova referência. Derivando e substituindo  $\dot{x}$ ,  $\dot{y}$  e  $\dot{\phi}$  da Equação 3, obtém-se a Equação 10.

$$\begin{cases} x' = x_o + l \cos \phi \\ y' = y_o + l \sin \phi \end{cases} \quad (9)$$

$$\begin{cases} \dot{x}' = \dot{x}_o - \dot{\phi} l \sin \phi = v \cos \phi - l \omega \sin \phi \\ \dot{y}' = \dot{y}_o + \dot{\phi} l \cos \phi = v \sin \phi + l \omega \cos \phi \end{cases} \quad (10)$$

A Equação 11 é idêntica à 10, porém, na forma matricial. Sua inversa estabelece um mapeamento direto entre as entradas  $[\dot{x} \ \dot{y}]^T$  do modelo de integrador-único para as entradas do modelo uniciclo (Equação 12), ou modelo diferencial (Equação 13).

$$\begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & l \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (11)$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{l} \end{bmatrix} \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} \quad (12)$$

$$\begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} = \begin{bmatrix} \frac{1}{R} & \frac{-L}{2R} \\ \frac{1}{R} & \frac{L}{2R} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{l} \end{bmatrix} \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} \quad (13)$$

Pelo que pode ser verificado em Glotfelter e Egerstedt (2018), os modelos uniciclo e de integrador-único não são equivalentes, já que  $l$  (distância entre a origem e o ponto  $P_r$  da Figura 3.b) não pode tender a zero, o que levaria a Equação 13 a ter um uma de suas matrizes um valor que tende a infinito. Um controle de modelo preditivo é estabelecido para ajustar  $l$  a fim de obter uma resposta que sacrifique o mínimo possível em precisão e manobrabilidade, comparando com uma escolha estática de parâmetro ( $l = \sqrt{\frac{1-\alpha}{\alpha} \frac{L v_{max}}{R}}$ , com  $\alpha = 0,99$ ).

Esta técnica, também chamada “*look-ahead*”, é utilizada, com modificações, em alguns trabalhos como passo intermediário antes de realizar uma linearização por realimentação (YUN; YAMAMOTO, 1992; NOVEL et al., 1995). Em Yun e Yamamoto (1992) é mostrado que o modelo dinâmico não pode ser linearizado por realimentação entrada-estado. A linearização por realimentação entrada-saída estática só é possível adotando esta nova referência.

Apesar de em Yun e Yamamoto (1992) a dinâmica do sistema ser considerada, em Novel et al. (1995) é dito que um modelo cinemático pode ser usado. Ao considerar a dinâmica (primeiro trabalho) uma realimentação é definida de forma a compensar não linearidades do sistema, a fim de tratá-lo como um modelo de duplo integrador ( $\ddot{\mathbf{x}} = \mathbf{u}$ ). A partir disso, o sistema pode ser controlado por técnicas de controle linear.

Para o modelo cinemático deste trabalho, a definição de um campo potencial não é tão conveniente, já que o negativo do gradiente associado especifica força, de modo que a aceleração é obtida considerando massa (YUN; TAN, 1997; HALLIDAY et al., 2012). Em Kwon et al. (2009), o campo vetorial definido indica velocidades. Os vetores velocidade calculados podem ser usados como entrada do modelo de integrador único.

Apesar do modelo não ser controlável por técnicas lineares, é possível utilizar controle PID para seguir ângulo, com a limitação de não ter controle sobre velocidade linear. Para aplicações onde a convergência não tem restrições temporais significativas, controlar o ângulo é suficiente e será utilizado no caso deste trabalho.

## 2.4 SISTEMAS A EVENTOS DISCRETOS

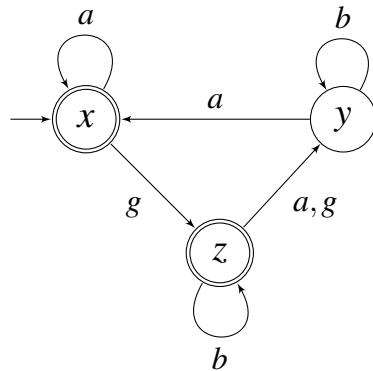
Um Sistema a Eventos Discretos (SED) pode ser caracterizado como um sistema dinâmico, cujos estados são discretos, sendo que transições entre estes estados ocorrem a partir da detecção de eventos assíncronos (CASSANDRAS; LAFORTUNE, 2008). Um evento pode ser definido como um estímulo que provoca transições instantâneas de estado. Na ausência de eventos, o sistema permanece no mesmo estado (CURY, 2001).

Um SED pode ser modelado usando o formalismo presente em teoria de computação:

linguagens regulares e autômatos finitos. O autômato finito determinístico é uma tupla  $M = (Q, \Sigma, \delta, q, F)$  formada por um conjunto de estados ( $Q$ ), um conjunto de símbolos ( $\Sigma$ ), uma função de transição ( $\delta$ ), um estado inicial ( $q$ ), e um conjunto de estados finais ( $F$ , que é subconjunto de  $Q$ ). (CASSANDRAS; LAFORTUNE, 2008; MAHESHWARI; SMID, 2019).

O conjunto de eventos admissíveis em um SED é o alfabeto de uma linguagem e um conjunto de eventos sucessivos seriam palavras desta linguagem. O autômato representa uma linguagem formal. Uma forma de representar o autômato é por meio de um diagrama de transição de estado. Um exemplo pode ser visto na Figura 4.

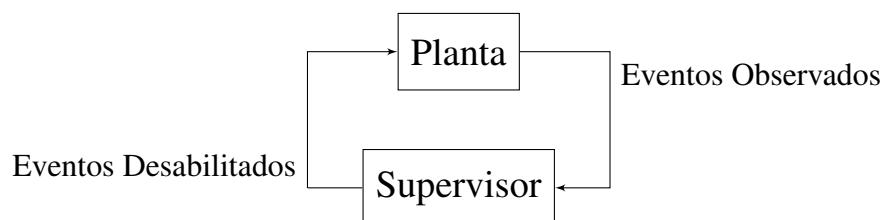
**Figura 4 – Diagrama de Transição de Estado**



**Fonte:** baseado em Cassandras e Lafortune (2008), p. 60.

O SED possui um comportamento, ou dinâmica, que pretende-se controlar, já que algumas cadeias de eventos aceitas pela linguagem podem não ser desejadas. A estrutura ou agente responsável pelo controle é chamado supervisor, que interage com a planta capturando eventos ocorridos (apenas aqueles que são observáveis) e atuando de forma a permitir ou suprimir eventos passíveis de ocorrência (CASSANDRAS; LAFORTUNE, 2008). Um esquema desta interação pode ser visto na Figura 5.

**Figura 5 – SED em malha fechada**



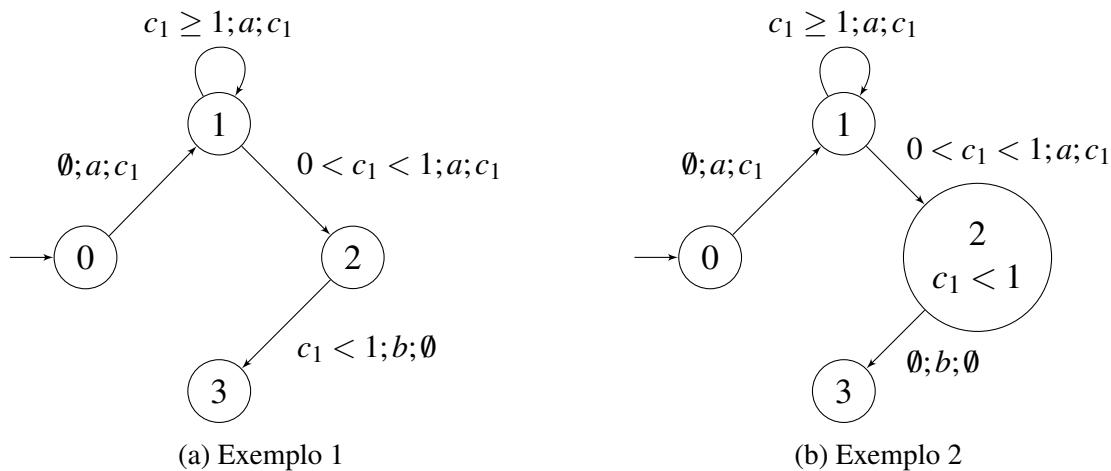
**Fonte:** baseado em Cury (2001), p. 49.

#### 2.4.1 AUTÔMATOS TEMPORIZADOS COM GUARDAS

Uma representação mais completa é o autômato temporizado com guardas, que possui um conjunto de variáveis contínuas e em função do tempo denominadas *clocks*, cuja dinâmica é linear. O *clock* é um temporizador que pode ser usado para estabelecer condições para efetuar transições. Durante essas transições, a variável temporizada pode sofrer *reset* (onde o valor 0 é atribuído) (CASSANDRAS; LAFORTUNE, 2008).

Transições temporizadas são tuplas da forma (guardas; eventos; *reset*), onde “guardas” é um conjunto de condições em variáveis do tipo *clock* que devem ser atendidas para ocorrência de transição, os eventos são componentes já definidos e *reset* especifica um conjunto de variáveis do tipo *clock* às quais deve ser atribuído valor 0. Quando o conjunto “guardas” é vazio  $\emptyset$ , considera-se valor lógico verdadeiro e quando o conjunto *reset* é vazio, nenhuma variável *clock* sofre atribuição de valor 0 (CASSANDRAS; LAFORTUNE, 2008). Dois exemplo de diagramas de transição são mostrados na Figura 6.

**Figura 6 – Exemplos de diagramas de transição para autômatos temporizados com guarda**



**Fonte:** baseado em Cassandras e Lafourture (2008), p. 301.

Para exemplificar, na Figura 6.a a transição do estado 1 para o estado 2 só ocorre se o evento ‘*a*’ acontecer enquanto o contador  $c_1$  possuir valor maior do que zero e menor do que 1. Se a condição de guarda deixar de ser válida, o evento ‘*a*’ não provoca transição para o estado 2, mas sim, ocorre a transição reflexiva “( $c_1 \geq 1; a; c_1$ )”, o valor do *reset* ( $c_1$ ) indica que este *clock* será zerado e a transição para o estado 2 volta a ser possível.

Em casos nos quais a condição de guarda deixa de ser válida e nenhuma outra transição puder acontecer, como, por exemplo, o estado 2 da Figura 6.a,  $c_1$  se torna maior ou igual a 1, ocorre *deadlock* por tempo (*timelock*). Quando é necessário forçar a ocorrência de um

evento para evitar *timelock*, modela-se como na Figura 6.b. A condição associada a um estado é chamada Invariante (CASSANDRAS; LAFORTUNE, 2008).

Por fim, o autômato pode ser representado como uma tupla  $(Q, E, C, \text{Tra}, \text{Inv}, q_0)$ , onde  $Q$  é um conjunto de estados,  $E$  é um conjunto de eventos,  $C$  é um conjunto de temporizadores (variáveis contínuas), “ $\text{Tra}$ ” é um conjunto de transições temporizadas, “ $\text{Inv}$ ” é um conjunto de invariantes e  $q_0$  é o estado inicial (CASSANDRAS; LAFORTUNE, 2008).

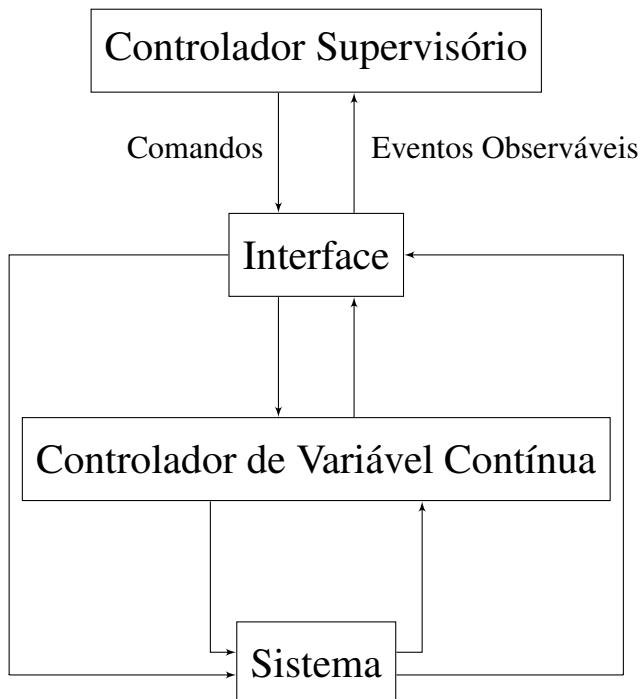
## 2.5 SISTEMAS HÍBRIDOS

Sistemas híbridos são formados quando Sistemas Dinâmicos de Variável Contínua (SDVC) e Sistemas a Eventos Discretos (SED) funcionam de maneira conjunta. Neste caso, controladores de variável contínua são abstraídos por um SED. Transições de estado provocam transições ou “chaveamento” entre controladores (CASSANDRAS; LAFORTUNE, 2008). A Figura 7 mostra o esquema da arquitetura de controle de um sistema híbrido, onde o bloco “Sistema” representa a planta a ser controlada, o bloco “Controlador de Variável Contínua” é o controlador de um sistema de controle clássico. Os dois blocos inferiores são abstraídos como um Sistema a Eventos Discretos. O bloco “Interface” é essa abstração, que recebe dos blocos inferiores informações de variações nas variáveis contínuas, transformando-as em eventos que informam o “Controlador Supervisório”. Este, por sua vez, devolve comandos, também em forma de eventos que são traduzidos pela interface em entradas para os controladores de variável contínua.

Neste sentido, é necessário introduzir o conceito de autômatos híbridos. Ao invés de estados discretos, os estados do sistema são dados por  $(q, \mathbf{x})$ , com  $q \in Q$ , onde  $Q$  é um conjunto de estados discretos e  $\mathbf{x} \in X$ , onde  $X$  é um conjunto de estados contínuos. O estado  $q$  determina o modo de operação do sistema. O autômato híbrido é, portanto, uma extensão do autômato temporizado com guardas, já que as funções dos temporizadores (lineares) podem ser substituídas por funções arbitrárias (CASSANDRAS; LAFORTUNE, 2008).

Esse novo autômato pode ser representado pela tupla  $G_h = (Q, X, E, U, f, \phi, \text{Inv}, \text{guard}, \rho, q_0, \mathbf{x}_0)$ , onde  $Q$  é um conjunto de estados discretos,  $X$  é um conjunto de variáveis contínuas (o espaço de estados  $X \subseteq \mathbb{R}^n$ ),  $E$  é um conjunto de eventos,  $U$  é um conjunto de entradas controláveis ( $U \subseteq \mathbb{R}^m$ ),  $f$  é um campo vetorial ( $f : Q \times X \times U \rightarrow X$ ),  $\phi$  é uma função de transição de estados discretos ( $\phi : Q \times X \times E \rightarrow Q$ ), “ $\text{Inv}$ ” é um conjunto de invariantes, ou domínio ( $\text{Inv} \subseteq Q \times X$ ), “ $\text{guard}$ ” é um conjunto de condições de guarda ( $\text{guard} \subseteq Q \times Q \times X$ ),  $\rho$  é uma função *reset* ( $\rho : Q \times Q \times X \times E \rightarrow X$ ) cujo

**Figura 7 – Controlador Híbrido**



**Fonte:** baseado em Cassandras e Lafortune (2008), p. 43.

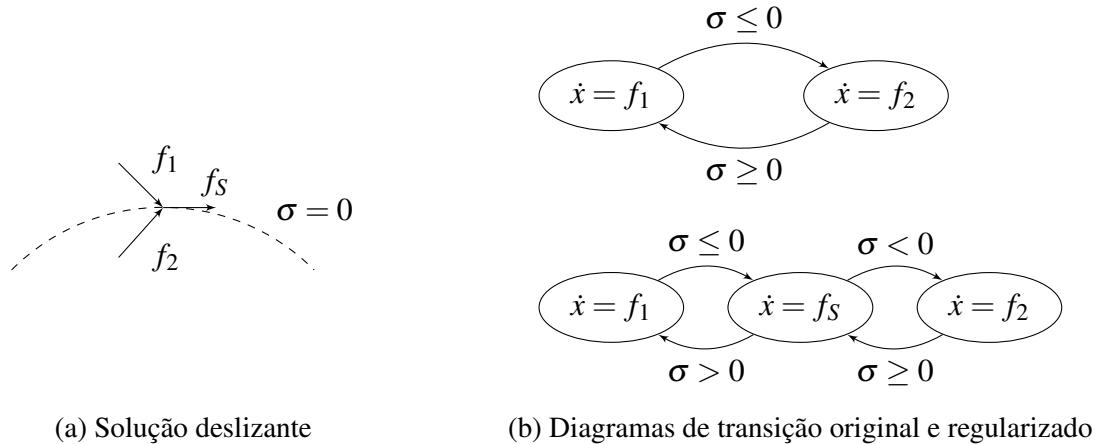
objetivo é alterar  $x$  quando ocorre mudança de modo ( $q$ ),  $q_0$  é o estado discreto inicial (modo de operação) e  $x_0$  é o estado contínuo inicial (CASSANDRAS; LAFORTUNE, 2008, pg. 316).

Em robótica móvel, autômatos fornecem um meio de implementar a arbitragem requisitada pela arquitetura baseada em comportamentos, onde cada modo de operação corresponde a um comportamento. Porém, o uso de autômatos híbridos traz uma particularidade potencialmente problemática: o comportamento de Zenão, no qual, teoricamente, transições infinitas de modo ocorrem em tempo finito, o que bloqueia a passagem de tempo (EGERSTEDT, 2000).

Isso ocorre quando o campo vetorial descontínuo subjacente a um sistema híbrido converge para uma superfície de separação entre modos, o que é chamado de deslize. Essa limitação do autômato, no ambiente real, provoca oscilações, já que mudanças de controladores ocorrem em curto espaço de tempo. Uma possível solução é por meio do processo de regularização, que na prática adiciona um estado de forma a implementar uma “dinâmica de deslize” sobre a fronteira que separa comportamentos (EGERSTEDT, 2000). Este conceito está ilustrado na Figura 8.

O termo “comportamento de Zenão” faz alusão ao filósofo grego Zenão de Eleia, conhecido pelos seus famosos paradoxos (EGERSTEDT, 2000).

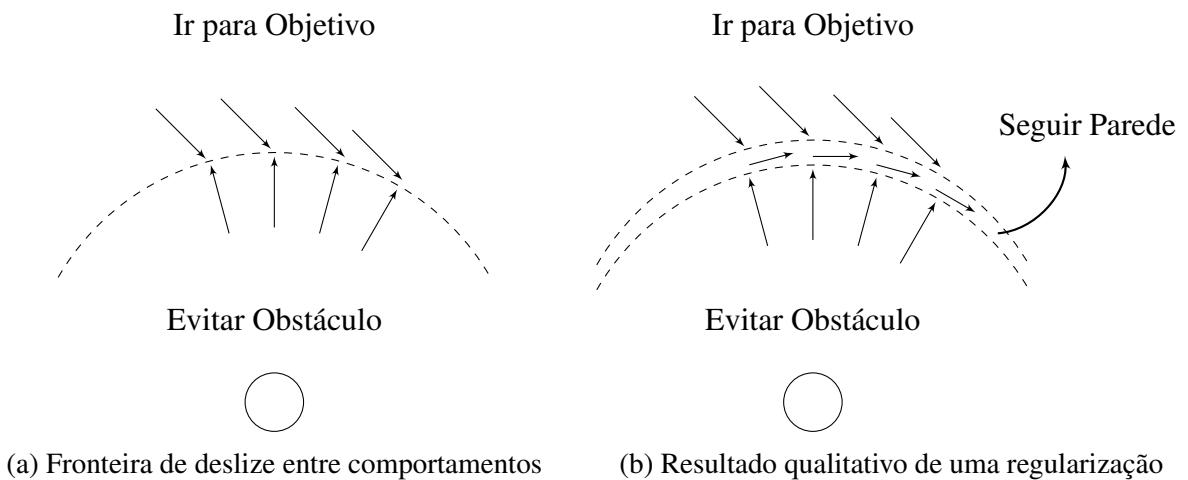
**Figura 8 – Modo deslizante e diagramas de transição**



**Fonte:** baseado em Egerstedt (2000)

Para exemplificar o processo de regularização em robótica móvel, a Figura 9.a mostra um obstáculo circular e uma fronteira de deslize entre comportamentos “Ir para Objetivo” e “Evitar Obstáculo”. Cada comportamento corresponde a uma dinâmica (sistema de variável contínua). As transições entre estados é papel do controlador supervisório. Na Figura 9.b, o comportamento “Seguir Parede” surge como resultado da regularização. As Figuras 9.a e 9.b representam os diagramas de transição da Figura 8.b.

**Figura 9 – Exemplo do modo deslizante em robótica móvel**



**Fonte:** baseado em Egerstedt (2000)

## 2.6 CONTROLADOR PID

O controlador PID é um tipo de controle por realimentação no qual a ação de controle é constituída por três termos: proporcional (P), integral (I) e derivativo (D) (JOHAN; MURRAY,

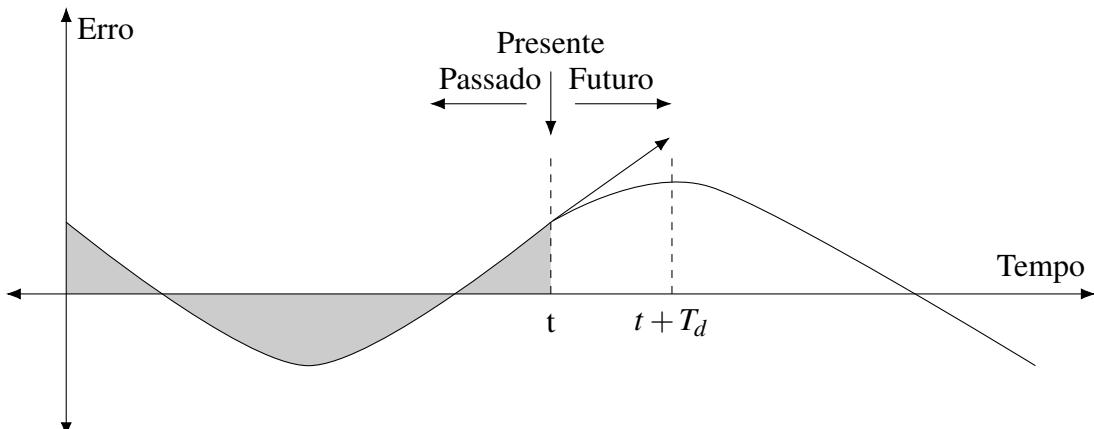
2021).

A Equação 14 mostra o cálculo da ação de controle, onde  $k_p$ ,  $k_i$  e  $k_d$  são constantes dos termos proporcional, integral e derivativo, respectivamente. O termo  $e(t)$  é o erro da variável a ser controlada (JOHAN; MURRAY, 2021).

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt} \quad (14)$$

A Figura 10 mostra uma representação gráfica da ação do controlador. O termo proporcional considera o erro atual  $e$ , portanto, leva em conta o presente. O termo integral considera a soma dos erros  $e$  por isso leva em conta o passado. O termo derivativo projeta a tendência do sinal de controle  $e$ , consequentemente, leva em conta uma espécie de previsão do futuro. As constantes retratam o grau de importância do termo para a composição do sinal de controle (JOHAN; MURRAY, 2021).

**Figura 10 – Representação gráfica da ação de um controle PID**



**Fonte:** baseado em Johan e Murray (2021), pg. 20

A Equação 14 representa a ação de controle em tempo contínuo. A equação em tempo discreto é necessária para implementação computacional. O termo integral é calculado por integração numérica e o termo derivativo por equação de diferenças. A Equação 15 representa a discretização do controlador PID, onde  $T_d$  é o período da discretização (JOHAN; MURRAY, 2021).

$$u(k) = k_p e(k) + k_i T_d \sum_0^k e(k) + k_d \frac{e(k) - e(k-1)}{T_d} \quad (15)$$

Duas considerações práticas são muito importantes em robôs móveis. O *Windup* é uma não-linearidade que ocorre quando a saída do controlador é maior que a capacidade dos

atuadores, o que leva à saturação e, consequentemente, degradação de performance. A zona morta é outra não-linearidade que ocorre quando a saída do controlador é muito pequena e o atuador, por conta de forças de atrito, é incapaz de responder ao sinal de controle (JOHAN; MURRAY, 2021).

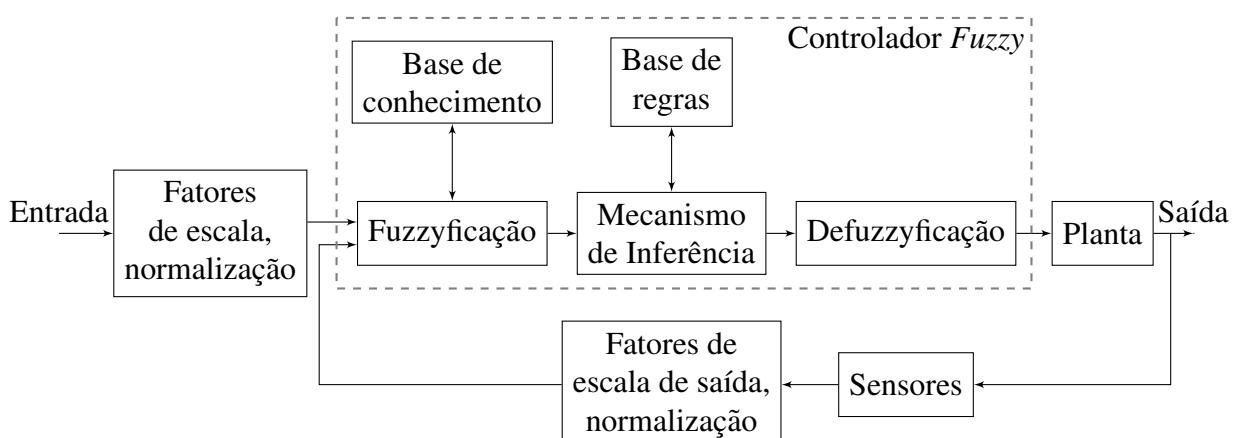
No caso do *Windup*, o robô perde capacidade de cumprir requisitos de velocidade angular. É importante, após o cálculo da atuação, verificar a ocorrência de saturação e, se necessário, sacrificar o requisito de velocidade linear a fim de cumprir o requisito de velocidade angular.

## 2.7 LÓGICA FUZZY EM CONTROLE

Em certas circunstâncias, a lógica Booleana pode não ser suficiente para descrever conhecimentos qualitativos com precisão. Como exemplo, ao determinar como “alto” todo indivíduo com altura maior que 1,70 metros, pessoas com 1,71 ou 2,00 metros seriam consideradas igualmente altas. A lógica Fuzzy acrescenta a informação de “pertinência”, que atribui um “grau de certeza” de um valor (neste exemplo, altura) em relação à classe avaliada (categoria “alto”) (LILLY, 2011).

Sistemas Fuzzy tem por objetivo mesclar aspectos exatos com conhecimentos imprecisos que caracterizam o pensamento humano. Deste modo, aspectos qualitativos de um problema, ou conhecimento especializado, são mapeados e utilizados em um sistema Fuzzy (LILLY, 2011). Um diagrama de blocos de um controlador Fuzzy e seus componentes atuando em um sistema pode ser visto na Figura 11.

**Figura 11 – Diagrama de blocos para um sistema com controlador Fuzzy**



**Fonte:** baseado em Ross (2009), p. 442 e Lilly (2011), p. 29.

Os sistemas Fuzzy, de acordo com Ross (2009), são úteis em contextos onde é

necessário lidar com sistemas muito complexos, compreendidos parcialmente, além de casos nos quais soluções rápidas são desejadas, mesmo que aproximadas.

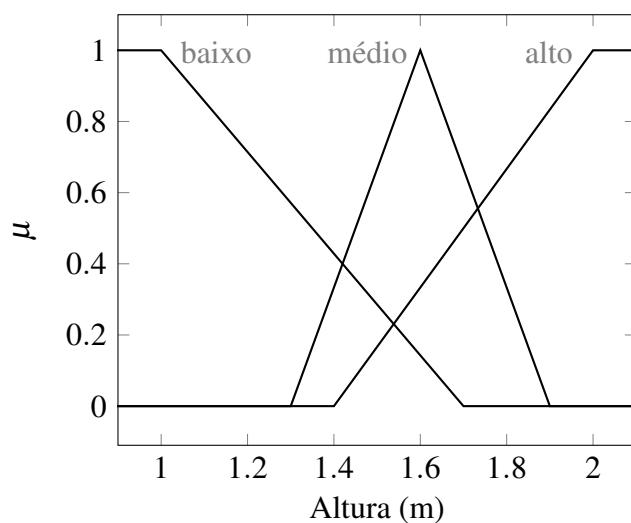
Já Lilly (2011) afirma que problemas não lineares, de difícil solução por controle clássico, são exemplos de situações nas quais sistemas *Fuzzy* podem ser utilizados. Entretanto, o autor alerta que, por conta do alto custo computacional desta solução, ela não é indicada para casos envolvendo problemas lineares e invariantes no tempo, nos quais soluções mais diretas estão disponíveis, por exemplo, controle PID e técnica de alocação de pólos.

### 2.7.1 CONJUNTOS FUZZY

No exemplo dado anteriormente, uma proposição da lógica booleana foi usada para definir um conjunto dos indivíduos “altos”. Este conjunto (convencional) é chamado “crisp”. Em conjuntos *Fuzzy*, associa-se a cada membro de um conjunto um valor real entre 0 e 1 chamado “grau de pertinência”, onde 0 representa exclusão absoluta, 1 representa pertinência total e qualquer valor intermediário representa pertinência parcial (LILLY, 2011).

Os valores numéricos possíveis para altura são chamados “universo de discurso”, o valor “alto” é dito ser um “valor linguístico” associado a “comprimento”, que é a variável linguística no exemplo dado. Para ilustrar o conceito, funções de pertinência associadas à variável linguística “altura” podem ser vistas na Figura 12, onde  $\mu$  é o grau de pertinência. Assim, valores de altura podem pertencer a mais de uma classe, porém com diferentes graus de pertinência (LILLY, 2011).

**Figura 12 – Conjuntos *Fuzzy* possíveis para a variável “altura”**



**Fonte:** baseado em Lilly (2011), p. 32.

As definições conceituais em teoria de conjuntos são diferentes para conjuntos *Fuzzy*.

As mais importantes, explicitadas em Lilly (2011), estão reunidas a seguir, onde  $\mu_1(x)$  e  $\mu_2(x)$  são graus de pertinência relacionados aos conjuntos Fuzzy  $M_1$  e  $M_2$ , respectivamente, definidas para uma mesma variável  $x$  no universo de discurso  $\chi$ .

- Subconjunto:  $M_1$  é subconjunto Fuzzy de  $M_2$  ( $M_1 \subseteq M_2$ ) se  $\mu_1 \leq \mu_2, \forall x \in \chi$ .
- Complemento: o complemento Fuzzy de  $M$  ( $\overline{M}$ ) é dado por  $\mu_{\overline{M}}(x) = 1 - \mu_M(x)$ .
- Intersecção (AND):

Pode ser usada qualquer função que cumpra os requisitos:

1. Um elemento do universo não pode pertencer a dois conjuntos Fuzzy com maior grau de pertinência que a cada conjunto individualmente.
2. Se um elemento não pertence a um dos conjuntos, não pode pertencer à intersecção.
3. Se um elemento pertence aos dois conjuntos Fuzzy com pertinência total, então pertence à intersecção com pertinência total.

Duas funções que cumprem os requisitos são:  $\mu_{M_1 \cap M_2}(x) = \min\{\mu_1(x), \mu_2(x) : x \in \chi\}$  (função mínimo) e  $\mu_{M_1 \cap M_2}(x) = \{\mu_1(x)\mu_2(x) : x \in \chi\}$  (produto algébrico).

- União (OR):

Pode ser usada qualquer função que cumpra os requisitos:

1. Um elemento do universo não pode pertencer à união de dois conjuntos Fuzzy com menor grau de pertinência que a cada conjunto individualmente.
2. Se um elemento pertence a um dos conjuntos, então deve pertencer à união.
3. Se um elemento não pertence a nenhum dos dois conjuntos Fuzzy, então não pertence à união.

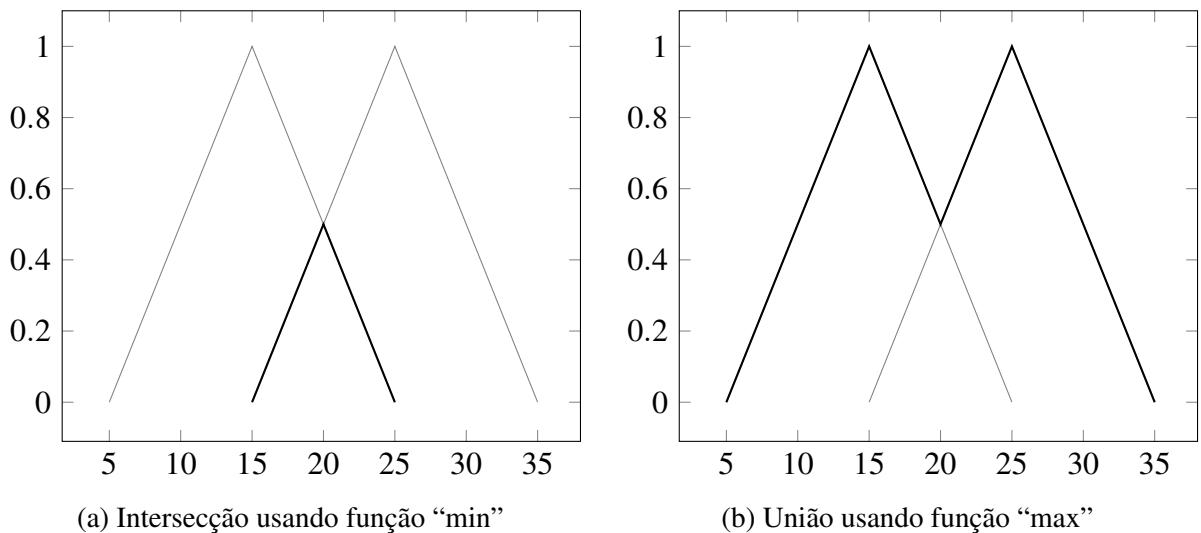
Duas funções que cumprem o requisito são:  $\mu_{M_1 \cup M_2}(x) = \max\{\mu_1(x), \mu_2(x) : x \in \chi\}$  (função máximo) e  $\mu_{M_1 \cup M_2}(x) = \{\mu_1(x) + \mu_2(x) - \mu_1(x)\mu_2(x) : x \in \chi\}$  (soma algébrica).

Um exemplo das operações união e intersecção pode ser visto na figura 13.

### 2.7.2 COMPONENTES DE UM SISTEMA FUZZY

Conhecimento qualitativo é utilizado para gerar regras do tipo “Se P então Q”, que relacionam valores linguísticos de entrada com valores linguísticos de saída. O conjunto de regras forma a “Base de Regras”, que pode ser vista como uma tabela de consulta, com número de dimensões igual ao número de variáveis (PASSINO; YURKOVICH, 1998).

**Figura 13 – Operações união e intersecção em conjuntos Fuzzy**



**Fonte:** baseado em Lilly (2011), p. 20 e 21.

### 2.7.2.1 FUZZIFICAÇÃO

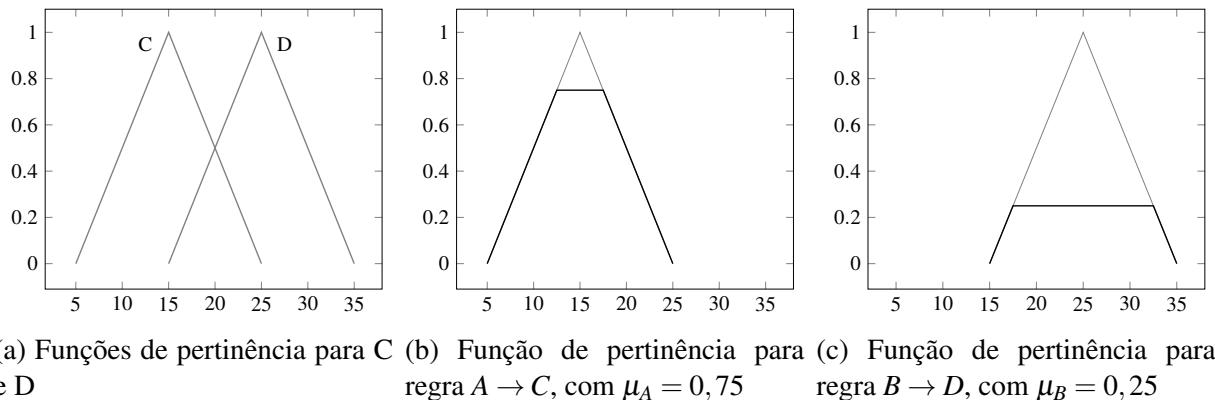
O processo de associar variáveis numéricas a conjuntos fuzzy, a fim de obter valores linguísticos com seus respectivos valores de pertinência, é denominado Fuzzificação. A saída deste processo é o conjunto de graus de pertinência vinculados aos valores linguísticos definidos (PASSINO; YURKOVICH, 1998).

### 2.7.2.2 MECANISMO DE INFERÊNCIA

Ao final do processo de fuzzificação, a Base de Regras deve ser utilizada para encontrar saídas definidas por variáveis linguísticas. Contudo, apenas entradas associadas a graus de pertinência maiores que zero devem ser levados em consideração, a fim de obter um subconjunto de regras (ativas) a serem avaliadas (PASSINO; YURKOVICH, 1998).

Um conjunto de regras ativas e os respectivos graus de pertinência de suas premissas são utilizados para calcular funções de pertinência para cada regra. Como exemplo, dadas as regras  $A \rightarrow C$  e  $B \rightarrow D$ , onde as premissas A e B possuem pertinências  $\mu_A = 0,75$  e  $\mu_B = 0,25$ , e os consequentes C e D possuem funções de pertinência mostradas na Figura 14.a. As Figuras 14.b e 14.c mostram funções de pertinência obtidas para as regras. As saídas destas regras são vistas como recomendações que, apesar de conflitantes, podem ser combinadas de acordo com sua importância (pertinência) (PASSINO; YURKOVICH, 1998).

**Figura 14 – Inferência em Lógica Fuzzy**



**Fonte:** baseado em Passino e Yurkovich (1998), p. 43 e 44.

### 2.7.2.3 DEFUZZIFICAÇÃO

O processo de associar recomendações distintas dadas pelo conjunto de regras ativas a variáveis numéricas de saída é denominado Defuzzificação. Existem vários meios de cumprir tal objetivo, sendo que um método bastante utilizado é o cálculo do “centro de gravidade” (COG) das funções de pertinência das regras. A Equação 16 mostra este cálculo, no qual  $b_i$  é o centro da função de pertinência (PASSINO; YURKOVICH, 1998).

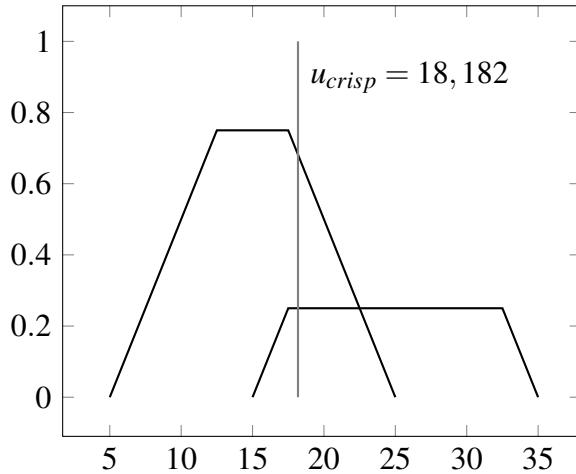
$$u_{crisp} = \frac{\sum_i b_i \int \mu_i}{\sum_i \int \mu_i} \quad (16)$$

O valor  $\int \mu_i$  pode ser obtido analiticamente para uma função de pertinência triangular. Dada uma base “b” e altura de corte “h”, pode-se mostrar que a área se iguala a  $b(h - \frac{h^2}{2})$  (PASSINO; YURKOVICH, 1998). Para o exemplo anterior, o processo de defuzzificação foi ilustrado na Figura 15.

As duas curvas da Figura 15 possuem áreas sobrepostas que estão sendo somadas duas vezes no cálculo de defuzzificação. Algumas bibliografias (por exemplo, Passino e Yurkovich (1998)) citam a existência de um cálculo global, que utiliza todas as funções para calcular a chamada “função de agregação” e o centro de massa é calculado sobre ela.

Nem todas as bibliografias mostram exemplos utilizando este método, por conta do custo computacional. Porém, ao longo do desenvolvimento, observou-se que o simulador (Matlab) estava computando a função de agregação por integração numérica, o que gerava diferenças de resultado no cálculo do microcontrolador. A função de agregação em simulação está sendo calculada aplicando a função *max()* sobre todas as funções parciais (verificado empiricamente). A Equação 17, encontrada em Ross (2009) mostra a saída *fuzzy* sobre a função

**Figura 15 – Defuzzificação COG**



**Fonte:** baseado em Passino e Yurkovich (1998), p. 47.

de agregação, pelo método do centroide.

$$u_{crisp} = \frac{\int \mu^*(x)xdx}{\int \mu^*(x)dx} \quad (17)$$

Para manter o resultado da implementação fiel à simulação, mas ao mesmo tempo evitar ter de realizar um método de integração numérica no microcontrolador, foi desenvolvida uma forma de calcular a integral analítica da curva, o que reduz complexidade computacional, ao custo de aumentar complexidade de espaço para o armazenamento de uma estrutura de dados.

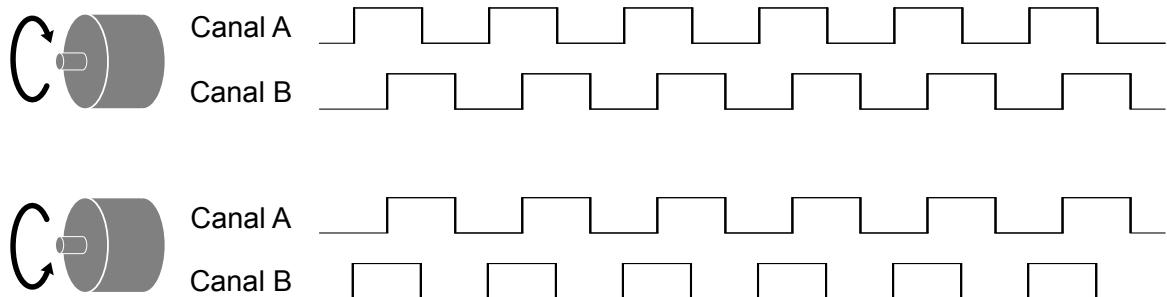
Foi utilizada uma lista encadeada para armazenar parâmetros das funções de pertinência. A lista é ordenada pela posição do centro de massa da função de pertinência no eixo X. Como essas funções são todas triangulares, a função de agregação é descontínua, porém formada por retas. O algoritmo para cálculo do centro de massa, ao percorrer a lista, calcula e soma as integrais definidas parciais.

## 2.8 ODOMETRIA

Odometria em robótica móvel é o processo de cálculo da posição do robô utilizando sensores. No caso específico de um robô móvel de acionamento diferencial, deseja-se calcular a coordenada em um plano ( $x$  e  $y$ ) e a orientação ( $\phi$ ) do robô a partir de sensores de efeito hall acoplados aos motores. (OLSON, 2009)

Cada motor possui dois sensores, de modo a produzir sinais em canais A e B, que se comportam conforme a Figura 16. Para medir velocidade, deve-se calcular a quantidade de pulsos gerados em um período de tempo, utilizando a especificação de pulsos por revolução

(PPR), bem como a medida de circunferência do pneu (DYNAPAR, 2020). A Equação 18 mostra o cálculo de velocidade, onde  $N_{ticks}$  é o período de amostragem,  $N_{PPR}$  é o número de partes por revolução,  $R$  é o raio do pneu e  $T$  é o período de amostragem.



**Figura 16 – Sinais de saída dos canais dos *encoders* para sentido de rotação**

**Fonte:** baseado em Dynapar (2020)

$$V_{pneu} = \frac{N_{ticks}}{N_{PPR}} \frac{2\pi R}{T} \quad (18)$$

A utilização de uma caixa de redução entre o motor e o pneu adiciona uma particularidade interessante, pois ao reduzir a velocidade máxima, aumenta a precisão dos sensores pela razão do sistema de engrenagens.

Para descobrir a direção de rotação, é necessário analisar a diferença de fase entre os sinais. Ao detectar borda de subida e descida em um dos canais e a seguir verificar se os dois possuem sinais lógicos iguais ou diferentes, um contador é incrementado ou decrementado.

Decidir se o contador será incrementado ou decrementado na presença de sinais iguais é um processo empírico, pois o sentido de movimento depende da fixação dos motores no chassi do robô. Como há diferença de 180 graus na fixação, um sentido horário de rotação pode ser "para frente" ou "para trás" dependendo de qual motor for analisado.

Pode-se utilizar apenas borda de subida para esse cálculo, porém, ao levar em consideração as duas bordas do sinal, duplica-se a contagem de pulsos por revolução.

O cálculo do estado atual a partir de estados anteriores e medições dos encoders pode ser obtido a partir da Equação 19, onde  $[x, y, \phi]^T$  é o estado anterior,  $[x' y' \phi']^T$  é o estado a ser calculado,  $L$  é a distância entre os pneus e  $d_{esq}$  e  $d_{dir}$  são as distâncias percorridas pelos pneus esquerdo e direito, respectivamente. (OLSON, 2009).

$$\begin{cases} x' = x + \frac{d_{esq} + d_{dir}}{2} \cos \phi \\ y' = y + \frac{d_{esq} + d_{dir}}{2} \sin \phi \\ \phi' = \phi + \frac{d_{dir} - d_{esq}}{L} \end{cases} \quad (19)$$

### 3 MATERIAIS

Este Capítulo descreve os requisitos materiais para o desenvolvimento do trabalho.

#### 3.1 COMPONENTES FÍSICOS E ELETRÔNICOS

Os componentes eletrônicos utilizados neste trabalho podem ser vistos na Tabela 1.

**Tabela 1 – Tabela de custos**

Item	Qty	Custo Unitário	Custo Total
Bateria LiPo Limskey 11,1v 3S 4200mAh	1	112,05	112,05
Ponte H dupla L298N	1	6,12	6,12
Sensor infravermelho GP2Y0A21YK0F, alcance de 10 a 80 cm	5	13,07	65,35
Módulo conversor Buck DC-DC, saída 5v	1	9,14	9,14
Módulo Bluetooth XM-15	1	25,00	25,00
Conjuntos de motores, caixa de redução 1:34 e sensores de efeito hall (341,2 PPR)	2	47,015	94,03
Esfera de rolagem	1	2,002	2,002
Tiva Connected LaunchPad TM4C1294	1	80,00	80,00
		Total:	393,69

Os componentes físicos para o corpo do robô, como chapa de MDF para o chassis, papel paraná para suporte estrutural, parafusos de rosca, abraçadeiras não tiveram seus custos contabilizados.

#### 3.2 MONTAGEM FÍSICA

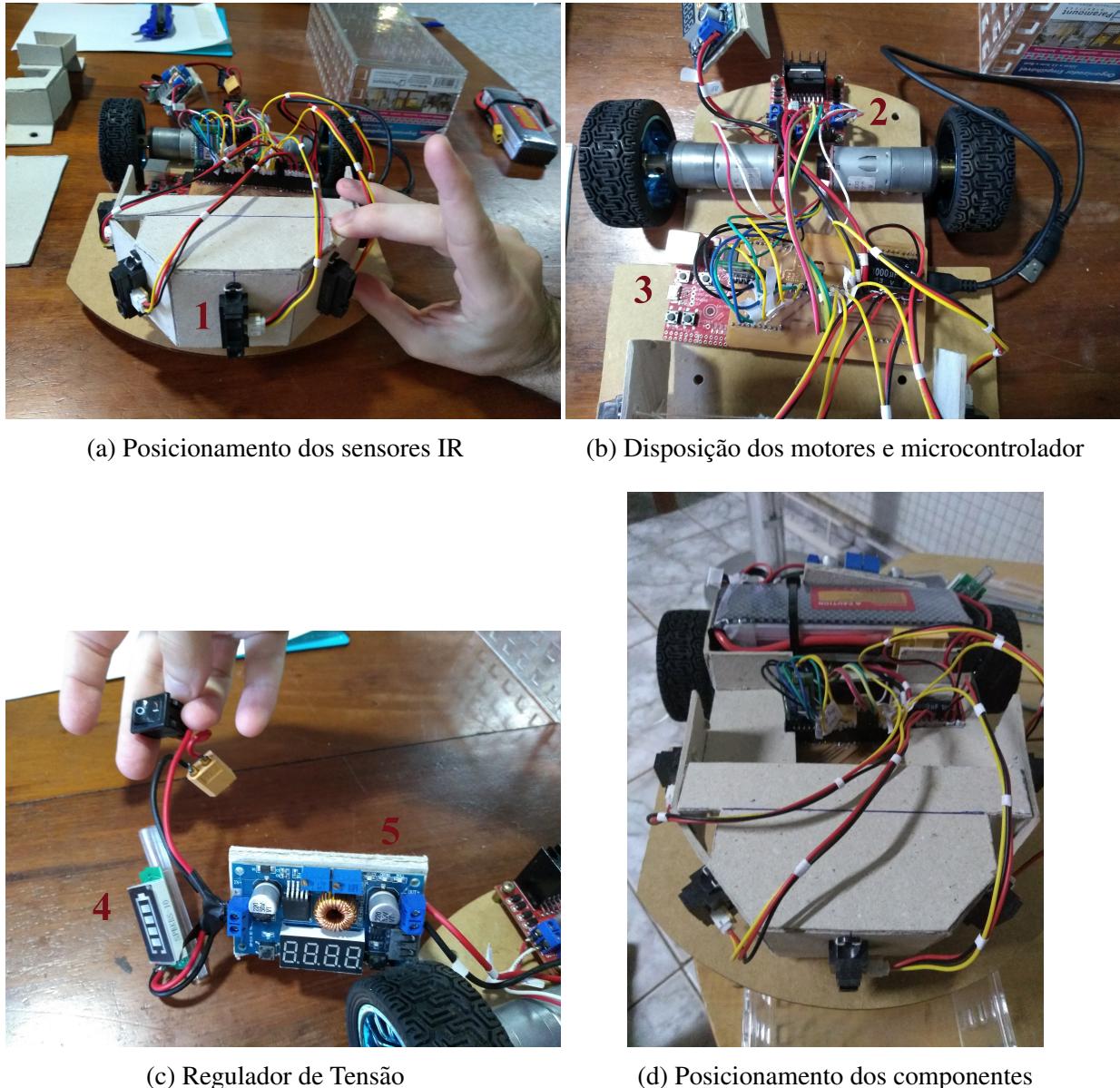
Nesta seção pretende-se mostrar as questões práticas necessárias para a construção do robô.

Os componentes físicos do robô podem ser vistos na Figura 17. Alguns números foram acrescentados às imagens ao lado dos periféricos utilizados, a fim de rotulá-los.

O número 1 na Figura 17.a indica o sensor infravermelho. Os números 2 e 3 na Figura 17.b indicam o driver e o microcontrolador, respectivamente. Os números 4 e 5 na Figura 17.c indicam, respectivamente, o exibidor de nível energia para bateria de lítio (que não é necessário para o funcionamento) e o regulador de tensão regulável.

A Figura 18.a mostra o posicionamento dos periféricos e parafusos roscados. A Figura 18.b retrata a montagem concluída.

**Figura 17 – Materiais e robô após montagem**



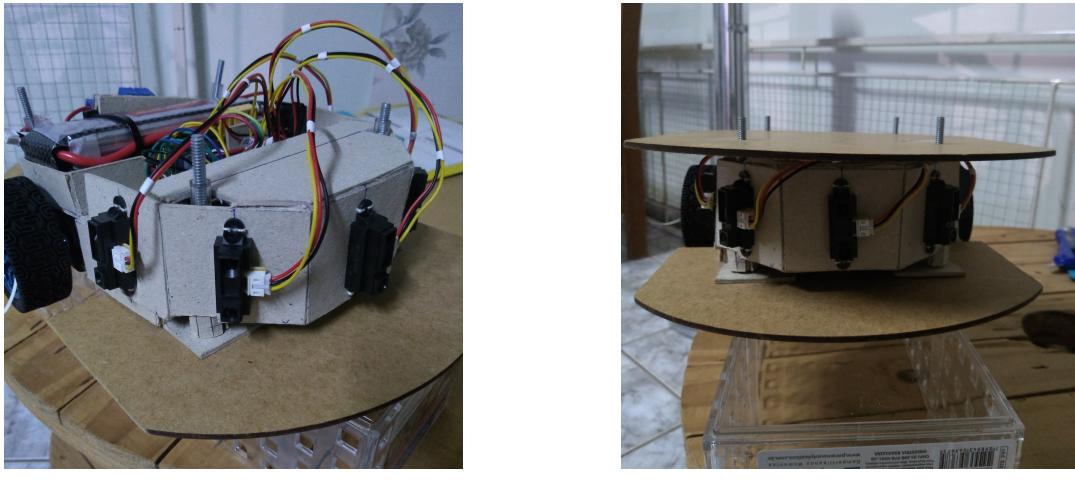
**Fonte: autoria própria**

### 3.2.1 ESPECIFICAÇÕES

As variáveis L e R na Equação 5, para o robô deste trabalho, valem respectivamente 18 cm e 3,4 cm.

O robô deste trabalho foi feito com base no robô “QuickBot”. Contudo, o sensor infravermelho do “QuickBot” é o “GP2Y0A41SK0F”, cuja região de medição está no intervalo entre 4 e 30 cm. Por conta deste sensor estar obsoleto, ele foi substituído pelo “GP2Y0A21YK0F” que possui a curva de tensão por distância representada na Figura 19. A faixa de distância do sensor deste trabalho está entre 10 e 80 cm, como pode ser visto na Figura.

**Figura 18 – Robô após montagem**



**Fonte: autoria própria**

Para a região da curva onde  $d > 4.95\text{cm}$ , o polinômio que a aproxima, associando a distância pela tensão ( $d(v)$ ), obtido utilizando a função “polyfit” do Matlab pode ser visto na Equação 20.

$$\begin{aligned} d(v) = & 2.7802212625v^6 - 35.1150300110v^5 + 179.6031433005v^4 \\ & - 477.9449116299v^3 + 706.3400747125v^2 - 569.7367375002v + 221.2678651473 \end{aligned} \quad (20)$$

### 3.2.2 CURVAS DOS MOTORES

A partir de um requisito de velocidade angular, é necessário definir a quantidade de acionamento que será utilizada no motor (porcentagem de PWM).

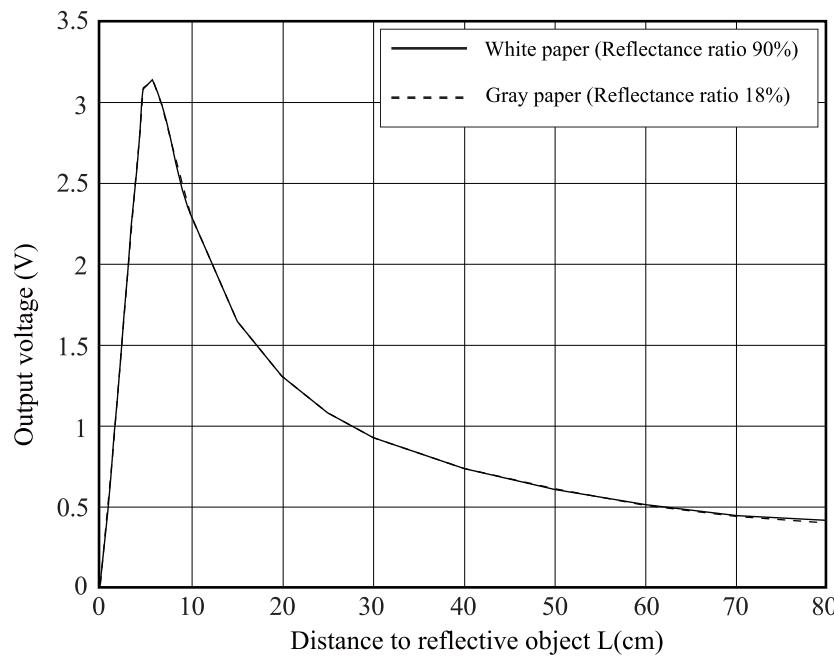
Para isso, foi feito um levantamento da curva dos motores. Ao alterar a porcentagem do acionamento PWM com incrementos de 1%, a saída do sensor de efeito *hall* é utilizada para calcular a velocidade em rotações por segundo.

A Figura 20 mostra essa curva para o caso de ausência de carga, ou seja, o teste foi realizado sem que o pneu tocasse o chão.

A Figura 21 mostra a curva para o caso de presença de carga. Neste caso, o teste foi feito com o robô no chão, em uma quadra de esportes que, por não ter o chão tão liso, levou a um gráfico bastante ruidoso. Os três pontos fora da curva ocorreram devido a pequenos chutes dados no robô a fim de evitar uma colisão com parede.

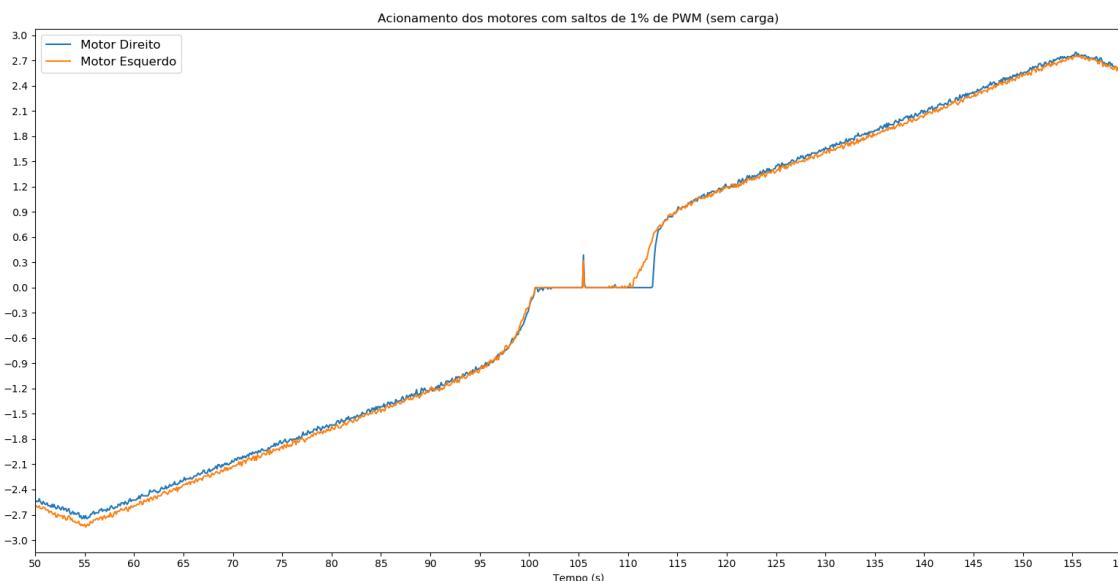
A partir do teste sem carga do motor direito, foi feita uma regressão linear utilizando a

**Figura 19 – Resposta do sensor infravermelho**



**Fonte: SHARP Corporation (2006)**

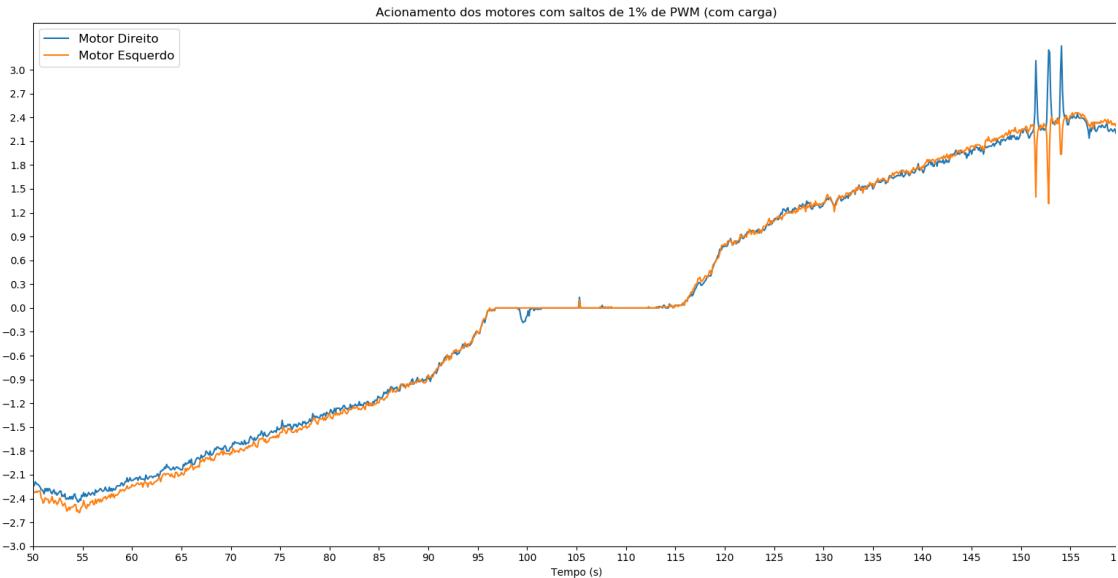
**Figura 20 – Curva do motor sem carga**



**Fonte: autoria própria**

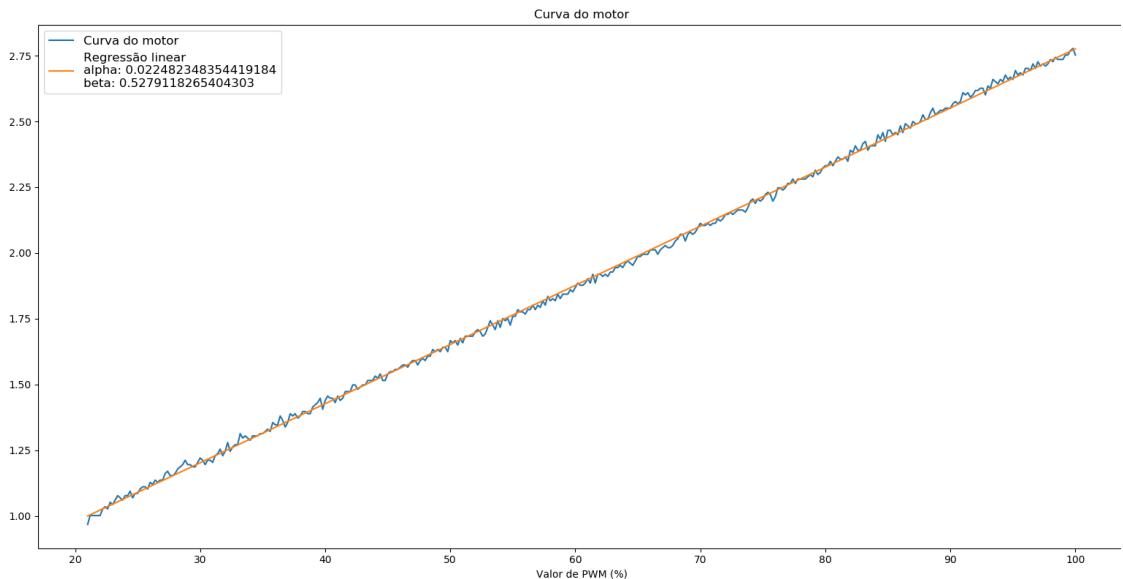
parte da curva cujo regime é linear. A Figura 22 mostra o resultado dessa regressão, bem como as constantes obtidas.

**Figura 21 – Curva do motor com carga**



**Fonte: autoria própria**

**Figura 22 – Regressão Linear para a curva do motor**



**Fonte: autoria própria**

### 3.3 O SIMULADOR SIMIAM

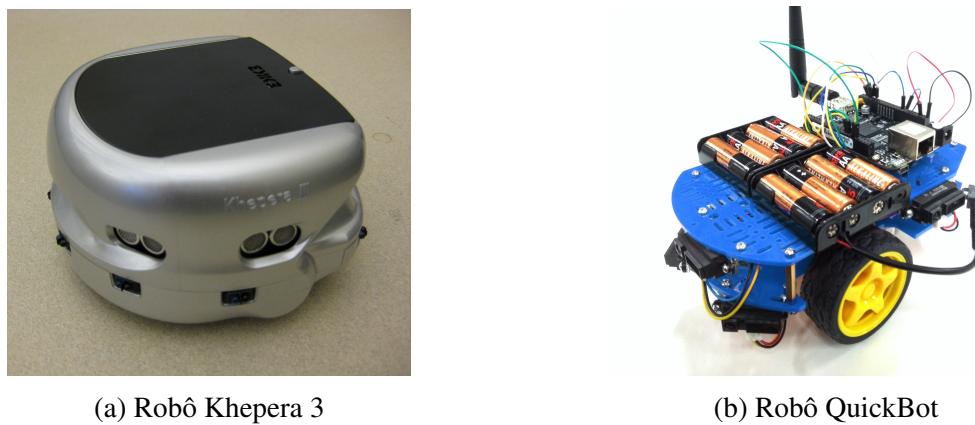
Entre as ferramentas, será utilizado *software* Matlab, com o simulador Simiam (GEORGIA ROBOTICS AND INTELLIGENT SYSTEMS LABORATORY, 2013), que é implementado como um aplicativo Matlab.

O Simulador Simiam foi implementado pela Universidade Georgia Tech, inicialmente

oferecendo apenas suporte ao robô Khepera 3. Posteriormente, para atender necessidades do curso *Control of Mobile Robots*, hospedado na plataforma “Coursera.org”, o robô de baixo custo QuickBot foi adicionado. Os robôs Khepera e QuickBot reais podem ser vistos na Figura 23. Suas contrapartes simuladas estão retratadas na Figura 24.

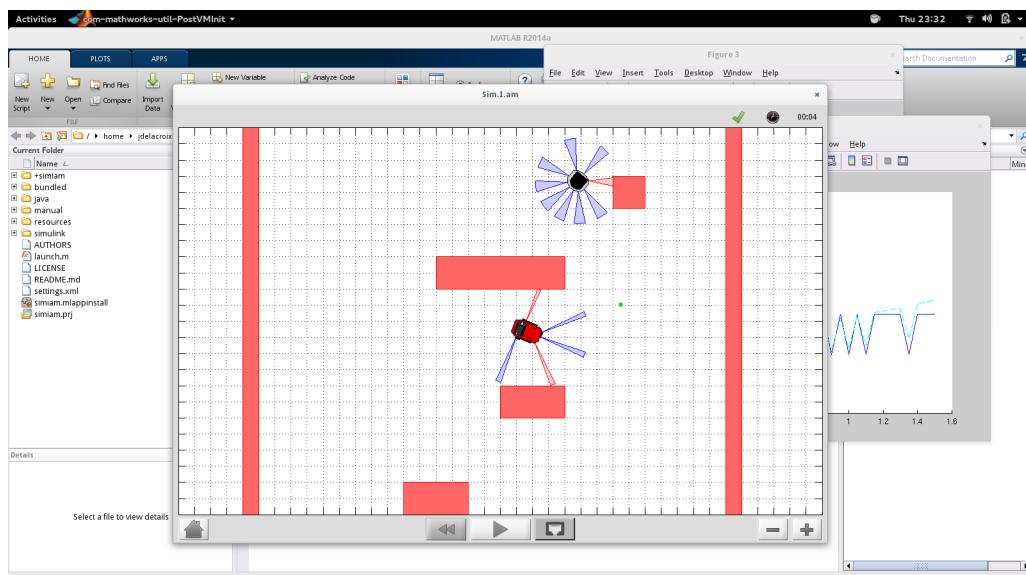
O curso mencionado foi removido da plataforma no dia 17 de Agosto de 2020 e apenas os alunos concluíntes possuem acesso. De qualquer forma, o simulador ainda pode ser obtido gratuitamente.

**Figura 23 – Robôs Khepera 3 e QuickBot**



**Fonte:** Wang (2008), Fuchs et al. (2013)

**Figura 24 – Robôs Khepera 3 e QuickBot em simulação**



**Fonte:** Croix (2013)

O simulador mencionado possui uma interface gráfica intuitiva e uma arquitetura interna simples, facilmente customizável. As subseções a seguir apresentam os pacotes mais

importantes, bem como as principais classes que os compõem e chama atenção para as alterações realizadas a fim de oferecer suporte ao robô implementado neste trabalho, tornando a simulação coerente com este novo robô.

### 3.3.1 PACOTE “UI”

O pacote “ui” é responsável pela interface gráfica do simulador. A classe “AppWindow” possui um método construtor que inicializa atributos e o método “load\\_ui” que invoca o método “create\\_layout”, responsável por criar o leiaute da interface.

O botão start, criado pelo método anterior é tratado por “ui\\_button\\_start”. Esta é a função principal responsável por criar o ambiente de simulação (definido no arquivo “settings.xml”), inserir um ou mais robôs e iniciar a simulação.

Alterações mínimas foram efetuadas nesta etapa. Foram adicionados comandos para maximizar a janela do simulador e para ajustar o “zoom” de modo a permitir uma visão panorâmica de todo o ambiente de simulação. O arquivo de configuração “settings.xml” foi alterado a fim de ajustar o “tamanho do mundo” ao tamanho da tela.

### 3.3.2 PACOTE “SIMULATOR”

O pacote “simulator” realiza de fato a simulação. A classe “World” é responsável por extrair informações do arquivo “.xml”, como quantidade e localização inicial de robôs e obstáculos, armazenado em estruturas de dados. A classe “Simulator” atualiza a simulação na janela, enquanto a classe “Physics” é responsável por detectar colisões de robôs com obstáculos e entre si (em caso simulações multirobôs).

As alterações neste pacote foram todas na classe “Simulator”, no intuito de possibilitar a captura da simulação em video “.mp4” ou “.gif”. Apesar de não ser uma alteração fundamental para a etapa de execução do trabalho, é necessária para a apresentação. Não foi criado um botão na interface para especificar ao simulador a captura em video. Assim, é necessário, no construtor da classe Simulator, atribuir valor verdadeiro aos atributos “gravarvideo” e/ou “gravargif”.

### 3.3.3 PACOTE “ROBOT”

O pacote “robot” define um ou mais robôs passíveis de serem instanciados. A classe “QuickBot” estabelece parâmetros físicos, tais como informações geométricas, posicionamento

dos sensores no corpo do robô e estabelece limitações, como velocidades de saturação e zona morta dos motores. Além de instanciar objetos das classes “DifferentialDrive”, “ProximitySensor” e “WheelEncoder”, “QuickBot” extende a classe “Robot”.

A classe “DifferentialDrive” implementa a “dinâmica” dos modelos de acionamento diferencial e uniciclo. Com os métodos “uni\_to\_diff” e “diff\_to\_uni”, a equivalência com o modelo apresentado na Equação 6 é estabelecida no espaço de simulação.

A classe “ProximitySensor” estabelece características do sensor infravermelho usado no QuickBot, tais como distâncias mínimas e máximas, espalhamento, localização e direção em relação ao corpo do robô e adiciona um ruído gaussiano. “WheelEncoder”, similarmente, estabelece características do *encoder*.

As alterações nesta etapa foram no intuito de incluir o robô deste trabalho no Simiam. O arquivo “settings.xml” deve determinar que um objeto (robô) da nova classe criada seja instanciado, ao invés do objeto da classe QuickBot.

### 3.3.4 PACOTE “CONTROLLER”

O pacote “controller” é responsável pela implementação de todos os controladores e supervisores dos robôs implementados. A classe “Supervisor” é extendida para obter o controlador supervisório de cada robô a ser simulado. Para o QuickBot e Khepera 3, os respectivos supervisores são definidos pelas classes “QBSupervisor” e “K3Supervisor”. Essas classes definem máquinas de estado, onde cada estado é associado a um controlador de variável contínua. Esses controladores, por sua vez, extendem a classe “Controller” e implementam os “comportamentos”, ou modos, dos robôs.

As alterações neste pacote foram responsáveis pela inclusão de suporte a controladores *Fuzzy*, além de adicionar o supervisório específico para o robô desenvolvido.

## 4 DESENVOLVIMENTO

Este Capítulo apresenta o que foi desenvolvido como resultado deste trabalho, visando alcançar os objetivos propostos.

### 4.1 DESENVOLVIMENTO DOS CONTROLADORES

Nesta seção pretende-se investigar a arquitetura comportamental em robótica móvel para as duas estratégias clássicas de implementação: arbitragem e fusão de comportamentos. Para isso, controladores híbrido e fuzzy serão criados, o primeiro para arbitragem e o segundo para fusão.

#### 4.1.1 CONTROLE DE SISTEMA HÍBRIDO

Nesta subseção, pretende-se demonstrar um controlador simples capaz de solucionar o problema de navegação, usando autômato híbrido. Para cada estado no autômato, um controlador é selecionado e uma recomendação de saída é calculada.

A recomendação para todos os comportamentos é dada por um vetor. Seu ângulo é calculado para o sistema de coordenadas global e, subtraindo o ângulo do robô, o erro é calculado. A partir do erro, um controlador PID calcula a velocidade angular.

O cálculo efetuado pelo controlador PID é mostrado na Equação 21, onde  $\omega$  é a velocidade angular,  $\varepsilon(k)$  é o erro para a k-ésima iteração e  $T$  é o período de amostragem.

$$\omega = K_p \varepsilon(k) + K_i \sum_0^k \varepsilon(k)T + k_d \frac{\varepsilon(k) - \varepsilon(k-1)}{T} \quad (21)$$

A saída do controlador PID fornece uma recomendação para velocidade angular do modelo uniciclo. A velocidade linear é arbitrária e inicialmente é definida como a velocidade máxima do robô, mas cada comportamento tem a prerrogativa de alterá-la, caso necessário.

A Equação 6, se utilizada diretamente, pode retornar valores de  $w_l$  e  $w_r$  saturados ou em região de zona morta. Pode-se concluir que a velocidade linear é uma recomendação de baixa prioridade em relação à velocidade angular, já que só é possível seguir ângulo definindo essa prioridade.

É necessário utilizar um algoritmo para reduzir velocidade linear caso recomendação ultrapasse limites máximos, a fim de atender recomendação de velocidade angular. O Algoritmo 1 mostra o procedimento capaz de garantir a conversão segura entre modelo uniciclo e de acionamento diferencial.

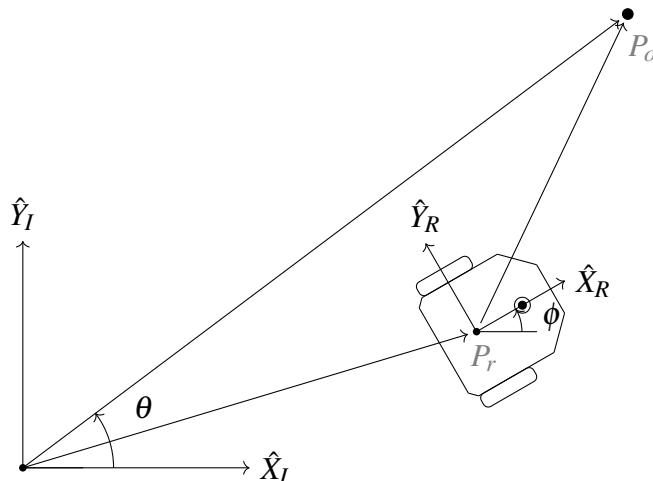
A ideia desse algoritmo é verificar se há saturação e, caso positivo, a diferença entre velocidade sugerida e velocidade máxima permitida pelos motores é subtraída de  $w_l$  e  $w_r$ . Além de respeitar o limite superior de velocidade, o limite inferior (região de zona morta) também é respeitado e, se necessário, a diferença é somada para garantir que não há travamento de motores devido a atrito.

As funções “uniToDiff()” e “diffToUni()” efetuam as conversões de modelos matemáticos das Equações 6 e 7, respectivamente.

#### 4.1.1.1 COMPORTAMENTO "IR PARA OBJETIVO"

O comportamento “Ir Para Objetivo”, exemplificado na Figura 25, calcula um vetor objetivo a partir do centro de massa do robô. A Equação 22 mostra o cálculo, onde  $P_o$  e  $P_r$  são pontos que podem ser vistos na Figura. A recomendação para velocidade linear é a máxima, já que o robô está livre de obstáculos.

**Figura 25 – Comportamento Ir para Objetivo**



**Fonte:** autoria própria

$$\mathbf{u}_{\text{ipo}} = P_o - P_r \quad (22)$$

O controlador PID para comportamento “Ir Para Objetivo” possui parâmetros  $k_p = 4$  e  $k_i = k_d = 0,01$ . O erro no ângulo, entrada do controlador, é calculado pela Equação 23, onde

---

**Algoritmo 1** Uniciclo para Acionamento Diferencial priorizando  $\omega$ 


---

**Entrada:**  $v, \omega$

**Saída:**  $\omega_l, \omega_r$

```

1: Se  $v > 0$  Faça
2:    $vlim \leftarrow \max(\min(\text{abs}(v), R \cdot VEL\_MAX), R \cdot VEL\_MIN)$ 
3:    $wlim \leftarrow \max(\min(\text{abs}(\omega), (R/L) \cdot (VEL\_MAX - VEL\_MIN)), 0)$ 
4:    $w_{l,lim}, w_{r,lim} \leftarrow \text{uniToDiff}(vlim, wlim)$ 
5:    $velocidadeMaior \leftarrow \max(w_{l,lim}, w_{r,lim})$ 
6:    $velocidadeMenor \leftarrow \min(w_{l,lim}, w_{r,lim})$ 
7:   Se  $velocidadeMaior > VEL\_MAX$  Faça
8:      $w_{l,lim} \leftarrow w_{l,lim} - (velocidadeMaior - VEL\_MAX)$ 
9:      $w_{r,lim} \leftarrow w_{r,lim} - (velocidadeMaior - VEL\_MAX)$ 
10:  Senão Se  $velocidadeMenor < VEL\_MIN$  Faça
11:     $w_{l,lim} \leftarrow w_{l,lim} + (VEL\_MIN - velocidadeMenor)$ 
12:     $w_{r,lim} \leftarrow w_{r,lim} + (VEL\_MIN - velocidadeMenor)$ 
13:  Fim Se
14:  Se  $v \geq 0$  Faça
15:     $vlim \leftarrow 1$ 
16:  Senão
17:     $vlim \leftarrow -1$ 
18:  Fim Se
19:  Se  $\omega \geq 0$  Faça
20:     $wlim \leftarrow 1$ 
21:  Senão
22:     $wlim \leftarrow -1$ 
23:  Fim Se
24:   $v, \omega \leftarrow \text{diffToUni}(w_{l,lim}, w_{r,lim})$ 
25:   $v \leftarrow v \cdot vlim$ 
26:   $\omega \leftarrow \omega \cdot wlim$ 
27: Senão
28:  Se  $\text{abs}(\omega) > (R/L) \cdot 2 \cdot VEL\_MIN$  Faça
29:    Se  $\omega \geq 0$  Faça
30:       $\omega \leftarrow \max(\min(\text{abs}(\omega), (R/L) \cdot 2 \cdot VEL\_MAX), (R/L) \cdot 2 \cdot VEL\_MIN)$ 
31:    Senão
32:       $\omega \leftarrow -\max(\min(\text{abs}(\omega), (R/L) \cdot 2 \cdot VEL\_MAX), (R/L) \cdot 2 \cdot VEL\_MIN)$ 
33:    Fim Se
34:  Senão
35:     $\omega \leftarrow 0$ 
36:  Fim Se
37: Fim Se
38:  $\omega_l, \omega_r \leftarrow \text{uniToDiff}(v, \omega)$ 

```

---

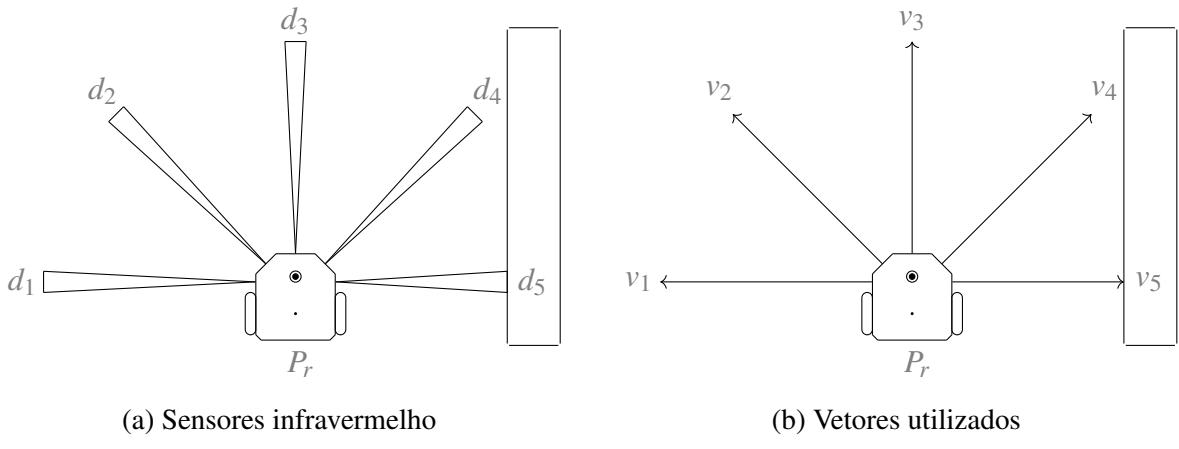
$\phi$  é o ângulo do robô e a função *atan2()* é o arco tangente com dois parâmetros, presente em muitas linguagens de programação, tendo a vantagem de calcular o ângulo sem dúvidas quanto ao quadrante.

$$e_\theta = \text{atan2}(u_{ipo,y}, u_{ipo,x}) - \phi \quad (23)$$

#### 4.1.1.2 COMPORTAMENTO "EVITAR OBSTÁCULO"

O comportamento “Evitar Obstáculo” calcula um vetor cujo objetivo é afastar o robô de barreiras encontradas. A Figura 26.a mostra a disposição dos sensores enquanto a Figura 26.b mostra vetores com origem nas coordenadas dos sensores e cujos módulos são iguais às distâncias medidas.

**Figura 26 – Comportamento Evitar Obstáculo**



**Fonte: autoria própria**

É interessante definir sistemas de coordenadas para cada sensor de modo que a coordenada do obstáculo possa ser representada como um vetor da forma  $[d_i \ 0]^T$ , onde  $d_i$  é a distância detectada pelo i-ésimo sensor. Assim, a representação é extremamente simples e, com matrizes de rotação, pode-se encontrar suas posições no sistemas de coordenadas do robô.

A representação dos obstáculos no sistema de coordenadas do robô está retratada na Equação 24, onde  $\theta_i$  é o ângulo do i-ésimo sensor,  $d_i$  é a distância ao obstáculo (pode estar saturada) e  $[o_x \ o_y]_i^T$  é um vetor na coordenada do robô que define um *offset*, apontando para a origem do sistema de coordenadas do sensor.

$$\begin{bmatrix} x \\ y \end{bmatrix}_i^R = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \\ \sin \theta_i & \cos \theta_i \end{bmatrix} \begin{bmatrix} d_i \\ 0 \end{bmatrix} + \begin{bmatrix} o_x \\ o_y \end{bmatrix}_i^R \quad (24)$$

Os cinco sensores foram posicionados de modo a formar ângulos de  $-\pi/2$ ,  $-\pi/4$ ,

0,  $\pi/4$  e  $\pi/2$ , respectivamente. Os cinco vetores no sistema de coordenadas do robô são combinados linearmente para formar uma única recomendação. A recomendação é dada pela Equação 25, onde os vetores  $v_1$  a  $v_5$  são os mesmos da Figura 26.b, os valores  $k_1$  a  $k_5$  são constantes e o vetor  $v_{eq}$  é um vetor arbitrário cuja escolha será explicada.

$$\mathbf{u}_{eo1} = k_1 \mathbf{v}_1 + k_2 \mathbf{v}_2 + k_3 \mathbf{v}_3 + k_4 \mathbf{v}_4 + k_5 \mathbf{v}_5 + \mathbf{v}_{eq} \quad (25)$$

As constantes  $k_1$  a  $k_5$  valem respectivamente, 0.7, 2, 1.2, 2 e 0.7. Elas foram definidas a partir de simulação. A razão de considerar peso maior para os sensores diagonais ( $-\pi/4$  e  $\pi/4$ ) é causar uma deflexão ao encontrar obstáculos frontais. O peso pequeno nos sensores laterais ( $-\pi/2$  e  $\pi/2$ ) é devido ao baixo risco de colisão, já que o sentido de movimento do robô é perpendicular a estes obstáculos.

Em condição de ausência de obstáculos, os módulos dos vetores  $v_1$  a  $v_5$  estarão saturados em 80 (cálculo dos sensores em centímetros). Neste caso, a soma dos cinco primeiros termos tem como resultante um vetor ao longo do eixo X no sistema de coordenadas do robô. Ao encontrar uma parede posicionada na perpendicular em relação ao sentido de movimento, a resultante continuaria no eixo X (no sistema de coordenadas do robô), provocando colisão. O vetor  $\mathbf{v}_{eq}$  tem por objetivo reduzir a magnitude da resultante. Seu módulo poderia ser definido de modo que a resultante se torne zero a uma distância segura do obstáculo.

A condição de equilíbrio é indesejável, pois o robô não atinge o objetivo, mas este caso é um excelente ponto de partida para compreender o comportamento de evitar obstáculo. A partir do equilíbrio, se for adicionado um obstáculo à frente de qualquer sensor (ou se um obstáculo já existente for aproximado), a magnitude do vetor correspondente diminui, e a resultante será no sentido contrário.

O valor de  $\mathbf{v}_{eq}$  foi definido como  $[-240 \ 0]^T$  (unidade em centímetros), de modo que o robô não para até se chocar com o obstáculo. Portanto, o efeito deste vetor é apenas reduzir a aparente “inércia” e aumentar o efeito de deflexão ao encontrar obstáculos em ângulos oblíquos.

Para aproximações com angulos retos, é necessário um cálculo distinto, que afaste a resultante do eixo x (no sistema de coordenadas do robô). Para este fim, o vetor da Equação 26 é rotacionado. Como este é um caso emergencial, pela iminência de colisão, a recomendação de velocidade linear é zerada, de modo que na etapa de conversão entre modelo uniciclo e modelo de acionamento diferencial, a saída é uma rotação pura, que dura apenas o suficiente para o obstáculo ser percebido como obliqua e o cálculo da Equação 25 voltar a ser adotado.

$$\mathbf{u}_{eo2} = k_1 \mathbf{v}_1 + k_2 \mathbf{v}_2 + k_3 \mathbf{v}_3 + k_4 \mathbf{v}_4 + k_5 \mathbf{v}_5 \quad (26)$$

O cálculo final para a recomendação “Evitar Obstáculo” é dado pela Equação 27, onde  $D_{insegura}$  vale 25 centímetros. Essa recomendação é no sistema de coordenadas do robô e deve ser convertida para o sistema global antes de continuar os cálculos.

$$\mathbf{u}_{eo} = \begin{cases} \mathbf{u}_{eo1} & \text{para } |\mathbf{v}_3| \geq D_{insegura} \\ \begin{bmatrix} \cos(90) & -\sin(90) \\ \sin(90) & \cos(90) \end{bmatrix} \mathbf{u}_{eo2} & \text{para } |\mathbf{v}_3| < D_{insegura} \wedge \\ & k_1 (|\mathbf{v}_5| - |\mathbf{v}_1|) + k_2 (|\mathbf{v}_4| - |\mathbf{v}_2|) > 0 \\ \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} \mathbf{u}_{eo2} & \text{para } |\mathbf{v}_3| < D_{insegura} \wedge \\ & k_1 (|\mathbf{v}_5| - |\mathbf{v}_1|) + k_2 (|\mathbf{v}_4| - |\mathbf{v}_2|) \leq 0 \end{cases} \quad (27)$$

O controlador PID para comportamento “Evitar Obstáculo” é calculado do mesmo modo que no comportamento “Ir Para Objetivo”.

#### 4.1.1.3 COMPORTAMENTO MESCLADO ”IR PARA OBJETIVO E EVITAR OBSTÁCULO”

Situações de transição entre comportamentos podem trazer problemas que foram discutidos na Seção 2.5. Uma forma de mitigar possíveis oscilações é por meio da definição de um novo comportamento em uma região intermediária entre as duas regiões (verificar Figura 8).

Vale salientar que um novo comportamento não resolve o problema, pois passam a existir duas fronteiras ao invés de uma. Contudo, se este novo comportamento for definido como combinação linear entre vetores das regiões vizinhas, as transições se vistas separadamente provocam mudanças mais suaves nas recomendações, o que é capaz de mitigar oscilações.

Essa abordagem é escalável, no sentido de que para cada fronteira, um novo comportamento pode ser criado. A ideia aqui é que, com infinitas iterações, um sistema vetorial descontínuo se tornaria um sistema contínuo.

Neste caso apenas uma iteração foi necessária e o cálculo pode ser visto na Equação 28, onde  $k$  vale 0,3. O vetor  $\hat{\mathbf{u}}_{ipo}$  é o mesmo definido na Subseção 4.1.1.1, porém normalizado e o vetor  $\hat{\mathbf{u}}_{eo}$  é da Equação 25, também normalizado e convertido para o sistema de coordenadas global.

$$\mathbf{u}_{eo.e.ip0} = k \hat{\mathbf{u}}_{ipo} + (1 - k) \hat{\mathbf{u}}_{eo} \quad (28)$$

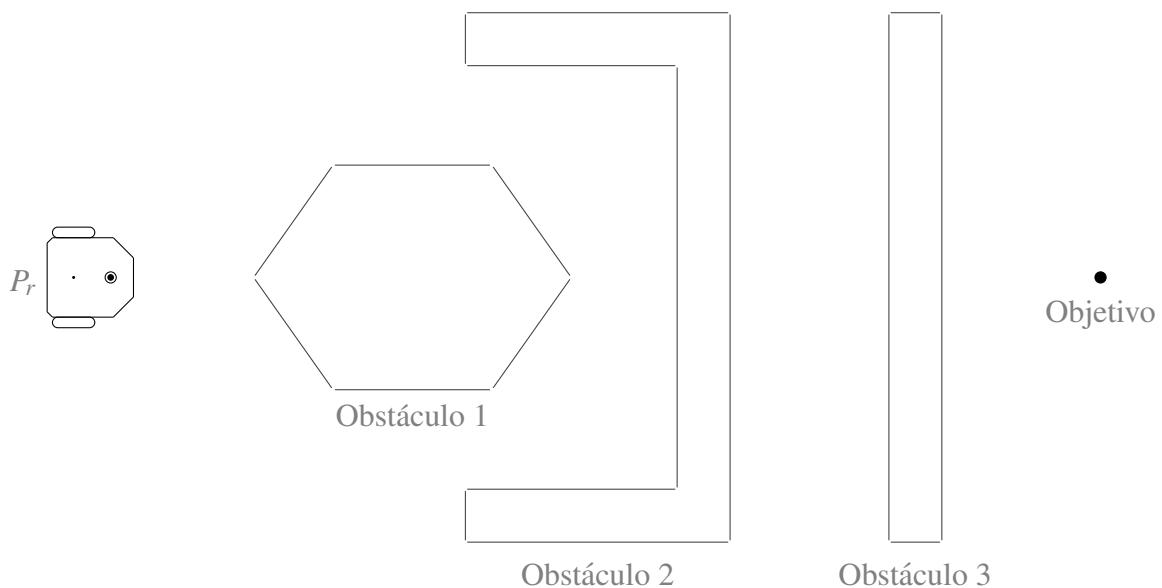
O controlador PID para comportamento “Ir Para Objetivo e Evitar Obstáculo” é calculado do mesmo modo que nos comportamentos apresentados anteriormente.

#### 4.1.1.4 MÍNIMOS LOCAIS

Como a arquitetura comportamental é baseada em campos vetoriais (e ainda descontínuos), mínimos locais, quando presentes, podem fazer o robô nunca alcançar um objetivo.

A figura 28 mostra diferentes tipos de obstáculos, a fim de discutir suas interações com um robô que utilize apenas comportamentos “Ir Para Objetivo”(IPO) e “Evitar Obstáculo”(EO). Um robô que se dirige ao objetivo tem o comportamento IPO ativo e ao encontrar um obstáculo muda o comportamento para EO.

**Figura 27 – Tipos de obstáculos**



**Fonte: autoria própria**

Se encontrar o Obstáculo 1 (convexo), o robô adota o comportamento EO e ocorre uma deflexão em seu movimento. Quando não puder mais detectar obstáculos, o comportamento IPO volta ao controle. O robô é capaz de contornar obstáculos desde que a deflexão não leve o robô para uma posição em que ele já esteve.

O Obstáculo 2, por ser côncavo, ilustra melhor a situação de mínimo local. A deflexão provocada pelo comportamento EO evita o obstáculo, mas visita uma região já conhecida. Portanto o robô se aproxima e se afasta do obstáculo indefinidamente, vagando pela região de mínimo local.

O Obstáculo 3 é um exemplo convexo com mínimo local. Na ausência de obstáculos, o vetor do comportamento IPO diminui em magnitude à medida que o robô avança. Se este comportamento leva a uma posição em que um obstáculo é simétrico em relação à linha entre

Robô e Objetivo, qualquer deflexão causada pelo comportamento EO é seguida de uma nova aproximação, pois o vetor IPO terá um componente no sentido no eixo de simetria.

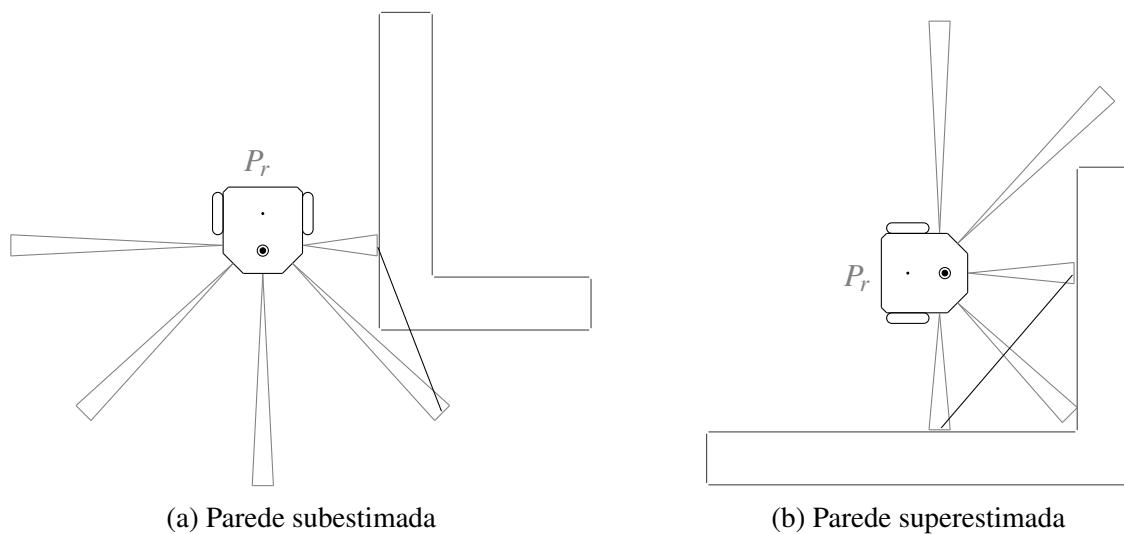
É necessário introduzir um comportamento capaz de afastar o robô de um mínimo local detectado e este comportamento deve interagir com os anteriores de modo a não criar novos mínimos locais. Neste caso, foi criado o comportamento “Seguir Parede”.

#### 4.1.1.5 COMPORTAMENTO ”SEGUIR PAREDE”

O comportamento “Seguir Parede” tem por intuito contornar obstáculos até retirar o robô de uma região de mínimo local. A detecção de que o robô entrou em um mínimo local ou saiu dele é papel do autômato e será discutido em outra seção. Aqui, parte-se do princípio de que o robô com certeza está próximo a uma parede, na iminência de retornar a uma região já visitada. A escolha de contornar no sentido horário ou anti-horário também fica a cargo do autômato.

A Figura 28 mostra dois casos de interesse para compreender o comportamento “Seguir Parede”. Como pode ser verificado, a detecção da parede utiliza sempre dois sensores, pois com dois pontos, forma-se uma reta. Contudo, a escolha é feita entre três sensores. Quando é decidido contornar obstáculo no sentido anti-horário, os três sensores escolhidos são do sensor frontal ao sensor esquerdo e, para o sentido horário de contorno, os sensores vão do sensor frontal ao sensor direito.

**Figura 28 – Comportamento Seguir Parede**



**Fonte: autoria própria**

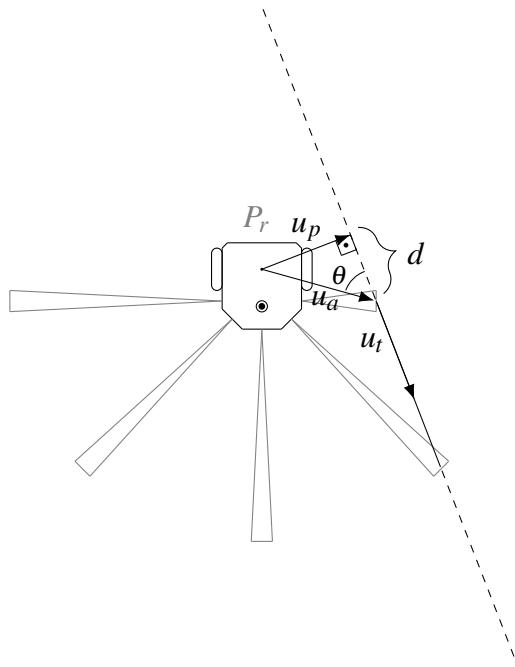
Uma vez definidos os três sensores de interesse, a reta que descreve a parede é

calculada utilizando os dois sensores com menor valor de distância e, caso tenha empate, a prioridade de escolha vai do sensor lateral ao frontal (lateral com a maior prioridade, frontal com a menor).

É necessário mencionar que a simplificação da parede como reta pode subestimar ou superestimar a parede real. A Figura 28.a é um caso de subestimativa, já que o segmento de reta secciona a parede real. A Figura 28.b mostra uma superestimativa, pois o segmento de reta está muito distante da parede real. A superestimativa não é um problema grave, pois o robô não corre risco de colidir. Subestimar a parede, por outro lado, leva a uma aproximação perigosa. Para evitar colisão, é necessário definir uma distância segura para seguir parede. Com uma distância suficiente, a aproximação não traz risco de colisão.

Na etapa seguinte do cálculo, parte-se de uma reta que descreve a parede e calcula-se um vetor a ser seguido. A Figura 29 ilustra essa etapa.

**Figura 29 – Recomendação vetorial**



**Fonte: autoria própria**

O vetor  $\mathbf{u}_t$  é um vetor normalizado que está ao longo da reta.  $\mathbf{u}_a$  é um vetor conhecido, centrado na origem do sistema de coordenadas do robô e que leva ao ponto associado a um dos sensores utilizado para o cálculo da reta.  $\mathbf{u}_p$  é perpendicular à reta e pode ser calculado a partir da Equação 29, onde  $d$  é a distância mostrada na figura.

$$\mathbf{u}_p = \mathbf{u}_a - d \cdot \mathbf{u}_t \quad (29)$$

O cálculo da distância  $d$  é feito por produto escalar, A Equação 30 mostra essa operação, simplificada na Equação 31.

$$\mathbf{u}_a \cdot \mathbf{u}_t = |\mathbf{u}_a| \cdot |\mathbf{u}_t| \cos(\theta) \quad (30)$$

$$\mathbf{u}_a \cdot \mathbf{u}_t = |\mathbf{u}_a| \cos(\theta) = d \quad (31)$$

A partir destes vetores, o vetor recomendação para o comportamento é dado pela Equação 32, onde  $\beta$  é uma constante e  $d_{fw}$  é a distância segura já citada anteriormente.

$$\mathbf{u}_{fw} = \mathbf{u}_t + \beta \cdot \left( \mathbf{u}_p - d_{fw} \cdot \frac{\mathbf{u}_p}{|\mathbf{u}_p|} \right) \quad (32)$$

O termo  $(\mathbf{u}_p - d_{fw} \cdot \mathbf{u}_p / |\mathbf{u}_p|)$  é um vetor perpendicular que aponta para uma linha paralela à parede, a uma distância  $d_{fw}$ . Assim, se a distância entre robô e parede for diferente de  $d_{fw}$ , essa componente será diferente de zero e seu efeito será no intuito de reduzir a diferença. A constante  $\beta$  serve para definir a importância do vetor perpendicular, já que  $u_t$  é um vetor normalizado. Após simulação, o valor  $\beta$  foi definido como 5,5.  $d_{fw}$  foi definido como sendo 50 centímetros.

O controlador para o comportamento “Seguir Parede” é apenas um controlador proporcional, pois o vetor referência muda constantemente. O termo derivativo, se utilizado de forma inadequada (sem filtro, ou com valores altos para  $k_d$ ), traz oscilações indesejadas. O termo integrador é lento para entradas que variam muito. Uma constante  $K_p = 2$  foi utilizada, escolhida empiricamente, em simulação.

#### 4.1.1.6 O AUTÔMATO PARA ARBITRAGEM DE COMPORTAMENTOS

Para o funcionamento do autômato, um último comportamento foi adicionado. O comportamento “Parar” apenas retira acionamento dos motores e deve assumir controle ao atingir o objetivo.

A Tabela 2 mostra os comportamentos existentes. Os índices são os estados do autômato.

As funções de transição são formadas por condições, que foram resumidas na Tabela 3. A tabela traz equações ou algoritmos para cálculo de cada condição.

A condição “no objetivo” verifica se o objeto chegou ao objetivo. Seu cálculo pode ser visto na Equação 33, onde  $\mathbf{v}_{objetivo}$  é o vetor que aponta para o objetivo,  $\mathbf{v}_{robô}$  é o vetor que aponta para a coordenada do robô e  $D_{STOP}$  é uma distância de tolerância, que foi definida como

**Tabela 2 – Estados do autômato**

Índice	Comportamento	Descrição
0	P	Parar
1	IPO	Ir para Objetivo
2	EO	Evitar Obstáculos
3	IPO E EO	Ir para Objetivo e Evitar Obstáculos (combinados)
4	SP AH	Seguir Parede sentido Anti-horário
5	SP H	Seguir Parede sentido Horário

**Tabela 3 – Condições utilizadas nas transições do autômato**

Legenda	Condições	Cálculo da condição
a	No objetivo	Equacao 33
b	Fez progresso	Algoritmo 2
c	Tem Obstáculo	Equacao 34
d	Está inseguro	Equacao 35
e	Livre de obstáculo	Equacao 36
f	Contornando pela esquerda	Algoritmo 3
g	Contornando pela direita	Algoritmo 3

15 centímetros. Assim, atingir o objetivo é alcançar uma região circular de raio  $D_{STOP}$ .

$$|\mathbf{v}_{\text{objetivo}} - \mathbf{v}_{\text{robô}}| < D_{STOP} \quad (33)$$

A condição “fez progresso” tem por intuito detectar se o robô está avançando em direção ao objetivo. Essa condição detecta entrada e saída de mínimos locais, já que, parar de fazer progresso significa que o robô está retornando a região já visitada. Quando voltar a fazer progresso, significa que saiu de mínimo local. O Algoritmo 2 mostra esse cálculo. A variável  $d_{prog}$  é a última distância verificada e é atualizada no mesmo algoritmo. No início da navegação ou quando o objetivo for alterado,  $d_{prog}$  deve ser inicializado para um valor infinito (no robô, foi usado valor de 1000 metros, já que nunca será requisitado ao robô percorrer tal distância). A constante  $D_{PROG\_EPSILON}$  foi definida como 2 centímetros e seu intuito é não permitir que distâncias consideradas infinitesimais sejam consideradas como progresso.

A condição “tem obstáculo” detecta se pelo menos algum sensor percebe existência de obstáculo. A Equação 34 mostra a função booleana, onde “any” é o quantificador lógico “algum” que verifica se algum item no vetor  $\mathbf{d}_f$  (que especifica distância detectada pelos cinco sensores) cumpre a condição especificada, de ser menor que a constante  $D_{EM\_OBSTÁCULO}$ , definida como 75 centímetros.

$$\text{any}(\mathbf{d}_s < D_{EM\_OBSTÁCULO}) \quad (34)$$

---

**Algoritmo 2** Verificação de progresso
 

---

**Entrada:**  $d_{prog}, \mathbf{v}_{objetivo}, \mathbf{v}_{robô}$

**Saída:**  $d_{prog}, retornoBooleano$

- 1: **Se**  $|\mathbf{v}_{objetivo} - \mathbf{v}_{robô}| < (d_{prog} - D_{PROG\_EPSILON})$  **Faça**
  - 2:    $d_{prog} \leftarrow \min(|\mathbf{v}_{objetivo} - \mathbf{v}_{robô}|, d_{prog})$
  - 3:    $retornoBooleano \leftarrow Verdadeiro$
  - 4: **Senão Se**  $\text{abs}(|\mathbf{v}_{objetivo} - \mathbf{v}_{robô}| - d_{prog}) \leq D_{PROG\_EPSILON}$  **Faça**
  - 5:    $retornoBooleano \leftarrow Verdadeiro$
  - 6: **Senão**
  - 7:    $retornoBooleano \leftarrow Falso$
  - 8: **Fim Se**
- 

A condição “está inseguro” detecta se algum sensor percebe a existência de obstáculo em proximidade preocupante. A Equação 35 mostra a função booleana. Neste caso, é verificado se algum item do vetor  $\mathbf{d}_f$  é menor que uma constante  $D_{INSEGURUO}$ , definida como 25 centímetros.

$$\text{any}(\mathbf{d}_s < D_{INSEGURUO}) \quad (35)$$

A condição “livre de obstáculo” verifica se todos os sensores estão saturados, ou seja, sem obstáculos detectados. A Equação 36 mostra a função booleana, onde “all” é o quantificador lógico “todos” que verifica se todos os itens do vetor  $\mathbf{d}_f$  cumprem a condição especificada no argumento (maior que  $D_{EM\_OBSTÁCULO}$ ). É importante salientar que “livre de obstáculos” não é a negação de “tem obstáculo”, apesar dos nomes.

$$\text{all}(\mathbf{d}_s > D_{EM\_OBSTÁCULO}) \quad (36)$$

As condições “contornando pela esquerda” e “contornando pela direita” têm por intuito detectar se o robô está seguindo uma fronteira entre dois comportamentos em sentido horário ou anti-horário, a fim de decidir o sentido em que adotará comportamento “Seguir Parede”. Essa verificação no autômato ocorre junto com a condição “não fez progresso”, de modo que, neste caso, os comportamentos “Ir Para Objetivo” e “Evitar Obstáculo” estão em conflito (ângulo obtuso).

É desejável que o comportamento “Seguir Parede” escolhido esteja entre os dois (similar ao que foi retratado na Figura 8). Então, por álgebra linear, deseja-se um vetor  $\mathbf{u}_{sp} = \sigma_1 \cdot \mathbf{u}_{ipo} + \sigma_2 \cdot \mathbf{u}_{eo}$  com constantes  $\sigma_1$  e  $\sigma_2$  maiores que zero. As Equações 37 e 38 ilustram o cálculo dos valores sigma.

$$[\mathbf{u}_{ipo} \ \mathbf{u}_{eo}] \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = \mathbf{u}_{sp} \quad (37)$$

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = [\mathbf{u}_{\text{ipo}} \ \mathbf{u}_{\text{eo}}]^{-1} \cdot \mathbf{u}_{\text{sp}} \quad (38)$$

O algoritmo 3 calcula essas condições. A entrada “sentidoDeContorno” define se o teste é para a condição “contornando pela esquerda” ou “contornando pela direita”. As funções “vetorIrParaObjetivo()”, “vetorEvitarObstaculo()” e “vetorSeguirParede()” calculam os vetores para os comportamentos “Ir Para Objetivo”, “Evitar Obstáculo” e “Seguir Parede”, respectivamente. O último precisa do argumento “sentidoDeContorno” para ser calculado. A função “obstaculoPresente()” verifica apenas sensor lateral e diagonal (esquerdo ou direito dependendo do argumento) e retorna verdadeiro se algum deles não estiver saturado.

---

### **Algoritmo 3** Verificação de situação de deslize em fronteira

---

**Entrada:** *sentidoDeContorno*

**Saída:** *retornoBooleano*

- 1:  $u_{\text{ipo},x}, u_{\text{ipo},y} \leftarrow \text{vetorIrParaObjetivo}()$
  - 2:  $u_{\text{eo},x}, u_{\text{eo},y} \leftarrow \text{vetorEvitarObstaculo}()$
  - 3:  $u_{\text{sp},x}, u_{\text{sp},y} \leftarrow \text{vetorSeguirParede}(\text{sentidoDeContorno})$
  - 4:  $\text{determinante} \leftarrow u_{\text{ipo},x}u_{\text{ao},y} - u_{\text{ipo},y}u_{\text{ao},x}$
  - 5:  $\sigma_1 \leftarrow (u_{\text{ao},y}u_{\text{sp},x} - u_{\text{ao},x}u_{\text{sp},y})/\text{determinante}$
  - 6:  $\sigma_2 \leftarrow (-u_{\text{ipo},y}u_{\text{sp},x} + u_{\text{ipo},x}u_{\text{sp},y})/\text{determinante}$
  - 7: **Se** *obstaculoPresente(sentidoDeContorno)* E  $\sigma_1 > 0$  E  $\sigma_2 > 0$  **Faça**
  - 8:    *retornoBooleano*  $\leftarrow$  Verdadeiro
  - 9: **Senão**
  - 10:    *retornoBooleano*  $\leftarrow$  Falso
  - 11: **Fim Se**
- 

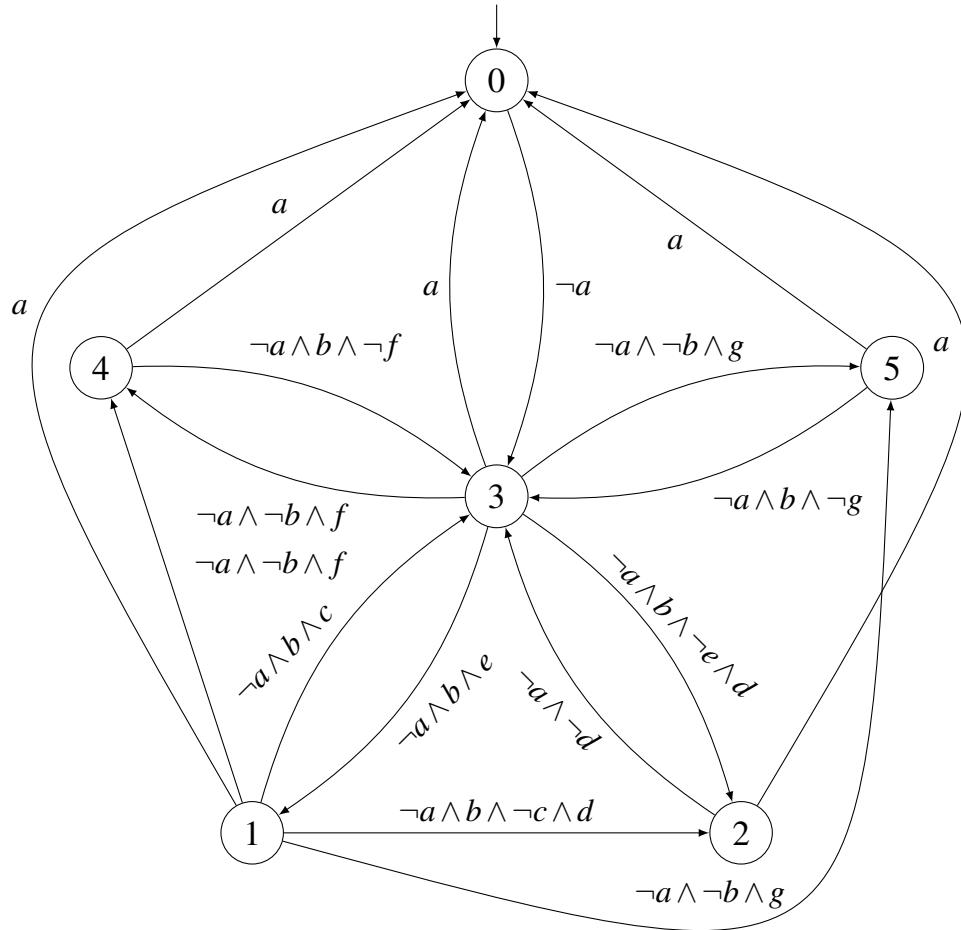
A Figura 30 exibe o autômato completo. Suas transições serão explicadas detalhadamente.

De todos os estados é possível chegar ao estado inicial “Parar” (0) por meio da condição “No Objetivo” (a), mas para sair dele, uma redefinição de objetivo provoca a negação da condição de parada ( $\neg a$ ) e o estado mesclado “Ir para Objetivo e Evitar Obstáculo” (3) assume controle.

A partir desse último, as transições para “Seguir Parede” (4 e 5) dependem da condição “Não Fez Progresso” ( $\neg b$ ), mas a escolha do sentido de contorno depende das condições “contornando pela esquerda” e “contornando pela direita” (f e g). A partir dos estados 4 ou 5, o retorno para o estado 3 depende de não ter encontrado objetivo ( $\neg a$ ), voltar a fazer progresso (b) e deixar a situação de seguir fronteira ( $\neg f$  ou  $\neg g$ ).

O estado 3 pode alcançar “Ir Para Objetivo” (1) quando está “livre de obstáculo” (e) e retorna quando volta a “ter obstáculo”(c). O estado 1 alcança “Seguir Parede” com as mesmas

**Figura 30 – Autômato Híbrido que soluciona o problema de navegação**



**Fonte: autoria própria**

condições utilizadas a partir do estado 3.

Os estados 1 e 3 podem alcançar “Evitar Obstáculo” (2) por meio da condição “está inseguro” ( $d$ ), mas uma vez neste estado, só pode retornar ao estado 3, com condição de “não estar inseguro” ( $\neg d$ ).

#### 4.1.2 CONTROLE FUZZY

Nesta seção, pretende-se demonstrar um controlador simples capaz de solucionar o problema de navegação, usando sistema *fuzzy*.

Já que intuito é implementar fusão de comportamentos, cada comportamento adicionado deve ser projetado pensando em sua interação com todos os já existentes, o que dificulta o projeto, pois, se existirem recomendações conflitantes com o novo comportamento, este deve ser capaz de suprimi-las. Isso é uma limitação de arquitetura e não pode ser

confundido com uma limitação do sistema *fuzzy*, já que ele poderia implementar a arquitetura de arbitragem também.

O controlador *fuzzy* foi definido de modo a controlar o modelo de acionamento diferencial, ao invés do modelo uniciclo. O algoritmo para priorizar velocidade angular sobre velocidade linear foi mantido, bem como a abordagem de representar comportamentos por vetores, de modo que “fundir” comportamentos é apenas combiná-los linearmente. Todos os comportamentos discutidos têm seus cálculos realizados no sistema de coordenadas do robô. O sistema global só é necessário para cálculo de odometria.

#### 4.1.2.1 COMPORTAMENTO ”IR PARA OBJETIVO”

O comportamento “Ir Para Objetivo” tem o mesmo intuito da contraparte híbrida, porém, como as saídas vetoriais são todas normalizadas, o cálculo de “IPO” é normalizado.

Esse cálculo não necessita de um controlador *fuzzy* para gerar recomendações. Como as coordenadas do objetivo e robô são conhecidas, o cálculo é direto. A Equação 39 mostra a normalização. A variável  $e_\theta$  é calculada pela Equação 23 e o vetor  $\mathbf{u}_{\text{ipo}}$  é da Equação 22.

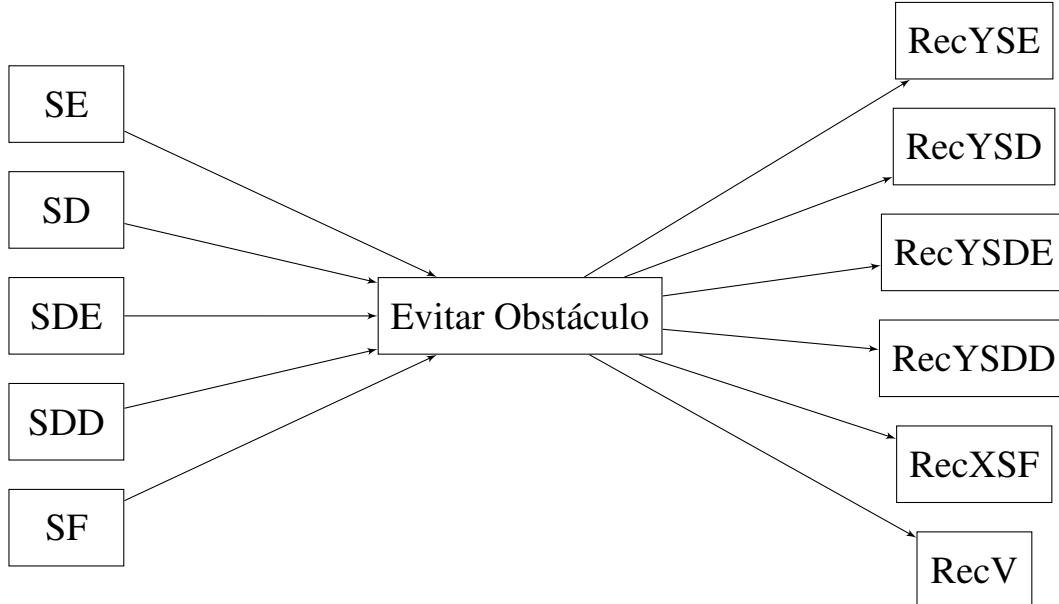
$$\mathbf{u}_{\text{ipo},\text{fuzzy}} = \begin{cases} \begin{bmatrix} \cos(e_\theta) \\ \sin(e_\theta) \end{bmatrix} & , \text{ para } |\mathbf{u}_{\text{ipo}}| \geq 1 \\ |\mathbf{u}_{\text{ipo}}| \cdot \begin{bmatrix} \cos(e_\theta) \\ \sin(e_\theta) \end{bmatrix} & , \text{ para } |\mathbf{u}_{\text{ipo}}| < 1 \end{cases} \quad (39)$$

#### 4.1.2.2 COMPORTAMENTO ”EVITAR OBSTÁCULO”

O comportamento “Evitar Obstáculo” é semelhante à contraparte híbrida. O diagrama do controlador *fuzzy*, relacionando entradas e saídas, pode ser visto na Figura 31, onde as entradas “SE”, “SD”, “SDE”, “SDD” e “SF” são valores para “Sensor Esquerdo”, “Sensor Direito”, “Sensor Diagonal Esquerdo”, “Sensor Diagonal Direito” e “Sensor Frontal”, respectivamente. As saídas “RecYSE”, “RecYSD”, “RecYSDE”, “RecYSDD”, “RecXSF” são recomendações vetoriais, onde X ou Y especifica o eixo e os sufixos “SE”, “SD”, “SDE”, “SDD” e “SF” indicam o sensor responsável pela recomendação. “RecV” é a recomendação para “Velocidade linear”.

A partir das saídas do sistema *fuzzy*, o vetor intermediário da Equação 40 é calculado

**Figura 31 – Diagrama do sistema Fuzzy “Evitar Obstáculo”**



**Fonte:** autoria própria

e o cálculo final para o Comportamento “Evitar Obstáculo” é dado pela Equação 41.

$$\mathbf{u}_{eo,temp} = \begin{bmatrix} 2 \cdot RecXSF \\ RecYSE + RecYSD + RecYSDE + RecYSDD \end{bmatrix} \quad (40)$$

$$\mathbf{u}_{eo,fuzzy} = \begin{cases} \mathbf{u}_{eo,temp} & , \text{ para } |\mathbf{u}_{eo,temp}| \leq 1 \\ \frac{\mathbf{u}_{eo,temp}}{|\mathbf{u}_{eo,temp}|} & , \text{ para } |\mathbf{u}_{eo,temp}| > 1 \end{cases} \quad (41)$$

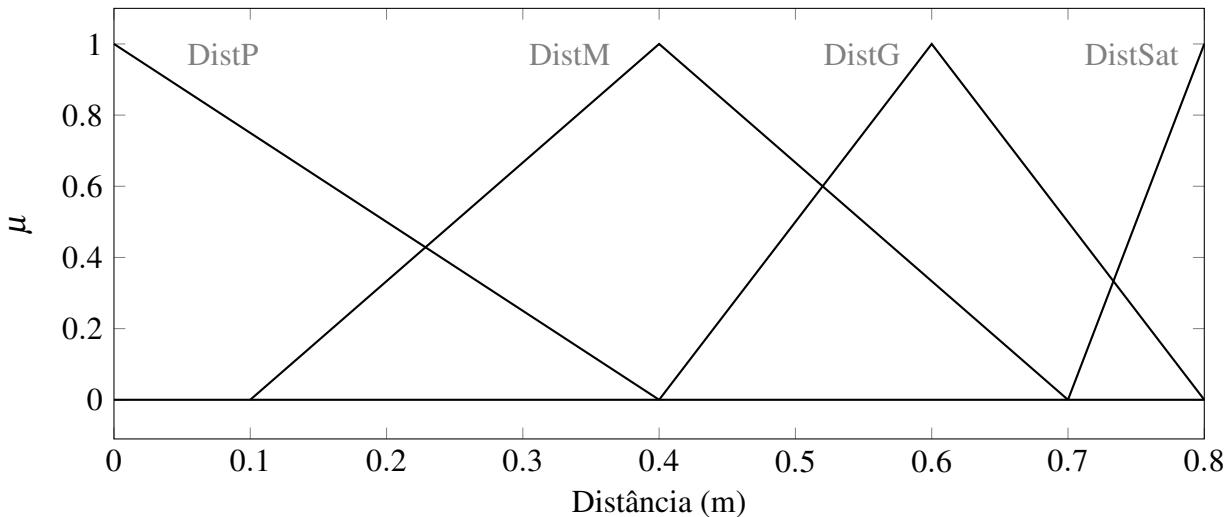
Os conjuntos *fuzzy* utilizados para as entradas dos sensores estão retratados na Figura 32, com as respectivas funções de pertinência.

Já os conjuntos *fuzzy* para saídas dos vetores associados a cada sensor, bem como funções de pertinência, estão retratados na Figura 33. As saídas dos controladores *fuzzy* estão sempre normalizadas.

Os conjuntos para a saída da recomendação em velocidade (“RecV”), bem como funções de pertinência, estão retratados na Figura 34.

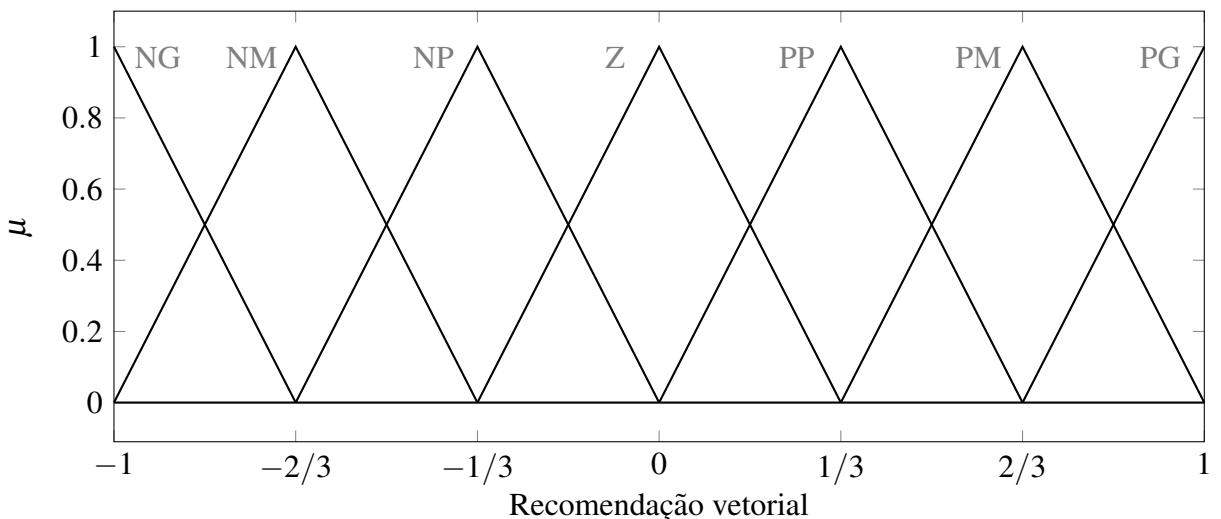
A Tabela 4 mostra um conjunto de regras utilizado. Apesar de ser um sistema MIMO (múltiplas entradas e multiplas saídas), poderia ser dividido em seis sistemas com apenas uma saída, cada qual possuindo um menor conjunto de regras. Dado um sensor específico, quando a distância mensurada é pequena, o vetor associado tem grande magnitude e à medida que a distância tende à saturação, a magnitude do vetor associado tende a zero.

**Figura 32 – Conjuntos Fuzzy para as entradas dos sensores**



**Fonte: autoria própria**

**Figura 33 – Conjuntos Fuzzy para as saídas vetoriais**

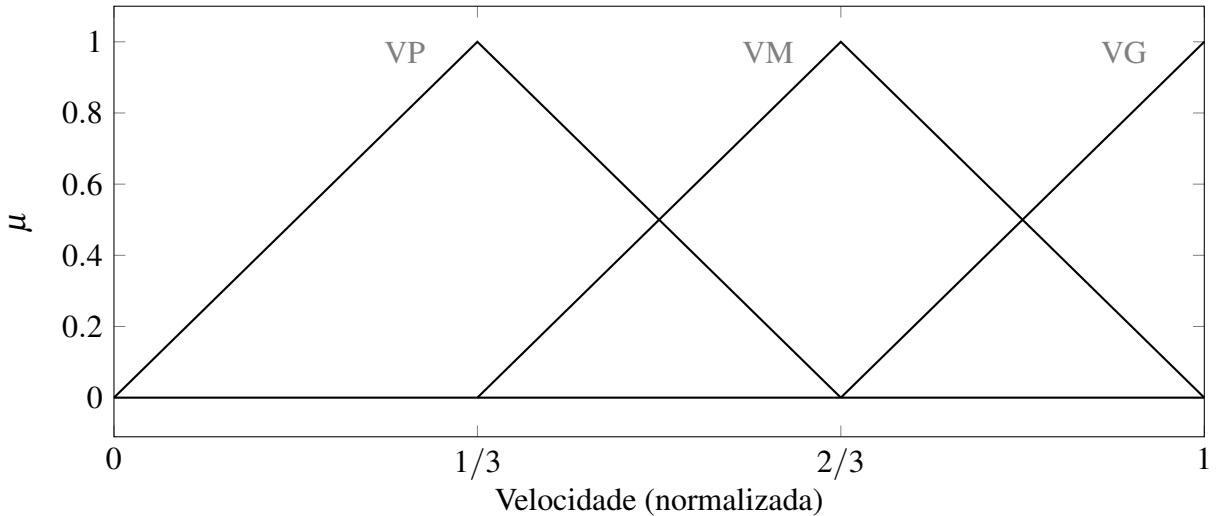


**Fonte: autoria própria**

Cada sensor pode ter no máximo dois “valores ativos” (pertinência maior que zero). Assim, para cinco entradas, pela tabela, teremos no máximo 10 regras ativas.

#### 4.1.2.3 COMPORTAMENTO ”SEGUIR PAREDE”

O comportamento “Seguir Parede” também é necessário no controlador *fuzzy* pela questão dos mínimos locais, discutida anteriormente. Este deve assumir controle do robô apenas quando necessário, suprimindo a tendência de retornar ao mínimo local. A necessidade de uma hierarquia entre comportamentos motiva o uso da arquitetura de subsunção, discutida brevemente na seção 1.1.1.

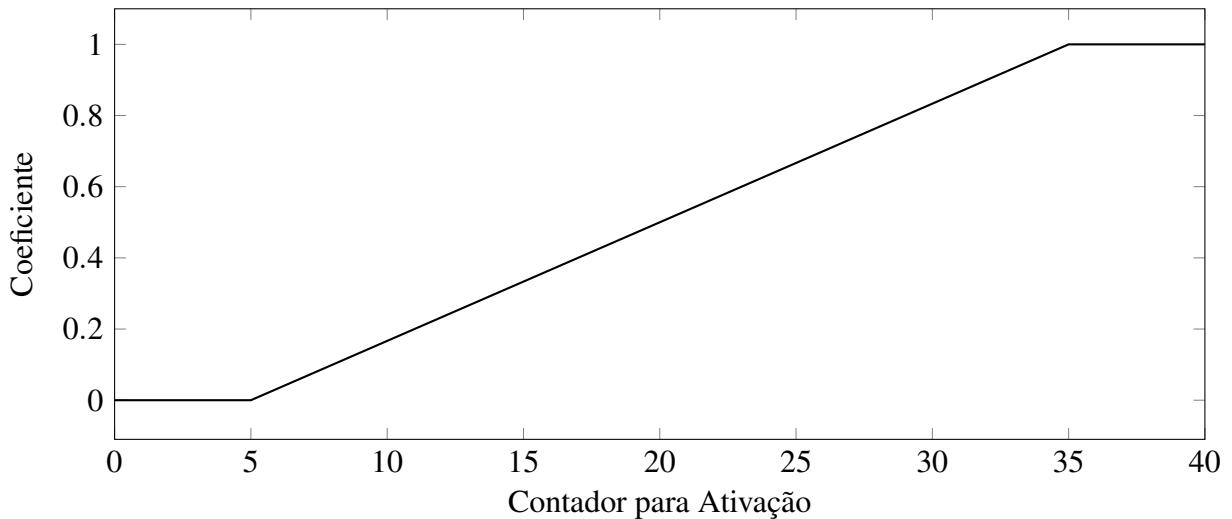
**Figura 34 – Conjuntos Fuzzy para a saída de velocidade****Fonte: autoria própria****Tabela 4 – Regras Fuzzy para sistema “Evitar Obstáculo”**

	Entradas					Saídas					
	SE	SD	SDE	SDD	SF	RecYSE	RecYSD	RecYSDE	RecYSDD	RecXSF	RecV
1	DistP	-	-	-	-	NG	-	-	-	-	-
2	DistM	-	-	-	-	NM	-	-	-	-	-
3	DistG	-	-	-	-	NP	-	-	-	-	-
4	DistSat	-	-	-	-	Z	-	-	-	-	VG
5	-	DistP	-	-	-	-	PG	-	-	-	-
6	-	DistM	-	-	-	-	PM	-	-	-	-
7	-	DistG	-	-	-	-	PP	-	-	-	-
8	-	DistSat	-	-	-	-	Z	-	-	-	VG
9	-	-	DistP	-	-	-	-	NG	-	-	VP
10	-	-	DistM	-	-	-	-	NM	-	-	VM
11	-	-	DistG	-	-	-	-	NP	-	-	VM
12	-	-	DistSat	-	-	-	-	Z	-	-	VG
13	-	-	-	DistP	-	-	-	-	PG	-	VP
14	-	-	-	DistM	-	-	-	-	PM	-	VM
15	-	-	-	DistG	-	-	-	-	PP	-	VM
16	-	-	-	DistSat	-	-	-	-	Z	-	VG
17	-	-	-	-	DistP	-	-	-	-	NG	VP
18	-	-	-	-	DistM	-	-	-	-	NM	VM
19	-	-	-	-	DistG	-	-	-	-	NP	VM
20	-	-	-	-	DistSat	-	-	-	-	Z	VG

A Figura 35 mostra uma maneira simples de inibir comportamentos. A função mostrada é responsável por definir o quanto ativo o comportamento “Seguir Parede” está. Saída 0 significa totalmente inativo e saída 1 significa totalmente ativo. Uma combinação linear pode mesclar recomendações ativando ou desativando comportamentos, dependendo do valor do coeficiente.

A figura 36 mostra o diagrama do controlador, que associa entradas e saídas. As entradas são as mesmas para o controlador “Evitar Obstáculo”. As saídas “SPRecX”,

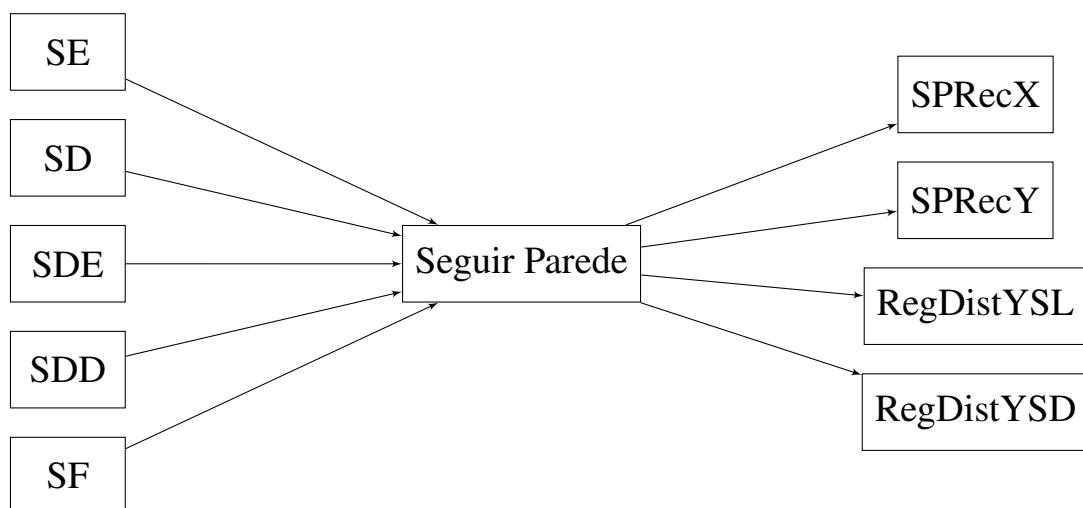
**Figura 35 – Curva de Ativação para comportamento Seguir Parede**



**Fonte: autoria própria**

“SPRecY”, “RegDistYSL” e “RegDistYSD” são recomendações vetoriais. “SPRecX” é uma recomendação no eixo x e “SPRecY” é recomendação ao longo de y. “RegDistYSL” e “RegDistYSD” tem por objetivo recomendar uma deflexão no eixo y, para que o robô siga a parede a uma distância segura. O sufixo “SL” designa que “Sensores Laterais” são responsáveis pela recomendação, enquanto “SD” responsabiliza os “Sensores Diagonais”. A distinção foi necessária para considerar pesos diferentes para sensores diagonais e laterais sem afetar outras saídas.

**Figura 36 – Diagrama do sistema Fuzzy “Seguir Parede”**



**Fonte: autoria própria**

A Equação 42 mostra o cálculo de um vetor temporário, definido a partir das saídas do sistema *fuzzy*. A constante “C” é o coeficiente de ativação e  $u_{eo,fuzzy}$  é a recomendação para “Evitar Obstáculos”, que é somada de modo que mesmo que o comportamento “SP” esteja

totalmente ativo, o robô ainda é capaz de evitar obstáculos.

$$\mathbf{u}_{sp,temp} = \begin{cases} \begin{bmatrix} 3 \cdot SPRecX \\ RegDistYSL + 2,5 \cdot RegDistYSD + 2 \cdot SPRecY \\ 0 \\ 0 \end{bmatrix} + \mathbf{u}_{eo,fuzzy} & , \text{ para } C > 0 \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & , \text{ para } C = 0 \end{cases} \quad (42)$$

A recomendação *fuzzy* para “Seguir Parede” (vetor  $\mathbf{u}_{sp,fuzzy}$ ) é calculado pelo Algoritmo 4. Cada função será discutida com detalhe.

---

**Algoritmo 4** Cálculo Final da Recomendação “Seguir Parede”

---

**Entrada:**  $t$

**Saída:**  $\mathbf{u}_{sp,fuzzy,x}, \mathbf{u}_{sp,fuzzy,y}$

- 1:  $C \leftarrow verificaAtivacaoSP()$
  - 2:  $u_{sp,temp,x}, u_{sp,temp,y} \leftarrow calculaSPFuzzy()$
  - 3: **Se**  $C > 0$  **Faça**
  - 4:    *definirSentidoDeContorno()*
  - 5:    *verificarPerdaDeReferencia()*
  - 6:    *normalizarEntradasVetor()*
  - 7: **Senão**
  - 8:     $u_{sp,fuzzy,x} \leftarrow u_{sp,temp,x}$
  - 9:     $u_{sp,fuzzy,y} \leftarrow u_{sp,temp,y}$
  - 10: **Fim Se**
- 

A função “verificaAtivacaoSP()” é responsável por calcular a constante de ativação e pode ser vista no Algoritmo 5. A função “fezProgresso()” é a mesma verificação utilizada no Algoritmo 2. A variável “ativacaoSP” é o contador, variável x no gráfico da Figura 35, que implicitamente mostra as constantes “ATIVACAO\_MARGEM”, definida como 5 e “ATIVACAO\_PASSOS”, estipulada como 40.

A função “calculaSPFuzzy()” calcula as saídas para o controlador *fuzzy* “Seguir Parede” e realiza o cálculo da Equação 42.

Se por algum motivo o robô perder a referência da parede (sensores saturados), o vetor calculado pela função anterior terá magnitude zero. Para evitar parada, as funções “definirSentidoDeContorno()” (Algoritmo 6) e “verificarPerdaDeReferencia()” (Algoritmo 7) foram criadas. O intuito da primeira é detectar o sentido de contorno (parede à esquerda ou à direita). Na segunda, se necessário, uma componente é atribuída ao vetor com o intuito de aproximar o robô da parede que perdeu.

Nas funções, “SpDir” especifica o sentido de contorno e a princípio, é inicializado

---

**Algoritmo 5** Verificar constante de ativação
 

---

**Entrada:**  $ativacaoSP$

**Saída:**  $ativacaoSP, SPDdir, C$

- 1: **Se**  $fezProgresso()$  **Faça**
  - 2:   **Se**  $ativacaoSP > 0$  **Faça**
  - 3:      $ativacaoSP \leftarrow ativacaoSP - 1$
  - 4:   **Fim Se**
  - 5:   **Se**  $ativacaoSP < ATIVACAO\_MARGEM$  **Faça**
  - 6:      $SPDdir \leftarrow 0$
  - 7:   **Fim Se**
  - 8: **Senão**
  - 9:   **Se**  $ativacaoSP < ATIVACAO\_PASSOS$  **Faça**
  - 10:      $ativacaoSP \leftarrow ativacaoSP + 1$
  - 11:   **Fim Se**
  - 12: **Fim Se**
  - 13:  $C \leftarrow \frac{\min(\max(ativacaoSP, ATIVACAO\_MARGEM), ATIVACAO\_PASSOS - ATIVACAO\_MARGEM)}{(ATIVACAO\_PASSOS - 2 * ATIVACAO\_MARGEM)}$
- 

---

**Algoritmo 6** Verificação do sentido de contorno
 

---

**Entrada:**  $SpDir, RegDistYSL, RegDistYSD$

**Saída:**  $SpDir$

- 1: **Se**  $SpDir = 0$  **Faça**
  - 2:   **Se**  $RegDistYSL + 2,5 \cdot RegDistYSD > \varepsilon$  **Faça**
  - 3:      $SpDir \leftarrow 1$
  - 4:   **Senão Se**  $RegDistYSL + 2,5 \cdot RegDistYSD < -\varepsilon$  **Faça**
  - 5:      $SpDir \leftarrow -1$
  - 6: **Senão**
  - 7:      $SpDir \leftarrow SpDir$
  - 8: **Fim Se**
  - 9: **Fim Se**
- 

---

**Algoritmo 7** Verificar Perda de Referência
 

---

**Entrada:**  $SpDir, u_{sp,fuzzy,x}, u_{sp,fuzzy,y}$

**Saída:**  $u_{sp,fuzzy,y}$

- 1: **Se**  $\sqrt{u_{sp,fuzzy,x}^2 + u_{sp,fuzzy,y}^2} < 0,05$  **Faça**
  - 2:   **Se**  $SpDir = 1$  **Faça**
  - 3:      $u_{sp,fuzzy,y} \leftarrow 0,3$
  - 4:   **Senão Se**  $SpDir = -1$  **Faça**
  - 5:      $u_{sp,fuzzy,y} \leftarrow -0,3$
  - 6: **Senão**
  - 7:      $u_{sp,fuzzy,y} \leftarrow u_{sp,fuzzy,y}$
  - 8: **Fim Se**
  - 9: **Fim Se**
-

como zero, que indica que o robô não está contornando obstáculo. Se 1, está contornando em sentido anti-horário (parede à esquerda) e se -1, sentido horário de contorno (parede à direita). O valor de  $\epsilon$  foi definido como 0,0001.

A função “normalizarEntradasVetor()” tem por objetivo manter os valores das componentes x e y do vetor  $u_{sp,fuzzy}$  no intervalo [-1, 1]. O nome “normalizar” na função, portanto, não é no sentido de normalizar o vetor em si, mas sim, normalizar as componentes. Isso foi feito dessa forma pois as entradas do sistema fuzzy para “Seguir Recomendação” devem ser normalizadas. Essa “normalização” é calculada pelo Algoritmo 8.

---

#### Algoritmo 8 Normalização de Componentes do vetor

---

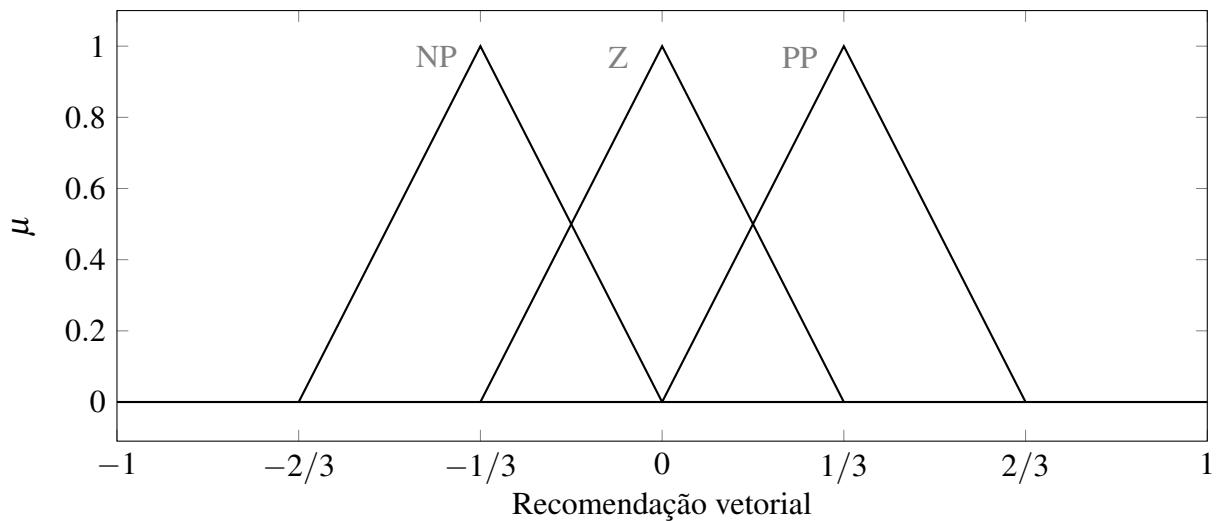
**Entrada:**  $u_{sp,fuzzy,x}, u_{sp,fuzzy,y}$

**Saída:**  $u_{sp,fuzzy,x}, u_{sp,fuzzy,y}$

- 1:  $modulo \leftarrow |u_{sp,fuzzy,x}|$
  - 2: **Se**  $modulo > 1$  **Faça**
  - 3:    $u_{sp,fuzzy,x} \leftarrow u_{sp,fuzzy,x}/modulo$
  - 4:    $u_{sp,fuzzy,y} \leftarrow u_{sp,fuzzy,y}/modulo$
  - 5: **Fim Se**
  - 6:  $modulo \leftarrow |u_{sp,fuzzy,y}|$
  - 7: **Se**  $modulo > 1$  **Faça**
  - 8:    $u_{sp,fuzzy,x} \leftarrow u_{sp,fuzzy,x}/modulo$
  - 9:    $u_{sp,fuzzy,y} \leftarrow u_{sp,fuzzy,y}/modulo$
  - 10: **Fim Se**
- 

Os conjuntos fuzzy para as saídas vetoriais, bem como suas funções de pertinência podem ser vistas na Figura 37.

**Figura 37 – Conjuntos Fuzzy para as saídas vetoriais**



**Fonte:** autoria própria

A Tabela 5 traz as regras fuzzy para este controlador. A coluna “Relação das Entradas”

define a operação booleana utilizada para associar entradas, quando há mais de uma na regra. Por exemplo, a linha 3 define a regra “Se ‘SE’ não é ‘DistSat’ e ‘SDE’ é ‘DistSat’, então ‘SPRecY’ é ‘PP’ e ‘RegDistYSL’ é ‘Z’”. As saídas são sempre associadas com operação booleana “E”.

**Tabela 5 – Regras Fuzzy para sistema “Seguir Parede”**

Relação das Entradas		Entradas					Saídas			
		SE	SD	SDE	SDD	SF	SPRecX	SPRecY	RegDistYSL	RegDistYSD
AND	1	DistSat	DistSat	DistSat	DistSat	-	Z	Z	-	-
OR	2	$\neg$ DistSat	$\neg$ DistSat	$\neg$ DistSat	$\neg$ DistSat	-	PP	-	-	-
AND	3	$\neg$ DistSat	-	DistSat	-	-	-	PP	Z	-
AND	4	-	$\neg$ DistSat	-	DistSat	-	-	NP	Z	-
AND	5	$\neg$ DistSat	-	-	-	$\neg$ DistSat	-	NP	Z	-
AND	6	-	$\neg$ DistSat	-	-	$\neg$ DistSat	-	PP	Z	-
-	7	DistP	-	-	-	-	-	-	Z	-
-	8	DistM	-	-	-	-	-	PP	-	
-	9	DistG	-	-	-	-	-	PP	-	
-	10	-	DistP	-	-	-	-	-	Z	-
-	11	-	DistM	-	-	-	-	-	NP	-
-	12	-	DistG	-	-	-	-	-	NP	-
-	13	-	-	DistP	-	-	-	-	-	Z
-	14	-	-	DistM	-	-	-	-	PP	
-	15	-	-	DistG	-	-	-	-	PP	
-	16	-	-	-	DistP	-	-	-	-	Z
-	17	-	-	-	DistM	-	-	-	-	NP
-	18	-	-	-	DistG	-	-	-	-	NP
AND	19	$\neg$ DistSat	-	$\neg$ DistSat	-	$\neg$ DistSat	-	NP	Z	-
AND	20	-	$\neg$ DistSat	-	$\neg$ DistSat	$\neg$ DistSat	-	PP	Z	-

#### 4.1.2.4 A ESTRATÉGIA PARA FUSÃO DE COMPORTAMENTOS

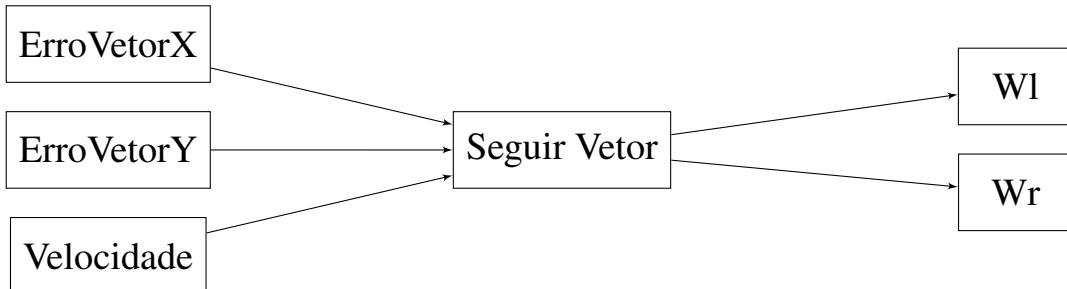
A fusão dos comportamentos “Ir Para Objetivo”, “Evitar Obstáculo” e “Seguir Parede” é dada pela Equação 43, onde  $C$  é a constante de ativação e  $\alpha$  é uma constante definida como 1,6. O vetor da recomendação final deve ter valores  $x$  e  $y$  no intervalo  $[-1, 1]$  e por isso, é utilizado novamente o Algoritmo 8 para deixá-lo em conformidade com a entrada do sistema fuzzy responsável por seguir recomendação.

$$\mathbf{u}_{\text{final}} = (1 - C) \cdot (\mathbf{u}_{\text{ipo}} + \alpha \cdot \mathbf{u}_{\text{ao,fuzzy}}) + C \cdot \mathbf{u}_{\text{sp,fuzzy}} \quad (43)$$

#### 4.1.2.5 CONTROLADOR FUZZY PARA ”SEGUIR RECOMENDAÇÃO”

A Figura 38 mostra o diagrama do sistema *fuzzy* responsável por seguir um vetor de recomendação. As entradas “ErroVetorX” e “ErroVetorY” são as componentes do vetor calculado pela Equação 43. A entrada “Velocidade” é dada pela saída “RecV” do comportamento “Evitar Obstáculo”. As saídas  $W_l$  e  $W_r$  são velocidades angulares dos motores esquerdo e direito, respectivamente.

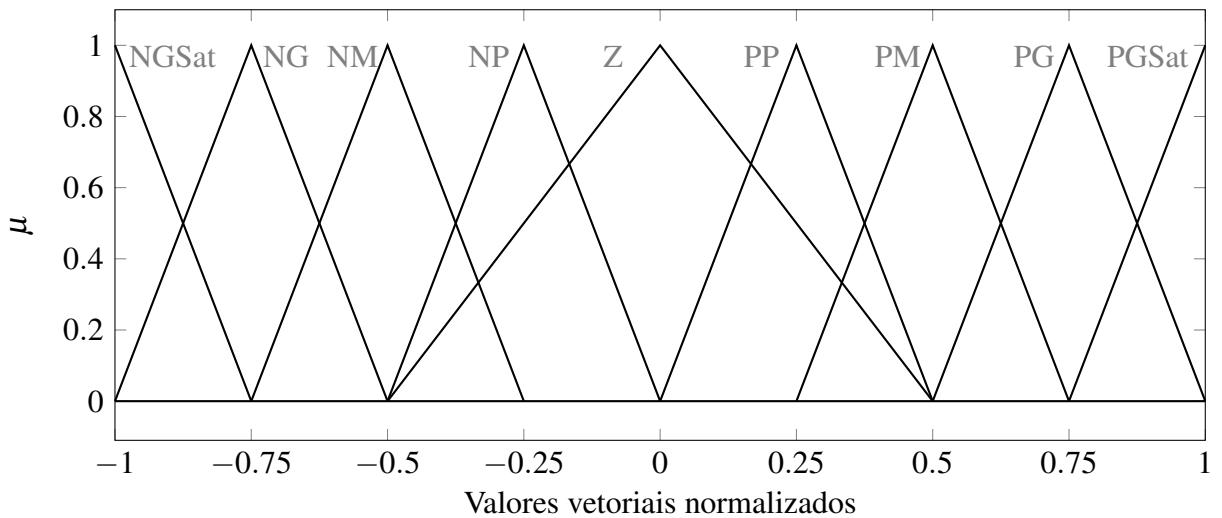
**Figura 38 – Diagrama do sistema Fuzzy “Seguir Recomendação”**



**Fonte: autoria própria**

Os conjuntos fuzzy para as entradas vetoriais, bem como suas funções de pertinência podem ser vistas na Figura 39. A função de pertinência para “Z” foi definida de modo a se sobrepor a outros valores, a fim de evitar oscilações.

**Figura 39 – Conjuntos Fuzzy para as entradas vetoriais**



**Fonte: autoria própria**

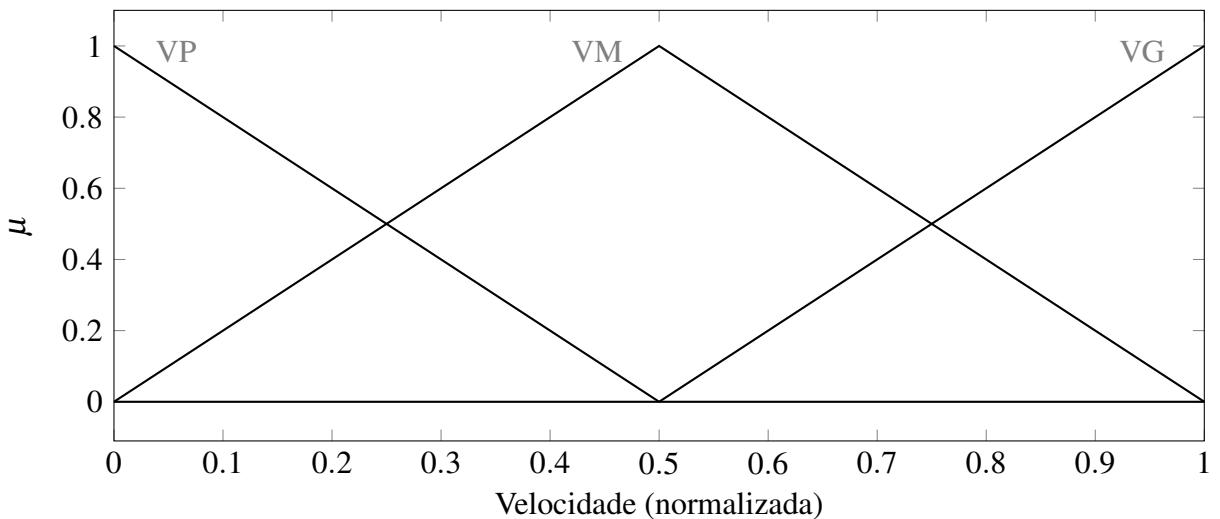
Os conjuntos fuzzy para a entrada da velocidade, bem como suas funções de pertinência podem ser vistas na Figura 40.

Os conjuntos fuzzy para as saídas em velocidade angular, bem como suas funções de pertinência podem ser vistas na Figura 41.

As regras *fuzzy* para esse sistema foram definidas de maneira dinâmica, pois a quantidade de combinações entre valores para entradas e saídas acaba trazendo complexidade. Com 9 valores possíveis para 2 vetores de entrada e 3 valores para velocidades, são necessárias 243 regras ( $9 \times 9 \times 3$ ) que combinem as três entradas em duas saídas.

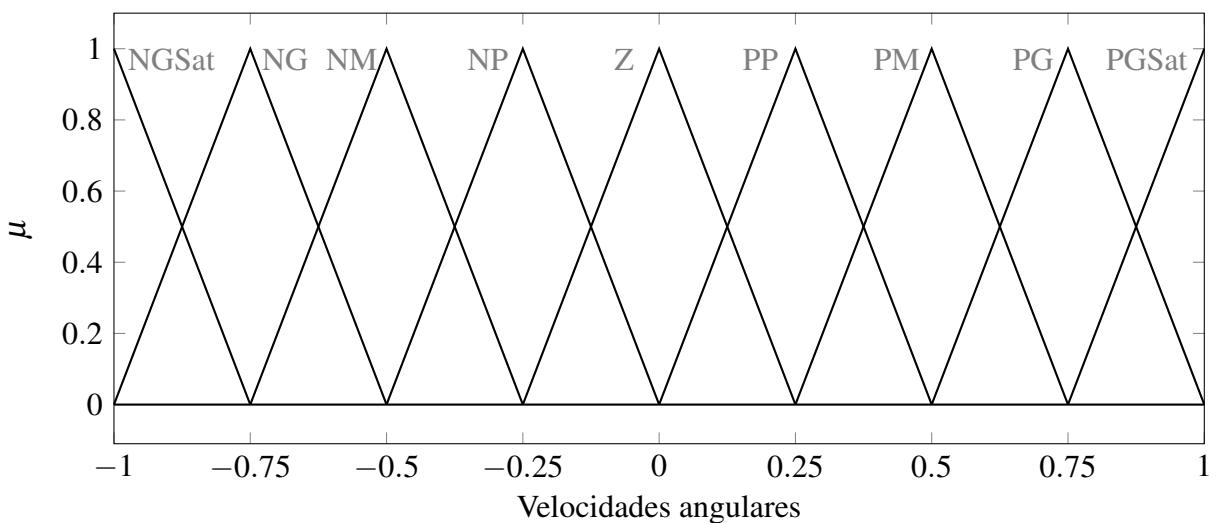
Contudo, computacionalmente, não necessariamente isso se converte em grande custo computacional, já que a quantidade máxima de regras ativas é muito menor. Cada vetor de

**Figura 40 – Conjuntos Fuzzy para a entrada de velocidade**



**Fonte:** autoria própria

**Figura 41 – Conjuntos Fuzzy para as velocidades angulares de saída**



**Fonte:** autoria própria

entrada tem no máximo 3 valores ativos e para a entrada de velocidade, no máximo 2. São, portanto, no máximo 18 regras ativas ( $3 \times 3 \times 2$ ).

Se forem associados os valores “NGSat”, “NG”, “NM”, [...], “PGSat” aos números no intervalo [-4, 4], as Tabelas 6 mostram valores intermediários, calculados com base nas entradas (ErroVetorX, ErroVetorY e Velocidade) e que são utilizados para determinar duas variáveis importantes, “RecV” e “RecW”, que são recomendações para velocidade linear e angular, respectivamente.

As Tabelas 6.a, 6.b e 6.c tem por intuito sugerir velocidades maiores quando o objetivo está distante, e reduzi-las para objetivo próximo, porém respeitando a recomendação

**Tabela 6 – Recomendações para velocidades**

X \ Y	-4	-3	-2	-1	0	1	2	3	4
-4	1	1	1	1	1	1	1	1	1
-3	1	1	1	1	1	1	1	1	1
-2	1	1	1	1	1	1	1	1	1
-1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1

(a) Recomendacão para velocidade VP

X \ Y	-4	-3	-2	-1	0	1	2	3	4
-4	2	2	2	2	2	2	2	2	2
-3	2	1	1	1	1	1	1	1	2
-2	2	1	1	1	1	1	1	1	2
-1	2	1	1	1	1	1	1	1	2
0	2	1	1	1	0	1	1	1	2
1	2	1	1	1	1	1	1	1	2
2	2	1	1	1	1	1	1	1	2
3	2	1	1	1	1	1	1	1	2
4	2	2	2	2	2	2	2	2	2

(b) Recomendacão para velocidade VM

X \ Y	-4	-3	-2	-1	0	1	2	3	4
-4	3	3	3	3	3	3	3	3	3
-3	3	2	2	2	2	2	2	2	3
-2	3	2	2	2	2	2	2	2	3
-1	3	2	2	1	1	1	2	2	3
0	3	2	2	1	0	1	2	2	3
1	3	2	2	1	1	1	2	2	3
2	3	2	2	2	2	2	2	2	3
3	3	2	2	2	2	2	2	2	3
4	3	3	3	3	3	3	3	3	3

(c) Recomendacao para velocidade VG

X \ Y	-4	-3	-2	-1	0	1	2	3	4
-4	4	4	4	4	4	4	4	4	4
-3	4	4	4	3	3	3	3	3	3
-2	4	4	3	2	2	2	2	2	2
-1	4	3	2	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
1	-4	-3	-2	-1	-1	-1	-1	-1	-1
2	-4	-4	-3	-2	-2	-2	-2	-2	-2
3	-4	-4	-4	-3	-3	-3	-3	-3	-3
4	-4	-4	-4	-4	-4	-4	-4	-4	-4

(d) Recomendação para velocidade angular

**Fonte: autoria própria**

de velocidade (“VP”, “VM” ou “VG”). A célula da tabela que corresponder aos valores x e y de entrada, tem seu valor atribuído à variável “RecV”.

A Tabela 6.d é usada para calcular “RecW”. Aqui, um valor positivo indica sentido anti-horário de rotação e valores negativos, sentido horário. Quando o vetor tem componente x próxima de zero, “RecW” é pequeno, de modo a manter o robô na direção do vetor.

Os valores de “RecV” e “RecW” são fáceis de calcular, já que é apenas uma pesquisa em tabela. Contudo, como até 18 regras podem estar ativas, esse cálculo é realizado para uma lista de valores de entrada ativos.

O cálculo para as recomendações da regra fuzzy para  $W_l$  e  $W_r$  são dadas pela Equação 44. As recomendações para “RecV” e “RecW” são somadas. Para motor esquerdo, o valor “RecW” foi subtraído porque sua rotação tende a mover o robô no sentido negativo da convenção (deslocamento angular é negativo para sentido horário de movimento). A aplicação da função mínimo e máximo permite saturar a saída, garantindo que o resultado fique no

intervalo [-4, 4].

$$\begin{bmatrix} RecWl \\ RecWr \end{bmatrix} = \begin{bmatrix} min(max(RecV - RecW, -4), 4) \\ min(max(RecV + RecW, -4), 4) \end{bmatrix} \quad (44)$$

Dessa forma, a saída numérica pode ser convertida novamente ao valor *fuzzy* associado (“NGSat”, “NG”, “NM”, …), o que permite calcular as regras do sistema em meio à execução.

Como exemplo, dado que a entrada para Velocidade é “VG” (3), ErroVetorX é “NM” (-2) e ErroVetorY é “PM” (2), pela tabela, “RecV” vale 2 e “RecW” vale 2. O cálculo da equação resulta em “RecWI” com valor 0 e “RecWr” com valor 4. A regra fuzzy completa pode ser entendida como “Se ‘ErroVetorX’ é ‘NM’, ‘ErroVetorY’ é ‘PM’ e ‘Velocidade’ é ‘VG’, então ‘WI’ é ‘Z’ e ‘Wr’ é ‘PGSat’”.

## 4.2 IMPLEMENTAÇÃO DO CIRCUITO E SISTEMA EMBARCADO

Nesta seção, os aspectos de projeto do circuito e implementação do sistema embarcado são desvelados.

### 4.2.1 CONSIDERAÇÕES PRÁTICAS, LIMITAÇÕES DO MODELO E DA SIMULAÇÃO

Uma questão prática relevante é a utilização de caixas de redução entre motores e pneu. A caixa de redução efetivamente reduz a velocidade máxima do robô pela razão de redução, já que o motor gira mais que o pneu. Porém, os *encoders* estão posicionados atrás do motor, de modo que a resolução efetiva é menor que a resolução individual dos *encoders*. Portanto, o cálculo de odometria fica mais preciso pelo uso de caixas de redução.

Por conta da dificuldade de controlar o modelo uniciclo (por ser subatuado), foi escolhido controlar a variável ângulo (controle clássico). Portanto, não foi realizado o controle por realimentação de estado que se pretendia no inicio deste trabalho (controle moderno). Os erros nas coordenadas *x* e *y* só são zerados alterando a referência constantemente. Desta forma não há garantia quanto ao tempo de convergência (o robô nunca aumenta velocidade para compensar atrasos, nem diminui para compensar adiantamentos).

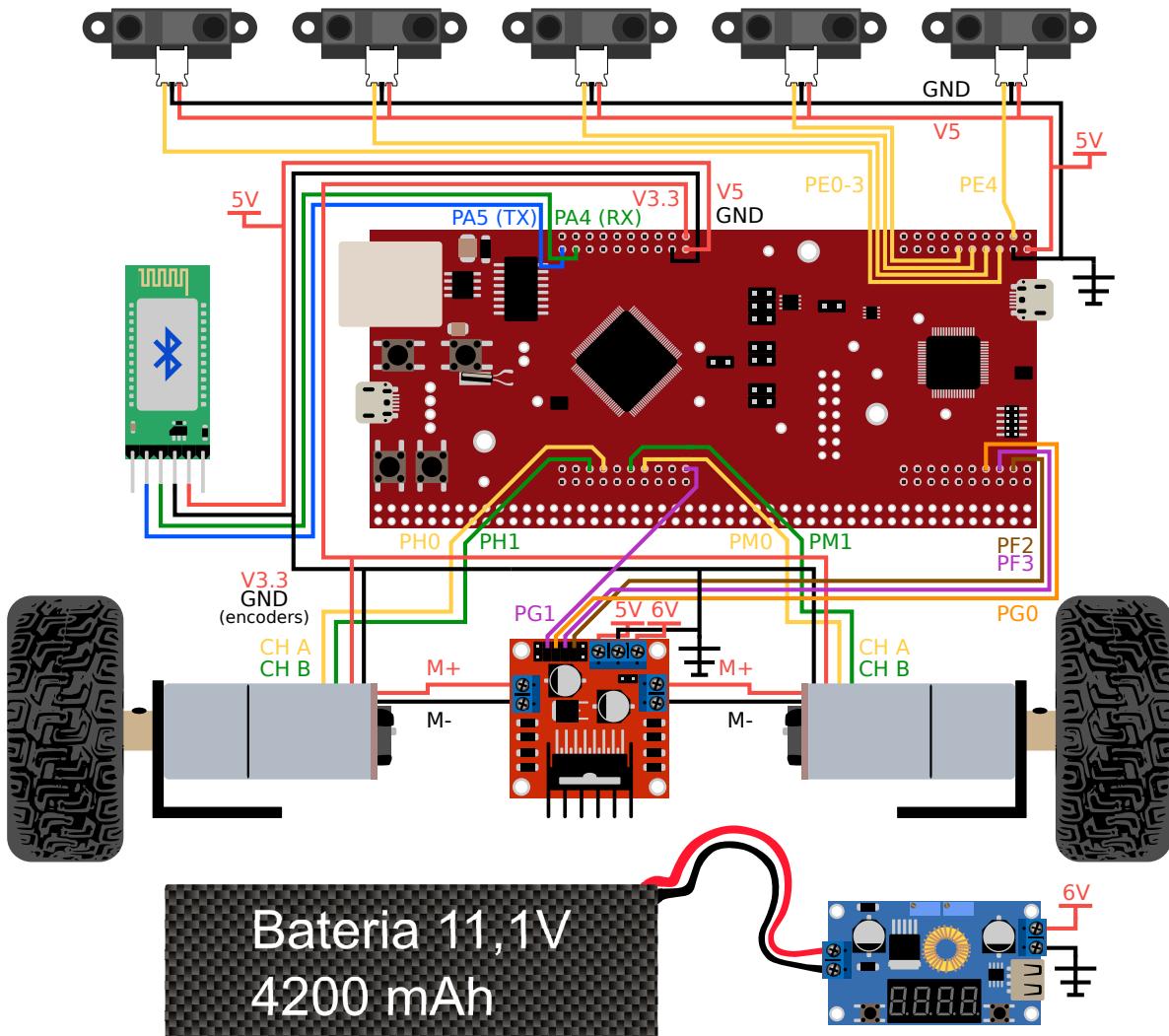
O controle é, portanto, realizado apenas por cinemática, desde a simulação até a implementação. Em robótica, desconsiderar a dinâmica do sistema é um grande problema para robôs que funcionem em altas velocidades e altas acelerações. Em termos qualitativos, desconsiderar dinâmica significa considerar que o robô pode alterar sua velocidade de maneira instantânea. Portanto, é esperado que o robô real se aproxime mais dos obstáculos do que o

robô simulado, por efeitos de inércia.

#### 4.2.2 CIRCUITO ELÉTRICO

Um diagrama do circuito eletrônico está retratado na Figura 42. Apesar de sua elegância, este circuito possui um erro de projeto.

**Figura 42 – Esquema lógico do circuito**



**Fonte: autoria própria**

O regulador de tensão foi ajustado para produzir saída de 6 volts, pois o acionamento dos motores requer esse nível de tensão. Além dessa saída de 6 volts, são necessárias saídas de 3,3 volts para microcontrolador e *encoders*, além de saída de 5 volts para módulo *bluetooth* e sensores infravermelho.

Durante fase de projeto, verificou-se que o *driver* “L298N” possui internamente um

regulador de tensão para obter 5 *volts*. O microcontrolador, por sua vez, tem um regulador interno para obter 3,3 *volts* a partir de 5 *volts* de entrada.

A escolha de utilizar o regulador interno ao driver parecia uma decisão sagaz, porém mostrou-se um fiasco, já que a queda de tensão nele é maior que 1 *volt*. Dessa forma, com a entrada de 6 *volts*, a saída obtida foi menor que os 5 *volts* teoricamente “prometidos”. Apesar do microcontrolador funcionar, os sensores infravermelho tiveram funcionamento bastante imprevisível, ora medindo adequadamente, ora medindo com erro.

Uma boa solução para este problema é utilizar outro regulador de tensão para obter 5 *volts* a partir da tensão da bateria. Apesar disso, uma solução fácil, porém muito questionável foi adotada: em vez de utilizar 6 *volts* nos motores, foi utilizado valor de 7 *volts*, o que compensa as quedas de tensão do regulador interno ao *driver*. Dessa forma, os motores funcionam além da especificação, reduzindo sua vida útil, mas foi possível continuar o desenvolvimento e validação dos algoritmos deste trabalho.

#### 4.2.3 CONFIGURAÇÃO DE PORTAS NO MICROCONTROLADOR

O microcontrolador utilizado neste trabalho possui dois módulos ADC, com até 20 canais. Cada ADC pode ser usado para a conversão de mais de um canal sem intervenção do processador, por meio de um *hardware* chamado sequenciador de amostra (*sample sequencer*). Estas informações podem ser vistas no *datasheet* (TEXAS INSTRUMENTS INCORPORATED, 2014), a partir da página 1053.

As portas “PE0” a “PE4”, conectadas aos sensores IR foram configuradas como entradas para conversores AD, de maneira distribuída, com os canais 1 a 3 (portas “PE2” a “PE0”) vinculados ao conversor “ADC0” e os canais 0 (“PE3”) e 9 (“PE4”) utilizando o conversor “ADC1”. A associação entre portas e canais podem ser vistos na tabela da página 1055 do *datasheet* (TEXAS INSTRUMENTS INCORPORATED, 2014).

O microcontrolador usado possui um módulo PWM com quatro blocos de geradores de sinal. Cada gerador de sinal, por sua vez, pode ter sua saída vinculada a duas portas distintas, dando um total de oito saídas de PWM. Além disso, existem geradores de banda morta, que são usados para evitar curto-circuitos em uma “ponte H” (*driver*). Os curtos-circuitos em uma “ponte H” são causados porque seus transistores internos não “ligam” e “desligam” instantaneamente. Assim, o gerador de banda morta cria dois sinais opostos (o segundo canal é invertido) e com *delays* para a mudança de nível, a fim de “respeitar” a dinâmica dos transistores do driver. Estas informações podem ser vistas a patir da página 1669 do *datasheet* (TEXAS

INSTRUMENTS INCORPORATED, 2014). A geração de banda morta é explicada brevemente na página 1675.

As portas “PF2” e “PF3” estão associadas às saídas “PWM2” e “PWM3” (gerador 1), enquanto as portas portas “PG0” e “PG1” estão vinculadas às saídas “PWM4” e “PWM5” (gerador 2). A associação entre portas e saídas PWM podem ser vistos na tabela da página 1672 do *datasheet* (TEXAS INSTRUMENTS INCORPORATED, 2014). As portas “PF” foram conectadas às entradas lógicas do *driver* para o motor esquerdo e as portas “PG”, às entradas do *driver* referentes ao motor direito.

As portas “PH0” e “PH1” foram configuradas para *encoders* do motor esquerdo e as portas “PM0” e “PM1” para *encoders* do motor direito. Estas portas foram configuradas como entradas lógicas. Interrupções por borda de descida e subida foram configuradas para as portas “PH0” e “PM0” (canais A dos *encoders* dos motores esquerdo e direito, respectivamente). Ao detectar uma interrupção em “PH0”, verificam-se os níveis lógicos das portas “PH0” e “PH1”. A comparação efetuada foi explicada na Seção 2.8.

Para comunicação com o módulo *bluetooth* foi utilizada a comunicação “UART”. O microcontrolador utilizado possui oito destes módulos. A “UART3” foi configurada para as portas “PA4” e “PA5” (RX e TX, respectivamente). Vale salientar que o receptor (RX) do microcontrolador deve se conectar ao transmissor (TX) do módulo *bluetooth* e vice-versa. Portanto, os rótulos RX e TX da Figura 42 são especificações para o microcontrolador. A associação entre portas e sinais RX e TX do módulo “UART” podem ser vistos na tabela da página 1163 do *datasheet* (TEXAS INSTRUMENTS INCORPORATED, 2014).

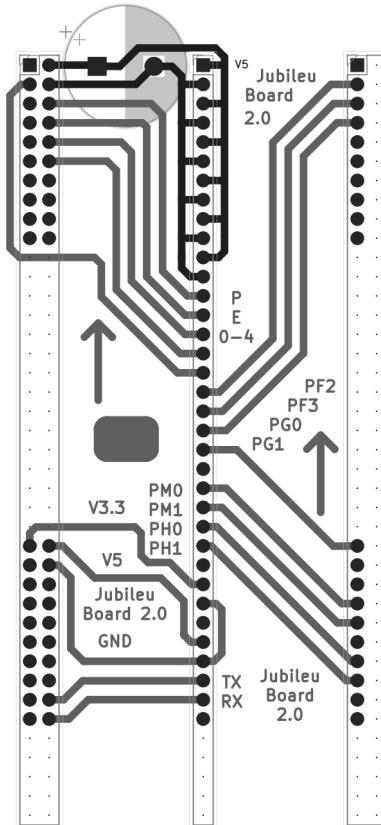
#### 4.2.4 PROJETO DA PLACA

Uma placa foi projetada no intuito de relacionar as portas do microcontrolador aos pinos utilizados pelos periféricos e assim, servir de interface. A Figura 43 mostra o circuito projetado com o *software* “Kicad”. O capacitor indicado na placa possui valor de  $1000\mu F$ , que é maior que o necessário.

#### 4.2.5 ESQUEMA LÓGICO DO SISTEMA EMBARCADO

Em nível de implementação, é uma boa prática de engenharia de *software* definir uma arquitetura capaz de abranger os diferentes requisitos. Como este trabalho utiliza dois tipos de controle, híbrido e fuzzy, para o código funcionar genericamente, a estratégia de fusão de comportamentos foi transformada em um autômato híbrido com dois estados: “Parar” e

**Figura 43 – Placa projetada**



**Fonte:** autoria própria

“Andar”. “Parar” toma controle quando o robô está no objetivo e quando deixa de estar (por alteração da coordenada objetivo) o comportamento “Andar” toma conta. Efetivamente, ainda é fusão de comportamentos, pois no estado “Andar” todos os comportamentos são calculados e mesclados em uma única recomendação, mas em termos de implementação, utiliza-se uma mesma arquitetura de autômatos.

Portanto, são dois autômatos (supervisores), um para arbitragem e outro para fusão de comportamentos. A fim de selecionar entre estratégias de controle, um autômato com dois estados ficou responsável por selecionar o supervisor utilizado. A grosso modo, é um autômato de seleção de supervisores.

Uma pequena interface por linha de comando via *bluetooth* foi implementada no sistema embarcado e é responsável por definir estratégias de controle que serão utilizadas na navegação, bem como estipular a coordenada objetivo e também para facilitar depuração. O nome desses comandos e respectivas descrições podem ser vistos na Tabela 7.

**Tabela 7 – Comandos disponíveis na interface por linha de comando via bluetooth**

Comando	Descrição
SUPERVISOR_HIBRIDO	Adota o controlador híbrido como a estratégia de navegação utilizada
SUPERVISOR_FUZZY	Adota o controlador fuzzy como a estratégia de navegação utilizada
SETCOORDOBJ(X.XX,Y.YY)	Redefine a coordenada objetivo
GETSTATE	Utilizado para obter a resposta do estado do robô ( $x$ , $y$ e $\theta$ ) a cada iteração, bem como o valor de seus sensores
EXIT	Necessário para solicitar ao robô para que deixe de informar seu estado a cada iteração

#### 4.3 SIMULAÇÃO

Nesta seção pretende-se mostrar resultados de simulação para as estratégias de arbitragem e fusão de comportamentos.

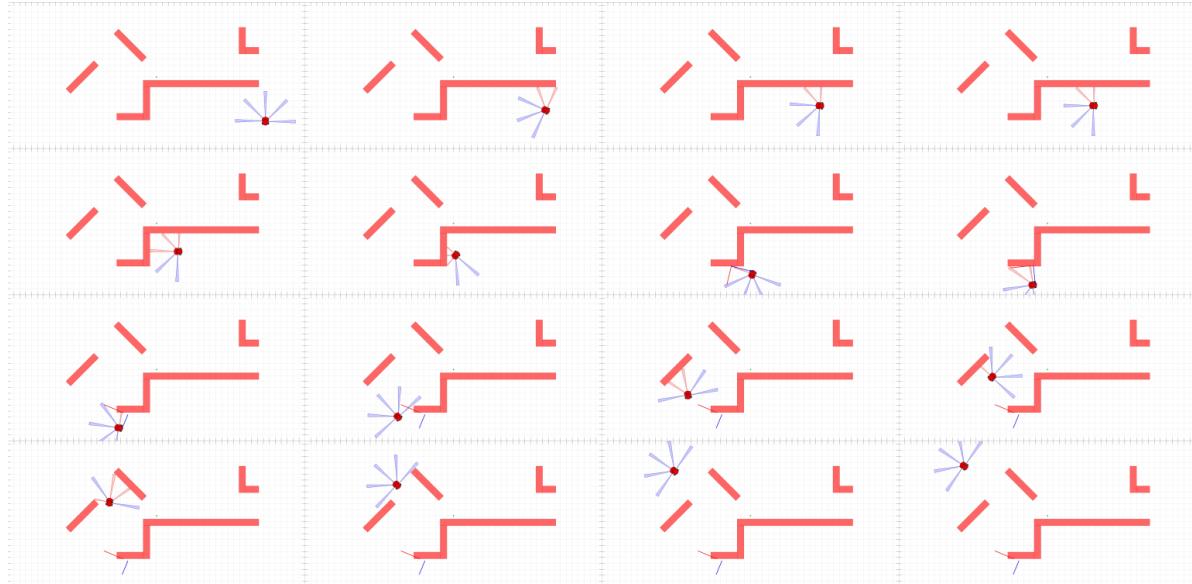
O código oficial para o simulador Simiam se encontra no repositório *Github* em Croix (2014). O código da simulação deste trabalho é um “*fork*” (ramificação de código) do repositório oficial e pode ser consultado em Carbonera (2021b).

##### 4.3.1 UTILIZANDO CONTROLADOR HÍBRIDO

A simulação para o controlador híbrido pode ser vista na Figura 44. A sequência das 16 imagens pode ser entendida analisando no sentido de leitura: da esquerda para direita, de cima para baixo. O robô adota comportamento “Ir para Objetivo” na primeira imagem e, quando encontra um obstáculo em formato côncavo (*frames* 2 a 5), adota o comportamento mesclado “Ir para objetivo e Evitar Obstáculos”. No *frame* 6, o robô adota o comportamento “Evitar Obstáculo”, o que leva o robô a se afastar do objetivo o suficiente para o comportamento “Seguir Parede” ser adotado (*frames* 7 a 9). No restante do trajeto, há uma troca constante entre comportamentos. Nos *frames* 10, 14, 15 e 16 o comportamento “Ir para Objetivo” está ativo, enquanto nos *frames* 11 a 13, o comportamento mesclado está no controle.

O comportamento “Evitar Obstáculos” ocorre apenas em situações muito específicas, normalmente quando algum obstáculo pequeno está em um ponto cego do robô, ou quando o sentido de movimento forma ângulo de 90 graus com o obstáculo. Em simulação a inércia não é considerada (simulação considera apenas cinemática). Na prática, a inércia é outro motivo para a adoção deste comportamento. No momento em que este comportamento é adotado a reação

**Figura 44 – Resultado em simulação para o controlador híbrido**



**Fonte:** autoria própria

do robô é tão rápida que é difícil mostrar em imagens.

#### 4.3.2 UTILIZANDO CONTROLADOR FUZZY

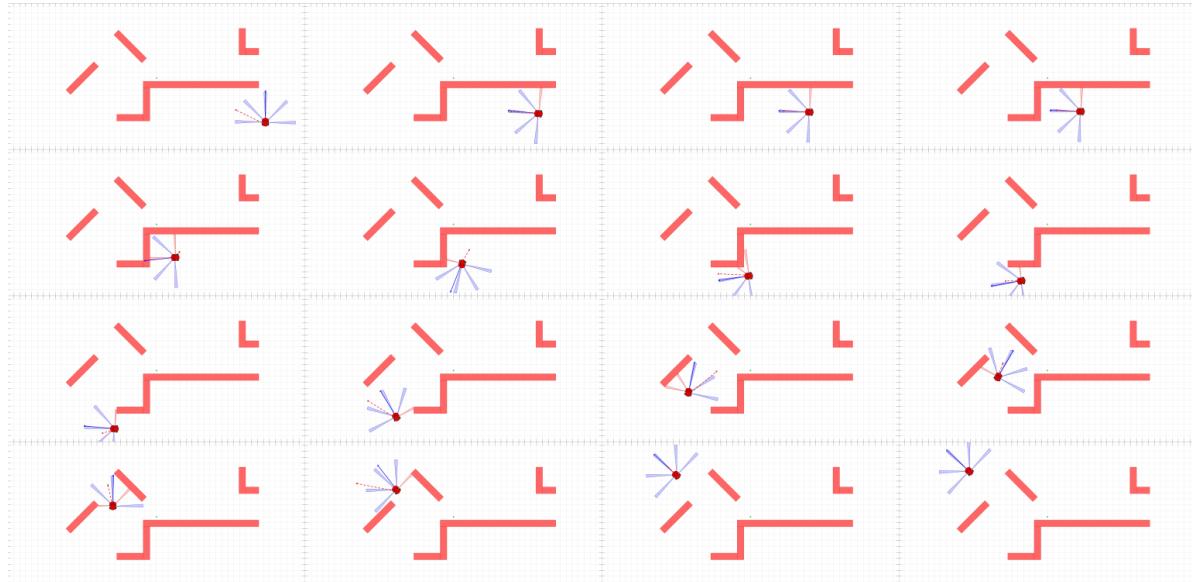
A simulação para o sistema *Fuzzy* pode ser vista na Figura 45. Aqui, por conta da arquitetura de fusão de comportamentos (com arquitetura de subsunção), existem muitas oscilações, que ocorrem quando o robô precisa decidir entre “Seguir Parede” ou “Ir Para Objetivo”. Como a arquitetura de subsunção é implementada com um contador que determina um peso para combinar linearmente dois comportamentos, oscilações no valor desse contador provocam oscilações na saída. Entre os *frames* 5 e 6, o robô chega a ficar parado por um tempo até o contador ter um valor suficiente para mudar a recomendação.

### 4.4 RESULTADO E DISCUSSÃO

Nesta seção, pretende-se mostrar resultados de implementação para as estratégias de arbitragem e fusão de comportamentos, além de discutir vantagens e desvantagens destas arquiteturas.

O ambiente utilizado para verificação do resultado é similar ao ambiente de simulação, mas não tem a mesma escala, por que a área disponível é menor. Vale salientar que quanto mais se popula o ambiente com obstáculos, mais ele se parece com um labirinto. Neste caso, a arquitetura deliberativa seria mais apropriada do que a arquitetura baseada em comportamento.

**Figura 45 – Resultado em simulação para o controlador fuzzy**



**Fonte:** autoria própria

Por isso que o ambiente de testes adotado é um ambiente esparsamente populado.

O código implementado para o microcontrolador pode ser consultado pelo repositório *Github* em Carbonera (2021a).

#### 4.4.1 RESULTADO PARA CONTROLADOR HÍBRIDO

Nesta subseção, pretende-se mostrar todos os comportamentos separadamente para depois apresentar o resultado com a lógica de arbitragem de comportamentos.

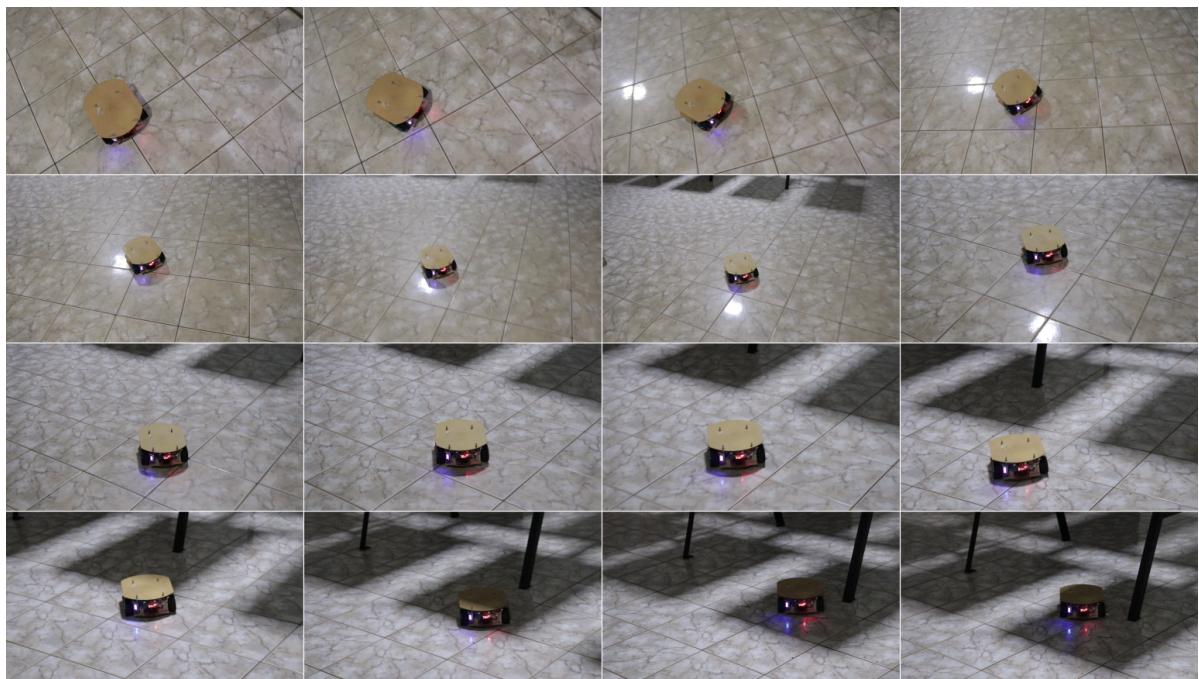
##### 4.4.1.1 COMPORTAMENTO IR PARA OBJETIVO

O resultado para o comportamento “Ir para Objetivo” pode ser visto na Figura 46. O ângulo de referência do robô converge rapidamente e, sem obstáculos, o robô segue com agilidade até o objetivo.

##### 4.4.1.2 COMPORTAMENTO EVITAR OBSTÁCULO

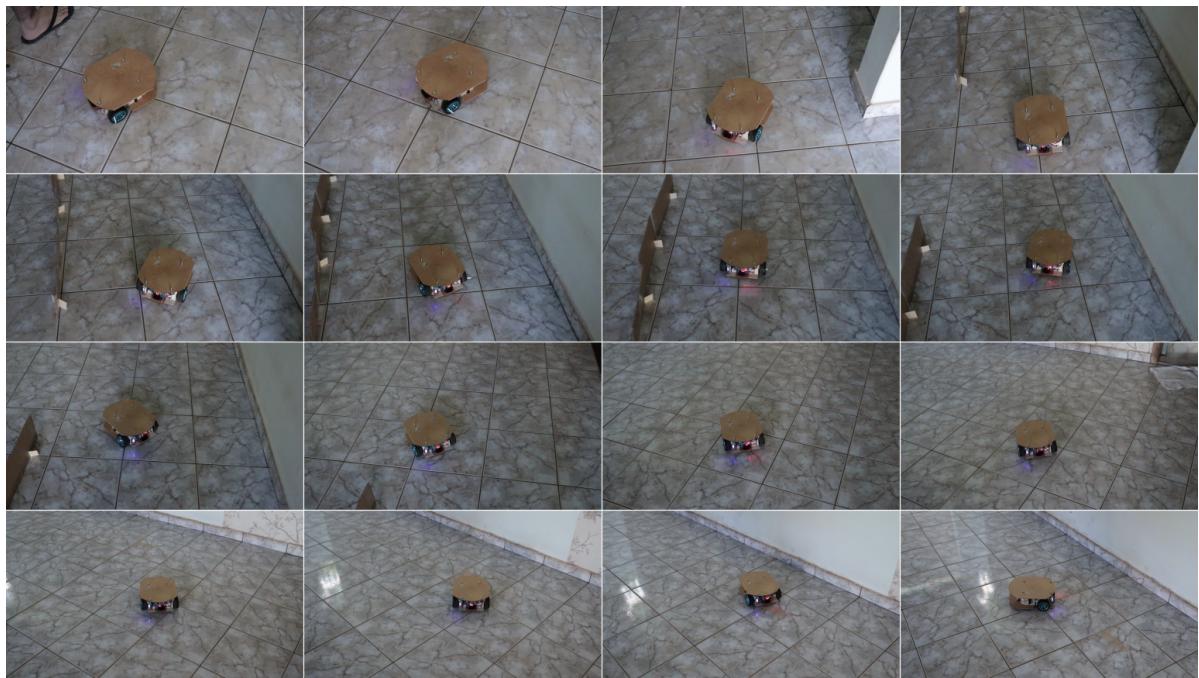
O resultado para o comportamento “Evitar Obstáculo” pode ser visto na Figura 47. O robô sempre busca adotar o caminho contrário ao obstáculo percebido, o que explica os movimentos de deflexão observados na Figura.

**Figura 46 – Resultado do comportamento “Ir Para Objetivo”**



**Fonte: autoria própria**

**Figura 47 – Resultado do comportamento “Evitar Obstáculo”**



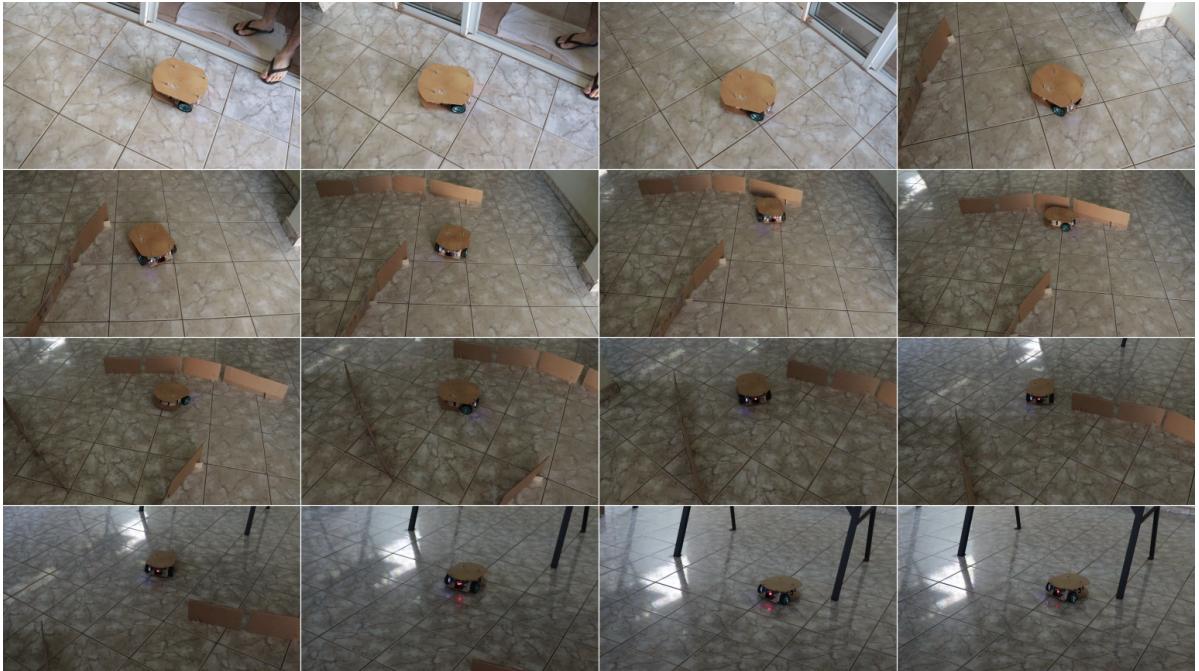
**Fonte: autoria própria**

#### 4.4.1.3 COMPORTAMENTO MESCLADO

O resultado para o comportamento mesclado “Ir para Objetivo e Evitar Obstáculo” pode ser visto na Figura 48. Com este comportamento o robô é capaz de navegar em ambientes

sem mínimos locais.

**Figura 48 – Resultado do comportamento “Mesclado”**



**Fonte:** autoria própria

#### 4.4.1.4 COMPORTAMENTO SEGUIR PAREDE

O resultado para o comportamento “Seguir Parede” pode ser visto na Figura 49. O sentido horário de contorno foi adotado e o robô contorna indefinidamente a parede. É papel do autômato definir o sentido antes de iniciar a execução do controlador.

#### 4.4.1.5 CONTROLADOR FINAL COM ARBITRAGEM

A Figura 50 mostra o resultado para o controlador híbrido no robô real. A mesma sequência de trocas entre comportamentos acontece aqui, mas por conta de efeitos de inércia e ruído nos sensores, ocorrem mudanças mais intensas entre comportamentos do que em simulação.

Durante a execução deste trabalho, verificou-se que a arquitetura de arbitragem de comportamentos pode ter um problema muito grave quando há presença de ruído. Nessa situação, o autômato pode fazer uma transição de estado indevida. Quando a transição ocorre para um estado no qual não há transição de retorno para o estado anterior, o robô não funciona de acordo com o que foi projetado em simulação e pode ficar preso em mínimos locais ou escolher seguir parede para o lado errado e com isso se chocar com obstáculos.

**Figura 49 – Resultado do comportamento “Seguir Parede”**



**Fonte: autoria própria**

Nos sensores utilizados neste trabalho, acontece um tipo de ruído que não pôde ser evitado: em questão de poucas iterações (entre 1 e 4) o sensor detecta obstáculo onde não tem. Neste caso, o sensor está detectando uma distância de saturação de 80 centímetros quando, de repente, há uma queda para cerca de 40 centímetros, que logo volta para o valor anterior.

O motivo deste ruído não foi descoberto, mas foi ponderado que pode ser alguma queda de tensão no circuito. Por não ter equipamento (osciloscópio) para testar esse palpite, algumas mudanças foram feitas em software. Foi implementado um filtro de média móvel com janela de 5 iterações para reduzir o efeito do ruído. Mas este valor filtrado foi utilizado apenas para verificar se é necessário fazer transições de estado. Para executar os controladores, utilizou-se o valor real com ruído mesmo. Isso foge um pouco da arquitetura baseada em comportamento, já que os componentes dessa arquitetura precisam ser reativos (valores atuais, sem considerar o passado do sistema). Deste modo, os comportamentos são reativos, mas o processo de “decisão” não pode ser classificado como tal, pois utiliza estados anteriores do sistema.

Além dessa alteração, foi definida uma saturação para os sensores diagonais e frontais. Quando estes sensores detectam valores maiores do que 70 centímetros, estes valores são alterados para 80 centímetros, a fim de reduzir o efeito dos ruídos no comportamento.

**Figura 50 – Resultado implementado para o controlador híbrido**



**Fonte:** autoria própria

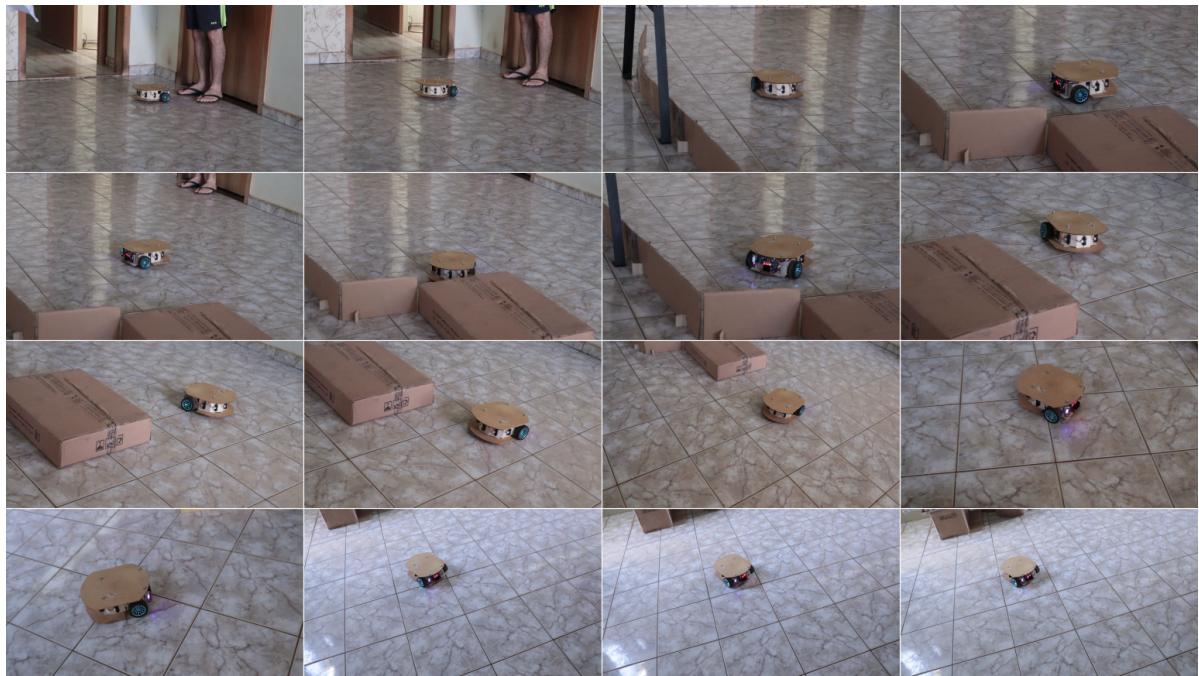
#### 4.4.2 RESULTADO PARA CONTROLADOR *FUZZY*

A Figura 51 mostra o resultado para o controlador *Fuzzy* no robô real. Aqui, por conta de ruído, o robô não chega a parar quando está prestes a adotar o comportamento de “Seguir Parede”. Ele oscila na região de mínimo local, o que pode ser constatado verificando os *frames* 4 a 8. Nos *frames* 14 a 16, nota-se uma certa lentidão no momento de se aproximar do objetivo, pois ocorre uma oscilação entre comportamentos de “Ir para Objetivo” e “Seguir Parede”. Isso pode ser explicado da seguinte forma: a proximidade do objetivo leva a uma recomendação de velocidade pequena e o atrito dos motores faz o robô parar. Deixando de fazer progresso, o contador da arquitetura de subsunção aumenta o peso para o comportamento de “Seguir Parede”, o que faz o robô voltar a fazer progresso e o valor do contador diminui novamente. Por isso o robô para e anda de pouco a pouco até chegar no objetivo.

As oscilações excessivas para o controlador *Fuzzy* são provocadas pela arquitetura de fusão de comportamentos e não pelo sistema *fuzzy* em si.

Os ruídos relatados para a arquitetura de arbitragem de comportamentos também ocorrem aqui. Contudo, apenas a definição de uma saturação para sensores diagonais e central (para valores acima de 70 centímetros) foi suficiente para fazer o controlador funcionar, não sendo necessário nenhum filtro de média móvel. Componentes reativos são muito robustos para sistemas que apresentam ruídos de pouca duração, já que um comportamento errático

**Figura 51 – Resultado implementado para o controlador *fuzzy***



**Fonte: autoria própria**

durará apenas enquanto os ruídos estiverem presentes e, logo a seguir, o controlador retorna ao funcionamento normal.

## 5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve por intuito defender que a definição de uma boa arquitetura para solução de um problema é mais importante que a tecnologia empregada. Com a definição de uma arquitetura robusta, o funcionamento do sistema independe de tecnologia. Neste trabalho, por exemplo, utilizou-se controladores PID e controladores *Fuzzy* para implementar as estratégias de arbitragem e fusão, mas elas poderiam ter sido utilizadas em conjunto sob uma mesma estratégia. Se um novo comportamento precisar ser implementado, qualquer tecnologia poderia ser usada para implementá-lo (desde soluções reativas simples até algoritmos de aprendizado de máquina), o que torna a solução deste trabalho escalável.

Nesse sentido, é necessário comparar arquiteturas de arbitragem e fusão de comportamento. A primeira é mais escalável que a segunda, já que cada comportamento pode ser testado de modo isolado. Na fusão de comportamentos, cada novo comportamento adicionado se relaciona com todos os outros e a complexidade tende a aumentar de maneira exponencial. O grande problema detectado na arquitetura de arbitragem é a situação de presença de ruído, que foi discutida na seção dos resultados. Este trabalho adotou uma solução que provavelmente não é a melhor. Filtrar o ruído atrasa a “velocidade de percepção”, já que demora mais tempo para que o robô perceba que houve uma mudança no ambiente.

Talvez exista alguma forma de detectar que o ruído ocorreu e desfazer transições indevidas. É interessante investigar em trabalhos futuros uma forma de tornar um autômato mais robusto na presença de ruídos.

A arquitetura de fusão de comportamentos tem a tendência de provocar muitas oscilações, pois ao integrar um novo comportamento, este deve interagir com todos os outros. Em certas circunstâncias ele pode ser suprimido, em outras, deve “cancelar” os outros a fim de se sobressair. As oscilações ocorrem por conta da interação entre comportamentos, quando está ora suprimido, ora se sobressaindo em relação aos demais.

A questão do custo computacional é outro ponto onde a arbitragem de comportamentos se sobressai, pois apenas um comportamento é executado por vez. Desde que cada comportamento execute dentro do tempo disponível, o custo computacional não compromete o funcionamento. Na fusão de comportamentos, por outro lado, todos os custos individuais se somam.

Outro aspecto importante neste trabalho é a conversão de modelos de acionamento diferencial para uniciclo. Pode parecer completamente desnecessário em uma primeira leitura, porém, se houver a necessidade de controlar algum outro robô, com outro modelo matemático, em um mesmo ambiente, percebe-se que, desde que seja possível converter seu modelo para o uniciclo, o mesmo controlador deste trabalho poderá ser usado para controlá-lo.

Considerando que outros robôs podem se beneficiar do mesmo algoritmo de controle deste trabalho, desde que seu modelo possa ser convertido para o uniciclo, seria interessante explorar essa integração em trabalhos futuros. O robô tipo “Segway” seria um bom ponto de partida, já que é necessário apenas um controlador a mais para controlar o ângulo vertical (controle de um pêndulo invertido), bem como uma forma de associar a inclinação do “pêndulo” com a velocidade linear do uniciclo.

O presente trabalho teve objetivos muito extensos e, portanto, deixou muitos pontos que são passíveis de melhoria. Para trabalhos futuros, é interessante investigar melhorias no simulador. Acredito que seria interessante reescrevê-lo em um “ambiente” de código aberto. O “Gazebo” do *Robot Operating System* (ROS) pode ser uma boa alternativa. Ou talvez, escrever uma espécie de extensão para o Blender, já que o aspecto de renderização já está implementado e seria possível obter animações muito bonitas ao concluir uma simulação.

Acredito que melhorias futuras devem levar em conta a importância de definir uma espécie de “infraestrutura” ou “ecossistema” para robótica, que seriam ferramentas e sistemas cujas funcionalidades facilitem o desenvolvimento e testes de robôs. A chave para isso acredito que está em desenvolver um simulador mais robusto, capaz de se conectar aos robôs, usar seus sensores como entrada para o algoritmo a ser testado e devolver o resultado a eles, para que realizem a atuação recomendada (a Universidade Georgia Tech possui algo nesse sentido para os seus robôs Khepera).

Com essa infraestrutura, é possível testar muito rapidamente quaisquer alterações no algoritmo. Com a utilização de câmeras, a coordenada do robô real pode ser estimada, de modo a calcular com maior precisão os erros de odometria, além de facilitar a extração de figuras e vídeos para análise posterior. A partir desse “ecossistema”, um tema mais moderno em robótica móvel pode ser explorado: navegação em grupos.

## REFERÊNCIAS

- ARKIN, R. C. **Behavior-Based Robotics**. [S.l.]: MIT Press, 1998. (Intelligent Robotics and Autonomous Agents Series). ISBN 9780262529204.
- ASTOLFI, A. Exponential stabilization of nonholonomic systems via discontinuous control. **IFAC Proceedings Volumes**, v. 28, n. 14, p. 661 – 666, 1995. ISSN 1474-6670. 3rd IFAC Symposium on Nonlinear Control Systems Design 1995, Tahoe City, CA, USA, 25-28 June 1995. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1474667017469047>>. Acesso em: 1 maio 2019.
- BANAVAR, R. N.; SANKARANARAYANAN, V. **Switched Finite Time Control of a Class of Underactuated Systems**. [S.l.]: Springer-Verlag Berlin Heidelberg, 2006. (Lecture Notes in Control and Information Sciences, v. 333).
- BIESEK, L. J. **Veículo autônomo: uma contribuição para estacionamento**. 50 f. Monografia (Trabalho de Conclusão de Curso (Graduação)) — Universidade Tecnológica Federal do Paraná, Pato Branco, 2016.
- CARBONERA, M. G. **JubileuBotFirmware**. [S.l.]: GitHub, 2021. <https://github.com/mcarbonera/JubileuBotFirmware>.
- CARBONERA, M. G. **Simiam**. [S.l.]: GitHub, 2021. <https://github.com/mcarbonera/simiam>.
- CARONA, R.; AGUIAR, A. P.; GASPAR, J. Control of unicycle type robots tracking, path following and point stabilization. In: **Proc. of IV Jornadas de Engenharia Electrónica e Telecomunicações e de Computadores**. [S.l.: s.n.], 2008.
- CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. 2. ed. [S.l.]: Springer US, 2008. ISBN 978-0-387-33332-8.
- CROIX, J. P. de la. **Quickbot project**. 2013. Disponível em: <<http://jpdelacroix.com/software/simiam.html>>. Acesso em: 21 maio 2019.
- CROIX, J. P. de la. **Simiam**. [S.l.]: GitHub, 2014. <https://github.com/jdelacroix/simiam>.
- CURY, J. E. R. **Teoria de Controle Supervisório de Sistemas a Eventos Discretos**. [S.l.], nov. 2001. In: V Simpósio Brasileiro de Automação Inteligente.
- DIB, A. **Commande non linéaire et asservissement visuel de robots autonomes**. Tese (Theses) — Supélec, out. 2011. Disponível em: <<https://tel.archives-ouvertes.fr/tel-00657022>>. Acesso em: 25 abr. 2019.
- DYNAPAR. **Quadrature Encoder Overview**. 2020. Disponível em: <[https://www.dynapar.com/technology/encoder\\_basics/quadrature\\_encoder/](https://www.dynapar.com/technology/encoder_basics/quadrature_encoder/)>. Acesso em: 2 nov. 2020.

EGERSTEDT, M. Behavior based robotics using hybrid automata. In: **Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control.** [S.l.: s.n.], 2000.

FUCHS, T. et al. **Using PySimiam in Coursera ‘Control of mobile robots’ course.** 2013. Disponível em: <<http://pysimiam.sourceforge.net/coursera.html>>. Acesso em: 21 maio 2019.

GEORGIA ROBOTICS AND INTELLIGENT SYSTEMS LABORATORY. **Sim.I.am.** 2013. Disponível em: <<http://gritslab.gatech.edu/home/2013/10/sim-i-am/>>. Acesso em: 8 abr. 2019.

GLOTFELTER, P.; EGERSTEDT, M. A parametric mpc approach to balancing the cost of abstraction for differential-drive mobile robots. In: **2018 IEEE International Conference on Robotics and Automation (ICRA).** [S.l.: s.n.], 2018. p. 732–737.

HALLIDAY, D.; RESNICK, R.; WALKER, J. **Fundamentos De Física - Volume 1 - Mecânica.** 9<sup>a</sup>. ed. [S.l.]: LTC, 2012. ISBN 9788521630357.

JAZAR, R. N. **Advanced Dynamics: Rigid Body, Multibody, and Aerospace Applications.** 1<sup>a</sup>. ed. [S.l.]: John Wiley, 2011. ISBN 9780470398357.

JOHAN, K. A.; MURRAY, R. M. **Feedback Systems An Introduction for Scientists and Engineers.** 2<sup>a</sup>. ed. [S.l.]: Princeton University Press, 2021. ISBN 0691193983.

KWON, J.-W.; KIM, C.-J.; CHWA, D. Vector field trajectory tracking control for wheeled mobile robots. **2009 IEEE International Conference on Industrial Technology**, p. 1–6, fev. 2009.

LAVALLE, S. **Planning Algorithms.** [S.l.]: Cambridge University Press, 2006. ISBN 9781139455176.

LILLY, J. **Fuzzy Control and Identification.** [S.l.]: Wiley, 2011. ISBN 9781118097816.

LOPES, W. A. **Projeto e implementação de robô autônomo seguidor de linha.** 114 f. Monografia (Trabalho de Conclusão de Curso (Graduação)) — Universidade Tecnológica Federal do Paraná, Pato Branco, 2017.

MAHESHWARI, A.; SMID, M. **Introduction to Theory of Computation.** School of Computer Science Carleton University, 2019. Disponível em: <<https://cglab.ca/~michiell/TheoryOfComputation/TheoryOfComputation.pdf>>. Acesso em: 7 maio 2019.

MARCHI, J. Dissertação (mestrado), **Navegação de robôs móveis autônomos: estudo implementação de abordagens.** Florianópolis: [s.n.], 2001. 119 f. Centro Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica.

MARINHO, D. L. **Aperfeiçoamento de um robô de sumô autônomo.** 67 f. Monografia (Trabalho de Conclusão de Curso (Graduação)) — Universidade Tecnológica Federal do Paraná, Pato Branco, 2017.

MATARIC, M. J. **Introdução à Robótica.** 1<sup>a</sup>. ed. [S.l.]: Editora Unesp, 2014. ISBN 9788539304905.

- MENDEL, J. M. Fuzzy logic systems for engineering: a tutorial. **Proceedings of the IEEE**, v. 83, n. 3, p. 345–377, mar. 1995. ISSN 0018-9219.
- NETO, A. M. Dissertação (mestrado), **Navegação de robôs autônomos baseada em monovisão**. Campinas: [s.n.], 2007. 110 f. Faculdade de Engenharia Mecânica. Programa de Pós-Graduação em Engenharia Mecânica.
- NOVEL, B.; CAMPION, G.; BASTIN, G. Control of nonholonomic wheeled mobile robots by state feedback linearization. **I. J. Robotic Res.**, v. 14, p. 543–559, dez. 1995.
- OLSON, E. A primer on odometry and motor control. MIT, p. 1–15, jun. 2009.
- ORIOLO, G. Wheeled robots. In: BAILLIEUL, J.; SAMAD, T. (Ed.). **Encyclopedia of Systems and Control**. London: Springer London, 2013. p. 1–9. ISBN 978-1-4471-5102-9. Disponível em: <[https://doi.org/10.1007/978-1-4471-5102-9\\_178-1](https://doi.org/10.1007/978-1-4471-5102-9_178-1)>. Acesso em: 1 maio 2019.
- OTTONI, G. de L. **Planejamento de Trajetórias para Robôs Móveis**. 82 f. Monografia (Trabalho de Conclusão de Curso (Graduação)) — Universidade Federal do Rio Grande, Rio Grande, 2000.
- PASSINO, K.; YURKOVICH, S. **Fuzzy Control**. [S.l.]: Addison-Wesley, 1998. ISBN 9780201180749.
- PETRY, M. L. **Controle híbrido de um robô autônomo seguidor de linha**. 82 f. Monografia (Trabalho de Conclusão de Curso (Graduação)) — Universidade Tecnológica Federal do Paraná, Pato Branco, 2016.
- ROSS, T. J. **Fuzzy Logic with Engineering Applications**. [S.l.]: Wiley, 2009. ISBN 9780470748510.
- SHARP CORPORATION. **GP2U0A21YK0F**. [S.l.], Dezembro de 2006.
- SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots**. [S.l.]: Bradford Company, 2004. ISBN 026219502X.
- TEXAS INSTRUMENTS INCORPORATED. **Tiva™ TM4C1294NCPDT Microcontroller**. [S.l.], Junho de 2014.
- WANG, J. **A Khepera III robot at the Georgia Institute of Technology**. 2008. Disponível em: <[https://en.wikipedia.org/wiki/Khepera\\_mobile\\_robot#/media/File:Khepera\\_III\\_robot.jpg](https://en.wikipedia.org/wiki/Khepera_mobile_robot#/media/File:Khepera_III_robot.jpg)>. Acesso em: 21 maio 2019.
- YAHMEDI, A. A.; FATMI, M. A. Fuzzy logic based navigation of mobile robots. In: TOPALOV, A. E. (Ed.). **Recent Advances in Mobile Robotics**. InTech, 2011. p. 287–310. ISBN 978-953-307-909-7. Disponível em: <<http://www.intechopen.com/books/recent-advances-in-mobile-robotics/fuzzy-logic-based-navigation-ofmobile-robots>>. Acesso em: 28 abr. 2019.
- YUN, X.; TAN, K.-C. A wall-following method for escaping local minima in potential field based motion planning. In: **1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97**. [S.l.: s.n.], 1997. p. 421–426. ISBN 0-7803-4160-0.

YUN, X.; YAMAMOTO, Y. **On Feedback Linearization of Mobile Robots.** 1992. 18 p.  
Disponível em: <[https://repository.upenn.edu/cgi/viewcontent.cgi?article=1524&context=cis\\_reports](https://repository.upenn.edu/cgi/viewcontent.cgi?article=1524&context=cis_reports)>. Acesso em: 20 maio 2019.