Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC

Hugo da Silva e Souza

Controle de velocidade de um robô móvel autônomo utilizando lógica fuzzy

São Carlos
2026

Hugo da Silva e Souza

Controle de velocidade de um robô móvel autônomo utilizando lógica fuzzy

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação .

Área de concentração: Robótica Móvel e Sistemas de Controle

Orientador: Profº Dr. Roberto Santos Inoue

São Carlos
2026

Dedico este trabalho aos meus pais Cleuzeli da Silva e Souza e José Pereira de Souza, por serem, além de minha inspiração, incentivadores e colaboradores para o meu crescimento pessoal e profissional ao longo de toda minha vida.

Agradecimentos

Primeiramente, a Deus e meus mentores espirituais, que me iluminaram na realização deste trabalho.

Aos meus pais Cleuzeli da Silva e Souza e José Pereira de Souza e a minha tia Neuseli da Silva, pela dedicação para a minha formação pessoal e profissional.

Ao meu orientador Prof. Dr. Roberto Santos Inoue, sempre presente, incentivando-me, apoiando-me. Obrigado pela confiança, orientação, tempo e paciência dedicado a essa pesquisa.

Aos meus amigos que fiz durante a graduação que me apoiaram durante a realização deste trabalho.

Aos meus amigos e membros do LARIS pela amizade, suporte, companheirismo durante a realização deste trabalho.

Aos professores e funcionários do Departamento de Computação da Universidade Federal de São Carlos pelas contribuições durante a minha graduação.

Resumo

A crescente utilização de robôs móveis autônomos em aplicações industriais, logísticas e acadêmicas impõe desafios significativos ao controle de velocidade, especialmente em cenários sujeitos a não linearidades, incertezas e variações de carga. Tradicionalmente, esse controle é realizado por meio de controladores clássicos, como o Proporcional–Integral (PI), que apresentam desempenho satisfatório em sistemas lineares, porém limitado em condições operacionais dinâmicas. Nesse contexto, este trabalho propõe o desenvolvimento e a validação de um sistema de controle de velocidade baseado em lógica fuzzy aplicado a um robô móvel autônomo com tração diferencial. O controlador fuzzy foi projetado utilizando um sistema de inferência do tipo Mamdani, com entradas definidas pelo erro e pela variação do erro de velocidade, e saída incremental de controle. A metodologia incluiu a caracterização experimental da planta, a implementação do controlador em sistema embarcado e a realização de ensaios experimentais em diferentes condições de carga. O desempenho do controlador fuzzy foi comparado ao de um controlador PI clássico por meio de métricas como tempo de resposta, erro em regime permanente e integral do erro absoluto (IAE). Os resultados experimentais demonstraram que o controlador fuzzy apresentou maior robustez e melhor desempenho no rastreamento da velocidade, especialmente sob variações de carga, validando a hipótese proposta e evidenciando o potencial da lógica fuzzy para aplicações em robótica móvel.

Palavras-chave: Robô móvel autônomo. Controle de velocidade. Lógica fuzzy. Sistemas de controle. Robótica móvel.

Abstract

The increasing adoption of autonomous mobile robots in industrial, logistics, and academic applications imposes significant challenges on speed control, especially in scenarios subject to nonlinearities, uncertainties, and load variations. Traditionally, speed control is performed using classical controllers, such as the Proportional–Integral (PI) controller, which provides satisfactory performance in linear systems but exhibits limitations under dynamic operating conditions. In this context, this work proposes the development and validation of a fuzzy logic–based speed control system applied to a differential-drive autonomous mobile robot. The fuzzy controller was designed using a Mamdani-type inference system, with the speed error and the variation of the error as input variables and an incremental control output. The methodology included experimental plant characterization, embedded system implementation, and experimental tests under different load conditions. The performance of the fuzzy controller was compared with that of a classical PI controller using metrics such as response time, steady-state error, and the Integral of Absolute Error (IAE). Experimental results demonstrate that the fuzzy controller achieved improved robustness and superior speed tracking performance, particularly under load variations, validating the proposed hypothesis and highlighting the potential of fuzzy logic for mobile robotics applications.

Keywords: Autonomous mobile robot. Speed control. Fuzzy logic. Control systems. Mobile robotics.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AGV	Automated Guided Vehicle – Veículo Guiado Automaticamente
AMR	Autonomous Mobile Robot – Robô Móvel Autônomo
ANFIS	Adaptive Neuro-Fuzzy Inference System
DC	Direct Current – Corrente Contínua
FIS	Fuzzy Inference System – Sistema de Inferência Fuzzy
FLC	Fuzzy Logic Controller – Controlador Lógico Fuzzy
IAE	Integral of Absolute Error – Integral do Erro Absoluto
MATLAB	Matrix Laboratory
PI	Proporcional–Integral
PID	Proporcional–Integral–Derivativo
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
ROS	Robot Operating System
SIF	Sistema de Inferência Fuzzy
SLAM	Simultaneous Localization and Mapping – Localização e Mapeamento Simultâneos
SLR	Systematic Literature Review
TSK	Takagi–Sugeno–Kang

Lista de símbolos

A	Conjunto fuzzy
$e(t)$	Erro de controle (diferença entre a referência e a saída)
F_L	Força aplicada na roda esquerda
F_R	Força aplicada na roda direita
$G(s)$	Função de transferência no domínio de Laplace
I_z	Momento de inércia em torno do eixo vertical
K_d	Ganho derivativo do controlador
K_i	Ganho integral do controlador
K_p	Ganho proporcional do controlador
L	Distância entre as rodas do robô
M	Massa total do robô
$\max(\cdot)$	Operador s-norma máximo
$\min(\cdot)$	Operador t-norma mínimo
$\mu_A(x)$	Função de pertinência fuzzy do conjunto A
r	Raio da roda
s	Variável complexa do domínio de Laplace
$u(t)$	Sinal de controle
U	Universo de discurso
v	Velocidade linear do robô
v_L	Velocidade linear da roda esquerda
v_R	Velocidade linear da roda direita

ω	Velocidade angular do robô
(x, y, θ)	Pose do robô no plano
$\dot{x}, \dot{y}, \dot{\theta}$	Derivadas temporais da pose do robô

Sumário

Capítulo 1

Introdução

1.1 Contextualização

A robótica móvel é uma área da robótica que se dedica ao desenvolvimento de sistemas autônomos, capazes de se deslocar em ambientes variados, interagindo com o meio, realizando tarefas inteligentes e adaptativas (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Com o avanço das tecnologias de automação e de inteligência artificial, e ao contrário dos robôs industriais fixos, os robôs móveis autônomos (Autonomous Mobile Robots – AMRs), operam em ambientes dinâmicos e na maioria das vezes não estruturados, tornando-se muito utilizados em aplicações logísticas, agrícolas, industriais, hospitalares e de segurança (HOFMANN et al., 2023; ABDALAMEER et al., 2023; PăUCă et al., 2024), destacando-se pela tomada de decisão em tempo real com mínima ou nenhuma intervenção humana (CORREIA; SILVA; MOREIRA, 2022; BROOK, 2022)

Dentre as diversas arquiteturas existentes de AMRs, destacam-se os robôs móveis sobre rodas, devido à sua construção simples, baixo custo e elevada mobilidade em diversos cenários do dia a dia. Em especial, os robôs com tração diferencial - compostos por duas rodas independentes e rodízios de apoio - são amplamente utilizados no meio acadêmico e em aplicações práticas, por possuírem movimentação linear e rotacional com controle direto nos motores (KOPCIK; JADLOVSKý, 2017).

A Figura 1 exibe o modelo de AMR da WEG[1] implantado em sua própria fábrica no Brasil, em Jaraguá do Sul, caracterizada pela completa automatização do sistema de

[1] https://www.weg.net/institutional/BR/pt/

fabricação de motores elétricos.

Figura 1 – Exemplo de AMR usado pela WEG do Brasil.

Fonte: WEG (2023)

Um dos principais desafios na navegação de robôs móveis autônomos é o controle preciso da velocidade dos motores que realizam a locomoção. Esse controle impacta fortemente a estabilidade do sistema da plataforma robótica, interferindo na capacidade de seguir uma trajetória e na segurança de operação, principalmente em ambientes em que os distúrbios, como atrito, inclinação ou volume de carga, sofrem alterações constantes. Usualmente, o controle de velocidade é realizado por controladores da teoria clássica, como o Proporcional-Integral-Derivativo (PID), amplamente utilizado na esfera acadêmica e no ramo industrial, devido à sua simplicidade e ao desempenho aceitável em sistemas lineares (ÅSTRöM; MURRAY, 2010). Contudo, esse tipo de controlador apresenta um desempenho limitado quando utilizado em sistemas não lineares ou que apresentam muitas perturbações externas, contexto comum na operação de robôs móveis autônomos.

Nesta conjuntura, a lógica *fuzzy*, introduzida inicialmente por Zadeh (1965), baseada em regras linguísticas e raciocínio aproximado, permite incorporar conhecimento heurístico ao projeto de desenvolvimento de controladores, dispensando uma modelagem matemática altamente precisa do sistema em questão. Essa abordagem demonstra-se particularmente vantajosa para lidar com sistemas incertos e altamente dinâmicos, oferecendo maior robustez e flexibilidade em comparação aos métodos tradicionais (ROSS, 2010; WANG, 1997). Saffiotti et al. (2018) aponta a eficácia dos sistemas de controle *fuzzy* em aplicações robóticas, incluindo o controle de velocidade em robôs móveis autônomos, demonstrando sua capacidade de melhorar o desempenho em condições operacionais reais.

1.2 Motivação

A crescente demanda do mercado por soluções logísticas autônomas tem impulsionado o desenvolvimento de robôs móveis autônomos capazes de operar de forma segura, eficiente e adaptativa em ambientes dinâmicos. Plataformas desse tipo são cada vez mais empregadas no transporte de materiais em indústrias, centros de distribuição e hospitais, onde as condições de carga variam constantemente ao longo das operações. Essa variabilidade afeta parâmetros físicos do robô, como inércia e posição do centro de massa, e exige sistemas de controle capazes de ajustar dinamicamente o comportamento da locomoção para manter precisão e estabilidade (BESSA; DUTRA; KREUZER, 2020; LIU; ZHANG; LI, 2022; ALMEIDA; FIGUEIREDO; SANTOS, 2023).

Em aplicações reais, pequenas variações de massa transportada podem causar degradação de desempenho, atrasos de resposta ou instabilidade em trajetórias, especialmente em robôs com tração diferencial, nos quais o controle preciso de velocidade é essencial para a correta execução dos planos de navegação. Nesse cenário, torna-se fundamental o uso de técnicas de controle adaptativo e inteligente, como o controle fuzzy, que possuem propriedades de robustez e adaptabilidade diante de mudanças nos parâmetros dinâmicos do sistema. Tais técnicas permitem ao controlador ajustar automaticamente seus ganhos e compensar incertezas externas, garantindo desempenho satisfatório mesmo em condições operacionais não lineares (BESSA; DUTRA; KREUZER, 2020; ALMEIDA; FIGUEIREDO; SANTOS, 2023).

Atualmente, um dos principais frameworks utilizados para navegação autônoma de robôs móveis é o *Navigation2* (Nav2), parte integrante do ecossistema ROS 2 (*Robot Operating System 2*) (MACENSKI et al., 2020; Open Robotics, 2025). O Nav2 é amplamente adotado tanto em pesquisas quanto na indústria, fornecendo uma arquitetura modular para planejamento global e local, controle de trajetória e detecção de obstáculos. Sua interface padrão gera comandos de velocidade linear e angular como saída do sistema de navegação, que devem ser convertidos, via equações cinemáticas, em velocidades individuais das rodas do robô.

Dessa forma, para que um sistema de controle de locomoção seja plenamente compatível com o padrão estabelecido pelo Nav2 e com as demandas de precisão do mercado, ele deve ser capaz de interpretar diretamente esses comandos e traduzi-los em movimentos reais de forma suave e estável. Isso requer um controle de baixo nível robusto, preciso e adaptativo, capaz de compensar variações na carga transportada.

Assim, este trabalho se motiva pela necessidade de projetar e avaliar um sistema de controle de velocidade para robôs diferenciais capaz de manter a precisão de velocidade dos motores sob diferentes condições de carga de transporte, garantindo integração direta com frameworks modernos de navegação autônoma, como o Nav2, e promovendo maior confiabilidade para aplicações industriais e de serviço.

1.3 Objetivos

O objetivo geral deste trabalho é propor, desenvolver e validar um sistema de controle adaptativo de velocidade para um robô móvel autônomo utilizando lógica *fuzzy*, com o propósito de aumentar a exatidão e/ou a precisão da locomoção em comparação com controladores clássicos.

Para atingir esse objetivo, foram definidos objetivos específicos que investigam os principais elementos que compõem o sistema de controle proposto e sua integração à plataforma robótica móvel:

- (OE1) Desenvolver um controlador *fuzzy* capaz de operar sob diferentes condições de carga, considerando as variáveis dinâmicas do AMR, de forma a permitir uma adaptação eficiente do controle de velocidade.

- (OE2) Avaliar e validar o desempenho do controlador fuzzy na plataforma robótica móvel, por meio de testes experimentais que verifiquem sua efetividade em comparação ao controlador clássico proporcional-integral (PI).

A validação da hipótese será realizada por meio de ensaios experimentais, embarcando o software de controle no AMR disponível em laboratório e comparando os indicadores de desempenho obtidos com aqueles do controlador PI, de modo a comprovar os ganhos de precisão e robustez do sistema proposto.

1.4 Delimitações do trabalho

Este estudo abordará uma parcela do hardware e do software do projeto do robô móvel sobre rodas com tração diferencial, englobando somente o sistema de controle de velocidade dos motores, o qual possui maior relevância para a pesquisa.

Os resultados apresentados e discutidos nesta pesquisa derivam de ensaios de testes realizados seguindo uma sistemática estabelecida. A discussão dos resultados concentra-se nas observações, nas métricas e na análise dos dados obtidos durante a execução dos ensaios planejados.

Os ensaios serão conduzidos no âmbito do projeto AMR, desenvolvido pelo Laboratório de Robôs Autônomos e Sistemas Inteligentes (LARIS). Este projeto específico servirá de cenário para a implementação e a execução do compensador proposto. O escopo dos ensaios estará limitado à aplicação do compensador adaptativo *fuzzy* no contexto desse projeto de AMR. O controle de alto nível (planejamento global de rotas e navegação autônoma) está fora do escopo.

1.5 Método de pesquisa e desenvolvimento

Este trabalho de pesquisa foi realizado utilizando o método hipotético-dedutivo. De acordo com LAKATOS e MARCONI (2003), o método se baseia nas etapas de que para um dado problema que se quer analisar, uma solução é proposta. A partir da observação e experimentação, tenta-se refutar a proposta levantada, a chamada hipótese. Se a hipótese for refutada pelos testes é uma nova solução é proposta. Caso contrário, observa-se que não podemos dar certeza a solução para o problema, mas sim apontar fortes indícios que corroboram com a sua veracidade.

Além disso, de acordo com Gil (2008), o método hipotético-dedutivo pode ser sintetizado como:

> *[...] quando os conhecimentos disponíveis sobre determinado assunto são insuficientes para a explicação de um fenômeno, surge o problema. Para tentar explicar as dificuldades expressas no problema, são formuladas conjecturas ou hipóteses. Das hipóteses formuladas, deduzem-se consequências que deverão ser testadas ou falseadas. Falsear significa tomar falsas as consequências deduzidas das hipóteses. Enquanto no método dedutivo se procura a todo custo confirmar a hipótese, no método hipotético-dedutivo, ao contrário, procuram-se evidências empíricas para derrubá-la. (GIL, 2008)*

O desenvolvimento desta pesquisa emergiu de um desafio concreto enfrentado no decorrer de um projeto de engenharia, destacando a necessidade de desenvolver sistemas adaptativos eficazes para veículos inteligentes. Esse desafio serviu de ponto de partida, impulsionando o desenvolvimento de um controlador *fuzzy* destinado a melhorar o desempenho da locomoção de um AMR, com capacidade de adaptação às variações da dinâmica decorrentes da adição de carga de transporte, removendo a limitação da adaptabilidade do controlador PI.

A revisão bibliográfica deve avaliar propostas de implementação de controladores *fuzzy*, preferencialmente desenvolvidos para robôs móveis autônomos sobre rodas. Após a análise do material, será proposta uma sistemática de implementação de controlador para AMR, mais especificamente para a plataforma robótica móvel desenvolvida pelo laboratório LARIS.

Em seguida, será possível realizar a validação, analisando se o controlador *fuzzy* proposto é mais eficiente na plataforma robótica móvel em relação a estratégias clássicas da literatura, como o controlador PI.

1.6 Organização do trabalho

Esse trabalho de conclusão de curso está organizado da seguinte forma. No Capítulo 2, Revisão Bibliográfica, expõem-se os resultados obtidos a partir da revisão integrativa

realizada. Nesta seção, são abordados estudos que tratam da lógica *fuzzy* aplicada à robótica, bem como à robótica em sistemas de controle e em robôs autônomos móveis.

O Capítulo 3, Fundamentação Teórica, apresenta os conceitos de lógica *fuzzy* e de sistemas de controle utilizados e citados para o desenvolvimento do controlador de velocidade do robô móvel autônomo e nas discussões levantadas no texto remanescente da dissertação. Além de expor a descrição da plataforma robótica e ferramentas utilizadas no processo de desenvolvimento e testes.

O Capítulo 4, Proposta do Trabalho, descreve a arquitetura desenvolvida do sistema de controle de velocidade destinado ao projeto do robô móvel autônomo. O capítulo também detalha o projeto do controlador *fuzzy* e a implementação do mesmo na plataforma robótica.

O Capítulo 5, Validação, aborda os ensaios e testes planejados para a validação do sistema de controle de velocidade utilizando a lógica *fuzzy*, também detalhando a execução dos ensaios, oferecendo uma análise aprofundada dos resultados, encerrando em uma discussão conclusiva sobre a hipótese do trabalho.

Por fim, o Capítulo 6, Conclusão, discute as contribuições geradas a partir desta pesquisa, assim como comenta o panorama para futuros estudos originários deste trabalho.

Capítulo 2

Revisão Bibliográfica

2.1 Considerações Iniciais

Este capítulo tem como propósito apresentar a revisão bibliográfica conduzida para embasar teoricamente o desenvolvimento deste trabalho. A seção de estratégia de pesquisa detalha a estratégia metodológica para a busca e seleção de estudos relevantes, a seção de revisão dos trabalhos apresenta os principais trabalhos encontrados como resultado da pesquisa na literatura sobre controle de velocidade em robôs móveis. Por fim, a seção de considerações finais sintetiza os aspectos importantes identificados durante a elaboração da revisão.

2.2 Estratégia de Pesquisa

Para garantir a reprodutibilidade, objetividade e abrangência da revisão bibliográfica, adotou-se a abordagem de Revisão Sistemática da Literatura (SLR), conforme o protocolo proposto por Kitchenham, Budgen e Brereton (2015) e a metodologia PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses), incluindo seu fluxograma de seleção (MOHER et al., 2009).

Para conduzir a Revisão Sistemática da Literatura deste trabalho, adotou-se o procedimento manual, estruturado conforme os princípios metodológicos definidos por Kitchenham, Budgen e Brereton (2015). O processo envolveu as seguintes etapas: definição de uma pergunta de pesquisa clara, elaboração de um protocolo com critérios de inclusão e

exclusão, seleção das bases de dados, construção das strings de busca, extração dos estudos relevantes e síntese dos dados obtidos. Todo o fluxo foi documentado por meio de planilhas eletrônicas organizadas, o que possibilitou a rastreabilidade e a reprodutibilidade do processo de seleção e análise dos artigos.

A estratégia de busca foi formulada com base em termos-chave definidos a partir da questão de pesquisa, utilizando operadores booleanos e ajustes conforme as particularidades de cada base de dados. A string de busca principal foi estruturada da seguinte forma:

("Robô Móvel Autônomo" OR "Autonomous Mobile Robot" OR "AMR") AND ("Controle de Velocidade" OR "Speed Control") AND ("Lógica *Fuzzy*" OR "*Fuzzy* Logic")

A consulta foi realizada nas bases Scopus, Web of Science e IEEE Xplore, abrangendo o período de 2017 a 2024, contudo, com o intuito de selecionar trabalhos atualizados dos últimos cinco anos.

Tabela 1 – Strings de busca utilizadas nas bases selecionadas (2019–2024)

Base de Dados	**String de Busca**	**Resultados Encontrados**
Scopus	`("autonomous mobile robot"OR "mobile robot"OR "AMR") AND ("speed control"OR "velocity control") AND ("fuzzy logic"OR "fuzzy control") AND ("adaptive control"OR "intelligent control")`	**623**
Web of Science	`TS=("autonomous mobile robot"OR "mobile robot"OR AMR) AND TS=("speed control"OR "velocity control") AND TS=("fuzzy logic"OR "fuzzy control") AND TS=("adaptive control"OR "intelligent control")`	**12**
IEEE Xplore	`("autonomous mobile robot"OR "mobile robot") AND ("fuzzy control"OR "fuzzy logic") AND ("speed control"OR "velocity control")`	**24**

Durante a fase de identificação, foram inicialmente identificados 659 estudos primários. Após a remoção de duplicatas e análise dos títulos e resumos, foram selecionados 50 estudos para a etapa de leitura completa. Ao final do processo de elegibilidade, 11 estudos foram considerados relevantes para os objetivos deste trabalho e compuseram o corpus da revisão.

2.3 Revisão dos Trabalhos

O estudo de Yang et al. (2022) propõe um controlador PID adaptativo com lógica *fuzzy* para o controle de velocidade em robôs móveis empregados em investigações científicas, especialmente em ambientes extremos, como o Planalto Tibetano. A pesquisa iniciou-se da constatação de que controladores PID tradicionais enfrentam dificuldades diante da variação dinâmica presente em robôs em ambiente real. Com isso, os autores desenvolvem um controlador *fuzzy* que ajusta dinamicamente os ganhos K_p, K_i e K_d com base no erro de velocidade e na taxa de variação desse erro.

O sistema proposto por Yang et al. (2022) utiliza funções de pertinência triangulares e regras *fuzzy* heurísticas definidas em termos linguísticos, viabilizando uma resposta eficiente sem necessidade de modelagem matemática precisa. Os testes experimentais, realizados com um robô real, demonstram que o controlador *fuzzy*-PID reduz o tempo de resposta em até 60% e minimiza oscilações e erros de regime permanente em comparação ao PID convencional. A faixa de operação analisada incluiu velocidades-alvo de 0,5 m/s e 1,0 m/s, ambas com desempenho superior ao do controlador adaptativo.

A principal contribuição do trabalho do Yang et al. (2022) reside na combinação eficaz entre controle clássico e controle inteligente, especialmente vantajosa para robôs móveis operando em ambientes incertos e desafiadores.

Sharma e Palwalia (2017) propõem uma abordagem de controle adaptativo baseada em lógica *fuzzy* para melhorar o desempenho de controladores PID aplicados ao controle de velocidade de motores de corrente contínua (DC). O estudo parte da constatação de que controladores PID convencionais, embora amplamente utilizados na indústria por sua simplicidade e robustez, apresentam limitações em ambientes com elevada presença de perturbações. Para superar essas deficiências, os autores desenvolvem um controlador PID com autoajuste por meio de regras *fuzzy*, que atualiza dinamicamente os ganhos K_p, K_i e K_d a partir dos erros de velocidade e de suas variações.

A estrutura proposta por Sharma e Palwalia (2017) combina um modelo matemático do motor DC com um sistema de inferência *fuzzy*, utilizando funções de pertinência trapezoidais e regras linguísticas baseadas em cinco níveis de variação. Simulações realizadas no software *MATLAB/Simulink* mostram que o controlador *fuzzy* adaptativo supera o PID convencional. Especificamente, o controlador fuzzy apresentou menor tempo de subida (79,67 ms vs. 94,62 ms), menor *overshoot* (0,46% vs. 8,15%) e menor tempo de acomodação (97,8 ms vs. 353 ms), evidenciando sua superioridade em robustez e desempenho dinâmico.

(SHARMA; PALWALIA, 2017) se destaca por validar, por meio de simulações, a eficácia da lógica *fuzzy* como técnica de ajuste em tempo real para sistemas não lineares, oferecendo uma alternativa promissora ao controle clássico em aplicações com motores DC, com potencial de aplicabilidade em robôs móveis autônomos que demandam controle de velocidade preciso e adaptativo.

Raafi'u et al. (2019) investigam e comparam duas abordagens híbridas de controle – *Fuzzy*-PI e *Fuzzy*-PID – aplicadas ao controle de velocidade de motores DC em um robô móvel de quatro rodas (FWMR). O estudo parte da modelagem da planta por meio da identificação de sistemas, resultando em uma função de transferência que representa o comportamento dinâmico do sistema. Ambos os controladores *fuzzy* utilizam como entradas o erro de velocidade e a variação do erro.

A implementação *fuzzy* de Raafi'u et al. (2019) adota o método de inferência de Mamdani, com funções de pertinência triangulares e regras heurísticas definidas com base no conhecimento especialista. A defuzzificação é realizada pelo método do "Largest of Maximum" (LOM). Resultados de simulação em *MATLAB/Simulink* evidenciam que o *Fuzzy*-PID apresenta desempenho superior: tempo de estabilização de 0,353s e overshoot nulo, contrastando com o Fuzzy-PI, que teve overshoot de 14,26% e tempo de estabilização de 0,494s. A presença do termo derivativo demonstrou-se essencial para reduzir o tempo de resposta e evitar picos indesejados.

(RAAFI'U et al., 2019) reforça a eficácia dos controladores *fuzzy* híbridos com agendamento de ganhos adaptativos, especialmente em sistemas dinâmicos não lineares como robôs móveis, apontando o *Fuzzy*-PID como solução mais robusta frente a variações de carga e perturbações

O estudo de Arumugam, Alagumalai e Srinivasan (2024) apresenta o desenvolvimento de um controlador *Fuzzy* (FLC) para robôs móveis com tração diferencial, visando aprimorar a capacidade de navegação em ambientes complexos. Utilizando a estrutura de inferência *fuzzy* do tipo Mamdani, os autores implementaram um sistema com duas entradas (distância e variação de velocidade) e duas saídas (velocidade linear e angular). O sistema foi projetado para lidar com incertezas e não linearidades inerentes aos modelos dinâmicos de robôs móveis, oferecendo maior robustez diante de obstáculos e variações no ambiente. Foi desenvolvido no software *MATLAB/Simulink*.

A metodologia utilizada por Arumugam, Alagumalai e Srinivasan (2024) inclui a modelagem cinemática e dinâmica do robô, seguida do projeto e da implementação do FLC com base em funções de pertinência otimizadas e em um conjunto de regras heurísticas. Os resultados de simulação evidenciaram desempenho satisfatório em tarefas de rastreamento de trajetória, com ajustes dinâmicos da velocidade das rodas em resposta a erros angulares e de posição. Além disso, foram conduzidos experimentos reais, cujos resultados demonstraram boa concordância com os dados simulados, validando a eficácia do controlador *fuzzy* em cenários práticos. Os autores concluem que o uso da lógica *fuzzy* proporciona controle adaptativo, simplicidade de implementação e maior segurança na navegação.

Maghzaoui, Aridhi e Mami (2023) propõem um sistema de controle baseado em lógica *fuzzy* para regular a velocidade de robôs móveis em ambientes dinâmicos e complexos. O estudo justifica o uso da lógica *fuzzy* como solução eficiente diante da incerteza inerente à

navegação autônoma, especialmente quando métodos tradicionais, como o controle PI, são limitados pela necessidade de modelos matemáticos muito precisos. A proposta consiste na modelagem cinemática e dinâmica do robô de quatro rodas, seguida da implementação de um controlador fuzzy que emprega variáveis linguísticas como distância ("Perto", "Média", "Longe") e ângulo do obstáculo ("Esquerda", "Frente", "Direita"), com inferência baseada em nove regras e defuzzificação pelo método do centro de gravidade.

Os testes em ambiente de simulação realizados por Maghzaoui, Aridhi e Mami (2023), demonstraram que o robô ajusta sua velocidade e trajetória de forma eficiente conforme a proximidade e o ângulo dos obstáculos, realizando as curvas necessárias para desviar dos obstáculos e continuar seu trajeto em linha reta. A comparação com o controle PI destaca a flexibilidade e robustez do sistema *fuzzy* diante de variabilidade e incertezas ambientais. O trabalho reforça a aplicabilidade da lógica *fuzzy* para navegação segura e adaptativa em robôs móveis autônomos.

O estudo de Batti et al. (2022) propõe uma arquitetura de controle baseada em lógica *fuzzy* voltada à navegação autônoma e ao desvio de obstáculos para robôs móveis em ambientes hostis. O artigo descreve a implementação de dois controladores *fuzzy* distintos: um para navegação em direção ao objetivo e outro para evitar colisões, ambos estruturados com regras mínimas no formato Takagi-Sugeno. A abordagem divide o sistema de controle de alto nível em duas unidades: uma responsável pela convergência ao destino e outra pela reação às proximidades de barreiras, utilizando entradas sensoriais provenientes de sensores ultrassônicos localizados nas laterais e parte frontal do robô Pioneer P3DX.

O controlador de navegação calcula o erro angular entre a orientação atual do robô e o objetivo, enquanto o controlador de desvio atua sobre a variação de velocidade das rodas com base nas distâncias detectadas. Com apenas oito regras *fuzzy*, o sistema demonstrou-se eficaz, conforme evidenciado nas simulações realizadas em ambientes de diferentes níveis de complexidade nos softwares *V-REP e MATLAB*. Os resultados demonstraram que o robô conseguiu atingir seus alvos sem colisões, adaptando sua trajetória de forma próxima ao ótimo (BATTI et al., 2022).

A principal contribuição de Batti et al. (2022) reside na simplicidade e na robustez da arquitetura *fuzzy* empregada, que não exige modelagens matemáticas complexas, sendo adequada a ambientes dinâmicos. A proposta destaca-se por sua capacidade de lidar com incertezas e pela fácil implementação prática em plataformas robóticas móveis.

Kafiev, Romanov e Romanova (2020) propõem um sistema de controle baseado em lógica *fuzzy* para veículos guiados automaticamente (Automated Guided Vehicles - AGVs), visando promover paradas suaves e precisas em pontos de carregamento e descarregamento. O artigo destaca os desafios do controle de velocidade em ambientes industriais, especialmente frente a não linearidade, à inércia e aos atrasos de resposta dos motores elétricos, contextos nos quais controladores PID tradicionais demonstram limitações. Os autores modelaram matematicamente um controlador *fuzzy* com base no algoritmo de

inferência de Mamdani, implementado em *MATLAB*. O sistema utiliza duas variáveis linguísticas de entrada — velocidade e distância até o obstáculo — e uma variável de saída — potência do motor, expressa como porcentagem do valor máximo. As funções de pertinência são trapezoidais, e as regras *fuzzy* foram derivadas a partir da experiência de especialistas e de simulações do comportamento humano.

A avaliação experimental utilizada por Kafiev, Romanov e Romanova (2020) demonstrou que o controlador *fuzzy* apresentou melhores desempenhos, em termos de precisão de parada e resposta dinâmica, em comparação com abordagens clássicas. A proposta destaca ainda a flexibilidade do sistema *fuzzy* diante das incertezas e variabilidades do ambiente, reforçando sua aplicabilidade em AGVs industriais. Por fim, os autores sugerem que o modelo pode ser estendido para outras plataformas robóticas, consolidando a lógica *fuzzy* como alternativa viável e eficaz para controle de velocidade em robôs móveis autônomos.

Kumar, Sahasrabudhe e Nirgude (2024) apresenta uma abordagem baseada em lógica *fuzzy* para navegação autônoma em ambientes internos não estruturados. O estudo propõe um sistema de controle baseado em dois controladores fuzzy: um para rastreamento de trajetória (TFLC) e outro para desvio de obstáculos (OAFLC), aplicados ao robô TurtleBot2 em ambiente simulado no software *Gazebo*. A motivação central decorre das limitações dos métodos clássicos que não conseguem lidar com incertezas inerentes ao ambiente, assim, a lógica *fuzzy* foi proposta para proporcionar robustez em decisões mais adaptativas.

Kumar, Sahasrabudhe e Nirgude (2024) utilizam sensores de profundidade (Kinect) para obter dados ambientais, que são processados por meio de inferência *fuzzy* Takagi-Sugeno-Kang (TSK) e defuzzificação por centróide. O TFLC atua com base na distância e no ângulo em relação ao alvo, enquanto o OAFLC considera distâncias para obstáculos em diferentes ângulos. É realizada uma fusão dos resultados dos controladores, via ponderação. Os resultados indicam que o robô conseguiu alcançar diferentes posições-alvo de forma satisfatória, mesmo diante de obstáculos inesperados.

Nguyen et al. (2022) propõem uma abordagem híbrida de controle de movimento para robôs móveis, baseada na integração entre a lógica *fuzzy* e o modelo cinemático de robôs com tração diferencial. O trabalho é motivado pelas limitações dos métodos convencionais baseados apenas em modelos cinemáticos, que não se adaptam bem a variações ambientais nem garantem suavidade entre segmentos de trajetória.

Para contornar essas limitações, os autores incorporaram a lógica *fuzzy* ao controle de velocidade linear e de direção angular, permitindo que o robô acompanhe trajetórias predefinidas com maior robustez. O sistema proposto utiliza como entradas o erro de orientação e a velocidade linear e produz como saída a velocidade angular, que, em conjunto com o modelo inverso de cinemática, determina as velocidades de cada roda.

A metodologia foi validada em (NGUYEN et al., 2022) por meio de simulações em

MATLAB e de experimentos com um robô real, equipado com sensores, como encoders, IMU e LIDAR. Os testes consideraram trajetórias com diferentes perfis (reta, curva e formato de oito) e velocidades variadas. Os resultados experimentais evidenciaram boa aderência à trajetória de referência, com erros de posição e velocidade dentro de limites aceitáveis, mesmo em trajetórias com transições abruptas. A comparação entre simulação e experimento evidenciou a eficácia do sistema *fuzzy* proposto em manter a estabilidade do movimento com baixa oscilação, demonstrando sua viabilidade prática para aplicações em robótica móvel autônoma.

Reguii, Hassani e Rekik (2022) propuseram uma arquitetura de controle híbrida para robôs móveis, combinando lógica *fuzzy* com redes neurais artificiais por meio do sistema ANFIS *(Adaptive Neuro-Fuzzy Inference System)*. O estudo utiliza como plataforma experimental o robô Khepera IV, adotando duas abordagens de controle: uma baseada somente em lógica *fuzzy*, e outra com um sistema *neuro-fuzzy* treinado. O controlador *fuzzy* clássico é estruturado com base no modelo Takagi-Sugeno de ordem zero, operando com duas entradas — a distância ao alvo e o ângulo relativo à direção desejada — e gerando comandos de velocidade para cada roda do robô. A navegação é dividida em dois comportamentos: atração ao alvo e desvio de obstáculos, sendo a troca entre eles condicionada pela proximidade de barreiras detectadas pelos sensores do robô.

O diferencial da proposta de Reguii, Hassani e Rekik (2022) está na aplicação do ANFIS para ajustar automaticamente os parâmetros consequentes das regras *fuzzy*, a partir de dados de simulação, utilizando funções de pertinência do tipo gaussiana e aprendizado híbrido (backpropagation e mínimos quadrados). A estrutura do ANFIS foi replicada em quatro modelos distintos para controlar, separadamente, as velocidades esquerda e direita em cada comportamento.

Os resultados de simulação demonstram que o controlador *neuro-fuzzy* proporcionou trajetórias mais eficientes, suaves e curtas em comparação ao controlador *fuzzy* tradicional, especialmente em ambientes congestionados. Os autores concluem que a combinação entre a capacidade de generalização das redes neurais e a interpretabilidade da lógica *fuzzy* torna o ANFIS uma solução eficaz e promissora para controle de navegação de robôs móveis autônomos em ambientes desconhecidos.

O estudo de Pour, Alsayegh e Jaradat (2022) propõe a implementação de um controlador PID adaptativo baseado em lógica *fuzzy* do tipo 2 (FAPID-T2) para um robô móvel diferencial, visando aprimorar o desempenho do sistema diante de incertezas e perturbações oriundas do ambiente. A abordagem combina a robustez do controle PID com a capacidade da lógica *fuzzy* do tipo 2 de lidar com incertezas nos dados sensoriais e nos parâmetros do sistema. O controlador fuzzy proposto utiliza como entradas o erro e a variação do erro, ajustando dinamicamente os ganhos K_p, K_i e K_d por meio de um sistema de inferência *fuzzy* do tipo 2, composto por cinco funções de pertinência triangulares e 75 regras fuzzy.

As simulações, realizadas por Pour, Alsayegh e Jaradat (2022) utilizando *MATLAB/Simulink*, envolveram trajetórias retilíneas, circulares e cenários com perturbações de torque. Os resultados indicam que, em condições ideais (sem perturbações), o desempenho do FAPID-T2 é semelhante ao do FAPID tipo 1, mas com tempo de subida ligeiramente menor. No entanto, sob perturbações, o controlador do tipo 2 demonstrou vantagens expressivas: menor tempo de subida e de acomodação à velocidade angular, além de uma resposta mais rápida à trajetória desejada. Os ganhos ajustados dinamicamente revelaram maior flexibilidade e adaptabilidade.

Pour, Alsayegh e Jaradat (2022) destaca a relevância da lógica *fuzzy* do tipo 2 para o controle da velocidade de robôs móveis em ambientes incertos, reforçando seu potencial para aplicações em robótica autônoma.

Com base na Tabela 2, observa-se uma predominância do uso de controladores baseados em lógica *fuzzy* Mamdani aplicados ao controle de velocidade e navegação de robôs móveis. Nota-se que a maioria dos trabalhos utiliza modelos Fuzzy-PID adaptativos ou *fuzzy* puros, demonstrando a eficácia dessa abordagem na compensação de não linearidades, variações de carga e incertezas do sistema. Além disso, a presença de estudos com validação experimental evidencia a maturidade e aplicabilidade prática dessas soluções. Nesse contexto, o controlador *fuzzy* Mamdani desenvolvido neste trabalho para o controle de velocidade dos motores de um AMR, testado em condições reais com variação de carga, alinha-se diretamente às tendências observadas na literatura, reforçando a atualidade e pertinência da proposta frente ao estado da arte.

Tabela 2 – Sumarização dos trabalhos revisados

Autores	Modelo de Controle	Tipo de Inferência Fuzzy	Modelo de AMR	Ambiente de Teste	Observações Principais
Yang et al. (2022)	Fuzzy-PID Adaptativo	Mamdani	Robô móvel científico	Experimental real e simulação	Controle de aceleração via borboleta de motor a combustível; robusto contra não linearidades.
Sharma & Palwalia (2020)	Fuzzy-PID Adaptativo	Mamdani	Motor DC	Simulação (MATLAB/Simulink)	Fuzzy ajusta ganhos PID dinamicamente; reduz tempo de acomodação e sobressinal.
Raafi'u et al. (2019)	Fuzzy-PID vs Fuzzy-PI	Mamdani	Robô móvel de 4 rodas	Simulação	Fuzzy-PID elimina overshoot e acelera a resposta (0,35 s).
Arumugam et al. (2024)	Fuzzy	Mamdani	Robô móvel diferencial	Simulação e testes físicos	Usa otimização de funções de pertinência e refinamento de regras; melhora a robustez.
Maghzaoui et al. (2023)	Fuzzy	Mamdani	Robô móvel genérico	Simulação	Controle de velocidade seguro; foco em navegação adaptativa e evasão de obstáculos.
Batti et al. (2022)	Fuzzy	Mamdani	Robô tipo unicycle	Simulação (V-REP e MATLAB)	Dois controladores (navegação livre e desvio de obstáculos); navegação autônoma eficiente.
Kafiev et al. (2020)	Fuzzy	Mamdani	AGV industrial	Teste real em linha automatizada	Controle do motor baseado em distância e velocidade; foco em parada suave e precisa.
Kumar et al. (2024)	Fuzzy tipo 2	Takagi-Sugeno-Kang	TurtleBot2	Simulação (Gazebo)	Dois FLC combinados para navegação e desvio; defuzzificação centróide.
Nguyen & Vu (2022)	Fuzzy	Mamdani	Robô móvel diferencial	Simulação e teste real	Combinação FLC + modelo cinemático melhora o rastreamento de trajetória e de velocidade.
Reguii et al. (2022)	Neuro-Fuzzy	Takagi-Sugeno	Robô Khepera IV	Simulação (MATLAB)	ANFIS ajusta parâmetros fuzzy; trajetória ótima e segura; comparação com FLC convencional

2.4 Considerações Finais

A expressiva quantidade de estudos sobre controle de velocidade em robôs móveis autônomos utilizando lógica *fuzzy* ressalta a relevância do tema. Diversas abordagens vêm sendo utilizadas, incluindo desde controladores *fuzzy* mais simples até modelos híbridos, como *Fuzzy-PID*, *neuro-fuzzy* e controladores adaptativos do tipo 2. Essa abundância comprova o esforço contínuo para aprimorar o desempenho e a robustez dos sistemas de controle em robôs móveis.

A lógica *fuzzy* tem-se mostrado especialmente eficaz em cenários incertos e dinâmicos, comuns à robótica móvel, graças à sua capacidade de lidar com imprecisões nas variáveis de controle e com modelagens complexas dos sistemas. Os artigos revisados demonstram que controladores *fuzzy* melhoram significativamente a estabilidade e a suavidade da resposta do sistema, tanto em simulações quanto em experimentos práticos.

Durante o processo de levantamento bibliográfico, foi identificado um entrave na utilização da string de busca em uma das bases de dados científicas. Muitos artigos foram desconsiderados na triagem inicial por não abordarem da temática de robôs móveis autônomos sobre rodas.

Adicionalmente, nota-se uma lacuna na padronização dos métodos de avaliação de desempenho entre os diferentes controladores *fuzzy* utilizados. Cada trabalho adota métricas de eficiência, ambientes de testes e configurações distintas, o que dificulta uma comparação direta entre eles. Do mesmo modo, são poucas as propostas que exploram estratégias sistemáticas de validação em ambientes reais e não controlados.

Nesse contexto, o desenvolvimento de um controlador de velocidade baseado em lógica *fuzzy*, voltado para aplicação prática em robôs móveis autônomos com tração diferencial, contribui significativamente para o avanço do estado da arte nessa temática. Ao conciliar uma fundamentação teórica sólida com experimentações controladas e sistemáticas, a proposta visa não apenas validar a efetividade do controle *fuzzy*, mas também fornecer subsídios para futuras implementações em sistemas robóticos reais.

Capítulo 3

Fundamentação Teórica

3.1 Considerações Iniciais

Este capítulo apresenta os fundamentos teóricos e os conceitos necessários para a e e o desenvolvimento do sistema de controle de velocidade de um robô móvel autônomo, utilizando lógica *fuzzy*. Inicialmente, serão discutidos os conceitos de robótica, com foco em robótica móvel e em plataformas robóticas. Em seguida, serão discutidos os princípios da teoria dos sistemas de controle, com ênfase nos controladores clássicos e nos desafios associados. Também será dedicada uma seção específica à lógica *fuzzy*, abordando suas origens, principais conceitos e definições, bem e sua aplicação em sistemas de controle. Além disso, será apresentada a descrição da arquitetura da plataforma robótica utilizada, incluindo seus componentes de hardware, bem como as ferramentas computacionais empregadas para simulação, modelagem e desenvolvimento do controlador proposto.

3.2 Fundamentação Teórica

3.2.1 Robótica Móvel

A robótica é um campo multidisciplinar que integra várias áreas de conhecimento, como a mecânica, eletrônica, computação e controle, visando projetar, construir e operar sistemas capazes de executar tarefas físicas repetitivas no mundo real. Entre as diversas subdivisões da robótica, destaca-se a robótica móvel, que se ocupa do estudo e do desenvolvimento de robôs com capacidade de locomoção em ambientes, seja em ambientes fechados ou abertos (SICILIANO et al., 2010).

Um robô pode ser definido, segundo Siegwart, Nourbakhsh e Scaramuzza (2011), como um sistema embarcado composto por sensores, atuadores e unidades de controle que interagem com o ambiente. Dentre os diversos tipos de robôs, os robôs móveis distinguem-se por possuírem mobilidade e a capacidade de alterar sua posição espacial de maneira controlada, por meio de ações autônomas ou pré-programadas. Um robô móvel autônomo é aquele capaz de tomar decisões e adaptar-se ao ambiente sem intervenção humana contínua. Isso envolve processos como percepção, mapeamento, localização, planejamento de trajetória e controle de movimento (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011; BEKEY, 2005).

O tipo de locomoção dos AMRs, segundo (CORKE, 2011), pode ser atribuído à sua plataforma mecânica. Dentre os diversos arranjos possíveis, o mais comum em ambientes acadêmicos e laboratórios é a locomoção sobre rodas, sobretudo com tração diferencial. Nesse modelo, a plataforma robótica possui duas rodas motrizes independentes, dispostas paralelamente de cada lado do chassi, e um conjunto de rodas livres para sustentação, comumente chamadas de rodízios. Essa configuração, por sua simplicidade mecânica, permite a realização de manobras de locomoção, como curvas ou rotações no seu próprio eixo, por meio do controle diferencial das velocidades angulares das rodas.

3.2.1.1 Modelagem Cinemática

A modelagem cinemática descreve o movimento da plataforma robótica móvel no plano, considerando o centro geométrico entre as rodas como ponto de referência. Supondo um robô com tração diferencial, com distância L entre as rodas, e velocidades v_R e v_L nas rodas direita e esquerda, respectivamente, tem-se:

$$v = \frac{v_R + v_L}{2}, \qquad \omega = \frac{v_R - v_L}{L}. \tag{1}$$

Essas expressões definem a velocidade linear v e a velocidade angular ω do robô. A pose do robô no plano (x, y, θ) evolui segundo as equações diferenciais:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = \omega. \tag{2}$$

Essas equações são amplamente utilizadas em algoritmos de navegação e de odometria, permitindo prever a posição e a orientação do robô ao longo do tempo.

3.2.1.2 Modelagem Dinâmica

Para fins de controle, é necessário modelar a dinâmica do robô móvel, ou seja, como as forças e os torques afetam seu movimento. A modelagem dinâmica considera as massas, inércias e forças envolvidas.

A equação dinâmica geral para um robô diferencial, desconsiderando forças externas e atrito lateral, pode ser escrita como:

$$M\dot{v} = \frac{r}{2}(F_R + F_L), \qquad I_z\dot{\omega} = \frac{rL}{2}(F_R - F_L), \tag{3}$$

sendo M a massa total do robô, I_z o momento de inércia em torno do eixo vertical, r o raio das rodas, e F_R e F_L as forças geradas pelos motores nas rodas direita e esquerda

Essas equações relacionam os comandos de torque aos motores com as acelerações lineares e angulares do robô, sendo fundamentais para o planejamento e o desenvolvimento de controladores em tempo real.

Para o controle de velocidade e de orientação de AMRs, exige-se uma compreensão maior de diversos outros fatores, como atrito dinâmico, variações de carga, respostas não lineares do sistema de acionamento de motores. Nesse contexto, técnicas de controle de sistemas são utilizadas no desenvolvimento de soluções embarcadas, podendo-se recorrer a técnicas clássicas, robustas ou inteligentes.

3.2.2 Sistemas de Controle

Sistemas de controle visam analisar e desenvolver mecanismos que regulam variáveis de um sistema dinâmico para mantê-las em parâmetros desejados, mesmo diante de perturbações oriundas do ambiente. Tais sistemas são fundamentais na engenharia, especialmente na robótica, onde o controle de posição, orientação e velocidade é imprescindível (OGATA, 2010).

3.2.2.1 Malha Aberta e Malha Fechada

Na área de sistemas de controle, segundo Ogata (2010), existem duas principais arquiteturas: malha aberta e malha fechada.

- No controle em malha aberta, o sinal de controle é aplicado ao sistema sem feedback. A saída do sistema não é monitorada por sensores, e qualquer perturbação ou erro de modelagem pode comprometer o desempenho.
- No controle em malha fechada ou realimentado, a saída do sistema de controle é medida e comparada ao sinal de referência. A diferença entre esses valores, chamada de erro, é utilizada para compensar no controlador. Essa abordagem torna o sistema mais robusto às incertezas e perturbações externas.

3.2.2.2 Controle PID

O controle Proporcional-Integral-Derivativo, mais conhecido como PID, é uma das técnicas mais empregadas em sistemas de controle, atribuída parcialmente ao seu bom desempenho em uma vasta gama de condições de operação e parcialmente à sua simplicidade funcional que permite aos engenheiros operá-lo de forma simples e direta (DORF; BISHOP, 2011).

A equação geral do PID no domínio contínuo do tempo é:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}, \tag{4}$$

sendo:

- $e(t)$ é o erro: diferença entre a referência e a saída real
- K_p: ganho proporcional — corrige o erro imediatamente
- K_i: ganho integral — corrige erro acumulado no tempo
- K_d: ganho derivativo — antecipa a tendência de erro

Para a implementação de um controlador PID, os três parâmetros de ganho devem ser determinados. Existem muitos métodos disponíveis para determinar valores aceitáveis de ganho do PID, e esse processo é frequentemente chamado de sintonia de PID. Uma abordagem comum para a sintonia é o uso de métodos manuais, nos quais os ganhos são obtidos por tentativa e erro com uma mínima análise analítica utilizando respostas a uma entrada do tipo degrau no sistema, via simulação. Mas existem métodos mais analíticos, dos quais um é conhecido como método de sintonia de Ziegler-Nichols.

3.2.3 Sistemas Fuzzy

A teoria de conjuntos *fuzzy* foi introduzida em 1965 pelo matemático o Lotfi Asker Zadeh, na Universidade da Califórnia em Berkeley, para dar um tratamento matemático a certos termos linguísticos subjetivos como: “aproximadamente”, “em torno de”, marcando um avanço significativo na forma de modelar sistemas complexos e imprecisos. A partir dessa teoria, foi possível desenvolver sistemas capazes de lidar com incertezas e subjetividades presentes em diversas áreas, como automação e controle, classificação de dados, tomada de decisões e visão computacional.

Os sistemas *fuzzy* se baseiam em uma representação computacional que busca imitar a maneira como os seres humanos processam informações ao tomar decisões. Diferentemente dos métodos tradicionais, que operam com lógica binária (verdadeiro ou falso, sim ou não), a lógica *fuzzy*, como já mencionado, permite trabalhar com termos linguísticos subjetivos, como “alto”, “baixo”, “próximo de”, entre outros. A associação de um elemento a um determinado conjunto *fuzzy* é expressa por um grau de pertinência, representado por um valor numérico no intervalo [0, 1], em que 0 indica ausência total de pertinência e 1 indica pertencimento completo ao conjunto (BROWN; HARRIS, 1994; TSOUKALAS; UHRIG, 1997).

Com base nessa teoria, a definição de pertencimento de um elemento a um conjunto *fuzzy* é dada por uma função de pertinência, que generaliza a função característica dos

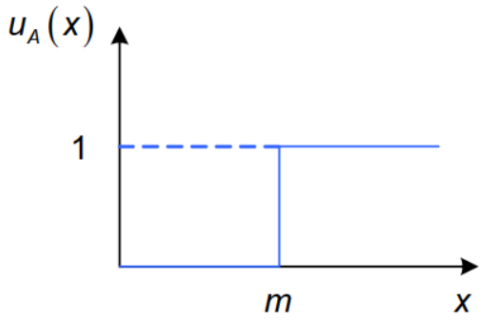
conjuntos clássicos, permitindo modelar a incerteza de forma mais próxima da lógica humana.

Considere um universo de discurso U, um elemento qualquer $x \in U$ e um conjunto clássico $A \subseteq U$. A esse conjunto está associada uma função característica $\mu_A(x) : U \to \{0, 1\}$, definida por:

$$\mu_A(x) = \begin{cases} 1, & \text{se } x \in A \\ 0, & \text{se } x \notin A \end{cases} \tag{5}$$

A função característica representa, de forma binária, a pertinência de um elemento ao conjunto A. Ou seja, um elemento pertence ao conjunto (valor 1) ou não (valor 0). A representação gráfica dessa função, no contexto da teoria clássica dos conjuntos, é usualmente ilustrada por um degrau abrupto, como exemplificado na Figura 2. Nesse modelo, valores inferiores a um limiar m não pertencem ao conjunto A, enquanto os superiores são considerados plenamente pertencentes.

Figura 2 – Função característica da teoria clássica dos conjuntos



Fonte: Elaborado pelo autor

Por outro lado, a proposta introduzida por Zadeh em 1965 amplia essa concepção, ao admitir que a associação de um elemento $x \in X$ a um conjunto A não é mais estritamente binária, mas sim graduada. Nesse contexto, define-se a função de pertinência *fuzzy* $\mu_A(x) : X \to [0, 1]$, como segue:

$$\mu_A(x) \in [0, 1], \quad \forall x \in X, \tag{6}$$

sendo $\mu_A(x)$ o grau de pertencimento do elemento x ao conjunto *fuzzy* A, e $X \subseteq U$ o universo de discurso. Dessa forma, a transição entre os elementos pertencentes e os não pertencentes ao conjunto ocorre de forma contínua, permitindo representar formalmente a incerteza e a subjetividade inerentes a muitos problemas reais.

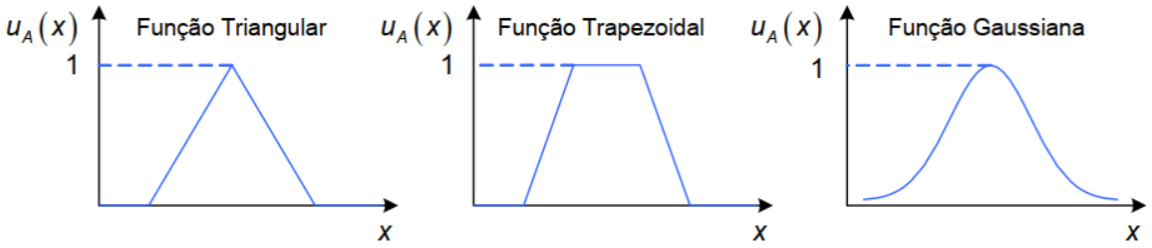
A função que estabelece a relação entre uma variável x e seu grau de pertencimento $\mu_A(x)$ a um conjunto *fuzzy* A é denominada *função de pertinência*. Essa função é responsável

por quantificar, em uma escala contínua de 0 a 1, o nível com que determinado elemento pertence ao conjunto em questão.

Na prática, diversas formas de funções de pertinência são utilizadas, sendo as mais comuns as de formato *triangular*, *trapezoidal* e *gaussiana*, devido à sua simplicidade computacional e capacidade de modelar incertezas eficientemente (BROWN; HARRIS, 1994; TSOUKALAS; UHRIG, 1997; MathWorks, 2025).

A Figura 3 ilustra essas três formas básicas.

Figura 3 – Funções de pertinência dos conjuntos fuzzy



Fonte: Elaborado pelo autor

3.2.4 Sistema de Inferência Fuzzy

Um Sistema de Inferência Fuzzy (SIF), ou simplesmente sistema *fuzzy*, é um sistema computacional que utiliza a lógica nebulosa para mapear um conjunto de entradas numéricas em uma saída numérica. Seu objetivo é emular a capacidade de tomada de decisão humana, que frequentemente se baseia em regras imprecisas e qualitativas.

O processo de inferência de um SIF é realizado conforme o diagrama de blocos apresentado na Figura 4. As variáveis de entrada do processo são resultantes de medições ou aferições, cujos valores não *fuzzy* serão fuzzificados pelo "Fuzzificador"(Seção 3.2.4.1).

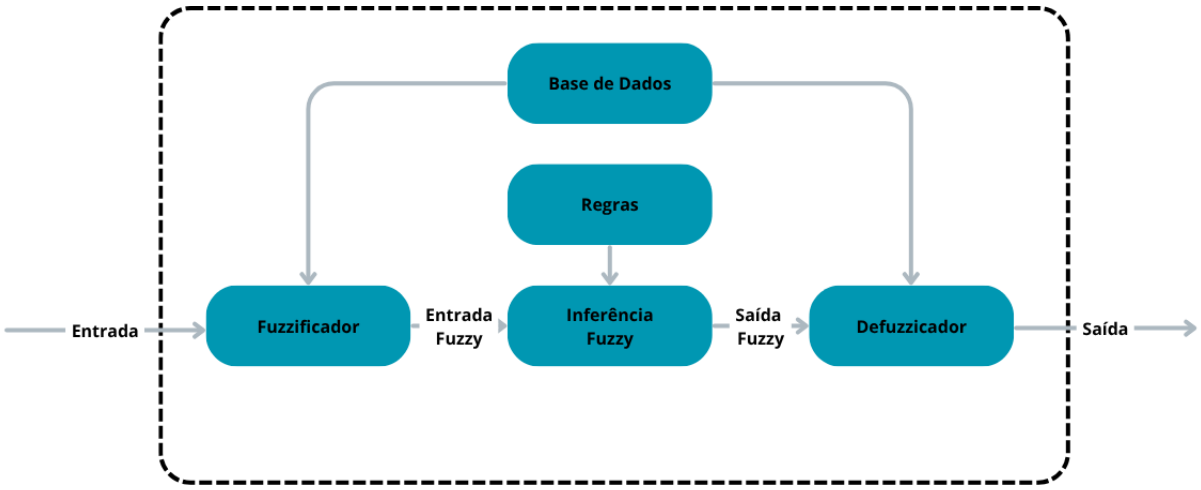
No bloco "Inferência Fuzzy"(Seção 3.2.4.4), as entradas são processadas por meio de um conjunto de regras linguísticas definidas em "Regras"(Seção 3.2.4.2), resultando em uma resposta ou ação de controle na saída.

O valor é defuzzificado no "Defuzzificador"(Seção 3.2.4.5) para que as variáveis de saída sejam fornecidas à planta de controle. Os detalhes de cada bloco serão descritos a seguir.

3.2.4.1 Fuzzificador

As entradas precisas são fornecidas ao sistema *fuzzy* por meio de medições ou observações realizadas por sensores. Nesse contexto, o Fuzzificador é responsável por converter esses valores numéricos em representações linguísticas, identificando a quais variáveis linguísticas pertencem e determinando seus respectivos graus de pertinência nos conjuntos

Figura 4 – Modelo de sistema de inferência fuzzy



Fonte: Elaborado pelo autor

fuzzy. Além disso, essa etapa é responsável por ativar as regras da base de conhecimento associadas às variáveis linguísticas acionadas, iniciando o processo de inferência *fuzzy* (JANG; SUN; MIZUTANI, 1997).

3.2.4.2 Regras

A base de conhecimento de um sistema de inferência *fuzzy* é composta por um conjunto de regras linguísticas estruturadas no formato "Se (antecedente), então (consequente)". Essas regras estabelecem o mapeamento entre as condições de entrada e as ações de saída associadas ao controle do processo. Em geral, são formuladas por especialistas, utilizando sentenças linguísticas que expressam o conhecimento empírico sobre o sistema. A confiabilidade e o desempenho do sistema *fuzzy* estão diretamente relacionados à consistência e à representatividade dessas regras no contexto da aplicação.

Além das regras, a base de conhecimento também inclui as funções de pertinência, que representam os termos linguísticos utilizados nas variáveis de entrada e de saída. Essas funções, que podem assumir formatos triangulares, trapezoidais ou gaussianos, são definidas e parametrizadas pelo projetista do sistema. Embora seja possível realizar a sintonia dessas funções manualmente, é comum a adoção de métodos automáticos. Nesse contexto, a integração entre sistemas *fuzzy* e técnicas de inteligência computacional, como redes neurais artificiais e algoritmos genéticos, tem se mostrado eficaz tanto para a otimização das funções de pertinência quanto para a geração automática de regras (JANG; SUN; MIZUTANI, 1997).

3.2.4.3 Base de Dados

A base de conhecimento armazena as definições das funções de pertinência por meio de discretizações e normalizações dos universos de discurso, representando as variáveis linguísticas de entrada e de saída em forma de partições *fuzzy*. Essa estruturação permite que o processo de inferência *fuzzy* seja realizado de forma computacionalmente eficiente, garantindo precisão e viabilidade na execução dos cálculos envolvidos (JANG; SUN; MIZUTANI, 1997).

3.2.4.4 Inferência Fuzzy

Com base nos conjuntos *fuzzy* e nas regras linguísticas previamente definidas na base de conhecimento, o processo de inferência realiza as operações necessárias para obter da saída *fuzzy*. Esse estágio envolve a ativação das regras linguísticas pertinentes, cujo processamento — conhecido como implicação — gera um conjunto *fuzzy* resultante que representa a agregação das contribuições individuais de todas as regras ativadas. Dentre os principais métodos de implicação *fuzzy* utilizados nesse contexto, destacam-se os propostos por Mamdani, Zadeh e Larsen (BROWN; HARRIS, 1994; TSOUKALAS; UHRIG, 1997), amplamente empregados em sistemas de controle *fuzzy* devido à sua simplicidade e à capacidade de modelar conhecimento humano por meio de regras heurísticas.

3.2.4.5 Defuzzificador

Após a obtenção do conjunto *fuzzy* de saída pelo processo de inferência, torna-se necessário realizar a defuzzificação, etapa necessária para converter a informação representada por conjuntos *fuzzy* em valores precisos. Essa conversão é essencial em aplicações reais, nas quais decisões ou ações concretas exigem valores numéricos específicos e não representações linguísticas imprecisas. Diversas técnicas de defuzzificação são propostas na literatura, sendo as mais recorrentes o método do centro de área (centroid) e o método da média dos máximos (mean of maxima) (BROWN; HARRIS, 1994; TSOUKALAS; UHRIG, 1997; MathWorks, 2025). O método do centro de área determina o valor de saída como o que divide a área sob a curva da função de pertinência em duas partes iguais. Já o método da média dos máximos calcula a média aritmética dos valores extremos no universo de discurso que apresentam os maiores graus de pertinência no conjunto *fuzzy* de saída.

3.2.5 Método de Implicação de Mamdani

O modelo de inferência *fuzzy* proposto por Mamdani caracteriza-se pela utilização de conjuntos *fuzzy* nos consequentes das regras (JANG; SUN; MIZUTANI, 1997). Nesse modelo, a saída da etapa de inferência é um conjunto *fuzzy* resultante da agregação dos consequentes das regras ativas. Posteriormente, esse conjunto é submetido a um processo

de defuzzificação, a fim de obter um valor de saída preciso e aplicável ao sistema de controle.

Uma das principais características do modelo de Mandani é que tanto os antecedentes quanto os consequentes das regras são representados por conjuntos *fuzzy*. Como exemplo, uma regra típica nesse modelo pode ser expressa da seguinte forma:

> **Se** o Erro é "Grande" **e** a Derivada do Erro é "Pequena" **então** o Torque é "Alto".

Quando as regras de inferência envolvem mais de uma variável de entrada, é necessário aplicar técnicas de agregação para combinar os conjuntos antecedentes. Comumente, essa agregação é realizada por meio de t-normas, como o operador mínimo (*min*) ou o produto (*prod*). Após a ativação das regras, são gerados N conjuntos consequentes, um para cada regra ativada. Para formar o conjunto *fuzzy* resultante da etapa de inferência, aplica-se a s-norma (geralmente o operador máximo, *max*) na composição dos N consequentes. Esse conjunto composto é então utilizado na defuzzificação, resultando na saída final do sistema (BROWN; HARRIS, 1994; TSOUKALAS; UHRIG, 1997; MathWorks, 2025).

3.2.6 Plataforma robótica

3.2.6.1 Chassi

A plataforma robótica foi montada com base em um chassi construído a partir de MDF, esquadrilhas de alumínio e barras roscadas, adotando um formato bastante comum em projetos acadêmicos e semelhante aos kits prontos amplamente disponíveis no mercado de robótica móvel, como mostrado na Figura 5. O uso do MDF como material principal proporcionou uma estrutura leve, de baixo custo e de fácil usinagem, enquanto as barras roscadas e as esquadrilhas foram utilizadas para garantir rigidez e possibilitar ajustes na altura entre os dois níveis da estrutura. Esse tipo de montagem modular facilita a fixação de componentes eletrônicos, como *drivers*, motores, sensores, microcontroladores e baterias, além de permitir futuras modificações no projeto.

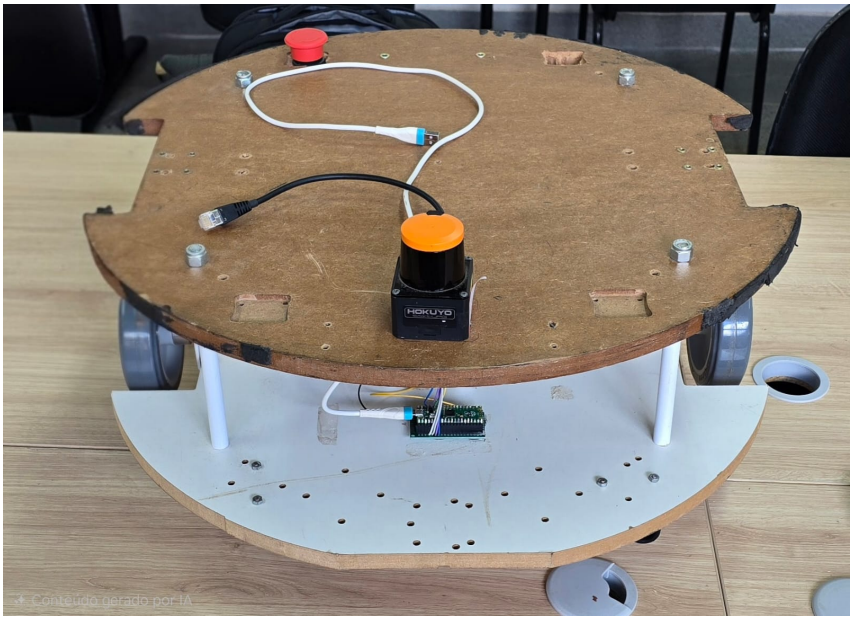
O layout escolhido segue um padrão já consolidado em ambientes educacionais e de pesquisa, favorecendo a integração com rodas, motores de corrente contínua e demais periféricos, mantendo a estabilidade e a simetria da plataforma durante o deslocamento.

3.2.6.2 Atuadores

Foram utilizados na plataforma robótica dois motores IG-42GM [1], de corrente contínua, amplamente empregados em aplicações de robótica móvel devido ao bom desempenho mecânico, à alta eficiência e à robustez.

[1] Datasheet disponível em: <https://www.superdroidrobots.com/store/robot-parts/electrical-parts/encoders-accessories/encoder-motors/product=849>. Acesso em: 07 jan. 2026.

Figura 5 – Chassi fabricado para o AMR



Fonte: Elaborado pelo autor

A versão utilizada no projeto, apresentada na Figura 6, opera com uma tensão nominal de 24 V e possui uma caixa de redução planetária com uma relação de redução de 49:1, permitindo obter um torque elevado e uma rotação controlada na saída, resultando em uma velocidade de aproximadamente 120 RPM.

Figura 6 – Motor IG-42GM

Além disso, o motor possui um *encoder* de quadratura montado na parte traseira do eixo do motor, *encoder* que possui dois canais (A e B) e gera 984 pulsos por rotação (PPR) do eixo do motor, permitindo medições precisas de posição e velocidade angular.

3.2.6.3 Acionamento dos Motores

Para acionar e controlar os motores, foi utilizado um *driver Sabertooth 2x12A* [2], que possui 2 saídas (M1 e M2) para motores de corrente contínua independentes com 12 A de corrente nominal. Em cada saída, foi conectado um motor a cada lado da plataforma. O *driver* também possui quatro modos de comando:

1. Entrada Analógica
   - ❏ Sinais de entrada variando de 0-5V;
   - ❏ Utilizado geralmente com potenciômetro.
2. Controle Remoto
   - ❏ Utiliza canais de rádio controle.
3. Serial Simplificado
   - ❏ Utiliza modo serial RS-232 de comunicação;
   - ❏ Interface PC/Microcontrolador com o *drive*.
4. Serial Empacotado
   - ❏ Utiliza modo serial RS-232 de comunicação;
   - ❏ Pacote contendo um *byte* de endereço, um *byte* de comando, um *byte* de dados e um *checksum*
   - ❏ Interface PC/Microcontrolador com até oito *drivers*

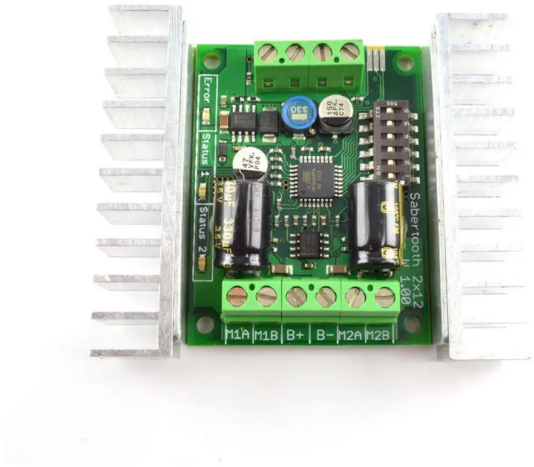
Optou-se pelo modo serial empacotado devido a maior resolução de para cada sentido no qual o motor gira, pelo fato de o pacote separar o sentido do motor por comandos e não no valor do dado enviado.

Como fonte de energia, é possível escolher um modelo de bateria dentre uma vasta gama disponível no mercado, como NiMH, NiCd, LiPo, LiOn e Chumbo Ácido. Para as baterias que possuem lítio em sua composição, o *driver* possui um circuito de proteção que evita o esgotamento da carga e, consequentemente, o dano à mesma. As funcionalidades do *driver* são gerenciadas por 6 chaves seletoras numeradas, para a aplicação desse projeto - com *baudrate* 9600, modo serial empacotado, endereço 128 - a configuração das chaves é a seguinte: Chaves em ON: 3,4,5,6. Chaves em OFF: 1,2.

Desta forma, com apenas um *driver*, é possível controlar os dois motores presentes na plataforma, utilizando lógica de acionamento dos motores, com o serial empacotado em um único canal serial. Nesse modo, os comandos são enviados como pacotes estruturados, contendo três bytes: o endereço do dispositivo, o comando a ser executado e o

[2] Datasheet disponível em: <https://www.dimensionengineering.com/datasheets/Sabertooth2x12.pdf>. Acesso em: 07 jan. 2026.

Figura 7 – Driver Sabertooth 2x12A



valor do parâmetro, seguidos, opcionalmente, por um byte de verificação de integridade (checksum).

O formato básico de um pacote é definido da seguinte forma:

`[Endereço] [Comando] [Valor] [Checksum]`

Cada byte do pacote possui uma função específica:

- **Endereço:** identifica qual Sabertooth no barramento deve receber o comando (útil em sistemas com múltiplos drivers);
- **Comando:** especifica a operação desejada (por exemplo, velocidade do motor 1, velocidade do motor 2, parada, etc.);
- **Valor:** define a intensidade ou o parâmetro associado ao comando, variando entre 0 e 127;
- **Checksum:** soma dos três bytes anteriores (módulo 128), utilizada para detectar erros de transmissão.

3.2.6.4 Microcontrolador

Para agregar os componentes apresentados nas seções anteriores, fez-se necessário a utilização de um microcontrolador Raspberry Pi Pico [3]de baixo custo e alto desempenho com interfaces digitais flexíveis, sendo que os principais recursos, que se destacam para o projeto do trabalho, são: processador de dois núcleos ARM Cortex-M0+ com clock

[3] Datasheet disponível em: <https://pip-assets.raspberrypi.com/categories/610-raspberry-pi-pico/documents/RP-008307-DS-1-pico-datasheet.pdf?disposition=inline>. Acesso em: 07 jan. 2026.

flexível de até 133 MHz, 2MB de memória Flash quad-SPI (QSPI) para armazenamento de código, bibliotecas de ponto flutuante com aceleração nativa em chip, entradas digitais com sinalização evento de borda de subida e descida para realizar função de decodificador de quadratura, e um tamanho compacto.

Adicionalmente, o dispositivo destaca-se pela presença de 26 pinos de entrada e saída (GPIO) multifuncionais, incluindo canais de conversão analógico-digital (ADC) de 12 bits e o inovador sistema de I/O Programável (PIO). Este último permite a execução de máquinas de estados independentes dos núcleos principais, garantindo alta precisão no timing de sinais de controle e leitura de sensores sem causar sobrecarga ao processamento da lógica de controle fuzzy. O microcontrolador é apresentado na Figura 8.

Figura 8 – Raspberry Pi Pico

Comunicação serial com o *driver* de motores

Como mencionado na Seção 3.2.6.4, optou-se pelo modo de comunicação serial simplificado, para envio dos comandos de acionamento dos motores. No código executado na Raspberry Pi Pico, definiu-se inicialmente o pino serial de saída, DRIVER_TX_PIN e também a taxa de transmissão, DRIVER_BAUDRATE, em seguida é aberta a comunicação dos dispositivos utilizando a biblioteca *hardware_uart* disponível no SDK oficial em C/C++ e MicroPython. Dessa forma é possível enviar o caractere referente a velocidade dos motores para o driver, através do comando *uart_putc*.

Comunicação serial via USB

Como a utilização da Raspberry Pi Pico é restrita ao sistema de controle de velocidade do AMR, com foco apenas nos cálculos da malha de controle, foi necessário estabelecer

comunicação com outro equipamento, o qual é o *core* principal da plataforma. Para isso, adotou-se a comunicação via USB proposta por (SGRIGNOLI, 2017). Utilizando o artifício de interrupção serial disponível no microcontrolador, é executado um código responsável pela mensageiria com formato específico - ($xxxxx,yyyyy#). Os caracteres $ e # delimitam o início e o fim da mensagem, respectivamente. O campo "xxxxx"representa a velocidade angular do motor M1, em ponto flutuante, com sinal. De maneira análoga, o campo "yyyyy"corresponde à velocidade do motor M2.

Com os valores extraídos, a estrutura de dados é preenchida com as velocidades correspondentes, que são então armazenadas em uma variável com o número de caracteres definido pelo protocolo de comunicação. Em seguida, a mensagem é enviada via interface USB serial. Na via oposta da comunicação, o microcontrolador recebe essas mensagens contendo as velocidades por meio da USB, utilizando um protocolo semelhante. Após o recebimento, a mensagem é interpretada e os valores obtidos são utilizados como referência pelos controladores responsáveis por cada roda.

3.3 Considerações Finais

Este capítulo apresentou os conceitos teóricos que fundamentam a proposta desse trabalho. Foram abordadas as bases da robótica móvel, os princípios de sistemas de controle com destaque para o controle PID, bem como a introdução e a estrutura dos sistemas *fuzzy*. Também foi descrita a plataforma robótica empregada no desenvolvimento e nos testes do controlador proposto. Esses elementos servirão de suporte teórico à proposta de implementação e validação experimental discutida nos capítulos seguintes.

Capítulo 4

Proposta do Trabalho

4.1 Considerações Iniciais

Este capítulo apresenta a proposta metodológica adotada para o desenvolvimento do controlador de velocidade baseado em lógica *fuzzy* aplicado ao robô móvel diferencial utilizado neste trabalho. Primeiramente, descrevem-se os objetivos e diretrizes gerais do sistema de controle, seguidos do processo de modelagem experimental da planta, etapa essencial para a validação posterior em ambiente de simulação. Em seguida, detalha-se o projeto do controlador *fuzzy*, incluindo a definição das variáveis linguísticas, das funções de pertinência, da base de regras e do mecanismo de inferência. Também é apresentada a implementação do controlador no sistema embarcado, assim como os desafios encontrados durante o processo e as decisões que fundamentaram a versão final do sistema.

Por fim, discute-se como o controlador *fuzzy* proposto se integra à arquitetura geral do robô móvel, destacando a interação entre os módulos de sensoriamento, processamento, comunicação e atuação.

4.2 Desenvolvimento

4.2.1 Definição dos Objetivos do Controle

O principal objetivo do controlador *fuzzy* desenvolvido é garantir que o robô móvel alcance e mantenha as velocidades angulares de referência de cada uma das rodas, conforme os comandos enviados via interface serial. Pretende-se que o controlador apresente boa capacidade de rejeição a distúrbios, tempo de resposta adequado, ausência de oscilações

indesejadas e robustez diante de incertezas inerentes ao sistema, como variações de atrito, carga transportada e flutuações na tensão de alimentação.

A escolha do compensador *fuzzy*, em detrimento de um controlador proporcional-integral (PI) tradicional, justifica-se pela possibilidade de incorporar conhecimento heurístico e de lidar com imprecisões e não linearidades características de sistemas mecatrônicos reais, especialmente motores DC com carga variável e dinâmica dependente da alimentação.

4.2.2 Caracterização da Planta

A modelagem da dinâmica dos motores é uma etapa fundamental para o projeto e a análise de controladores aplicados ao robô móvel. No presente trabalho, optou-se pela identificação experimental da planta, uma vez que o motor IG42GM não possui especificações detalhadas suficientes para a obtenção de um modelo analítico confiável, em especial no que diz respeito às constantes mecânicas, elétricas e às características de atrito.

O procedimento experimental consistiu em aplicar um sinal degrau de referência, em porcentagem do PWM, ao *driver* Sabertooth, configurado em modo serial empacotado, enquanto a velocidade real da roda era medida em tempo real, com o robô suspenso em um cavalete. A leitura de velocidade foi obtida a partir do *encoder* incremental acoplado ao eixo do motor, cujos pulsos foram decodificados em quadratura por meio de interrupções de hardware no microcontrolador Raspberry Pi Pico. O microcontrolador foi responsável tanto pelo envio do comando de excitação quanto pela aquisição dos dados de velocidade, registrados em formato CSV para posterior tratamento.

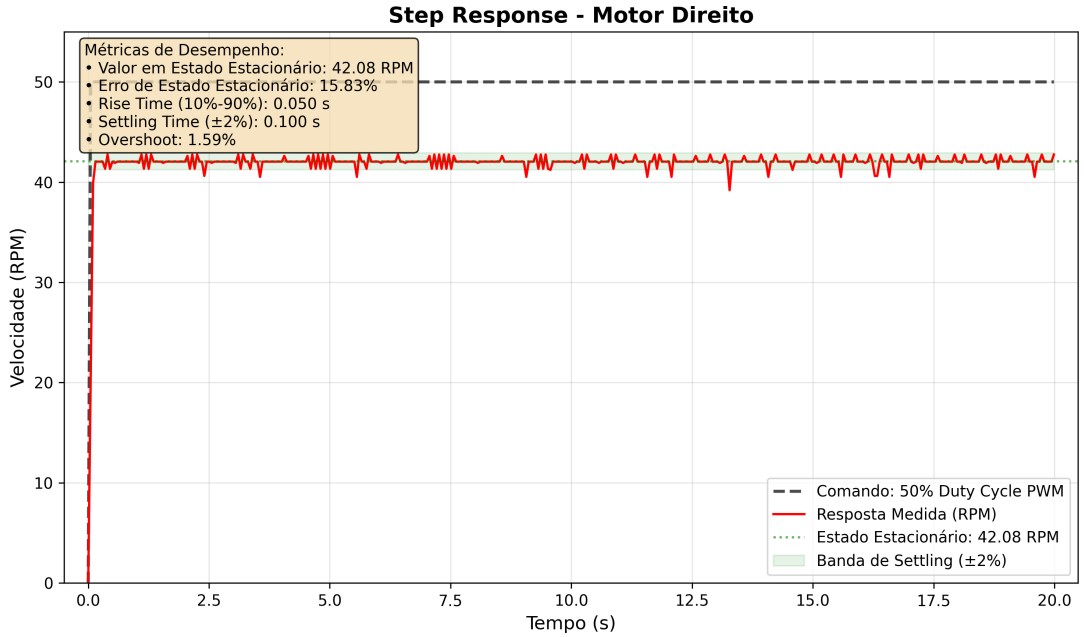
Após a etapa de coleta, os dados foram importados no *MATLAB*, onde se realizou a identificação da função de transferência por meio do *System Identification Toolbox*. Foram testados modelos de primeira e segunda ordem, com e sem atraso de transporte, adotando-se como critério de seleção o menor erro quadrático médio (MSE) e a coerência física do modelo obtido.

A resposta experimental apresentou comportamento monotônico crescente, indicando que a dinâmica do motor pode ser adequadamente representada por meio de um modelo de primeira ordem do tipo:

$$G(s) = \frac{K}{\tau s + 1} \tag{7}$$

sendo, K o ganho estático do sistema e τ a constante de tempo associada ao motor e às perdas mecânicas. A partir da curva ajustada, Figura 9 , observou-se que o modelo obtido apresentou boa correlação com os dados medidos, definido na função de transferência (8), especialmente na região de regime transitório, o que a torna adequada para o uso em simulação e no projeto preliminar do controlador *fuzzy*.

Figura 9 – Resposta ao degrau do motor IG42GM



Fonte: Elaborado pelo autor

$$G(s) = \frac{0.8409}{0.0167\,s + 1} \tag{8}$$

Posteriormente, o processo de identificação da função de transferência com o robô livre foi realizado com o AMR se movimentando no chão, para obter a função de transferência com os efeitos inerciais do chassi, da roda e da carga nominal da plataforma. Como esperado, observou-se uma mudança expressiva no novo modelo, definindo uma nova função de transferência (9), revelando diferenças significativas entre as duas condições de operação. Com os motores sem carga, obteve-se $\tau = 16,7$ ms, enquanto, com o robô no chão, a constante de tempo aumentou para $\tau = 678,8$ ms. O ganho reduziu em 10% devido ao atrito das rodas com o solo, e a constante de tempo aumentou 36 vezes devido à inércia do *chassis*, das rodas e da carga.

$$G(s) = \frac{0.76}{0.6788\,s + 1} \tag{9}$$

Esse processo de identificação permitiu não apenas compreender a dinâmica do motor, mas também identificar o sistema nas condições reais de operação, para obter um ajuste adequado da base de regras do controlador *fuzzy*, garantindo que o compensador projetado respeite as características reais do sistema e opere de forma robusta frente à variabilidade inerente ao AMR.

4.2.3 Projeto do Sistema Fuzzy

O projeto do controlador *fuzzy* de velocidade teve como objetivo principal desenvolver um mecanismo de compensação capaz de corrigir o erro entre a velocidade angular desejada e a velocidade medida das rodas do robô móvel, garantindo uma resposta suave, robusta e estável, mesmo diante de incertezas e dinâmicas não lineares características do sistema eletromecânico utilizado. Para isso, adotou-se um controlador do tipo Mamdani, com estrutura incremental, amplamente empregado em aplicações de controle devido à sua interpretabilidade e à capacidade de incorporar conhecimento heurístico em forma de regras linguísticas.

A estrutura incremental foi escolhida porque motores DC, do tipo 0, requerem ação integral para eliminar o erro em regime permanente. Dessa forma, a saída do controlador *fuzzy* representa o incremento do sinal de controle (Δu), que é acumulado a cada período de amostragem conforme a equação:

$$u[k] = u[k-1] + \Delta u[k] \tag{10}$$

sendo $u[k]$ o sinal de controle aplicado no instante k e $\Delta u[k]$ é o incremento calculado pelo controlador *fuzzy*. Essa abordagem confere ao sistema características semelhantes às de um controlador PI, porém com a flexibilidade e a robustez inerentes à lógica *fuzzy*.

4.2.3.1 Definição das Variáveis Linguísticas

O sistema de controle foi estruturado a partir de duas variáveis de entrada:

- **Erro de velocidade (e)**: diferença entre a velocidade de referência e a velocidade medida, definida como $e = \omega_{ref} - \omega_{medido}$.
- **Variação do erro (Δe)**: derivada discreta do erro, calculada como $\Delta e = (e[k] - e[k-1])/T_s$, que representa a tendência instantânea da dinâmica do sistema.

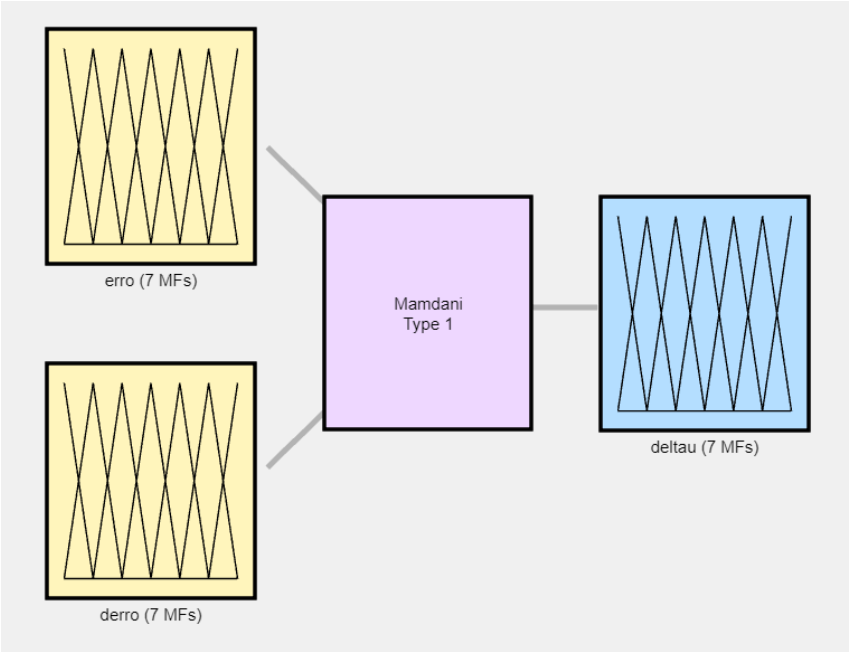
A variável de saída corresponde ao **incremento do sinal de controle *fuzzy*** (Δu), responsável por ajustar o comando enviado ao *driver* Sabertooth, aumentando ou reduzindo a potência aplicada aos motores, conforme a necessidade de correção, como ilustrado na Figura 10.

Para cada variável, foram definidas funções de pertinência do tipo triangular, distribuídas uniformemente em sete regiões linguísticas:

$$\{NG, NM, NS, ZE, PS, PM, PG\} \tag{11}$$

NG (Negativo Grande) e PG (Positivo Grande) representam os extremos de atuação, enquanto ZE (Zero) corresponde à região de equilíbrio próxima da velocidade desejada, como exibido nas Figuras 11, 12 e 13. O uso de sete conjuntos linguísticos oferece maior

Figura 10 – Estrutura do sistema de inferência *fuzzy* (FIS) baseado no modelo Mamdani com saída incremental



Fonte: Elaborado pelo autor

granularidade e suavidade na ação de controle, favorecendo a transição contínua entre diferentes níveis de correção.

Os universos de discurso foram definidos com base nas características operacionais do sistema:

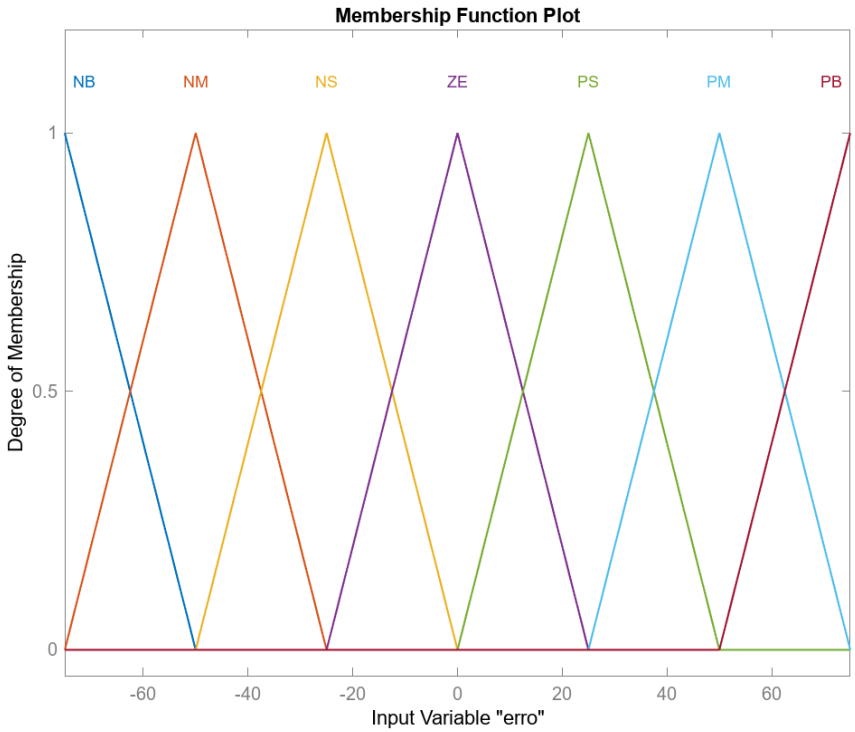
- **Erro (e)**: intervalo $[-75,\ 75]$ RPM, correspondente à faixa de operação dos motores IG42GM.
- **Variação do erro (Δe)**: intervalo $[-6,\ 6]$ RPM/ciclo, dimensionado para o período de amostragem $T_s = 50$ ms.
- **Incremento de controle (Δu)**: intervalo $[-100,\ 100]$ %, que representa a variação percentual máxima do sinal PWM por ciclo de controle.

As funções de pertinência triangulares foram definidas com centros igualmente espaçados e larguras que garantem sobreposição adequada entre conjuntos adjacentes. A Tabela 3 apresenta os parâmetros das funções de pertinência para cada variável linguística.

Tabela 3 – Parâmetros das funções de pertinência triangulares

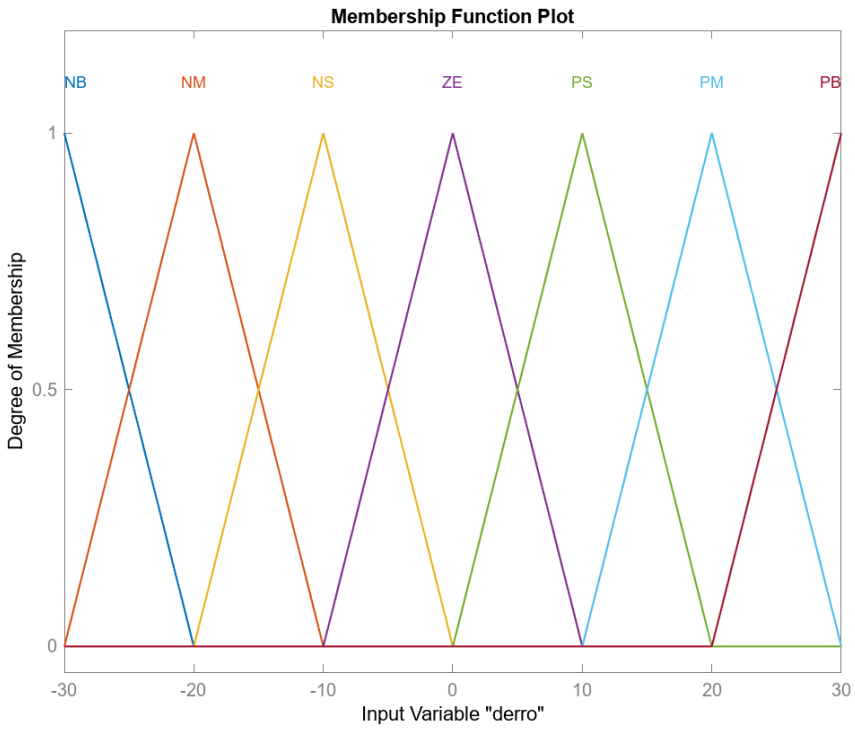
Variável	**Centros**	**Largura**	**Universo**
Erro (e)	$\{-75, -50, -25, 0, 25, 50, 75\}$	25 RPM	$[-75, 75]$
Variação (Δe)	$\{-6, -4, -2, 0, 2, 4, 6\}$	2 RPM/ciclo	$[-6, 6]$
Saída (Δu)	$\{-100, -66.7, -33.3, 0, 33.3, 66.7, 100\}$	33.3 %	$[-100, 100]$

Figura 11 – Funções de pertinência da variável de entrada ”erro”



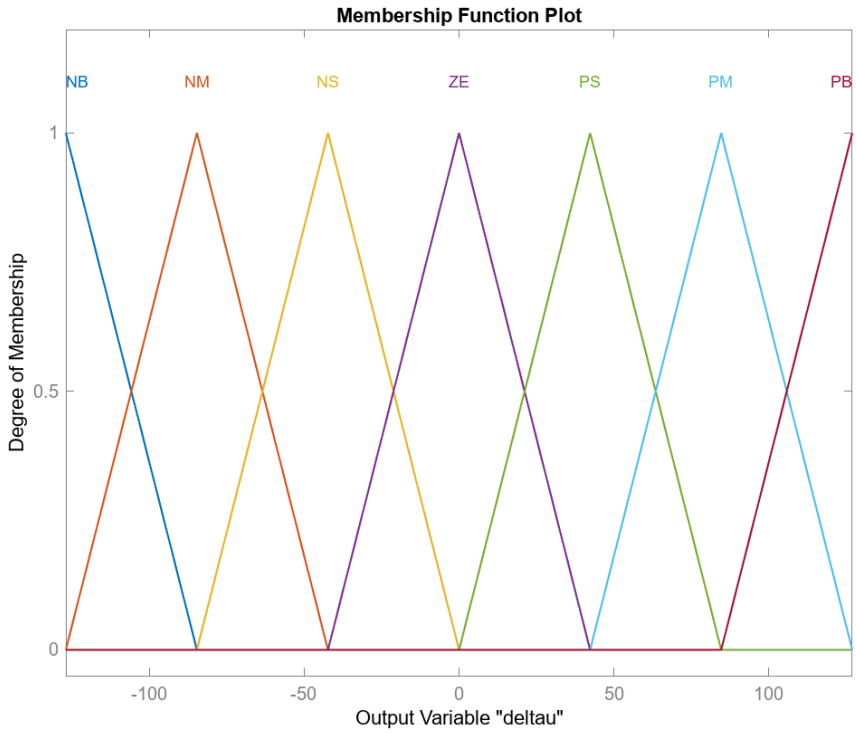
Fonte: Elaborado pelo autor

Figura 12 – Funções de pertinência da variável de entrada “variação do erro”



Fonte: Elaborado pelo autor

Figura 13 – Funções de pertinência da variável de saída “incremento de controle”



Fonte: Elaborado pelo autor

A função de pertinência triangular é definida matematicamente como:

$$\mu(x;\, a,\, b,\, c) = \begin{cases} 0, & x \leq a \\ \dfrac{x-a}{b-a}, & a < x \leq b \\ \dfrac{c-x}{c-b}, & b < x < c \\ 0, & x \geq c \end{cases} \tag{12}$$

em que a, b e c representam, respectivamente, o limite inferior, o centro (pico) e o limite superior da função triangular.

4.2.3.2 Base de Regras Fuzzy

A base de regras foi construída com base em heurísticas clássicas de controladores PD incrementais e em observações experimentais da planta identificada. Cada regra possui a forma genérica:

$$\textbf{Se } e = A_i \textbf{ e } \Delta e = B_j \textbf{ então } \Delta u = C_{ij}$$

em que A_i, B_j e C_{ij} são conjuntos linguísticos associados às funções de pertinência das entradas e da saída. A base final é composta por 49 regras (combinando as sete catego-

rias de cada entrada), cobrindo todo o espaço de operação do controlador e garantindo respostas diferenciadas conforme a magnitude e a tendência do erro.

A Tabela 4 apresenta a matriz de regras implementada, em que as linhas correspondem aos conjuntos de erro e as colunas aos conjuntos da variação do erro.

Tabela 4 – Matriz de regras do controlador *fuzzy*

Erro	**Variação do Erro (Δe)**						
	NG	**NM**	**NS**	**ZE**	**PS**	**PM**	**PG**
NG	NG	NG	NG	NM	NS	NS	ZE
NM	NG	NG	NM	NM	NS	ZE	PS
NS	NG	NM	NS	NS	ZE	PS	PM
ZE	NM	NS	NS	ZE	PS	PS	PM
PS	NM	NS	ZE	PS	PS	PM	PG
PM	NS	ZE	PS	PM	PM	PG	PG
PG	ZE	PS	PS	PM	PG	PG	PG

NG: Negativo Grande; NM: Negativo Médio; NS: Negativo Pequeno; ZE: Zero;
PS: Positivo Pequeno; PM: Positivo Médio; PG: Positivo Grande.

A lógica da tabela de regras segue os seguintes princípios:

- Quando o erro é grande e positivo (velocidade abaixo da referência), o incremento de controle deve ser positivo para aumentar a velocidade.
- Quando o erro diminui rapidamente (variação negativa), a ação de controle pode ser moderada para evitar *overshoot*.
- Quando o erro é próximo de zero e estável, o incremento deve ser o mínimo necessário para manter o equilíbrio.
- A diagonal principal da matriz corresponde à situação de erro e de variação proporcionais, resultando em ações de controle intermediárias.

4.2.3.3 Mecanismo de Inferência e Defuzzificação

Foi utilizado o método de inferência de Mamdani, devido à sua compatibilidade com estratégias heurísticas e à facilidade de interpretação da superfície de controle resultante. As operações *fuzzy* adotadas foram:

- **Operador AND**: mínimo (`min`) entre os graus de pertinência das entradas.
- **Operador OR**: máximo (`max`) entre as contribuições das regras.
- **Implicação**: método do mínimo.
- **Agregação**: máximo entre as contribuições das regras ativadas.

❑ **Defuzzificação**: método do centroide.

O grau de ativação de cada regra R_{ij} é calculado como:

$$\alpha_{ij} = \min\left(\mu_{A_i}(e),\ \mu_{B_j}(\Delta e)\right) \tag{13}$$

A agregação das saídas é realizada pelo operador máximo:

$$\mu_{out}(k) = \max_{i,j}\left(\alpha_{ij}\right) \quad \text{para cada conjunto de saída } C_k \tag{14}$$

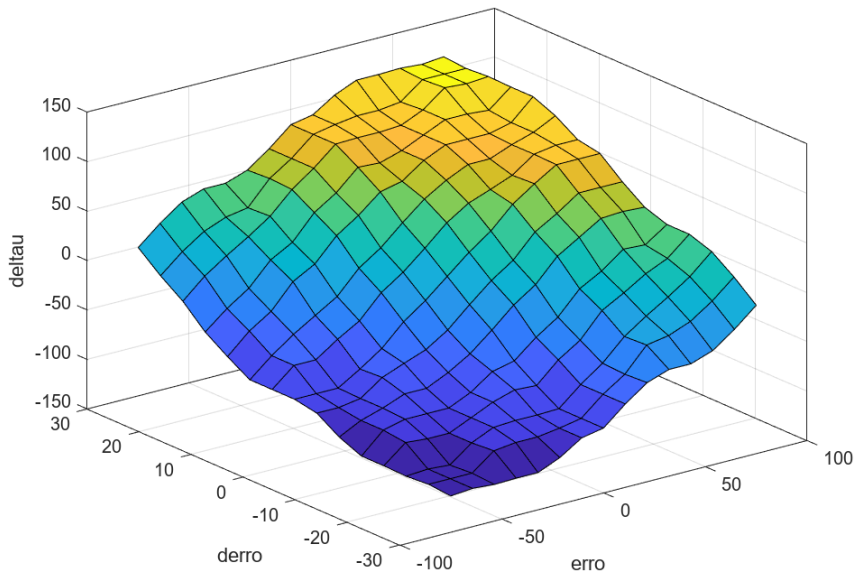
A defuzzificação pelo método do centroide discreto é calculada como:

$$\Delta u = \frac{\sum_{k=1}^{7} \mu_{out}(k) \cdot c_k}{\sum_{k=1}^{7} \mu_{out}(k)} \tag{15}$$

sendo c_k o centro do k-ésimo conjunto de saída e $\mu_{out}(k)$ é o grau de ativação agregado desse conjunto.

A escolha do método do centroide deve-se à sua capacidade de fornecer sinais de controle contínuos, evitando saltos abruptos que poderiam induzir oscilações ou ruídos no acionamento dos motores. Além disso, a implementação discreta do centroide é computacionalmente eficiente, adequada para execução em microcontroladores com recursos limitados.

Figura 14 – Superfície de resposta do sistema *fuzzy* em função do erro e da variação do erro

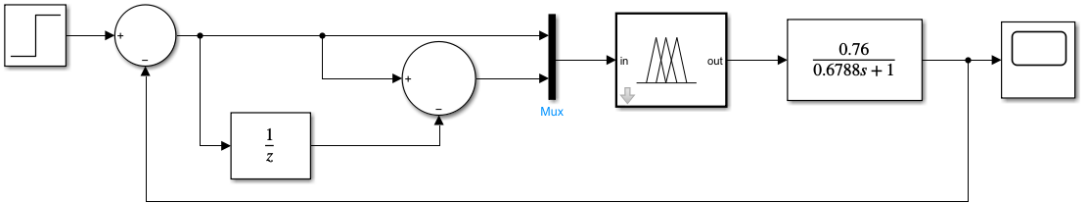


Fonte: Elaborado pelo autor

4.2.3.4 Ajuste e Validação do Controlador

O controlador *fuzzy* foi inicialmente projetado e testado no ambiente MATLAB/Simulink, conforme mostrado em Figura 15, utilizando o modelo da planta identificado experimentalmente. Realizaram-se simulações para diferentes perfis de referência e condições de operação, avaliando-se métricas como o tempo de subida, o erro estacionário, o *overshoot* e a suavidade da resposta. A partir dessas análises, ajustaram-se iterativamente os parâmetros das funções de pertinência e a base de regras, de modo a proporcionar um comportamento dinâmico adequado.

Figura 15 – Diagrama de blocos do controlador *fuzzy* no MATLAB/Simulink para um único motor



Fonte: Elaborado pelo autor

Posteriormente, a lógica do controlador foi transcrita em código em MicroPython e implementada no microcontrolador Raspberry Pi Pico.

4.2.4 Implementação no Sistema Embarcado

A implementação do controlador *fuzzy* no sistema embarcado foi realizada com o microcontrolador Raspberry Pi Pico, programado em MicroPython, responsável pela execução em tempo real das tarefas de leitura dos *encoders*, cálculo das velocidades angulares, aplicação do controlador *fuzzy* e envio dos comandos ao *driver Sabertooth*. O desenvolvimento do código-fonte foi realizado no ambiente Visual Studio Code (VS Code), utilizando extensões específicas para suporte ao MicroPython. O processo de implantação envolveu a compilação e transferência manual dos arquivos para o sistema de arquivos do microcontrolador. A arquitetura de software desenvolvida foi organizada modularmente, permitindo a operação determinística do laço de controle e facilitando futuras manutenções ou expansões do sistema.

4.2.4.1 Aquisição dos Sinais e Cálculo da Velocidade

A leitura dos *encoders* incrementais acoplados aos motores foi realizada por meio de decodificação em quadratura, implementada por meio de interrupções de hardware (IRQ) do Raspberry Pi Pico. Cada canal do *encoder* (A e B) foi configurado para disparar interrupções em ambas as bordas (subida e descida), permitindo a detecção de todas as transições de estado e, consequentemente, máxima resolução na contagem de pulsos.

O algoritmo de decodificação utiliza uma máquina de estados que identifica a direção de rotação com base na sequência de transições dos canais A e B, conforme a Tabela 5.

Tabela 5 – Tabela de transições para decodificação em quadratura

Estado Anterior	**Estado Atual**	**Direção**
00	01	Avanço (+1)
01	11	Avanço (+1)
11	10	Avanço (+1)
10	00	Avanço (+1)
00	10	Recuo (−1)
10	11	Recuo (−1)
11	01	Recuo (−1)
01	00	Recuo (−1)

A velocidade angular de cada roda é calculada a partir da variação da contagem de pulsos em um intervalo de tempo conhecido:

$$\omega = \frac{\Delta p \cdot 60}{CPR \cdot T_s} \quad [\text{RPM}] \tag{16}$$

sendo Δp a variação de pulsos no intervalo, $CPR = 1684$ é o número de pulsos por revolução do *encoder* e $T_s = 0{,}05$ s é o período de amostragem.

Como os sinais provenientes dos *encoders* estão sujeitos a ruídos de alta frequência e pequenas flutuações numéricas, aplicou-se um filtro passa-baixa de primeira ordem sobre a velocidade medida:

$$\omega_f[k] = \alpha \cdot \omega_f[k-1] + (1-\alpha) \cdot \omega[k] \tag{17}$$

sendo $\alpha = 0{,}8$ o coeficiente do filtro.

4.2.4.2 Execução do Controlador Fuzzy

O controlador *fuzzy* previamente projetado em *MATLAB* foi implementado em linguagem MicroPython, adaptada à estrutura de processamento do Raspberry Pi Pico. As funções de pertinência e a base de regras foram convertidas em estruturas de dados estáticas, garantindo baixo custo computacional e execução determinística. O mecanismo de inferência de Mamdani e o método de defuzzificação pelo centroide foram implementados conforme definidos na etapa de projeto, preservando a lógica original utilizada nas simulações.

O laço de controle foi executado a uma taxa fixa de 20 Hz (período $T_s = 50$ ms), definida com base no tempo de amostragem escolhido no projeto do sistema. Em cada iteração, o algoritmo realiza as seguintes etapas:

1. Leitura da velocidade real das rodas a partir dos contadores de pulsos dos *encoders*;

2. Aplicação do filtro passa-baixa nas velocidades medidas;
3. Cálculo do erro de velocidade ($e = \omega_{ref} - \omega_{medido}$);
4. Cálculo da variação do erro ($\Delta e = (e[k] - e[k-1])/T_s$);
5. Fuzzificação das entradas utilizando funções triangulares;
6. Avaliação das 49 regras com operador mínimo para o antecedente;
7. Agregação por máximo e defuzzificação por centroide discreto;
8. Aplicação do fator de escala e acumulação do incremento de controle;
9. Saturação do comando, respeitando os limites físicos dos motores ($\pm 100\%$);
10. Envio dos comandos de velocidade ao *driver* Sabertooth via interface UART.

O sinal de controle final é calculado e limitado conforme:

$$u[k] = \text{sat}\left(u[k-1] + K_s \cdot \Delta u[k],\ -100,\ 100\right) \tag{18}$$

no qual K_s é o fator de escala (tipicamente entre 0,3 e 1,0) utilizado para sintonia fina da resposta do controlador, e a função $\text{sat}(\cdot)$ implementa a saturação nos limites especificados.

Essa abordagem permite que o controlador opere de forma autônoma no microcontrolador, sem depender de uma unidade de processamento superior para manter a estabilidade do sistema.

4.2.4.3 Interface com o Driver Sabertooth

A comunicação com o *driver Sabertooth 2x12A* foi realizada em modo serial empacotado via UART, operando a 9600 baud. O protocolo de comunicação utiliza pacotes de 4 bytes no formato:

`[Endereço] [Comando] [Dado] [Checksum]`

sendo, endereço padrão é 128, o comando indica motor e direção (0/1 para motor 1, 4/5 para motor 2), o dado representa a intensidade (0-127), e o *checksum* é calculado como $(endereco + comando + dado) \wedge 0\text{x}7\text{F}$.

Os valores de saída do controlador *fuzzy* são convertidos de porcentagem (-100% a $+100\%$) para o formato esperado pelo *driver* (0-127), incluindo a seleção automática do comando de avanço ou recuo conforme o sinal:

$$dado = \left\lfloor \frac{|u| \cdot 127}{100} \right\rfloor \tag{19}$$

O modo serial empacotado assegura maior confiabilidade na transmissão e reduz a probabilidade de comandos inválidos devido a ruídos na comunicação.

4.2.4.4 Sistema de Registro de Dados

Para facilitar a análise e validação do controlador, foi implementado um sistema de registro de dados (*logging*) com três modos de operação:

- **Modo FILE**: armazena os dados em um arquivo CSV na memória do Raspberry Pi Pico, adequado para testes curtos com número limitado de amostras.
- **Modo SERIAL**: transmite os dados em tempo real pela porta USB serial, permitindo a captura contínua por um computador externo para testes de longa duração.
- **Modo NONE**: desabilita o registro, reduzindo o tempo de processamento.

No modo SERIAL, cada amostra é transmitida em formato compacto:

```
#timestamp,refL,refR,measL,measR,uL,uR$
```

permitindo taxa de transmissão adequada para o período de amostragem de 50 ms sem impacto significativo no desempenho do laço de controle.

4.2.4.5 Recepção de Comandos de Velocidade

O sistema permite a recepção de comandos de velocidade via interface serial no formato:

```
$+XXX.XXX,+YYY.YYY,A#
```

sendo `XXX.XXX` e `YYY.YYY` representam as velocidades de referência em RPM para as rodas esquerda e direita, respectivamente, e `A` é um indicador de ação. Essa estrutura foi adaptada a partir do código legado de comunicação com ROS, mantendo compatibilidade com sistemas de supervisão e de controle de alto nível existentes.

4.2.4.6 Sincronização e Desempenho Temporal

A estabilidade e o desempenho do laço de controle dependem diretamente da precisão temporal da sua execução. Para esse fim, utilizou-se o temporizador interno do Raspberry Pi Pico (`ticks_ms()`) para garantir a execução periódica do controle. Os tempos de ciclo foram monitorados durante o desenvolvimento, demonstrando que a implementação *fuzzy* em MicroPython apresentou desempenho compatível com a taxa de amostragem de 50 ms estabelecida, mesmo considerando o custo computacional da fuzzificação, inferência e defuzzificação.

A eficiência da implementação no microcontrolador assegurou que o controlador *fuzzy* operasse dentro dos limites de tempo exigidos, sem perdas de iterações ou atrasos perceptíveis, permitindo sua execução confiável em cenários reais de movimentação do robô móvel.

4.2.5 Desafios Encontrados e Decisões Tomadas

Durante o desenvolvimento do controlador *fuzzy* e sua implementação no sistema embarcado, diversos desafios foram identificados, exigindo ajustes sucessivos tanto na modelagem quanto na arquitetura de controle. Esses desafios estão relacionados às limitações físicas dos sensores, às particularidades de acionamento dos motores e às condições reais de operação do robô móvel, especialmente à mecânica interna dos motores. A seguir, detalham-se os principais pontos observados ao longo do processo.

4.2.5.1 Necessidade de Ação Integral para Eliminação do Erro Estacionário

O desafio mais significativo encontrado durante o projeto foi a constatação de que controladores *fuzzy* do tipo PD direto são incapazes de eliminar o erro em regime permanente quando aplicados a sistemas do tipo 0, como é o caso dos motores DC utilizados. Nas primeiras implementações, observou-se que o sistema estabilizava com um erro estacionário significativo, especialmente em velocidades de referência mais baixas.

A solução adotada foi a modificação da estrutura do controlador para o tipo incremental, em que a saída do sistema *fuzzy* representa o incremento do sinal de controle (Δu) em vez do valor absoluto. Essa abordagem introduz implicitamente a ação integral no sistema, permitindo a eliminação do erro em regime permanente, enquanto mantém as vantagens da lógica *fuzzy* para o tratamento de não linearidades e incertezas.

4.2.5.2 Ruído nos Sinais dos Encoders

Outro obstáculo encontrado foi a presença de ruído nos sinais provenientes dos *encoders* incrementais. Embora a decodificação em quadratura por interrupções ofereça alta resolução, constatou-se que pequenas flutuações nos pulsos lidos resultavam em estimativas de velocidade angular com oscilações de alta frequência. Esses ruídos, quando não tratados, afetavam diretamente o cálculo do erro e, principalmente, da variação do erro, resultando em respostas mais agressivas do que o desejado.

Para mitigar esse problema, adotou-se a aplicação de um filtro passa-baixa de primeira ordem nas leituras de velocidade, com coeficiente $\alpha = 0{,}8$. Além disso, a limitação do universo de discurso da variação do erro ($[-6, 6]$ RPM/ciclo) proporcionou robustez adicional contra valores espúrios.

4.2.5.3 Não Linearidades da Dinâmica dos Motores

Outra dificuldade observada diz respeito às não linearidades na dinâmica dos motores IG42GM, especialmente em baixas velocidades e durante mudanças bruscas de carga. O comportamento do motor associado ao *driver* Sabertooth mostrou-se dependente da tensão da bateria, da resistência mecânica imposta pela estrutura do robô e das características do piso. Esses fatores dificultaram a obtenção de um modelo único e representativo e

exigiram um controlador capaz de acomodar variações de ganho e de constante de tempo ao longo da operação.

A solução adotada foi intensificar a granularidade das regras *fuzzy* com sete conjuntos linguísticos por variável, permitindo ações mais precisas quando o erro é pequeno e respostas mais enérgicas quando o sistema se encontra distante da referência. Adicionalmente, o fator de escala K_s permite ajuste fino da agressividade do controlador conforme as condições operacionais.

4.2.5.4 Limitações de Processamento em MicroPython

O Raspberry Pi Pico, embora eficiente, apresenta limitações de processamento quando programado em MicroPython, em comparação com implementações em C/C++. Durante as primeiras implementações, alguns ciclos de controle aproximaram-se do limite do período de amostragem estabelecido, principalmente devido ao custo computacional da avaliar as 49 regras *fuzzy*.

Para garantir determinismo temporal, otimizou-se a estrutura do controlador, utilizando estruturas de dados estáticas para os centros e as larguras das funções de pertinência, implementando a defuzzificação por centroide discreto (mais eficiente que a integração numérica) e evitando a alocação dinâmica de memória durante a execução do laço de controle. Essas otimizações permitiram manter o tempo de execução do controlador *fuzzy* dentro dos limites impostos pelo período de amostragem de 50 ms.

4.2.5.5 Decisões Fundamentadas em Evidências Experimentais

Todas as decisões tomadas durante o desenvolvimento foram guiadas pela análise dos dados experimentais e pela observação direta do comportamento do robô móvel em cenários reais. A combinação entre modelagem teórica, testes práticos e ajustes iterativos mostrou-se essencial para alcançar um desempenho satisfatório. Esse processo reforça a importância de uma abordagem híbrida, que considere tanto fundamentos matemáticos quanto aspectos empíricos da aplicação, especialmente em sistemas mecatrônicos com incertezas intrínsecas.

4.3 Discussão da Arquitetura

A arquitetura de hardware e software do robô móvel desempenha papel fundamental na correta operação do controlador *fuzzy* e, consequentemente, na estabilidade do sistema de controle de velocidade. Esta seção visa apresentar uma visão integrada dos componentes embarcados, dos fluxos de dados e dos níveis hierárquicos de processamento envolvidos, destacando como cada elemento contribui para o desempenho global do sistema.

4.3.1 Arquitetura Geral do Robô Móvel

O robô móvel utilizado neste trabalho apresenta uma arquitetura distribuída composta, essencialmente, por três módulos funcionais principais:

1. **Módulo de Controle de Baixo Nível**: implementado no microcontrolador Raspberry Pi Pico programado em MicroPython, responsável pelo laço de controle de velocidade em tempo real;
2. **Módulo de Potência e Atuação**: composto pelo *driver* Sabertooth 2x12A e pelos motores IG42GM acoplados aos *encoders* incrementais;
3. **Módulo de Supervisão e Comunicação**: responsável pela interface serial para o envio de comandos de referência e o registro de dados.

4.3.2 Níveis de Controle

O sistema apresenta uma organização hierárquica em dois níveis principais:

4.3.2.1 Controle de Alto Nível

Nesse nível, situado fora do microcontrolador de baixo nível, definem-se os comandos de referência de velocidade das rodas. Esses comandos podem ser enviados via interface serial USB, seja por um computador ou por um sistema externo de navegação autônoma desenvolvido em ROS com o Nav2. O controle de alto nível não atua diretamente sobre os motores, mas define o comportamento global do robô, tais como:

- ❏ velocidade linear e angular desejadas;
- ❏ comandos para deslocamento, parada ou manobras;
- ❏ lógica de navegação e tomada de decisão.

4.3.2.2 Controle de Baixo Nível

Localizado no Raspberry Pi Pico, este nível realiza todas as tarefas críticas de tempo real, incluindo:

- ❏ leitura dos *encoders* via interrupções de hardware;
- ❏ cálculo da velocidade angular das rodas com filtragem passa-baixa;
- ❏ avaliação do controlador *fuzzy* incremental;
- ❏ acumulação e saturação do sinal de controle;

- ❏ envio de comandos ao *driver* Sabertooth via UART;
- ❏ registro de dados para análise posterior;
- ❏ envio de dados de odometria para controle de alto nível.

Essa separação permite que o controlador *fuzzy* opere de forma determinística, independentemente das variações na carga computacional do sistema de alto nível.

4.3.3 Fluxo de Dados do Sistema

O processamento interno do robô segue um fluxo de dados bem definido:

1. O módulo de supervisão envia um comando de referência de velocidade para cada roda no formato `$+XXX.XXX,+YYY.YYY,A#`;
2. O Raspberry Pi Pico recebe o comando via USB/UART e atualiza as variáveis de referência internas;
3. A cada ciclo de controle (50 ms), o Pico calcula a velocidade atual a partir da variação dos contadores de pulsos dos *encoders*;
4. As velocidades são filtradas e utilizadas para calcular o erro e a variação do erro;
5. O controlador *fuzzy* recebe essas informações, realiza fuzzificação, inferência e defuzzificação, produzindo um incremento de controle;
6. O incremento é acumulado ao sinal de controle anterior e saturado nos limites permitidos;
7. O sinal é convertido para o formato esperado pelo Sabertooth e transmitido via UART;
8. O Sabertooth ajusta a tensão e corrente enviadas aos motores IG42GM;
9. Os dados de odometria são enviados via USB/UART para o módulo de supervisão;
10. O movimento dos motores produz novos pulsos nos *encoders*, fechando o ciclo de controle.

4.3.4 Comunicação entre os Componentes

A arquitetura de comunicação foi projetada de forma a assegurar robustez e baixa latência, empregando protocolos e interfaces adequados a cada necessidade:

- ❏ **GPIO com IRQ** entre o Raspberry Pi Pico e os *encoders*, garantindo detecção precisa de todos os pulsos com mínima latência;

- **UART** para comunicação com o *driver* Sabertooth a 9600 baud, em modo serial empacotado, assegurando confiabilidade no envio dos comandos de atuação;
- **USB Serial** para recepção de comandos e transmissão de dados de registro vindos do nível superior.

Essas interfaces foram escolhidas considerando o compromisso entre velocidade, simplicidade e estabilidade da comunicação.

4.3.5 Considerações sobre Determinismo e Robustez

A execução do controlador *fuzzy* em tempo real requer controle rigoroso sobre o determinismo da execução. Assim, o temporizador interno do Raspberry Pi Pico foi utilizado para garantir um período fixo de 50 ms para cada iteração do laço. Além disso, cuidados adicionais foram adotados:

- minimização do tempo de processamento em cada ciclo através de otimizações algorítmicas, simplificando a execução dinâmica por estática do sistema de inferência *fuzzy*;
- utilização de estruturas de dados estáticas para evitar alocação dinâmica;
- tratamento de saturação para evitar sobrecomandos ao Sabertooth;
- filtragem das velocidades para aumentar a estabilidade do controlador;
- estrutura incremental que confere robustez intrínseca a variações de parâmetros.

Essas decisões garantem que o sistema continue operando de forma estável mesmo diante de ruídos, variações de tensão ou cargas computacionais esporádicas.

4.3.6 Síntese da Arquitetura

A partir da análise apresentada, observa-se que a arquitetura do robô móvel foi concebida para assegurar um fluxo de controle robusto, modular e determinístico. O microcontrolador Raspberry Pi Pico desempenha papel central ao executar o laço de controle *fuzzy* com precisão temporal, enquanto os módulos de potência e supervisão fornecem suporte estrutural e funcional. A escolha do controlador incremental, aliada às otimizações de implementação em MicroPython, permitiu o desenvolvimento de um sistema flexível, de fácil manutenção e capaz de adaptar-se às incertezas inerentes ao ambiente operacional e à dinâmica dos motores.

4.4 Considerações Finais

Este capítulo apresentou a proposta de desenvolvimento do sistema de controle de velocidade baseado em lógica *fuzzy* para o robô móvel diferencial, abrangendo desde a definição dos objetivos do controle até a caracterização experimental da planta, o projeto do controlador e sua implementação no sistema embarcado.

Destacaram-se a importância da identificação da dinâmica dos motores IG42GM por meio de ensaios em degrau e o papel das simulações no *MATLAB* na definição das funções de pertinência, da base de regras e da estratégia de inferência Mamdani empregadas no controlador *fuzzy*. A escolha da estrutura incremental mostrou-se fundamental para garantir a eliminação do erro em regime permanente, característica essencial para o controle preciso de velocidade em sistemas com motores DC.

O controlador final utiliza 49 regras distribuídas em uma matriz 7×7, com funções de pertinência triangulares uniformemente espaçadas nos universos de discurso adequados às características operacionais do sistema: erro em $[-75, 75]$ RPM, variação do erro em $[-6, 6]$ RPM/ciclo e incremento de controle em $[-100, 100]\%$. A implementação em MicroPython no Raspberry Pi Pico opera a uma taxa de 20 Hz (período de 50 ms), com leitura de *encoders* via interrupções de hardware e comunicação com o *driver* Sabertooth via UART em modo serial empacotado.

Além disso, foram descritos os principais desafios enfrentados ao longo do desenvolvimento — como a necessidade de ação integral, ruídos nos *encoders*, não linearidades dos motores e restrições de tempo real em MicroPython — e as soluções adotadas para garantir robustez e estabilidade no controle.

Por fim, discutiu-se a arquitetura geral do robô móvel, evidenciando a integração entre o microcontrolador, os sensores, os atuadores e o módulo de supervisão. O conjunto dessas etapas estabelece a base metodológica para as análises experimentais e para as validações comparativas entre o controlador *fuzzy* e o controlador PI tradicional, que serão apresentadas nos capítulos subsequentes.

Capítulo 5

Validação Experimental

5.1 Considerações Iniciais

A validação experimental constitui etapa fundamental para demonstrar a eficácia prática do controlador proposto neste trabalho. Embora modelos matemáticos, simulações e análises teóricas forneçam subsídios relevantes para o entendimento do comportamento dinâmico da planta, apenas a experimentação direta no robô móvel permite avaliar, de forma conclusiva, a capacidade do controlador *fuzzy* de lidar com não linearidades, incertezas e perturbações presentes no ambiente real.

Este capítulo descreve detalhadamente o conjunto de ensaios realizados com o robô diferencial do laboratório, incluindo a configuração experimental, o planejamento metodológico, os critérios de desempenho e o processo de coleta de dados. O objetivo central é comparar, sob condições idênticas, o desempenho do controlador *fuzzy* incremental projetado com o controlador PI clássico com *anti-windup*, avaliando qual deles apresenta a melhor resposta no controle de velocidade das rodas sob diferentes condições de operação e carga.

Por fim, os resultados obtidos são analisados criticamente, utilizando métricas quantitativas de desempenho, para verificar a validade da hipótese de que a abordagem *fuzzy* fornece respostas mais robustas e adaptativas à dinâmica real do robô.

5.2 Metodologia de Validação

A validação do controlador *fuzzy* foi conduzida por meio de ensaios práticos realizados com o robô móvel diferencial apresentado nas seções anteriores, onde se buscou avaliar o comportamento dos motores sob diferentes condições de operação. A estratégia consistiu

em submeter o robô a perfis de velocidade padronizados, repetidos múltiplas vezes sob dois regimes de controle: primeiro, utilizando o controlador PI com *anti-windup* ajustado como solução de referência; segundo, aplicando o controlador *fuzzy* incremental desenvolvido ao longo deste trabalho.

Os experimentos foram estruturados com o propósito de analisar, de forma quantitativa, indicadores clássicos de desempenho em sistemas de controle:

- **Integral do erro absoluto (IAE)**: $\int_0^T |e(t)|\, dt$, que quantifica o erro acumulado ao longo do experimento;
- **Integral do tempo pelo erro absoluto (ITAE)**: $\int_0^T t \cdot |e(t)|\, dt$, que penaliza erros persistentes no tempo;
- **Erro em regime permanente (e_{ss})**: diferença percentual entre a referência e a velocidade medida após estabilização;
- **Esforço de controle**: soma das variações absolutas do sinal de controle, indicando a intensidade de atuação.

Adicionalmente, o robô realizou os testes sob três níveis distintos de carga adicional, permitindo avaliar a robustez dos controladores diante de alterações significativas em sua dinâmica mecânica. A análise desses resultados permite verificar se o controlador *fuzzy* atende à hipótese inicial: a de que oferece maior estabilidade, menor erro e maior robustez frente a variações de carga, quando comparado ao PI clássico.

5.3 Descrição dos Experimentos

5.3.1 Controladores Avaliados

Os experimentos envolveram a comparação entre dois tipos de compensadores implementados em MicroPython no Raspberry Pi Pico:

- **Controlador PI com *anti-windup***: implementação clássica com ganhos $K_p = 0{,}1$ e $K_i = 0{,}1$, incluindo o mecanismo de *back-calculation* para prevenir a saturação do integrador. Os ganhos foram ajustados adaptativamente conforme a faixa de velocidade de referência;
- **Controlador Fuzzy incremental**: baseado no modelo de Mamdani com 49 regras, utilizando funções de pertinência triangulares distribuídas em sete conjuntos linguísticos (NG, NM, NS, ZE, PS, PM, PG). Os universos de discurso foram configurados conforme as características operacionais do sistema, com fator de escala $K_S = 0{,}3$ e coeficiente de filtro $\alpha = 0{,}8$.

Ambos os controladores foram executados com período de amostragem $T_s = 50$ ms (frequência de 20 Hz), utilizando um filtro passa-baixa nas leituras de velocidade para atenuar o ruído.

5.3.2 Perfil de Teste de Velocidade

Para avaliar o desempenho dinâmico dos controladores em diferentes pontos de operação, foi definido um perfil de teste composto por degraus sequenciais de velocidade. A sequência de referências utilizada foi:

1. Repouso inicial 0 RPM por 2 segundos;
2. Degrau para 30 RPM, mantido por 8 segundos;
3. Degrau para 60 RPM, mantido por 8 segundos;
4. Retorno para 30 RPM, mantido por 8 segundos;
5. Parada final 0 RPM por 2 segundos.

Esse perfil permite avaliar o comportamento em diferentes pontos de operação, incluindo transições positivas (aceleração), negativas (desaceleração) e paradas completas. A duração total de cada ensaio foi de aproximadamente 28 segundos, com 569 amostras coletadas por execução.

5.3.3 Condições de Carga

Para avaliar o desempenho sob diferentes esforços mecânicos, foram definidos três níveis de carga adicional posicionada sobre a plataforma do robô:

- **Sem carga**: massa total aproximada de 13,7 kg (somente o robô), simulando operação nominal, Figura 16 ;
- **Carga 1**: adição de 5,636 kg, representando condições típicas de transporte, massa total de aproximadamente 19,3 kg, Figura 17;
- **Carga 2**: adição de 11,272 kg, próxima ao limite operacional do robô, com massa total de aproximadamente 25,0 kg, o que impõe maior esforço aos motores, Figura 18.

Para a realização dos ensaios com as cargas mencionadas, foram utilizados conjuntos de baterias estacionárias de 12V disponíveis no laboratório. Estas baterias foram posicionadas de forma distribuída no segundo piso do AMR, permitindo atingir o peso necessário para os testes de desempenho. A variação de carga altera significativamente a dinâmica do sistema, modificando a inércia e o torque necessários à aceleração, o que permite avaliar a capacidade de adaptação de cada controlador a perturbações na planta.

Figura 16 – Protótipo AMR em configuração de teste sem carga (vazio)

Fonte: Elaborado pelo autor

Figura 17 – Protótipo AMR operando com carga 1: 5,636 kg

Fonte: Elaborado pelo autor

Figura 18 – Protótipo AMR operando com carga 2: 11,272 kg

Fonte: Elaborado pelo autor

5.3.4 Procedimentos de Execução

A sequência metodológica adotada para cada experimento foi:

1. Preparação física do robô, instalação das cargas e verificação do nível de tensão das baterias (mínimo 24 V);
2. Configuração do tipo de controlador (`CONTROLLER_TYPE = 'PI'` ou `'FUZZY'`) no código MicroPython;
3. Posicionamento inicial do robô e *reset* dos contadores de *encoders*;
4. Início da captura de dados via interface serial;
5. Execução automática do perfil de teste;
6. Coleta contínua dos dados em formato CSV;
7. Finalização do teste e salvamento do arquivo;
8. Repetição do ensaio para validação estatística.

Cada experimento foi repetido três vezes por condição de carga e por tipo de controlador, totalizando **18 execuções** (2 controladores × 3 cargas × 3 repetições), permitindo o cálculo de médias e desvios-padrão dos indicadores avaliados.

5.3.5 Plataforma Experimental

Os experimentos foram realizados utilizando a seguinte infraestrutura:

- **Plataforma mecânica**: robô diferencial com estrutura em alumínio, motores IG42GM 24 V com redução 49:1 e rodas de 5 polegadas;
- **Sensoriamento**: *encoders* incrementais com resolução de 1684 pulsos por revolução (CPR), leitura via decodificação em quadratura;
- **Controlador embarcado**: Raspberry Pi Pico com *firmware* em MicroPython, executando o laço de controle a 20 Hz;
- ***Driver* de potência**: Sabertooth 2×12A em modo serial, comunicação UART a 9600 baud;
- **Alimentação**: 2 baterias de chumbo-ácido 12V/7Ah em série (24V nominal);
- **Ambiente**: corredor adjacente ao laboratório LARIS com piso de granilite polido. Figura 19

5.3.6 Coleta e Organização dos Dados

Os dados foram transmitidos em tempo real pelo Raspberry Pi Pico via USB serial e capturados pelo computador. Os arquivos CSV resultantes contêm as seguintes variáveis, registradas a cada período de amostragem:

- `time_ms`: *timestamp* em milissegundos;
- `ref_left`, `ref_right`: velocidades de referência em RPM;
- `meas_left`, `meas_right`: velocidades medidas (filtradas) em RPM;
- `u_left`, `u_right`: sinais de controle normalizados (-100% a $+100\%$).

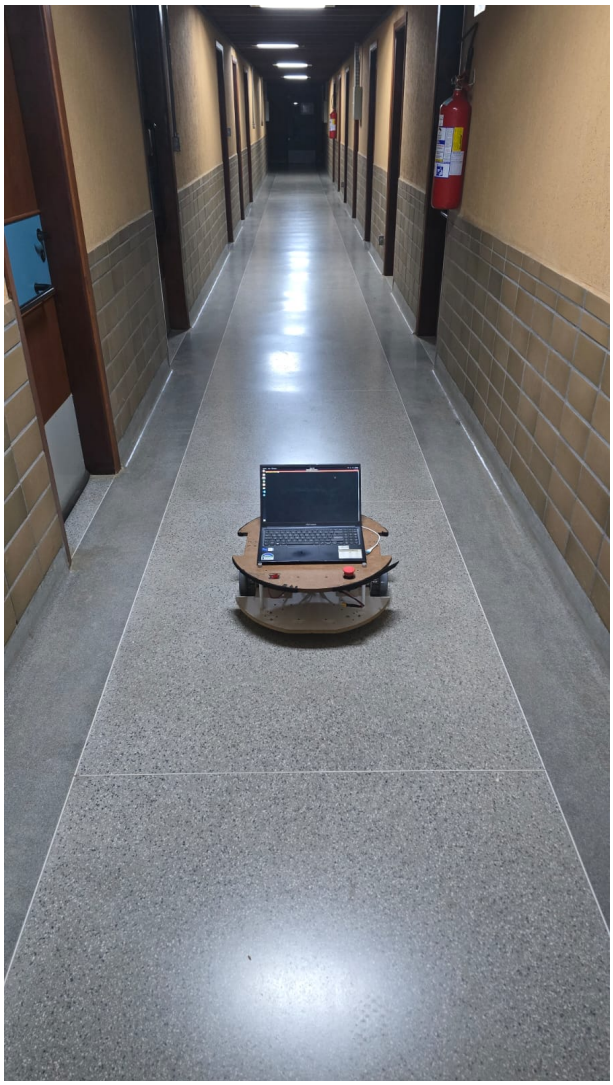
O processamento posterior foi realizado em ambiente Python, utilizando as bibliotecas NumPy, Pandas e Matplotlib para o cálculo das métricas de desempenho e a geração de gráficos comparativos.

5.4 Resultados Obtidos

5.4.1 Resposta Temporal Comparativa

A Figura 20 apresenta a resposta temporal dos controladores PI e *fuzzy* ao perfil de degraus descrito, comparando as três condições de carga. Observa-se que ambos os controladores conseguem rastrear a referência, porém com características dinâmicas distintas.

Figura 19 – Corredor utilizado durante os testes para avaliação de desempenho do controlador no AMR



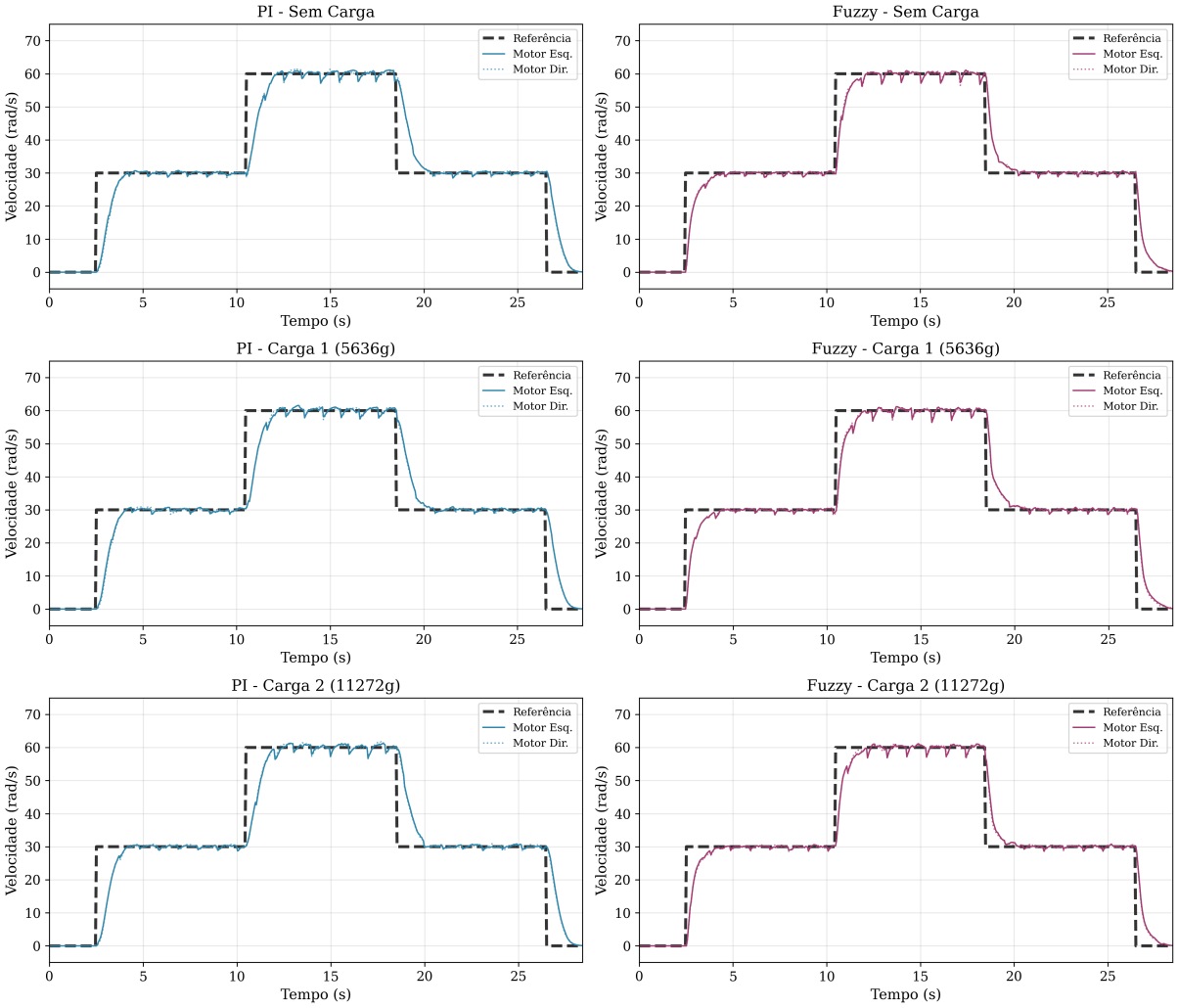
Fonte: Elaborado pelo autor

Visualmente, o controlador *fuzzy* apresenta rastreamento mais próximo da referência em todas as transições, com menor área entre a curva de resposta e o sinal de referência. Essa observação qualitativa é confirmada quantitativamente pelas métricas apresentadas nas seções seguintes.

A Figura 21 apresenta a sobreposição das respostas de ambos os controladores para cada condição de carga, facilitando a comparação direta do desempenho.

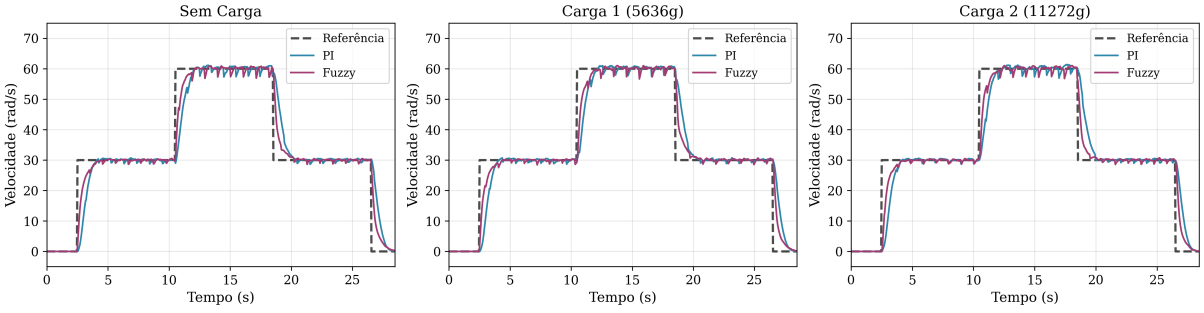
Observa-se, em todas as respostas temporais, a presença de pequenas oscilações de alta frequência na velocidade medida, caracterizadas por um efeito de serrilhado. Esse comportamento está associado ao processo de estimação da velocidade a partir de encoders incrementais, no qual a velocidade é calculada com base na contagem discreta de pulsos em janelas de tempo finitas. Como a contagem de pulsos é quantizada e o período de amostragem é fixo, variações mínimas no número de pulsos registrados a cada amostra

Figura 20 – Comparação da resposta temporal dos controladores PI e *fuzzy* ao perfil de degraus para as três condições de carga



Fonte: Elaborado pelo autor

Figura 21 – Sobreposição das respostas dos controladores PI e *fuzzy* por condição de carga



Fonte: Elaborado pelo autor

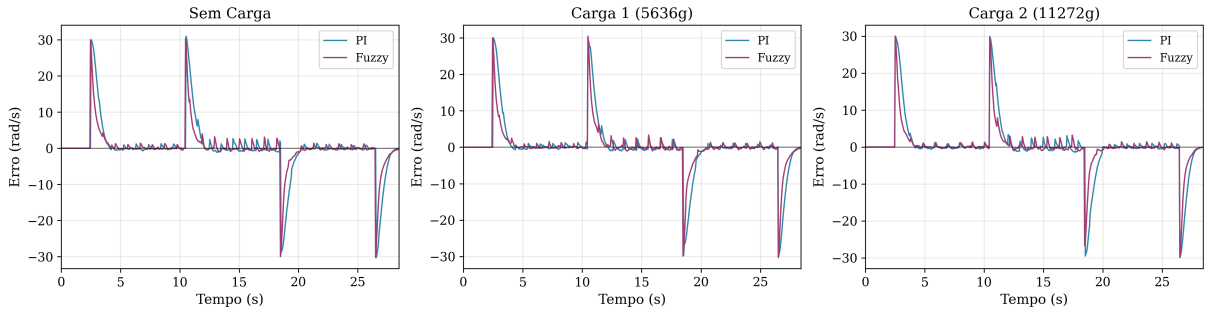
resultam em pequenas flutuações na velocidade estimada, mesmo quando a velocidade real do motor é aproximadamente constante.

Ressalta-se que esse efeito é inerente ao método de medição empregado e não caracteriza instabilidade ou degradação do desempenho dos controladores. Ademais, o fenômeno manifesta-se de forma semelhante em ambos os controladores avaliados, não comprometendo a análise comparativa. As métricas de desempenho utilizadas, baseadas na integração do erro ao longo do tempo, mitigam a influência desse efeito de quantização, garantindo a validade das conclusões obtidas.

5.4.2 Análise do Erro de Rastreamento

A Figura 22 apresenta o erro de rastreamento ao longo do tempo para ambos os controladores nas três condições de carga. O erro é calculado como a média dos erros dos motores esquerdo e direito.

Figura 22 – Comparação do erro de rastreamento dos controladores PI e *fuzzy*



Fonte: Elaborado pelo autor

Observa-se que o controlador *fuzzy* mantém o erro em uma banda mais estreita ao longo de todo o experimento, tanto nas transições quanto em regime permanente. O controlador PI apresenta picos de erro maiores nas transições entre os níveis de velocidade.

5.4.3 Métricas de Desempenho do Perfil Completo

A Tabela 6 apresenta os valores das métricas de desempenho calculadas para o perfil completo de teste (todos os degraus), considerando a média dos três ensaios realizados para cada condição.

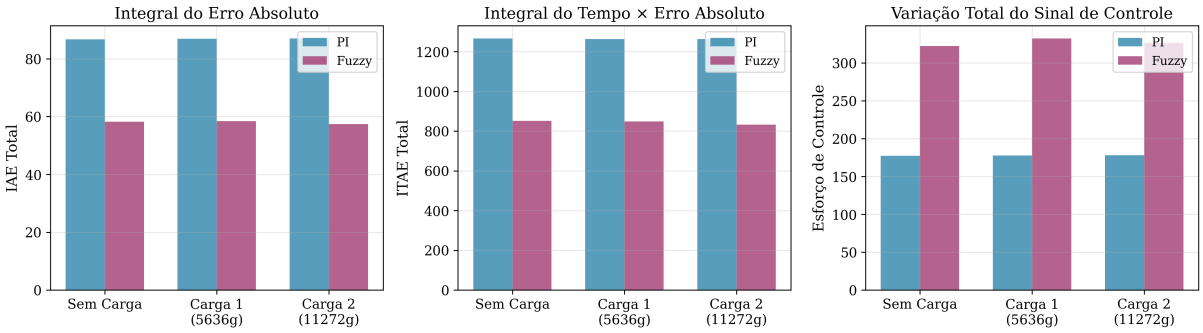
Os resultados demonstram que o controlador *fuzzy* apresenta IAE aproximadamente **33% menor** do que o controlador PI em todas as condições de carga. O erro em regime permanente é semelhante para ambos os controladores (cerca de 1%), indicando que a estrutura incremental do *fuzzy* proporciona uma ação efetiva.

A Figura 23 apresenta, graficamente, a comparação das métricas IAE, ITAE e do esforço de controle entre os controladores.

Tabela 6 – Métricas de desempenho para o perfil completo (valores médios ± desvio padrão)

Controlador	Carga	IAE	ITAE	e_{ss} (%)	Esforço
PI	Sem carga	86,74 ± 0,20	1266,91	1,1	177,3
	5636 g	86,94 ± 0,59	1263,63	1,2	177,7
	11272 g	87,08 ± 0,68	1263,51	1,1	178,2
Fuzzy	Sem carga	58,28 ± 0,28	851,05	1,1	322,5
	5636 g	58,40 ± 0,52	848,85	1,0	332,6
	11272 g	57,47 ± 2,13	833,08	1,0	326,8

Figura 23 – Comparação gráfica das métricas de desempenho por condição de carga



Fonte: Elaborado pelo autor

5.4.4 Análise por Tipo de Degrau

Para uma análise mais detalhada, a Tabela 7 apresenta o IAE calculado para cada transição do perfil de teste, considerando a média de todas as condições de carga.

Tabela 7 – IAE por tipo de degrau (média de todas as condições de carga)

Transição	PI	Fuzzy	Redução
0 → 30 rpm (aceleração)	22,10	14,38	−34,9%
30 → 60 rpm (aceleração)	23,54	15,96	−32,2%
60 → 30 rpm (desaceleração)	20,53	13,26	−35,4%
30 → 0 rpm (parada)	17,23	11,00	−36,2%

O controlador *fuzzy* apresenta redução consistente do IAE em todos os tipos de transição, com ganhos variando de 32% a 36%. A maior redução ocorre na transição de parada (30 → 0 rpm), o que demonstra a eficácia do controlador *fuzzy* em situações que exigem desaceleração controlada.

5.4.5 Análise de Robustez à Variação de Carga

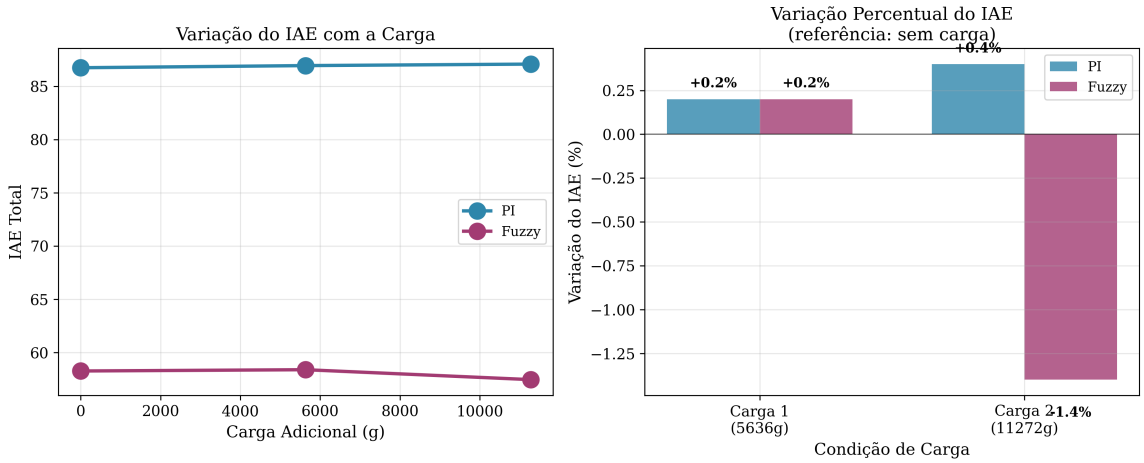
Para quantificar a robustez dos controladores, calculou-se a variação percentual do IAE entre as diferentes condições de carga, utilizando a condição sem carga como referência. Os resultados são apresentados na Tabela 8 e ilustrados na Figura 24.

Tabela 8 – Variação percentual do IAE em relação à condição sem carga

Controlador	Δ Carga 1	Δ Carga 2	Índice de Robustez
PI	+0,2%	+0,4%	0,6%
Fuzzy	+0,2%	**−1,4%**	1,6%

Índice de Robustez = $|\Delta_1| + |\Delta_2|$ (soma das variações absolutas)

Figura 24 – Análise de robustez: variação do IAE com a carga



Fonte: Elaborado pelo autor

Um resultado particularmente interessante é observado no controlador *fuzzy*: seu IAE **diminui** com a carga máxima (−1,4%), indicando uma **melhoria de desempenho** sob maior esforço mecânico. Em contraste, o controlador PI apresenta degradação monotônica (+0,4%) com o aumento da carga.

Essa característica sugere que o controlador *fuzzy* possui capacidade adaptativa implícita, em que suas regras não lineares permitem ajustar a intensidade da ação de controle de forma mais adequada quando a planta apresenta dinâmica mais lenta devido à maior inércia.

5.4.6 Análise do Sinal de Controle

A Figura 25 apresenta uma comparação dos sinais de controle gerados por ambos os controladores durante os testes. O esforço de controle, definido como a soma das variações absolutas do sinal de controle, quantifica a "agressividade" da atuação.

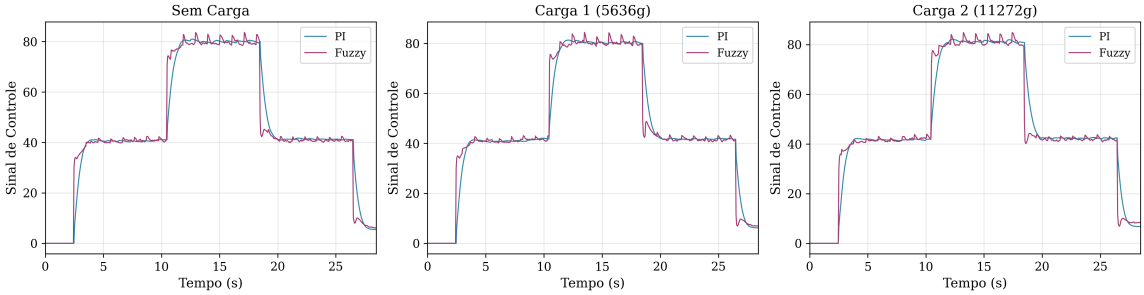


Figura 25 – Comparação dos sinais de controle dos controladores PI e *fuzzy*

Fonte: Elaborado pelo autor

Observa-se que o controlador *fuzzy* apresenta um esforço de controle aproximadamente 80% maior do que o PI (média de 327 vs. 178). Essa característica indica que o *fuzzy* atua com maior intensidade para manter o rastreamento preciso, justificando seu menor IAE. A maior atividade do sinal de controle decorre da estrutura incremental, com fator de escala otimizado ($K_S = 0{,}8$), que permite respostas mais rápidas às variações do erro.

5.4.7 Síntese dos Resultados

A Tabela 9 apresenta uma síntese comparativa do desempenho dos controladores nos principais critérios avaliados.

Tabela 9 – Síntese comparativa dos controladores

Critério	**Melhor Desempenho**	**Diferença**
IAE (erro acumulado)	Fuzzy	$\approx$ 33% menor
ITAE (erro persistente)	Fuzzy	$\approx$ 33% menor
Erro em regime permanente	Empate	$\approx$ 1% ambos
Robustez à carga máxima	Fuzzy	Melhora vs. degrada
Esforço de controle	PI	$\approx$ 45% menor

5.5 Discussão dos Resultados

Os resultados experimentais demonstram que o controlador *fuzzy* incremental apresenta desempenho significativamente superior ao controlador PI clássico no critério principal de avaliação (IAE), com vantagens particularmente evidentes em:

1. **Precisão de rastreamento**: A redução de aproximadamente 33% no IAE indica que o controlador *fuzzy* mantém a velocidade medida mais próxima da referência ao longo de todo o perfil de teste. Essa melhoria é consistente em todos os tipos de

transição (aceleração, desaceleração e parada) e em todas as condições de carga avaliadas.

2. **Comportamento adaptativo**: O resultado mais notável é a capacidade do controlador *fuzzy* de *melhorar* seu desempenho sob carga máxima, enquanto o PI apresenta degradação esperada. Essa característica é atribuída à natureza não linear das regras *fuzzy*, que permitem ações de controle diferenciadas conforme a magnitude e a tendência do erro, adaptando-se implicitamente às variações da dinâmica do sistema.

3. **Consistência estatística**: Os baixos desvios-padrão observados nas três repetições de cada condição ($\pm 0{,}2$ a $\pm 2{,}1$ no IAE) demonstram a repetibilidade dos resultados e a confiabilidade das conclusões.

O controlador PI apresentou vantagem apenas no esforço de controle, com sinal aproximadamente 45% menos ativo. Essa característica pode ser relevante em aplicações onde o consumo energético ou o desgaste de atuadores é crítico. Entretanto, o maior esforço do *fuzzy* resulta em um erro de rastreamento significativamente menor, representando um *trade-off* favorável para aplicações que priorizam precisão.

A equivalência no erro em regime permanente (aproximadamente 1% para ambos) confirma que a estrutura incremental do controlador *fuzzy*, com acumulação do sinal de controle, proporciona ação integral efetiva para eliminação do erro estacionário, característica essencial em sistemas de controle de velocidade.

5.6 Considerações Finais

Este capítulo apresentou o processo de validação experimental da metodologia proposta, incluindo a descrição detalhada dos procedimentos de teste, as métricas de desempenho utilizadas e a análise crítica dos resultados obtidos.

A comparação entre o controlador PI com *anti-windup* e o controlador *fuzzy* incremental, realizada por meio de 18 ensaios experimentais sob três condições de carga, demonstrou empiricamente a superioridade da abordagem *fuzzy* no rastreamento de velocidade. Os principais resultados quantitativos são:

- ❑ Redução de **32,8% a 34,0%** no IAE em todas as condições de carga;
- ❑ Redução de **32,2% a 36,2%** no IAE por tipo de degrau;
- ❑ **Melhoria de 1,4%** no desempenho sob carga máxima (vs. degradação de 0,4% do PI);
- ❑ Erro em regime permanente equivalente ($\approx 1\%$) em ambos os controladores.

Os ensaios experimentais corroboram a proposição inicial deste trabalho, evidenciando que o controlador *fuzzy* incremental é mais adequado para aplicação em sistemas de robótica móvel que operam sob condições dinâmicas variáveis. A estrutura com 49 regras, funções de pertinência triangulares e defuzzificação por centroide mostrou-se computacionalmente viável para implementação em microcontroladores de recursos limitados como o Raspberry Pi Pico, mantendo o período de amostragem de 50 ms sem comprometer o determinismo do laço de controle.

Os resultados obtidos validam não apenas o projeto do controlador *fuzzy*, mas também toda a metodologia de desenvolvimento apresentada nos capítulos anteriores, desde a identificação experimental da planta até a implementação embarcada em MicroPython.

Capítulo 6

Conclusão

6.1 Síntese do Trabalho

Este trabalho apresentou o desenvolvimento, a implementação e a validação experimental de um sistema de controle de velocidade baseado em lógica *fuzzy* para um Robô Móvel Autônomo (AMR) de configuração de tração diferencial. A proposta surgiu da necessidade de controladores mais robustos e adaptativos para aplicações de robótica móvel, nas quais incertezas paramétricas, variações de carga e perturbações externas são inerentes ao ambiente operacional.

O controlador *fuzzy* incremental foi projetado utilizando o modelo de inferência de Mamdani com 49 regras, funções de pertinência triangulares distribuídas em sete conjuntos linguísticos e método de defuzzificação por centroide. A estrutura incremental, que acumula variações no sinal de controle em vez de calcular valores absolutos, proporcionou ação integral implícita para a eliminação do erro em regime permanente, característica essencial em sistemas de controle de velocidade.

A implementação embarcada foi realizada em MicroPython no microcontrolador Raspberry Pi Pico, demonstrando a viabilidade computacional do algoritmo *fuzzy* em plataformas de recursos limitados. O sistema manteve o período de amostragem de 50 ms (frequência de 20 Hz) sem comprometer o determinismo do laço de controle, o que valida a aplicabilidade da abordagem em sistemas de tempo real.

A validação experimental, conduzida por meio de 18 ensaios sistemáticos sob três condições de carga distintas (sem carga, 5,636 kg e 11,272 kg adicionais), permitiu a comparação quantitativa entre o controlador *fuzzy* proposto e um controlador PI clássico com mecanismo *anti-windup*. Os resultados demonstraram a superioridade da abordagem *fuzzy* no critério principal de avaliação, com redução média de 33% na Integral do Erro

Absoluto (IAE) em todas as condições testadas.

6.2 Contribuições

As principais contribuições deste trabalho podem ser sintetizadas nos seguintes aspectos:

1. **Metodologia de projeto para controladores *fuzzy* incrementais**: Foi apresentada uma metodologia sistemática para o projeto de controladores *fuzzy* aplicados ao controle de velocidade em robôs móveis, incluindo a definição dos universos de discurso com base nas características operacionais do sistema, a construção da base de regras a partir de conhecimento heurístico e a sintonia dos parâmetros de escala através de análise experimental iterativa.

2. **Validação experimental com análise de robustez**: A validação foi conduzida de forma rigorosa, com múltiplas repetições e variação controlada das condições de carga, permitindo não apenas a comparação de desempenho absoluto, mas também a análise da robustez dos controladores frente a perturbações na dinâmica da planta. Os resultados quantitativos obtidos foram:

   - ❏ Redução de 32,8% a 34,0% no IAE do controlador *fuzzy* em relação ao PI;
   - ❏ Redução de 32,2% a 36,2% no IAE por tipo de transição (aceleração, desaceleração e parada);
   - ❏ Melhoria de 1,4% no desempenho do *fuzzy* sob carga máxima, contrastando com degradação de 0,4% do PI;
   - ❏ Erro em regime permanente equivalente (aproximadamente 1%) para ambos os controladores.

3. **Demonstração de capacidade adaptativa implícita**: Um dos resultados mais significativos foi a constatação de que o controlador *fuzzy* não apenas mantém, mas *melhora* seu desempenho sob a carga máxima. Essa característica, não observada no controlador PI convencional, evidencia a capacidade adaptativa inerente às estruturas não lineares baseadas em regras linguísticas, que ajustam implicitamente a intensidade da ação de controle às condições operacionais.

4. **Implementação embarcada de baixo custo**: A implementação bem-sucedida do algoritmo *fuzzy* em MicroPython no Raspberry Pi Pico demonstra que técnicas de inteligência computacional podem ser aplicadas em plataformas de hardware acessíveis, sem necessidade de processadores especializados ou unidades de ponto flutuante dedicadas. Essa contribuição é particularmente relevante para aplicações educacionais e para projetos com restrições orçamentárias.

5. **Documentação técnica e código reproduzível**: Todo o desenvolvimento foi documentado para permitir a reprodução dos experimentos e a adaptação da metodologia a outras plataformas robóticas, contribuindo para a disseminação do conhecimento na área de controle inteligente aplicado à robótica móvel.

6.3 Delimitações do Trabalho

Embora os resultados obtidos tenham sido satisfatórios e validem a hipótese inicial do trabalho, algumas delimitações devem ser consideradas na interpretação das conclusões:

1. **Escopo do sistema de controle**: O trabalho focou exclusivamente no controle de velocidade das rodas (malha interna), sem abordar o controle de trajetória ou a navegação autônoma (malha externa). A integração com sistemas de planejamento de caminho e de localização permanece como uma extensão natural do trabalho.

2. **Condições experimentais controladas**: Os ensaios foram realizados em ambiente interno com piso regular e sem obstáculos dinâmicos. O desempenho dos controladores em condições adversas (superfícies irregulares, rampas, ambientes externos) não foi avaliado de forma sistemática.

3. **Faixa de operação**: Os testes foram conduzidos em velocidades de 0 a 60 RPM, correspondentes à faixa típica de operação do robô. O comportamento em velocidades muito baixas (próximas à zona morta dos motores) ou em condições de saturação não foi explorado em profundidade.

4. **Sintonia dos controladores**: O controlador PI utilizado como referência foi sintonizado empiricamente, sem a aplicação de métodos formais de otimização. Uma sintonia mais refinada do PI poderia reduzir a diferença de desempenho observada, embora provavelmente não eliminasse a vantagem do *fuzzy* em termos de robustez.

5. **Métricas de avaliação**: A análise concentrou-se em métricas de erro de rastreamento (IAE, ITAE) e esforço de controle. Outros critérios relevantes, como o consumo energético, o desgaste mecânico e o comportamento acústico, não foram quantificados.

6.4 Conclusões Finais

Os resultados experimentais obtidos neste trabalho permitem concluir que:

1. O controlador *fuzzy* incremental é uma alternativa viável e eficaz ao controlador PI clássico para o controle de velocidade em robôs móveis diferenciais, apresentando desempenho superior em termos de precisão de rastreamento.

2. A redução média de 33% no IAE representa um ganho significativo na qualidade do controle, traduzindo-se em trajetórias mais precisas e menor acúmulo de erros na odometria do robô.

3. A capacidade do controlador *fuzzy* de manter ou melhorar seu desempenho sob variações de carga confirma sua maior robustez em comparação ao PI, característica especialmente relevante para aplicações de transporte de materiais, nas quais a carga varia durante a operação.

4. A implementação em MicroPython no Raspberry Pi Pico demonstra que a complexidade computacional do algoritmo *fuzzy* com 49 regras é compatível com microcontroladores de baixo custo, o que viabiliza sua aplicação em projetos com restrições orçamentárias.

5. O *trade-off* entre precisão e esforço de controle (o *fuzzy* apresenta aproximadamente 80% mais variação no sinal de controle) deve ser considerado no projeto, sendo favorável em aplicações que priorizam a precisão de rastreamento.

Portanto, a hipótese inicial de que o controlador *fuzzy* oferece maior robustez e melhor desempenho no rastreamento de velocidade foi validada experimentalmente, contribuindo para consolidar o uso de técnicas de inteligência computacional em sistemas de robótica móvel.

6.5 Trabalhos Futuros

Com base nas contribuições e delimitações identificadas, propõem-se as seguintes direções para trabalhos futuros:

6.5.1 Aprimoramentos na Plataforma Robótica

- ❑ **Revisão do projeto mecânico**: Análise estrutural da plataforma para reduzir vibrações e folgas mecânicas que introduzem não linearidades no sistema. Avaliação de materiais alternativos e de configurações de suspensão para operação em superfícies irregulares.

- ❑ **Modernização do sistema eletrônico**: Substituição do *driver* Sabertooth por soluções baseadas em ponte H com realimentação de corrente, permitindo a implementação de controle de torque e de detecção de sobrecarga. Adição de sensores inerciais (IMU) para a estimação de estado e a compensação de derrapagem.

- ❑ **Aprimoramento da aquisição de velocidade dos encoders**: Modificação do método de leitura dos encoders por meio da utilização de uma placa dedicada de contagem

de pulsos, reduzindo a dependência de interrupções no microcontrolador. Essa abordagem tem como objetivo mitigar efeitos de *jitter*, perda de pulsos e irregularidades na estimativa de RPM, proporcionando uma leitura de velocidade mais contínua e confiável para a malha de controle.

- ❑ **Sistema de alimentação**: Migração para baterias de íon-lítio (Li-Ion) ou polímero de lítio (LiPo) para maior densidade energética e estabilidade de tensão ao longo da descarga, reduzindo variações paramétricas durante operação prolongada.

6.5.2 Evolução do Sistema de Controle

- ❑ **Controladores *fuzzy* adaptativos**: Implementação de mecanismos de adaptação *online* dos parâmetros do controlador *fuzzy* (fatores de escala, funções de pertinência), utilizando técnicas como o gradiente descendente ou algoritmos genéticos, permitindo o ajuste automático às características da planta.

- ❑ **Controle *fuzzy* tipo-2**: Investigação de controladores *fuzzy* do tipo 2, que incorporam incerteza nas próprias funções de pertinência, potencialmente oferecendo maior robustez em ambientes com elevado grau de incerteza.

- ❑ **Comparação com outras técnicas**: Avaliação comparativa com controladores baseados em redes neurais, controle preditivo baseado em modelo (MPC) e técnicas de aprendizado por reforço, estabelecendo um *benchmark* mais abrangente para controle de velocidade em AMRs.

- ❑ **Controle de trajetória**: Extensão do trabalho para incluir a malha externa de controle de trajetória, integrando o controlador de velocidade desenvolvido com algoritmos de seguimento de caminho (*path following*) e de controle de formação para múltiplos robôs.

6.5.3 Validação em Cenários Ampliados

- ❑ **Testes de longa duração**: Realização de ensaios prolongados (horas de operação contínua) para avaliar deriva térmica, degradação de componentes e robustez do sistema de controle ao longo do tempo.

- ❑ **Ambientes externos**: Validação do sistema em condições de piso irregular, rampas e ambientes externos, avaliando a necessidade de adaptações no controlador para lidar com perturbações mais severas.

- ❑ **Análise de consumo energético**: Instrumentação do sistema para medição de consumo de corrente dos motores, estabelecendo relação entre estratégia de controle, eficiência energética e autonomia do robô.

Estas propostas de continuidade visam ampliar o escopo e a aplicabilidade do trabalho desenvolvido, contribuindo para o avanço do estado da arte no controle inteligente para robótica móvel autônoma.

Referências

ABDALAMEER, I. M. et al. Development of a control system for autonomous mobile robots: a literature review. In: . [S.l.: s.n.], 2023.

ALMEIDA, R. F.; FIGUEIREDO, J.; SANTOS, M. A fuzzy-pid speed control approach for differential drive autonomous mobile robots under variable load conditions. **Journal of Intelligent Robotic Systems**, 2023.

ARUMUGAM, V.; ALAGUMALAI, V.; SRINIVASAN, V. Development of an intelligent fuzzy logic control based on a differential drive wheeled mobile robot. In: **2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)**. [S.l.: s.n.], 2024. p. 1–6.

BATTI, H. et al. Fuzzy logic based control for autonomous mobile robot navigation and obstacles avoidance. In: **2022 IEEE Information Technologies Smart Industrial Systems (ITSIS)**. [S.l.: s.n.], 2022. p. 1–6.

BEKEY, G. A. **Autonomous Robots: From Biological Inspiration to Implementation and Control**. [S.l.]: MIT Press, 2005.

BESSA, W.; DUTRA, M.; KREUZER, E. Adaptive fuzzy control of uncertain nonlinear systems with unknown dead zones: Application to mobile robots. **Control Engineering Practice**, v. 102, p. 104524, 2020.

BROOK, T. Autonomous mobile robot for inventory management in retail industry. In: **Lecture Notes in Electrical Engineering**. [S.l.: s.n.], 2022.

BROWN, M.; HARRIS, C. **Neurofuzzy Adaptive Modelling and Control**. New York: Prentice Hall, 1994. 508 p. (Prentice-Hall International Series in Systems and Control Engineering). ISBN 978-0-13-134453-2.

CORKE, P. **Robotics, Vision and Control: Fundamental Algorithms in MATLAB**. [S.l.]: Springer, 2011.

CORREIA, D.; SILVA, M.; MOREIRA, A. P. A survey of high-level teleoperation, monitoring and task assignment to autonomous mobile robots. 2022.

DORF, R. C.; BISHOP, R. H. **Modern Control Systems**. 12. ed. Upper Saddle River, NJ: Pearson, 2011.

GIL, A. C. **Métodos e técnicas de pesquisa social**. [S.l.]: 6. ed. Ediitora Atlas SA, 2008.

HOFMANN, C. et al. Towards adaptive environment perception and understanding for autonomous mobile robots. In: **Proceedings of the IEEE SDF-MFI 2023**. [S.l.: s.n.], 2023.

JANG, J.-S. R.; SUN, C.-T.; MIZUTANI, E. **Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence**. [S.l.]: Prentice-Hall, Inc., 1997.

KAFIEV, I.; ROMANOV, P.; ROMANOVA, I. Fuzzy logic based control system for automated guided vehicle. In: **2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)**. [S.l.: s.n.], 2020. p. 1–6.

KITCHENHAM, B.; BUDGEN, D.; BRERETON, P. **Evidence-Based Software Engineering and Systematic Reviews**. [S.l.: s.n.], 2015. ISBN 9780429157653.

KOPCIK, M.; JADLOVSKý, J. Embedded control system for mobile robots with differential drive. **Acta Electrotechnica et Informatica**, v. 17, n. 3, p. 25–32, 2017.

KUMAR, A.; SAHASRABUDHE, A.; NIRGUDE, S. **Fuzzy Logic Control for Indoor Navigation of Mobile Robots**. 2024. Disponível em: <https://arxiv.org/abs/2409.02437>.

LAKATOS, E. M.; MARCONI, M. d. A. **Fundamentos de metodologia científica**. [S.l.]: Atlas, 2003. OCLC: 53849497. ISBN 978-85-224-3397-1.

LIU, Y.; ZHANG, W.; LI, C. Fuzzy adaptive control for differential-drive mobile robots with dynamic parameter uncertainties. **IEEE Access**, v. 10, p. 12245–12258, 2022.

MACENSKI, S. et al. The marathon 2: A navigation system. In: **IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.: s.n.], 2020. p. 2718–2725.

MAGHZAOUI, A.; ARIDHI, E.; MAMI, A. Fuzzy control of mobile robot speed for safe and adaptive navigation. In: **2023 IEEE Third International Conference on Signal, Control and Communication (SCC)**. [S.l.: s.n.], 2023. p. 1–6.

MathWorks. **Fuzzy Logic Toolbox Documentation**. 2025. Accessed: 2025-04-22. Disponível em: <https://www.mathworks.com/help/fuzzy/index.html>.

MOHER, D. et al. Moher d, liberati a, tetzlaff j, altman dg, group ppreferred reporting items for systematic reviews and meta-analyses: the prisma statement. plos med 6: e1000097. **Open medicine : a peer-reviewed, independent, open-access journal**, v. 3, p. e123–30, 07 2009.

NGUYEN, A.-T. et al. **Mobile Robot Motion Control Using a Combination of Fuzzy Logic Method and Kinematic Model**. 2022. Disponível em: <https://arxiv.org/abs/2211.05575>.

OGATA, K. **Modern Control Engineering**. 5. ed. Upper Saddle River, NJ: Prentice Hall, 2010.

Open Robotics. **ROS 2 Navigation Stack (Nav2)**. 2025. Disponível em: <https://navigation.ros.org/>. Acesso em: 29 dez. 2025.

POUR, P. D.; ALSAYEGH, K. M.; JARADAT, M. A. Type-2 fuzzy adaptive pid controller for differential drive mobile robot: A mechatronics approach. In: **2022 Advances in Science and Engineering Technology International Conferences (ASET)**. [S.l.: s.n.], 2022. p. 1–6.

PăUCă, O. et al. Coalitional control for mobile robots formation in logistic applications. In: . [S.l.: s.n.], 2024.

RAAFI'U, B. et al. Comparative study of fuzzy-pid and fuzzy-pi control systems on dc motor speed for four-wheeled mobile robotic. In: **2019 International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA)**. [S.l.: s.n.], 2019. p. 129–133.

REGUII, I.; HASSANI, I.; REKIK, C. Neuro-fuzzy control of a mobile robot. In: **2022 IEEE 21st international Ccnference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)**. [S.l.: s.n.], 2022. p. 45–50.

ROSS, T. J. **Fuzzy Logic with Engineering Applications**. [S.l.]: Wiley, 2010.

SAFFIOTTI, A. et al. A fuzzy controller for mobile robot navigation. **Soft Computing**, v. 22, n. 5, p. 1575–1590, 2018.

SGRIGNOLI, V. **Prototipagem de um Robô de Tração Diferencial**. 2017. Trabalho de Conclusão de Curso (Engenharia Elétrica) – Universidade Federal de São Carlos (UFSCar), São Carlos, Brasil.

SHARMA, K.; PALWALIA, D. K. A modified pid control with adaptive fuzzy controller applied to dc motor. In: **2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC)**. [S.l.: s.n.], 2017. p. 1–6.

SICILIANO, B. et al. **Robotics: Modelling, Planning and Control**. [S.l.]: Springer, 2010.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. **Introduction to Autonomous Mobile Robots**. 2nd. ed. [S.l.]: MIT Press, 2011.

TSOUKALAS, L. H.; UHRIG, R. E. **Fuzzy and Neural Approaches in Engineering**. 1. ed. New York: Wiley-Interscience, 1997. Foreword by Lotfi A. Zadeh. ISBN 978-0-471-16003-8.

WANG, L.-X. **A Course in Fuzzy Systems and Control**. [S.l.]: Prentice Hall, 1997.

WEG. **Projeto WEG Mobile Robot**. 2023. Disponível em: <https://www.weg.net/institutional/BR/pt/digital-solutions/solutions/wegmobilerobot>. Acesso em: 23 de jan. de 2025.

YANG, J. et al. A fuzzy pid speed controller for mobile robot: Application in scientific investigation. In: **2022 IEEE International Conference on Unmanned Systems (ICUS)**. [S.l.: s.n.], 2022. p. 1627–1632.

ZADEH, L. A. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338–353, 1965.

ÅSTRöM, K. J.; MURRAY, R. M. **Feedback Systems: An Introduction for Scientists and Engineers**. [S.l.]: Princeton University Press, 2010.

APÊNDICE A

Código-fonte do Arquivo de Configuração do Controlador Fuzzy (.fis)

Este apêndice apresenta o conteúdo do arquivo *.fis* desenvolvido no trabalho com o Fuzzy Logic Designer, utilizado para configurar o controlador no ambiente *MATLAB/Simulink*, contendo as definições das funções de pertinência e a base de regras.

```
[System]
Name='fuzzy_controller_amr'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=49
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='erro'
Range=[-75 75]
NumMFs=7
MF1='NG':'trimf',[-115 -75 -35]
```

```
MF2='NM':'trimf',[-58 -35 -12]
MF3='NS':'trimf',[-24 -12 0]
MF4='ZE':'trimf',[-12 0 12]
MF5='PS':'trimf',[0 12 24]
MF6='PM':'trimf',[12 35 58]
MF7='PG':'trimf',[35 75 115]

[Input2]
Name='derro'
Range=[-6 6]
NumMFs=7
MF1='NG':'trimf',[-8 -6 -4]
MF2='NM':'trimf',[-6 -4 -2]
MF3='NS':'trimf',[-4 -2 0]
MF4='ZE':'trimf',[-2 0 2]
MF5='PS':'trimf',[0 2 4]
MF6='PM':'trimf',[2 4 6]
MF7='PG':'trimf',[4 6 8]

[Output1]
Name='delta_u'
Range=[-100 100]
NumMFs=7
MF1='NG':'trimf',[-160 -100 -40]
MF2='NM':'trimf',[-72 -40 -8]
MF3='NS':'trimf',[-16 -8 0]
MF4='ZE':'trimf',[-8 0 8]
MF5='PS':'trimf',[0 8 16]
MF6='PM':'trimf',[8 40 72]
MF7='PG':'trimf',[40 100 160]

[Rules]
1 1, 1 (1) : 1
1 2, 1 (1) : 1
1 3, 1 (1) : 1
1 4, 2 (1) : 1
1 5, 3 (1) : 1
1 6, 3 (1) : 1
1 7, 4 (1) : 1
```

```
2 1, 1 (1) : 1
2 2, 1 (1) : 1
2 3, 2 (1) : 1
2 4, 2 (1) : 1
2 5, 3 (1) : 1
2 6, 4 (1) : 1
2 7, 5 (1) : 1
3 1, 1 (1) : 1
3 2, 2 (1) : 1
3 3, 3 (1) : 1
3 4, 3 (1) : 1
3 5, 4 (1) : 1
3 6, 5 (1) : 1
3 7, 6 (1) : 1
4 1, 2 (1) : 1
4 2, 3 (1) : 1
4 3, 3 (1) : 1
4 4, 4 (1) : 1
4 5, 5 (1) : 1
4 6, 5 (1) : 1
4 7, 6 (1) : 1
5 1, 2 (1) : 1
5 2, 3 (1) : 1
5 3, 4 (1) : 1
5 4, 5 (1) : 1
5 5, 5 (1) : 1
5 6, 6 (1) : 1
5 7, 7 (1) : 1
6 1, 3 (1) : 1
6 2, 4 (1) : 1
6 3, 5 (1) : 1
6 4, 6 (1) : 1
6 5, 6 (1) : 1
6 6, 7 (1) : 1
6 7, 7 (1) : 1
7 1, 4 (1) : 1
7 2, 5 (1) : 1
7 3, 5 (1) : 1
7 4, 6 (1) : 1
```

```
7 5, 7 (1) : 1
7 6, 7 (1) : 1
7 7, 7 (1) : 1
```

APÊNDICE B

Código de Programação da Raspberry Pi Pico

Este apêndice contém o código desenvolvido em MicroPython para a Raspberry Pi Pico. O código foi estruturado de forma modular para garantir o determinismo do laço de controle.

```
"""
Controlador Fuzzy para Robô Móvel Autônomo (AMR)
================================================

Sistema de controle de velocidade para robô móvel diferencial utilizando:
- Controle PI adaptativo com ganho escalonado
- Controle Fuzzy incremental com 49 regras
- Odometria diferencial para rastreamento de posição
- Encoders quadratura para medição de velocidade
- Driver Sabertooth para acionamento dos motores

Desenvolvido por: Hugo da Silva e Souza - LARIS/UFSCar
Orientador: Prof. Roberto Santos Inoue
"""

from machine import Pin, UART
import utime as time
import sys
```

```
import math
import select

# ==================== CONFIGURAÇÕES DE HARDWARE ====================

# LED indicador de status
LED_PIN = 25

# --- Configuração Sabertooth (Driver de Motores) ---
SABERTOOTH_UART_ID = 0          # ID da UART no microcontrolador
SABERTOOTH_BAUD = 9600          # Taxa de comunicação serial
SABERTOOTH_TX_PIN = 16          # Pino TX para comunicação
SABERTOOTH_ADDRESS = 128        # Endereço do Sabertooth no barramento

# Comandos do protocolo Sabertooth para controle individual dos motores
SABERTOOTH_M1_FORWARD = 0       # Motor 1 - sentido direto
SABERTOOTH_M1_REVERSE = 1       # Motor 1 - sentido reverso
SABERTOOTH_M2_FORWARD = 4       # Motor 2 - sentido direto
SABERTOOTH_M2_REVERSE = 5       # Motor 2 - sentido reverso

# --- Configuração dos Encoders (Quadratura) ---
ENCODER_LEFT_A = 4              # Canal A do encoder esquerdo
ENCODER_LEFT_B = 5              # Canal B do encoder esquerdo
ENCODER_RIGHT_A = 2             # Canal A do encoder direito
ENCODER_RIGHT_B = 3             # Canal B do encoder direito

# --- Parâmetros Físicos do Robô ---
PI = 3.14159265359
WHEEL_RADIUS = 0.065            # Raio da roda em metros
WHEEL_SEPARATION = 0.558        # Distância entre rodas em metros
MAX_RPM = 75.0                  # RPM máximo dos motores
CPR = 1684.0                    # Counts Per Revolution dos encoders

# ==================== PARÂMETROS DO CONTROLE ====================

CONTROL_PERIOD_MS = 50          # Período de controle em milissegundos (20Hz)
CONTROLLER_TYPE = 'FUZZY'       # Tipo de controlador: 'PI' ou 'FUZZY'
ALPHA_LPF = 0.8                 # Coeficiente do filtro passa-baixas (0-1)
M_SAT = 100.0                   # Limite de saturação do sinal de controle (%)
```

```
# ==================== PARÂMETROS DO CONTROLADOR PI ====================

KP_DEFAULT = 0.1                # Ganho proporcional padrão
KI_DEFAULT = 0.1                # Ganho integral padrão
TT_ANTIWINDUP = 60.0            # Constante de tempo do anti-windup

# ==================== PARÂMETROS DO CONTROLADOR FUZZY ====================

KS_FUZZY = 0.30                 # Ganho de escalonamento do controlador fuzzy

# --- Faixas de operação das variáveis fuzzy ---
ERRO_MIN = -75.0                # Erro mínimo em RPM
ERRO_MAX = 75.0                 # Erro máximo em RPM
DERRO_MIN = -6.0                # Derivada do erro mínima em RPM/s
DERRO_MAX = 6.0                 # Derivada do erro máxima em RPM/s
DELTAU_MIN = -100.0             # Variação mínima do sinal de controle
DELTAU_MAX = 100.0              # Variação máxima do sinal de controle

# --- Funções de pertinência para o ERRO (7 MFs com largura variável) ---
# Centros dos conjuntos fuzzy: NG, NM, NS, ZE, PS, PM, PG
ERRO_CENTERS = [-75, -35, -12, 0, 12, 35, 75]
# Larguras correspondentes (mais largas nos extremos para maior cobertura)
ERRO_WIDTHS = [40, 23, 12, 12, 12, 23, 40]

# --- Funções de pertinência para DERRO (7 MFs com largura uniforme) ---
DERRO_CENTERS = [-6, -4, -2, 0, 2, 4, 6]
DERRO_WIDTH = 2.0               # Largura uniforme para todas as MFs

# --- Funções de pertinência para DELTA_U (7 MFs com largura variável) ---
DELTAU_CENTERS = [-100, -40, -8, 0, 8, 40, 100]
DELTAU_WIDTHS = [60, 32, 8, 8, 8, 32, 60]

# --- Base de Regras Fuzzy (7x7 = 49 regras) ---
# Matriz de inferência: linha = erro, coluna = derro
# Valores representam o índice do conjunto fuzzy de saída (DELTAU)
FUZZY_RULES = [
    # derro:  NG   NM   NS   ZE   PS   PM   PG
    [0, 0, 0, 1, 2, 2, 3],  # erro = NG (muito negativo)
```

```
    [0, 0, 1, 1, 2, 3, 4],  # erro = NM
    [0, 1, 2, 2, 3, 4, 5],  # erro = NS
    [1, 2, 2, 3, 4, 4, 5],  # erro = ZE - mais conservador
    [1, 2, 3, 4, 4, 5, 6],  # erro = PS
    [2, 3, 4, 5, 5, 6, 6],  # erro = PM
    [3, 4, 4, 5, 6, 6, 6],  # erro = PG (muito positivo)
]

# ==================== INICIALIZAÇÃO DO HARDWARE ====================

# Inicializa LED de status
led = Pin(LED_PIN, Pin.OUT)
led.value(1)  # LED aceso indica sistema inicializado

# Inicializa UART para comunicação com Sabertooth
uart = UART(SABERTOOTH_UART_ID, baudrate=SABERTOOTH_BAUD, tx=Pin(SABERTOOTH_TX_PIN))

# Configura pinos dos encoders com pull-up interno
encoder_left_a = Pin(ENCODER_LEFT_A, Pin.IN, Pin.PULL_UP)
encoder_left_b = Pin(ENCODER_LEFT_B, Pin.IN, Pin.PULL_UP)
encoder_right_a = Pin(ENCODER_RIGHT_A, Pin.IN, Pin.PULL_UP)
encoder_right_b = Pin(ENCODER_RIGHT_B, Pin.IN, Pin.PULL_UP)

# Estado inicial dos encoders (quadratura de 2 bits)
left_state = (encoder_left_a.value() << 1) | encoder_left_b.value()
right_state = (encoder_right_a.value() << 1) | encoder_right_b.value()

# ==================== VARIÁVEIS GLOBAIS DE ESTADO ====================

# --- Encoders ---
left_position = 0              # Posição acumulada do encoder esquerdo
right_position = 0             # Posição acumulada do encoder direito
prev_left_pos = 0              # Posição anterior (para cálculo de velocidade)
prev_right_pos = 0

# --- Velocidades ---
left_rpm_raw = 0.0             # RPM instantâneo (sem filtro)
right_rpm_raw = 0.0
left_rpm_filtered = 0.0        # RPM filtrado (passa-baixas)
```

```
right_rpm_filtered = 0.0

# --- Referências de velocidade ---
rpm_ref_left = 0.0              # Referência de RPM para motor esquerdo
rpm_ref_right = 0.0             # Referência de RPM para motor direito

# --- Controlador PI ---
sum_error_left = 0.0            # Somatório do erro (termo integral) - esquerdo
sum_error_right = 0.0           # Somatório do erro (termo integral) - direito

# --- Controlador Fuzzy ---
prev_error_left = 0.0           # Erro anterior para cálculo da derivada
prev_error_right = 0.0
u_left_accumulated = 0.0        # Sinal de controle acumulado (integral da saída fu
u_right_accumulated = 0.0

# --- Sinais de controle atuais ---
u_left = 0.0                    # Sinal de controle motor esquerdo (%)
u_right = 0.0                   # Sinal de controle motor direito (%)

# --- Odometria ---
x_pos = 0.0                     # Posição X no plano
y_pos = 0.0                     # Posição Y no plano
theta = 0.0                     # Orientação (ângulo) em radianos

# --- Modo de operação ---
action = 0                      # 0: Parado, 1: Controle, 2: PWM direto

# --- Logging ---
LOG_MODE = 'SERIAL'             # 'SERIAL' ou 'FILE'
data_log = []                   # Buffer para armazenar dados de log
MAX_LOG_SAMPLES = 500           # Máximo de amostras no buffer

# ==================== FUNÇÕES DOS ENCODERS ====================


def encoder_left_isr(pin):
    """
    Interrupt Service Routine para o encoder esquerdo.
```

```
    Detecta transições de estado na quadratura (A/B) e incrementa/decrementa
    a posição conforme a direção de rotação.

    Sequência de estados na rotação direta: 00 -> 01 -> 11 -> 10 -> 00
    Sequência de estados na rotação reversa: 00 -> 10 -> 11 -> 01 -> 00

    Args:
        pin: Pino que gerou a interrupção (não utilizado)
    """
    global left_position, left_state

    # Lê estado atual dos canais A e B
    a = encoder_left_a.value()
    b = encoder_left_b.value()
    new_state = (a << 1) | b  # Combina em um valor de 2 bits

    # Verifica se houve mudança de estado
    if left_state != new_state:
        # Detecta rotação no sentido direto
        if ((left_state == 0 and new_state == 1) or
            (left_state == 1 and new_state == 3) or
            (left_state == 3 and new_state == 2) or
                (left_state == 2 and new_state == 0)):
            left_position += 1
        # Detecta rotação no sentido reverso
        elif ((left_state == 0 and new_state == 2) or
              (left_state == 2 and new_state == 3) or
              (left_state == 3 and new_state == 1) or
                (left_state == 1 and new_state == 0)):
            left_position -= 1
        left_state = new_state


def encoder_right_isr(pin):
    """
    Interrupt Service Routine para o encoder direito.

    Funcionamento idêntico ao encoder_left_isr, mas para o motor direito.
```

```
    Args:
        pin: Pino que gerou a interrupção (não utilizado)
    """
    global right_position, right_state

    a = encoder_right_a.value()
    b = encoder_right_b.value()
    new_state = (a << 1) | b

    if right_state != new_state:
        if ((right_state == 0 and new_state == 1) or
            (right_state == 1 and new_state == 3) or
            (right_state == 3 and new_state == 2) or
                (right_state == 2 and new_state == 0)):
            right_position += 1
        elif ((right_state == 0 and new_state == 2) or
              (right_state == 2 and new_state == 3) or
              (right_state == 3 and new_state == 1) or
                (right_state == 1 and new_state == 0)):
            right_position -= 1
        right_state = new_state


# Registra as ISRs para detectar mudanças em ambos os canais (A e B)
encoder_left_a.irq(trigger=Pin.IRQ_RISING | Pin.IRQ_FALLING, handler=encoder_left_i
encoder_left_b.irq(trigger=Pin.IRQ_RISING | Pin.IRQ_FALLING, handler=encoder_left_i
encoder_right_a.irq(trigger=Pin.IRQ_RISING | Pin.IRQ_FALLING, handler=encoder_right
encoder_right_b.irq(trigger=Pin.IRQ_RISING | Pin.IRQ_FALLING, handler=encoder_right


def calculate_rpm(dt):
    """
    Calcula o RPM dos motores baseado na variação de posição dos encoders.

    Aplica filtro passa-baixas (EMA - Exponential Moving Average) para
    suavizar as medições e reduzir ruído.

    Fórmula: RPM = (delta_position * 60) / (CPR * dt)
```

```
    Args:
        dt: Intervalo de tempo desde a última medição (segundos)
    """
    global prev_left_pos, prev_right_pos
    global left_rpm_raw, right_rpm_raw
    global left_rpm_filtered, right_rpm_filtered

    if dt <= 0:
        return

    # Calcula variação de posição desde última amostra
    left_delta = left_position - prev_left_pos
    right_delta = right_position - prev_right_pos

    # Converte contagens para RPM
    left_rpm_raw = (left_delta * 60.0) / (CPR * dt)
    right_rpm_raw = (right_delta * 60.0) / (CPR * dt)
    right_rpm_raw = -right_rpm_raw  # Inverte sinal devido à montagem mecânica

    # Aplica filtro passa-baixas (EMA)
    # y[n] =  * y[n-1] + (1-) * x[n]
    left_rpm_filtered = ALPHA_LPF * left_rpm_filtered + (1 - ALPHA_LPF) * left_rpm_raw
    right_rpm_filtered = ALPHA_LPF * right_rpm_filtered + (1 - ALPHA_LPF) * right_rpm_ra

    # Atualiza posições anteriores
    prev_left_pos = left_position
    prev_right_pos = right_position


# ==================== FUNÇÕES DO MOTOR (Sabertooth) ====================


def _sabertooth_send(command: int, data: int) -> None:
    """
    Envia comando via protocolo Sabertooth simplificado.

    Formato do pacote: [ADDRESS][COMMAND][DATA][CHECKSUM]
    - ADDRESS: Endereço do driver (128)
```

```
    - COMMAND: Tipo de comando (0-5 para motores)
    - DATA: Valor de 0-127 representando 0-100% do PWM
    - CHECKSUM: Soma dos 3 bytes anteriores, limitada a 7 bits

    Args:
        command: Código do comando Sabertooth
        data: Valor do comando (0-127)
    """
    # Garante que data está no range válido
    if data < 0:
        data = 0
    if data > 127:
        data = 127

    # Calcula checksum (7 bits menos significativos)
    checksum = (SABERTOOTH_ADDRESS + command + data) & 0x7F

    # Monta pacote de 4 bytes
    packet = bytes((SABERTOOTH_ADDRESS, command, data, checksum))

    try:
        uart.write(packet)
    except Exception as e:
        print("UART ERROR:", e)


def motor_send_cmd(left_cmd: float, right_cmd: float) -> None:
    """
    Envia comandos PWM para ambos os motores.

    Converte comandos em porcentagem (-100 a +100) para o formato
    Sabertooth (0-127 com comandos separados para frente/ré).

    Args:
        left_cmd: Comando motor esquerdo em % (-100 a +100)
        right_cmd: Comando motor direito em % (-100 a +100)
    """
    percent_left = left_cmd
    percent_right = -right_cmd  # Inverte devido à montagem mecânica
```

```
    for motor in (1, 2):
        percent = percent_left if motor == 1 else percent_right

        # Converte porcentagem para escala 0-127
        data = int(abs(percent) * 127.0 / 100.0)

        # Seleciona comando baseado no sinal
        if motor == 1:
            cmd = SABERTOOTH_M1_FORWARD if percent >= 0 else SABERTOOTH_M1_REVERSE
        else:
            cmd = SABERTOOTH_M2_FORWARD if percent >= 0 else SABERTOOTH_M2_REVERSE

        _sabertooth_send(cmd, data)


def motor_stop() -> None:
    """
    Para ambos os motores enviando comando de 0%.
    """
    motor_send_cmd(0.0, 0.0)


# ==================== CONTROLADOR PI ====================


def get_pi_gains(rpm_ref):
    """
    Retorna ganhos PI adaptativos baseados na velocidade de referência.

    Implementa escalonamento de ganhos (gain scheduling):
    - Baixas velocidades (<10 RPM): Kp mais alto para melhor resposta
    - Velocidades médias (10-40 RPM): Kp intermediário
    - Altas velocidades (>40 RPM): Kp padrão

    Args:
        rpm_ref: Velocidade de referência em RPM

    Returns:
```

```
        tuple: (kp, ki) - Ganhos proporcional e integral
    """
    rpm_abs = abs(rpm_ref)
    if rpm_abs < 10:
        return 0.15, KI_DEFAULT
    elif rpm_abs < 40:
        return 0.12, KI_DEFAULT
    else:
        return KP_DEFAULT, KI_DEFAULT


def pi_controller_left(rpm_ref, rpm_measured):
    """
    Controlador PI com anti-windup para o motor esquerdo.

    Implementa controle PI discreto:
    u(k) = Kp * e(k) + Ki * e(k)

    Anti-windup: Limita o termo integral quando o sinal de controle satura,
    usando back-calculation com constante de tempo TT_ANTIWINDUP.

    Args:
        rpm_ref: Velocidade de referência em RPM
        rpm_measured: Velocidade medida em RPM

    Returns:
        float: Sinal de controle u (%) limitado a ±M_SAT
    """
    global sum_error_left

    # Obtém ganhos adaptativos
    kp, ki = get_pi_gains(rpm_ref)

    # Calcula erro
    erro = rpm_ref - rpm_measured

    # Acumula erro (termo integral)
    sum_error_left += erro
```

```
    # Lei de controle PI
    u = kp * erro + ki * sum_error_left

    # Anti-windup com back-calculation
    if u > M_SAT:
        sum_error_left -= (u - M_SAT) / TT_ANTIWINDUP
        u = M_SAT
    elif u < -M_SAT:
        sum_error_left -= (u + M_SAT) / TT_ANTIWINDUP
        u = -M_SAT

    return u


def pi_controller_right(rpm_ref, rpm_measured):
    """
    Controlador PI com anti-windup para o motor direito.

    Implementação idêntica ao pi_controller_left, mas com variáveis independentes.

    Args:
        rpm_ref: Velocidade de referência em RPM
        rpm_measured: Velocidade medida em RPM

    Returns:
        float: Sinal de controle u (%) limitado a ±M_SAT
    """
    global sum_error_right

    kp, ki = get_pi_gains(rpm_ref)
    erro = rpm_ref - rpm_measured
    sum_error_right += erro
    u = kp * erro + ki * sum_error_right

    if u > M_SAT:
        sum_error_right -= (u - M_SAT) / TT_ANTIWINDUP
        u = M_SAT
    elif u < -M_SAT:
        sum_error_right -= (u + M_SAT) / TT_ANTIWINDUP
```

```
        u = -M_SAT

    return u


# ==================== CONTROLADOR FUZZY v3 ====================


def trimf_var(x, center, width):
    """
    Função de pertinência triangular com largura variável.

    Calcula o grau de pertinência de x no conjunto fuzzy triangular
    centrado em 'center' com largura 'width'.

    Geometria do triângulo:
    - Base esquerda: center - width
    - Pico: center ( = 1.0)
    - Base direita: center + width

    Args:
        x: Valor a ser fuzzificado
        center: Centro do conjunto fuzzy
        width: Largura do triângulo (distância do centro à base)

    Returns:
        float: Grau de pertinência no intervalo [0, 1]
    """
    a = center - width  # Base esquerda
    b = center          # Pico
    c = center + width  # Base direita

    # Fora do suporte
    if x <= a or x >= c:
        return 0.0
    # Rampa ascendente
    elif x <= b:
        if b == a:
            return 1.0
```

```
        return (x - a) / (b - a)
    # Rampa descendente
    else:
        if c == b:
            return 1.0
        return (c - x) / (c - b)


def fuzzify_erro(erro):
    """
    Fuzzifica o erro de velocidade.

    Calcula os graus de pertinência do erro nos 7 conjuntos fuzzy:
    NG (Negativo Grande), NM, NS, ZE (Zero), PS, PM, PG (Positivo Grande)

    Args:
        erro: Erro de velocidade em RPM

    Returns:
        list: Vetor com 7 graus de pertinência [0, 1]
    """
    mf = []
    for center, width in zip(ERRO_CENTERS, ERRO_WIDTHS):
        mf.append(trimf_var(erro, center, width))
    return mf


def fuzzify_derro(derro):
    """
    Fuzzifica a derivada do erro.

    Calcula os graus de pertinência de derro nos 7 conjuntos fuzzy
    com largura uniforme.

    Args:
        derro: Derivada do erro em RPM/s

    Returns:
        list: Vetor com 7 graus de pertinência [0, 1]
```

```
    """
    mf = []
    for center in DERRO_CENTERS:
        mf.append(trimf_var(derro, center, DERRO_WIDTH))
    return mf


def fuzzy_inference(erro_mf, derro_mf):
    """
    Motor de inferência fuzzy usando método de Mamdani com defuzzificação CoG.

    Processo:
    1. Para cada uma das 49 regras (7x7):
       - Calcula ativação: min(_erro, _derro)
       - Obtém conjunto de saída correspondente
       - Acumula contribuição ponderada
    2. Defuzzificação: Centro de Gravidade (CoG)
       Saída = (ativação_i * centro_i) / (ativação_i)

    Args:
        erro_mf: Vetor com graus de pertinência do erro [7]
        derro_mf: Vetor com graus de pertinência do derro [7]

    Returns:
        float: Valor defuzzificado de delta_u
    """
    numerator = 0.0
    denominator = 0.0

    # Percorre todas as 49 regras
    for i in range(7):  # Índice do erro
        for j in range(7):  # Índice do derro
            # Operador AND: mínimo entre as premissas
            activation = min(erro_mf[i], derro_mf[j])

            if activation > 0:
                # Obtém índice do conjunto de saída
                output_idx = FUZZY_RULES[i][j]
                output_center = DELTAU_CENTERS[output_idx]
```

```
                # Acumula para defuzzificação CoG
                numerator += activation * output_center
                denominator += activation

    # Defuzzificação: Centro de Gravidade
    if denominator > 0:
        return numerator / denominator
    return 0.0


def fuzzy_controller_left(rpm_ref, rpm_measured, dt):
    """
    Controlador Fuzzy incremental (tipo PD) para motor esquerdo.

    Estratégia de controle:
    1. Calcula erro e derivada do erro
    2. Fuzzifica erro e derro
    3. Inferência fuzzy -> delta_u
    4. Integra delta_u: u(k) = u(k-1) + Ks * delta_u
    5. Aplica saturação

    Vantagens da abordagem incremental:
    - Sem necessidade de anti-windup complexo
    - Integração natural do sinal de controle
    - Melhor para sistemas com perturbações

    Args:
        rpm_ref: Velocidade de referência em RPM
        rpm_measured: Velocidade medida em RPM
        dt: Período de amostragem (não utilizado nesta versão)

    Returns:
        float: Sinal de controle u (%) limitado a ±M_SAT
    """
    global prev_error_left, u_left_accumulated

    # Calcula erro
    erro = rpm_ref - rpm_measured
```

```
    # Calcula derivada do erro (aproximação por diferenças finitas)
    derro = erro - prev_error_left

    # Limita derro ao range válido
    derro = max(DERRO_MIN, min(DERRO_MAX, derro))

    # Fuzzificação
    erro_mf = fuzzify_erro(erro)
    derro_mf = fuzzify_derro(derro)

    # Inferência fuzzy
    delta_u = fuzzy_inference(erro_mf, derro_mf)

    # Escalonamento e integração
    delta_u_scaled = delta_u * KS_FUZZY
    u_left_accumulated += delta_u_scaled

    # Saturação
    if u_left_accumulated > M_SAT:
        u_left_accumulated = M_SAT
    elif u_left_accumulated < -M_SAT:
        u_left_accumulated = -M_SAT

    # Atualiza erro anterior
    prev_error_left = erro

    return u_left_accumulated


def fuzzy_controller_right(rpm_ref, rpm_measured, dt):
    """
    Controlador Fuzzy incremental para motor direito.

    Implementação idêntica ao fuzzy_controller_left, mas com variáveis independente

    Args:
        rpm_ref: Velocidade de referência em RPM
        rpm_measured: Velocidade medida em RPM
```

```
        dt: Período de amostragem (não utilizado)

    Returns:
        float: Sinal de controle u (%) limitado a ±M_SAT
    """
    global prev_error_right, u_right_accumulated

    erro = rpm_ref - rpm_measured
    derro = erro - prev_error_right
    derro = max(DERRO_MIN, min(DERRO_MAX, derro))

    erro_mf = fuzzify_erro(erro)
    derro_mf = fuzzify_derro(derro)
    delta_u = fuzzy_inference(erro_mf, derro_mf)

    delta_u_scaled = delta_u * KS_FUZZY
    u_right_accumulated += delta_u_scaled

    if u_right_accumulated > M_SAT:
        u_right_accumulated = M_SAT
    elif u_right_accumulated < -M_SAT:
        u_right_accumulated = -M_SAT

    prev_error_right = erro
    return u_right_accumulated


# ==================== RESET ====================


def reset_controllers():
    """
    Reseta todos os estados internos dos controladores.

    Deve ser chamado quando:
    - Muda modo de operação (parado -> controle -> PWM)
    - Inicia nova trajetória
    - Detecta mudança brusca de referência
```

```
    Zera:
    - Termos integrais (PI e Fuzzy)
    - Erros anteriores
    - Sinais acumulados
    """
    global sum_error_left, sum_error_right
    global prev_error_left, prev_error_right
    global u_left_accumulated, u_right_accumulated

    sum_error_left = 0.0
    sum_error_right = 0.0
    prev_error_left = 0.0
    prev_error_right = 0.0
    u_left_accumulated = 0.0
    u_right_accumulated = 0.0


# ==================== ODOMETRIA ====================


def update_odometry():
    """
    Atualiza a pose do robô usando odometria diferencial.

    Modelo cinemático:
    1. Calcula distâncias percorridas por cada roda
    2. Distância linear do centro: d_C = (d_R + d_L) / 2
    3. Variação angular:  = (d_R - d_L) / L
    4. Atualiza orientação:  +=
    5. Atualiza posição:
       x += d_C * cos()
       y += d_C * sin()

    Nota: Assume movimento em arco circular entre atualizações.
    """
    global x_pos, y_pos, theta

    # Calcula distâncias percorridas pelas rodas (em metros)
    d_L = 2 * PI * WHEEL_RADIUS * (left_position / CPR)
```

```
    d_R = 2 * PI * WHEEL_RADIUS * (right_position / CPR)

    # Distância linear do centro de massa
    d_C = (d_R + d_L) / 2

    # Variação angular
    delta_theta = (d_R - d_L) / WHEEL_SEPARATION

    # Atualiza orientação
    theta += delta_theta

    # Atualiza posição cartesiana
    x_pos += d_C * math.cos(theta)
    y_pos += d_C * math.sin(theta)


# ==================== COMUNICAÇÃO ====================


def parse_velocity_command():
    """
    Processa comandos de velocidade recebidos via stdin.

    Formato do comando: $xxxxxxxx yyyyyyy A#
    - $: Marcador de início
    - xxxxxxxx: Referência motor esquerdo (8 chars, float)
    - (espaço)
    - yyyyyyyy: Referência motor direito (8 chars, float)
    - A: Modo de ação (1 char, int)
      * 0: Parado
      * 1: Controle de velocidade
      * 2: PWM direto (identificação)
    - #: Marcador de fim

    Exemplo: $+0025.50 -0030.00 1#
    """
    global rpm_ref_left, rpm_ref_right, action

    # Verifica se há dados disponíveis (non-blocking)
```

```
if sys.stdin in select.select([sys.stdin], [], [], 0)[0]:
    try:
        char = sys.stdin.read(1)
        if char == '$':
            # Lê próximos 20 caracteres
            msg = sys.stdin.read(20)
            if len(msg) >= 20 and msg[19] == '#':
                try:
                    # Parse dos valores
                    new_ref_left = float(msg[0:8])
                    new_ref_right = float(msg[9:17])
                    new_action = int(msg[18])

                    # Detecta mudança de modo
                    if new_action != action:
                        reset_controllers()
                        if new_action == 2:
                            print(">>> MODO IDENTIFICAÇÃO (PWM direto)")
                        elif new_action == 1:
                            print(f">>> MODO CONTROLE ({CONTROLLER_TYPE})")
                        elif new_action == 0:
                            print(">>> MODO PARADO")

                    action = new_action

                    # Aplica limites conforme o modo
                    if action == 1:  # Modo controle
                        rpm_ref_left = max(-MAX_RPM, min(MAX_RPM, new_ref_left)
                        rpm_ref_right = max(-MAX_RPM, min(MAX_RPM, new_ref_righ
                    elif action == 2:  # Modo PWM direto
                        rpm_ref_left = max(-100.0, min(100.0, new_ref_left))
                        rpm_ref_right = max(-100.0, min(100.0, new_ref_right))
                    else:  # Modo parado
                        rpm_ref_left = 0.0
                        rpm_ref_right = 0.0

                    # Feedback
                    if action == 2:
                        print(f"CMD: PWM L={rpm_ref_left:.1f}% R={rpm_ref_right
```

```
                        else:
                            print(f"CMD: RPM L={rpm_ref_left:.1f} R={rpm_ref_right:.1f}

                    except ValueError:
                        pass  # Ignora comandos mal formatados
        except:
            pass  # Ignora erros de leitura


# ==================== LOGGING ====================


def log_data(timestamp_ms, ref_l, ref_r, rpm_meas_l, rpm_meas_r, u_l, u_r):
    """
    Registra dados para análise posterior.

    Suporta dois modos:
    - 'FILE': Armazena em buffer para salvamento posterior
    - 'SERIAL': Envia imediatamente via serial

    Formato serial: #timestamp,ref_l,ref_r,meas_l,meas_r,u_l,u_r$

    Args:
        timestamp_ms: Timestamp em milissegundos
        ref_l: Referência motor esquerdo
        ref_r: Referência motor direito
        rpm_meas_l: RPM medido motor esquerdo
        rpm_meas_r: RPM medido motor direito
        u_l: Sinal de controle esquerdo
        u_r: Sinal de controle direito
    """
    global data_log

    if LOG_MODE == 'FILE':
        if len(data_log) < MAX_LOG_SAMPLES:
            data_log.append([timestamp_ms, ref_l, ref_r, rpm_meas_l, rpm_meas_r, u_l, u_
    elif LOG_MODE == 'SERIAL':
        print(f"#{timestamp_ms},{ref_l:.2f},{ref_r:.2f},{rpm_meas_l:.2f},{rpm_meas_r:.2f
```

```
def save_log_to_file(filename):
    """
    Salva buffer de log em arquivo CSV.

    Args:
        filename: Nome do arquivo de saída

    Returns:
        bool: True se salvamento foi bem-sucedido
    """
    try:
        with open(filename, 'w') as f:
            # Cabeçalho
            f.write("time_ms,ref_left,ref_right,meas_left,meas_right,u_left,u_right
            # Dados
            for sample in data_log:
                f.write(f"{sample[0]},{sample[1]:.3f},{sample[2]:.3f},"
                        f"{sample[3]:.3f},{sample[4]:.3f},{sample[5]:.3f},{sample[6
        print(f" Log salvo em '{filename}' ({len(data_log)} amostras)")
        return True
    except Exception as e:
        print(f" Erro ao salvar: {e}")
        return False


# ==================== MAIN ====================


def main():
    """
    Loop principal de controle.

    Fluxo de execução:
    1. Inicialização
    2. Loop infinito:
       a. Verifica se chegou período de controle (50ms)
       b. Calcula RPM dos motores
       c. Executa controlador conforme modo
```

```
   d. Envia comandos aos motores
   e. Atualiza odometria
   f. Registra dados
   g. Atualiza LED de status
   h. Processa comandos de entrada
3. Em caso de interrupção (Ctrl+C):
   - Para motores
   - Salva log (se modo FILE)
   - Finaliza
"""
global u_left, u_right, data_log

# Inicialização
motor_stop()
time.sleep_ms(500)

print("\n Loop iniciado. Aguardando comandos...\n")

last_control_time = time.ticks_ms()
start_time = time.ticks_ms()

try:
    while True:
        now = time.ticks_ms()

        # ===== PERÍODO DE CONTROLE =====
        if time.ticks_diff(now, last_control_time) >= CONTROL_PERIOD_MS:
            # Calcula dt em segundos
            dt = time.ticks_diff(now, last_control_time) / 1000.0

            # Atualiza medições de RPM
            calculate_rpm(dt)

            # ===== SELEÇÃO DO CONTROLADOR =====
            if action == 0:  # Modo parado
                u_left = 0.0
                u_right = 0.0

            elif action == 1:  # Modo controle
```

```
            if CONTROLLER_TYPE == 'PI':
                u_left = pi_controller_left(rpm_ref_left, left_rpm_filtered
                u_right = pi_controller_right(rpm_ref_right, right_rpm_filt
            else:  # FUZZY
                u_left = fuzzy_controller_left(rpm_ref_left, left_rpm_filte
                u_right = fuzzy_controller_right(rpm_ref_right, right_rpm_f

        elif action == 2:  # Modo PWM direto (identificação)
            u_left = rpm_ref_left
            u_right = rpm_ref_right

        # Envia comandos aos motores
        motor_send_cmd(u_left, u_right)

        # Atualiza odometria
        update_odometry()

        # Registra dados
        elapsed = time.ticks_diff(now, start_time)
        log_data(elapsed, rpm_ref_left, rpm_ref_right,
                 left_rpm_filtered, right_rpm_filtered, u_left, u_right)

        # ===== CONTROLE DO LED DE STATUS =====
        if action == 0:  # Parado: LED aceso contínuo
            led.value(1)
        elif action == 1:  # Controle: Pisca lento (500ms)
            if elapsed % 500 < 100:
                led.value(0)
            else:
                led.value(1)
        elif action == 2:  # PWM direto: Pisca rápido (200ms)
            if elapsed % 200 < 100:
                led.value(0)
            else:
                led.value(1)

        last_control_time = now

    # Processa comandos recebidos
```

```
            parse_velocity_command()

            # Pequeno delay para não sobrecarregar CPU
            time.sleep_ms(1)

    except KeyboardInterrupt:
        print("\n\n Interrupção detectada!")

    finally:
        # Procedimentos de desligamento seguro
        motor_stop()
        print(" Motores parados")

        # Salva log se estiver em modo FILE
        if LOG_MODE == 'FILE' and len(data_log) > 0:
            filename = f"log_{CONTROLLER_TYPE.lower()}.csv"
            print(f"\n Salvando {len(data_log)} amostras...")
            save_log_to_file(filename)

        print("\n Finalizado!")


# ==================== PONTO DE ENTRADA ====================

if __name__ == "__main__":
    main()
```