UNIVERSIDADE DO ESTADO DO AMAZONAS
EST - ESCOLA SUPERIOR DE TECNOLOGIA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MARCOS VINICIUS SOUZA DA COSTA

APLICAÇÃO DE TÉCNICAS DE CONTROLE E VISÃO COMPUTACIONAL EM ROBÓTICA MÓVEL COM CONTROLADOR *FUZZY* E PID

MANAUS
2024

MARCOS VINICIUS SOUZA DA COSTA

APLICAÇÃO DE TÉCNICAS DE CONTROLE E VISÃO COMPUTACIONAL EM ROBÓTICA MÓVEL COM CONTROLADOR *FUZZY* E PID

Trabalho de Conclusão do Curso apresentado à coordenação do curso de Engenharia de Controle e Automação da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, em cumprimento às exigências legais para a obtenção do título de Graduado no Curso de Engenharia de Controle e Automação.

Área de concentração: Engenharia de Controle e Automação.

Orientador: Prof. Dr. Almir Kimura Junior

MANAUS
2024

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

M321aa da Costa, Marcos Vinicius Souza
Aplicação de técnicas de controle e visão computacional em robótica móvel com controlador Fuzzy e PID / Marcos Vinicius Souza da Costa. Manaus : [s.n], 2024.
99 f.: color.; 297 cm.

TCC - Graduação em Engenharia de Controle e Automação; - Universidade do Estado do Amazonas, Manaus, 2024.
Inclui bibliografia
Orientador: Dr. Almir Kimura Junior

1. Robótica autônoma. 2. Visão computacional. 3. Controlador Fuzzy. 4. Processamento de imagens. 5. Controlador PID. I. Dr. Almir Kimura Junior (Orient.). II. Universidade do Estado do Amazonas. III. Aplicação de técnicas de controle e visão computacional em robótica móvel com controlador Fuzzy e PID

MARCOS VINICIUS SOUZA DA COSTA

APLICAÇÃO DE TÉCNICAS DE CONTROLE E VISÃO COMPUTACIONAL EM ROBÓTICA MÓVEL COM CONTROLADOR *FUZZY* E PID

Trabalho de Conclusão do Curso apresentado à coordenação do curso de Engenharia de Controle e Automação da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, em cumprimento às exigências legais para a obtenção do título de Graduado no Curso de Engenharia de Controle e Automação.

Área de concentração: Engenharia de Controle e Automação.

Aprovada em: 27/02/2024.

BANCA EXAMINADORA

Prof. Dr. Rodrigo Farias Araújo
Examinador interno (EST/UEA)

Prof. Dr. Moisés Bastos
Examinador interno (EST/UEA)

Vitor Fernando de Souza Gadelha
Convidado

AGRADECIMENTOS

Agradecimentos são estendidos a todas as pessoas que fizeram parte da minha evolução intelectual e pessoal, a todos que disponibilizaram seu tempo para me auxiliar e me ensinaram como progredir tecnicamente. Um agradecimento especial é dedicado à Universidade do Estado do Amazonas (UEA) e ao Samsung Ocean, que proporcionaram meu crescimento técnico e contribuíram para o desenvolvimento deste trabalho.

Além disso, gostaria de expressar minha profunda gratidão à minha mãe, Hilma Duarte Souza, cujo amor, apoio e inspiração foram fundamentais em todas as etapas deste percurso acadêmico. Sua dedicação e incentivo foram a luz que guiou meu caminho, e por isso, dedico este trabalho a ela.

"Ninguém vai bater mais forte do que a vida. Não importa como você bate e sim o quanto aguenta apanhar e continuar lutando; o quanto pode suportar e seguir em frente. É assim que se ganha."

(Rocky Balboa)

RESUMO

Este trabalho de conclusão de curso apresenta o desenvolvimento de um robô seguidor de linha, utilizando processamento digital de imagens e a implementação de controladores *Fuzzy* e PID. O objetivo do projeto é construir um sistema que permita ao robô seguir uma trajetória pré-definida de forma autônoma, utilizando um sensor de câmera para capturar imagens da linha de percurso. O processo de implementação do sistema envolveu a utilização de técnicas de processamento digital de imagens para identificar a linha de percurso e determinar a posição do robô em relação a ela. Além disso, foram incorporados um controlador *Fuzzy* e um controlador proporcional-integral-derivativo (PID) para otimizar o desempenho do robô na correção de trajetória. Os resultados deste projeto evidenciam a eficácia do sistema desenvolvido, destacando a capacidade do robô de seguir a trajetória com precisão, mesmo em condições adversas de iluminação e variações na linha de percurso. A avaliação comparativa dos controladores *Fuzzy* e PID, utilizando a Odometria como métrica, revelou um desempenho superior do *Fuzzy*. O controlador *Fuzzy* alcançou um percentual de acerto de 80% na trajetória e na quantidade de trajetos realizados, enquanto o PID obteve um percentual de 60%. A integração do processamento digital de imagens com os controladores *Fuzzy* e PID proporcionou uma resposta rápida e eficiente do robô diante das mudanças no ambiente. O robô resultante deste trabalho demonstrou coesão com a proposta inicial de aplicar PDI com controle *Fuzzy* e PID, evidenciando a eficiência do controle *Fuzzy* em relação ao PID neste projeto. Essas técnicas, utilizadas com sucesso, podem servir como referência em futuros projetos de conclusão de curso.

Palavras-chave: Robótica autônoma, Visão computacional, Seguidor de linha, Processamento de imagens, Controlador PID, Controlador *Fuzzy*.

ABSTRACT

This undergraduate thesis presents the development of a line-following robot using digital image processing and the implementation of *Fuzzy* and PID controllers. The project's objective is to build a system that enables the robot to autonomously follow a predefined trajectory using a camera sensor to capture images of the path. The implementation process involved the use of digital image processing techniques to identify the path and determine the robot's position relative to it. Additionally, a *Fuzzy* controller and a proportional-integral-derivative (PID) controller were incorporated to optimize the robot's performance in trajectory correction. The project results demonstrate the effectiveness of the developed system, showcasing the robot's ability to accurately follow the trajectory even under challenging lighting conditions and path variations. The comparative evaluation of *Fuzzy* and PID controllers, using Odometry as a metric, revealed superior performance by the *Fuzzy* controller. The *Fuzzy* controller achieved an 80% accuracy rate in trajectory following and the number of completed paths, while the PID controller achieved a 60% accuracy rate. The integration of digital image processing with *Fuzzy* and PID controllers provided a quick and efficient response of the robot to changes in the environment. The resulting robot from this work exhibited coherence with the initial proposal of applying Digital Image Processing (PDI) with *Fuzzy* and PID control, highlighting the efficiency of the *Fuzzy* controller compared to PID in this project. These successfully applied techniques can serve as a reference for future thesis projects.

Keywords: Autonomous robotics, Computer vision, Line follower, Image processing, PID Controller, Controlador *Fuzzy*.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

PDI	Digital Image Processing
OpenCV	Open Source Computer Vision Library
RGB	Red Green Blue
HSV	Hue Saturation Value
AGV	Automated Guided Vehicle
RPM	Revolutions Per Minute
RPS	Revolutions Per Second
PID	Proporcional, Integral e Derivativo
FPS	Frames Per Second
PWM	Pulse Width Modulation
USB	Universal Serial Bus
VNC	Virtual Network Computing
Venv	Virtual Environment
CNC	Computer Numerical Control
STL	Stereolithograph
ABS	Acrilonitrila Butadieno Estireno
PLA	Polylactic Acid
PETG	Polyethylene Terephthalate Glycol
PVA	Polyvinyl Alcohol
DC	Direct Current
GND	Ground

SUMÁRIO

1 INTRODUÇÃO

No cenário do século 21, observou-se uma mudança de paradigma, caracterizada pelo crescimento notável de sistemas autônomos em ambientes não industriais, como residências, hospitais e escolas. Esses sistemas, desempenhando funções como assistentes virtuais para o controle de luzes e temperatura, impactaram setores como o automotivo, evidenciado pelo avanço dos veículos autônomos, exemplificados pela liderança da Tesla e projetos como o carro autônomo da Google. A prevalência da visão computacional para controlar a trajetória desses veículos suscita a questão fundamental: estão esses sistemas verdadeiramente preparados para lidar com variações em ambientes não lineares e sujeitos a mudanças temporais?

A hipótese apresentada propõe que muitos veículos, ao seguir trajetórias, enfrentam limitações devido ao uso de sensores digitais, restringindo assim o controle sobre as variações entre a trajetória planejada e a executada. Argumenta-se que a inclusão de visão computacional, por meio do processo de segmentação de imagem, permite medir essas variações. Adicionalmente, a introdução de um controlador *Fuzzy* revela-se eficaz para lidar com trajetórias não predefinidas, utilizando a variação dos valores de pixels das imagens como entrada.

A visão computacional, ao identificar e rastrear objetos através da segmentação de imagem, aliada ao controlador *Fuzzy*, emerge como uma abordagem crucial para o controle de trajetória em veículos autônomos. Este sistema utiliza a variação entre a trajetória esperada e a trajetória realizada como entrada, permitindo que o controlador *Fuzzy* ajuste o PWM dos motores para manter o veículo dentro de uma margem de segurança predefinida. Este cenário é especialmente relevante em sistemas não lineares e sujeitos a variações temporais.

A justificativa para esta pesquisa reside na estreita relação entre visão computacional e controle, essenciais para o desenvolvimento de sistemas autônomos interagindo com ambientes não lineares do mundo real. A implementação de controladores não lineares, aliados à visão computacional, é crucial para veículos autônomos, promovendo um trânsito mais seguro, livre de intervenções humanas propensas a acidentes. Além da aplicação em veículos, essa abordagem estende-se a robôs autônomos, proporcionando segurança em atividades perigosas, como resgates, desativação de bombas e exploração de ambientes hostis, onde a capacidade de adaptação a adversidades é essencial, impulsionada pela inteligência artificial. Assim, a pesquisa proposta busca contribuir para avanços significativos na integração eficaz de visão computacional e controle em sistemas autônomos, ampliando suas aplicações em diversos cenários.

1.1 Objetivos

1.1.1 Objetivo geral

Aplicação das técnicas de controle PID e *Fuzzy*, em conjunto com processamento digital de imagem, em um robô seguidor de linha para verificar e comparar a eficiência desses controladores na tarefa de seguir uma trajetória.

1.1.2 Objetivos específicos

1. Pesquisar estudos sobre a aplicação de Processamento Digital de Imagens (PDI) aliado com controlador do tipo *Fuzzy* e PID;
2. Projetar o modelo 3D do robô seguidor de linha e realizar sua manufatura;
3. Fazer a identificação do sistema do robô seguidor de linha para aprimorar a eficiência do controlador;
4. Implementar o código de segmentação das imagens capturadas pela câmera do robô;
5. Projetar o controlador *Fuzzy* e PID utilizando os valores de variação da imagem processados pelo PDI;
6. Integrar o controlador *Fuzzy* e o PID no Raspberry Pi 4;
7. Realizar testes para avaliar a acurácia do robô em diferentes trajetórias;
8. Comparar o desempenho das técnicas de controle PID e *Fuzzy* em conjunto com o processamento digital de imagem para determinar a abordagem mais eficaz na tarefa de seguir uma trajetória.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os livros e estudos que foram utilizados como base do estudo para o desenvolvimento desse projeto, tanto a parte teórica quanto a parte prática. Os conteúdos que vão ser abordados são conceitos sobre robótica e processamento digital de imagem, seguido de noções básicas sobre o controle *Fuzzy* e como ele pode ser aplicado em sistemas não lineares e variantes no tempo. Finalizando com um estudo sobre identificação de sistemas e como isso impacta na melhoria do projeto de controle.

2.1 Robótica

2.1.1 Introdução

De acordo com as afirmações de Niku (2020), os robôs são construídos e projetados para serem controlados por um computador ou um dispositivo semelhante. Esses dispositivos executam um programa que controla os movimentos do robô através de um controlador. Alterações no programa causam mudanças nas ações do robô, tornando-o extremamente flexível e capaz de realizar diversas tarefas sem precisar passar por novas etapas de projetos. Ao contrário, um simples manipulador não pode realizar tais tarefas sem a operação contínua de um operador. Existem padrões diferentes para definir o que é considerado um robô em diferentes países, sendo que nos Estados Unidos um dispositivo deve ser facilmente reprogramável para ser considerado um robô, enquanto dispositivos de manuseio manual ou robôs de sequência fixa não são incluídos nessa categoria.

De acordo com Siegwart e Nourbakhsh (2004), na área de robótica móvel é necessário compreender o comportamento mecânico dos robôs para projetar máquinas móveis adequadas para as tarefas e criar software de controle para controlar o hardware do robô móvel. Essa análise não é exclusiva para robôs móveis, já que os robôs manipuladores também exigem estudos aprofundados há mais de três décadas. De fato, os robôs manipuladores são ainda mais complexos do que os primeiros robôs móveis, com um robô padrão de soldagem podendo ter mais de cinco articulações, enquanto as primeiras máquinas móveis eram simples máquinas de acionamento diferencial. Nos últimos anos, a comunidade de robótica alcançou um nível considerável de entendimento da cinemática e dinâmica dos robôs manipuladores.

2.1.2 Cinemática local

A cinemática local é a descrição do movimento de uma parte do robô em relação ao seu sistema de coordenadas local. Ela é importante para a análise de movimentos

individuais, como a rotação de uma junta ou o movimento de uma extremidade do robô. A cinemática local envolve a utilização de matrizes de transformação homogêneas para descrever o movimento de cada junta do robô em relação ao seu sistema de coordenadas local.

Segundo Spong, Hutchinson e Vidyasagar (2020), a cinemática local é geralmente representada por matrizes de transformação homogêneas, que são usadas para descrever a posição e orientação de cada junta do robô em relação à sua base. As matrizes de transformação homogêneas descrevem a rotação e translação de um objeto no espaço tridimensional e são fundamentais para o controle de robôs.

2.1.3 Cinemática global

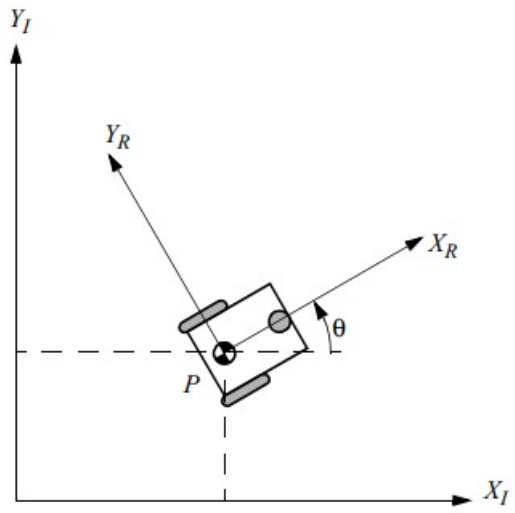
A cinemática global é a descrição do movimento do robô como um todo, em relação a um sistema de coordenadas global. Ela é importante para a análise do movimento do robô como um todo em relação a sua posição no espaço. A cinemática global envolve a utilização de equações de transformação para descrever o movimento do robô como um todo.

Conforme afirmado por Spong, Hutchinson e Vidyasagar (2020), a cinemática global envolve a utilização de equações de transformação para descrever a posição e orientação do robô em relação ao sistema de coordenadas global. Essas equações permitem determinar a posição e orientação do robô em qualquer ponto no espaço.

2.1.4 Cinemática da trajetória de robôs

Podemos estabelecer uma relação entre o referencial global do plano e o referencial local do robô, ilustrado na Figura 1, a fim de especificar a posição do robô no plano. Utilizando os eixos X_1 e Y_1, podemos definir uma base inercial arbitrariamente no plano como referencial global, tendo como origem O:$\{X_1, Y_1\}$. Para especificar a posição do robô, é necessário selecionar um ponto P no chassi do robô como referência de posição. A base $\{X_R, Y_R\}$ define dois eixos relativos a P no chassi do robô, que é o referencial local do robô. As coordenadas X e Y indicam a posição de P no referencial global, enquanto θ representa a diferença angular entre o referencial global e o local. É possível descrever a pose do robô como um vetor com esses três elementos, ilustrado na Equação 2.1.

Figura 1 – O referencial global e o referencial local do robô.



Fonte: Siegwart e Nourbakhsh (2004, p. 49)

$$\xi_1 = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.1}$$

A fim de explicar o movimento do robô em termos de movimentos de componentes, é preciso transformar o movimento ao longo dos eixos do referencial global em movimento ao longo dos eixos do referencial local do robô. É importante ressaltar que essa transformação depende da posição atual do robô. Essa conversão é realizada através do uso da matriz de rotação ortogonal:

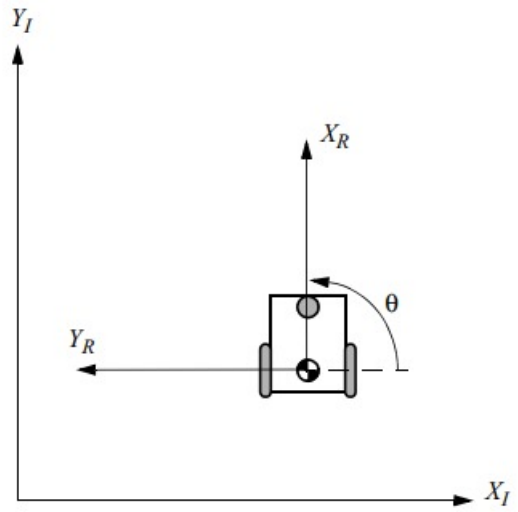
$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Essa matriz pode ser empregada para converter o movimento no referencial global $\{X_1, Y_1\}$ em movimento com relação ao referencial local $\{X_R, Y_R\}$. A designação $R(\theta)\dot{\xi}_I$ é utilizada porque o cálculo dessa operação depende do valor de θ: Por exemplo, considere o robô na Figura 2. Para este robô, devido $\theta = \frac{\pi}{2}$ é possível calcular facilmente a matriz de rotação instantânea R, utilizando as equações Equação 2.3 e Equação 2.4.

$$\dot{\xi}_R = R\left(\frac{\pi}{2}\right)\dot{\xi}_I \tag{2.3}$$

$$R\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Figura 2 – O referencial global e o referencial local do robô.



Fonte: Siegwart e Nourbakhsh (2004, p. 50)

Dado uma velocidade $(\dot{x}, \dot{y}, \dot{\theta})$ no referencial global, podemos calcular os componentes de movimento ao longo dos eixos X_R e Y_R locais desse robô. Nesse caso, devido ao ângulo específico do robô, o movimento ao longo de X_R é igual a $\dot{y}$ e o movimento ao longo de Y_R é -$\dot{x}$:

$$\dot{\xi_R} = R\left(\frac{\pi}{2}\right)\dot{\xi_I} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{pmatrix} \tag{2.5}$$

2.2 Visão computacional

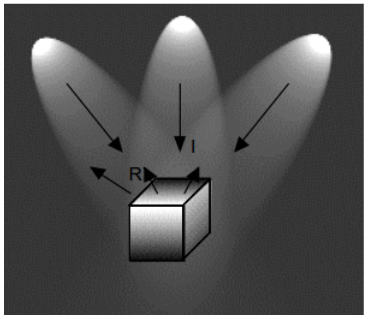
Segundo Backes, Junior e Mesquita (2016), a área de estudo que busca dotar máquinas com a capacidade da visão é chamada de visão computacional, de acordo com a definição apresentada no texto. A visão não se resume à captura de imagens, mas envolve um processo complexo e rico que inclui aprimoramento das imagens, como remoção de ruídos e aumento de contraste, separação de regiões ou objetos de interesse em uma cena, extração de várias informações da imagem, como forma, cor e textura, e relacionamento das imagens com outras vistas anteriores. Embora a captura de imagens seja uma habilidade impressionante (como evidenciado pela complexidade do funcionamento do olho), é apenas o começo do processo de visão.

2.2.1 Processamento digital de Imagem

De acordo Filho e Neto (1999) com , uma imagem em escala de cinza pode ser matematicamente descrita por uma função f(x,y) que representa a intensidade luminosa em cada ponto espacial (x,y). Essa intensidade é proporcional ao brilho da imagem no ponto correspondente. A função f(x,y) é o resultado da interação entre a iluminância i(x,y), que indica a quantidade de luz que incide sobre o objeto, e as propriedades de refletância ou transmitância do objeto, representadas pela função r(x,y). A função r(x,y) indica a fração de luz incidente que o objeto reflete ou transmite no ponto (x,y). Esses conceitos são demonstrados na Equação 2.6.

$$f(x,y) = i(x,y).r(x,y) \; com \; 0 < i(x,y) < \infty \; e \; 0 < r(x,y) < 1 \tag{2.6}$$

Figura 3 – Os componentes iluminância (I) e refletância (R) de uma imagem.



Fonte: Filho e Neto (1999, p. 20)

O processamento digital de imagens é uma área que envolve o processamento de imagens digitais por meio de um computador. Vale destacar que uma imagem digital é formada por um número finito de elementos que possuem valores e localizações específicas, e esses elementos são conhecidos como elementos pictóricos, pels, elementos de imagem ou pixels. Entre esses termos, o mais comum para se referir aos elementos de uma imagem digital é o pixel, (Gonzalez; Woods, 2009).

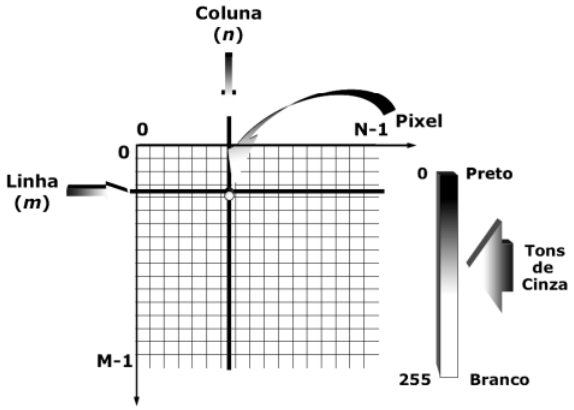
2.2.2 Digitalização

Para tornar o sinal analógico de vídeo adequado para o processamento de imagens por computador, é necessário realizar uma discretização tanto espacial quanto em amplitude. A etapa de discretização espacial é conhecida como amostragem e a de discretização em amplitude é denominada quantização. Durante a amostragem, a imagem analógica é convertida em uma matriz de pixels, com tamanho M por N. Cada ponto da matriz corresponde a um pixel ou elemento de imagem:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \vdots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix} \tag{2.7}$$

A quantização faz com que cada um dos pixels assuma um valor inteiro que está na faixa de 0 a 2^n-1. Quanto maior for o valor de n, maior vai ser o número de níveis de cinza presentes na imagem digitalizada. A Figura 4, é apresentada uma matriz de MxN onde seus valores variam de 0 (preto) a 255 (branco) que são os tons de cinza que os pixels podem ter.

Figura 4 – Matriz MxN dos pixels.



Fonte: Queiroz e Gomes (2006, p. 8)

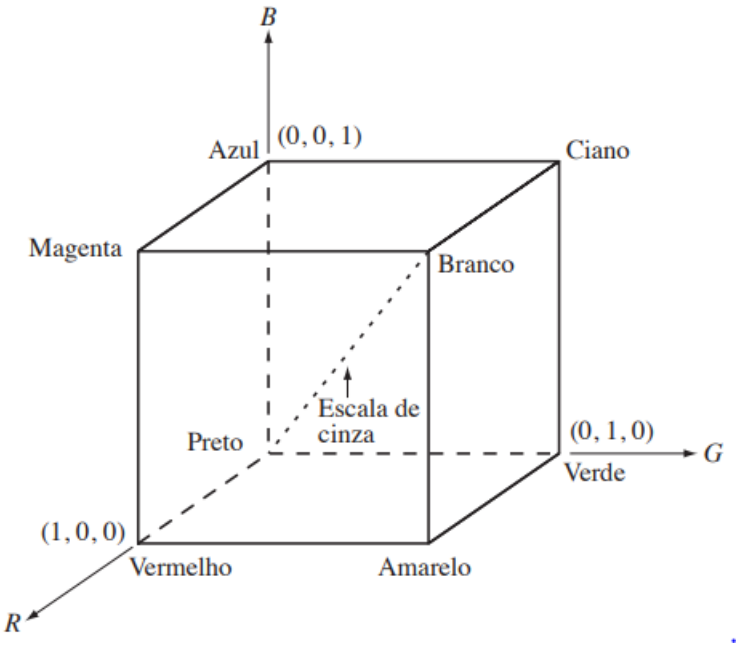
2.2.3 Modelos de representação de cores

O propósito dos modelos de cores é estabelecer uma forma padronizada e universalmente aceita para especificar cores. Essencialmente, um modelo de cores é uma representação em três dimensões em que cada cor é identificada por um ponto no sistema de coordenadas tridimensionais. A maioria dos modelos de cores utilizados atualmente foi desenvolvida para aplicações específicas em hardware (por exemplo, impressoras ou monitores de cores) ou para programas que manipulam cores. Entre os modelos mais empregados para representação de cores estão o RGB (vermelho, verde e azul), o CMY (ciano, magenta e amarelo), o CMYK (uma variação do modelo CMY, em que K representa a cor preta), o YCbCr (padrão normalizado pela recomendação ITU-R BT.601 e utilizado em várias técnicas de compressão de vídeo), o YIQ (padrão de TV NTSC para cores) e o HSI (matiz, saturação e intensidade), que às vezes é chamado de HSV (matiz, saturação e valor),(Filho; Neto, 1999).

2.2.4 Modelo RGB

No sistema de cores RGB, cada cor é composta pelos componentes espectrais primários de vermelho, verde e azul. Esse modelo utiliza um sistema de coordenadas cartesianas e o cubo que contém as cores de interesse é apresentado na Figura 5. Os valores primários de RGB são encontrados em três vértices, enquanto as cores secundárias (ciano, magenta e amarelo) estão em outros três vértices. O preto está localizado na origem e o branco no vértice mais distante da origem. O modelo RGB apresenta uma escala de cinza que varia do preto ao branco ao longo de um segmento de reta que une esses dois pontos. As diferentes cores são representadas como pontos dentro do cubo ou sobre ele, definidas por vetores que se estendem a partir da origem. Para facilitar, é assumido que todos os valores de cor foram normalizados e, portanto, o cubo apresentado na Figura 5 é o cubo unitário, onde os valores de R, G e B variam no intervalo de [0,1].

Figura 5 – Esquema do cubo de cores RGB.



Fonte: Filho e Neto (1999, p. 121)

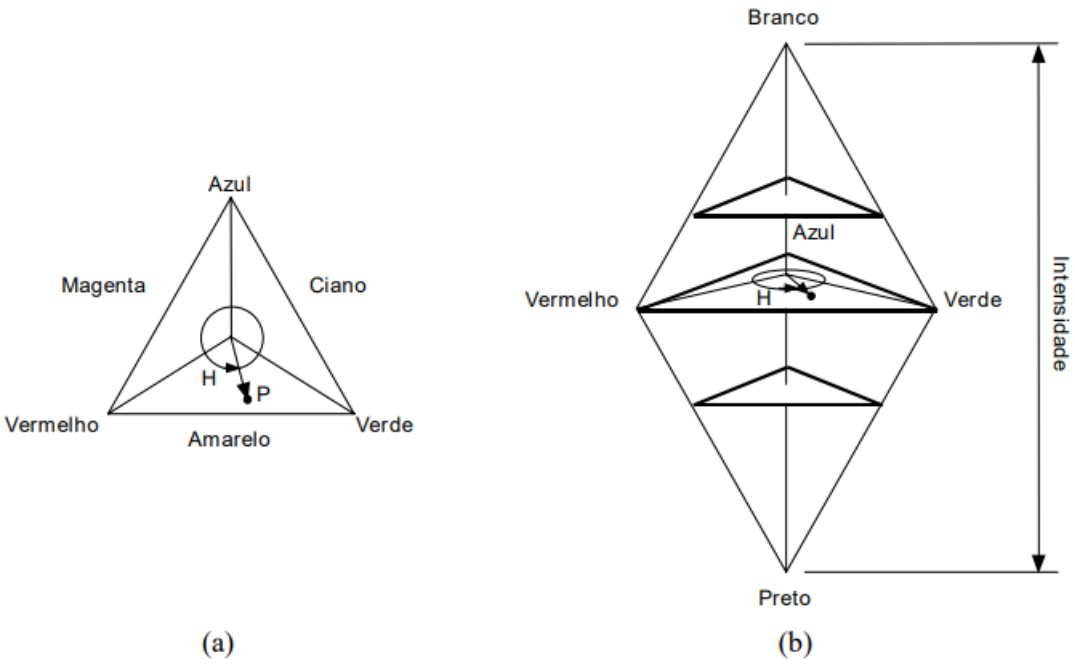
Cada pixel em uma imagem colorida pode ser considerado como um vetor com componentes que correspondem aos valores de intensidade das cores vermelha, verde e azul. As imagens coloridas são criadas através da combinação de três imagens monocromáticas que representam as cores RGB.

$$f(x, y) = f_R(x, y) + f_G(x, y) + f_B(x, y) \tag{2.8}$$

2.2.5 Modelo HSI

O modelo HSI é muito útil porque permite separar as componentes de matiz, saturação e intensidade da informação de cor em uma imagem, de forma semelhante à percepção humana de cores. Esse modelo é amplamente utilizado em sistemas de visão artificial que se baseiam na percepção de cor pelo ser humano, como em um sistema automatizado de colheita de frutas que precisa avaliar a maturidade da fruta com base em sua coloração externa. Geometricamente, o modelo HSI pode ser representado como um sólido (indicado na Figura 6(b)), em que os cortes horizontais produzem triângulos (Figura 6(a)). Os vértices desses triângulos representam as cores primárias, e o centro corresponde à combinação dessas cores em proporções iguais, que pode estar mais próxima do preto ou do branco, dependendo da altura do corte realizado.

Figura 6 – Esquema do cubo de cores HSI.

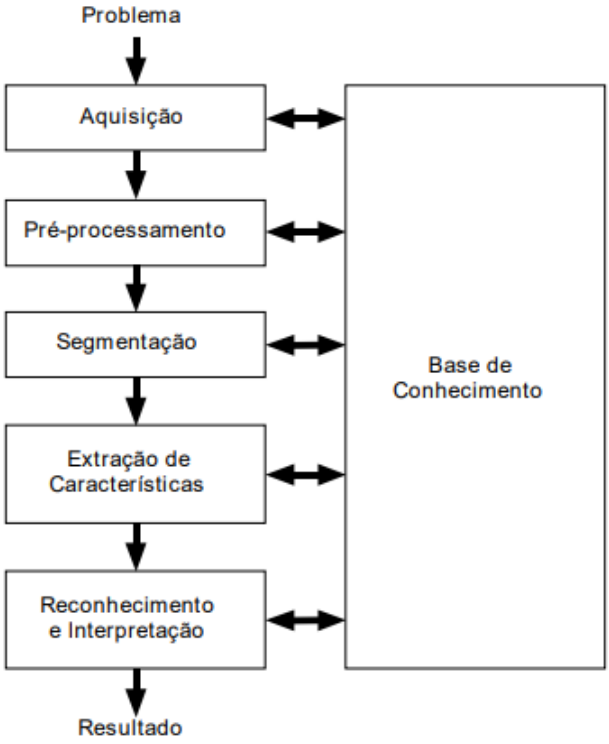


Fonte: Filho e Neto (1999, p. 123)

2.2.6 Etapas de Processamento Digital de Imagem

Para realizar o Processamento Digital de Imagens (PDI), é fundamental seguir uma sequência que envolva a captura, processamento, segmentação, extração de características, reconhecimento e interpretação da imagem. Essas etapas são de extrema importância para garantir a obtenção das características desejadas. A Figura 7 oferece uma representação visual das fases mencionadas anteriormente.

Figura 7 – Passos fundamentais em processamento digital de imagens.



Fonte: Filho e Neto (1999, p. 9)

2.2.7 Captura de Imagens

De acordo com Filho e Neto (1999), a etapa inicial do processo envolve a aquisição de imagens dos envelopes. Essa operação requer a utilização de um sensor e de um digitalizador. O sensor desempenha o papel de converter informações ópticas em sinais elétricos, enquanto o digitalizador realiza a transformação da imagem analógica para o formato digital.

A captura de imagens é o primeiro estágio crucial no processo de Processamento Digital de Imagens (PDI). Essa fase consiste na obtenção de dados visuais a partir de fontes diversas, como câmeras digitais, scanners, satélites ou outros dispositivos de aquisição de imagens.

Durante a captura, os dispositivos convertem informações visuais do mundo real em formato digital, representando a imagem por meio de pixels. Cada pixel contém dados sobre a cor, intensidade e outras propriedades da luz naquele ponto específico da imagem.

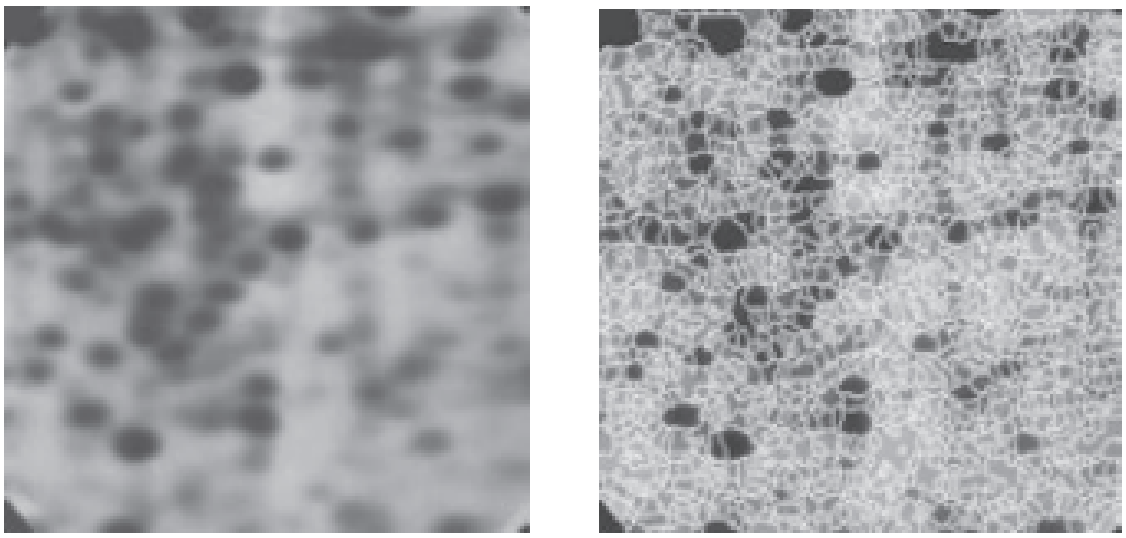
2.2.8 Pré-processamento

Esta fase visa preparar e aprimorar as imagens para análises posteriores, removendo imperfeições, reduzindo ruídos e realçando características de interesse.

As técnicas de pré-processamento são aplicadas para melhorar a qualidade visual das imagens, tornando-as mais adequadas para análises automáticas ou manuais. Isso pode incluir operações como correção de contraste, ajuste de brilho, remoção de ruído, nitidez de bordas, normalização de cores e redimensionamento.

Além disso, o pré-processamento também pode envolver etapas específicas de preparação de imagens para determinadas aplicações, como a segmentação de regiões de interesse, binarização para destacar objetos de interesse ou mesmo a eliminação de artefatos indesejados. Essas transformações podem ser observadas na Figura 8.

Figura 8 – Exemplo de Pré-processamento.



(a) Imagem original, antes do pré-processamento ser aplicado.

(b) Resultado da aplicação do algoritmo de segmentação de watershed.

Fonte: Gonzalez e Woods (2009, p. 510)

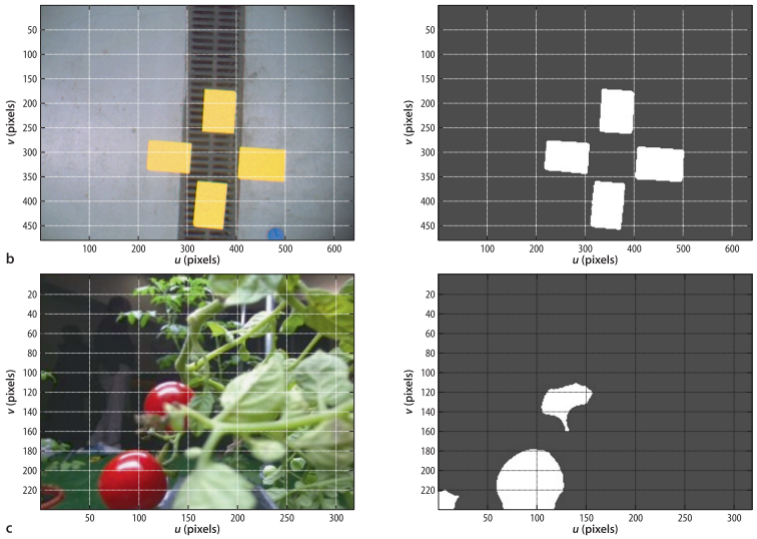
2.2.9 Segmentação de Imagens

De acordo com Queiroz e Gomes (2006), a segmentação é o processo de dividir uma imagem em regiões ou objetos que a compõem. O nível de detalhe da subdivisão é determinado pelo problema a ser resolvido, ou seja, a segmentação deve ser interrompida assim que os objetos ou regiões de interesse para a aplicação forem identificados. Por exemplo, na inspeção automatizada de componentes eletrônicos, o objetivo é analisar imagens dos produtos para detectar anomalias específicas, como a falta de componentes ou circuitos de conexão interrompidos. Não há razão para continuar a segmentação além do nível necessário para identificar esses elementos.

De acordo com Corke e Khatib (2011), a segmentação de imagem é o processo de dividir uma imagem em regiões significativas para a aplicação, conforme mostrado na Figura 9. O objetivo é separar os pixels que representam os objetos de interesse de todos os

outros pixels da cena. Embora seja uma abordagem conceitualmente simples, é um desafio significativo na prática. Um requisito fundamental para a segmentação é a robustez, que se refere à capacidade do método de se adaptar às mudanças nas condições da cena, como variações na iluminação ou no ponto de vista.

Figura 9 – Segmentação baseada na zona de interesse.



Fonte: Gonzalez e Woods (2009, p. 414)

2.2.10 Extração de Características

A etapa subsequente tem como objetivo extrair características das imagens resultantes da segmentação por meio de descritores que permitam uma caracterização precisa de cada dígito e tenham uma boa capacidade de discriminação entre dígitos semelhantes, como o "5"e o "6". Estes descritores devem ser representados por uma estrutura de dados adequada para o algoritmo de reconhecimento. É importante ressaltar que, nesta etapa, a entrada ainda é uma imagem, mas a saída é um conjunto de dados correspondente a essa imagem. Por exemplo, pode-se considerar as coordenadas normalizadas x e y do centro de gravidade do dígito e a razão entre sua altura e largura como descritores para descrevê-lo. Nesse caso, um vetor de três elementos pode ser uma estrutura de dados adequada para armazenar essas informações para cada dígito processado nessa etapa, (Filho; Neto, 1999).

2.2.11 Reconhecimento e Interpretação

Conforme descrito por Gonzalez e Woods (2009), o reconhecimento é o processo de associar um rótulo a um objeto com base em suas características, que são expressas por meio de seus descritores. Nesse contexto, a interpretação refere-se à atribuição de significado a um conjunto de objetos previamente reconhecidos.

O reconhecimento de padrões geralmente envolve a aplicação de algoritmos de aprendizado de máquina, técnicas de visão computacional ou processamento de sinais para identificar objetos, rostos, padrões geográficos, entre outros elementos presentes nas imagens.

Já a interpretação de imagens consiste na análise e compreensão do contexto em que esses padrões ou objetos são encontrados. Isso pode incluir a análise do ambiente em que a imagem foi capturada, a relação entre diferentes objetos identificados, ou mesmo a inferência de informações mais complexas com base nos elementos presentes na imagem.

2.2.12 Binarização

Segundo Marchand-Maillet e Sharaiha (1999), as imagens em tons de cinza geralmente possuem 8 bits por pixel. Embora o processamento dessas imagens seja, de certa forma, mais fácil do que o processamento de imagens coloridas, existe uma forma mais simples de imagem, a imagem binária, na qual o processamento é ainda mais direto. Na verdade, uma parte significativa das aplicações práticas de visão computacional foi desenvolvida usando visão binária.

Uma imagem binária é aquela em que há apenas um único bit por pixel (ou seja, preto ou branco). Essas imagens são criadas por meio de limiarização, em que os limiares utilizados são determinados de várias maneiras. As imagens binárias resultantes são frequentemente pós-processadas usando morfologia matemática , e as regiões binárias segmentadas resultantes são extraídas da imagem usando análise de componentes conectados, isso pode ser visto na Figura 10.

Figura 10 – Aplicação De Binarização.

Fonte: Sha, Hou e Cui (2016, p. 11)

2.2.13 Morfologia matemática

A Morfologia Matemática, originalmente concebida por Coster e Chermant. (1982), concentra-se na análise da estrutura geométrica das entidades presentes em imagens. Essa abordagem encontra aplicação em diversas áreas do processamento e análise de

imagens, abrangendo objetivos como realce, filtragem, segmentação, detecção de bordas, esqueletização, afinamento, entre outros.

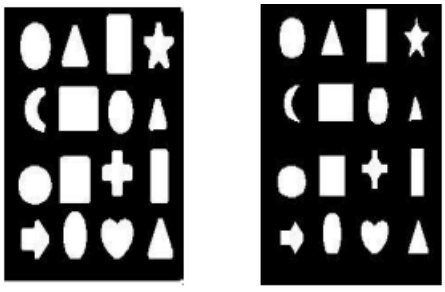
O princípio fundamental da morfologia matemática reside na extração de informações relacionadas à geometria e topologia de um conjunto desconhecido (representado por uma imagem). Isso é realizado por meio da transformação desse conjunto usando outro conjunto previamente definido, denominado elemento estruturante. Assim, a teoria de conjuntos constitui a base essencial da morfologia matemática.

Por exemplo, considere o conjunto de todos os pixels pretos em uma imagem binária. Esse conjunto descreve integralmente a imagem, uma vez que os demais pontos só podem ser brancos. Em imagens binárias, os conjuntos em questão pertencem ao espaço inteiro bidimensional **Z2**, onde cada elemento do conjunto é um vetor 2D com coordenadas (x, y) representando a posição do pixel preto na imagem. Para imagens com mais níveis de cinza, a representação é estendida a conjuntos cujos elementos pertencem ao espaço **Z3**. Nesse caso, os vetores têm três elementos, sendo os dois primeiros as coordenadas do pixel e o terceiro o seu nível de cinza.

2.2.14 Erosão de Imagens

A erosão de imagens é um processo morfológico que envolve a remoção de pixels da borda da região de interesse. Essa técnica é comumente utilizada para suavizar contornos, eliminar detalhes indesejados e realçar características essenciais. No âmbito do processamento de imagens médicas, a erosão pode ser empregada na segmentação de estruturas anatômicas, contribuindo para uma análise mais precisa e eficiente, isso pode ser visto na Figura 11.

Figura 11 – Erosão da imagem.



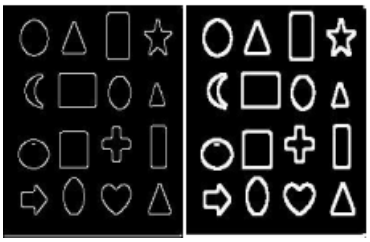
Fonte: Chudasama *et al.* (2015, p. 18)

2.2.15 Dilatação de Imagens

Por outro lado, a dilatação de imagens consiste na adição de pixels à borda da região de interesse. Essa técnica é valiosa para a expansão de objetos e o preenchimento de

lacunas, favorecendo a conectividade e melhorando a detecção de características específicas. Na área de reconhecimento de padrões, a dilatação de imagens é frequentemente aplicada para realçar a identificação de objetos em cenários complexos, isso pode ser visto em Figura 12.

Figura 12 – Dilatação da imagem.



Fonte: Chudasama *et al.* (2015, p. 18)

2.3 Controlador PID

O controle Proporcional, Integral e Derivativo (PID) é uma técnica amplamente empregada em sistemas de controle automático, oferecendo uma abordagem eficaz para estabilizar e otimizar o desempenho de sistemas dinâmicos. A teoria PID é fundamental para compreender a dinâmica e a resposta de sistemas em tempo real, proporcionando uma base sólida para implementações práticas em diversas áreas da engenharia.

Segundo Ogata (2010, p. 512), a utilidade dos controles PID está na sua aplicabilidade geral à maioria dos sistemas de con-trole. Em particular, quando o modelo matemático da planta não é conhecido e, portanto, métodosde projeto analítico não podem ser utilizados, controles PID se mostram os mais úteis. Na áreados sistemas de controle de processos, sabe-se que os esquemas básicos de controle PID e oscontroles PID modificados provaram sua utilidade conferindo um controle satisfatório, emboraem muitas situações eles possam não proporcionar um controle ótimo. O PID pode ser definido utilizando o diagrama da Figura 13.

A componente proporcional (P) é responsável por ajustar a saída proporcionalmente ao erro, que é a diferença entre o valor desejado e o valor real do sistema. Essa ação visa reduzir o erro proporcionalmente, proporcionando uma resposta rápida a perturbações.

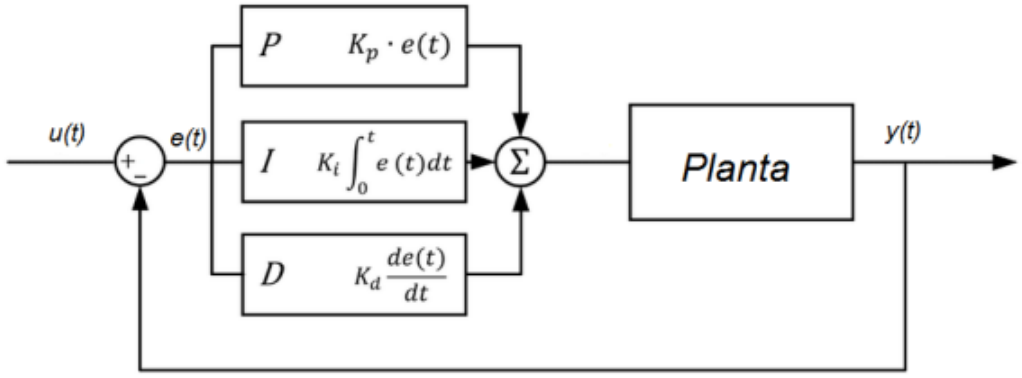
A componente integral (I) atua na correção de erros acumulados ao longo do tempo. Essa ação é crucial para eliminar o erro persistente e reduzir o efeito de distúrbios de longa duração, contribuindo para a estabilidade do sistema.

A componente derivativa (D) visa prevenir a ocorrência de oscilações ao ajustar a saída com base na taxa de variação do erro. Essa ação é particularmente eficaz na

minimização de overshooting e na resposta a mudanças abruptas.

A combinação ponderada dessas três componentes, ajustadas por constantes de ganho (Kp, Ki, Kd), forma o controlador PID. Sua capacidade de se adaptar dinamicamente às condições do sistema torna-o uma ferramenta versátil para o controle preciso de sistemas dinâmicos em uma variedade de aplicações, incluindo automação industrial, robótica, controle de motores e processos de manufatura.

Figura 13 – Diagrama de blocos do controlador PID.



Fonte: Bastos *et al.* (2019, p. 40)

2.4 Controlador *Fuzzy*

Os controladores *Fuzzy* são sistemas de controle baseados na lógica *Fuzzy*, que permite a modelagem de sistemas imprecisos ou incertos. A lógica *Fuzzy* estende a lógica booleana ao permitir a descrição de um sistema através de conjuntos *Fuzzy*, que são conjuntos onde um elemento pode ter um grau de pertinência parcial a um conjunto, em vez de ser totalmente ou não pertencente.

Segundo Klir e Yuan (1995), os controladores *Fuzzy* são capazes de utilizar conhecimentos obtidos de operadores humanos, o que é crucial em problemas de controle onde é difícil ou impossível construir modelos matemáticos precisos. Essas dificuldades podem ser causadas por diversos fatores, como não-linearidades inerentes, perturbações ambientais imprevisíveis e sensores com degradação. Observou-se que operadores humanos experientes geralmente são capazes de desempenhar bem nessas circunstâncias. Assim, o conhecimento de um operador humano experiente pode ser usado como alternativa a um modelo preciso do processo controlado, expresso em um conjunto de regras de controle *Fuzzy*, um exemplo disso é:

SE a temperatura estiver muito alta

E a pressão é ligeiramente baixa
ENTÃO a mudança de calor deve ser ligeiramente negativa,

De acordo com Jang (1993), um controlador *Fuzzy* pode ser definido como um sistema de inferência *Fuzzy* composto de três partes principais: a etapa de fuzzificação, a etapa de inferência e a etapa de defuzzificação. A etapa de fuzzificação consiste em converter a entrada do sistema em valores de pertinência para cada um dos conjuntos *Fuzzy* que descrevem as variáveis do sistema. A etapa de inferência aplica as regras *Fuzzy* para determinar a saída do sistema, que é representada como um conjunto *Fuzzy*. A etapa de defuzzificação converte o conjunto *Fuzzy* de saída em um valor numérico, que é a saída final do controlador.

2.4.1 Variável linguística

Klir e Yuan (1995) afirmam que o conceito de número *Fuzzy* é fundamental na formulação de variáveis quantitativas *Fuzzy*. Essas variáveis têm estados representados por números *Fuzzy*, e quando esses números são utilizados para representar conceitos linguísticos, como muito pequeno, pequeno, médio, entre outros, em um contexto específico, as variáveis resultantes são chamadas de variáveis linguísticas. Cada variável linguística é definida em termos de uma variável base, que tem valores numéricos reais dentro de uma faixa específica. A variável base pode ser uma variável física, como temperatura, pressão, umidade, ou qualquer outra variável numérica, como idade, desempenho, probabilidade, entre outras. Em uma variável linguística, os termos linguísticos que representam valores aproximados da variável base são representados por números *Fuzzy*.

As variáveis linguísticas podem ser definidas em termos de uma variável base, que é uma variável numérica convencional, mas cujos valores são representados como números *Fuzzy*. As variáveis base podem ser qualquer coisa, desde a temperatura até a velocidade, e os termos linguísticos usados para descrevê-las podem variar de acordo com o contexto.

Em resumo, as variáveis linguísticas em controle *Fuzzy* são usadas para representar conceitos vagos e imprecisos em termos de números *Fuzzy*, permitindo que o sistema de controle *Fuzzy* lide com entradas e saídas que são difíceis de quantificar de outra forma.

2.4.2 Função de Pertinência

A função de pertinência é uma função matemática que descreve a relação de pertinência ou grau de pertinência de um elemento a um conjunto *Fuzzy*. Essa função é usada no processo de fuzzificação, que é a conversão de valores de entrada em seus respectivos valores *Fuzzy*. A função de pertinência atribui a cada valor de entrada um valor de pertinência, que indica o grau em que esse valor pertence ao conjunto *Fuzzy*. Ela é usada para definir a forma como o conjunto *Fuzzy* é representado, bem como para calcular

as operações de união, interseção e complemento de conjuntos *Fuzzy*. Existem vários tipos de funções de pertinência, cada uma com sua própria forma e características específicas. Algumas das funções de pertinência mais comuns incluem a função triangular, a função trapezoidal, a função gaussiana e a função sigmoidal.

2.4.3 Fuzzificação

De acordo com Niku (2020), a fuzzificação é o procedimento de transformar os valores de entrada e saída em suas respectivas funções de pertinência. O resultado da fuzzificação é uma coleção de gráficos ou equações que descrevem o grau de pertinência de diferentes valores em variáveis *Fuzzy* distintas.

Para fuzzificar uma variável, o intervalo possível de seus valores é dividido em vários conjuntos, cada um deles referente a uma porção particular do intervalo. Em seguida, cada conjunto é representado por uma equação ou um gráfico que caracteriza o grau de verdade ou pertinência de cada valor dentro do intervalo. O número de conjuntos, o intervalo que cada conjunto representa e o tipo de representação são arbitrários e uma escolha do projetista do sistema.

Existem várias representações possíveis para cada conjunto. Se alguém criar seu próprio sistema *Fuzzy*, é possível usar qualquer representação que pareça apropriada. No entanto, quando se utiliza um sistema comercial, pode haver limitações em relação ao que está disponível.

2.4.4 Conjunto de regras

Os conjuntos de regras de um controlador *Fuzzy* são um conjunto de regras lógicas que relacionam as variáveis de entrada do sistema *Fuzzy* as variáveis de saída. Essas regras são mantidas nas funções de pertinência das variáveis de entrada e saída, e geralmente são expressas em termos de "se...então". Por exemplo, uma regra pode ser formulada como "se a temperatura é alta e a umidade é baixa, então o ar condicionado deve ser ligado em alta potência".

As regras do controlador *Fuzzy* são usadas para definir como o sistema *Fuzzy* deve responder a diferentes entradas e condições do ambiente. O processo de inferência é usado para determinar a saída do sistema *Fuzzy* com base nas regras de entrada e nas condições atuais do ambiente. As regras podem ser escritas de forma manual pelo designer do sistema *Fuzzy* ou podem ser aprendidas automaticamente usando algoritmos de aprendizado de máquina.

2.4.5 Desfuzzificação

Desfuzzificação é o processo de transformar a saída de um controlador *Fuzzy*, que é uma representação *Fuzzy* da resposta desejada, em uma resposta numérica ou crisp que possa ser usada para controlar um sistema. Em outras palavras, é o processo de converter uma saída *Fuzzy* em um valor específico. Existem várias técnicas de desfuzzificação disponíveis, como o centro de gravidade, o máximo do máximo e o método do menor erro quadrático. A escolha da técnica de desfuzzificação depende do sistema específico e dos requisitos de controle.

2.5 Impressão 3D

Na impressão 3D, são empregados diversos tipos de filamento, como PLA, Flex, ABS, PETG, entre outros. Cada filamento possui características específicas que o tornam mais adequado para diferentes tipos de impressoras. Além disso, há modelos de impressoras direcionados para a produção de casas, peças metálicas e até alimentos. Isso implica o uso de processos de impressão distintos entre si, evidenciando a necessidade de existirem diferentes modelos de impressoras que empregam diversas técnicas de impressão.

Diversas tecnologias podem ser aplicadas em impressoras 3D, sendo as mais comuns FDM ou FFF, SLA, DLP, SLS e SLM. As divergências entre essas tecnologias residem nos materiais utilizados e na forma como são construídas as camadas para a formação dos objetos impressos.

2.5.1 Impressoras FDM

A tecnologia de Modelagem por Deposição Fundida (FDM), também conhecida como Fabricação por Filamento Fundido (FFF), é uma das abordagens mais difundidas na impressão 3D. Nesse método, um filamento termoplástico, como PLA, ABS ou PETG, é aquecido até atingir o estado líquido e, em seguida, depositado camada por camada para construir o objeto desejado.

A versatilidade do FDM permite a utilização de diversos tipos de filamentos, cada um com propriedades específicas. O PLA, por exemplo, é amplamente escolhido devido à sua natureza biodegradável e facilidade de impressão, enquanto o ABS oferece resistência térmica e durabilidade. A flexibilidade do filamento Flex é ideal para aplicações que requerem elasticidade.

As impressoras FDM são populares não apenas pela diversidade de materiais, mas também pela sua aplicabilidade em vários setores. Elas são empregadas na criação de protótipos, peças de uso doméstico, componentes industriais e até mesmo em projetos inovadores, como a construção de estruturas habitacionais e a produção de alimentos.

A tecnologia FDM destaca-se pelo seu baixo custo inicial, simplicidade operacional e ampla disponibilidade de filamentos, tornando-a mais acessível que outros modelos de impressoras. Ela é mais fácil de ser montada, pois suas peças, como motores de passo, estruturas finas de alumínio e uma mesa aquecida, são de baixo custo. Com isso, esse modelo é mais popular do que outros no mercado.

As impressoras do tipo FDM são geralmente compostas por três eixos de movimentação, um extrusor responsável por extrusar o filamento de forma controlada, mantendo uma temperatura pré-definida. Além disso, elas possuem uma mesa aquecida responsável por manter o filamento fixo para que os eixos consigam imprimir as peças em camadas. Esse modelo é bastante simples em comparação com outros tipos de impressoras 3D.

2.5.2 Estrutura de uma Impressora FDM

Estrutura Geral: As impressoras FDM são geralmente construídas com uma estrutura robusta e modular. A estrutura é composta por perfis de alumínio ou outros materiais leves, garantindo estabilidade e rigidez necessárias para o processo de impressão. A configuração da estrutura pode variar entre modelos, mas muitas delas seguem um design compacto e fechado para controlar melhor o ambiente de impressão.

Eixos de Movimentação: Uma impressora FDM típica possui três eixos de movimentação: X, Y e Z. O eixo X controla o movimento horizontal da cabeça de impressão de um lado para o outro, o eixo Y controla o movimento da mesa de impressão de frente para trás, e o eixo Z controla o movimento vertical, ajustando a altura da camada de impressão.

Extrusor: O extrusor é uma parte crucial da impressora FDM. Ele é responsável por extrudar o filamento de plástico aquecido em camadas sucessivas durante o processo de impressão. O filamento passa por um conjunto de engrenagens e é empurrado através de um bico de impressão, onde é depositado na plataforma de construção.

Mesa Aquecida: Muitas impressoras FDM possuem uma mesa aquecida. Essa característica é especialmente útil ao imprimir com materiais como o ABS, pois ajuda a evitar problemas como a contração térmica. A mesa aquecida mantém a temperatura da base, proporcionando uma superfície estável para a aderência do material durante o processo de impressão.

Controlador e Eletrônica: A impressora FDM é controlada por uma unidade eletrônica que interpreta os comandos do arquivo de impressão. Geralmente, é utilizado um microcontrolador, como o Arduino, para controlar os movimentos dos motores e outros parâmetros durante a impressão.

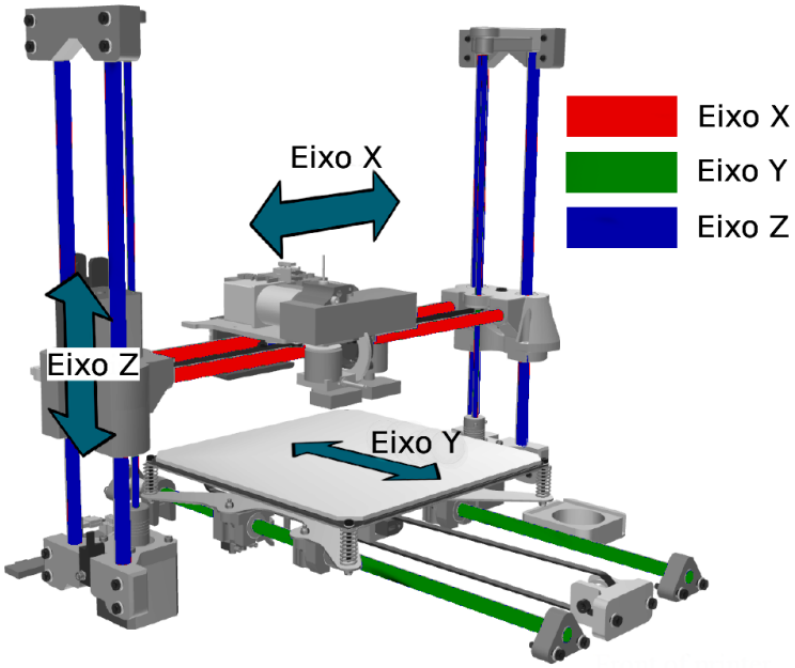
Software de Controle: Os usuários interagem com a impressora FDM por meio de um software de controle. Este software é utilizado para preparar os arquivos de modelo

3D, ajustar configurações de impressão, como temperatura e velocidade, e iniciar o processo de impressão.

Sensores e Componentes Adicionais: Algumas impressoras FDM são equipadas com sensores para monitorar a temperatura, a umidade ou a detecção de filamento, proporcionando um controle mais preciso e evitando possíveis problemas durante a impressão.

Em resumo, a estrutura de uma impressora FDM é projetada para garantir movimentos precisos e controlados, facilitando a construção camada por camada de objetos tridimensionais a partir do filamento termoplástico aquecido. Essa abordagem simples e eficaz tornou as impressoras FDM extremamente populares na impressão 3D. A Figura 14 apresenta um modelo de impressora 3D do tipo FDM.

Figura 14 – Estrutura de uma Impressora 3D FDM.



Fonte: 3DLAB (2018)

2.6 OpenCV

OpenCV é uma biblioteca de visão computacional de código aberto, desenvolvida pela Intel, destacando-se pela eficiência computacional, especialmente em aplicações de tempo real (Barelli, 2018). Suportando interfaces para C++, Python, Java e MATLAB, além de ser compatível com uma variedade de sistemas operacionais, como Windows, Linux, Android e Mac OS, o OpenCV é uma ferramenta versátil e amplamente adotada (OpenCV, 2024).

A biblioteca é especialmente projetada para aplicações em tempo real e otimizada para aproveitar instruções. Com mais de 500 algoritmos e aproximadamente 10 vezes mais funções que os suportam, o OpenCV é nativamente escrito em C++ e possui uma interface bem adaptada para contêineres STL (OpenCV, 2024).

Com uma gama extensa de mais de 2500 algoritmos otimizados, abrangendo desde técnicas clássicas até abordagens de ponta em visão computacional e aprendizado de máquina, o OpenCV oferece funcionalidades diversas. Esses algoritmos são empregados em detecção e reconhecimento facial, identificação de objetos, classificação de ações humanas em vídeos, rastreamento de movimentos de câmeras, rastreamento de objetos em movimento, extração de modelos 3D, produção de nuvens de pontos 3D por meio de câmeras estéreo, entre outras aplicações (OpenCV, 2024).

A comunidade de usuários do OpenCV é robusta, com mais de 47 mil membros, e o número de downloads estimado ultrapassa 18 milhões. A biblioteca é amplamente adotada em empresas, grupos de pesquisa e órgãos governamentais (OpenCV, 2024).

Além de gigantes estabelecidos como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota, muitas startups, como Applied Minds, VideoSurf e Zeitera, fazem uso extensivo do OpenCV. As aplicações práticas da biblioteca são diversificadas, desde a costura de imagens de streetview até a detecção de intrusões em vídeos de vigilância, monitoramento de equipamentos de minas, navegação e manipulação de objetos por robôs, detecção de afogamentos em piscinas, execução de arte interativa, verificação de pistas de detritos e inspeção de etiquetas de produtos (OpenCV, 2024).

2.7 Motores de Corrente Contínua

A flexibilidade e versatilidade são características fundamentais das máquinas de corrente contínua (CC). Antes da ampla adoção dos acionamentos de motores de corrente alternada (CA), as máquinas CC eram praticamente a única opção para inúmeras aplicações que exigiam um alto nível de controle. Embora apresentem vantagens notáveis, como a capacidade de controle preciso, as principais desvantagens das máquinas CC estão na complexidade do enrolamento da armadura e no sistema de comutador e escova (Umans, 2014).

Essa complexidade adicional não apenas aumenta os custos em comparação com as máquinas CA concorrentes, mas também aumenta as necessidades de manutenção, o que pode reduzir a confiabilidade dessas máquinas. No entanto, mesmo diante desses desafios, as vantagens dos motores CC permanecem substanciais. Eles continuam a manter uma posição competitiva robusta, tanto em tamanhos de grande porte para aplicações industriais, quanto em tamanhos menores, utilizados em uma variedade de contextos (Umans, 2014).

A parte interna dos motores de corrente contínua (CC) é composta por elementos fundamentais que conferem a esses motores sua notável flexibilidade e versatilidade. Esses motores incluem uma armadura, composta por um enrolamento de fios condutores, e um sistema de comutador e escova.

A armadura desempenha um papel crucial no funcionamento do motor CC, gerando o campo magnético rotativo necessário. Seu enrolamento permite a condução da corrente elétrica, criando um campo magnético que interage com o campo magnético do ímã fixo no motor.

O sistema de comutador e escova é vital para inverter a direção da corrente elétrica na armadura, garantindo a rotação contínua do motor. As escovas entram em contato com o comutador, invertendo a direção da corrente elétrica em intervalos regulares.

Além disso, é importante destacar que os motores de corrente contínua têm sido peças-chave em sistemas de controle avançados, especialmente em ambientes onde a precisão e a resposta dinâmica são essenciais (Umans, 2014).

2.8 Encoders

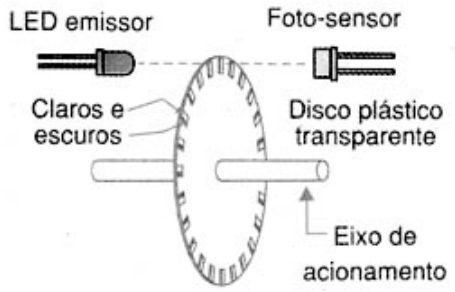
Os encoders desempenham um papel crucial em diversas aplicações que demandam medição precisa de movimento, posicionamento e rotação. Esses dispositivos convertem o deslocamento angular ou linear de um eixo mecânico em um sinal elétrico para determinar a posição ou a velocidade de um objeto em movimento. Existem vários tipos de encoders, sendo os mais conhecidos os encoders ópticos e magnéticos. No entanto, o princípio de funcionamento desses sensores é bastante semelhante.

O encoder óptico utiliza uma fonte de luz, geralmente um LED, e um sensor óptico para criar padrões de luz e sombra em um disco rotativo. Este disco possui regiões opacas e transparentes que, ao passarem pela luz do LED em direção ao sensor, geram pulsos elétricos. A contagem desses pulsos ao longo do tempo permite determinar o movimento angular ou linear com alta resolução. A Figura 15 ilustra um encoder óptico acoplado ao eixo de um atuador para medir sua velocidade.

O princípio de funcionamento de um encoder magnético é similar, mas em vez do par fotoemissor-fotorreceptor, um sensor de efeito Hall é utilizado. O disco plástico ou metálico é substituído por um disco com vários ímãs, geralmente do tipo neodímio. Cada vez que um dos ímãs passa pelo sensor de efeito Hall, um pulso é gerado para o sistema, permitindo também a geração de um sinal retangular. A vantagem desse tipo de encoder é sua menor susceptibilidade a interferências causadas por sujeira, que poderiam bloquear a passagem de luz nos sensores ópticos.

Dentro desses dois grupos de encoders, existem os encoders de quadratura. Os

Figura 15 – Exemplo de um encoder óptico.



Fonte: Braga (2017)

encoders de quadratura geram dois sinais de dados que estão eletricamente defasados em 90° um em relação ao outro, uma relação conhecida como quadratura. Esses sinais, muitas vezes denominados canais, proporcionam informações essenciais sobre o sentido da direção de giro do dispositivo monitorado. Em cada ciclo completo, que contém quatro transições ou bordas, um encoder que gera, por exemplo, 2500 ciclos por revolução fornece 10.000 bordas por revolução (Instruments, 2002).

A eletrônica associada aos encoders de quadratura utiliza a "detecção de borda"para contar e determinar a direção do movimento. As transições provenientes do encoder atuam como gatilho para causar uma contagem. Em cada transição, a eletrônica não apenas gera uma contagem, mas também determina a direção do movimento, contando para cima ou para baixo. Isso é feito considerando se a transição está indo para o alto ou para o baixo, juntamente com o estado do outro sinal, conforme mostrado pela Tabela 1 de detecção de borda. Essa abordagem proporciona uma precisão e capacidade de controle essenciais para diversas aplicações que demandam medição precisa de movimento, posicionamento e rotação (Instruments, 2002).

Tabela 1 – Tabela de Detecção de Borda.

Transição	Estado Atual	Direção do Movimento
Forward Travel	Alto	Contagem para Cima (Up)
Forward Travel	Baixo	Contagem para Baixo (Down)
Reverse Travel	Alto	Contagem para Baixo (Down)
Reverse Travel	Baixo	Contagem para Cima (Up)

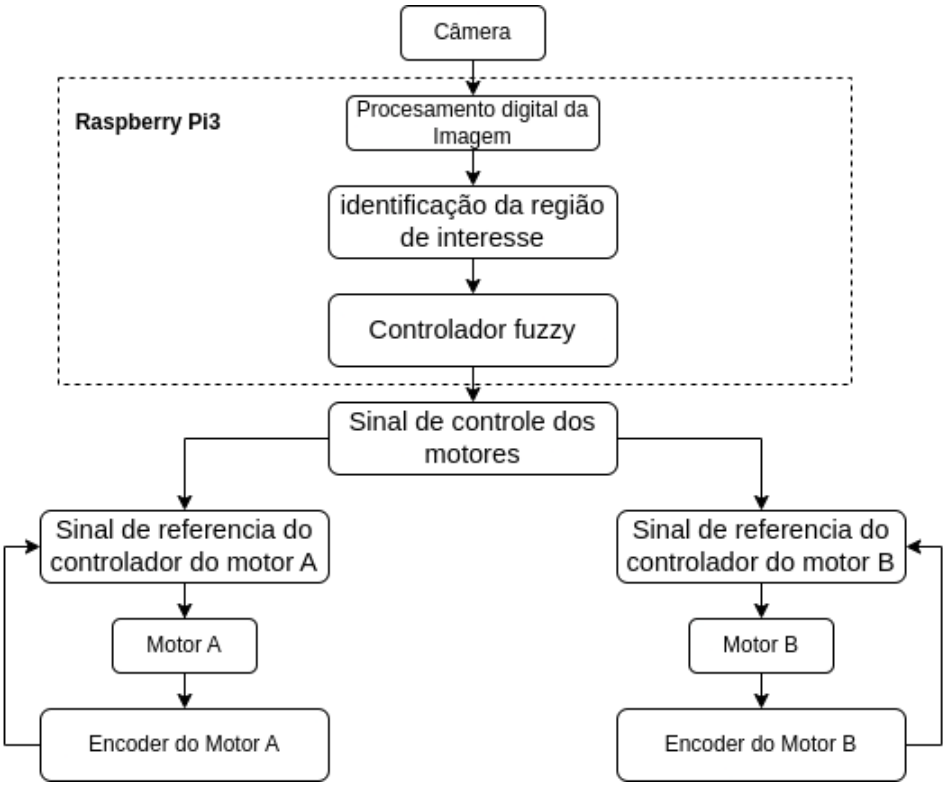
Fonte: Adaptado de Instruments (2002)

3 MATERIAIS E MÉTODOS

3.1 Metodologia

A metodologia empregada neste projeto consiste em realizar capturas de imagem da fita, representando a trajetória do robô. Em seguida, é realizado o processamento dessa imagem utilizando a conversão de cores de RGB para HSV. Posteriormente, definem-se limites superiores e inferiores para eliminar todos os elementos da imagem que não estejam na mesma faixa de valores possíveis da fita, que é a região de interesse. Para eliminar possíveis ruídos ou falhas nessa região, serão utilizadas máscaras para dilatar e extrudar a imagem, empregando morfologia matemática. Após identificar a região de interesse, ocorre a binarização da imagem, onde a parte da trajetória é mantida em nível 0 (branco) e o resto da imagem em nível 255 (preto).

Figura 16 – Fluxograma do desenvolvimento do projeto de pesquisa.



Fonte: Elaborada pelo autor, 2023.

Conforme observado na Figura 16, a imagem é capturada pela câmera e processada pelo Processamento Digital de Imagens (PDI). A imagem segmentada é utilizada como dados de entrada para a parte de visão computacional, responsável por identificar a região

de interesse. Esta parte busca encontrar o centro da fita, representando a trajetória, e compará-lo com uma linha imaginária que representa o centro do robô. A distância entre o centro da fita e do robô é o dado enviado para o controlador *Fuzzy*.

O controlador *Fuzzy* recebe a informação da variação de distância entre o centro do robô e a fita, gerando a variável de desvio de erro ao comparar a distância entre o centro do robô e a posição atual da fita com o valor previamente medido. Esses dois valores funcionam como variáveis linguísticas para o controlador *Fuzzy*. Em seguida, são determinados os valores de saída das variáveis linguísticas para os motores esquerdo e direito. Esses valores são transmitidos para os motores como um sinal de PWM, desempenhando um papel crucial ao manter o robô dentro de uma faixa de valores aceitáveis para seguir a trajetória.

Similar ao controlador *Fuzzy*, aplica-se o controlador Proporcional, Integral e Derivativo (PID), que é responsável por receber o mesmo valor da variação de distância entre o centro do robô e da fita, proveniente da identificação da região de interesse. Neste caso, o valor do PID é aplicado no valor de PWM dos motores, aumentando ou diminuindo a velocidade das rodas para que o robô permaneça sobre a trajetória da mesma forma que o controle *Fuzzy*.

Os encoders dos motores são utilizados para receber o valor dos pulsos do robô e, a partir disso, recriar a trajetória feita por ele, gerando um mapa de movimentação do robô que será utilizado para verificar a eficácia dos controladores aplicados.

O sistema é embarcado em um *Raspberry Pi* 4, responsável por processar todos os dados da parte de visão computacional e do controlador *Fuzzy*. O hardware também inclui um módulo de câmera de 720P e dois motores DC de 6V com encoder.

O robô será exposto a diferentes condições para avaliar a eficiência do controlador *Fuzzy* aplicado. Os motores também serão testados com diferentes controladores para avaliar suas respostas.

3.2 Motor DC com encoder

Os motores selecionados para este projeto são motores DC de 6V, com uma rotação de 2500 RPM (rotações por minuto). Eles podem ser vistos na Figura 17. Eles são alimentados por uma fonte de tensão contínua, neste caso, uma bateria. Cada motor está equipado com um motor de quadratura, um dispositivo sensor que converte movimento ou posição angular em sinais elétricos. Esse motor de quadratura fornece feedback de posição, velocidade, ou ambos, para um sistema de controle.

Existem dois tipos principais de motores de quadratura: o absoluto e o incremental. O motor de quadratura absoluto oferece a posição absoluta do eixo em relação a uma referência, enquanto o motor de quadratura incremental fornece informações sobre o

movimento relativo, geralmente em pulsos. No projeto em questão, foi utilizado o modelo incremental, com o propósito de validar o controle de velocidade dos motores.

O motor de quadratura em uso possui um conector com 6 cabos. Dois desses cabos são destinados ao controle da direção do motor, enquanto os outros dois são dedicados à alimentação e ao GND. Os dois cabos restantes são específicos para o motor de quadratura, proporcionando a comunicação necessária para a obtenção de dados sobre a posição e velocidade do motor. Essa configuração permite uma gestão mais precisa e eficiente do desempenho dos motores no contexto do projeto.

Figura 17 – Foto do Motor com Encoder.

Fonte: Elaborada pelo autor, 2024.

3.3 Ponte H

A ponte H é, fundamentalmente, um driver empregado em motores de corrente contínua, possibilitando ao motor executar rotações tanto no sentido horário quanto no sentido anti-horário. Para além da capacidade de inverter o sentido de rotação do motor, essa estrutura requer uma quantidade mínima de energia do circuito de controle Alves (2018).

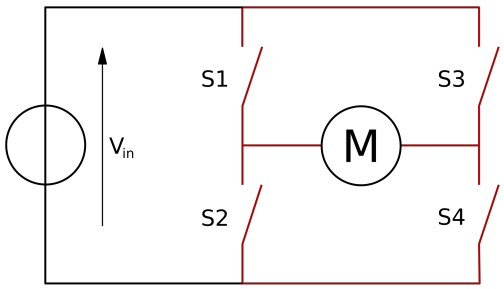
A designação "ponte H"torna-se intuitiva ao examinar o esquema de ligação, no qual a carga está conectada entre as quatro chaves, configurando-se de maneira semelhante à letra H, como indicado na Figura 18, Alves (2018).

Para controlar os motores DC, escolheu-se utilizar a ponte H modelo L298N, que pode ser vista na Figura 19. Esse componente é amplamente empregado em projetos de robótica e automação devido à sua popularidade e facilidade de uso. Operando em uma faixa de tensão de alimentação de 5V a 35V, essa ponte H oferece uma capacidade de

corrente de até 2A por canal em muitos casos. Além disso, possui duas entradas de *enable*, permitindo o controle preciso da velocidade dos motores através de sinais PWM. Sua versatilidade se destaca ao aceitar alimentação de 12V ou 5V, proporcionando flexibilidade para se adequar às necessidades específicas do sistema. Todas essas características fazem da ponte H L298N uma escolha prática e eficiente para diversas aplicações que demandam controle dinâmico e preciso de motores DC.

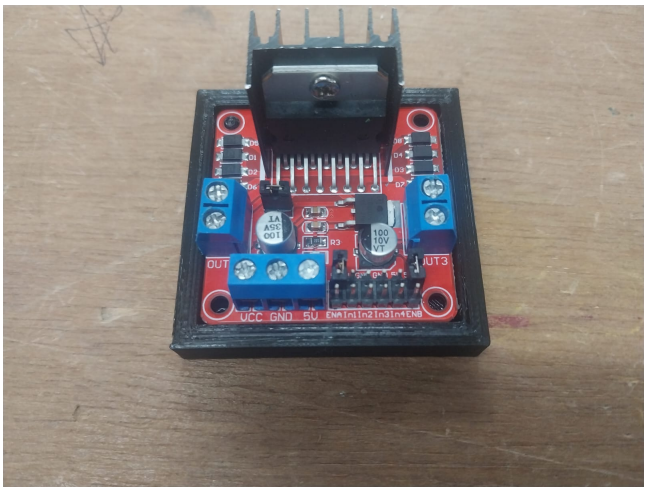
A ponte H em questão também dispõe de entradas lógicas para o controle da direção de rotação dos motores. Em suma, se duas entradas lógicas do motor permanecerem simultaneamente em estado alto ou baixo, os motores não serão acionados. Para controlar a direção desejada, é essencial habilitar um dos pinos de entrada em nível alto e o outro em nível baixo. Essa configuração possibilita que o motor realize a rotação para frente ou para trás.

Figura 18 – Exemplo de esquema elétrico da Ponte H.



Fonte: BUTTAY (2006)

Figura 19 – Ponte H L298N.



Fonte: Elaborada pelo autor, 2024.

3.4 *Raspberry Pi*

O *Raspberry Pi*, caracterizado por sua operação silenciosa e eficiência energética, é uma escolha notável para diversas aplicações. Seu design sem ventoinhas contribui para um ambiente de trabalho tranquilo, enquanto o consumo de energia é significativamente inferior se comparado a outros computadores. Conectividade é uma prioridade no *Raspberry Pi* 4, oferecendo uma rede rápida e confiável. Equipado com Gigabit Ethernet, o dispositivo também suporta conexões sem fio e Bluetooth, proporcionando flexibilidade para diversas configurações de rede. A velocidade de transferência de dados é aprimorada pelo *Raspberry Pi* 4 com a incorporação de portas USB 3, uma atualização notável em relação às duas portas USB 2 já presentes. Com a capacidade de transferir dados até dez vezes mais rápido, as portas USB 3 oferecem maior eficiência em operações de leitura e gravação, Pi (2019).

O *Raspberry Pi* 4 Model B, ilustrado na Figura 20, representa uma notável evolução no universo dos minicomputadores, apresentando avanços significativos em termos de desempenho e funcionalidades. Equipado com um processador Broadcom BCM2711 quad-core Cortex-A72 de 64 bits, operando a uma velocidade de até 1,5 GHz, e possuindo 4 GB de memória RAM, este dispositivo suporta sistemas operacionais Windows e Linux, além de oferecer 40 pinos de entrada e saída de dados.

Este modelo foi escolhido para o presente projeto devido às características necessárias para processar os dados da câmera e realizar os processos de morfologia matemática necessários para detectar o trajeto do seguidor de linha. Além disso, foi utilizado para controlar os motores aplicando um sinal de controle sobre eles. Entretanto, esse sinal não foi enviado diretamente, pois os motores podem gerar uma corrente reversa quando acionados, o que poderia danificar os pinos do *Raspberry* e comprometer seu funcionamento.

O barramento de 40 pinos foi empregado para controlar os pinos de *enable* e *logic input* da ponte H, além de ser utilizado para receber os dados dos encoders dos motores. Vale destacar que esta abordagem visa garantir a integridade do *Raspberry Pi*, protegendo-o contra possíveis danos causados pela corrente reversa dos motores durante o acionamento.

A câmera utilizada foi conectada à porta USB 3.0 do *Raspberry*, proporcionando uma interface eficiente para a captura de dados visuais. Esta configuração visa otimizar a capacidade de processamento do *Raspberry Pi* em conjunto com os demais componentes do projeto.

Figura 20 – *Raspberry Pi* 4 Modelo B.

Fonte: Elaborada pelo autor, 2024.

3.5 Rodas

Foram empregados dois tipos de rodas neste projeto. O primeiro modelo, apresentado na Figura 21, foi produzido por meio de uma impressora 3D, disponibilizado por um colega que gentilmente forneceu as peças para utilização neste projeto. Essas rodas foram impressas com um filamento flexível, proporcionando amortecimento eficaz e uma notável resistência. Os materiais utilizados nas rodas apresentam diferentes cores, mas compartilham as mesmas características de flexibilidade. O design 3D dessas rodas foi baseado no modelo original fornecido com os motores DC de 6V.

Essas rodas possuem um raio de 3.25 cm e comprimento de 20.420 cm, sendo destinadas à parte dianteira do robô, onde desempenham um papel crucial em sua movimentação.

Figura 21 – Foto das Rodas .

Fonte: Elaborada pelo autor, 2024.

O segundo modelo de roda adotado no projeto, apresentado na Figura 22, é a roda castor, que é uma roda de giro livre. Esta configuração foi escolhida para ser a roda traseira do robô. Seu movimento é controlado pelas rodas dianteiras, que estão conectadas aos motores controlados pela ponte H e pelo *Raspberry Pi* 4 Model B.

Figura 22 – Foto da Roda Castor .

Fonte: Elaborada pelo autor, 2024.

Essa escolha estratégica de rodas visa otimizar a mobilidade do robô, permitindo uma direção mais precisa e eficiente. Cada tipo de roda desempenha um papel específico no funcionamento geral do projeto, contribuindo para a dinâmica e desempenho do robô seguidor de linha.

3.6 Filamento

Os filamentos de impressora 3D, apresentados na Figura 23, são componentes essenciais utilizados no processo de impressão tridimensional. Fabricados na forma de fios enrolados em carretéis, esses filamentos desempenham um papel crucial na produção de objetos tridimensionais. Podem ser comparados aos cartuchos de impressoras convencionais, que são usados para imprimir em papel Bittencourt (2021).

O filamento é o material utilizado em impressoras 3D como matéria-prima para as impressões. Esse material pode ser categorizado em vários tipos, tais como PLA, ABS, Flex, PETG, Wood, PVA, entre outros. Todos esses tipos de filamento possuem características distintas que os tornam adequados para diferentes aplicações. Um exemplo disso é o

filamento Flex, extremamente flexível e resistente, porém sujeito a alguns problemas relacionados à retração. A retração ocorre quando o extrusor realiza uma sucção do filamento para evitar que ele se deforme durante o deslocamento do extrusor de um ponto ao outro.

Os filamentos mais comuns são o ABS, PLA e PETG, amplamente populares no mercado de impressoras 3D devido à sua abundância e facilidade de produção. Cada um desses materiais apresenta características específicas, tornando-os ideais para diferentes finalidades. Essa diversidade de filamentos oferece aos usuários uma gama de opções, possibilitando a impressão de uma variedade de objetos com propriedades mecânicas e estéticas distintas.

Figura 23 – Foto dos Filamentos Utilizados.

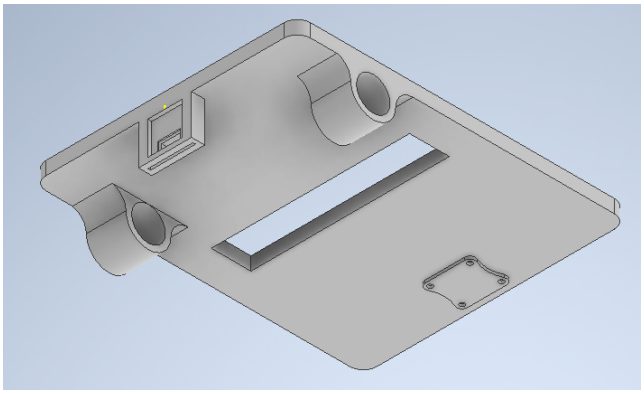
Fonte: Elaborada pelo autor, 2024.

3.7 Modelagem do Chassi

O modelo inicial do carrinho foi concebido para suportar dois motores, uma ponte H, baterias, câmera e o *Raspberry Pi*. Com esse objetivo em mente, foi projetado utilizando o Inventor, um *software* de modelagem 3D disponibilizado pela Autodesk, uma empresa de *software* de design e conteúdo digital. O Inventor foi escolhido por ser gratuito para estudantes e amplamente utilizado na indústria de modelagem 3D. Isso possibilitou a importação de peças específicas do projeto, uma vez que existem muitos repositórios online que disponibilizam isso de forma gratuita.

A ideia inicial era que o chassi do carrinho fosse de fácil impressão, para economizar filamento de impressora 3D e ter um tempo de impressão reduzido. Por isso, optou-se por criar uma chapa lisa na parte superior, com espaços para encaixar a fonte, a ponte H e o conector do *Raspberry Pi*. Na parte inferior do chassi, foram localizados os encaixes dos motores para mantê-los fixos e um suporte para a câmera. Isso foi pensado levando em consideração que quanto mais próxima a câmera estivesse do solo, mais fácil seria para ela detectar e rastrear a trilha, além de facilitar a impressão do chassi, .

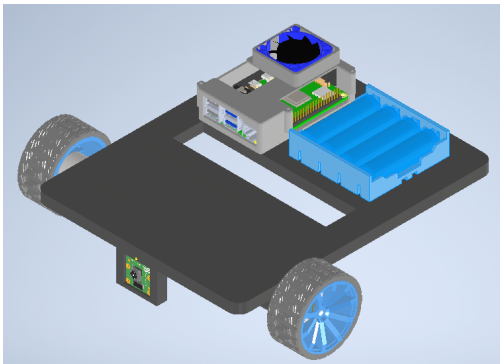
Figura 24 – Foto do Primeiro Chassi.



Fonte: Elaborada pelo autor, 2024.

O chassi apresenta um rasgo no meio para permitir a passagem de cabos do motor e da câmera para o *Raspberry Pi* e para a ponte H dos motores. O modelo 3D foi apresentado na Figura 24. Após a conclusão da modelagem do chassi, foi utilizado a ferramenta de montagem do Inventor. Essa é uma opção que o *software* oferece, permitindo a seleção de peças modeladas e a união delas em um único arquivo. Isso possibilita testar efeitos físicos, como pressão, encaixes, entre outros, isso pode ser visto na Figura 25.

Figura 25 – Foto da Montagem no Inventor.



Fonte: Elaborada pelo autor, 2024.

Na montagem, o chassi e as outras peças que fazem parte do carrinho foram importados. Essas outras peças foram baixadas do site GrabCAD, que é um repositório de peças modeladas que permite o download gratuito, além de disponibilizar as peças em diferentes formatos para serem utilizadas em diversos *softwares*. A montagem foi concluída, resultando no modelo final do carrinho. Algumas alterações foram realizadas, incluindo a redução do seu comprimento. Inicialmente, o comprimento era de 30 cm, e esse valor foi ajustado para 20 cm. Além disso, uma pequena modificação no diâmetro do encaixe dos motores foi feita, pois estava menor do que o necessário para acomodar os motores.

Após a simulação de montagem, uma modificação adicional no chassi foi feita para permitir a fixação das braçadeiras nos motores, garantindo assim mais estabilidade para eles. Isso foi alcançado inserindo cortes nas proximidades dos encaixes dos motores. Essas adaptações contribuíram para aprimorar o desempenho e a robustez do carrinho. Com a última modificação aplicada, procedeu-se então à exportação da peça no formato STL. Este é o formato que as impressoras 3D entendem, representando a peça por meio de pequenos triângulos. O formato STL é amplamente utilizado na impressão 3D, pois fornece uma representação tridimensional da geometria da peça.

3.8 Fatiamento no PrusaSlicer

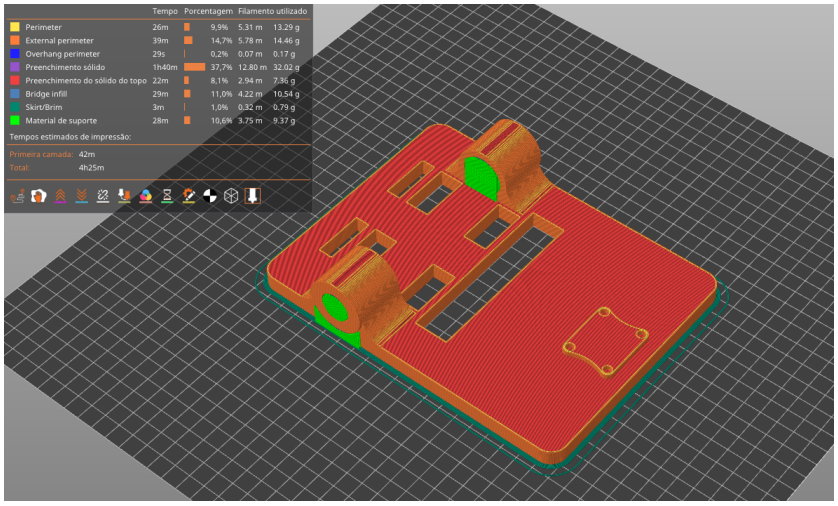
O PrusaSlicer é um *software* de fatiamento gratuito e de código aberto desenvolvido pela Prusa Research, uma empresa de impressão 3D sediada na República Tcheca. Originado a partir do *software* Slic3r, tornou-se Slic3r PE (Prusa Edition) e posteriormente foi comercializado como um fatiador interno exclusivo. Com o tempo, a Prusa adicionou recursos e modificações, resultando no PrusaSlicer, lançado como um *software* completamente novo. Apesar de sua associação com as impressoras Prusa, o PrusaSlicer é compatível com todas as impressoras 3D. Desde seu lançamento em 2019, tornou-se um dos *softwares* de fatiamento mais populares e crescentes. Sendo apoiado pela Prusa, os usuários podem contar com atualizações regulares, depuração e melhorias com novos recursos úteis Hailey (2023).

Durante o processo de fatiamento, o modelo STL da peça foi importado para o PrusaSlicer, um *software* gratuito que oferece uma ampla gama de opções de personalização para facilitar a impressão 3D. O PrusaSlicer permite configurar parâmetros essenciais, como camadas de impressão, densidade de preenchimento, suportes e outros detalhes, garantindo controle preciso sobre o processo de impressão. Essas etapas são vitais para assegurar uma impressão 3D bem-sucedida e de alta qualidade.

No PrusaSlicer, realizou-se a calibração do chassi destinado à impressora Sethi3D S4X. Optou-se pelo uso de PLA preto para essa impressão específica. O PLA é um plástico amplamente empregado em impressoras 3D, notadamente nas impressoras abertas,

conhecido por sua facilidade de impressão e estabilidade dimensional, sendo menos suscetível a variações de temperatura. A escolha do PLA contribui para uma impressão mais estável e confiável do chassi do carrinho. Os valores dos parâmetros de impressão utilizados no *software* estão detalhados na Tabela 2, e a Figura 26 apresenta a visualização do chassi no PrusaSlicer.

Figura 26 – Chassi no PrusaSlicer.



Fonte: Elaborada pelo autor, 2024.

Tabela 2 – Parâmetros de Impressão

Parâmetro	**Configuração**
Altura de Camada	0.3 mm
Velocidade de Impressão	60 mm/s
Preenchimento	20%
Suporte	55°
Temperatura do Bico	210 °C
Temperatura da Mesa	60 °C

3.9 Impressão 3D

A impressora utilizada, como mencionado anteriormente, foi o modelo Sethi3D S4X, que oferece um volume interno de 64.000 cm^3 ou 64 L. Com uma velocidade de impressão de 150 mm/s, velocidade de deslocamento de 300 mm/s, temperatura máxima do bico de 275ºC, e a capacidade de imprimir materiais como ABS, PLA, PETG e Flex, essa impressora possui uma resolução de camada ajustável entre 50 mícrons (0.05 mm) até 400 mícrons (0.4 mm).

A Sethi3D S4X é uma impressora do tipo sdm, utilizando os eixos X, Y e Z para movimentar o extrusor e realizar as trajetórias necessárias para imprimir uma peça. Além

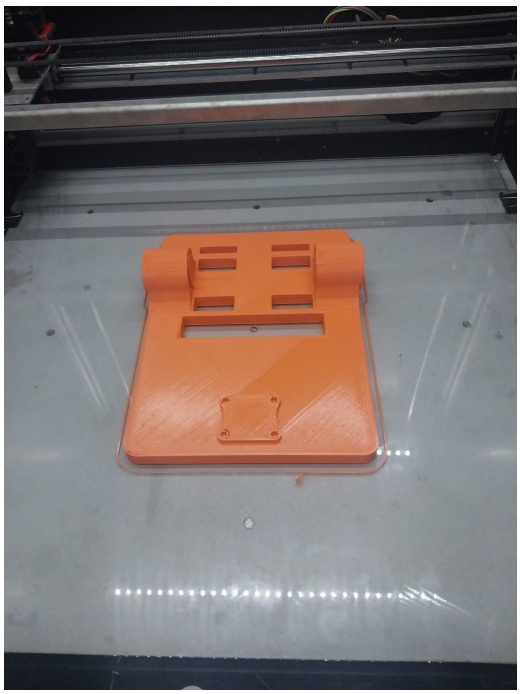
disso, ela é uma impressora do tipo direct drive, apresentando um motor de passo que alimenta o extrusor com filamento. Este motor está localizado acima do eixo do extrusor, justificando o nome "direct drive".

O extrusor da impressora é composto por diversas partes, incluindo uma guia para o filamento, um dissipador de calor, um cooler para resfriar o bico, um tubo de teflon, a garganta do extrusor, um bloco de aquecimento com um sensor de temperatura e uma resistência para aquecer o bloco. O bico da impressora, responsável por definir a resolução da peça, determina valores como a altura da camada, onde diâmetros menores resultam em maior resolução, mas também aumentam o tempo de impressão significativamente.

A impressora utiliza o G-Code, uma linguagem de programação comumente utilizada em máquinas CNC para controlar trajetórias e operações da ferramenta de corte. Cada linha de código G contém uma instrução que direciona a máquina sobre o que fazer, abrangendo desde movimentos lineares até operações específicas como extrusão de filamento, limpeza do extrusor e aquecimento da peça.

O arquivo G-code gerado pelo fatiador PrusaSlicer foi enviado para a impressora 3D, resultando na impressão do primeiro chassi do robô. Conforme mencionado anteriormente na modelagem do carrinho, ajustes foram realizados no chassi, gerando um novo STL que foi fatiado novamente no PrusaSlicer e resultou na impressão de um novo chassi, visto na Figura 27.

Figura 27 – Foto da Impressão do Chassi.



Fonte: Elaborada pelo autor, 2024.

3.10 Venv

O módulo venv proporciona suporte para a criação de "ambientes virtuais", que são espaços isolados e leves, cada um com seu próprio conjunto independente de pacotes Python instalados em diretórios específicos. Esses ambientes virtuais são criados em cima de uma instalação existente do Python, conhecida como o Python "base"do ambiente virtual, e podem ser configurados para serem isolados dos pacotes presentes no ambiente base. Isso garante que apenas os pacotes explicitamente instalados no ambiente virtual estejam disponíveis para utilização Python (2019).

Venv é uma abreviação para "Ambiente Virtual"e refere-se a uma funcionalidade no ecossistema Python que permite a criação de ambientes virtuais isolados para projetos específicos. Esses ambientes virtuais ajudam a gerenciar independentemente as dependências do Python, evitando conflitos entre diferentes projetos que possam depender de versões distintas das mesmas bibliotecas.

A utilização do venv tornou-se necessária devido à limitação do Python no *Raspberry Pi*, que impedia que alguns pacotes essenciais para o OpenCV e outras bibliotecas fizessem alterações no sistema. Portanto, foi criado um ambiente virtual no qual foram instaladas as bibliotecas necessárias para este projeto. Essa abordagem assegura um ambiente isolado e funcional, permitindo a execução eficiente do projeto sem interferir no sistema global do *Raspberry Pi*.

3.11 VNC

O VNC (Virtual Network Computing) é um protocolo de internet que permite a visualização de interfaces gráficas remotas através de uma conexão segura. Trocando em miúdos, você pode ver e acessar todo o conteúdo de outro computador remotamente, através da internet. Geralmente é bastante utilizado por profissionais que prestam assistência técnica a outros usuários, já que esse protocolo permite a interação completa com o computador conectado (conhecido com VNC Server) pelo computador cliente (o VNC Viewer). Os dois computadores não precisam ter o mesmo sistema operacional instalado, sendo perfeitamente possível acessar um PC remoto que esteja rodando o Windows 7 através do Linux e vice-versa. A conexão estabelecida entre as duas máquinas é altamente protegida, fazendo uso tanto de encriptação de dados quanto de logon seguro, o que faz essa modalidade de conexão bastante famosa e largamente utilizada tanto na indústria quanto no meio acadêmico Cipoli (2012).

O programa VNC (Virtual Network Computing) foi empregado para possibilitar a transmissão da imagem do *Raspberry Pi* para outro computador remotamente. Essa ferramenta opera através de um sistema de computação remota, composto por um servidor VNC instalado no *Raspberry Pi* e um cliente VNC instalado no computador receptor. O

servidor VNC, no *Raspberry Pi*, captura o conteúdo da tela, transmitindo essas informações pela rede. Por outro lado, o cliente VNC, instalado no computador remoto, recebe essas informações e reproduz o ambiente gráfico do *Raspberry Pi* em sua própria interface.

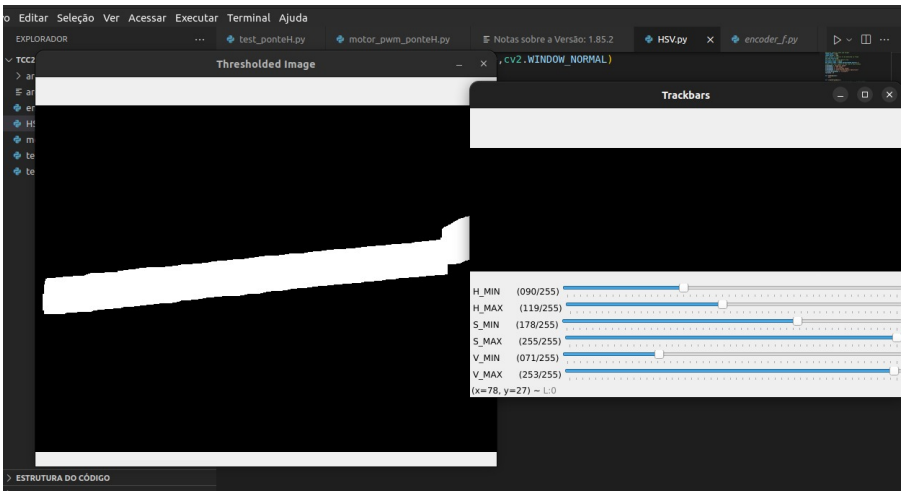
Essa abordagem facilitou a verificação da detecção da trajetória, além de simplificar a execução do programa a partir do outro computador. Através do VNC, tornou-se mais fácil identificar falhas e efetuar ajustes no código, eliminando a necessidade de conectar o *Raspberry Pi* a um monitor para verificar a detecção da trajetória, uma vez que todo o controle e visualização puderam ser realizados remotamente.

3.12 Detecção da Faixa na Trajetória

Para a detecção da faixa na trajetória, foi inicialmente implementada uma análise de cor. Essa abordagem envolveu o desenvolvimento de um código em Python, utilizando a biblioteca OpenCV. O código permitiu a manipulação dos valores de H (matiz), S (saturação) e V (valor ou luminosidade) tanto no espectro máximo quanto no mínimo, dentro da imagem capturada pela câmera.

A manipulação desses parâmetros possibilitou restringir a faixa de valores correspondentes à cor da fita na trajetória, isso pode ser visto na Figura 28. Após a definição dos valores máximo e mínimo para H, S e V, eles foram incorporados ao código principal do OpenCV para serem utilizados no acompanhamento contínuo da trajetória. Essa estratégia de análise de cor, implementada através do código Python e integrada ao OpenCV, revelou-se eficaz na detecção da faixa desejada na trajetória do robô. Essa detecção é crucial para o sistema autônomo, pois permite que o robô siga a trajetória previamente estabelecida de maneira precisa e robusta.

Figura 28 – Detecção da Faixa.



Fonte: Elaborada pelo autor, 2024.

3.13 Programação do OpenCV

Inicialmente, a imagem da câmera foi capturada e, posteriormente, submetida a uma conversão de RGB para HSV. Essa etapa foi crucial, uma vez que as imagens capturadas pela câmera eram do tipo RGB, e o modelo HSV destaca-se por lidar melhor com variações de luz, um dos desafios deste projeto.

Após a conversão da imagem, tornou-se necessário definir os valores máximo e mínimo da cor da fita em HSV. Esses valores são fundamentais para excluir cores que estejam fora da faixa desejada no espaço HSV. Com isso, as cores indesejadas na imagem capturada pela câmera são representadas em preto, enquanto as cores de interesse são destacadas em branco, caracterizando, essencialmente, uma binarização da imagem. Os valores de máximo e mínimo foram determinados através da Detecção da Faixa na Trajetória.

Com essa fase concluída, o caminho foi identificado, e começaram os testes para determinar o melhor método para seguir a linha. Duas abordagens foram consideradas: a primeira consistia em manter o robô sobre a linha e seguir o seu centro; a segunda permitia que o robô se movimentasse em direção à linha sem necessariamente estar exatamente sobre ela.

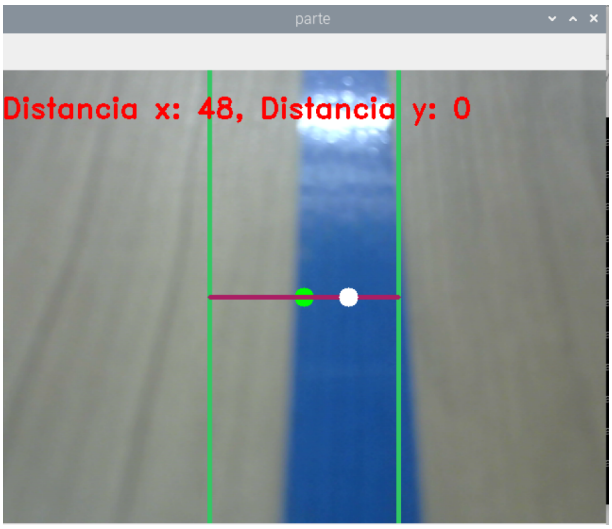
Inicialmente, o código foi desenvolvido para seguir o centro da linha, utilizando uma linha reta horizontal para analisar a quantidade de pixels nela e definir o centro de sua localização. Adicionalmente, foi criada uma linha vertical que dividia o centro do robô. Se mais pixels estivessem à esquerda, um valor entre -1 e 0 seria atribuído, dependendo do centro de concentração desses pixels. Caso houvesse mais pixels à direita, o valor seria entre 0 e 1, dependendo também do centro de concentração dos pixels. Se os pixels tivessem um centro de concentração no meio do robô, o valor da variável seria 0.

O valor retornado com base no centro horizontal dos pixels é enviado para o controlador PID, que recebe esse valor e, a partir disso, tenta movimentar o robô mais para a esquerda quando o valor de retorno está entre -1 e 0, ou mais para a direita se estiver entre 0 e 1. Esse controle dos motores é feito utilizando o PWM do *Raspberry Pi*, cujo valor varia de 0 a 100. Com isso, os motores aceleram ou desaceleram dependendo do valor de PWM recebido por cada motor na ponte H. Este modelo foi testado, mas apresentou muitas falhas durante os testes, sendo necessário utilizar outro método. A Figura 29(a) apresenta esse modelo que foi projetado, onde os valores de distância x e y representam as distâncias vertical e horizontal para o centro do robô.

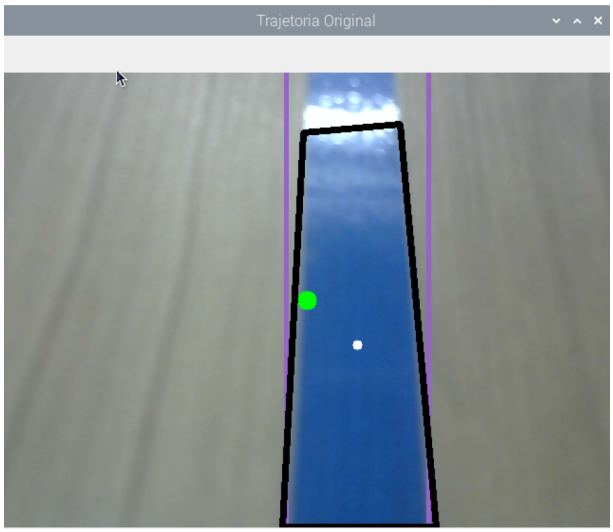
O novo método desenvolvido consiste em utilizar o valor da coordenada **X** do centro do carrinho e compará-lo com o valor da coordenada **X** do centro da trajetória. Isso é feito utilizando a detecção do centro da trajetória rastreada. A diferença entre o centro do robô e da trajetória é enviada para o controlador PID, que controla os motores esquerdo e

direito. Este modelo mostrou-se muito mais efetivo e eficiente para seguir a trajetória. A Figura 29(b) apresentam os sistemas utilizados para capturar os dados para o controlador. Os sistemas de captura foram testados tanto para um controlador PID quanto para um controlador *Fuzzy*.

Figura 29 – Sistemas utilizados para captura.



(a) Primeiro modelo de captura projetado.



(b) Modelo final de captura.

Fonte: Elaborada pelo autor, 2024.

3.14 Alimentação do Sistema

A alimentação adotada neste projeto foi cuidadosamente planejada para garantir desempenho contínuo e testes prolongados. Utilizou-se uma bateria de 7.4 V, que pode ser visto na Figura 30, para alimentar a ponte H, enquanto o *Raspberry Pi* 4 foi alimentado por uma fonte de energia proveniente de um *Power Bank*, que pode ser visto na Figura 31.

Essa escolha estratégica teve como objetivo realizar testes ao longo de um extenso período de tempo, minimizando a necessidade de recargas frequentes da bateria do projeto durante as fases de teste. A integração do *Power Bank* permitiu a execução de testes prolongados sem a necessidade de recarregá-lo com frequência, proporcionando uma eficiência notável durante as fases críticas de teste do projeto.

Figura 30 – Bateria dos Motores

Fonte: Elaborada pelo autor, 2024.

Figura 31 – *Power Bank* para o *Raspberry Pi*.

Fonte: Elaborada pelo autor, 2024.

3.15 Câmera

A câmera do *Raspberry Pi* foi substituída por uma câmera Logitech C270, ela pode ser vista na Figura 32. A escolha desse modelo foi motivada pelo seu baixo custo em comparação com a maioria das câmeras disponíveis. Esta câmera possui uma resolução de 720p e uma taxa de atualização de 30 FPS, sendo conectada via USB.

A substituição foi realizada devido a problemas de compatibilidade entre a câmera original e o *Raspberry Pi*. A câmera Logitech demonstrou uma ampla compatibilidade e, além disso, oferece uma qualidade superior em comparação com a câmera original do *Raspberry Pi*.

Figura 32 – Câmera Logitech C27 .

Fonte: Elaborada pelo autor, 2024.

3.16 Controlador PID da Trajetória

Para seguir a trajetória do robô, utilizando como dados de entrada a diferença entre o centro do robô e o centro dos pixels da trajetória, foi aplicado um controlador PID. Este controlador é responsável por receber esse valor de diferença e, com isso, acelerar ou desacelerar os motores do robô. Essa implementação foi realizada no código Python do *Raspberry Pi*, utilizando a biblioteca "simple-pid". Esta biblioteca permite configurar os valores das constantes K_p, K_i e K_d, além de solicitar o valor do Setpoint na sua definição.

Após definir o controlador, foi necessário passar o valor da diferença entre o centro do robô e da trajetória para o mesmo. Com isso, o valor de saída do PID foi calculado e implementado nos motores esquerdo e direito do robô. Os valores de K_p, K_i e K_d foram definidos através de testes durante o desenvolvimento deste projeto.

3.17 Controlador *Fuzzy* da Trajetória

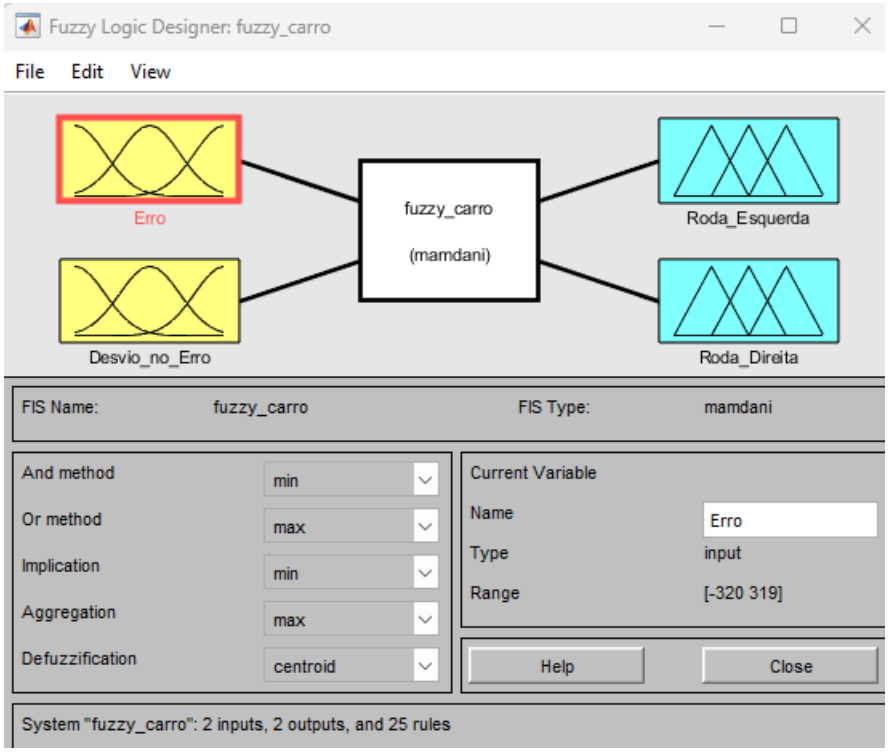
O controlador *Fuzzy* foi inicialmente escolhido como a principal medida de controle para seguir a trajetória do robô. Durante o desenvolvimento deste projeto, constatou-se que o sistema apresentava um valor de entrada não linear e variante no tempo, tornando mais desafiador o controle utilizando técnicas mais convencionais, como o controlador PID. Por esse motivo, o controlador *Fuzzy* foi escolhido para este sistema.

Primeiramente, para a construção desse controle, foi necessário avaliar quais seriam as variáveis de entrada e saída do sistema para, então, definir as regras do controle *Fuzzy*. Foi estabelecido como variável de entrada o valor de erro entre o centro do carrinho e o centro da trajetória. Diferentemente do PID, foi adicionada uma segunda variável de entrada que representa a diferença entre os erros. Essa variável é essencial para verificar se ocorreu alguma mudança brusca na trajetória do robô no momento em que ele tenta

se alinhar com o centro da trajetória. Essa variável é extremamente útil, pois realiza um ajuste fino no controle do robô, semelhante ao parâmetro K_d do controle PID.

Inicialmente, o modelo *Fuzzy* foi desenvolvido no Matlab e posteriormente refeito em Python. O sistema também apresenta duas variáveis de saída que correspondem aos motores Esquerdo e Direito do robô, como ilustrado na Figura 33, onde o modelo do tipo Mamdani foi projetado.

Figura 33 – Model *Fuzzy* desenvolvido no Matlab.



Fonte: Elaborada pelo autor, 2024.

A câmera utilizada neste projeto possui uma resolução de 640 pixels por 480. Devido a isso, o centro da trajetória do carrinho estava localizado em 320 pixels na coordenada **X**. Com esse valor, o centro do carrinho foi definido, e então o erro entre o centro do carrinho e o centro da trajetória foi calculado, considerando que o centro da trajetória pode variar entre 1 e 640 pixels. Portanto, o valor do erro pode variar entre 319 e -320 pixels, como observado na Equação 3.1.

$$E_p = C_r - C_t \tag{3.1}$$

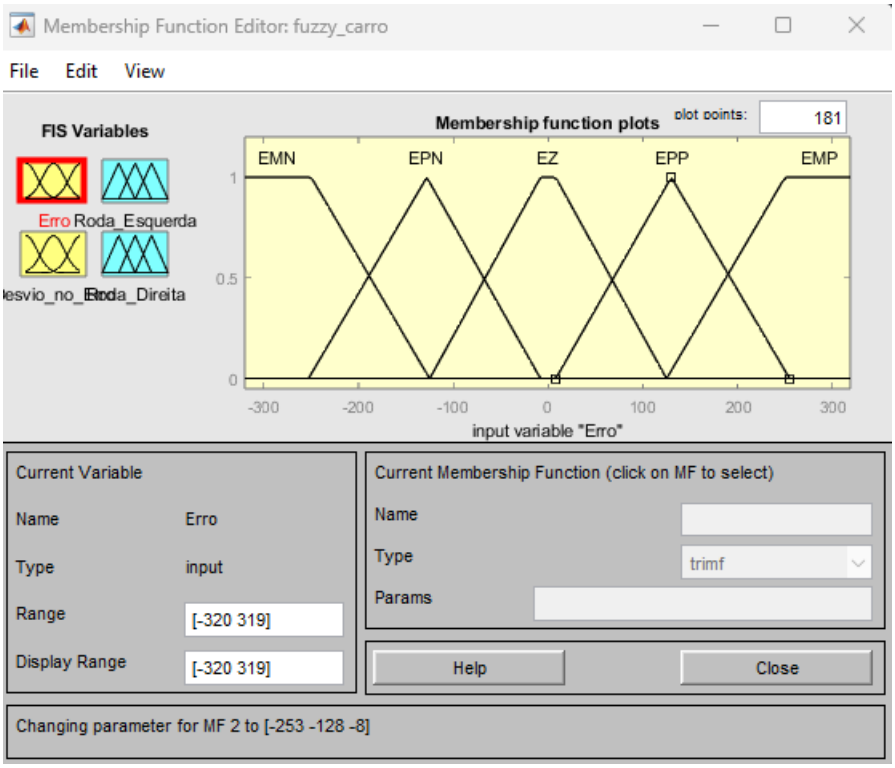
Onde C_r representa o valor do centro do robô em pixels, C_t é o valor do centro da trajetória em pixels, e E_p é o valor do erro calculado. Como mencionado anteriormente, o

valor do centro pode ser substituído por 320, resultando na Equação 3.2:

$$E_p = 320 - C_t \tag{3.2}$$

O valor de E_p será negativo quando o centro da trajetória estiver mais à direita do centro do robô, com um valor máximo negativo de -320 pixels, e será positivo quando estiver mais à esquerda do centro do robô, com um valor máximo de 319 pixels. Após definir os possíveis valores de E_p, foi necessário criar sua variável linguística, como observado na Figura 34.

Figura 34 – Variável Linguística do Erro em Pixels.



Fonte: Elaborada pelo autor, 2024.

A variável linguística "Erro"possui os termos EMN, EPN, EZ, EPP e EMP. Os valores EMN e EPN representam erro muito negativo e erro pouco negativo, respectivamente. Os termos EPP e EMP representam erro pouco positivo e erro muito positivo, respectivamente, enquanto o termo EZ representa erro zero. Os termos EMN, EMP e EZ são do tipo trapezoidal, enquanto EPN e EPP são do tipo triangular.

O valor de erro é considerado zero para um pequeno intervalo que pode estar entre -8 e 8, pois o sistema não consegue ficar perfeitamente sobre a linha da trajetória. Por isso, foi escolhido um trapezoide para representar o erro igual a zero. Esta variável linguística

tem um intervalo de -320 até 319, como já foi discutido anteriormente. Os intervalos dos termos escolhidos são:

- EMN (Erro Muito Negativo): $[-320, -320, -250, -120]$
- EPN (Erro Pouco Negativo): $[-255, -125, -8]$
- EZ (Erro Zero): $[-120, -8, 8, 120]$
- EPP (Erro Pouco Positivo): $[8, 125, 255]$
- EMP (Erro Muito Positivo): $[120, 250, 319, 319]$

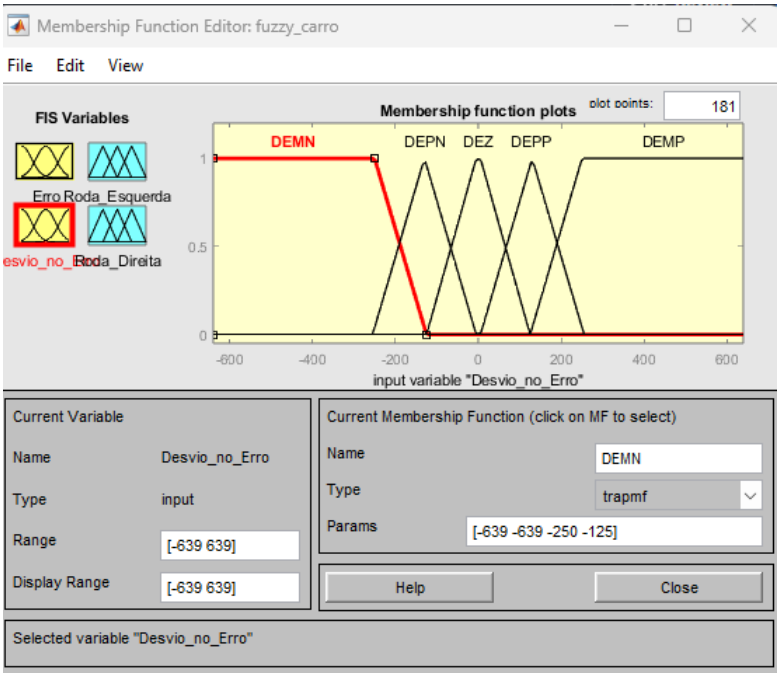
O desvio do erro é a diferença entre o erro atual e o erro anterior. Esse valor representa como o robô está se comportando em relação à trajetória seguida. Quando esse valor se torna muito grande, significa que o robô está se afastando ou se aproximando muito rapidamente da trajetória, dependendo do valor do erro. O desvio do erro pode ser calculado pela Equação 3.3:

$$E_d = E_p - E_{pa} \tag{3.3}$$

Onde E_d é o valor do desvio do erro, E_p é o valor do erro atual em pixels, e E_{pa} é o valor de erro anterior do sistema. Considerando que o robô está com um valor de E_p positivo (o centro da trajetória está à esquerda do centro do robô) e E_d positivo, isso indica que a trajetória está mais à esquerda do robô e está se afastando do robô pela esquerda. Se esse valor fosse negativo, significaria que a trajetória estaria se aproximando do robô pela esquerda. No caso de E_p ser negativo (o centro da trajetória está à direita do centro do robô) e E_d positivo, isso indicaria que a trajetória está mais à direita do robô e está se aproximando do robô pela direita. Se esse valor fosse negativo, significaria que a trajetória estaria se afastando do robô pela direita.

Considerando a Equação 3.3, os valores máximos e mínimos de E_d são -639 e 639. Esses valores de E_d são quase impossíveis de ocorrer, mas para criar a variável linguística do desvio do erro, esses valores foram utilizados para o range do sistema. A variável linguística "Desvio do Erro"possui os termos DEMN, DEPN, DEZ, DEPP e DEMP. Os valores DEMN e DEPN representam desvio de erro muito negativo e desvio de erro pouco negativo, respectivamente. Os termos DEPP e DEMP representam desvio de erro pouco positivo e desvio de erro muito positivo, respectivamente, enquanto o termo DEZ representa desvio de erro zero. Os termos DEMN, DEMP e DEZ são do tipo trapezoidal, enquanto DEPN e DEPP são do tipo triangular. A Figura 35 representa a variável linguística do Desvio do Erro.

Figura 35 – Variável Linguística do Desvio do Erro em Pixels.



Fonte: Elaborada pelo autor, 2024.

Assim como foi feito com o erro, foi considerado um pequeno valor de intervalo de -5 a 5 no desvio para considerar como desvio igual a zero. Por isso, foi escolhido um trapezoide para representar o desvio do erro igual a zero. Como foi apresentado anteriormente, o range desse sistema é de -639 até 639. Os intervalos dos termos escolhidos são:

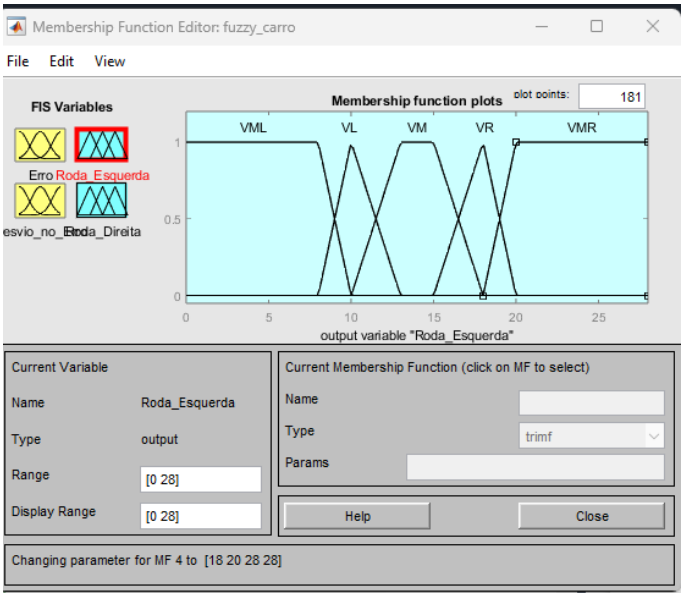
- DEMN (Desvio Erro Muito Negativo): $[-639, -639, -250, -125]$
- DEPN (Desvio Erro Pouco Negativo): $[-255, -130, -5]$
- DEZ (Desvio Erro Zero): $[-125, -5, 5, 125]$
- DEPP (Desvio Erro Pouco Positivo): $[5, 130, 255]$
- DEMP (Desvio Erro Muito Positivo): $[125, 250, 639, 639]$

O sistema *Fuzzy* apresenta duas variáveis linguísticas de saída: Roda Esquerda e Roda Direita. Estas variáveis possuem intervalos de 0 a 25.5 para a Roda Esquerda e de 0 a 24.5 para a Roda Direita, correspondendo aos valores de PWM nos quais os motores podem operar. Ambas as variáveis linguísticas compartilham as mesmas funções de pertinência.

Os termos utilizados pelas variáveis linguísticas de saída dos motores são VML, VL, VM, VR e VMR. Os valores VML e VL denotam velocidade muito lenta e velocidade

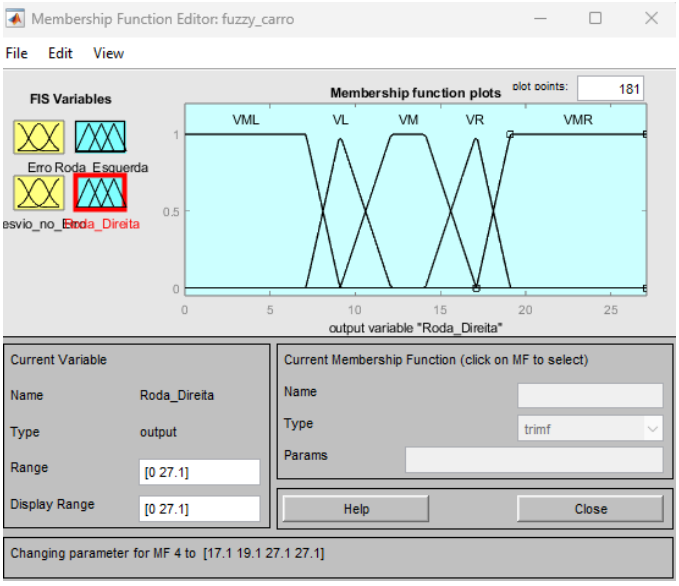
lenta, respectivamente. Por sua vez, os termos VR e VMR representam velocidade rápida e velocidade muito rápida, enquanto VM indica velocidade média. VML, VMR e VM possuem características trapezoidais, enquanto VL e VR são triangulares. A Figura 36 ilustra a representação da Roda Esquerda, enquanto a Figura 37 representa a Roda Direita.

Figura 36 – Variável Linguística da Roda Esquerda.



Fonte: Elaborada pelo autor, 2024.

Figura 37 – Variável Linguística da Roda Direita.



Fonte: Elaborada pelo autor, 2024.

Assim como nas outras variáveis linguísticas apresentadas, incorporou-se uma função de pertinência trapezoidal para a velocidade média, assegurando maior estabilidade do sistema quando este se encontra em uma região mais consistente, onde o erro entre o centro do robô e a trajetória é nulo. Isso permite que o robô mantenha sua velocidade média. Os valores de saída das duas variáveis linguísticas são transmitidos para os motores como PWM, conforme mencionado anteriormente. Os valores de velocidade lenta e rápida são obtidos por meio de testes do PWM dos motores. Os valores de PWM utilizados nas funções de pertinência das rodas esquerda e direita são, respectivamente:

Para a Roda Esquerda

- VML (Velocidade Muito Lenta): $[0, 0, 8.5, 10.5]$
- VL (Velocidade Lenta): $[8.5, 10.5, 13.6]$
- VM (Velocidade Média): $[10.5, 13.6, 15.6, 18.5]$
- VR (Velocidade Rápida): $[15.6, 18.5, 20.5]$
- VMR (Velocidade Muito Rápida): $[18.5, 20.5, 25.5, 25.5]$

Para a Roda Direita

- VML (Velocidade Muito Lenta): $[0, 0, 7.6, 9.6]$
- VL (Velocidade Lenta): $[7.6, 9.6, 12.6]$
- VM (Velocidade Média): $[9.6, 12.8, 14.8, 17.6]$
- VR (Velocidade Rápida): $[14.8, 17.6, 19.6]$
- VMR (Velocidade Muito Rápida): $[17.6, 19.6, 24.6, 24.6]$

Ao definir as variáveis linguísticas e seus termos correspondentes, foi desenvolvido o conjunto de regras para o controlador *Fuzzy*. Com cinco termos na variável linguística de Erro e cinco termos na variável linguística de Desvio do Erro, resultou em vinte e cinco regras para o sistema *Fuzzy*. É crucial ressaltar que, quando as velocidades nas rodas são idênticas, o veículo mantém uma trajetória linear. No entanto, para realizar curvas, é necessário desacelerar uma roda e acelerar a outra, conforme especificado nas regras, levando em consideração que o centro da trajetória seja muito positivo ou muito negativo. As regras mencionadas estão ilustradas na Figura 38.

Figura 38 – Regras do *Fuzzy*

Fonte: Elaborada pelo autor, 2024.

3.18 Cálculos para Odometria

Utilizando os conceitos de Cinemática da trajetória de robôs do capítulo 2, é possível capturar a trajetória realizada pelo robô e plotá-la em um gráfico de x e y. Para fazer isso, foi necessário considerar algumas características do robô deste projeto, como o valor de RPS que foi mensurado, o raio e comprimento das rodas e a distância entre elas. Com esses valores, é possível gerar a trajetória realizada pelo robô, utilizando como parâmetro de entrada os valores dos pulsos dos encoders.

Primeiramente, é necessário verificar a variação entre os pulsos dos encoders no momento atual com o momento anterior, conforme representado na Equação 3.4 e 3.5. Posteriormente, é necessário calcular a distância percorrida pelas rodas, como ilustrado nas equações Equação 3.6 e 3.7.

$$\delta_{Encoder_E} = Encoder_E - lastEncoder_E \tag{3.4}$$

$$\delta_{Encoder_D} = Encoder_D - lastEncoder_D \tag{3.5}$$

$$Distancia_E = \frac{2.\pi.R.\delta_{Encoder_E}}{PPR} \tag{3.6}$$

$$Distancia_D = \frac{2.\pi.R.\delta_{Encoder_D}}{PPR} \tag{3.7}$$

As variáveis $Encoder_E$ e $Encoder_D$ representam os números dos pulsos capturados pelos encoders esquerdo e direito, enquanto $lastEncoder_E$ e $lastEncoder_D$ representam os valores anteriores capturados pelos encoders esquerdo e direito, resultando em $\delta_{Encoder_E}$ e $\delta_{Encoder_D}$, que representam a diferença entre os pulsos do encoder atual e anterior.

A variável R representa o raio das rodas do robô, e PPR representa o número de pulsos por rotação necessários para as rodas realizarem uma rotação. Com isso, é possível calcular $Distancia_E$ e $Distancia_D$, que representam as distâncias percorridas pelas rodas.

Com os valores de distância das rodas calculados, é necessário calcular o deslocamento linear do robô utilizando a equação Equação 3.8:

$$D_{Linear} = \frac{Distancia_E + Distancia_D}{2} \tag{3.8}$$

O deslocamento angular é calculado utilizando a equação Equação 3.9, onde l representa a distância entre as rodas do robô.

$$\delta_\theta = \frac{Distancia_E - Distancia_D}{l} \tag{3.9}$$

Utilizando as equações Equação 3.9 e 3.8, foi possível criar as equações Equação 3.10, 3.11 e 3.12 para capturar a trajetória do robô. Mesmo que essa trajetória seja uma representação não tão fidedigna, pois este método é muito útil para o monitoramento de trajetória de robôs, ele pode apresentar algumas variações devido a erros como travamento das rodas, deslizamento das rodas ou problemas na trajetória, que podem interferir seriamente neste método de captura de trajetória.

$$x \mathrel{+}= D_{Linear}.\cos\left(\theta + \frac{\delta_\theta}{2}\right) \tag{3.10}$$

$$y \mathrel{+}= D_{Linear}.\sin\left(\theta + \frac{\delta_\theta}{2}\right) \tag{3.11}$$

$$\theta \mathrel{+}= \delta_\theta \tag{3.12}$$

As variáveis x, y e θ iniciam como zero para o momento inicial do robô, e esses valores são utilizados para plotar os gráficos da trajetória.

4 RESULTADOS E DISCUSSÃO

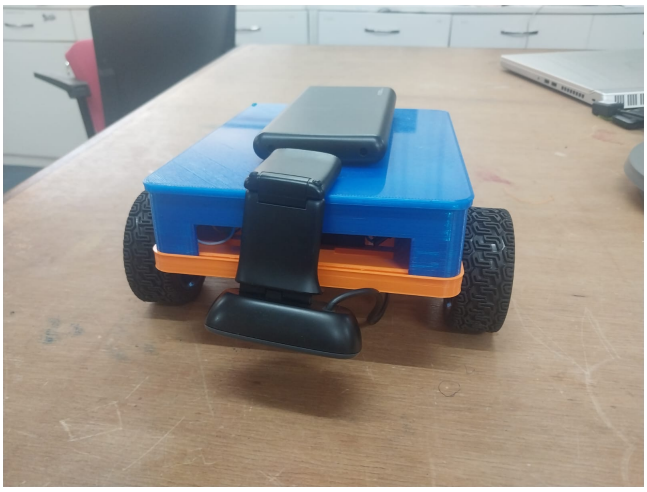
Nesta seção, serão apresentados os resultados e discussões deste projeto, que foram fundamentais para sua conclusão.

4.1 Montagem do robô

O chassi, conforme apresentado no capítulo de Materiais e Métodos, passou por modificações para acomodar a nova câmera, que substituiria a câmera da *Raspberry*. Durante esse processo, o suporte da câmera foi removido do chassi. Os encaixes para os motores e suas caixas de redução foram mantidos, mas ajustados para garantir a fixação adequada dos motores. Além disso, foram adicionadas braçadeiras para assegurar a estabilidade dos motores durante o funcionamento do robô. Uma modificação significativa em relação ao modelo projetado foi a redução das dimensões do chassi, que originalmente tinha 25 cm por 20 cm e foi alterado para 18 cm por 15 cm. Essa alteração foi muito útil, pois diminuiu consideravelmente o peso do robô, permitindo que ele realizasse curvas com mais facilidade.

Como discutido no Capítulo 3, o chassi foi fatiado utilizando o PrusaSlicer, resultando na visualização apresentada na Figura 26, onde o chassi foi impresso, conforme ilustrado na Figura 27. Esse modelo foi produzido utilizando a Impressora 3D Sethi S4X. Os parâmetros de impressão, incluindo altura de camada, velocidade de impressão, densidade de preenchimento, suporte, temperatura do bico e temperatura da mesa, podem ser observados na Tabela 2.

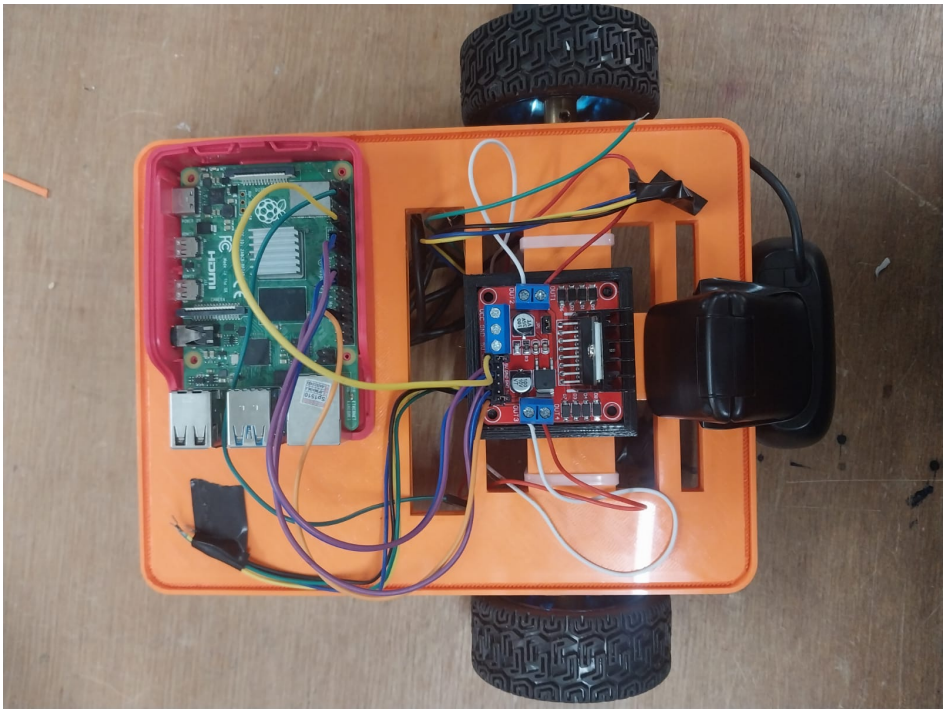
Figura 39 – Robô Final.



Fonte: Elaborada pelo autor, 2024.

O chassi foi projetado para acomodar o *Raspberry Pi*, o *Power Bank*, a Ponte H e a bateria dos motores, como pode ser observado na Figura 40. Uma tampa foi incorporada ao chassi para ocultar os cabos, proporcionando um aspecto mais organizado. Essas modificações culminaram no chassi final, conforme ilustrado na Figura 39.

Figura 40 – Parte interna do Robô.

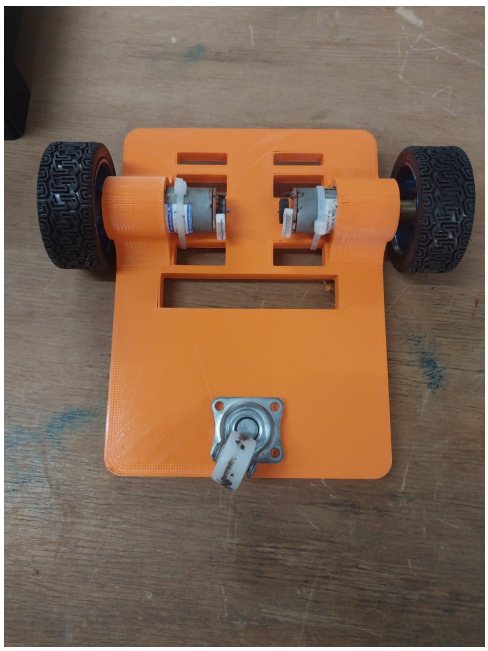


Fonte: Elaborada pelo autor, 2024.

Com o chassi montado, foi necessário encaixar os motores e, em seguida, implementar as rodas neles. É relevante destacar que os motores desempenhavam um papel crucial no controle do movimento do robô, determinando a direção em que ele se deslocaria. Para otimizar essa capacidade, adicionou-se uma roda do tipo castor na parte traseira do robô. Essa roda simplesmente acompanhava o movimento das rodas dianteiras, sem interferir no deslocamento geral do robô. A disposição das rodas pode ser visualizada na Figura 41.

A câmera do robô foi posicionada no encaixe em cima da tampa do chassi, possibilitando que ela ficasse direcionada para a linha da trajetória. Isso evitou muitas variações na trajetória que poderiam ocorrer se estivesse em uma posição mais baixa no robô. Dessa forma, encontrava-se em uma posição ideal para detectar a trajetória, permitindo que o controlador empregado alcançasse uma maior eficiência em sua aplicação.

Figura 41 – Vista das rodas do robô.



Fonte: Elaborada pelo autor, 2024.

4.2 Aplicação do Código de Processamento Digital de Imagens

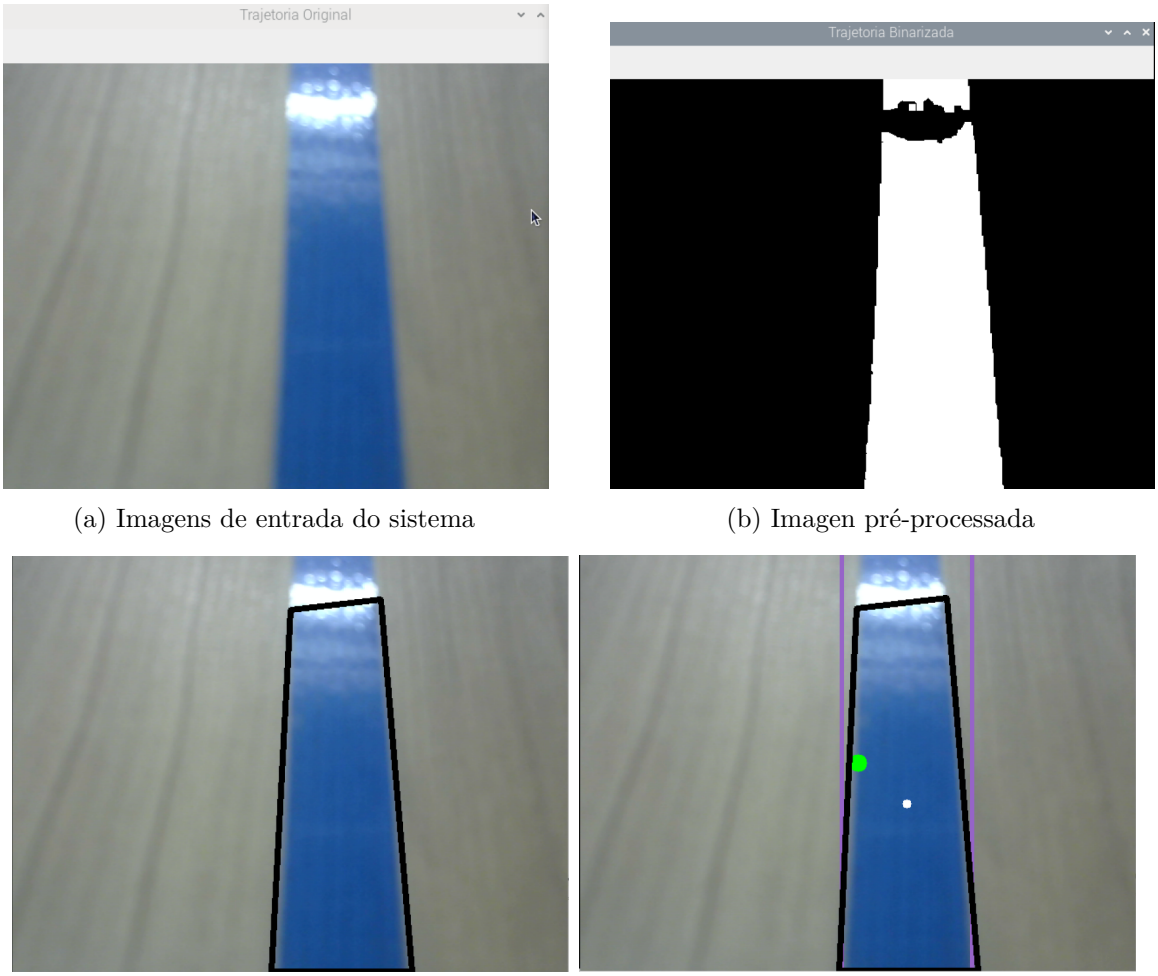
Como detalhado no capítulo de Materiais e Métodos, a detecção inicial da trajetória é um aspecto crucial, utilizando um código de rastreamento para delimitar a região onde a cor escolhida está definida. Esse código foi empregado para determinar os valores associados à cor da trajetória, e essa etapa está diretamente ligada à aquisição e pré-processamento da imagem, incluindo a transformação da imagem de RGB para HSV. Posteriormente, os valores são ajustados para uma faixa desejada de cor, resultando na binarização da imagem, onde o branco representa a cor da trajetória desejada e o preto representa tudo que não faz parte da trajetória. Essa transformação pode ser visualizada na Figura 42(b), apresentando a imagem capturada pela câmera juntamente com a imagem processada, evidenciando apenas a trajetória que o robô deve seguir.

Para reduzir ruídos na imagem, realizou-se uma operação de erosão seguida por uma dilatação, assegurando que ruídos não influenciassem no modelo final desejado.

Após a conclusão desse pré-processamento, ocorreu a segmentação da imagem utilizando os valores de HSV definidos no código de rastreamento para separar e definir a trajetória, conforme demonstrado na Figura 42(c). Com a trajetória identificada, realizou-se a extração de características, sendo as bordas essenciais para detectar o centro. Essa etapa revelou-se crucial para que o robô pudesse quantificar a distância entre seu centro e o centro da trajetória, utilizando essas informações para calcular o erro, conforme ilustrado

na Figura 42(d). Nessa imagem, a trajetória é identificada, e seu centro é representado pelo ponto branco. Além disso, é possível identificar o centro do robô, destacado como o ponto verde na imagem.

Figura 42 – Processamento Digital da Imagem de Entrada.



(a) Imagens de entrada do sistema

(b) Imagen pré-processada

(c) Detecção da trajetória.

(d) Detecção do centro da trajetória.

Fonte: Elaborada pelo autor, 2024.

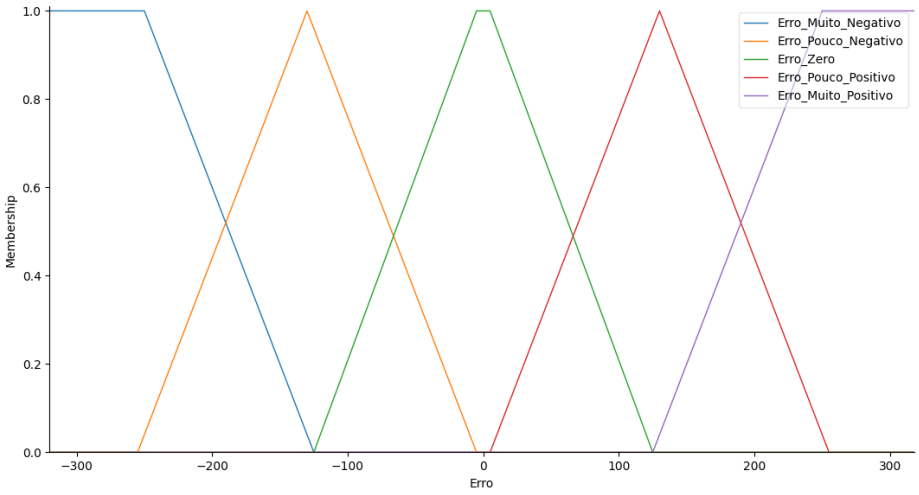
4.3 Controlador *Fuzzy* em python

O controlador *Fuzzy* desenvolvido no MATLAB foi refeito em Python para ser utilizado em conjunto com o código OpenCV para detectar a trajetória, bem como com o código de controle dos motores. Essa adaptação foi necessária, uma vez que o código *Fuzzy* gerado pelo MATLAB não pode ser interpretado diretamente pelo Python, assim sendo necessário reescrevê-lo utilizando a biblioteca scikit-*Fuzzy*.

O código do controlador *Fuzzy* possui duas variáveis linguísticas de entrada e duas de saída, além de 25 regras para o sistema, conforme apresentado no capítulo de Materiais e Métodos. O controlador *Fuzzy* em Python gerou gráficos para as variáveis linguísticas

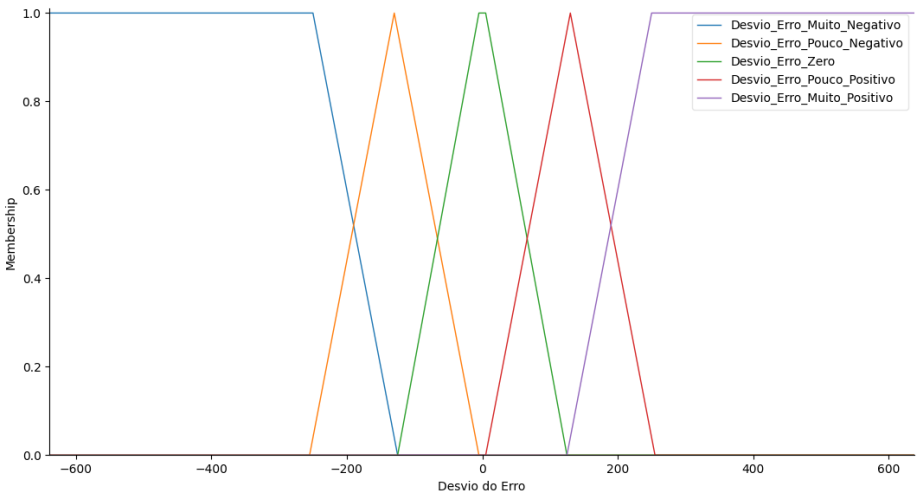
de entrada e saída. É importante ressaltar que foram utilizados os mesmos valores das variáveis linguísticas definidas no MATLAB, com exatamente o mesmo intervalo. Os resultados podem ser observados nas Figuras 43, 44, 45 e 46. Além disso, as 25 regras estão representadas na Figura 47.

Figura 43 – Variável Linguística do Erro em Pixels no Python.



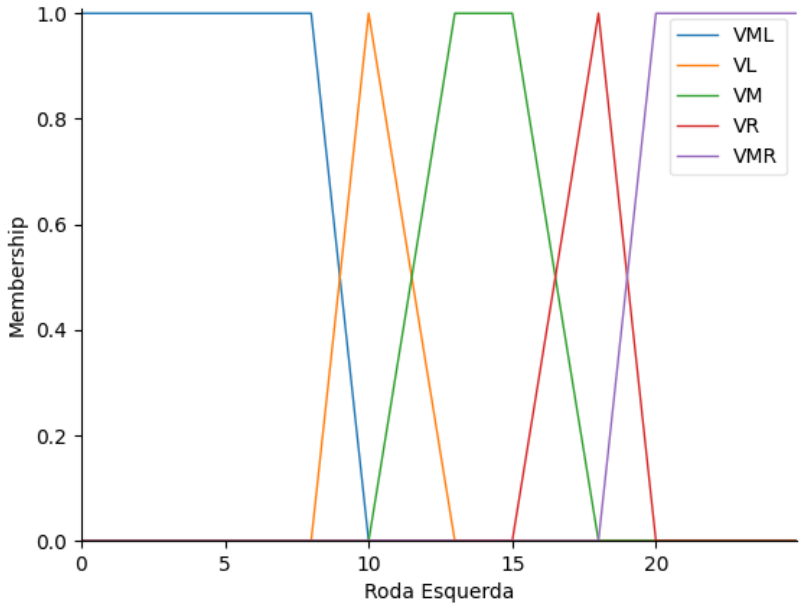
Fonte: Elaborada pelo autor, 2024.

Figura 44 – Variável Linguística do Desvio do Erro em Pixels no Python.



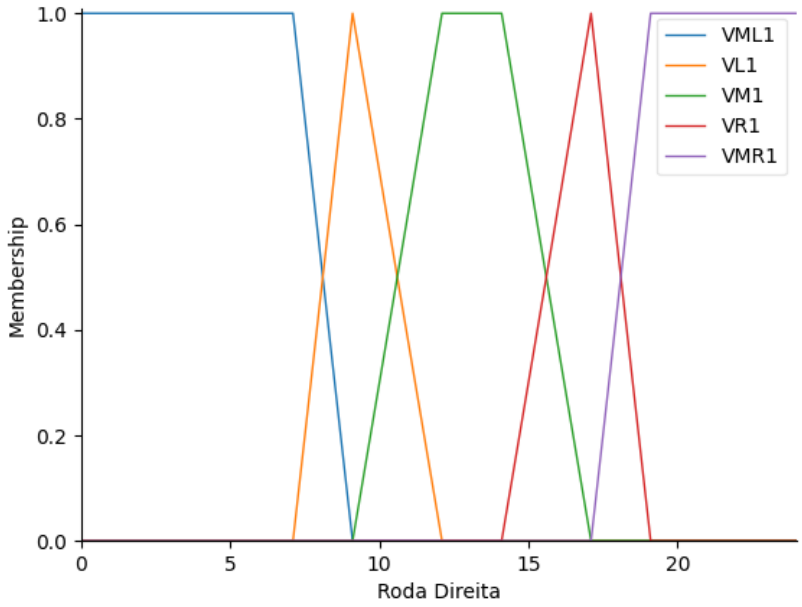
Fonte: Elaborada pelo autor, 2024.

Figura 45 – Variável Linguística da Roda Esquerda no Python.



Fonte: Elaborada pelo autor, 2024.

Figura 46 – Variável Linguística da Roda Direita no Python.



Fonte: Elaborada pelo autor, 2024.

Figura 47 – Regras do *Fuzzy* em Python.

```
# Define as regras fuzzy
regra1 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['Desvio_Erro_Zero'], (Roda_Esquerda['VM'], Roda_Direita['VM1']))
regra2 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VM'], Roda_Direita['VM1']))
regra3 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VM1']))
regra4 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VL'], Roda_Direita['VM1']))
regra5 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VL1']))
#------------------------
regra6 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['Desvio_Erro_Zero'], (Roda_Esquerda['VL'], Roda_Direita['VM1']))
regra7 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VL'], Roda_Direita['VR1']))
regra8 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VML'], Roda_Direita['VR1']))
regra9 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VR1']))
regra10 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VL'], Roda_Direita['VM1']))
#------------------------
regra11 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['Desvio_Erro_Zero'], (Roda_Esquerda['VM'], Roda_Direita['VL1']))
regra12 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VR'], Roda_Direita['VL1']))
regra13 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VR'], Roda_Direita['VML1']))
regra14 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VR'], Roda_Direita['VM1']))
regra15 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VL1']))
#------------------------
regra16 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['Desvio_Erro_Zero'], (Roda_Esquerda['VML'], Roda_Direita['VM1']))
regra17 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VML'], Roda_Direita['VR1']))
regra18 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VML'], Roda_Direita['VMR1']))
regra19 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VL'], Roda_Direita['VMR1']))
regra20 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VMR1']))
#------------------------
regra21 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['Desvio_Erro_Zero'], (Roda_Esquerda['VM'], Roda_Direita['VML1']))
regra22 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VR'], Roda_Direita['VML1']))
regra23 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VMR'], Roda_Direita['VM1']))
regra24 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VMR'], Roda_Direita['VL1']))
regra25 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VMR'], Roda_Direita['VML1']))
```

Fonte: Elaborada pelo autor, 2024.

Com isso, implementou-se o código de controle *Fuzzy* junto com os outros códigos, gerando assim uma relação em que o código de detecção da trajetória envia para o código *Fuzzy* os valores de erro e desvio do erro. Esses valores são então aplicados no controlador *Fuzzy*, que retorna os valores de PWM para os motores esquerdo e direito do robô, dependendo de qual regra os valores de erro e desvio se encaixem. Essa abordagem integrada permite que o robô ajuste sua trajetória com base nas condições detectadas, proporcionando um controle mais adaptativo e preciso.

A Figura 48 exibe um teste do sistema para entradas específicas. Conforme observado, o sistema acelera mais a roda direita, ao mesmo tempo em que desacelera a roda esquerda, resultando em uma curva. Isso ocorre devido ao erro ser muito positivo, indicando a necessidade de o robô realizar uma curva para a esquerda na tentativa de aproximar o erro entre o centro do robô e o centro da trajetória a zero.

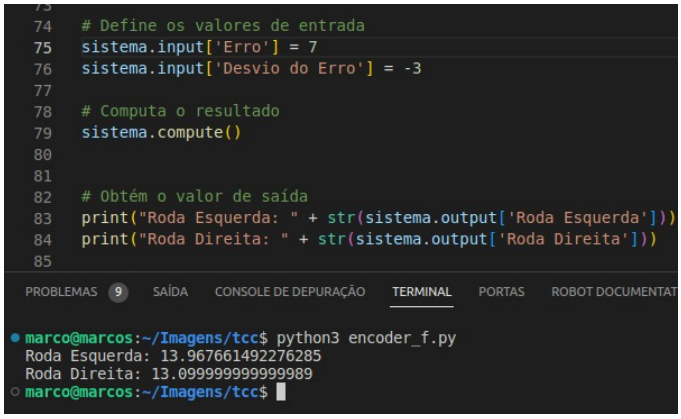
Figura 48 – Teste com entradas predefinidas.

```
74  # Define os valores de entrada
75  sistema.input['Erro'] = 241
76  sistema.input['Desvio do Erro'] = 44
77
78  # Computa o resultado
79  sistema.compute()
80
81
82  # Obtém o valor de saída
83  print("Roda Esquerda: " + str(sistema.output['Roda Esquerda']))
84  print("Roda Direita: " + str(sistema.output['Roda Direita']))
85

PROBLEMAS 9   SAÍDA   CONSOLE DE DEPURAÇÃO   TERMINAL   PORTAS   ROBOT DOCUMENTAT

marco@marcos:~/Imagens/tcc$ python3 encoder_f.py
Roda Esquerda: 6.181221032159587
Roda Direita: 15.674237358707478
marco@marcos:~/Imagens/tcc$
```

Fonte: Elaborada pelo autor, 2024.

A Figura 49 mostra uma saída em que o erro está dentro da margem permitida pela função de pertinência do tipo trapezoidal do erro. Nesse caso, o sistema mantém uma velocidade média para ambos os motores, mesmo para um valor de erro diferente de zero, desde que esteja dentro da margem de erro permitida. Esse comportamento ilustra a capacidade do controlador *Fuzzy* de lidar de maneira adaptativa com diferentes situações, mantendo o robô em movimento suave mesmo em condições variadas.

Figura 49 – Teste com entradas predefinidas.



Fonte: Elaborada pelo autor, 2024.

4.4 Controlador PID do robô

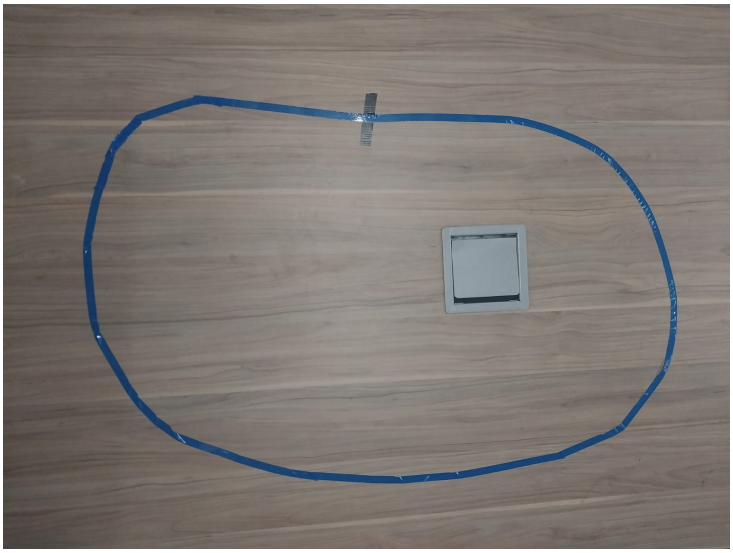
O controlador PID foi implementado de maneira semelhante ao controlador *Fuzzy*. Ele utiliza como entrada o valor do erro entre o centro do robô e o centro da trajetória, sendo definido um setpoint igual a zero para esse sistema. O valor de saída do PID é enviado para os motores como um valor de PWM. Para ter um controle mais preciso dos motores, foi estabelecido que a velocidade medida do robô resultaria em valores de PWM iguais a 15 para a roda esquerda e 14.1 para a roda direita. O valor calculado pelo PID era somado ou subtraído do valor de PWM, dependendo de quão à esquerda ou à direita o centro da trajetória estava. A implementação do controle dos motores e o PDI da câmera foram os mesmos aplicados no controle *Fuzzy*. Os valores das constantes K_p, K_d e K_i foram definidos por meio de testes com o robô, ajustados para garantir um desempenho adequado.

4.5 Odometria

Utilizando as equações do capítulo de Materiais e Métodos, na seção sobre Cálculos para Odometria, foi desenvolvida e implementada em Python a aplicação dessas equações em conjunto com os controladores *Fuzzy* e PID. Após a elaboração do código, foram conduzidos testes para avaliar o desempenho dos controladores.

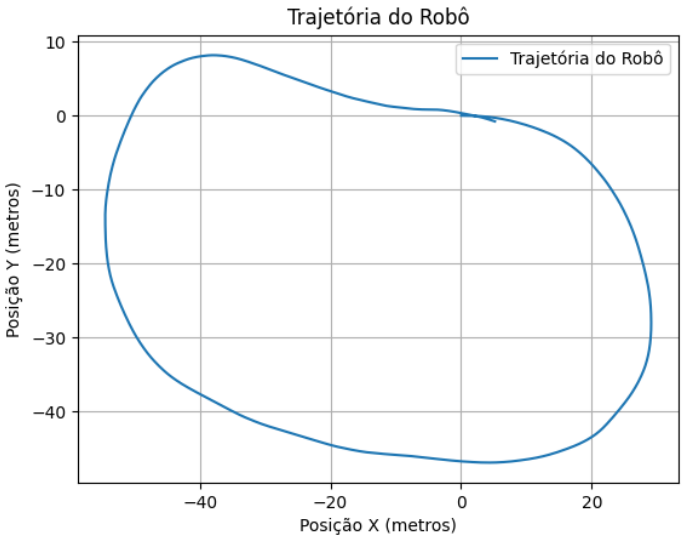
É relevante salientar que a odometria foi empregada exclusivamente para capturar a trajetória do robô. No entanto, devido a desafios como o deslizamento das rodas e algumas interferências causadas por variações de luminosidade na trajetória do robô, a odometria apresentou algumas imprecisões. As Figura 50, 51 e 52 proporcionam uma comparação entre a trajetória projetada para o robô e os resultados obtidos pela odometria nos controladores *Fuzzy* e PID.

Figura 50 – Trajetória Original.



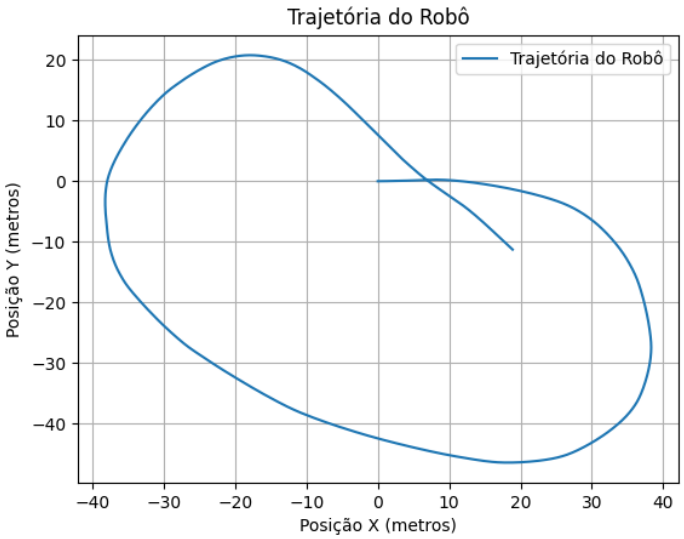
Fonte: Elaborada pelo autor, 2024.

Figura 51 – Odometria gerada pelo Controlador *Fuzzy*.



Fonte: Elaborada pelo autor, 2024.

Figura 52 – Odometria gerada pelo Controlador PID.



Fonte: Elaborada pelo autor, 2024.

Analisando as respostas resultantes das odometrias dos controladores, é possível afirmar que o robô não iniciou no mesmo ponto para ambas as odometrias, apresentando um erro ao retornar para o ponto inicial do robô. Apesar desses desvios, ambas as odometrias estão próximas da trajetória original. Além disso, os resultados das odometrias dos controladores demonstraram uma proximidade, indicando que a trajetória estava sendo seguida corretamente.

5 CONSIDERAÇÕES FINAIS

Com os resultados obtidos no Capítulo 4, foi possível compreender que o sistema desenvolvido em Python, projetado para capturar imagens e processá-las posteriormente utilizando o OpenCV para detectar a trajetória e identificar seu centro, permitiu que a utilização do controlador *Fuzzy* mantivesse o robô próximo ao centro da trajetória.

Para que o robô pudesse seguir a trajetória, foi necessário inicialmente definir o que seria essa trajetória e, a partir disso, determinar seu centro. Esse processo foi essencial, pois saber onde está localizado o centro da trajetória possibilita determinar se o robô está muito à esquerda ou à direita em relação a esse centro. Esse aspecto foi crucial para o seguimento da trajetória. Se o centro do robô estivesse muito distante do centro da trajetória e não tentasse se aproximar, o robô não conseguiria manter-se na trajetória, comprometendo assim o propósito de um seguidor de linha.

O processamento digital de imagem aplicado na detecção da trajetória e do seu centro mostrou-se muito satisfatório. Foram aplicadas técnicas morfológicas de erosão e dilatação para eliminar ruídos, garantindo assim uma detecção precisa da trajetória. Esse processamento também se mostrou eficiente para variações de luminosidade que ocorreram nas trajetórias, como pode ser visto na Figura 42(b), onde devido à luminosidade refletida pela trajetória, ocorria um erro na detecção da trajetória, pois a falha não era reconhecida como parte da trajetória.

Com o resultado da diferença entre o centro do robô e o centro da trajetória, foi possível mensurar a distância do robô em relação à trajetória, e esse valor de diferença foi utilizado no controlador *Fuzzy*, que foi implementado utilizando o método Mamdani. Nesse método, foram criadas variáveis linguísticas e, com isso, estabeleceram-se as regras do sistema para controlar a velocidade dos motores esquerdo e direito com base em valores de PWM definidos por meio de testes.

Os resultados do controlador *Fuzzy* foram considerados muito adequados. Um dos principais pontos no desenvolvimento do controlador *Fuzzy* foi a variável linguística da roda esquerda e direita, pois não foi aplicado um controlador diretamente nesses motores para garantir que o valor de PWM aplicado estivesse correto, e com isso, os motores estavam expostos a falhas físicas que poderiam resultar em erros ao seguir a trajetória, mas isso acabou não ocorrendo.

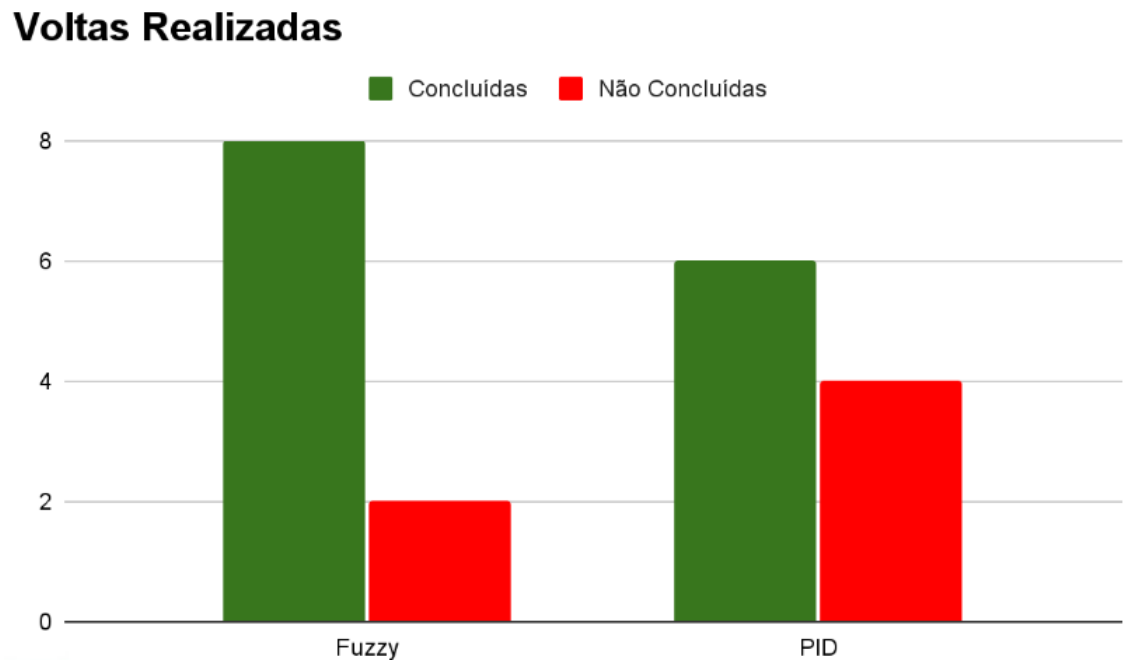
Ao analisar os resultados obtidos a partir da odometria em cinco testes para ambos os controladores, *Fuzzy* e PID, observou-se que o controlador *Fuzzy* apresentou desempenho superior. Nos cinco testes realizados com ambos os controladores, o *Fuzzy* gerou odometrias mais próximas da trajetória original em comparação ao PID. O controlador *Fuzzy* alcançou

uma taxa de sucesso de 100%, produzindo trajetórias mais precisas em todos os testes, enquanto o PID obteve sucesso em apenas 2 a 3 testes para a mesma quantidade.

Embora o controlador PID tenha demonstrado a capacidade de concluir a trajetória em menos tempo, com uma média de 26 segundos em 10 testes, o controlador *Fuzzy*, com uma média de 32 segundos, mostrou-se mais consistente na geração de odometrias precisas. A análise revelou que o PID apresentou uma vantagem em termos de tempo de conclusão, mas o *Fuzzy* manteve um erro menor entre o centro do robô e a trajetória original, conforme indicado pelos resultados da odometria.

Outro ponto relevante é a frequência de falhas. O controlador *Fuzzy* teve um índice de falha de 20%, não conseguindo completar a trajetória em 2 dos 10 testes. Em contraste, o PID falhou em 40% dos testes, não conseguindo concluir a trajetória em 4 dos 10 casos, isso pode ser visto na Figura 53. Essa discrepância sugere que o controlador *Fuzzy* apresenta uma maior adaptabilidade para realizar curvas ou corrigir a direção do robô em comparação ao PID.

Figura 53 – Gráfico de Comparação entre *Fuzzy* e PID.



Fonte: Elaborada pelo autor, 2024.

Em resumo, apesar de o PID ter a vantagem do tempo de conclusão e ter concluído um maior número de testes, o controlador *Fuzzy* demonstrou superioridade na precisão da odometria, adaptabilidade e taxa de falhas mais baixa. Essas características tornam o controlador *Fuzzy* mais adequado para a aplicação específica deste projeto.

A odometria foi utilizada como parâmetro para comparar a trajetória real com a trajetória desenvolvida pelo robô nos dois controladores desenvolvidos.

Acredita-se que este trabalho tenha alcançado seu objetivo de aplicar o controlador *Fuzzy* com processamento digital de imagens, além de permitir que futuros trabalhos utilizem os resultados da comparação entre o controlador *Fuzzy* e o controlador PID, além de ajudar no desenvolvimento de futuros projetos com processamento digital de imagens aplicados para detectar trajetórias.

Sugere-se para trabalhos futuros aplicar um controlador nos motores para manter o valor de PWM enviado pelo controlador para o motor. Outra ideia seria criar uma relação da velocidade das rodas com o PWM dos motores e, com isso, utilizar os encoders dos motores para controlar essa velocidade. Uma outra melhoria seria aprimorar o processamento das imagens para reduzir o tempo em que o robô define qual será o valor de PWM com base na distância do centro do robô para o centro da trajetória.

A implementação de uma sub-rotina para reconhecer a trajetória e segui-la, mesmo quando o robô não estiver em cima da trajetória, poderia ser uma grande evolução no projeto. Isso possibilitaria não apenas o seguimento da trajetória, mas também a localização e movimentação do robô até o início dela.

A inclusão de um segundo sensor visa aprimorar a odometria gerada pelo sistema, permitindo a implementação de um sistema de posicionamento mais preciso para o robô.

REFERÊNCIAS

3DLAB. Conheça os tipos de impressão 3D e os seus benefícios! 2018. Disponível em: <https://3dlab.com.br/tipos-de-impressao-3d-e-beneficios/>. Acesso em: fevereiro de 2024. Citado na página 34.

ALVES, P. **Ponte H – O que é e como funciona!** 2018. Disponível em: <https://www.manualdaeletronica.com.br/ponte-h-o-que-e-como-funciona/>. Acesso em: fevereiro de 2024. Citado na página 40.

BACKES, A. R.; JUNIOR, S.; MESQUITA, J. J. de. **Introdução à visão computacional usando Matlab**. [S.l.]: Alta Books Editora, 2016. Citado na página 18.

BARELLI, F. **Introdução à visão computacional: Uma abordagem prática com Python e OpenCV**. [S.l.]: Editora Casa do Código, 2018. Citado na página 34.

BASTOS, F. F. *et al.* Estudo e implementação de controladores fuzzy e pid para controle de direção e velocidade de um agv com visão computacional. Universidade Federal Rural do Semi-Árido, 2019. Citado na página 29.

BITTENCOURT, M. R. d. **Filamentos de impressora 3D**. 2021. Disponível em: <https://www.camboriu.ifc.edu.br/labifmaker/wp-content/uploads/sites/19/2021/10/CT_01-Filamentos-de-impressora-3D.pdf>. Acesso em: fevereiro de 2024. Citado na página 44.

BRAGA, I. N. C. **Como funcionam os encoders (MEC128)**. 2017. Disponível em: <https://www.newtoncbraga.com.br/index.php/como-funciona/5454-mec128>. Acesso em: fevereiro de 2024. Citado na página 37.

BUTTAY, C. **Ponte H**. 2006. Disponível em: <https://pt.wikipedia.org/wiki/Ponte_H>. Acesso em: fevereiro de 2024. Citado na página 41.

CHUDASAMA, D. *et al.* Image segmentation using morphological operations. **International Journal of Computer Applications**, Citeseer, v. 117, n. 18, 2015. Citado nas páginas 27 e 28.

CIPOLI, P. **O que é VNC?** 2012. Disponível em: <https://canaltech.com.br/seguranca/o-que-e-vnc/>. Acesso em: fevereiro de 2024. Citado na página 50.

CORKE, P. I.; KHATIB, O. **Robotics, vision and control: fundamental algorithms in MATLAB**. [S.l.]: Springer, 2011. v. 73. Citado na página 24.

COSTER, M.; CHERMANT., J.-L. **Image analysis and mathematical morphology for civil engineering materials**. [S.l.]: Jean Serra, 1982. Citado na página 26.

FILHO, O. M.; NETO, H. V. **Processamento digital de imagens**. [S.l.]: Brasport, 1999. Citado nas páginas 19, 20, 21, 22, 23 e 25.

GONZALEZ, R. C.; WOODS, R. C. **Processamento digital de imagens .** [S.l.]: Pearson Educación, 2009. Citado nas páginas 19, 24 e 25.

HAILEY, N. **Prusa Slicer - Getting Started with Prusa Slicer (for Any 3D Printer)**. 2023. Disponível em: <https://www.obico.io/blog/prusa-slicer-getting-started/#:~:text=PrusaSlicer%20is%20a%20free%2C%20open,its%20own%20in%2Dhouse%20slicer.> Acesso em: fevereiro de 2024. Citado na página 47.

INSTRUMENTS, G. P. **Understanding Quadrature**. 2002. Disponível em: <https://static1.squarespace.com/static/5b9183b8fcf7fdac4d3a1067/t/5b93d02e4d7a9cece56a9d7c/1536413742474/Understanding_Quadrature.pdf>. Acesso em: fevereiro de 2024. Citado na página 37.

JANG, J.-S. Anfis: adaptive-network-based fuzzy inference system. **IEEE transactions on systems, man, and cybernetics**, IEEE, v. 23, n. 3, p. 665–685, 1993. Citado na página 30.

KLIR, G.; YUAN, B. **Fuzzy sets and fuzzy logic**. [S.l.]: Prentice hall New Jersey, 1995. v. 4. Citado nas páginas 29 e 30.

MARCHAND-MAILLET, S.; SHARAIHA, Y. M. **Binary digital image processing: a discrete approach**. [S.l.]: Academic Press, 1999. Citado na página 26.

NIKU, S. B. **Introduction to robotics: analysis, control, applications**. [S.l.]: John Wiley & Sons, 2020. Citado nas páginas 15 e 31.

OGATA, K. **Modern control engineering**. [S.l.]: Pearson Universidades, 2010. Citado na página 28.

OPENCV. **About**. 2024. Disponível em: <https://opencv.org/about/>. Acesso em: fevereiro de 2024. Citado nas páginas 34 e 35.

PI, R. **Raspberry Pi 4**. 2019. Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>. Acesso em: fevereiro de 2024. Citado na página 42.

PYTHON. **venv — Criação de ambientes virtuais**. 2019. Disponível em: <https://docs.python.org/pt-br/3/library/venv.html>. Acesso em: fevereiro de 2024. Citado na página 50.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. **Rita**, v. 13, n. 2, p. 11–42, 2006. Citado nas páginas 20 e 24.

SHA, C.; HOU, J.; CUI, H. A robust 2d otsu's thresholding method in image segmentation. **Journal of Visual Communication and Image Representation**, Elsevier, v. 41, p. 339–351, 2016. Citado na página 26.

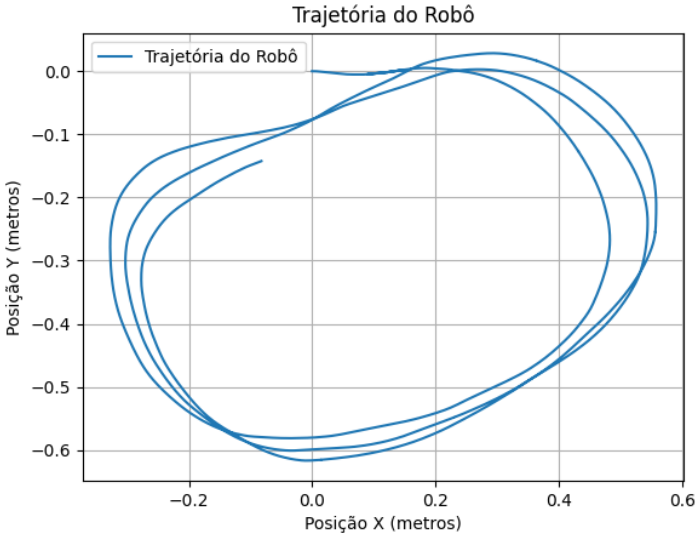
SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robot, 1a edição**. [S.l.: s.n.], 2004. Citado nas páginas 15, 17 e 18.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. **Robot modeling and control**. [S.l.]: John Wiley & Sons, 2020. Citado na página 16.

UMANS, S. D. **Máquinas Elétricas de Fitzgerald e Kingsley-7**. [S.l.]: AMGH Editora, 2014. Citado nas páginas 35 e 36.

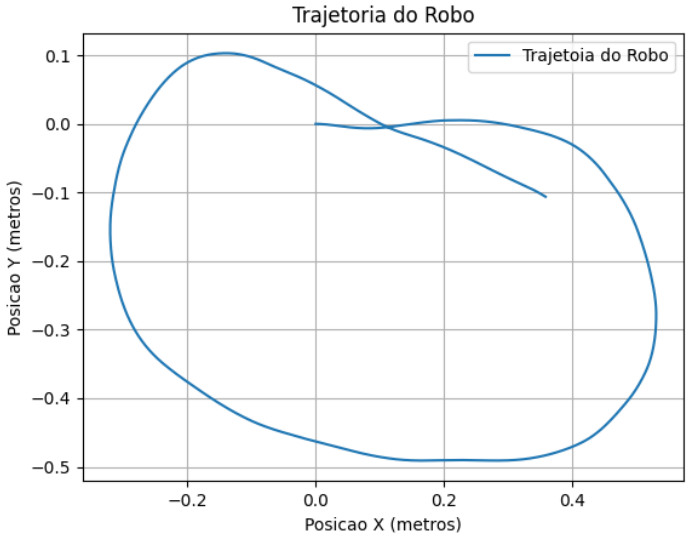
APÊNDICE A – IMAGENS DO TESTE DE ODOMETRIA

Figura 54 – Odometria gerada pelo Controlador Fuzzy.



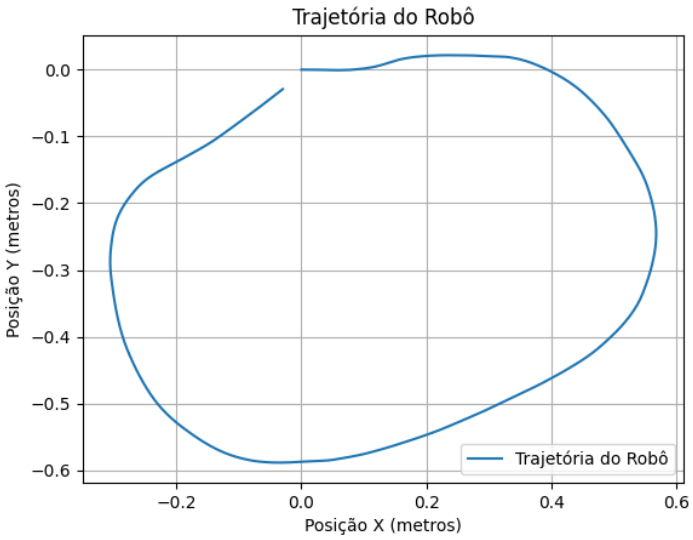
Fonte: Elaborada pelo autor, 2024.

Figura 55 – Odometria gerada pelo Controlador PID.



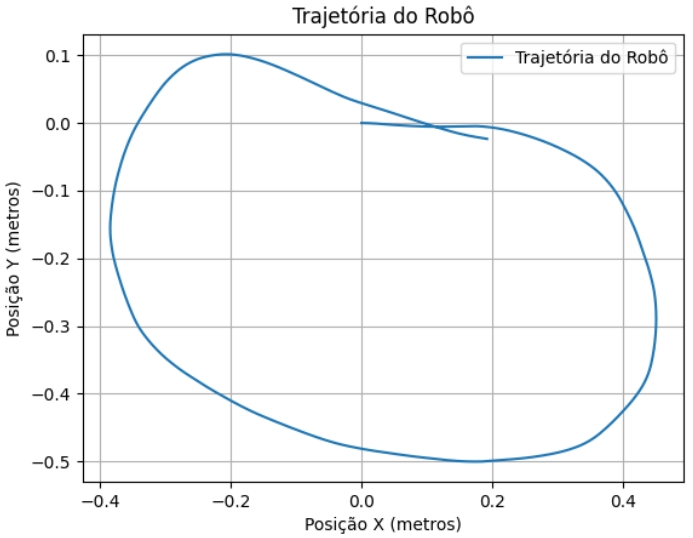
Fonte: Elaborada pelo autor, 2024.

Figura 56 – Odometria gerada pelo Controlador Fuzzy.



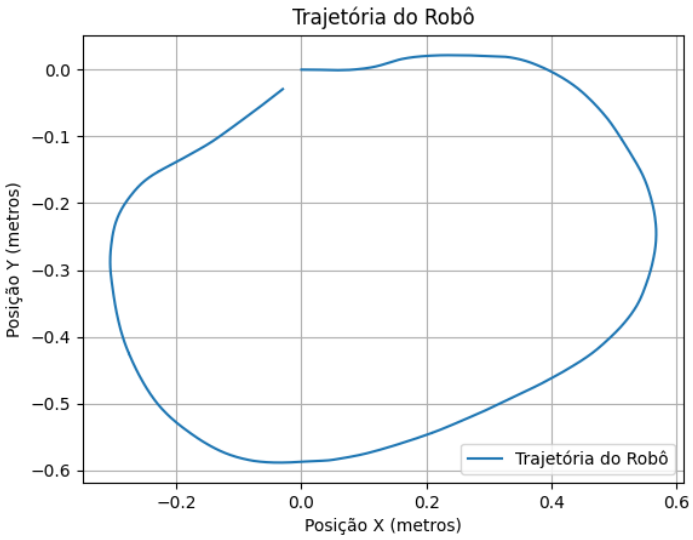
Fonte: Elaborada pelo autor, 2024.

Figura 57 – Odometria gerada pelo Controlador PID.



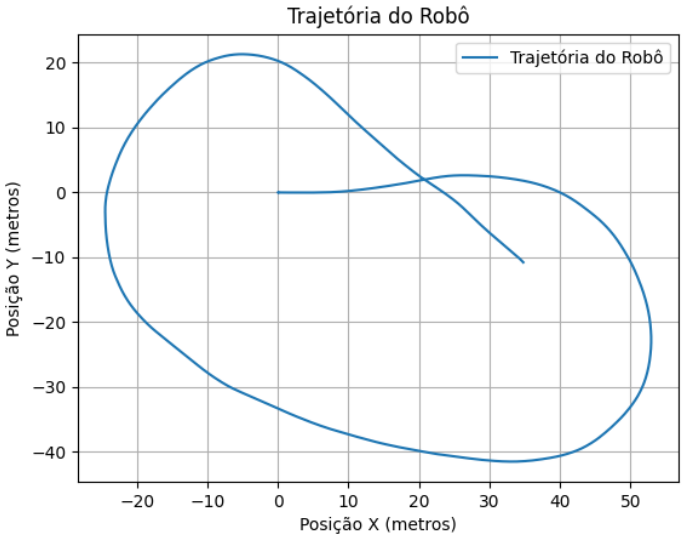
Fonte: Elaborada pelo autor, 2024.

Figura 58 – Odometria gerada pelo Controlador Fuzzy.



Fonte: Elaborada pelo autor, 2024.

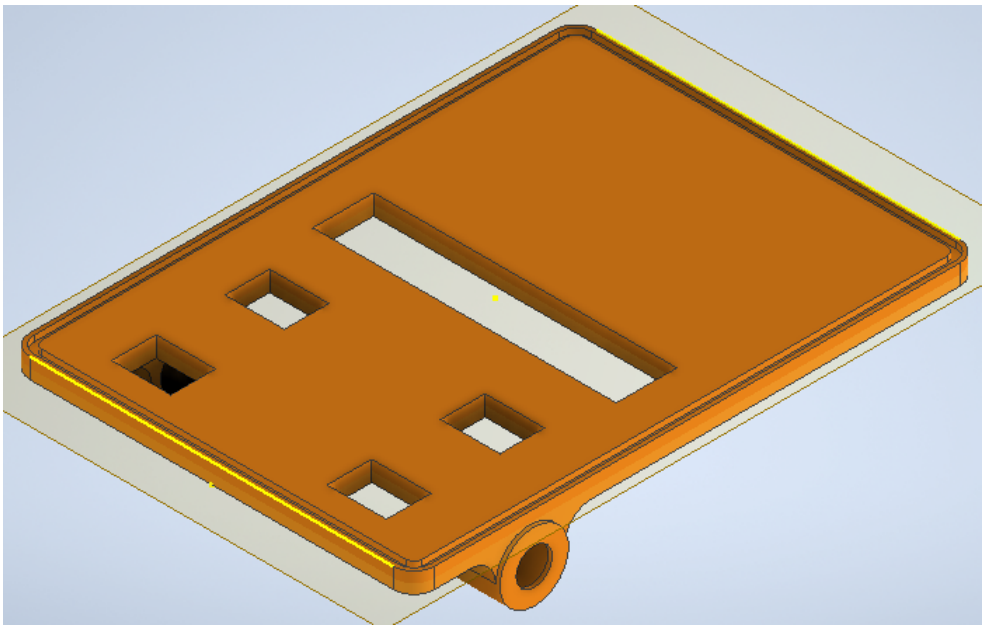
Figura 59 – Odometria gerada pelo Controlador PID.



Fonte: Elaborada pelo autor, 2024.

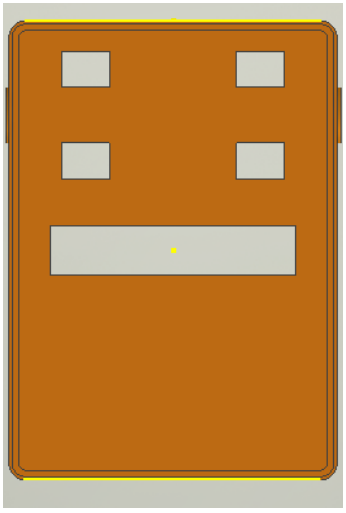
APÊNDICE B – IMAGENS DO MODELO 3D DO CHASSI DO ROBÔ

Figura 60 – Chassi do Robô.



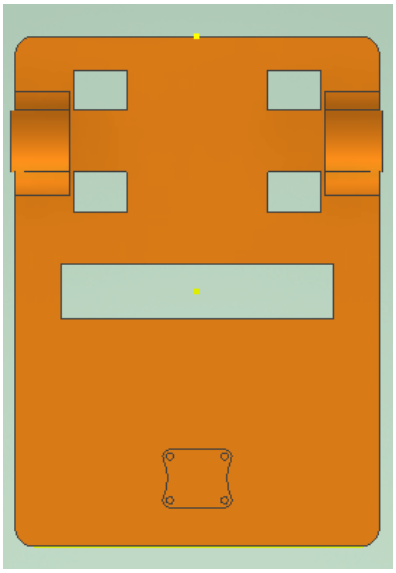
Fonte: Elaborada pelo autor, 2024.

Figura 61 – Vista superior do Chassi do Robô.



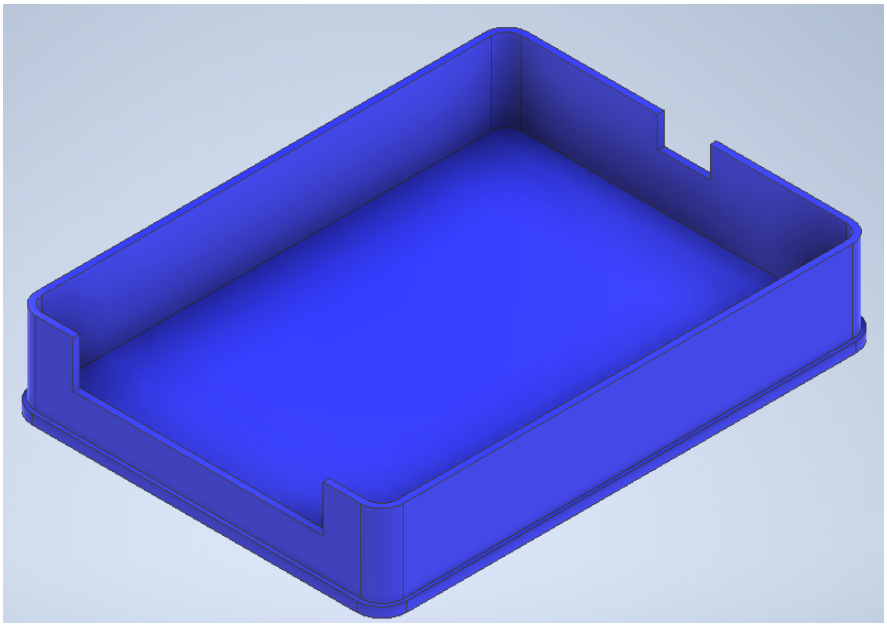
Fonte: Elaborada pelo autor, 2024.

Figura 62 – Vista inferior do Chassi do Robô.



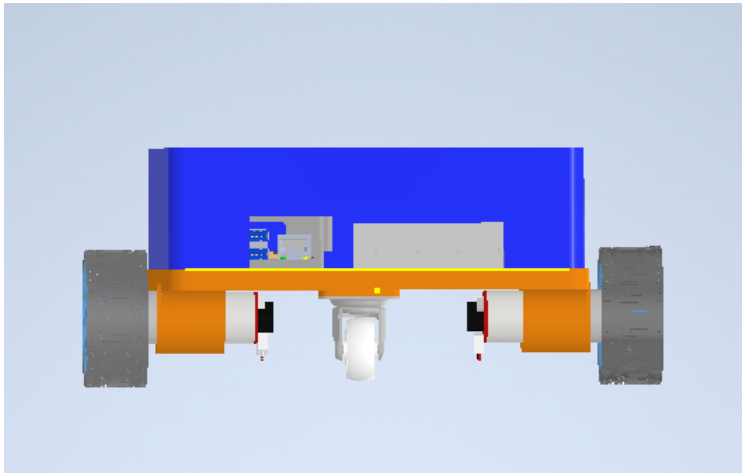
Fonte: Elaborada pelo autor, 2024.

Figura 63 – Tampa do Chassi.



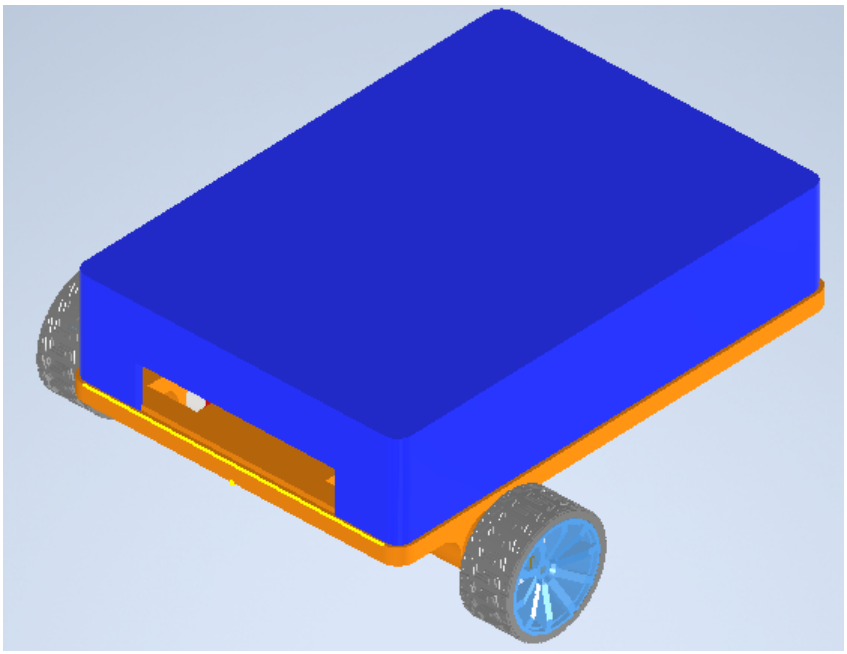
Fonte: Elaborada pelo autor, 2024.

Figura 64 – Vista frontal do Chassi do Robô.



Fonte: Elaborada pelo autor, 2024.

Figura 65 – Modelo 3D final do Robô.



Fonte: Elaborada pelo autor, 2024.

APÊNDICE C – CÓDIGO DE DETECÇÃO

```
import cv2

# Definindo os intervalos de cor HSV para a segmentacao
H_MIN = 0
H_MAX = 255
S_MIN = 0
S_MAX = 255
V_MIN = 0
V_MAX = 255

# Configuracoes da captura de video
FRAME_WIDTH = 640
FRAME_HEIGHT = 480
MAX_NUM_OBJECTS = 50
MIN_OBJECT_AREA = 20 * 20
MAX_OBJECT_AREA = FRAME_HEIGHT * FRAME_WIDTH / 1.5

# Nomes das janelas
windowName = "Imagem Original"
windowName1 = "Imagem HSV"
windowName2 = "Imagem Thresholded"
windowName3 = "Apos Operacoes Morfologicas"
trackbarWindowName = "Trackbars"
testando = 0

# Funcao de callback para a trackbar (nao faz nada)
def nothing(val):
    pass

# Funcao para criar as trackbars na janela
def createTrackbars():
    cv2.namedWindow(trackbarWindowName, cv2.WINDOW_NORMAL)
    cv2.resizeWindow(trackbarWindowName, 600, 400)  # Redimensiona a janela da trackbar
    cv2.createTrackbar("H_MIN", trackbarWindowName, H_MIN, H_MAX, nothing)
    cv2.createTrackbar("H_MAX", trackbarWindowName, H_MAX, H_MAX, nothing)
    cv2.createTrackbar("S_MIN", trackbarWindowName, S_MIN, S_MAX, nothing)
    cv2.createTrackbar("S_MAX", trackbarWindowName, S_MAX, S_MAX, nothing)
    cv2.createTrackbar("V_MIN", trackbarWindowName, V_MIN, V_MAX,
```

```
    nothing)
39      cv2.createTrackbar("V_MAX", trackbarWindowName, V_MAX, V_MAX,
    nothing)
40
41  # Funcao para desenhar o objeto rastreado na imagem
42  def drawObject(x, y, frame):
43      x = int(x)
44      y = int(y)
45      cv2.circle(frame, (x, y), 20, (0, 255, 0), 2)
46      if (y - 25 > 0):
47          cv2.line(frame, (x, y), (x, y - 25), (0, 255, 0), 2)
48      else:
49          cv2.line(frame, (x, y), (x, 0), (0, 255, 0), 2)
50      if (y + 25 < FRAME_HEIGHT):
51          cv2.line(frame, (x, y), (x, y + 25), (0, 255, 0), 2)
52      else:
53          cv2.line(frame, (x, y), (x, FRAME_HEIGHT), (0, 255, 0), 2)
54      if (x - 25 > 0):
55          cv2.line(frame, (x, y), (x - 25, y), (0, 255, 0), 2)
56      else:
57          cv2.line(frame, (x, y), (0, y), (0, 255, 0), 2)
58      if (x + 25 < FRAME_WIDTH):
59          cv2.line(frame, (x, y), (x + 25, y), (0, 255, 0), 2)
60      else:
61          cv2.line(frame, (x, y), (FRAME_WIDTH, y), (0, 255, 0), 2)
62
63      cv2.putText(frame, str(x) + "," + str(y), (x, y + 30), 2, 1, (0,
    255, 255), 2)
64
65  # Funcao para realizar operacoes morfologicas na imagem thresholded
66  def morphOps(thresh):
67      erodeElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
68      dilateElement = cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8))
69      thresh = cv2.erode(thresh, erodeElement)
70      thresh = cv2.dilate(thresh, dilateElement)
71      return thresh
72
73  # Funcao para rastrear o objeto filtrado na imagem
74  def trackFilteredObject(x, y, threshold, cameraFeed):
75      global testando
76
77      temp = threshold
78
79      contours, hierarchy = cv2.findContours(temp, cv2.RETR_CCOMP, cv2.
    CHAIN_APPROX_SIMPLE)
80      refArea = 0
81      objectFound = False
```

```
82      try:
83          if (hierarchy.size > 0):
84              numObjects = hierarchy.size / 4
85              if (numObjects < MAX_NUM_OBJECTS):
86                  index = 0
87                  while index >= 0:
88                      moment = cv2.moments(contours[index])
89                      area = moment['m00']
90                      if (area > MIN_OBJECT_AREA and area <
    MAX_OBJECT_AREA and
91                              area > refArea):
92                          x = moment['m10'] / area
93                          y = moment['m01'] / area
94                          objectFound = True
95                          refArea = area
96                      else:
97                          objectFound = False
98                      index = hierarchy[0][index][0]
99                      testando = hierarchy
100
101             if (objectFound == True):
102                 cv2.putText(cameraFeed, "Rastreando Objeto", (0, 50), 2,
     1, (0, 255, 0), 2)
103                 drawObject(x, y, cameraFeed)
104         else:
105             cv2.putText(cameraFeed, "MUITO RUIDO! AJUSTE O FILTRO", (0,
    50), 1, 2, (0, 0, 255), 2)
106
107     except:
108         pass
109
110 x = 0
111 y = 0
112
113 def main():
114     trackObjects = True
115     useMorphOps = True
116
117     createTrackbars()
118
119     capture = cv2.VideoCapture(0)
120
121     capture.set(cv2.CAP_PROP_FRAME_WIDTH, FRAME_WIDTH)
122     capture.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT)
123
124     while (1):
125         ret, cameraFeed = capture.read()
```

```
126         HSV = cv2.cvtColor(cameraFeed, cv2.COLOR_BGR2HSV)
127         threshold = cv2.inRange(HSV,
128                                 (cv2.getTrackbarPos("H_MIN",
    trackbarWindowName),
129                                  cv2.getTrackbarPos("S_MIN",
    trackbarWindowName),
130                                  cv2.getTrackbarPos("V_MIN",
    trackbarWindowName)),
131                                 (cv2.getTrackbarPos("H_MAX",
    trackbarWindowName),
132                                  cv2.getTrackbarPos("S_MAX",
    trackbarWindowName),
133                                  cv2.getTrackbarPos("V_MAX",
    trackbarWindowName)))
134
135         if (useMorphOps):
136             threshold = morphOps(threshold)
137
138         if (trackObjects):
139             trackFilteredObject(x, y, threshold, cameraFeed)
140
141         cv2.imshow(windowName2, threshold)
142
143         teclou = cv2.waitKey(30) & 0xFF
144         if teclou == ord('q') or teclou == 27:  # se apertar q ou ESC
145             capture.release()
146             cv2.destroyAllWindows()
147             break
148
149 if __name__ == "__main__":
150     main()
```

Listing C.1 – Código Python de Detecção

APÊNDICE D – CÓDIGO PRINCIPAL DO FUZZY

```
import cv2
import numpy as np
import time
from motor_pwm_Fuzzy import test_stop, configure_motors, posicoes_x,
    posicoes_y
import RPi.GPIO as GPIO
from Controlador_Fuzzy import Fuzzy
import matplotlib.pyplot as plt

# Configuracao inicial
cap = cv2.VideoCapture(0)
time.sleep(3)
configure_motors()
status_esterio = "OFF"
epa = 0

# Loop principal
while True:
    # Captura o frame da camera
    ret, frame = cap.read()

    # Definicao do intervalo de cores a serem detectadas
    low_b = np.array([45, 49, 57])
    high_b = np.array([120, 254, 246])

    # Convertendo o frame para o espaco de cores HSV
    hsv_detectar = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv_detectar, low_b, high_b)

    # Aplicacao de erosao e dilatacao para melhorar a deteccao
    erodeElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    dilateElement = cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8))
    mask = cv2.erode(mask, erodeElement)
    mask = cv2.dilate(mask, dilateElement)

    # Encontrar contornos na mascara
    contours, hierarchy = cv2.findContours(mask, 1, cv2.
    CHAIN_APPROX_NONE)

    # Processamento dos contornos encontrados
    if len(contours) > 0:
        c = max(contours, key=cv2.contourArea)
        M = cv2.moments(c)
```

```
42
43          if M["m00"] != 0:
44              cx = int(M['m10'] / M['m00'])
45              cy = int(M['m01'] / M['m00'])
46              ep = 320 - cx
47              de = ep - epa
48              epa = ep
49
50              # Aplicacao do controlador fuzzy
51              Fuzzy(ep, de)
52
53              # Desenhar informacoes no frame
54              cv2.circle(frame, (cx, cy), 5, (255, 255, 255), -1)
55              cv2.line(frame, (cx - 75, 0), (cx - 75, frame.shape[0]),
    (200, 100, 150), 3)
56              cv2.line(frame, (cx + 75, 0), (cx + 75, frame.shape[0]),
    (200, 100, 150), 3)
57              approx = cv2.approxPolyDP(c, 0.02 * cv2.arcLength(c, True),
    True)
58              cv2.drawContours(frame, [approx], 0, (0, 0, 0), 5)
59          else:
60              print("Nao vejo a linha")
61              test_stop()
62      else:
63          print("Nao vejo a linha")
64          test_stop()
65
66      # Exibicao do frame original
67      cv2.imshow("Trajetoria Original", frame)
68
69      # Exibicao do frame Binarizado
70      #cv2.imshow("Trajetoria Binarizada",mask)
71      #cv2.imshow("Frame",frame)
72
73      # Verificacao da tecla 'q' para encerrar o programa
74      if cv2.waitKey(1) & 0xff == ord('q'):
75          test_stop()
76          break
77
78 # Limpeza e encerramento
79 GPIO.cleanup()
80 cap.release()
81 cv2.destroyAllWindows()
82
83 # Exibicao da trajetoria do robo
84 plt.plot(posicoes_x, posicoes_y, label='Trajetoria do Robo')
85 plt.xlabel('Posicao X (metros)')
```

```
86 plt.ylabel('Posicao Y (metros)')
87 plt.title('Trajetoria do Robo')
88 plt.legend()
89 plt.grid(True)
90 plt.show()
```

Listing D.1 – Código Python do main do Fuzzy

APÊNDICE E – CÓDIGO DO CONTROLADOR FUZZY

```
from matplotlib import pyplot as plt
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from motor_pwm_Fuzzy import test_pwm

# Cria as variaveis de entrada e saida
Erro = ctrl.Antecedent(np.arange(-320, 319, 1), 'Erro')
Desvio_Erro = ctrl.Antecedent(np.arange(-639, 639, 1), 'Desvio do Erro')
Roda_Esquerda = ctrl.Consequent(np.arange(0, 25.5, 0.1), 'Roda Esquerda'
    )
Roda_Direita = ctrl.Consequent(np.arange(0, 24.6, 0.1), 'Roda Direita')

# Define as funcoes de pertinencia para as variaveis de entrada e saida
Erro['Erro_Muito_Negativo'] = fuzz.trapmf(Erro.universe, [-320, -320,
    -250, -125])
Erro['Erro_Pouco_Negativo'] = fuzz.trimf(Erro.universe, [-255, -130,
    -5])
Erro['Erro_Zero'] = fuzz.trapmf(Erro.universe, [-125, -5, 5, 125])
Erro['Erro_Pouco_Positivo'] = fuzz.trimf(Erro.universe, [5, 130, 255])
Erro['Erro_Muito_Positivo'] = fuzz.trapmf(Erro.universe, [125, 250, 319,
     319])

Desvio_Erro['Desvio_Erro_Muito_Negativo'] = fuzz.trapmf(Desvio_Erro.
    universe, [-639, -639, -250, -125])
Desvio_Erro['Desvio_Erro_Pouco_Negativo'] = fuzz.trimf(Desvio_Erro.
    universe, [-255, -130, -5])
Desvio_Erro['Desvio_Erro_Zero'] = fuzz.trapmf(Desvio_Erro.universe,
    [-125, -5, 5, 125])
Desvio_Erro['Desvio_Erro_Pouco_Positivo'] = fuzz.trimf(Desvio_Erro.
    universe, [5, 130, 255])
Desvio_Erro['Desvio_Erro_Muito_Positivo'] = fuzz.trapmf(Desvio_Erro.
    universe, [125, 250, 639, 639])

Roda_Esquerda['VML'] = fuzz.trapmf(Roda_Esquerda.universe, [0, 0, 8.5,
    10.5])
Roda_Esquerda['VL'] = fuzz.trimf(Roda_Esquerda.universe, [8.5, 10.5,
    13.6])
Roda_Esquerda['VM'] = fuzz.trapmf(Roda_Esquerda.universe, [10.5, 13.6,
    15.6, 18.5])
Roda_Esquerda['VR'] = fuzz.trimf(Roda_Esquerda.universe, [15.6, 18.5,
    20.5])
Roda_Esquerda['VMR'] = fuzz.trapmf(Roda_Esquerda.universe, [18.5, 20.5,
```

```
    25.5, 25.5])
31
32 Roda_Direita['VML1'] = fuzz.trapmf(Roda_Direita.universe, [0, 0, 7.6,
    9.6])
33 Roda_Direita['VL1'] = fuzz.trimf(Roda_Direita.universe, [7.6, 9.6,
    12.6])
34 Roda_Direita['VM1'] = fuzz.trapmf(Roda_Direita.universe, [9.6, 12.8,
    14.8, 17.6])
35 Roda_Direita['VR1'] = fuzz.trimf(Roda_Direita.universe, [14.8, 17.6,
    19.6])
36 Roda_Direita['VMR1'] = fuzz.trapmf(Roda_Direita.universe, [17.6, 19.6,
    24.6, 24.6])
37
38 # Define as regras fuzzy
39 regra1 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['Desvio_Erro_Zero'],
    (Roda_Esquerda['VM'], Roda_Direita['VM1']))
40 regra2 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['
    Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VM'], Roda_Direita['VM1
    ']))
41 regra3 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['
    Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VM1
    ']))
42 regra4 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['
    Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VL'], Roda_Direita['VM1
    ']))
43 regra5 = ctrl.Rule(Erro['Erro_Zero'] & Desvio_Erro['
    Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VL1
    ']))
44 #-------------------------
45 regra6 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['
    Desvio_Erro_Zero'], (Roda_Esquerda['VL'], Roda_Direita['VM1']))
46 regra7 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VL'], Roda_Direita['VR1
    ']))
47 regra8 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['
    Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VML'], Roda_Direita['
    VR1']))
48 regra9 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VR1
    ']))
49 regra10 = ctrl.Rule(Erro['Erro_Pouco_Positivo'] & Desvio_Erro['
    Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VL'], Roda_Direita['VM1
    ']))
50 #-------------------------
51 regra11 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['
    Desvio_Erro_Zero'], (Roda_Esquerda['VM'], Roda_Direita['VL1']))
52 regra12 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['
```

```
    Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VR'], Roda_Direita['VL1
    '']))
regra13 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['
    Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VR'], Roda_Direita['
    VML1']))
regra14 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VR'], Roda_Direita['VM1
    ']))
regra15 = ctrl.Rule(Erro['Erro_Pouco_Negativo'] & Desvio_Erro['
    Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['VL1
    ']))
#-------------------------
regra16 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['
    Desvio_Erro_Zero'], (Roda_Esquerda['VML'], Roda_Direita['VM1']))
regra17 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VML'], Roda_Direita['
    VR1']))
regra18 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['
    Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VML'], Roda_Direita['
    VMR1']))
regra19 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VL'], Roda_Direita['
    VMR1']))
regra20 = ctrl.Rule(Erro['Erro_Muito_Positivo'] & Desvio_Erro['
    Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VM'], Roda_Direita['
    VMR1']))
#-------------------------
regra21 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['
    Desvio_Erro_Zero'], (Roda_Esquerda['VM'], Roda_Direita['VML1']))
regra22 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Negativo'], (Roda_Esquerda['VR'], Roda_Direita['
    VML1']))
regra23 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['
    Desvio_Erro_Muito_Negativo'], (Roda_Esquerda['VMR'], Roda_Direita['
    VM1']))
regra24 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['
    Desvio_Erro_Pouco_Positivo'], (Roda_Esquerda['VMR'], Roda_Direita['
    VL1']))
regra25 = ctrl.Rule(Erro['Erro_Muito_Negativo'] & Desvio_Erro['
    Desvio_Erro_Muito_Positivo'], (Roda_Esquerda['VMR'], Roda_Direita['
    VML1']))

# Cria o sistema de controle fuzzy
sistema_controle = ctrl.ControlSystem([regra1, regra2, regra3, regra4,
    regra5, regra6, regra7, regra8, regra9, regra10, regra11, regra12,
    regra13, regra14, regra15, regra16, regra17, regra18, regra19,
    regra20, regra21, regra22, regra23, regra24, regra25])
```

```
71 sistema = ctrl.ControlSystemSimulation(sistema_controle)
72
73 def Fuzzy(ep, de):
74     # Define os valores de entrada
75     sistema.input['Erro'] = ep
76     sistema.input['Desvio do Erro'] = de
77
78     # Computa o resultado
79     sistema.compute()
80
81     Esquerda = sistema.output['Roda Esquerda']
82     Direita = sistema.output['Roda Direita']
83
84     # Chama a funcao de controle PWM
85     test_pwm(Direita, Esquerda)
86
87     # Exibe as velocidades e parametros de erro
88     print("Velocidade Esquerda: " + str(Direita))
89     print("Velocidade Direita: " + str(Esquerda))
90
91     print("Erro: " + str(ep))
92     print("Desvio do Erro: " + str(de))
```

Listing E.1 – Código Python do Controlador Fuzzy

APÊNDICE F – CONTROLE DOS MOTORES UTILIZANDO FUZZY

```
import RPi.GPIO as GPIO
import numpy as np
import math
import matplotlib.pyplot as plt

# Par metros das rodas
distancia_entre_rodas = 0.16  # Distancia entre as rodas do robo
pulsos_por_rotacao = 470  # Numero de pulsos por rotacao dos encoders
raio_roda = 0.0325  # Raio da roda do robo

# Vari veis globais para a odometria
x = 0.0  # Posicao x do robo no plano
y = 0.0  # Posicao y do robo no plano
theta = 0.0  # Orientacao do robo
posicoes_x = []  # Lista para armazenar as posicoes x ao longo do tempo
posicoes_y = []  # Lista para armazenar as posicoes y ao longo do tempo

# Configuracoes iniciais do primeiro motor
MOTOR1_PWM_PIN = 18
MOTOR1_IN1_PIN = 23
MOTOR1_IN2_PIN = 24

# Configuracoes iniciais do segundo motor
MOTOR2_PWM_PIN = 13
MOTOR2_IN1_PIN = 7
MOTOR2_IN2_PIN = 8

# Configuracao dos pinos do encoder
pin_encoder1 = 27  # Pino de saida do encoder
pin_encoder2 = 17  # Pino de saida do encoder
pin_encoder3 = 5  # Pino de saida do encoder
pin_encoder4 = 6  # Pino de saida do encoder

# Variaveis para armazenar a contagem dos encoders
encoder_count1 = 0
encoder_count2 = 0
last_encoder_count1 = 0  # Contagem do encoder da roda esquerda
last_encoder_count2 = 0  # Contagem do encoder da roda direita

# Configuracao da GPIO
GPIO.setmode(GPIO.BCM)

```

```
43 # Estados anteriores dos sinais dos encoders
44 GPIO.setup(pin_encoder1, GPIO.IN)
45 GPIO.setup(pin_encoder2, GPIO.IN)
46 GPIO.setup(pin_encoder3, GPIO.IN)
47 GPIO.setup(pin_encoder4, GPIO.IN)
48
49 # Configuracao do primeiro motor
50 GPIO.setup(MOTOR1_PWM_PIN, GPIO.OUT)
51 GPIO.setup(MOTOR1_IN1_PIN, GPIO.OUT)
52 GPIO.setup(MOTOR1_IN2_PIN, GPIO.OUT)
53
54 # Configuracao do segundo motor
55 GPIO.setup(MOTOR2_PWM_PIN, GPIO.OUT)
56 GPIO.setup(MOTOR2_IN1_PIN, GPIO.OUT)
57 GPIO.setup(MOTOR2_IN2_PIN, GPIO.OUT)
58
59 # Estados anteriores dos sinais dos encoders
60 last_state_A1 = GPIO.input(pin_encoder1)
61 last_state_B1 = GPIO.input(pin_encoder2)
62 last_state_A2 = GPIO.input(pin_encoder3)  # Pino para o segundo encoder
      A
63 last_state_B2 = GPIO.input(pin_encoder4)  # Pino para o segundo encoder
      B
64
65 # Configuracao dos pinos dos encoders como entrada com pull-down
66 GPIO.setup(pin_encoder1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
67 GPIO.setup(pin_encoder2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
68 GPIO.setup(pin_encoder3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
69 GPIO.setup(pin_encoder4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
70
71 # Estados anteriores dos sinais dos encoders
72 Direcao1 = True
73 Direcao2 = True
74
75 # Configuracao do objeto PWM para o primeiro motor
76 motor1_pwm = GPIO.PWM(MOTOR1_PWM_PIN, 100)  # Frequencia de PWM de 1 kHz
77 motor1_pwm.start(0)  # Inicia com um ciclo de trabalho de 0 (motor
      parado)
78
79 # Configuracao do objeto PWM para o segundo motor
80 motor2_pwm = GPIO.PWM(MOTOR2_PWM_PIN, 100)  # Frequencia de PWM de 1 kHz
81 motor2_pwm.start(0)  # Inicia com um ciclo de trabalho de 0 (motor
      parado)
82
83 # Funcao para configurar os motores
84 def configure_motors():
85     GPIO.output(MOTOR1_IN1_PIN, GPIO.HIGH)
```

```
86      GPIO.output(MOTOR1_IN2_PIN, GPIO.LOW)
87      GPIO.output(MOTOR2_IN1_PIN, GPIO.HIGH)
88      GPIO.output(MOTOR2_IN2_PIN, GPIO.LOW)
89
90
91  def test_pwm(valor_Esquerda, valor_Direita):
92      motor1_pwm.ChangeDutyCycle(valor_Esquerda)
93      motor2_pwm.ChangeDutyCycle(valor_Direita)
94   #    time.sleep(0.1)
95
96  def test_stop():
97      # Motor 1
98          motor1_pwm.ChangeDutyCycle(0)
99
100     # Motor 2
101         motor2_pwm.ChangeDutyCycle(0)
102
103 # Funcao para calcular a odometria
104 def calcular_odometria(encoder_count1, encoder_count2):
105     global x, y, theta, last_encoder_count1, last_encoder_count2,
        raio_roda, pulsos_por_rotacao
106
107     # Calcular a diferenca nas contagens dos encoders
108     delta_encoder1 = encoder_count1 - last_encoder_count1
109     delta_encoder2 = encoder_count2 - last_encoder_count2
110
111     # Atualizar as contagens dos encoders
112     last_encoder_count1 = encoder_count1
113     last_encoder_count2 = encoder_count2
114
115     # Calcular as distancias percorridas pelas rodas
116     distancia_roda1 = (2 * math.pi * raio_roda * delta_encoder1) /
        pulsos_por_rotacao
117     distancia_roda2 = (2 * math.pi * raio_roda * delta_encoder2) /
        pulsos_por_rotacao
118
119     # Calcular a distancia total percorrida
120     distancia_total = (distancia_roda1 + distancia_roda2) / 2.0
121
122     # Calcular a variacao na orientacao do robo (theta)
123     delta_theta = (distancia_roda1 - distancia_roda2) /
        distancia_entre_rodas
124
125     # Atualizar a posicao do robo
126     x += distancia_total * math.cos(theta + delta_theta / 2.0)
127     y += distancia_total * math.sin(theta + delta_theta / 2.0)
128     theta += delta_theta
```

```
129
130     # Atualizar as listas de posicao
131     posicoes_x.append(x)
132     posicoes_y.append(y)
133
134     # Variaveis para armazenar as contagens dos encoders anteriores
135     last_encoder_count1 = encoder_count1
136     last_encoder_count2 = encoder_count2
137
138 # Funcao para ser chamada na borda de subida do encoder1
139 def encoder1_callback(channel):
140     global encoder_count1, Direcao1, last_state_A1
141     state_A1 = GPIO.input(pin_encoder1)
142     state_B1 = GPIO.input(pin_encoder2)
143
144     if last_state_A1 == GPIO.LOW and state_A1 == GPIO.HIGH:
145         val = state_B1
146         if val == GPIO.LOW and Direcao1:
147             Direcao1 = False  # Reverse
148         elif val == GPIO.HIGH and not Direcao1:
149             Direcao1 = True  # Forward
150
151     last_state_A1 = state_A1
152
153     if not Direcao1:
154         encoder_count1 += 1
155     else:
156         encoder_count1 -= 1
157
158     print("Encoder 2 Count:", encoder_count1)
159     # Atualizar a odometria
160     calcular_odometria(encoder_count1, encoder_count2)
161
162 # Funcao para ser chamada na borda de subida do encoder2
163 def encoder2_callback(channel):
164     global encoder_count2, Direcao2, last_state_A2
165
166     state_A2 = GPIO.input(pin_encoder3)  # Pino para o segundo encoder A
167     state_B2 = GPIO.input(pin_encoder4)  # Pino para o segundo encoder B
168
169     if last_state_A2 == GPIO.LOW and state_A2 == GPIO.HIGH:
170         val = state_B2
171         if val == GPIO.LOW and Direcao2:
172             Direcao2 = False  # Reverse
173         elif val == GPIO.HIGH and not Direcao2:
174             Direcao2 = True  # Forward
175
```

```
176     last_state_A2 = state_A2
177 
178     if not Direcao2:
179         encoder_count2 += 1
180     else:
181         encoder_count2 -= 1
182 
183     print("Encoder 1 Count:", encoder_count2)
184     # Atualizar a odometria
185     calcular_odometria(encoder_count1, encoder_count2)
186 
187 # Adicionar deteccao de eventos para os pulsos dos encoders
188 GPIO.add_event_detect(pin_encoder1, GPIO.BOTH, callback=
    encoder1_callback)
189 GPIO.add_event_detect(pin_encoder3, GPIO.BOTH, callback=
    encoder2_callback)
```

Listing F.1 – Código Python do Controle dos Motores utilizando Fuzzy