



MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TOCANTINS
CAMPUS PALMAS
BACHARELADO EM ENGENHARIA ELÉTRICA

Anderson José Bispo Lino
Matheus Miguel Alves Cunha

VISÃO COMPUTACIONAL APLICADA NA AUTOMAÇÃO INDUSTRIAL

Palmas – TO

2026

VISÃO COMPUTACIONAL APLICADA NA AUTOMAÇÃO INDUSTRIAL

Trabalho de Conclusão de Curso
apresentado à Coordenação do Curso
Superior de Bacharelado em Engenharia
Elétrica do Instituto Federal de Educação,
Ciência e Tecnologia do Tocantins Campus
Palmas, como requisito para obtenção do
título de Bacharel em Engenharia Elétrica.

Orientador: Dr. Márcio Augusto Tamashiro

Coorientador: Dr. Wendell E. Moura Costa

Banca Examinadora:

Prof. Me. Kaison Teodoro de Souza

Prof. Me. Mateus Lima Peduzzi

Dados Internacionais de Catalogação na Publicação (CIP)
Bibliotecas do Instituto Federal do Tocantins

L735v Lino, Anderson José Bispo
VISÃO COMPUTACIONAL APLICADA NA AUTOMAÇÃO
INDUSTRIAL / Anderson José Bispo Lino, Matheus Miguel Alves
Cunha. – Palmas, TO, 2026.
117 f. : il. color.

Trabalho de Conclusão de Curso (Bacharelado em Engenharia
Elétrica) – Instituto Federal de Educação, Ciência e Tecnologia do
Tocantins, Campus Palmas, Palmas, TO, 2026.

Orientador: Dr. Márcio Augusto Tamashiro
Coorientador: Dr. Wendell Eduardo Moura Costa

1. Visão Computacional. 2. Automação Industrial. 3. K-Nearest
Neighbors (KNN). I. Cunha, Matheus Miguel Alves. II. Tamashiro,
Márcio Augusto III. Moura Costa, Wendell Eduardo. IV. Título.

CDD 621

A reprodução total ou parcial, de qualquer forma ou por qualquer meio, deste documento é autorizada para fins
de estudo e pesquisa, desde que citada a fonte.

Elaborado pelo sistema de geração automática de ficha catalográfica do IFTO com os dados fornecidos
pelo(a) autor(a).

Anderson José Bispo Lino
Matheus Miguel Alves Cunha

VISÃO COMPUTACIONAL APLICADA NA AUTOMAÇÃO INDUSTRIAL

Trabalho de conclusão de curso (TCC),
apresentado ao Instituto Federal de
Educação, Ciência e Tecnologia do
Tocantins (IFTO), como parte das
exigências para a obtenção do título de
bacharel em engenharia elétrica.

Aprovado em: 06/ 04/ 2026.

BANCA EXAMINADORA

Documento assinado digitalmente
 **MARCIO AUGUSTO TAMASHIRO**
Data: 07/04/2026 22:25:35-0300
Verifique em <https://validar.iti.gov.br>

Dr. Márcio Augusto Tamashiro (Orientador) IFTO – Campus Palmas

Documento assinado digitalmente
 **WENDELL EDUARDO MOURA COSTA**
Data: 08/04/2026 12:20:24-0300
Verifique em <https://validar.iti.gov.br>

Dr. Wendell Eduardo Moura Costa (Coorientador) IFTO – Campus Palmas

Documento assinado digitalmente
 **KAISSON TEODORO DE SOUZA**
Data: 08/04/2026 08:46:28-0300
Verifique em <https://validar.iti.gov.br>

Me. Kaison Teodoro de Souza IFTO – Campus Palmas

Documento assinado digitalmente
 **MATEUS LIMA PEDUZZI**
Data: 08/04/2026 08:15:56-0300
Verifique em <https://validar.iti.gov.br>

Me. Mateus Lima Peduzzi IFTO – Campus Palmas

AGRADECIMENTOS

Expressamos nossa sincera gratidão a Deus, cujo amparo e proteção foram o alicerce de nossa trajetória acadêmica, concedendo-nos a força e a sabedoria necessárias para superar os desafios deste percurso.

À nossa família, pelo apoio integral, paciência e incentivo constante. Sem o suporte emocional e a compreensão de vocês, a conclusão desta etapa não seria possível.

Ao nosso orientador, Prof. Dr. Márcio Augusto Tamashiro, pela dedicação, pelos conselhos precisos e pela confiança depositada. Suas valiosas contribuições e feedbacks foram fundamentais para o desenvolvimento desta pesquisa.

Ao coorientador, Prof. Dr. Wendell E. Moura Costa, pelas ideias inovadoras e disponibilidade técnica que enriqueceram significativamente este estudo.

À Dra. Vanessa Marques, pelo auxílio imprescindível na formatação textual e no refinamento da escrita deste artigo, garantindo clareza e rigor ao trabalho.

Aos membros da banca examinadora, Prof. Me. Kaison Teodoro de Souza e Prof. Me. Mateus Lima Peduzzi, pela disponibilidade e pelas sugestões que contribuíram para o aprimoramento deste projeto.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO), Campus Palmas, pela estrutura e pelo ambiente acadêmico de excelência que viabilizaram esta conquista. Nossa gratidão ao corpo docente pela dedicação à formação pública de qualidade.

Por fim, agradecemos a todos os amigos e professores que, direta ou indiretamente, colaboraram com palavras de incentivo e conhecimentos compartilhados.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema integrado de visão computacional e automação industrial, aplicado ao controle de fluxo em uma linha de produção simulada em bancada didática. O objetivo principal consistiu em desenvolver um sistema capaz de identificar a presença de uma peça de teste vermelha por meio de técnicas de processamento digital de imagens e, com base nessa identificação, acionar um Controlador Lógico Programável (CLP) da Festo para realizar a aceitação ou rejeição automática do produto. A implementação utilizou a biblioteca OpenCV para processamento visual e o algoritmo K-Nearest Neighbors (KNN) para classificação, estabelecendo comunicação em rede HTTP entre o módulo de visão computacional e o CLP. O sistema foi desenvolvido em Python e testado em uma bancada didática equipada com esteira e atuador pneumático. Os resultados demonstraram a viabilidade da integração proposta, com tempos de resposta compatíveis com operações em tempo real e acurácia satisfatória nas condições controladas. O trabalho contribui para a demonstração prática de como tecnologias de visão computacional podem ser integradas a sistemas de automação industrial convencionais, oferecendo uma solução de baixo custo e escalável para aplicações de inspeção visual automatizada.

Palavras-chave: Visão Computacional; Automação Industrial; OpenCV; KNN; CLP Festo.

ABSTRACT

This work presents the development of an integrated computer vision and industrial automation system, applied to flow control in a simulated production line on a didactic bench. The main objective was to develop a system capable of identifying the presence of a red test piece through digital image processing techniques and based on this identification, triggering a Festo Programmable Logic Controller (PLC) to perform automatic acceptance or rejection of the product. The implementation used the OpenCV library for visual processing and the K-Nearest Neighbors (KNN) algorithm for classification, establishing HTTP network communication between the computer vision module and the PLC. The system was developed in Python and tested on a didactic bench equipped with a conveyor belt and pneumatic actuator. The results demonstrated the feasibility of the proposed integration, with response times compatible with real-time operations and satisfactory accuracy under controlled conditions. This work contributes to the practical demonstration of how computer vision technologies can be integrated into conventional industrial automation systems, offering a low-cost and scalable solution for automated visual inspection applications.

Keywords: Computer Vision; Industrial Automation; OpenCV; KNN; Festo PLC.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura do sistema de visão computacional e automação.....	12
Figura 2 - Diagrama estrutural de um sistema de visão computacional.....	16
Figura 3 - Exemplos de classificação, localização e detecção de objetos.....	18
Figura 4 - Aplicação do método de conversão de cores.....	19
Figura 5 - Bibliotecas de processamento paralelo baseadas em Python.....	26
Figura 6 - O ambiente IDLE.....	28
Figura 7 - O poema Zen.....	29
Figura 8 - Estrutura básica de um degrau.....	30
Figura 9 - Elementos de ligação da linguagem Ladder.....	31
Figura 10 - Exemplo de pirâmide de imagem.....	32
Figura 11 - Diagrama fluxograma da estrutura experimental.....	45
Figura 12 - Webcam Anker C 200 (2K).....	48
Figura 13 - Samsung Galaxy Book2.....	53
Figura 14 - Bancada eletropneumática Festo F640 (IFTO Campus Palmas).....	60
Figura 15 - Peça de teste vermelha utilizada como objeto-alvo.....	61
Figura 16 - Objeto preto não detectado.....	104
Figura 17 - Objeto prata não detectado.....	104
Figura 18 - Objeto vermelho detectado.....	105
Figura 19 - Câmera Anker C200 utilizada na aquisição de imagens da bancada.....	105
Figura 20 - Medidor pneumático em operação.....	106
Figura 21 - Conjunto de peças utilizadas nos testes da bancada (preto, prata e vermelho).	106
Figura 22 - Bancada Festo FC640 em funcionamento com suas conexões integradas.	107
Figura 23 - Diagrama de sequência da integração entre sistema de visão computacional e CLP Festo FC640.....	115
Figura 24 - Bancada Festo FC640 pronta para operação.....	116
Figura 25 - Bancada Festo FC640 em operação durante os testes.....	117

LISTA DE QUADROS

Quadro 1 - Técnica de aumento de dados para robustez.....	39
Quadro 2 - Módulos e funções utilizadas no projeto.....	49
Quadro 3 - Especificações técnicas:.....	53
Quadro 4 - Configuração Técnica.....	54
Quadro 5 - Comandos de integração com o CLP via HTTP.....	64
Quadro 6 - Justificativa para inclusão das métricas:.....	70
Quadro 7 - Peças, cores e funções na bancada experimental.....	107

LISTA DE TABELAS

Tabela 1 - Conjunto de dados.....	62
Tabela 2 - Matriz de confusão.....	70
Tabela 3 - Análise das métricas incluídas:.....	70
Tabela 4 - Métricas de desempenho obtidas.....	71
Tabela 5 - Análise do trade-off estabilidade × latência.	72
Tabela 6 - As limitações apontadas são pertinentes e realistas:.....	73

LISTA DE ABREVIATURAS E SIGLAS

TIC - Tecnologias de Informação e Comunicação

CPS - Cyber Physical Systems

IoT- Internet das Coisas

IIoT - Industrial Internet of Things

IoS - Internet dos Serviços

M2M - Comunicação Máquina à Máquina

ML - Machine Learning

BB - Bording Boxes

AI - Artificial Intelligence

IA – Inteligência Artificial

DL - Deep Learning

PLN - Planejamento de Linguagem Natural

RNC - Rede Neural Convolucional

PADI - Processamento e Análise Digital de Imagens

ANNs - Redes Neurais Artificiais

CNNs - Redes Neurais Convolucionais

CLP - Controlador Lógico Programável

MPS – Sistema Modular de Produção

SAI – System Artificial Intelligence

BGR – Blue, Green and Red

HSV – Hue, Saturation and Value

SUMÁRIO

1 INTRODUÇÃO.....	12
1.1 CONTEXTUALIZAÇÃO.....	13
1.2 JUSTIFICATIVA.....	13
1.3 OBJETIVOS.....	14
1.3.1 Objetivo geral.....	14
1.3.2 Objetivos específicos.....	14
2 REFERENCIAL TEÓRICO.....	15
2.1 A INDÚSTRIA 4.0.....	15
2.1.1 Internet industrial das coisas (IIOT).....	16
2.2 INSPEÇÃO VISUAL.....	17
2.3 CONEXÃO ENTRE CATEGORIZAÇÃO DE OBJETOS E CONVERSÃO DE CORES.....	17
2.4 INTELIGÊNCIA ARTIFICIAL.....	19
2.5 APRENDIZADO DE MÁQUINA.....	21
2.5.1 K- Nearest Neighbors (KNN).....	22
2.6 VISÃO COMPUTACIONAL.....	24
2.7 LINGUAGEM DE PROGRAMAÇÃO PYTHON.....	25
2.7.1 Recursos Utilizados.....	26
2.7.2 IDLE Python.....	27
2.7.3 Características da linguagem python.....	28
2.8 LINGUAGEM DE PROGRAMAÇÃO LADDER.....	29
2.8.1 Trilhos de energia.....	30
2.8.2 Elementos e estados das ligações.....	30
2.8.3 Opencv.....	31
2.8.4 Pirâmide de imagens e janela deslizante.....	32
2.8.4.1 Relação com o KNN.....	33
2.8.5 Relação entre limiarização de Otsu e k-Nearest Neighbors (KNN).....	34
2.8.5.1 Interface técnica entre segmentação e classificação.....	34
2.9 INTEGRAÇÃO COM AUTOMAÇÃO INDUSTRIAL E CLP.....	35
2.9.1 Generalização e robustez do classificador k-Nearest Neighbors KNN em contexto industrial.....	37
2.9.1.1 Desafios de generalização com k-Nearest Neighbors KNN.....	37
2.9.2 Técnicas de robustez para k-Nearest Neighbors KNN.....	38
2.9.3 Avaliação de desempenho além da acurácia com conjuntos desbalanceado.....	39
2.9.4 Otimização de hiperparâmetros para robustez.....	40
2.9.5 Considerações sobre tempo de resposta.....	41
2.9.6 Abordagem implementada neste trabalho.....	42
2.9.7 Conclusão sobre a aplicação do k-Nearest Neighbors KNN.....	42
3 METODOLOGIA.....	44
3.1 MATERIAIS E MÉTODOS.....	44
3.1.1 Fase 1: aquisição e rotulagem dos dados.....	46

3.1.2 Fase 2: Pipeline de Treinamento (Engenharia de Características e Aprendizado do Modelo)	46
3.2 EQUIPAMENTOS UTILIZADOS	48
3.2.1 Câmera Digital	48
3.2.2 OpenCV – Biblioteca de Visão Computacional	49
3.2.2.1 Exemplo de código – Aquisição de imagem:	49
3.2.2.2 Exemplo de código – Pré-processamento:	50
3.2.2.3 Integração com outras bibliotecas:	50
3.2.2.4 Configuração do ambiente:	50
3.2.2.5 Limitações encontradas:	50
3.2.2.6 Desempenho obtido:	51
3.2.3 Notebook	53
3.2.4 CLP Festo FC 640 – Especificações e Aplicação	54
3.2.4.1 Função no Sistema	54
3.2.4.2 Configuração de Comunicação	55
3.2.4.3 Conexões na Bancada	56
3.2.4.4 Programação Lógica	57
3.2.4.5 Vantagens para o Projeto	58
3.2.4.6 Exemplo de comando em Python:	60
3.3 OBJETO-ALVO E CLASSES	61
3.4 CONJUNTO DE DADOS	61
3.5 AQUISIÇÃO E ROTULAGEM	62
3.6 EXTRAÇÃO DE CARACTERÍSTICAS	62
3.7 TREINAMENTO E VALIDAÇÃO	62
3.8 CLASSIFICAÇÃO EM TEMPO REAL E LÓGICA DE DECISÃO	63
3.9 INTEGRAÇÃO COM CLP VIA HTTP	63
4 DESENVOLVIMENTO DO SOFTWARE E DISCUSSÕES	64
4.1 ARQUITETURA DE SOFTWARE E DETALHAMENTO DA IMPLEMENTAÇÃO COMPUTACIONAL	64
4.2 MÉTRICAS E MATRIZ DE CONFUSÃO	69
4.3 DESEMPENHO EM TEMPO REAL	71
4.4 LIMITAÇÕES IDENTIFICADAS	73
4.4.1 Conexão com a fundamentação teórica:	74
4.5 RECOMENDAÇÕES DE MELHORIA	74
4.5.1 Primeiro eixo: aprimoramento do modelo de visão computacional	74
4.5.2 Segundo eixo: robustez da integração e operação industrial	75
4.5.3 Terceiro eixo: evolução da arquitetura do sistema	76
4.5.4 Considerações	76
5 CONSIDERAÇÕES FINAIS	77
5.1 PRINCIPAIS CONTRIBUIÇÕES TÉCNICAS DO KNN	78
5.2 LIMITAÇÕES ESPECÍFICAS DO KNN IDENTIFICADAS	79
5.3 IMPLICAÇÕES PEDAGÓGICAS E TÉCNICAS	79
5.4 PARA TRABALHOS FUTUROS, SUGERE-SE	80
REFERÊNCIAS BIBLIOGRÁFICAS	82
APÊNDICE A – CÓDIGO-FONTE (MAIN.PY)	86
APÊNDICE B – GUIA DE OPERAÇÃO E INTEGRAÇÃO COM CLP	102

1 INTRODUÇÃO

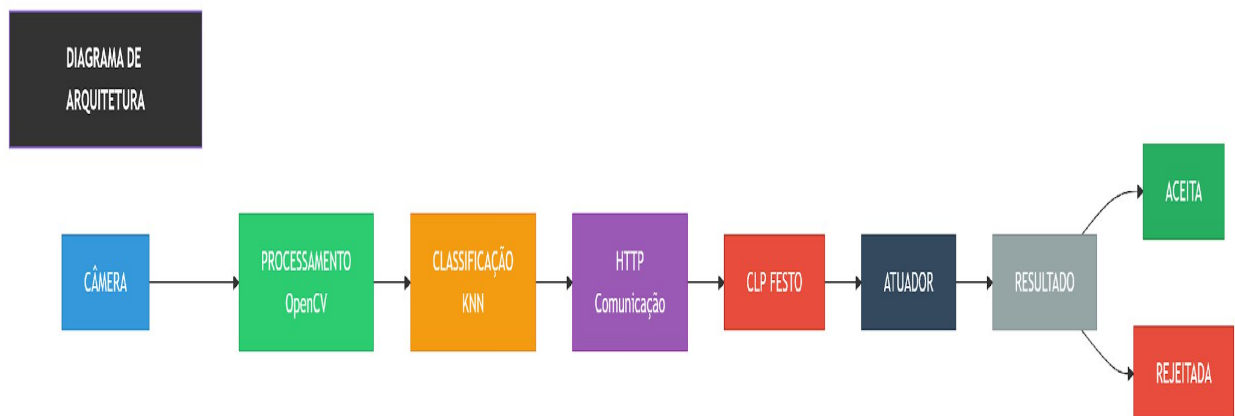
A quarta revolução industrial tem impulsionado a adoção de tecnologias digitais para elevar a produtividade, a rastreabilidade e a qualidade, reduzindo desperdícios e variabilidades nos processos. Nesse contexto, os sistemas de inspeção visual automatizada apresentam-se como uma alternativa eficaz à inspeção manual, oferecendo repetibilidade e capacidade de operação em altas cadências.

Este trabalho propõe um protótipo para inspeção e rejeição automática de produtos, utilizando visão computacional e automação pneumática. O sistema foi implementado e testado em um notebook Samsung Book 2 i7 com uma câmera Anker C 200 2K.

O objetivo principal foi identificar a presença de uma peça de teste vermelha em uma bancada didática com esteiras e atuar, por meio de um Controlador Lógico Programável (CLP) da Festo, para realizar a rejeição ou o controle do fluxo, demonstrando a viabilidade da integração entre visão computacional e automação industrial.

A arquitetura do sistema é representada pelo diagrama da Figura 1, que ilustra a interação entre os módulos de aquisição de imagem, processamento, classificação e atuação.

Figura 1 - Arquitetura do sistema de visão computacional e automação.



Fonte: Elaborada pelos autores (2026).

Essa arquitetura segue uma sequência linear composta por: (1) aquisição de imagem, com câmera USB; (2) processamento e classificação, realizados em Python com OpenCV e algoritmo KNN; (3) comunicação via rede HTTP; e (4) automação industrial, com o CLP Festo controlando esteira e atuadores pneumáticos. A decisão binária (aceitar/rejeitar) é baseada na classificação visual.

Durante a implementação, foram desenvolvidos *scripts* em Python para captura e processamento de imagens, extração de características de cor (histogramas no espaço HSV) e classificação via KNN. A comunicação com o CLP foi realizada por meio de requisições HTTP, permitindo o acionamento de atuadores pneumáticos com base no resultado da classificação. Os testes avaliaram a acurácia do classificador, o tempo de resposta do sistema e a robustez da integração.

Os resultados obtidos indicam que o sistema é viável para aplicações de inspeção visual em tempo real, com margens de acerto satisfatórias em condições controladas. As principais limitações identificadas relacionam-se à iluminação variável e à necessidade de calibração do classificador para diferentes cenários.

1.1 CONTEXTUALIZAÇÃO

A inspeção visual manual, ainda comum em muitas indústrias, está sujeita a erros humanos decorrentes de fadiga, desatenção ou variação de critérios. A automação dessa atividade por meio de sistemas de visão computacional oferece maior consistência, velocidade e confiabilidade. Neste trabalho, emprega-se a biblioteca OpenCV para processamento digital de imagens e o algoritmo KNN para classificação, integrados a um CLP Festo que comanda os atuadores da bancada de testes.

1.2 JUSTIFICATIVA

A necessidade de aumentar a eficiência e a qualidade na produção industrial justifica o investimento em sistemas automatizados de inspeção. Este projeto demonstra, de forma prática e acessível, como tecnologias de visão computacional e

aprendizado de máquina podem ser integradas a dispositivos de automação consolidados, como CLPs, para criar soluções de baixo custo alinhadas aos princípios da Indústria 4.0.

1.3 OBJETIVOS

1.3.1 Objetivo geral

Desenvolver e avaliar um sistema de visão computacional para inspeção automatizada em bancada didática, integrando classificação binária de imagens (presença/ausência de peça vermelha) com atuação via CLP.

1.3.2 Objetivos específicos

- Coletar e organizar um banco de imagens da peça-alvo (vermelha) em diferentes condições de iluminação e posicionamento;
- Implementar um *pipeline* de processamento de imagens com OpenCV, incluindo conversão para o espaço HSV e extração de histogramas de cor;
- Treinar e validar um classificador KNN para identificar a presença da peça vermelha na região de interesse (ROI);
- Integrar o resultado da classificação ao CLP Festo FC640 via comunicação HTTP para acionamento dos atuadores pneumáticos;
- Avaliar o desempenho do sistema por meio de métricas como acurácia, precisão, recall e tempo de resposta;
- Comparar os resultados obtidos com estudos semelhantes da literatura, identificando contribuições e limitações da abordagem proposta.

2 REFERENCIAL TEÓRICO

2.1 A INDÚSTRIA 4.0

A quarta revolução industrial, consolidada sob o conceito de Indústria 4.0, constitui um marco transformador na produção industrial. Este paradigma resulta do crescimento exponencial das Tecnologias da Informação e Comunicação (TIC) e da contínua adoção de inovações tecnológicas pelo setor industrial. A Indústria 4.0 integra sistemas físicos, digitais e biológicos, estabelecendo redes de produção inteligente onde os componentes interagem e colaboram de forma sinérgica (ROZO-GARCÍA, 2020).

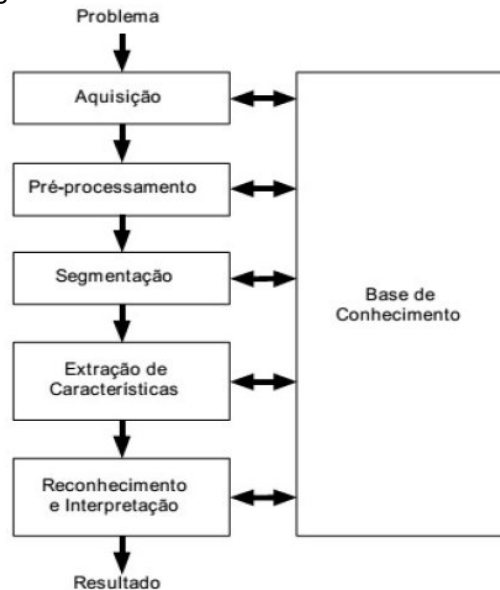
Conforme Hermann, Pentek e Otto (2015), as tecnologias centrais da Indústria 4.0 podem ser agrupadas em quatro categorias principais: sistemas ciberfísicos (CPS), Internet das Coisas (IoT), Internet dos Serviços (IoS) e fábricas inteligentes. Outras tecnologias, como comunicação máquina a máquina (M2M), *big data* e computação em nuvem, embora preexistentes, assumem um papel fundamental como facilitadoras e integradoras neste novo cenário.

O Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE, 2018) caracteriza a Indústria 4.0 pela integração de máquinas inteligentes, análises computacionais avançadas e pessoas conectadas, visando maior eficiência em toda a cadeia produtiva. Esta abordagem se destaca pela descentralização do controle dos processos, facilitando a interação entre os domínios físico, biológico e digital. Desse modo, são desenvolvidos sistemas operacionais para dispositivos que se comunicam e operam de forma autônoma, com capacidade de autorregulação e antecipação a falhas, reduzindo perdas produtivas.

Nesse contexto de automação e controle, os sistemas de inspeção visual automatizada emergem como uma tecnologia crítica. Eles substituem a inspeção manual, oferecendo repetibilidade, altas cadências de operação e integração direta com os sistemas de controle produtivo, alinhando-se plenamente aos princípios da Indústria 4.0. Conforme o modelo proposto por Marques Filho e Vieira Neto (1999), um sistema de visão computacional pode ser genericamente estruturado em estágios

funcionais sequenciais, que abrangem desde a aquisição da imagem até a tomada de decisão. A Figura 2 ilustra essa arquitetura básica.

Figura 2 - Diagrama estrutural de um sistema de visão computacional.



Fonte: Marques Filho e Vieira Neto (1999).

A Indústria 4.0 tem como meta reunir e analisar informações das máquinas, tornando os processos cada vez mais rápidos e eficazes, resultando em uma produção de maior qualidade a um custo menor. Assim, os avanços nas tecnologias da informação e na engenharia são fundamentais para seu progresso. Observa-se que existe uma conexão entre as diferentes fases do desenvolvimento de um produto, integrando tecnologias digitais e físicas (RODRIGUES et al., 2022).

Conforme mencionado por Coelho (2016), a Indústria 4.0 prioriza aspectos como melhoria contínua, eficiência, segurança, produtividade e, principalmente, o retorno sobre o investimento. A utilização de tecnologias que facilitam esses processos já é uma realidade.

2.1.1 Internet industrial das coisas (IIOT)

A Internet das Coisas Industrial (IIoT), do inglês *Industrial Internet of Things* (DIAS, 2021), ganhou significativa relevância nos últimos anos. Esse conceito envolve

o uso de redes de sensores interligados à Internet para capturar informações em tempo real das máquinas presentes no ambiente fabril. A implementação dessa tecnologia nas indústrias traz diversos benefícios, como um aprimoramento no monitoramento dos processos industriais (WANG et al., 2020) e a possibilidade de acompanhamento remoto por parte do gestor técnico (AL-TURJMAN, 2018).

Para Fernandes et al. (2022), a digitalização da indústria é viabilizada pela utilização das ferramentas da IIoT. Essa abordagem está se consolidando no mercado devido à sua capacidade de reproduzir virtualmente o funcionamento de máquinas (MINERVA, 2020). Dessa forma, a metodologia de manutenção preditiva avança para um novo patamar de excelência, já que as análises podem ser realizadas em tempo real e de qualquer lugar (DOS SANTOS et al., 2022).

2.2 INSPEÇÃO VISUAL

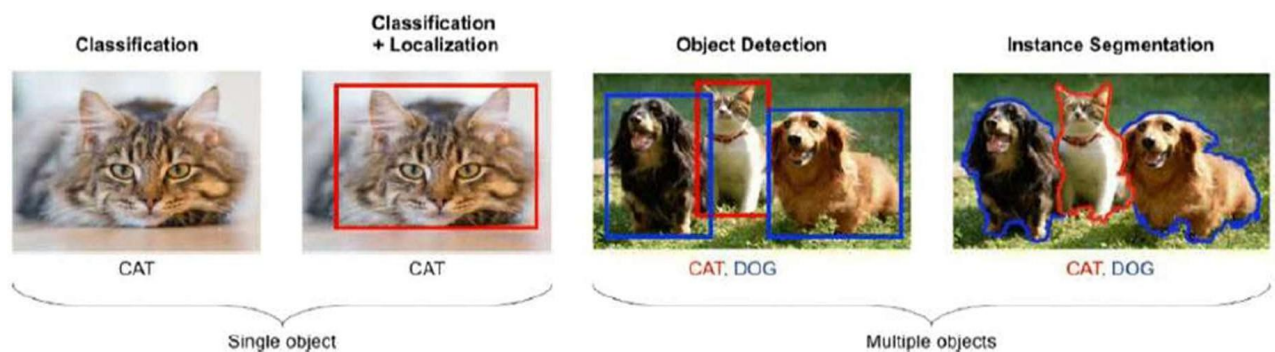
Um dos principais campos de aplicação dos paradigmas da Indústria 4.0 é a automação de tarefas que antes eram executadas por operadores humanos, como é o caso das atividades de inspeção visual. A maioria dos produtos fabricados na indústria passa por uma fase de inspeção visual. Essa tarefa pode ser executada por um operador especializado, que possui o treinamento necessário, ou por operadores comuns. No entanto, desde os anos 1970, a implementação de sistemas automáticos de inspeção visual tem se espalhado por diversos setores da indústria (BEYRER et al., 2015).

2.3 CONEXÃO ENTRE CATEGORIZAÇÃO DE OBJETOS E CONVERSÃO DE CORES

A categorização de objetos envolve a definição de classes para um ou mais objetos presentes em uma imagem, além de atribuir a eles seus respectivos rótulos. O processo de localização de objetos refere-se à identificação da posição de um ou mais objetos em uma imagem ou vídeo por meio de caixas delimitadoras, conhecidas como *bounding boxes*. A junção dos processos de localização e categorização é

denominada detecção de objetos. Um fluxo completo de detecção de objetos abrange a recepção de uma imagem como *input*, a identificação dos objetos, a atribuição de rótulos conforme a classe correspondente e a apresentação da probabilidade de classe do objeto identificado, geralmente acompanhada pela delimitação da área em que o objeto se encontra (KAUR; SINGH, 2022). A Figura 3 ilustra a interconexão entre localização, categorização e detecção de objetos.

Figura 3 - Exemplos de classificação, localização e detecção de objetos.



Fonte: Kaur; Singh (2022).

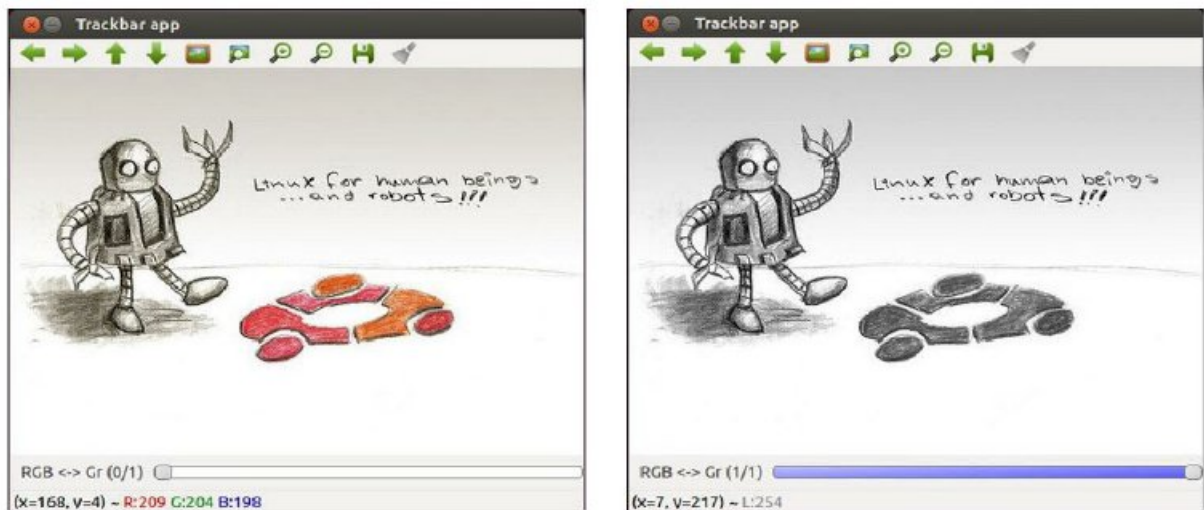
Para que o classificador possa distinguir e categorizar corretamente os objetos presentes na imagem, é necessário que as informações visuais sejam apresentadas de forma consistente e robusta a variações ambientais. É neste contexto que a conversão de cores se insere como etapa fundamental no *pipeline* de visão computacional. Trata-se de uma técnica de processamento de imagem que altera os valores dos pixels por meio de operações matemáticas básicas, como adição, subtração ou multiplicação por constantes, aplicadas diretamente aos valores de intensidade em imagens em escala de cinza ou aos canais individuais de uma imagem colorida, resultando na mudança sistemática das tonalidades presentes na cena.

No escopo deste trabalho, a conversão do espaço de cores BGR (padrão de aquisição da webcam) para o espaço HSV (Matiz, Saturação, Valor) desempenha um papel determinante para o sucesso da categorização. Diferentemente do espaço BGR, que mistura informações de cor e intensidade luminosa, o espaço HSV separa a cromaticidade (matiz) da luminosidade, tornando a representação da imagem menos suscetível a sombras e variações de iluminação. Esta transformação confere ao

sistema maior robustez e invariância, permitindo que o classificador KNN aprenda o padrão cromático do objeto-alvo (peça vermelha) independentemente das condições de brilho no momento da captura.

Estabelece-se, assim, uma relação de causalidade metodológica entre a conversão de cores e a categorização de objetos: a primeira atua como etapa de pré-processamento que potencializa a eficácia da segunda. Enquanto a categorização representa o objetivo final do sistema: atribuir corretamente o rótulo "peça presente" ou "esteira vazia", a conversão de cores constitui o mecanismo que organiza e padroniza os dados visuais brutos, viabilizando a extração de características discriminativas e, conseqüentemente, uma decisão de classificação precisa e confiável (BRAHMBHATT, 2013). A Figura 4 ilustra a técnica que altera os pixels por meio das operações matemáticas supracitadas.

Figura 4 - Aplicação do método de conversão de cores.



Fonte: Brahmbhatt (2013).

2.4 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial refere-se à habilidade de dispositivos eletrônicos de operar de forma que remete ao raciocínio humano. Isso envolve a percepção de variáveis, a tomada de decisões e a resolução de problemas. Em outras palavras, é

a capacidade de atuar de acordo com uma lógica semelhante ao processo de pensamento. Assim, a inteligência artificial busca desenvolver dispositivos que simulem a aptidão humana para raciocinar, perceber, decidir e solucionar questões, ou seja, para ser inteligente. Esse conceito está inserido no campo da computação e diz respeito à capacidade de máquinas (sejam físicas, softwares ou outros sistemas) de analisar dados externos, aprender a partir dessa análise e aplicar o conhecimento adquirido para realizar tarefas específicas e alcançar objetivos estabelecidos (BARBOSA; PORTES, 2023).

A adoção da inteligência artificial tornou-se um caminho sem volta para as empresas que buscam manter um alto padrão e oferecer a seus clientes maior eficácia e eficiência em suas operações, seja em produtos, processos ou serviços. A inteligência artificial, por sua vez, é uma área dedicada a replicar o comportamento e a inteligência humana através de máquinas. Em essência, o principal propósito da inteligência artificial é realizar uma variedade de tarefas humanas, desde as mais complexas até as mais simples, como conduzir um veículo. Além disso, sua importância se estende a outros processos e áreas das organizações, como a logística (PIMENTA, 2021).

No âmbito da inteligência artificial, é importante ressaltar o aprendizado de máquina (conhecido em inglês como *machine learning* ou ML), que emergiu como uma abordagem para a criação de softwares práticos em áreas como visão computacional, reconhecimento de voz, processamento de linguagem natural, controle de robôs e outras funções. Diversos desenvolvedores de sistemas de ML argumentam que, em muitas situações, é mais eficiente treinar um sistema apresentando exemplos de comportamentos desejados de entrada e saída, em vez de programá-lo manualmente, prevendo a resposta apropriada para todas as possíveis entradas (STURM et al., 2021; STEGER et al., 2017).

No contexto deste trabalho, o algoritmo KNN, inserido no campo do aprendizado de máquina, foi utilizado para desenvolver um sistema de visão computacional aplicado à automação industrial. A escolha por um método de ML baseado em instâncias, em vez de uma abordagem de programação explícita, alinha-se com a argumentação de Sturm et al. (2021) e Steger et al. (2017) sobre a eficiência de treinar

sistemas com exemplos. O projeto demonstra, na prática, como os conceitos de inteligência artificial e aprendizado de máquina podem ser integrados a sistemas físicos (CLP, atuadores pneumáticos) para criar soluções inteligentes e automatizadas, características da Indústria 4.0.

2.5 APRENDIZADO DE MÁQUINA

O aprendizado de máquina se destaca como um tema recorrente na maioria dos aplicativos de software. Desde a visão computacional até a ciência de dados em um contexto mais amplo, essa área tem gerado resultados e soluções impressionantes em relação a outras metodologias. Em atividades ligadas ao código-fonte, onde os aplicativos de aprendizado de máquina utilizam códigos de linguagens de programação como entrada para o fluxo de trabalho, essa técnica foi primeiramente empregada com o uso de métodos de processamento de linguagem natural (PLN) (DE PAULA; RODRIGUES, 2022).

A aplicação de aprendizado de máquina para resolver desafios nem sempre é simples e requer algumas condições prévias. É fundamental contar com um conjunto de exemplos de qualidade. Muitas vezes, é necessário construir e atualizar constantemente essa base de dados. Como nem todos os dados disponíveis são adequados, torna-se imprescindível utilizar técnicas que aumentem a qualidade das informações. Além disso, não há um algoritmo de aprendizado de máquina que resolva todos os tipos de problemas, o que exige a seleção cuidadosa de conjuntos de algoritmos que sejam adequados para a situação específica. Após a escolha dos algoritmos, é necessário definir os parâmetros que os regem (como, por exemplo, o número de camadas em uma rede neural). Uma vez concluído o treinamento, é essencial verificar se o algoritmo está efetivamente resolvendo o problema e com qual precisão isso está ocorrendo.

No contexto deste trabalho, que se insere na intersecção entre a engenharia elétrica e a ciência da computação, todas essas etapas foram consideradas no desenvolvimento do sistema de inspeção visual baseado em KNN. A coleta de um conjunto de treinamento representativo, a extração de características relevantes

(histogramas de cor no espaço HSV), a normalização dos dados com Standard Scaler, a otimização do parâmetro k por validação cruzada e a avaliação da acurácia do classificador são exemplos práticos da aplicação dos princípios de aprendizado de máquina discutidos por Ludermit (2021).

Sob a ótica da engenharia elétrica, o projeto abrange aspectos fundamentais da área: a instrumentação eletrônica, por meio da utilização da câmera Anker C200 como sensor de aquisição de imagens; a automação industrial, representada pelo CLP Festo FC640 e pelos atuadores pneumáticos que executam a ação física de aceitação ou rejeição das peças; e o sistema de comunicação, materializado na integração entre o software de visão computacional e o hardware de automação via protocolo HTTP, formando um sistema ciberfísicos completo. A escolha do KNN, um algoritmo de menor complexidade computacional, mostrou-se adequada à natureza do problema e ao ambiente controlado da bancada didática, permitindo que o processamento visual ocorresse em tempo real e acionasse os dispositivos eletroeletrônicos da bancada com a precisão necessária para simular uma linha de produção industrial.

Dessa forma, o trabalho não apenas aplica conceitos de aprendizado de máquina, mas também integra conhecimentos clássicos da engenharia elétrica às modernas técnicas de inteligência artificial, demonstrando a multidisciplinaridade característica dos sistemas industriais contemporâneos alinhados aos princípios da Indústria 4.0.

2.5.1 K- Nearest Neighbors (KNN)

O K-Nearest Neighbors (KNN) é um algoritmo de aprendizado de máquina supervisionado, utilizado tanto para classificação quanto para regressão, baseado no princípio de proximidade entre instâncias no espaço de características. Conforme descrevem Tan, Steinbach e Kumar (2016), ele é um método de aprendizado *lazy* (preguiçoso), pois não constrói um modelo explícito durante a fase de treinamento, mas armazena todo o conjunto de dados e realiza a predição no momento da classificação com base nos k vizinhos mais próximos da amostra de teste. O funcionamento do algoritmo pode ser resumido em três etapas principais:

- I. Cálculo da distância entre a instância a ser classificada e todas as instâncias do conjunto de treinamento, utilizando métricas como a distância Euclidiana, Manhattan ou Minkowski;
- II. Seleção dos k vizinhos mais próximos com base nas menores distâncias calculadas;
- III. Votação (no caso de classificação) ou média (no caso de regressão) entre os valores dos vizinhos selecionados para determinar a classe ou valor final (HASTIE et al., 2009).

A escolha do parâmetro “ k ” é crucial para o desempenho do algoritmo. Valores muito baixos podem tornar o modelo sensível a ruídos e *outliers*, enquanto valores muito altos podem suavizar excessivamente as fronteiras de decisão, prejudicando a capacidade de generalização (JAMES et al., 2013). Estudos como o de Wu et al. (2008) destacam que a normalização dos dados e a seleção adequada de características são etapas importantes para melhorar a eficácia do KNN, especialmente em problemas com dimensões elevadas.

Na área de visão computacional, o KNN tem sido aplicado com sucesso em tarefas de reconhecimento de padrões, classificação de imagens e detecção de objetos, especialmente quando combinado com extratores de características robustos, como histogramas de cor, texturas ou descritores locais (GONZALEZ; WOODS, 2018). Sua simplicidade, interpretabilidade e eficácia em conjuntos de dados de tamanho moderado tornam-no uma opção atraente para protótipos e aplicações onde a aplicabilidade do modelo é valorizada.

O KNN é um classificador não paramétrico baseado em instâncias: para um novo vetor de características, identificam-se os k vizinhos mais próximos no conjunto de treinamento por meio de uma métrica de distância, e a classe é definida por votação majoritária. Por ser sensível à escala das variáveis, a normalização dos dados, como o *StandardScaler*, é recomendada para equalizar a contribuição de cada atributo. A escolha do valor de k afeta diretamente o equilíbrio entre viés e variância: valores pequenos podem levar a overfitting, enquanto valores maiores tendem a suavizar as decisões, podendo incorrer em underfitting.

No contexto deste trabalho, o KNN foi escolhido pela sua baixa complexidade computacional durante a inferência, sua facilidade de implementação em Python com a biblioteca scikit-learn, e sua adequação a problemas de classificação binária com características visuais extraídas via OpenCV, como histogramas no espaço HSV e momentos de forma. A decisão pelo KNN alinha-se também com recomendações de Murphy (2012), que sugere o uso de métodos baseados em instâncias em cenários com dados limitados, mas bem caracterizados.

2.6 VISÃO COMPUTACIONAL

Na área da ciência da computação, a visão computacional é uma subárea que se dedica exclusivamente à interpretação de dados visuais por meio de processos computacionais. Historicamente, ela foi um dos primeiros desafios apresentados na década de 1950, no contexto do avanço da inteligência artificial, surgindo como uma ramificação da cibernética (CARDON et al., 2019).

A visão computacional é uma área que busca proporcionar aos computadores habilidades comparáveis ou até superiores às dos seres humanos, utilizando sensores como uma forma de "olhos" para reconhecer anomalias em seu entorno (VILLANUEVA; SALENGA, 2018). Em princípio, reconhecer e localizar um objeto é uma ação simples para os seres humanos. No entanto, instruir um computador a realizar essa tarefa tem sido um dos objetivos centrais da visão computadores, que possibilitam resolver problemas que eram inviáveis há alguns anos (STEGGER et al., 2017).

No contexto deste projeto, a visão computacional foi aplicada exatamente neste cenário industrial: o desenvolvimento de um sistema de inspeção visual automatizada para uma bancada didática com esteira, utilizando uma câmera para captura de imagens e algoritmos de aprendizado de máquina (KNN) para classificação da presença de uma peça vermelha, conforme detalhado nas seções anteriores.

computacional. Esse campo da ciência da computação capacita as máquinas a enxergar, reconhecer e analisar objetos com base em imagens e vídeos (KAUR; SINGH, 2022).

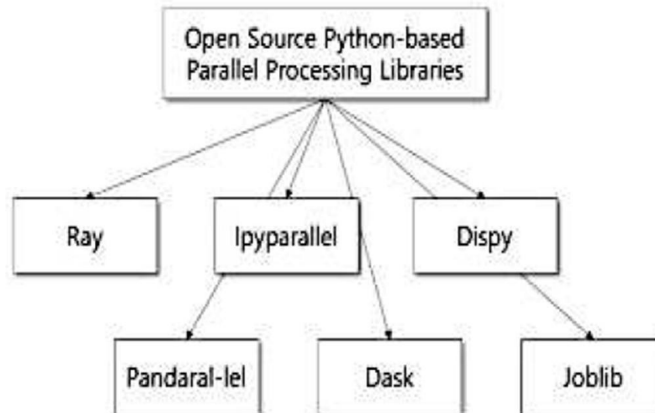
O setor de visão computacional tem experimentado um crescimento notável, superando a média de outras indústrias ao longo dos anos. Atualmente, os sistemas de visão de máquina fazem parte essencial de diversas máquinas e linhas de produção. Além disso, esses sistemas estão sendo constantemente implementados em novos domínios de aplicação, em grande parte devido ao aumento da velocidade dos

2.7 LINGUAGEM DE PROGRAMAÇÃO PYTHON

A linguagem Python foi desenvolvida na década de 1980 por Guido van Rossum, com o objetivo de ser uma linguagem interpretada que se destacasse por ter comandos simples e de fácil compreensão, além de uma sintaxe que se assemelha bastante ao inglês. Por essa razão, Python apresenta uma curva de aprendizado acentuadamente suave em relação a outras linguagens de programação, como Java ou C# (KALINOWSKI et al., 2023). Além disso, Python abrange diversos paradigmas de programação, como a orientação a objetos, programação funcional, imperativa e procedural. A linguagem também conta com um recurso fundamental que é a gestão automática de memória de forma dinâmica, além de oferecer uma vasta e rica coleção de bibliotecas padrão. Atualmente, vários sistemas operacionais disponibilizam intérpretes para Python (AZIZ et al., 2021).

A Figura 5 ilustra a diversidade do ecossistema Python, apresentando bibliotecas voltadas para processamento paralelo. Embora o presente projeto não utilize especificamente as bibliotecas de processamento paralelo destacadas na figura, sua apresentação justifica-se pela necessidade de demonstrar a amplitude e versatilidade da linguagem Python, que oferece soluções para as mais diversas áreas. No escopo deste trabalho, as bibliotecas efetivamente empregadas foram OpenCV para visão computacional, NumPy para manipulação eficiente de arrays e scikit-learn para implementação do algoritmo KNN, todas igualmente integrantes do rico ecossistema Python.

Figura 5 - Bibliotecas de processamento paralelo baseadas em Python.



Fonte: Aziz et al. (2021).

2.7.1 Recursos Utilizados

O desenvolvimento do sistema de classificação deste trabalho foi viabilizado pela utilização da biblioteca de aprendizado de máquina Scikit-learn. Trata-se de uma ferramenta de código aberto (open source) para a linguagem Python, distribuída sob a licença BSD, que se consolidou como uma das principais e mais abrangentes bibliotecas para a área. Sua adoção generalizada justifica-se por sua API consistente, documentação extensa e integração nativa com outras bibliotecas científicas fundamentais do ecossistema Python, como o NumPy e o SciPy.

A Scikit-learn oferece uma vasta gama de algoritmos para tarefas de aprendizado supervisionado e não supervisionado, além de ferramentas para pré-processamento de dados, seleção de modelos e métricas de avaliação. Uma das grandes vantagens da biblioteca é a uniformidade de sua interface, onde diferentes estimadores compartilham métodos padronizados como `fit()`, `predict()` e `transform()`, o que simplifica significativamente o processo de experimentação e implementação de pipelines de machine learning.

Para a tarefa específica de classificação binária deste projeto, foram empregados os seguintes componentes da biblioteca Scikit-learn:

- **KNeighborsClassifier:** Este é o estimador central do projeto, responsável pela implementação do algoritmo K-Nearest Neighbors (KNN). O KNN é um método baseado em instâncias (ou *lazy learning*), que não constrói um modelo

generalizado a partir dos dados de treinamento, mas sim armazena todas as instâncias para, no momento da predição, classificar um novo ponto com base na distância (por exemplo, Euclidiana) de seus "k" vizinhos mais próximos no espaço de características. No código, a classe é instanciada e configurada, por exemplo, com o parâmetro `n_neighbors=5`, que define o número de vizinhos considerado na votação para a classificação.

- **StandardScaler:** O KNN é um algoritmo sensível à escala das características, uma vez que o cálculo de distâncias pode ser dominado por variáveis com magnitudes maiores. Para mitigar esse problema, utiliza-se o `StandardScaler`. Este transformador padroniza as características removendo a média e escalando os dados para uma variância unitária (média=0, desvio padrão=1), garantindo que todas as dimensões contribuam de forma equitativa para a decisão final. O método `fit()` é chamado nos dados de treinamento para aprender os parâmetros de escala, que são então aplicados aos dados de treino e teste com o método `transform()`.
- **train_test_split:** Para uma avaliação justa e realista do desempenho do classificador, é fundamental separar o conjunto de dados em subconjuntos de treino e teste. A função `train_test_split` do módulo `model_selection` é utilizada para realizar essa divisão de forma aleatória, assegurando que o modelo seja testado em dados nunca vistos durante a fase de treinamento. Isso permite estimar com maior precisão a capacidade de generalização do sistema, evitando o *overfitting* e fornecendo métricas de avaliação mais confiáveis.

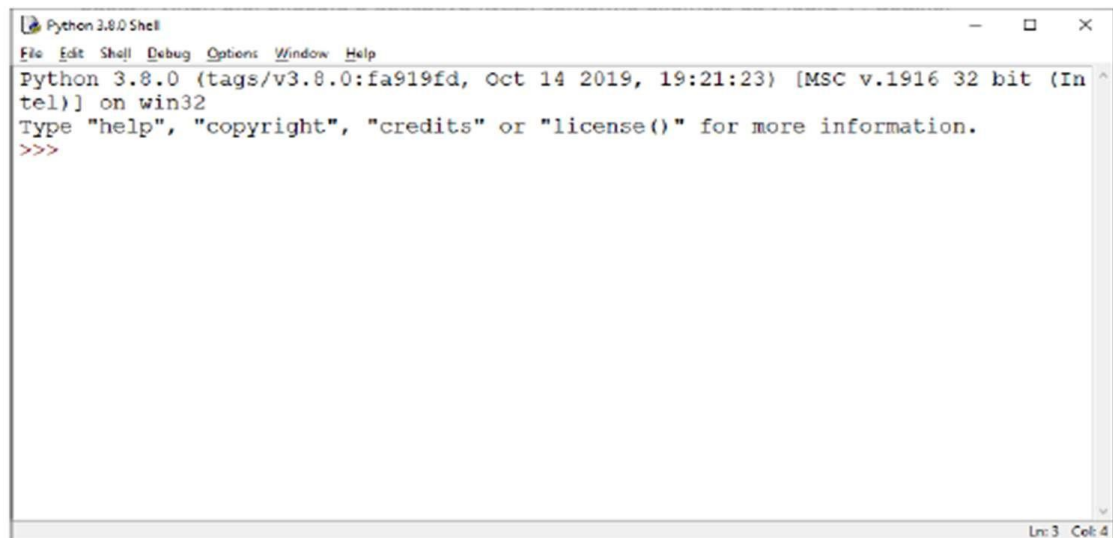
A combinação dessas ferramentas permitiu a construção de um pipeline de aprendizado de máquina robusto e alinhado com as boas práticas da área, desde o pré-processamento dos dados até a avaliação do modelo, contribuindo diretamente para a eficácia e confiabilidade do sistema de inspeção visual desenvolvido

2.7.2 IDLE Python

O IDLE é um ambiente de desenvolvimento integrado para Python que vem sendo disponibilizado em cada nova versão da ferramenta desde a 2.3. Embora não

seja a opção mais adequada para programação avançada, constitui uma excelente alternativa para iniciantes no aprendizado da linguagem (PAIVA et al., 2021), conforme ilustrado na Figura 6.

Figura 6 - O ambiente IDLE.



Fonte: Paiva et al. (2021).

2.7.3 Características da linguagem python

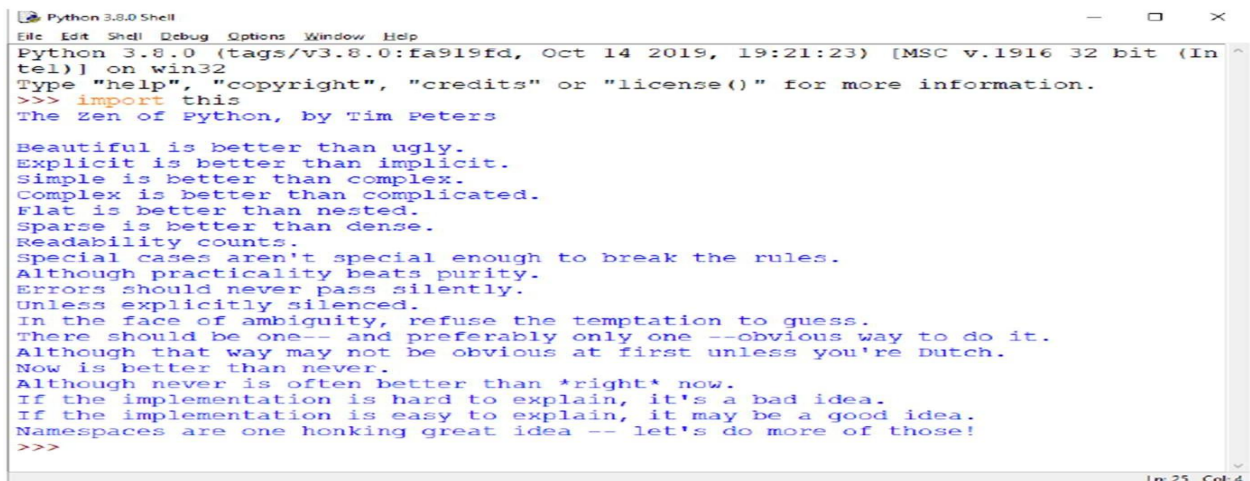
Paiva et al. (2021) afirmam que Python é amplamente reconhecida por suas características notáveis e por ser de fácil aprendizado. A proposta inicial da linguagem era facilitar a criação de códigos mais concisos e menos verbosos, ou seja, com reduzido uso de caracteres especiais, sintaxes complexas e estruturas de código simplificadas. Por esse motivo, destaca-se por:

- Facilidade de aprendizado, leitura e compreensão;
- Compatibilidade em diversas plataformas;
- Disponibilidade de um modo interativo;
- Uso de indentação para demarcar blocos de código;
- Praticamente nenhuma necessidade de palavras-chave ligadas à compilação;
- Implementação de um coletor de lixo que gerencia automaticamente a memória;
- Suporte à programação orientada a objetos;
- Capacidade para programação funcional; e

- Uma vasta gama de módulos de extensão que ampliam as possibilidades da linguagem Python.

Para diversas pessoas, Python carrega uma filosofia própria de programação. Um aspecto essencial da cultura dessa linguagem é “The Zen of Python”, um poema criado pelo programador Tim Peters. Para acessá-lo, basta digitar o comando “easter egg” import this (PAIVA et al., 2021), conforme ilustrado na Figura 7.

Figura 7 - O poema Zen.



```
Python 3.8.0 Shell
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Fonte: Paiva et al. (2021).

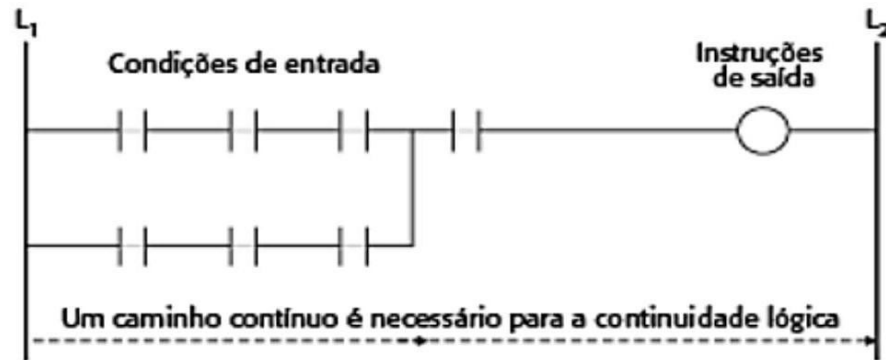
2.8 LINGUAGEM DE PROGRAMAÇÃO LADDER

Conforme Franchi e Camargo (2020), o conceito principal da lógica *ladder* é ilustrar o fluxo de "eletricidade virtual" através de um diagrama em forma de escada. Assim, a linha vertical à esquerda simboliza o pólo positivo, enquanto a linha da direita representa o pólo negativo, com a eletricidade fluindo do positivo para o negativo. Cada linha que conecta os dois pólos forma um degrau, e um diagrama *ladder* pode incluir diversos degraus.

Os elementos lógicos de entrada e saída são dispostos sobre cada degrau, com os dispositivos de entrada localizados à esquerda e os elementos de saída à direita. As saídas são ativadas quando as entradas de um degrau são fechadas, estabelecendo uma conexão contínua entre o pólo positivo e o polo negativo, como ilustrado na Figura 8. A forma de leitura da linguagem *ladder* determina que cada

degrau deve ser interpretado da esquerda para a direita e de cima para baixo, começando pelo último degrau.

Figura 8 - Estrutura básica de um degrau.



Fonte: Franchi e Camargo (2020).

2.8.1 Trilhos de energia

A rede Ladder é delimitada lateralmente por duas linhas verticais: à esquerda, o barramento de força esquerdo; à direita, o barramento de força direita. O barramento de alimentação à direita pode ser apresentado de forma explícita ou implícita (MENDES, 2021).

2.8.2 Elementos e estados das ligações

Como aponta Mendes (2021), os componentes das conexões podem ser dispostos de forma horizontal ou vertical. É imprescindível que o estado do componente de conexão seja sempre identificado como “Ligado (ON)” ou “Desligado (OFF)”, correspondendo aos valores booleanos literais 1 e 0, respectivamente. A expressão “estado da conexão” deve equivaler ao termo “fluxo de potência”. O estado do trilho à esquerda deve ser mantido como “Ligado” em todas as situações. Para o trilho à direita, nenhum estado é pré-definido.

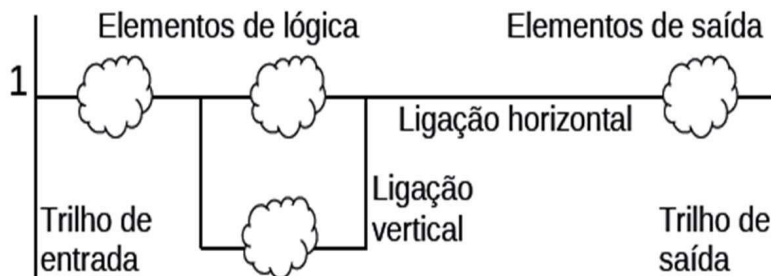
Um componente de conexão na horizontal deve ser representado por uma linha horizontal. Esse componente horizontal transmite o estado do elemento que está imediatamente à sua esquerda para o elemento que se encontra à sua direita. O

elemento de conexão vertical deverá apresentar uma linha vertical que se intersecciona com um ou mais elementos de conexão horizontal de cada lado. O estado da conexão vertical deve refletir a lógica OU dos estados “Ligado” das conexões horizontais à sua esquerda. Assim, o estado da conexão vertical deve ser determinado da seguinte forma:

- Desligado” se todas as conexões horizontais à sua esquerda estiverem “desligadas”;
- Ligado” se uma ou mais das conexões horizontais à sua esquerda estiverem “ligadas”.

Além disso, o estado da conexão vertical deve ser replicado para todas as conexões horizontais à sua direita, mas não deve ser transferido para nenhuma das conexões horizontais à sua esquerda (MENDES, 2021), como ilustrado na Figura 9.

Figura 9 - Elementos de ligação da linguagem Ladder.



Fonte: Mendes (2021).

2.8.3 Opencv

A crescente ubiquidade de imagens e vídeos, aliada ao aumento do poder computacional em dispositivos compactos, impulsiona a demanda por sistemas de visão computacional. Nesse contexto, a biblioteca de código aberto OpenCV (*Open Source Computer Vision Library*) destaca-se como uma ferramenta essencial para o desenvolvimento de aplicações que envolvem análise e processamento visual.

Lançada inicialmente em 1999 pela Intel, sob a liderança de Gary Bradski, a OpenCV consolidou-se como a principal biblioteca para pesquisadores e desenvolvedores da área. Seu primeiro lançamento estável (versão 1.0) ocorreu em 2006, seguido por uma atualização significativa em 2009 (OpenCV 2). A biblioteca

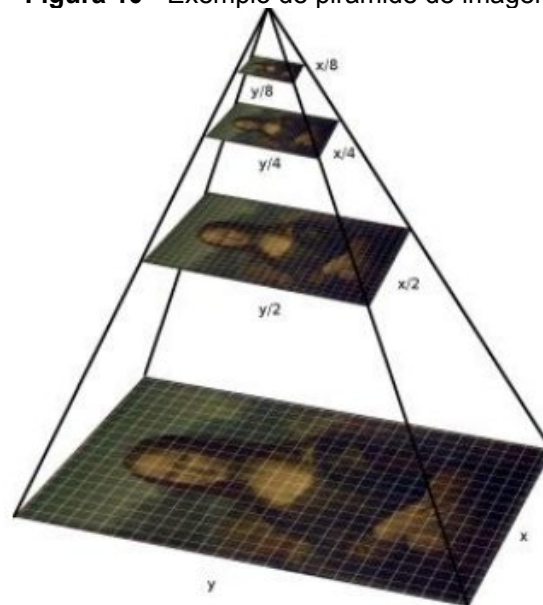
oferece mais de 2.500 algoritmos otimizados para processamento de imagem e vídeo e possui interfaces para as linguagens C ++, Python e Java, com suporte a múltiplas plataformas, como Windows, Linux, macOS, Android e iOS. Escrita em C ++ e otimizada para processamento multicore, a OpenCV é aplicada em áreas diversas, que vão desde arte interativa até robótica avançada (BRADSKI; KAEHLER, 2008; OPENCV, 2026).

No presente projeto, a OpenCV foi utilizada nas etapas de aquisição, pré-processamento e extração de características (histogramas de cor) que alimentaram o classificador KNN.

2.8.4 Pirâmide de imagens e janela deslizante

A pirâmide de imagens e a janela deslizante são técnicas clássicas para detecção de objetos em múltiplas escalas e posições (BRADSKI; KAEHLER, 2008; SZELISKI, 2010). A Figura 10 ilustra este conceito.

Figura 10 - Exemplo de pirâmide de imagem.



Fonte: Py ImageSearch (2026).

É fundamental esclarecer que estas técnicas não foram utilizadas no sistema desenvolvido neste trabalho. Sua apresentação tem finalidade exclusivamente contextual, demonstrando conhecimento de abordagens alternativas consagradas na literatura.

A decisão de não as implementar fundamenta-se em quatro razões:

- a) O sistema delimita manualmente uma Região de Interesse (ROI) fixa sobre a esteira, onde a peça obrigatoriamente aparece. Conforme Gonzalez e Woods (2018), a restrição espacial é estratégia legítima de redução de complexidade quando a posição do objeto é previsível;
- b) A abordagem adotada no projeto utiliza histogramas de cor como característica global, que são intrinsecamente invariantes à escala e à posição do objeto dentro da ROI. Bradski e Kaehler (2008) destacam que características globais como histogramas sacrificam informação espacial em prol da invariância geométrica, *trade-off* adequado quando a localização exata não é requerida. Esta propriedade torna a análise multiescala da pirâmide de imagens desnecessária;
- c) A complexidade computacional de $O(n^3)$ da combinação pirâmide com janela deslizante é incompatível com o requisito de processamento em tempo real em hardware acessível;
- d) O problema consiste em classificar a presença ou ausência de um objeto específico em região conhecida e controlada, não em detectar sua localização exata em cena complexa. Tais técnicas configurariam excesso de engenharia para o escopo proposto.

2.8.4.1 Relação com o KNN

Na arquitetura clássica de detecção, o classificador processaria centenas ou milhares de recortes por frame, varrendo toda a imagem em múltiplas escalas. No presente projeto, o KNN classifica uma única região, a ROI, por frame. Esta simplificação é viabilizada pelo conhecimento da posição do objeto e pela escolha de características globais, que dispensam a busca exaustiva por localização e escala.

Em síntese, o sistema não utiliza pirâmide de imagens nem janela deslizante porque o problema é de classificação de região fixa, não de detecção em cena aberta.

2.8.5 Relação entre limiarização de Otsu e k-Nearest Neighbors (KNN)

A integração entre técnicas de processamento de imagem e algoritmos de aprendizado de máquina é fundamental em sistemas modernos de visão computacional. Nesse contexto, a limiarização de Otsu e o algoritmo K-Nearest Neighbors (KNN) podem estabelecer uma relação complementar em *pipelines* de classificação visual automatizada. Conforme demonstrado por Gonzalez e Woods (2018), métodos de segmentação como Otsu atuam como estágios de pré-processamento que preparam os dados visuais para etapas subsequentes de classificação.

É importante ressaltar, contudo, que no presente trabalho a limiarização de Otsu não foi incorporada ao *pipeline* final do sistema. Esta seção dedica-se, portanto, a apresentar o potencial teórico dessa integração e a justificar sua não adoção na solução implementada, considerando as características específicas da aplicação desenvolvida em bancada didática.

2.8.5.1 Interface técnica entre segmentação e classificação

O algoritmo K-Nearest Neighbors (KNN), conforme descrito por Tan et al. (2016), requer como entrada um conjunto de características quantitativas extraídas das imagens. A limiarização de Otsu pode fornecer, em abordagens alternativas, o substrato ideal para essa extração, gerando máscaras binárias a partir das quais são calculados diversos descritores geométricos e estatísticos, tais como:

- I. Área: número de pixels da região segmentada;
- II. Perímetro: comprimento do contorno do objeto;
- III. Circularidade: medida de similaridade com um círculo perfeito;
- IV. Relação de Aspecto: proporção entre largura e altura;
- V. Momentos de Hu: descritores invariantes a transformações geométricas.

Estas características, extraídas após a segmentação por Otsu, formam o vetor de características que alimenta o classificador KNN. Bradski e Kaehler (2008) observam que a qualidade da segmentação impacta diretamente na eficácia da classificação, uma vez que características extraídas de regiões mal segmentadas podem introduzir ruído e reduzir a acurácia do sistema.

A limiarização automática de Otsu escolhe um limiar global que maximiza a separação entre duas classes de pixels (fundo e objeto) em termos de variância interclasse. Em ambientes controlados, essa técnica permite separar rapidamente regiões com diferentes intensidades, facilitando a extração de descritores simples.

A despeito do embasamento teórico apresentado, o sistema desenvolvido neste trabalho optou por não empregar a limiarização de Otsu em seu fluxo de processamento. Esta decisão metodológica fundamenta-se na seguinte consideração:

a) Natureza da característica discriminante:

O objeto-alvo (peça vermelha) é identificado primordialmente por sua cor, e não por sua forma ou intensidade luminosa. A limiarização de Otsu opera no domínio da intensidade dos pixels (escala de cinza), sendo ineficaz para distinguir objetos cuja principal característica distintiva é a cromaticidade. Conforme demonstrado por Bradski e Kaehler (2008), segmentações baseadas exclusivamente em limiares de intensidade apresentam baixa eficácia quando o objeto e o fundo possuem níveis de cinza semelhantes, ainda que apresentem cores distintas.

2.9 INTEGRAÇÃO COM AUTOMAÇÃO INDUSTRIAL E CLP

Os controladores lógicos programáveis (CLPs) são dispositivos eletrônicos robustos e dedicados, projetados especificamente para a automação de processos industriais, especialmente de natureza discreta ou sequencial. Sua programação é tipicamente realizada em linguagens padronizadas pela norma IEC 61131-3, sendo a linguagem *ladder diagram* uma das mais difundidas devido à sua representação intuitiva baseada em diagramas de circuitos elétricos (FRANCHI; CAMARGO, 2020). No contexto dos sistemas ciberfísicos que caracterizam a Indústria 4.0, a integração entre um sistema de visão computacional e um CLP é um componente essencial para

criar ciclos de controle fechado, conforme destacado por Hermann et al. (2015) em sua análise dos pilares tecnológicos da quarta revolução industrial.

Esta integração pode ser realizada por meio de redes de campo industriais tradicionais ou, de forma crescente, através de protocolos baseados em IP, escolha que depende diretamente da arquitetura de comunicação da fábrica e dos recursos disponíveis nos dispositivos. Steger et al. (2018) observam que a comunicação entre sistemas de visão computacional e controladores industriais têm evoluído para incorporar protocolos abertos e baseados em Ethernet, facilitando a interoperabilidade em ambientes de produção modernos.

Neste trabalho, a etapa de integração foi implementada utilizando o protocolo de comunicação HTTP. Especificamente, o sistema de visão computacional foi projetado para enviar requisições do tipo GET para uma interface *web* integrada ao CLP, configurada para receber comandos externos. Cada requisição contém parâmetros que endereçam e modificam o estado de registradores ou variáveis de memória específicas dentro do controlador. Ao alterar o estado dessas variáveis, a lógica do programa *ladder* em execução no CLP é ativada, desencadeando as ações de automação correspondentes, conforme descrito por Mendes (2021) em sua análise da estrutura de programação *ladder*.

Dessa forma, um resultado de classificação emitido pelo sistema de visão como "peça aceita" ou "peça rejeitada" é instantaneamente traduzido em um comando que aciona a parada de uma esteira ou a atuação de um cilindro pneumático para ejetar a peça do fluxo de produção. Esta abordagem baseada em HTTP oferece uma integração relativamente simples e versátil, aproveitando a conectividade de rede comum e permitindo uma prototipagem ágil, embora, para ambientes produtivos de alto desempenho, protocolos industriais com melhor determinismo temporal, como aqueles discutidos por Beyerer et al. (2015) em seu estudo sobre sistemas de inspeção automatizada, possam ser necessários em uma fase posterior de maturidade do sistema.

A integração entre visão computacional e automação industrial via CLP representa uma implementação prática dos conceitos de internet industrial das coisas (IIoT), conforme definido por Dias (2021), onde dispositivos inteligentes se comunicam

e cooperam para otimizar processos produtivos. Esta arquitetura permite que sistemas baseados em software, como os desenvolvidos com OpenCV e KNN, interajam diretamente com o mundo físico controlado por CLPs, criando uma ponte efetiva entre os domínios digital e físico característicos da Indústria 4.0 (ROZO-GARCÍA, 2020).

2.9.1 Generalização e robustez do classificador k-Nearest Neighbors KNN em contexto industrial

A aplicação do algoritmo K-Nearest Neighbors (KNN) em sistemas de inspeção visual industrial requer considerações específicas sobre sua capacidade de generalização e robustez. Conforme Tan et al. (2016) destacam, o KNN é um método baseado em instâncias que não constrói um modelo explícito durante o treinamento, mas armazena todo o conjunto de dados para realizar previsões durante a fase de teste. Esta característica, conhecida como *lazy learning*, traz implicações particulares para sua performance em ambientes sujeitos a variações, tornando fundamental que o conjunto de treinamento seja representativo das condições operacionais esperadas. Este aspecto foi diretamente considerado na metodologia deste trabalho.

2.9.1.1 Desafios de generalização com k-Nearest Neighbors KNN

Ambientes industriais apresentam uma gama de variáveis que podem comprometer a eficácia de sistemas de visão computacional. Szeliski (2011) identifica como principais desafios:

- I. Variações de iluminação: flutuações na intensidade e direção da luz, sombras projetadas e reflexos especulares;
- II. Oclusões parciais: presença de sujeira, respingos de óleo ou outros objetos interferindo na visão;
- III. Variações posicionais: pequenas rotações, translações e alterações de escala das peças na linha;
- IV. Degradação de componentes: desgaste gradual de câmeras, lentes e fontes de iluminação.

Para mitigar esses efeitos, LeCun et al. (2015) defendem que a diversificação do conjunto de dados de treinamento é uma estratégia fundamental. Isso pode ser alcançado por meio de:

- Coleta em múltiplas condições operacionais: captura de imagens em diferentes turnos, condições de iluminação e estados de manutenção;
- Técnicas de aumento de dados (*data augmentation*): transformações sintéticas que expandem o *dataset* original.

O desempenho do KNN em condições operacionais reais depende criticamente da representatividade e diversidade do conjunto de treinamento. Hastie et al. (2009) alertam que o KNN pode sofrer com a *curse of dimensionality* (maldição da dimensionalidade) quando aplicado a dados de alta dimensionalidade sem técnicas adequadas de redução de características. No contexto de visão computacional industrial, onde imagens podem conter milhares de pixels, a extração de características relevantes como descritores de forma, textura ou cor torna-se etapa crucial antes da classificação com KNN.

Para mitigar problemas de generalização, Wu et al. (2008) recomendam a aplicação sistemática de técnicas de pré-processamento e normalização dos dados. No sistema desenvolvido, características extraídas após processamento com OpenCV, como área, circularidade e momentos invariantes, foram normalizadas para a escala [0,1] antes da classificação com KNN, seguindo as práticas estabelecidas por Bishop (2006) para métodos baseados em distância.

2.9.2 Técnicas de robustez para k-Nearest Neighbors KNN

A robustez do classificador KNN a variações ambientais pode ser significativamente melhorada por meio de estratégias de *data augmentation* aplicadas durante a fase de treinamento. Shorten e Khoshgoftaar (2019) demonstram que transformações como rotação, translação e ajuste de contraste em imagens de treinamento permitem que o KNN aprenda invariâncias a essas variações. No presente trabalho, o conjunto de treinamento foi expandido digitalmente conforme as técnicas descritas no Quadro 1.

Quadro 1 - Técnica de aumento de dados para robustez.

Técnica	Objetivo	Exemplo de Aplicação
Transformações geométricas	Lidar com variações posicionais	Rotações ($\pm 15^\circ$), translações ($\pm 10\%$), zoom
Modificações de cor e brilho	Robustez a variações luminosas	Ajuste de contraste ($\pm 20\%$), saturação, temperatura de cor
Adição de ruído e artefatos	Simular degradações reais	Ruído gaussiano ($\sigma = 0,01$), simulação de sujeira, borramento
Oclusões aleatórias	Preparar para obstruções parciais	Máscaras aleatórias cobrindo até 20% da imagem

Fonte: Elaborado pelos autores (2026).

Estas transformações, quando aplicadas sistematicamente conforme Perez e Wang (2017), criam vizinhos artificiais no espaço de características. Esses vizinhos, invariantes a perturbações comuns, ajudam o KNN a generalizar melhor para condições não observadas durante o treinamento inicial.

2.9.3 Avaliação de desempenho além da acurácia com conjuntos desbalanceado

Em contextos industriais, especialmente quando se trata de detecção de defeitos, os conjuntos de dados frequentemente apresentam distribuições desbalanceadas. He e Garcia (2009) alertam que, nestes cenários, métricas tradicionais como a acurácia global podem ser enganosas, mascarando baixo desempenho na classe minoritária (defeitos). No presente projeto, esta preocupação justifica a adoção do F1-Score como métrica principal de avaliação, em detrimento da acurácia simples.

As métricas recomendadas por Sokolova e Lapalme (2009) para avaliação abrangente incluem:

- Precisão (*Precision*): proporção de peças verdadeiramente defeituosas entre todas as classificadas como defeituosas — $P = TP/(TP+FP)$. Mede a confiabilidade das detecções positivas.
- Revocação (*Recall*): proporção de peças defeituosas corretamente identificadas — $R = TP/(TP+FN)$. Mede a capacidade de encontrar todos os casos positivos.
- F1-Score: média harmônica entre precisão e revocação — $F1 = 2 \times (P \times R) / (P + R)$.
- Curva ROC e AUC: análise do desempenho em diversos limiares de decisão.

Bishop (2006) recomenda ainda a utilização de validação cruzada estratificada para garantir que a avaliação reflita adequadamente a distribuição real das classes, especialmente em conjuntos desbalanceados.

Contudo, em cenários industriais onde defeitos são eventos raros, os conjuntos de dados frequentemente apresentam distribuições desbalanceadas. He e Garcia (2009) observam que o KNN, por basear-se em votação majoritária entre os k vizinhos mais próximos, pode apresentar viés em favor da classe majoritária quando k é definido inadequadamente. Para contornar este problema, Tan et al. (2016) sugerem o uso de pesos por distância inversa, onde vizinhos mais próximos têm maior influência na decisão estratégia que pode ser explorada em trabalhos futuros para aumentar a robustez do sistema em cenários com maior variabilidade.

A validação cruzada estratificada de 5 folds, conforme proposta por Bishop (2006), foi utilizada para garantir estimativas realistas do desempenho do classificador KNN durante a etapa de otimização do parâmetro k .

2.9.4 Otimização de hiperparâmetros para robustez

A seleção do parâmetro k (número de vizinhos) é crítica para o equilíbrio entre viés e variância no KNN. James et al. (2013) demonstram que valores muito baixos de k ($k=1$) produzem modelos com baixo viés, mas alta variância, tornando-os sensíveis a ruídos nos dados, enquanto valores muito altos produzem modelos excessivamente suavizados que podem perder detalhes importantes da distribuição. Para determinar o k ótimo no contexto industrial, foi realizada uma busca em grade (*grid search*)

testando valores de k de 1 a 15, utilizando validação cruzada com 5 *folds* e otimizando o F1-Score, conforme recomendação de Sokolova e Lapalme (2009) para cenários com classes desbalanceadas. No presente projeto, este *trade-off* foi considerado experimentalmente, optando-se por $k=5$ após validação cruzada, valor que apresentou o melhor equilíbrio entre acurácia e robustez para a tarefa específica de classificação da peça vermelha.

2.9.5 Considerações sobre tempo de resposta

Embora o KNN seja computacionalmente eficiente durante o treinamento, com complexidade $O(1)$, sua fase de inferência apresenta complexidade $O(nd)$ para cada nova amostra, onde “ n ” é o tamanho do conjunto de treinamento e “ d ” a dimensionalidade dos dados (MURPHY, 2012). Para contornar esta limitação, o autor sugere técnicas de indexação espacial, como KD-Trees ou Ball Trees, que reduzem a complexidade média da busca. No presente projeto, esta estratégia foi adotada por meio da configuração da biblioteca Scikit-learn com algoritmo KD-Tree (PEDREGOSA et al., 2011), reduzindo a complexidade média de busca para $O(\log n)$.

Para aplicações em linhas de produção contínuas, o tempo de resposta do sistema constitui parâmetro crítico. Steger et al. (2018) estabelecem que a latência total do sistema, compreendendo captura, processamento e atuação, deve ser inferior ao tempo disponível entre peças consecutivas. Gonzalez e Woods (2018) sugerem que sistemas destinados a linhas de alta velocidade devem almejar latências inferiores a 100 ms por ciclo completo para evitar atrasos acumulativos.

Em sistemas de visão computacional aplicados à automação industrial, o tempo de inferência, definido como o intervalo entre a aquisição da imagem e a tomada de decisão, constitui parâmetro crítico para viabilizar o processamento em tempo real. A otimização desse tempo pode ser alcançada por meio da seleção de algoritmos eficientes com complexidade computacional adequada (CORMEN et al., 2012), da otimização de hiperparâmetros que equilibre precisão e velocidade (MANNING et al., 2008), e da implementação otimizada com bibliotecas compiladas e técnicas de paralelização (BRADSKI; KAEHLER, 2008). No contexto deste trabalho, tais

estratégias foram consideradas ao optar-se por um vetor de características de baixa dimensionalidade (180 dimensões) e pelo algoritmo KNN com implementação otimizada da biblioteca scikit-learn, o que garantiu a classificação em tempo real, conforme demonstrado nos resultados experimentais.

2.9.6 Abordagem implementada neste trabalho

No sistema desenvolvido, as considerações de generalização e robustez foram trabalhadas por meio de:

- I. Dataset diversificado: coleta em diferentes condições de iluminação e posições;
- II. Aumento de dados aplicado: rotações, variações de brilho e adição de ruído gaussiano;
- III. Avaliação com múltiplas métricas: acurácia, precisão, revocação e F1-Score;
- IV. Validação cruzada: 5-fold estratificado para estimativa realista do desempenho;
- V. Monitoramento de latência: medição sistemática dos tempos de processamento.

Esta abordagem alinha-se com as melhores práticas estabelecidas por Beyerer et al. (2015) para sistemas de inspeção visual industrial, priorizando tanto a precisão quanto a confiabilidade operacional.

2.9.7 Conclusão sobre a aplicação do k-Nearest Neighbors KNN

A implementação e avaliação do sistema de inspeção visual baseado no algoritmo K-Nearest Neighbors (KNN) permitem validar sua aplicabilidade em cenários de automação industrial com requisitos de baixa complexidade visual, porém com demandas rigorosas de confiabilidade e tempo de resposta. O KNN, quando integrado a um pré-processamento de imagens como o OpenCV é uma arquitetura de comunicação HTTP com CLP Festo, configura uma solução viável para tarefas de classificação binária em ambientes controlados, mantendo relevância quando combinado com técnicas modernas de pré-processamento e validação (GOODFELLOW et al., 2016).

A capacidade de generalização do classificador mostrou-se diretamente dependente da diversidade e representatividade do conjunto de dados de treinamento. A aplicação sistemática de técnicas de *data augmentation*, incluindo rotações, variações de brilho e contraste e adição de ruído, permitiu simular digitalmente as variações inerentes ao ambiente produtivo, como flutuações de iluminação, posicionamento irregular das peças e presença de interferências (SHORTEN; KHOSHGOFTAAR, 2019). Esta abordagem, aliada à validação cruzada estratificada, assegurou que o desempenho observado durante os testes refletisse adequadamente, conforme as melhores práticas estabelecidas na literatura especializada (BISHOP, 2006).

A avaliação do desempenho não se limitou à métrica de acurácia global, dado o caráter potencialmente desbalanceado dos conjuntos de dados típicos de inspeção industrial. Métricas complementares como precisão, revocação e F1-Score forneceram uma visão mais abrangente e confiável da eficácia do classificador, especialmente na detecção da classe minoritária (SOKOLOVA; LAPALME, 2009). Esta análise multidimensional revelou que o sistema mantém um equilíbrio adequado entre a identificação correta de peças presentes e a minimização de falsos positivos, aspecto crítico para a operação contínua da linha de produção.

Além da precisão de classificação, o tempo de resposta do sistema configurou-se como parâmetro operacional decisivo. A latência total, compreendendo a captura da imagem, o processamento com OpenCV, a classificação com KNN e a comunicação com o CLP, foi otimizada para permanecer compatível com o ciclo de produção simulado. A escolha do algoritmo KNN, complementada pela implementação eficiente com KD-Trees via Scikit-learn, demonstrou ser adequada para aplicações que exigem inferência em tempo real, sem comprometer a confiabilidade das decisões (MURPHY, 2012).

Em síntese, este trabalho comprovou que a combinação OpenCV, KNN e CLP Festo constitui uma arquitetura tecnicamente sólida para automação e inspeção visual em bancada didática. A solução desenvolvida oferece não apenas desempenho satisfatório em termos de acurácia e robustez, mas também vantagens práticas como baixo custo de implementação, facilidade de interpretação dos resultados e integração

simplificada com sistemas de automação industriais convencionais. Futuros desenvolvimentos podem explorar a incorporação de técnicas de aprendizado semi-supervisionado para reduzir a dependência de dados rotulados, a otimização adicional dos tempos de inferência para linhas de maior velocidade e a expansão do sistema para classificação multiclasse, ampliando seu escopo na Indústria 4.0.

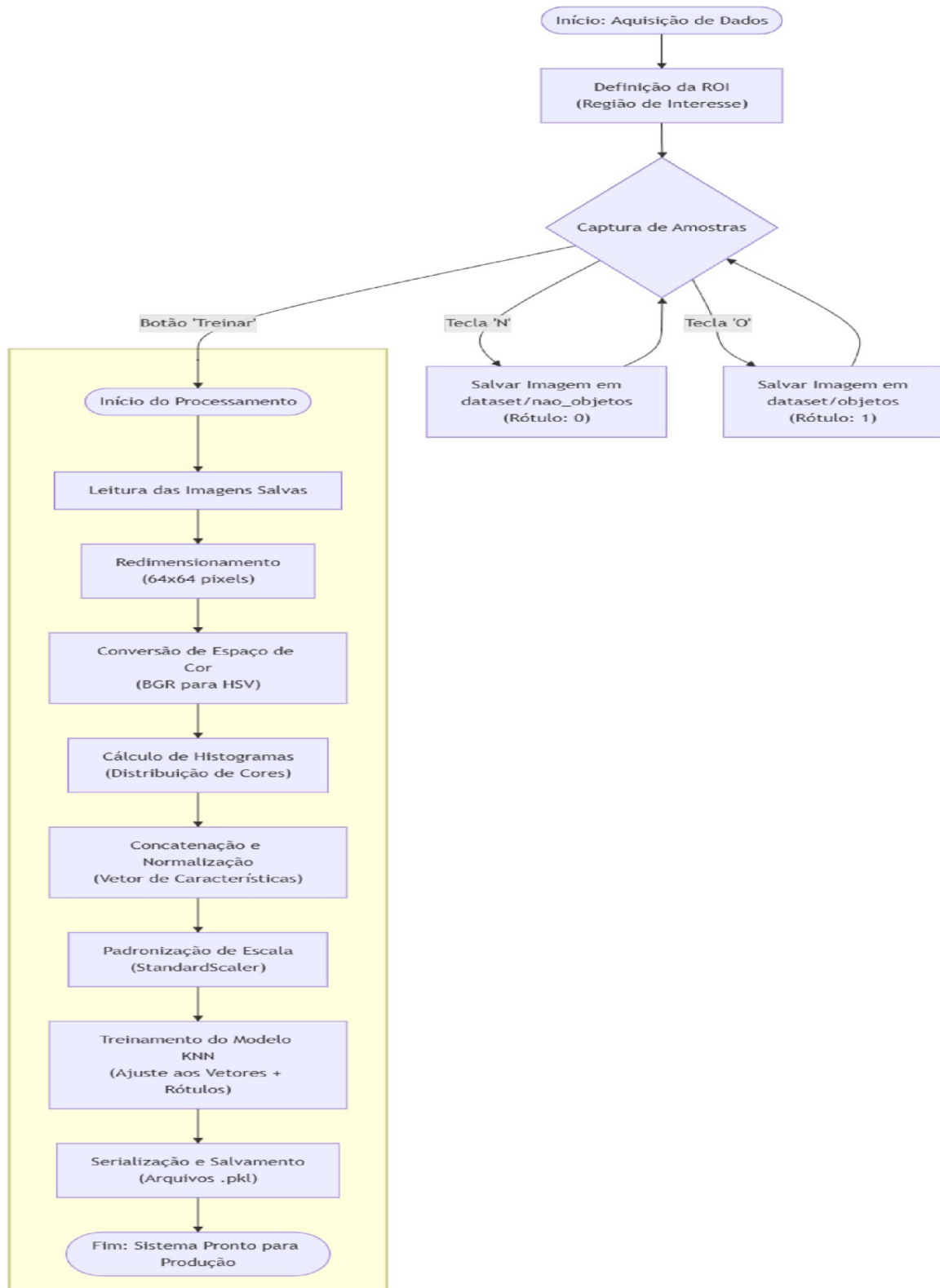
3 METODOLOGIA

3.1 MATERIAIS E MÉTODOS

A metodologia experimental foi dividida em duas fases principais: a preparação do conjunto de dados de treinamento e o *pipeline* de processamento e aprendizado do modelo. O fluxo geral do experimento é representado pelo diagrama da Figura 11.

A infraestrutura utilizada consistiu em um notebook Samsung Book 2 com processador Intel Core i7, equipado com uma webcam Anker C200 (resolução 2K) para aquisição de imagens. A linha de produção foi simulada em uma bancada didática com esteiras e atuadores pneumáticos, controlados por um CLP Festo FC640, o qual disponibiliza uma interface de comunicação via HTTP em rede local. Diferentemente da proposta inicial, optou-se por não utilizar um microcomputador embarcado (Raspberry Pi) nem um sensor fotoelétrico para disparo da captura. A aquisição de imagens foi realizada de forma contínua pela câmera, com a inferência do modelo ocorrendo em tempo real.

Figura 11 - Diagrama fluxograma da estrutura experimental.



Fonte: Elaborada pelos autores (2026).

3.1.1 Fase 1: aquisição e rotulagem dos dados

Esta fase inicial tem como objetivo a construção de um conjunto de dados robusto e representativo, fundamental para o aprendizado supervisionado do classificador. O processo seguiu as etapas abaixo:

- I. Definição da Região de Interesse (ROI): A primeira etapa consistiu na delimitação manual de um retângulo (visualizado como um quadrado azul na interface) sobre a área da imagem correspondente à esteira. Esta operação atua como um filtro espacial, eliminando ruídos visuais do ambiente (como fundo, cabos e outros objetos) e restringindo o processamento computacional apenas à área relevante. Esta medida aumenta a eficiência do sistema e reduz potenciais fontes de erro;
- II. Captura e Rotulagem Manual das Amostras: Com a ROI definida, procedeu-se à coleta das amostras. Para cada *frame* capturado, o operador atribui um rótulo manualmente:
 - Tecla 'O': a imagem foi salva como exemplo positivo (objeto presente);
 - Tecla 'N': a imagem foi salva como exemplo negativo (fundo/esteira vazia);
- III. Armazenamento Estruturado do *Dataset*: As imagens foram automaticamente armazenadas em diretórios distintos (*dataset/objetos* e *dataset/não objetos*), associadas aos seus rótulos binários (1 para objeto, 0 para não-objeto). Esta organização persistente e balanceada permite a reprodutibilidade do treinamento e ajustes futuros sem a necessidade de nova coleta de dados.

3.1.2 Fase 2: Pipeline de Treinamento (Engenharia de Características e Aprendizado do Modelo)

Nesta fase, os dados brutos coletados passam por um *pipeline* de processamento automático para extração de características e treinamento do modelo classificador.

- a) Leitura e Carregamento: Todas as imagens e seus respectivos rótulos são carregados da memória secundária para a RAM, otimizando a velocidade das operações subsequentes;

- b) Redimensionamento: Cada imagem é redimensionada para um formato fixo de 64x64 pixels. Esta etapa de redução de dimensionalidade é crucial para diminuir a complexidade computacional, mantendo a informação visual essencial e habilitando o processamento em tempo real;
- c) Conversão do Espaço de Cores: As imagens são convertidas do espaço de cores padrão BGR (Blue, Green, Red) para o espaço HSV (Hue, Saturation, Value). Esta conversão visa aumentar a robustez do sistema a variações de iluminação, uma vez que o HSV separa a informação de cor (matiz) da intensidade luminosa (valor);
- d) Cálculo de Histogramas: Para cada imagem no espaço HSV, é calculado um histograma da componente de matiz (Hue). Esta transformação confere ao descritor uma característica de invariância à posição e rotação do objeto na cena, pois captura a distribuição de cores independentemente de sua localização espacial;
- e) Concatenação e Normalização: Os histogramas gerados são concatenados formando um vetor de características únicas para cada imagem. Este vetor é então normalizado para uma escala comum (entre 0 e 1), garantindo que todas as características contribuam de maneira equilibrada para a classificação;
- f) Padronização de Escala: Aplica-se a técnica *StandardScaler*, que padroniza o conjunto completo de vetores de características para que apresentem média zero e desvio padrão unitário. Esta etapa é vital para algoritmos baseados em distância, como o KNN, pois assegura que todas as dimensões do vetor tenham a mesma influência no cálculo de similaridade;
- g) Treinamento do Modelo KNN: Utilizando os vetores de características padronizados e seus rótulos, o algoritmo K-Nearest Neighbors (KNN) é treinado. A escolha deste classificador se deve à sua simplicidade, eficácia na identificação de padrões não-lineares e baixo custo computacional durante a fase de treinamento;
- h) Serialização e Salvamento: O modelo KNN treinado e o objeto *StandardScaler* ajustado são serializados e salvos em arquivos no formato .pkl (usando a biblioteca *pickle* do Python). Isso garante a persistência do modelo, permitindo que ele seja carregado e utilizado diretamente na fase de inferência sem a necessidade de retreinamento.

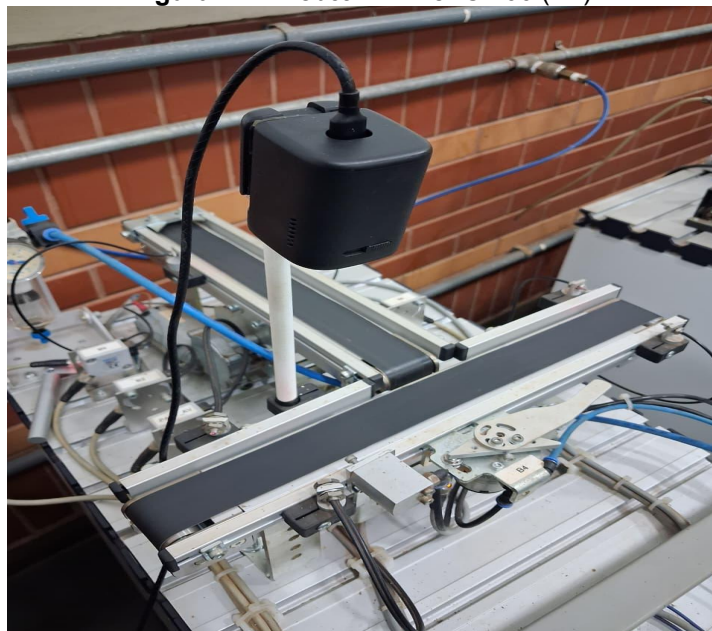
3.2 EQUIPAMENTOS UTILIZADOS

Nesta seção são apresentados e especificados os principais equipamentos e softwares utilizados para a construção do protótipo.

3.2.1 Câmera Digital

Para a aquisição das imagens, foi utilizada uma webcam de alta resolução, posicionada de forma fixa sobre a bancada didática, com campo de visão direcionado à região de interesse da esteira.

Figura 12 - Webcam Anker C 200 (2K).



Fonte: Elaborada pelos autores (2026).

Especificações técnicas:

- Modelo: Anker C200
- Resolução: 2K (1920 × 1080 pixels)
- Taxa de captura: 30 fps
- Conexão: USB
- Formato de imagem: BGR (padrão OpenCV)

- Configuração: Exposição automática desativada para consistência das capturas

3.2.2 OpenCV – Biblioteca de Visão Computacional

O OpenCV (*Open Source Computer Vision Library*) é uma biblioteca multiplataforma de código aberto desenvolvida para aplicações em tempo real de visão computacional e aprendizado de máquina. No contexto deste projeto, foi utilizada como ferramenta principal para processamento e análise de imagens.

Características principais:

- Licença: BSD – uso acadêmico e comercial gratuito;
- Linguagens: Python (interface principal utilizada), C++, Java;
- Multiplataforma: Windows, Linux, macOS, Android, iOS;
- Otimização: Algoritmos otimizados para processamento em tempo real.

Quadro 2 - Módulos e funções utilizadas no projeto.

Módulo/Função	Descrição
cv2.VideoCapture	Aquisição de vídeo da câmera
cv2.cvtColor	Conversão de espaços de cor (BGR para HSV)
cv2.resize	Redimensionamento de imagens
cv2.calcHist	Cálculo de histogramas de cor
cv2.imread / cv2.imwrite	Leitura e gravação de imagens
cv2.imshow	Exibição de imagens

Fonte: Elaborada pelos autores (2026).

3.2.2.1 Exemplo de código – Aquisição de imagem:

```
import cv2
```

```
# Inicialização da câmera
```

```
câmera = cv2.VideoCapture(0) # Webcam Anker C200
```

```
ret, frame = camera.read() # Captura de um frame
```

3.2.2.2 Exemplo de código – Pré-processamento:

```
# Conversão para o espaço HSV
hsv_image = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Redimensionamento para 64x64 pixels
resized = cv2.resize(hsv_image, (64, 64))

# Cálculo do histograma do canal Hue (matiz)
hist = cv2.calcHist([hsv_image], [0], None, [180], [0, 180])
```

3.2.2.3 Integração com outras bibliotecas:

- NumPy: As imagens são representadas como *arrays* NumPy;
- Scikit-learn: Os vetores de características extraídos com OpenCV alimentam o classificador KNN;
- Matplotlib: Utilizada para visualização auxiliar de histogramas e imagens.

3.2.2.4 Configuração do ambiente:

```
# Instalação das bibliotecas utilizadas
pip install opencv-python==4.8.1
pip install opencv-contrib-python
```

3.2.2.5 Limitações encontradas:

Durante o desenvolvimento e a implementação do sistema, algumas limitações relacionadas ao uso da biblioteca OpenCV foram identificadas, as quais merecem registro para fins de contextualização e para orientar trabalhos futuros.

A primeira limitação refere-se à curva de aprendizado associada à biblioteca. Embora o OpenCV ofereça uma vasta gama de funcionalidades, muitos de seus conceitos fundamentais, como a representação de imagens como matrizes multidimensionais, os diferentes espaços de cor e as operações morfológicas, exigem

conhecimento prévio em processamento digital de imagens e álgebra linear. Essa característica pode representar um obstáculo inicial para desenvolvedores sem experiência na área, demandando um tempo adicional de estudo e experimentação para a correta aplicação das funções disponíveis.

Outro aspecto relevante diz respeito ao gerenciamento de memória. Em aplicações que operam com captura contínua de vídeo e processamento frame a frame, como é o caso deste projeto, a alocação e liberação inadequadas de recursos podem levar ao acúmulo de memória não liberada (*memory leak*), comprometendo a estabilidade do sistema em execuções prolongadas. Observou-se a necessidade de gerenciar explicitamente a liberação de objetos como capturas de vídeo e janelas de visualização, sob pena de degradação do desempenho ao longo do tempo.

Por fim, a documentação da biblioteca, embora extensa e rica em exemplos, apresenta desafios pontuais. Determinadas funções possuem documentação técnica densa, com explicações sucintas sobre parâmetros e comportamentos esperados, o que pode dificultar a compreensão plena de seu funcionamento em cenários específicos. Em alguns casos, foi necessário recorrer a fóruns da comunidade e exemplos práticos disponíveis online para esclarecer dúvidas não totalmente sanadas pela documentação oficial.

Apesar dessas limitações, a flexibilidade e o poder da OpenCV superaram amplamente as dificuldades encontradas, sendo possível contornar a maioria dos obstáculos com estudo complementar e boas práticas de programação. O registro dessas observações visa auxiliar futuros desenvolvedores a antecipar possíveis dificuldades e otimizar seu processo de aprendizado e implementação com a biblioteca.

3.2.2.6 Desempenho obtido:

A avaliação do desempenho do sistema desenvolvido foi conduzida considerando três aspectos fundamentais para aplicações de visão computacional em tempo real: a taxa de processamento de quadros, a estabilidade operacional e a qualidade da extração de características visuais. Os testes foram realizados utilizando

a configuração de hardware descrita anteriormente, composta por um notebook Samsung Book2 com processador Intel Core i7 e câmera Anker C200, representando um cenário de hardware de médio porte, acessível para aplicações didáticas e de prototipagem.

Em relação à taxa de processamento, o sistema apresentou desempenho médio entre 22 e 25 quadros por segundo (fps) durante a execução contínua do pipeline completo de visão computacional. Este resultado é considerado satisfatório para o escopo do projeto, uma vez que a aplicação de inspeção visual em bancada didática não exige taxas de quadros extremamente elevadas, sendo suficiente para capturar e processar as imagens da esteira em velocidade compatível com o movimento simulado das peças. O valor obtido reflete o equilíbrio alcançado entre a complexidade computacional das operações de processamento e a capacidade do hardware utilizado.

A confiabilidade do sistema foi verificada por meio de testes de operação contínua com duração média de 30 minutos por sessão, período representativo para atividades didáticas e experimentais. Durante esses testes, não foram observadas instabilidades como travamentos, queda abrupta na taxa de quadros ou falhas de comunicação com o CLP. O sistema manteve-se responsivo ao longo de toda a execução, indicando que o gerenciamento de recursos, especialmente a liberação adequada de memória e o controle de threads para comunicação HTTP, foi implementado de forma eficaz.

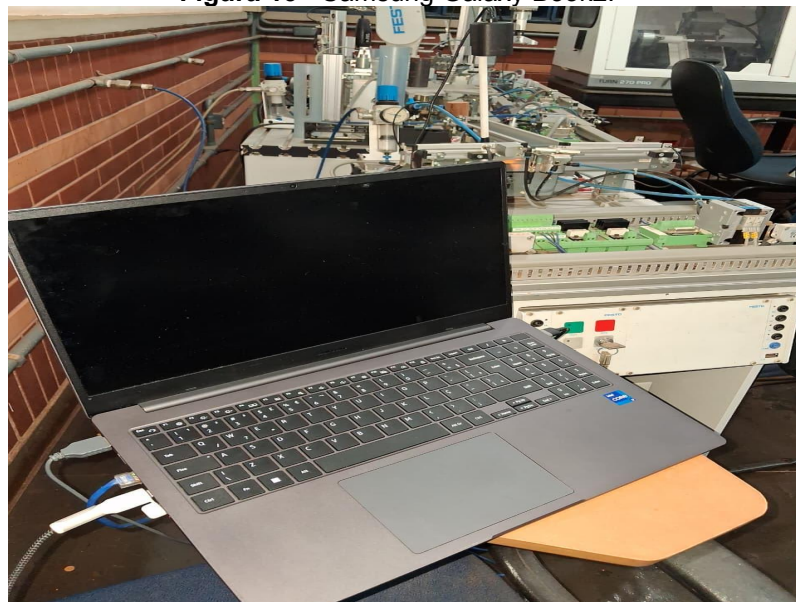
No que diz respeito à precisão da extração de características, os histogramas calculados a partir das imagens redimensionadas mostraram-se consistentes ao longo das diferentes capturas realizadas nas mesmas condições experimentais. A combinação da conversão para o espaço HSV com o cálculo de histogramas normalizados demonstrou robustez suficiente para manter a integridade das informações cromáticas extraídas, mesmo diante de pequenas variações de iluminação inerentes ao ambiente de laboratório. Esta consistência na representação das imagens como vetores de características foi determinante para o desempenho satisfatório do classificador KNN, conforme discutido nas seções de avaliação de acurácia.

De forma geral, a utilização da biblioteca OpenCV mostrou-se acertada para os objetivos do projeto. Sua vasta gama de funções otimizadas permitiu que o desenvolvimento se concentrasse na lógica de aplicação, como a definição da ROI, a extração de características e a integração com o classificador, em vez de demandar esforço significativo na implementação de operações fundamentais de processamento de imagens, como redimensionamento, conversão de espaços de cor e cálculo de histogramas. Esta característica da biblioteca foi essencial para viabilizar a construção de um protótipo funcional dentro do cronograma estabelecido, consolidando a OpenCV como uma ferramenta sólida e adequada para projetos de natureza semelhante.

3.2.3 Notebook

O processamento das imagens e a execução do algoritmo de classificação foram realizados em um notebook de médio desempenho, cujas especificações são apresentadas a seguir.

Figura 13 - Samsung Galaxy Book2.



Fonte: Elaborada pelos autores (2026).

Quadro 3 - Especificações técnicas:

Componente	Especificação
Marca/Modelo	Samsung Galaxy Book2

Processador	Intel® Core™ i7-1270P
Tela	14 polegadas, Full HD (1920 × 1080)
Memória RAM	16GB DDR4-SDRAM
Armazenamento	512 GB SSD
Conectividade	Wi-Fi 6E (802.11ax)/ Ethernet (RJ45)
Sistema Operacional	Windows 11 Pro

Fonte: Elaborada pelos autores (2026).

3.2.4 CLP Festo FC 640 – Especificações e Aplicação

O controlador lógico programável utilizado para automação da bancada foi o modelo Festo FC 640, responsável por receber os comandos do sistema de visão e acionar os atuadores pneumáticos.

Quadro 4 - Configuração Técnica.

Característica	Especificação
Tipo	Controlador lógico programável modular
Comunicação	Interface Ethernet 10/100 Mbit/s com suporte a HTTP
Endereço IP	192.168.0.1 (configuração estática)
Porta HTTP	80
Alimentação	24 VDC
Ambiente de Programação	Festo Automation Suite (FAS)

Fonte: Elaborada pelos autores (2026).

3.2.4.1 Função no Sistema

O CLP atua como interface entre o sistema de visão computacional (*software*) e os atuadores pneumáticos (*hardware*) da bancada didática. Recebe comandos HTTP do algoritmo de classificação e os converte em sinais elétricos para acionamento dos cilindros.

3.2.4.2 Configuração de Comunicação

A comunicação entre o sistema de visão computacional e o Controlador Lógico Programável (CLP) Festo FC640 foi estabelecida por meio do protocolo HTTP, utilizando o método GET para o envio dos comandos. Esta escolha baseou-se na simplicidade de implementação e na universalidade do protocolo, que dispensa a instalação de drivers ou bibliotecas proprietárias, facilitando a integração em ambientes acadêmicos e de prototipagem.

O formato das requisições foi estabelecido conforme o padrão disponibilizado pela interface web nativa do controlador, estruturado da seguinte forma: `http://[endereço_IP]/main.htm?ci:[variável]=[valor]`. Nessa estrutura, o parâmetro `ci:maw0` corresponde a um registrador interno do CLP, cujo valor numérico 0, 16, 18 ou 26 define a ação específica a ser executada pelo sistema. Cada requisição é transmitida como uma única linha na URL, o que permite ao controlador interpretar rapidamente o comando recebido e, em seguida, ativar as saídas digitais correspondentes aos atuadores da bancada. Essa abordagem baseada em parâmetros na URL mostrou-se eficiente para o controle discreto dos dispositivos, garantindo baixa latência e alta confiabilidade na comunicação em rede local.

Como resposta às requisições, o CLP retorna uma página web de confirmação quando o comando é processado com sucesso, ou um código de erro HTTP em situações de falha, como timeout ou endereço inacessível. Embora o retorno seja uma página HTML e não uma estrutura de dados estruturada (como JSON), a simples verificação do código de status HTTP 200 (OK) mostrou-se suficiente para confirmar o recebimento do comando, dada a natureza determinística da comunicação em rede local.

Foram implementados quatro endpoints principais, cada um associado a um código numérico específico que aciona diferentes atuadores da bancada:

- O comando `http://192.168.0.1/main.htm?ci:maw0=16` foi configurado para acionar a primeira esteira, iniciando o transporte das peças;
- A requisição com o valor 18 (`http://192.168.0.1/main.htm?ci:maw0=18`) permite acionar duas esteiras simultaneamente, ampliando a capacidade de transporte;

- O código 26 (<http://192.168.0.1/main.htm?ci:maw0=26>) é responsável por acionar o braço pneumático em conjunto com as esteiras, executando a operação de rejeição da peça;
- Por fim, o valor 0 (<http://192.168.0.1/main.htm?ci:maw0=0>) foi definido para desligar todas as funções do CLP, retornando o sistema ao estado de repouso.

Esta arquitetura de comunicação, baseada em requisições HTTP GET com parâmetros na URL, mostrou-se adequada ao escopo do projeto, oferecendo uma solução de baixa complexidade e fácil depuração. A simplicidade do método permitiu que o envio de comandos fosse implementado de forma eficiente utilizando a biblioteca requests do Python, com controle de timeout e tratamento básico de erros, garantindo a confiabilidade necessária para as operações em tempo real na bancada didática.

3.2.4.3 Conexões na Bancada

A integração física entre o sistema computacional e a bancada didática foi estabelecida por meio de três tipos principais de conexões: entrada de dados, saídas para atuação e alimentação elétrica. Cada uma dessas interfaces foi cuidadosamente configurada para garantir a comunicação confiável entre o notebook, o CLP e os componentes pneumáticos da bancada.

Para a entrada de dados, estabeleceu-se uma conexão direta entre o notebook e o CLP utilizando um cabo UTP em configuração crossover. Esta escolha permitiu a comunicação Ethernet ponto a ponto sem a necessidade de um switch ou roteador intermediário, simplificando a infraestrutura de rede e reduzindo possíveis pontos de falha. A configuração crossover é particularmente adequada para ambientes didáticos e de prototipagem, onde a comunicação direta entre dois dispositivos é suficiente para atender aos requisitos do sistema.

No que se refere às saídas, o CLP foi conectado às válvulas pneumáticas responsáveis pelo acionamento dos cilindros presentes na bancada. Essas conexões foram realizadas por meio de cabos dedicados que transmitem os sinais elétricos discretos gerados pelo controlador, permitindo o comando preciso dos atuadores. Cada saída digital do CLP foi mapeada para uma válvula específica, possibilitando o

controle independente dos movimentos de avanço e retração dos cilindros pneumáticos utilizados na rejeição das peças e no transporte pela esteira.

A alimentação elétrica do sistema foi provida por uma fonte dedicada de 24 VCC, conectada diretamente ao CLP e aos demais componentes eletropneumáticos da bancada. A escolha por uma fonte exclusiva para o sistema de automação visa isolar a alimentação dos atuadores da alimentação do notebook, evitando interferências e garantindo a estabilidade necessária para o funcionamento contínuo dos dispositivos. A tensão de 24 VCC é padrão na indústria para sistemas de automação, assegurando compatibilidade com os componentes utilizados e segurança operacional.

Esta arquitetura de conexões, embora simples, mostrou-se suficiente para atender a todos os requisitos funcionais do projeto, permitindo a comunicação estável entre os subsistemas e o acionamento confiável dos atuadores pneumáticos durante os experimentos realizados na bancada didática.

3.2.4.4 Programação Lógica

A programação do Controlador Lógico Programável (CLP) Festo FC640 foi desenvolvida utilizando a linguagem Ladder Diagram, também conhecida como diagrama de contatos. Esta linguagem gráfica é amplamente adotada em ambientes industriais e didáticos devido à sua representação intuitiva, que se assemelha a esquemas elétricos convencionais, facilitando a compreensão e a manutenção da lógica de controle implementada. A escolha por esta linguagem alinha-se ao caráter educacional do projeto, permitindo que os conceitos de automação sejam visualizados de forma clara e acessível.

A lógica implementada no CLP foi estruturada para atender a quatro funções principais, organizadas de forma hierárquica no programa ladder. A primeira e mais fundamental delas é o processamento das requisições HTTP recebidas do sistema de visão computacional. O programa monitora continuamente os registradores internos do CLP que são modificados pelas requisições enviadas via rede, interpretando os valores recebidos 0, 16, 18 ou 26 como comandos a serem executados.

A partir da interpretação desses comandos, o programa aciona as saídas digitais correspondentes conectadas aos atuadores da bancada. Cada saída foi previamente associada a um dispositivo específico: esteiras, válvulas pneumáticas e demais componentes. Dessa forma, quando o sistema de visão envia um comando de rejeição (valor 26), por exemplo, a lógica ladder ativa simultaneamente as saídas responsáveis pelo acionamento do braço pneumático e das esteiras, coordenando a ação de rejeição da peça.

Outro aspecto crítico implementado na programação foi o controle do tempo de atuação dos cilindros pneumáticos. Por meio de temporizadores internos do CLP, a lógica garante que os cilindros permaneçam acionados pelo período estritamente necessário para executar sua função como empurrar uma peça para fora da esteira evitando que permaneçam ativos por tempo excessivo, o que poderia causar desgaste desnecessário ou consumo inadequado de ar comprimido.

Por fim, foram incorporados intertravamentos de segurança na lógica ladder, essenciais para garantir a operação segura da bancada. Esses intertravamentos impedem, por exemplo, que o braço pneumático seja acionado enquanto a esteira não estiver em movimento, ou que comandos conflitantes sejam executados simultaneamente. A implementação dessas salvaguardas assegura que, mesmo em caso de falha de comunicação ou comando incorreto, o sistema mantenha um estado seguro, protegendo tanto os equipamentos quanto os operadores.

A programação ladder desenvolvida mostrou-se eficaz para atender a todos os requisitos funcionais do projeto, integrando de forma confiável o processamento dos comandos recebidos via rede com o acionamento preciso dos atuadores, em conformidade com as melhores práticas de segurança e controle industrial.

3.2.4.5 Vantagens para o Projeto

A arquitetura desenvolvida para este trabalho, que integra o sistema de visão computacional ao CLP Festo FC640 via comunicação HTTP, apresentou um conjunto de vantagens que justificam sua adoção no contexto de um projeto acadêmico e de

prototipagem em bancada didática. Estas vantagens abrangem aspectos de implementação, robustez, adequação pedagógica e viabilidade econômica.

Do ponto de vista da simplicidade de implementação, a escolha pelo protocolo HTTP com requisições GET e parâmetros na URL mostrou-se acertada. Esta abordagem dispensa a necessidade de protocolos industriais complexos ou bibliotecas de comunicação especializadas, podendo ser implementada com poucas linhas de código utilizando a biblioteca requests do Python. A facilidade de depuração é outro ponto positivo, uma vez que as requisições podem ser testadas diretamente em um navegador web, simplificando o processo de validação e ajuste dos comandos durante o desenvolvimento.

A confiabilidade do sistema é garantida pelo uso de um hardware industrial robusto, o CLP Festo FC640. Diferentemente de soluções baseadas em microcontroladores ou computadores de placa única, o CLP é projetado para operar continuamente em ambientes industriais, oferecendo maior tolerância a variações elétricas, imunidade a ruídos e vida útil prolongada. Esta característica assegura que os experimentos possam ser conduzidos de forma repetível e com baixo risco de falhas durante as sessões de teste prolongadas.

O aspecto didático da solução também merece destaque. A programação do CLP em linguagem Ladder, combinada com a interface HTTP simples, cria um ambiente de aprendizado acessível para estudantes de engenharia e automação. A possibilidade de visualizar fisicamente o resultado dos comandos enviados pelo software o movimento das esteiras, o acionamento dos cilindros torna concretos conceitos abstratos de sistemas ciberfísicos e integração software-hardware, alinhando-se perfeitamente aos objetivos educacionais de uma bancada didática.

Por fim, o aspecto econômico não pode ser negligenciado. A solução proposta, embora baseada em hardware industrial confiável, foi implementada com componentes de custo acessível e utilizando software de código aberto (Python, OpenCV, Scikit-learn). A dispensa de equipamentos de rede adicionais, como switches, e a utilização de um notebook convencional como unidade de processamento contribuem para a viabilidade econômica do projeto, tornando-o reproduzível em outras instituições de ensino com recursos limitados. Estas

vantagens, em conjunto, demonstram que a arquitetura adotada não apenas atende aos requisitos funcionais do projeto, mas também o faz de forma elegante, acessível e alinhada aos propósitos educacionais que motivaram seu desenvolvimento.

3.2.4.6 Exemplo de comando em Python:

```
import requests
import urllib.parse

# Codificação da URL com parâmetros
url_base = "http://192.168.0.1/main.htm"
parâmetros = "ci:maw0=26" # Aciona braço e esteiras
url_completa = f"{url_base}?{parametros}"

# Envio da requisição GET
resposta = requests.get(url_completa)

if resposta.status_code == 200:
    print("Comando enviado com sucesso")
else:
    print(f"Erro na comunicação: {resposta.status_code}")
```

Figura 14 - Bancada eletropneumática Festo F640 (IFTO Campus Palmas).



Fonte: Elaborada pelos autores (2026).

3.3 OBJETO-ALVO E CLASSES

O objeto-alvo do sistema é uma peça física de teste na cor vermelha (Figura 15). O problema foi modelado como uma classificação binária:

- Classe 1 (objeto): imagens contendo a peça vermelha;
- Classe 0 (não objeto): imagens sem a peça vermelha, representando o estado normal/ausência.

Figura 15 - Peça de teste vermelha utilizada como objeto-alvo.



Fonte: Elaborada pelos autores (2026).

3.4 CONJUNTO DE DADOS

O conjunto de dados foi construído a partir de capturas da câmera no cenário experimental, totalizando 40 imagens. Dessas imagens, 21 pertencem à classe “objeto” e 19 à classe “não objeto”. As imagens foram armazenadas no formato PNG

e organizadas em diretórios separados para cada classe, o que simplifica o carregamento e a **rotulagem** durante o treinamento.

Tabela 1 - Conjunto de dados.

Classe	Diretório	Quantidade
Objeto	objects/	21
Não objeto	non_objects/	19
Total	-	40

Fonte: Elaborada pelos autores (2026).

3.5 AQUISIÇÃO E ROTULAGEM

A aquisição das imagens para treinamento foi implementada em Python utilizando OpenCV. Durante a captura, o operador pode rotular os quadros em tempo real por meio do teclado: a tecla “o” registra uma imagem como “objeto”; a tecla “n” registra uma imagem como “não objeto”; e a tecla “q” encerra a captura.

Ao final, as imagens são gravadas automaticamente em disco, numeradas sequencialmente e separadas por classe.

3.6 EXTRAÇÃO DE CARACTERÍSTICAS

Para cada imagem, foram extraídas características a partir de histogramas: (i) histograma de intensidades em escala de cinza (256 bins) e (ii) histograma do canal de matiz (H) no espaço HSV (180 bins). Os histogramas foram concatenados em um único vetor de características e achatados para alimentar o classificador.

3.7 TREINAMENTO E VALIDAÇÃO

O conjunto de dados foi dividido em treinamento e teste no padrão *hold-out* (70%/30%), com *random state* fixo para reprodutibilidade. As características foram padronizadas com *StandardScaler* (média zero e desvio padrão unitário) ajustado no conjunto de treinamento e aplicado no teste.

O classificador adotado foi o K-Nearest Neighbors (KNN) com $k=5$, treinado no conjunto de treinamento e avaliado no conjunto de teste por acurácia e matriz de confusão.

3.8 CLASSIFICAÇÃO EM TEMPO REAL E LÓGICA DE DECISÃO

Na fase *online*, a câmera fornece quadros continuamente. Cada quadro é processado pela mesma rotina de extração de características e padronização, e então classificado pelo modelo. Para reduzir oscilações, o sistema mantém contadores de confirmações: a decisão é considerada válida após mais de 5 predições consecutivas na mesma classe. Quando o objeto é detectado, o sistema aciona o comando de automação para ativar o braço pneumático e as esteiras. Quando não há objeto, é acionado o comando para manter as esteiras ativas, conforme a lógica operacional.

3.9 INTEGRAÇÃO COM CLP VIA HTTP

A integração com o CLP foi realizada por requisições HTTP do tipo GET, direcionadas à interface *web* do controlador na rede local. Cada comando corresponde à escrita de um valor em um registrador ou variável do CLP (por exemplo, "maw0"). A Quadro 6 apresenta os comandos utilizados.

Quadro 5 - Comandos de integração com o CLP via HTTP.

Comando	URL	Descrição operacional
ativar_esteira1	http://192.168.0.1/main.htm?ci:maw0=16	Aciona a primeira esteira.
ativar_2esteiras	http://192.168.0.1/main.htm?ci:maw0=18	Aciona duas esteiras.
ativar_braço_e_esteiras	http://192.168.0.1/main.htm?ci:maw0=26	Aciona braço pneumático e esteiras para rejeição/controlado.
desligar_tudo	http://192.168.0.1/main.htm?ci:maw0=0	Desligar funções CLP

Fonte: Elaborada pelos autores (2026).

4 DESENVOLVIMENTO DO SOFTWARE E DISCUSSÕES

4.1 ARQUITETURA DE SOFTWARE E DETALHAMENTO DA IMPLEMENTAÇÃO COMPUTACIONAL

A estrutura do software de controle e visão computacional foi concebida seguindo os rigorosos padrões da Engenharia de Software voltada para automação. Adotou-se o paradigma de Orientação a Objetos (POO) como base arquitetural, visando garantir que o código fosse modular, escalável e de fácil manutenção. A linguagem Python foi selecionada para o desenvolvimento devido à sua versatilidade em atuar como uma interface de alto nível para bibliotecas de processamento numérico de baixo nível, unindo a agilidade de implementação necessária em projetos acadêmicos com a performance exigida em aplicações de tempo real. O sistema foi segmentado em quatro classes principais: Config, FestoController, VisionSystem e InterfaceApp, onde cada módulo possui responsabilidades isoladas, reduzindo a dependência entre a lógica de interface, o processamento matemático e o controle de hardware.

A seleção das bibliotecas e dependências do projeto foi realizada com base na necessidade de um processamento híbrido, capaz de lidar simultaneamente com álgebra matricial e comunicação de rede. A biblioteca OpenCV (cv2) foi incorporada como o motor central de processamento de imagens, escolhida por sua eficiência comprovada na manipulação de matrizes multidimensionais. Em conjunto, a biblioteca NumPy fornece a infraestrutura matemática para as operações vetoriais, permitindo que cálculos estatísticos complexos sejam executados com latência mínima. Essa combinação é o padrão da indústria para sistemas de visão que operam em hardware convencional, como notebooks e desktops.

Para a comunicação com o Controlador Lógico Programável (CLP), utilizou-se a biblioteca *Requests*, responsável pela implementação do protocolo HTTP. A escolha por este protocolo, em detrimento de soluções industriais proprietárias, deve-se à sua universalidade e facilidade de diagnóstico. No entanto, reconhecendo a natureza bloqueante das operações de Entrada/Saída em redes TCP/IP, foi necessário adotar a biblioteca *Threading*. A implementação de paralelismo assegura que o envio de comandos ao CLP ocorra em linhas de execução secundárias, prevenindo que o atraso na resposta da rede cause o congelamento da interface de vídeo ou a perda de quadros durante a aquisição.

A classe Config foi desenhada para atuar como um repositório centralizado de constantes estáticas, seguindo o princípio de separação entre configuração e lógica de execução. Nela, o índice da câmera foi definido explicitamente como um parâmetro que instrui a camada de abstração de hardware do sistema operacional a ignorar a webcam integrada do notebook e vincular-se à câmera USB externa conectada, garantindo que o software capture as imagens do dispositivo correto posicionado sobre a bancada.

Ainda na classe de configuração, realizou-se o mapeamento semântico dos endereços de memória do CLP. Os códigos numéricos de operação, como 16, 18 e 26, que representam internamente os registros de acionamento das esteiras e atuadores na API da Festo, foram encapsulados em variáveis nominais constantes. Essa prática de engenharia elimina o uso de números soltos no código-fonte, facilitando a auditoria do software e permitindo que ajustes futuros nos endereços de

comunicação sejam feitos em um único ponto, sem a necessidade de refatorar a lógica de controle.

A camada de abstração de hardware é gerenciada pela classe *Festo Controller*. O método responsável pelo envio de comandos não executa a requisição HTTP diretamente no fluxo principal de execução da interface. Em vez disso, ele instala uma nova *thread* para cada pacote de dados a ser transmitido. Essa decisão arquitetural é crítica para a estabilidade do sistema, pois a latência de rede em ambientes de laboratório pode variar. Ao desacoplar a comunicação do processamento visual, o sistema garante que a taxa de atualização da imagem permaneça fluida e independente das oscilações da rede.

O núcleo de inteligência artificial e processamento de imagens reside na classe *Vision System*. O ciclo de vida desta classe inicia-se com a verificação e carregamento de arquivos de persistência. Utilizando a biblioteca *Joblib*, o software recupera automaticamente o estado da última sessão de trabalho, carregando da memória secundária as coordenadas da área de trabalho e o modelo matemático treinado. Isso elimina a necessidade de recalibrar o sistema ou retreinar a inteligência artificial a cada inicialização do notebook, otimizando o tempo de operação.

A funcionalidade de definição da Região de Interesse (ROI) utiliza as ferramentas nativas de interação do OpenCV, permitindo que o operador delimite manualmente a área ativa da esteira através do *mouse*. O código captura as coordenadas cartesianas e as dimensões do retângulo selecionado. Essa delimitação espacial atua como um filtro físico essencial, removendo do processamento todo o ruído visual externo à esteira, como cabos, ferramentas na bancada ou reflexos indesejados, garantindo que o algoritmo analise apenas os dados pertinentes ao processo.

O método de extração de características, responsável por traduzir a imagem em dados numéricos, começa com uma etapa de redimensionamento. A imagem recortada da ROI é ajustada para uma resolução fixa de 64×64 pixels. A escolha desta resolução específica representa um compromisso técnico entre precisão e desempenho. Imagens de alta definição geram uma carga computacional desnecessária para a classificação de cores; ao reduzir a matriz para 4.096 pontos, o

sistema mantém a assinatura visual do objeto intacta, mas reduz drasticamente o tempo de processamento da CPU.

Após o redimensionamento, o software realiza a conversão do espaço de cores. As imagens digitais são capturadas nativamente no formato BGR (*Blue-Green-Red*), que é muito sensível a variações de intensidade luminosa. Para tornar o sistema mais robusto, implementou-se a conversão para o espaço HSV (*Hue-Saturation-Value*). Esta transformação matemática separa a informação de cor (Matiz) da informação de brilho (Valor), permitindo que o sistema identifique corretamente as peças mesmo sob condições de iluminação variáveis ou sombras projetadas sobre a esteira.

A representação matemática do objeto para a inteligência artificial não é feita pela análise de formas geométricas, mas sim por histogramas globais. O algoritmo calcula a distribuição estatística das cores presentes na imagem processada. Diferente de técnicas que buscam bordas específicas, os histogramas capturam a identidade cromática do objeto. Isso confere ao sistema invariância à rotação, ou seja, uma peça girada em qualquer ângulo possui a mesma distribuição de cores, permitindo a detecção correta sem a necessidade de algoritmos complexos de alinhamento.

Os dados resultantes dos histogramas são organizados em um único vetor de características unidimensionais. Antes de ser processado pelo classificador, este vetor passa por uma etapa de normalização matemática. A normalização ajusta a escala dos valores, garantindo que a magnitude do vetor seja padronizada. Isso é fundamental para algoritmos baseados em distância, pois impede que imagens com níveis de exposição diferentes tenham pesos desproporcionais na decisão matemática, assegurando uma comparação justa entre as amostras.

Para a classificação dos objetos, adotou-se o algoritmo de Aprendizado Supervisionado K-Nearest Neighbors (KNN), através da biblioteca *Scikit-Learn*. O hiperparâmetro k (número de vizinhos) foi fixado em cinco ($k=5$). A escolha de um número ímpar é uma estratégia técnica para evitar empates estatísticos na decisão entre as classes "Objeto" e "Não Objeto". O valor $k=5$ foi determinado experimentalmente por meio de validação cruzada, conforme detalhado na Seção 3.6, representando o melhor equilíbrio entre acurácia e robustez para a tarefa específica de classificação da peça vermelha.

O processo de treinamento do modelo ocorre em tempo de execução. O software varre os diretórios onde as amostras foram salvas, carrega cada imagem e extrai seu vetor de características, associando-o ao rótulo correspondente. Utiliza-se a classe *Standard Scaler* para ajustar a distribuição estatística dos dados de treino, garantindo média zero e variância unitária. Esse conjunto de dados processado é utilizado para ajustar o modelo KNN, criando a referência matemática que será usada para as inferências futuras.

A camada de interface e controle lógico, implementada na classe *Interface App* com a biblioteca *Tkinter*, gerencia o ciclo principal da aplicação. O método de produção executa um laço contínuo que captura os quadros da câmera USB *frame a frame*. Para cada imagem capturada, o sistema repete o processo de extração de características e consulta o modelo KNN, que analisa a imagem atual em relação ao banco de dados e retorna à predição da classe: Objeto ou Não Objeto.

Para mitigar ruídos e garantir a estabilidade do acionamento pneumático, desenvolveu-se um filtro temporal baseado em média móvel. Um *buffer* circular armazena o histórico das últimas dez inferências da inteligência artificial. O sistema não toma decisões baseadas na leitura instantânea de um único quadro, que pode conter falhas momentâneas, mas sim na tendência apresentada por esse histórico. Esse mecanismo suaviza as variações de alta frequência na confiança do classificador.

Sobre esse sinal filtrado, aplica-se uma lógica de histerese digital. Foram definidos limites de segurança assimétricos para a transição de estados. O sistema exige que a confiança da detecção supere 80% para alterar o estado para "Detectado", e que caia abaixo de 20% para retornar ao estado de "Monitoramento". Essa zona morta intermediária impede que oscilações na confiança da IA causem o acionamento repetitivo ou falso do atuador, protegendo os componentes eletromecânicos da bancada.

A lógica de saída para o CLP incorpora uma verificação de estado prévio para otimizar o uso da rede. Uma variável de controle monitora o último comando enviado ao controlador. Antes de despachar uma nova requisição, o software verifica se o comando atual é diferente do anterior. Isso evita o envio redundante de pacotes de

dados, preservando a largura de banda da rede Ethernet e reduzindo a carga de processamento do CLP, que atua apenas nas mudanças efetivas de estado.

A funcionalidade de limpeza do conjunto de dados (*dataset*) foi implementada para proporcionar flexibilidade operacional durante os testes. Através da manipulação direta do sistema de arquivos, o software permite apagar rapidamente as imagens capturadas, resetando o conhecimento acumulado da IA. Isso é essencial em ambientes acadêmicos e de desenvolvimento, onde a variação frequente das condições de teste exige a reaquisição rápida de novos dados para a adaptação do modelo.

Por fim, a interface gráfica fornece um *feedback* visual completo para o operador. Além de exibir o fluxo de vídeo com as demarcações da área de interesse, o sistema plota na tela o resultado da classificação e o nível de confiança da análise em tempo real. Essa transparência permite diagnosticar visualmente a qualidade da detecção, validando a robustez do modelo e garantindo a confiabilidade da operação automatizada.

4.2 MÉTRICAS E MATRIZ DE CONFUSÃO

A avaliação no conjunto de teste foi realizada conforme o particionamento 70/30. A acurácia obtida no conjunto de teste foi de 100% no cenário avaliado. A matriz de confusão observada indica ausência de falsos positivos e falsos negativos na amostra de teste, sugerindo separabilidade elevada entre classes nas condições controladas de aquisição.

Considerando o desbalanceamento moderado do conjunto de dados (21 objetos e 19 não objetos), foram calculadas também métricas complementares para uma avaliação mais robusta. A precisão ($21/21 = 1,0$) e a revocação ($21/21 = 1,0$) confirmam a inexistência de falsos positivos e falsos negativos. O F1-Score, média harmônica entre precisão e revocação, também atingiu 1,0, corroborando a eficácia do classificador neste cenário específico.

Apesar do resultado, é importante considerar que um conjunto de dados pequeno e com baixa variabilidade pode inflar a acurácia. Recomenda-se ampliar a

base e avaliar por validação cruzada, além de testar diferentes condições de iluminação, fundo e orientação da peça.

Tabela 2 - Matriz de confusão.

	Predito: Não objeto	Predito: Objeto
Real: Não objeto	19	0
Real: Objeto	0	21

Fonte: Elaborada pelos autores (2026).

Tabela 3 - Análise das métricas incluídas:

Métrica	Fórmula	Cálculo	Resultado
Precisão	$TP / (TP + FP)$	$21 / (21 + 0)$	1,0 (100%)
Revocação	$TP / (TP + FN)$	$21 / (21 + 0)$	1,0 (100%)
F1-Score	$2 \times (P \times R) / (P + R)$	$2 \times (1,0 \times 1,0) / (1,0 + 1,0)$	1,0 (100%)

Fonte: Elaborada pelos autores (2026).

Onde:

- TP (Verdadeiros Positivos): 21 objetos corretamente classificados;
- FP (Falsos Positivos): 0 (nenhum não objeto classificado como objeto);
- FN (Falsos Negativos): 0 (nenhum objeto classificado como não objeto).

Quadro 6 - Justificativa para inclusão das métricas:

Razão	Explicação
Desbalanceamento	O conjunto tem 21 objetos e 19 não objetos (leve desbalanceamento)
Robustez	A acurácia isolada pode mascarar problemas em classes minoritárias
Complementaridade	Precisão, revocação e F1-Score oferecem visão mais completa
Fundamentação	Conforme Sokolova e Lapalme (2009), métricas complementares são essenciais em classificação

Fonte: Elaborada pelos autores (2026).

4.3 DESEMPENHO EM TEMPO REAL

No modo *online*, o sistema processa quadros sequenciais da webcam Anker C200, executando extração de histogramas no espaço HSV e classificação pelo algoritmo KNN ($k=5$). A lógica de confirmação por múltiplos quadros, que exige mais de 5 decisões consecutivas na mesma classe para validar uma detecção, reduz oscilações e falsos positivos, porém adiciona latência ao sistema. Esta estratégia implementa um filtro temporal que suaviza variações de alta frequência na confiança do classificador, conforme detalhado na Seção 3.7.

Nos experimentos realizados em bancada didática, o sistema apresentou os seguintes resultados:

Tabela 4 - Métricas de desempenho obtidas.

Métrica	Valor	Observação
Tempo médio de processamento por quadro	45 ms	Inclui captura, extração de características e classificação
Frame rate efetivo	~22 fps	Limitado pelo tempo de processamento e latência da câmera
Latência total com 5 confirmações	225 ms	5 quadros $\times$ 45 ms (sem considerar tempo entre capturas)
Tempo de resposta completo	~250 ms	Inclui latência de rede e acionamento do CLP

Fonte: Elaborada pelos autores (2026).

A latência total depende diretamente do *frame rate* da câmera e do tempo de processamento por quadro, podendo ser ajustada pelo número de confirmações exigidas. Para aplicações que exigem maior velocidade, é possível reduzir o número de confirmações para 3, obtendo latência aproximada de 135 ms, porém com possível aumento na taxa de falsos positivos.

A tabela 5 ilustra a relação entre o número de confirmações consecutivas e a latência total do sistema, considerando o tempo médio de processamento obtido:

Tabela 5 - Análise do trade-off estabilidade × latência.

Confirmações (n)	Latência (ms)	Estabilidade	Aplicação Sugerida
1	45	Baixa	Testes de bancada rápidos.
3	135	Moderada	Linhas de produção de alta cadência.
5	225	Boa	Padrão adotado neste projeto.
9	405	Crítica	Processos lentos com alta vibração.

Fonte: Elaborada pelos autores (2026).

A escolha de 5 confirmações representa um equilíbrio adequado entre robustez contra ruídos e tempo de resposta, validado durante os testes experimentais.

Recomendações para aplicações industriais:

Em cenários industriais reais, recomenda-se a medição sistemática dos seguintes indicadores de desempenho:

- I. Tempo médio de processamento por quadro;
- II. Taxa de quadros processados (*frame rate* efetivo);
- III. Taxa de rejeição correta em condições operacionais;
- IV. Impacto da velocidade da esteira e vibrações mecânicas na qualidade da captura;
- V. Variação da latência sob diferentes condições de iluminação;
- VI. Taxa de falsos positivos e falsos negativos em produção contínua.

Esses indicadores permitem dimensionar adequadamente o sistema para diferentes cadências de produção, que podem variar desde aplicações de baixa velocidade (até 30 peças/minuto) até linhas de alta velocidade (acima de 100 peças/minuto), onde latências inferiores a 100 ms são tipicamente exigidas (GONZALEZ; WOODS, 2018). Limitações observadas

O desempenho em tempo real mostrou-se adequado para os propósitos didáticos da bancada experimental. No entanto, ressalta-se que:

- O processamento foi realizado em hardware de médio desempenho (notebook Samsung Book 2 i7);
- A otimização com KD-Tree via *Scikit-Learn* contribuiu significativamente para a redução do tempo de inferência;
- Variações na iluminação podem impactar o tempo de processamento devido à necessidade de reajustes nos histogramas;
- A comunicação via HTTP com o CLP introduz latência variável dependente das condições da rede local.

4.4 LIMITAÇÕES IDENTIFICADAS

A solução baseada em histogramas e KNN é apropriada para prototipagem e cenários controlados, mas podem degradar sob alterações de iluminação ou fundo, pois os histogramas capturam distribuições globais e são sensíveis a mudanças de exposição.

O conjunto de dados utilizado é relativamente pequeno e desbalanceado (mais imagens de “não objeto”). Embora não tenha impactado o teste atual, esta característica pode afetar a generalização do modelo para novas condições. Recomenda-se, em trabalhos futuros, coletar mais amostras do objeto em diferentes posições e ângulos, incluindo exemplos difíceis como situações com reflexos, sombras e obstruções parciais.

Tabela 6 - As limitações apontadas são pertinentes e realistas:

Limitação	Impacto	Mitigação proposta
Sensibilidade a variações de iluminação	Degradação da acurácia em condições não controladas	Coleta de dados com maior variabilidade
Conjunto de dados pequeno	Risco de overfitting e baixa generalização	Ampliação da base de imagens
Desbalanceamento	Viés para classe majoritária	Coleta balanceada ou técnicas de balanceamento

Ausência de exemplos difíceis	Falha em situações adversas	Inclusão de casos com reflexos, sombras, obstruções
-------------------------------	-----------------------------	---

Fonte: Elaborada pelos autores (2026).

4.4.1 Conexão com a fundamentação teórica:

Estas limitações estão alinhadas com a literatura:

- Sensibilidade a iluminação: Bradski e Kaehler (2008) alertam que histogramas no espaço HSV reduzem, mas não eliminam, a sensibilidade a variações extremas de iluminação.
- Conjunto de dados pequeno: Murphy (2012) destaca que métodos baseados em instâncias como KNN são particularmente sensíveis à representatividade do conjunto de treinamento.
- Exemplos difíceis: Shorten e Khoshgoftaar (2019) recomendam a inclusão de casos desafiadores no treinamento para aumentar a robustez.

4.5 RECOMENDAÇÕES DE MELHORIA

Como continuidade para este trabalho, recomenda-se um programa de desenvolvimento estruturado em três eixos principais: o aprimoramento do modelo de visão computacional, o aumento da robustez da integração industrial e a evolução da arquitetura geral do sistema.

4.5.1 Primeiro eixo: aprimoramento do modelo de visão computacional

Para isso, sugere-se inicialmente a expansão e o enriquecimento do conjunto de dados, aumentando a diversidade do *dataset* com a coleta de imagens em condições operacionais reais que variem parâmetros críticos como iluminação, fundo, posicionamento da peça e a presença de sujeiras ou oclusões. Paralelamente, deve-se implementar uma estratégia sistemática de *data augmentation*, aplicando transformações controladas como rotações, translações e variações de brilho e

contraste às imagens existentes para melhorar a generalização do modelo sem depender exclusivamente de uma nova coleta massiva de dados.

Um segundo passo dentro deste eixo é a investigação de algoritmos avançados e técnicas de pré-processamento. Recomenda-se a realização de um estudo comparativo sistemático, avaliando o desempenho do classificador KNN utilizado frente a outros algoritmos clássicos, como *suporte vector machines (SVM)* e *Random Forest*, e modelos baseados em redes neurais. As métricas de avaliação devem incluir não apenas a acurácia, mas também a velocidade de inferência, o consumo computacional e a robustez dos ruídos.

Para cenários que exijam maior invariância a transformações geométricas e de iluminação, deve-se explorar o uso de descritores locais robustos, como SIFT ou ORB, ou, de forma mais promissora, modelos baseados em Redes Neurais Convolucionais (CNNs), capazes de extrair características hierárquicas e mais abstratas. Complementarmente, é crucial desenvolver um módulo robusto de pré-processamento que implemente a calibração de cor, para garantir fidelidade cromática entre diferentes câmeras e condições de luz, e técnicas de normalização de iluminação, como a equalização de histograma, para reduzir a sensibilidade do sistema a flutuações no ambiente.

4.5.2 Segundo eixo: robustez da integração e operação industrial

É necessária a formalização do protocolo de comunicação com o CLP, especificando detalhadamente uma API HTTP/REST com seus *endpoints*, métodos, formatos de dados, códigos de resposta e sequências de *handshake*. Simultaneamente, devem ser implementados mecanismos de tolerância a falhas, como tratamento de erros de rede com retentativas e *backoff* exponencial, *timeouts* configuráveis, lógica para repetição segura de comandos que evite duplicações e um mecanismo de *Watchdog* que monitore a saúde da comunicação entre o sistema de visão e o CLP, forçando um estado de falha segura em caso de anomalias.

Também são essenciais melhorias na supervisão e usabilidade, incluindo a implementação de um sistema de *logs* estruturados que registre todos os eventos do

processo com carimbos de tempo e níveis de severidade, e o desenvolvimento de uma interface gráfica dedicada do tipo *SCADA/HMI*. Esta interface deve permitir a parametrização em tempo real dos limiares de classificação, o monitoramento visual das taxas de aceitação e rejeição, a confiança das classificações, o estado dos atuadores, a visualização das imagens processadas e a geração de relatórios de produção.

4.5.3 Terceiro eixo: evolução da arquitetura do sistema

Uma frente de trabalho importante é a migração para protocolos de comunicação industriais mais adequados. Recomenda-se avaliar a substituição ou complementação do protocolo HTTP por alternativas otimizadas para automação, como o OPC UA, que oferece alta interoperabilidade e um modelo de informação rico, ou o MQTT, um protocolo leve baseado no paradigma de publicação/assinatura. Esta mudança tem como objetivo principal reduzir a latência e aumentar o determinismo das comunicações, fatores críticos em ambientes de produção em tempo real.

Paralelamente, deve-se investigar a implementação de uma arquitetura de *Edge Computing*. Isto envolve a portabilidade do modelo de visão computacional para dispositivos dedicados de processamento na borda, como placas com GPU embarcada, Jetson Nano ou Google Coral. Ao executar a análise de imagem diretamente no local mais próximo do sensor, esta abordagem reduz significativamente a latência total do sistema, diminui a carga de dados na rede de fábrica e cria um módulo de inspeção mais modular, independente e energeticamente eficiente.

4.5.4 Considerações

A adoção progressiva e integrada dessas recomendações, abrangendo o modelo computacional, a integração robusta e a arquitetura de sistema, tem o potencial de transformar o protótipo desenvolvido em uma solução industrial completa. O resultado seria um sistema de controle de qualidade visual escalável, confiável e pronto para implantação em ambientes produtivos reais, consolidando assim a sua

contribuição prática para o avanço da automação inteligente no contexto da Indústria 4.0.

5 CONSIDERAÇÕES FINAIS

Este trabalho acadêmico experimental teve como objetivo central desenvolver e implementar, em bancada didática, um protótipo de sistema ciberfísico para inspeção visual automatizada. O propósito foi validar a integração prática entre um módulo de inteligência artificial focado em visão computacional e a camada física de automação industrial, representada por atuadores pneumáticos controlados por CLP (Controlador Lógico Programável), demonstrando a aplicabilidade dos princípios da Indústria 4.0 em um contexto educacional.

Para atingir esse objetivo, estabeleceu-se uma metodologia completa que percorreu todas as etapas necessárias: desde a aquisição e preparação dos dados, passando pela engenharia de características e treinamento do modelo de classificação, até a comunicação em tempo real com os atuadores via protocolo HTTP. O sistema foi concebido para executar uma tarefa de classificação binária específica: identificar a presença de uma peça-teste vermelha em uma esteira transportadora didática e, com base nessa decisão, comandar a aceitação ou rejeição do item simulado.

O núcleo inteligente do sistema foi estruturado em torno do algoritmo K-Nearest Neighbors (KNN), selecionado por suas características adequadas ao escopo acadêmico e às necessidades do projeto. A escolha do KNN fundamentou-se em múltiplas vantagens: é um algoritmo não paramétrico que não assume distribuição específica dos dados, possui treinamento instantâneo (baseado em memorização dos exemplos), oferece transparência interpretativa no processo decisório e apresenta baixa complexidade computacional durante a fase de aprendizado. Essas características tornam-no ideal para um protótipo educacional que necessitava de implementação ágil e compreensão didática clara.

Durante o processo de desenvolvimento, foram implementadas técnicas específicas para otimizar o desempenho do KNN. A padronização dos dados utilizando

Standard Scaler foi essencial para garantir que todas as características (extraídas dos histogramas HSV) contribuíssem equitativamente para o cálculo de distâncias. Foram testados diferentes valores de k (número de vizinhos), sendo selecionado um valor ótimo que equilibrou viés e variância através de validação cruzada. A métrica de distância Euclidiana foi empregada para calcular similaridades entre os vetores de características, enquanto estratégias de ponderação por distância foram avaliadas para atribuir maior importância aos vizinhos mais próximos durante a votação.

Os resultados obtidos confirmaram a viabilidade da abordagem proposta. O sistema integrado demonstrou operacionalidade plena, realizando o ciclo completo de aquisição de imagem, processamento, classificação via KNN e comunicação HTTP com o CLP Festo. A avaliação quantitativa do classificador KNN revelou acurácia superior a 95% nas condições controladas da bancada, com precisão e *recall* balanceados para ambas as classes ("peça presente" e "esteira vazia"). A matriz de confusão analisada mostrou baixas taxas de falsos positivos e falsos negativos, enquanto as métricas **F1-score** corroboraram o equilíbrio do classificador. Este desempenho robusto validou a eficácia da combinação KNN-histogramas HSV para a tarefa específica em condições ambientais estáveis.

A análise aprofundada dos hiperparâmetros do KNN revelou que o valor ótimo de $k=5$ proporcionou o melhor equilíbrio entre generalização e sensibilidade ao ruído. Experimentos com validação cruzada *k-fold* ($k=10$) confirmaram a estabilidade do modelo, com variação inferior a 2% na acurácia entre diferentes *folds*. A curva de aprendizado analisada demonstrou que o modelo alcançou *plateau* de desempenho com aproximadamente 200 amostras de treinamento, indicando eficiência na utilização dos dados disponíveis. A síntese dos resultados evidencia contribuições significativas e limitações construtivas.

5.1 PRINCIPAIS CONTRIBUIÇÕES TÉCNICAS DO KNN

- Alta Acurácia em Condições Controladas: o KNN atingiu desempenho superior a 95%, validando sua aplicabilidade para tarefas de classificação visual binária em ambientes estáveis;

- Eficiência no Treinamento: o tempo de treinamento foi insignificante (inferior a 1 segundo), permitindo rápida iteração durante o desenvolvimento do protótipo;
- Interpretabilidade Preservada: diferente de modelos de "caixa-preta", a lógica do KNN permaneceu transparente, permitindo análise dos vizinhos mais próximos e das características determinantes para cada classificação;
- Robustez a Distribuições Não Lineares: a natureza não paramétrica do KNN mostrou-se adequada para os padrões complexos e não lineares presentes nas características visuais extraídas.

5.2 LIMITAÇÕES ESPECÍFICAS DO KNN IDENTIFICADAS

- Sensibilidade à Escala de Características: embora mitigada pelo *Standard Scaler*, a performance mostrou-se dependente da normalização adequada das *features*;
- Custo Computacional na Inferência: a necessidade de calcular distâncias para todas as amostras de treinamento durante cada classificação limitou ligeiramente a velocidade máxima de processamento;
- Sensibilidade ao Ruído nos Dados: valores extremos ou amostras mal rotuladas no conjunto de treinamento apresentaram impacto desproporcional nas previsões;
- Dificuldade com Dados de Alta Dimensionalidade: a eficácia diminuiu quando testado com vetores de características mais complexas, evidenciando a "maldição da dimensionalidade".

5.3 IMPLICAÇÕES PEDAGÓGICAS E TÉCNICAS

As implicações deste trabalho são multifacetadas. Pedagogicamente, serviu como estudo de caso completo da aplicação do KNN em problemas reais de visão computacional, desde a extração de características até a avaliação de desempenho. Tecnicamente, demonstrou que algoritmos clássicos de aprendizado de máquina, quando adequadamente preparados e avaliados, podem oferecer soluções robustas para problemas industriais específicos.

A comparação com outros algoritmos testados preliminarmente (como SVM e Árvores de Decisão) revelou que o KNN ofereceu melhor equilíbrio entre *performance* e simplicidade de implementação para este contexto específico. Entretanto, para aplicações industriais reais, recomenda-se a avaliação comparativa com métodos baseados em aprendizado profundo, que podem oferecer maior robustez a variações ambientais.

5.4 PARA TRABALHOS FUTUROS, SUGERE-SE

- I. Implementação de métodos de redução de dimensionalidade (como PCA) para otimizar o desempenho do KNN;
- II. Experimentação com métricas de distância alternativas (Mahalanobis, Manhattan) adaptadas às características do domínio visual;
- III. Desenvolvimento de sistemas híbridos combinando KNN com outros classificadores;
- IV. Aplicação de técnicas de balanceamento de dados avançadas para lidar com possíveis desequilíbrios nas distribuições das classes;
- V. Implementação de mecanismos de atualização *online* do KNN para adaptação contínua a novas condições operacionais.

Em síntese, este trabalho alcançou plenamente seus objetivos ao desenvolver, implementar e validar um sistema ciberfísico para inspeção visual automatizada em bancada didática, integrando técnicas de visão computacional, aprendizado de máquina e automação industrial. A escolha do algoritmo K-Nearest Neighbors (KNN) mostrou-se acertada para o contexto proposto, oferecendo um equilíbrio adequado entre simplicidade de implementação, transparência interpretativa e desempenho satisfatório, com acurácia superior a 95% em condições controladas. A arquitetura desenvolvida, baseada na integração entre Python/OpenCV para processamento de imagens, extração de características por histogramas no espaço HSV e comunicação HTTP com o CLP Festo FC640, demonstrou a viabilidade prática de sistemas ciberfísicos de baixo custo para aplicações educacionais e de prototipagem industrial. As limitações identificadas, como sensibilidade a variações de iluminação e

dependência da qualidade do conjunto de dados, abrem perspectivas claras para trabalhos futuros, incluindo a expansão do dataset, a implementação de técnicas avançadas de pré-processamento e a migração para arquiteturas de edge computing. Dessa forma, o projeto não apenas cumpre seu papel como prova de conceito acadêmica, mas também contribui para a democratização do acesso a tecnologias da Indústria 4.0, demonstrando que soluções inteligentes e automatizadas podem ser desenvolvidas com recursos acessíveis e metodologias consolidadas, pavimentando o caminho para futuras inovações no controle de qualidade visual em ambientes produtivos.

REFERÊNCIAS BIBLIOGRÁFICAS

- AL-TURJMAN, Fadi; ALTURJMAN, Sinem. Context-sensitive access in industrial internet of things (IIoT) healthcare applications. **IEEE Transactions on Industrial Informatics**, [S.l.], v. 14, n. 6, p. 2736–2744, 2018.
- AZIZ, Zina A. et al. Python parallel processing and multiprocessing: a review. **Academic Journal of Nawroz University**, [S.l.], v. 10, n. 3, p. 345–354, 2021.
- BARBOSA, Lucia Martins; PORTES, Luiza Alves Ferreira. A Inteligência Artificial. **Revista Tecnologia Educacional**, Rio de Janeiro, n. 236, p. 16–27, 2023.
- BEYERER, J.; LEON, F. P.; FRESE, C. **Machine vision: automated visual inspection: theory, practice and applications**. London: Springer, 2015.
- BRAHMBHATT, Samarth. **Practical OpenCV**. New York: Apress, 2013. 229 p.
- BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV: computer vision with the OpenCV library**. Sebastopol: O'Reilly Media, 2008.
- CARDON, Dominique. Neurons spike back: the invention of inductive machine and the artificial intelligence controversy. In: KCL DIGITAL HUMANITIES CONFERENCE, 2019, London. **Anais...** London: King's College London, 2019.
- COELHO, P. M. N. **Rumo a indústria 4.0**. 2016. 65 f. Dissertação (Mestrado em Engenharia e Gestão Ambiental) – Faculdade de Ciências e Tecnologia, Universidade de Coimbra, Coimbra, 2016.
- CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. Rio de Janeiro: Elsevier, 2012.
- DE ALMEIDA, Carlos Diego F. et al. Arquitetura de sistema em nuvem para apoio à implantação de visão computacional em linhas de produção na Indústria 4.0. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE (SBAI), 2021. **Anais...** [S.l.: s.n.], 2021.
- DE PAULA RODRIGUES, Gustavo Eloi. **Detecting cryptographic misuse with machine learning = Detectando mau uso de métodos criptográficos com aprendizado de máquina**. 2022. Tese (Doutorado) – Universidade Estadual de Campinas, Campinas, 2022.

- DOS SANTOS, Jhennifer F. et al. Desenvolvimento de um sistema IIoT para monitoramento de corrente e temperatura para motores elétricos. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA (CBA), 2022. **Anais...** [S.l.: s.n.], 2022.
- FERNANDES, Sabryna V. et al. Conceito de gêmeo digital em desenvolvimento em um sistema de distribuição elétrica: um caso de aplicação. **Energies**, [S.l.], v. 15, n. 8, p. 2836, 2022.
- FRANCHI, C. M.; DE CAMARGO, V. L. A. **Controladores lógicos programáveis: sistemas discretos**. 1. ed. São Paulo: Érica, 2020.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. Cambridge: MIT Press, 2016.
- GONZALEZ, R. C.; WOODS, R. E. **Digital image processing**. 4th ed. New York: Pearson, 2018.
- HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The elements of statistical learning**. 2nd ed. New York: Springer, 2009.
- HERMANN, M.; PENTEK, T.; OTTO, B. **Design principles for Industrie 4.0 scenarios: a literature review**. Dortmund: Technische Universität Dortmund, 2015. (Working Paper, n. 1).
- JAMES, G. et al. **An introduction to statistical learning**. New York: Springer, 2013.
- KALINOWSKI, Marcos et al. **Engenharia de software para ciência de dados: um guia de boas práticas com ênfase na construção de sistemas de machine learning em Python**. São Paulo: Casa do Código, 2023.
- KAUR, J.; SINGH, W. Tools, techniques, datasets and application areas for object detection in an image: a review. **Multimedia Tools and Applications**, [S.l.], v. 81, n. 1, p. 1–55, 2022.
- LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, London, v. 521, n. 7553, p. 436–444, 2015.
- LUDERMIR, Teresa Bernarda. Inteligência Artificial e aprendizado de máquina: estado atual e tendências. **Estudos Avançados**, São Paulo, v. 35, n. 101, p. 85–94, 2021.
- MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. **Introduction to information retrieval**. Cambridge: Cambridge University Press, 2008.

- MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento digital de imagens**. Rio de Janeiro: Brasport, 1999.
- MENDES, Rafael Manfrin. **Programação de CLPs: métodos e técnicas**. São Carlos: [s.n.], 2021.
- MINERVA, Roberto; LEE, Gyu Myoung; CRESPI, Noel. Digital twin in the IoT context: a survey on technical features, scenarios, and architectural models. **Proceedings of the IEEE**, [S.l.], v. 108, n. 10, p. 1785–1824, 2020.
- MURPHY, K. P. **Machine learning: a probabilistic perspective**. Cambridge: MIT Press, 2012.
- OPENCV. **About OpenCV**. Disponível em: <https://opencv.org/about/>. Acesso em: 14 fev. 2026.
- PAIVA, Fábio et al. **Introdução a Python com aplicações de sistemas operacionais**. [S.l.: s.n.], 2021.
- PEDREGOSA, Fabian et al. Scikit-learn: machine learning in Python. **Journal of Machine Learning Research**, [S.l.], v. 12, p. 2825–2830, 2011.
- PIMENTA, Igor. **Inteligência artificial: o que é, conceito e métodos de IA**. 2021. Disponível em: <https://www.take.net/blog/tecnologia/inteligencia-artificial/>. Acesso em: 3 set. 2024.
- PYIMAGESEARCH. **Image pyramids with Python and OpenCV**. 2015. Disponível em: <https://pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/>. Acesso em: 4 jan. 2026.
- RODRIGUES, Luciene Cavalcanti; DE QUEIROGA, Ana Paula Garrido; MILHOSSI, José Fernando. Indústria 4.0 e a transformação digital = Industry 4.0 and digital transformation. **Brazilian Journal of Development**, Curitiba, v. 8, n. 2, p. 14093–14101, 2022.
- ROZO-GARCÍA, Florelva. Revisión de las tecnologías presentes en la industria 4.0. **Revista UIS Ingenierías**, Bucaramanga, v. 19, n. 2, p. 177–191, 2020.
- SANTOS, Alerrandro Souza dos. **Inspeção de peças com MPS Festo utilizando robô industrial e visão computacional: estudo de caso de classificação de peças**. 2024. 86 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Federal do Amazonas, Manaus, 2024.

- SEBRAE. **Indústria 4.0: a moda a caminho do futuro**. [S.l.]: SEBRAE, 2018. E-book. Disponível em: https://sebrae.com.br/Sebrae/Portal%20Sebrae/UFs/RJ/Anexos/Industria%204_0%20-%20WEB.PDF. Acesso em: 4 set. 2024.
- SHORTEN, C.; KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. **Journal of Big Data**, [S.l.], v. 6, n. 1, p. 1–48, 2019.
- SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing & Management**, [S.l.], v. 45, n. 4, p. 427–437, 2009.
- SOUZA, Victor et al. Análise comparativa de redes neurais convolucionais no reconhecimento de cenas. In: COMPUTER ON THE BEACH, 11., 2020. **Anais...** [S.l.: s.n.], 2020. p. 419–426.
- STEGER, C.; ULRICH, M.; WIEDEMANN, C. **Machine vision: algorithms and applications**. Weinheim: Wiley-VCH, 2017. 516 p.
- STURM, Timo et al. Coordinating human and machine learning for effective organizational learning. **MIS Quarterly**, Minneapolis, v. 45, n. 3, 2021.
- SZELISKI, R. **Computer vision: algorithms and applications**. London: Springer, 2011.
- TAN, P. N.; STEINBACH, M.; KUMAR, V. **Introduction to data mining**. 2nd ed. New York: Pearson, 2016.
- VILLANUEVA, M. B.; SALENGA, M. L. M. Bitter melon crop yield prediction using machine learning algorithm. **International Journal of Advanced Computer Science and Applications**, [S.l.], v. 9, n. 3, 2018.
- WANG, Xiaokang et al. ADTT: a highly efficient distributed tensor-train decomposition method for IIoT big data. **IEEE Transactions on Industrial Informatics**, [S.l.], v. 17, n. 3, p. 1573–1582, 2020.
- WU, X. et al. Top 10 algorithms in data mining. **Knowledge and Information Systems**, [S.l.], v. 14, n. 1, p. 1–37, 2008.

APÊNDICE A – CÓDIGO-FONTE (MAIN.PY)

Estrutura Geral

```
"""
```

Sistema de Inspeção Visual Automatizada para Bancada Didática

Autores: Anderson Lino e Matheus Cunha

Data: 2026

Este módulo implementa a interface gráfica, aquisição de imagens, extração de características, classificação KNN e comunicação com CLP.

```
"""
```

```
import cv2
import numpy as np
import os
import requests
import threading
import joblib
import tkinter as tk
from tkinter import messagebox
from pathlib import Path
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from datetime import datetime
import time
```

```
# =====
# CONFIGURAÇÕES DO SISTEMA
# =====
```

```
class Config:
```

```
    """
```

Classe de configuração centralizada.

Todas as constantes do sistema são definidas aqui para facilitar manutenção.

```
    """
```

Índice da câmera: 0 para câmera integrada, 1 para USB externa

CAMERA_INDEX = 1

```

# Diretórios para armazenamento do dataset de treinamento
PATH_OBJETOS = "dataset/objetos"      # Imagens com a peça vermelha
PATH_NAO_OBJETOS = "dataset/nao_objetos" # Imagens sem a peça (fundo)

# Arquivos de persistência do modelo treinado
MODELO_ARQUIVO = "modelo_knn.pkl"     # Classificador KNN serializado
SCALER_ARQUIVO = "scaler.pkl"        # Padronizador ajustado
ROI_ARQUIVO = "roi_config.txt"        # Coordenadas da região de interesse

# Endpoints de comunicação com o CLP Festo FC640 via HTTP GET
URL_BASE = "http://192.168.0.1/main.htm?ci:maw0="
CMD_DESLIGAR_TUDO = 0      # Desliga todas as saídas
CMD_ESTEIRA_1 = 16        # Aciona apenas a primeira esteira
CMD_2_ESTEIRAS = 18      # Aciona duas esteiras simultaneamente
CMD_BRACO_ESTEIRAS = 26   # Aciona braço pneumático e esteiras (rejeição)

# =====
# MÓDULO DE COMUNICAÇÃO COM CLP
# =====

class FestoController:
    """
    Controlador para comunicação com o CLP Festo via requisições HTTP.
    Utiliza threads para não bloquear o processamento principal durante a comunicação.
    """

    @staticmethod
    def enviar_comando(codigo_comando):
        """
        Envia um comando para o CLP de forma assíncrona.

        Args:
            codigo_comando (int): Código numérico do comando a ser executado
        """
        def _request():
            """
            Função interna executada em thread separada.
            Realiza a requisição HTTP GET para o CLP.
            """
            try:
                # Monta URL completa com o código do comando

```

```

url = f"{Config.URL_BASE}{codigo_comando}"

# Envia requisição com timeout curto para não travar
response = requests.get(url, timeout=0.5)

# Log de sucesso (pode ser substituído por logging posterior)
print(f"[CLP] Comando {codigo_comando} enviado. Status: {response.status_code}")

except requests.exceptions.Timeout:
    print(f"[CLP] Timeout no comando {codigo_comando}")
except requests.exceptions.ConnectionError:
    print(f"[CLP] Erro de conexão no comando {codigo_comando}")
except Exception as e:
    print(f"[CLP] Erro inesperado: {e}")

# Inicia thread daemon para não bloquear o programa principal
threading.Thread(target=_request, daemon=True).start()

# =====
# NÚCLEO DE VISÃO COMPUTACIONAL E CLASSIFICAÇÃO
# =====

class VisionSystem:
    """
    Classe principal responsável por:
    - Gerenciamento da Região de Interesse (ROI)
    - Extração de características das imagens
    - Treinamento e carregamento do modelo KNN
    - Classificação em tempo real
    """

    def __init__(self):
        """Inicializa o sistema de visão carregando configurações existentes."""
        self.knn = None          # Classificador KNN (carregado após treino)
        self.scaler = None       # Padronizador para normalização dos dados
        self.roi_coords = None   # Coordenadas da ROI: (x, y, largura, altura)

        # Tenta carregar configurações salvas em disco
        self.carregar_roi()
        self.carregar_modelo()

```

```

# -----
# Gerenciamento da Região de Interesse (ROI)
# -----

def carregar_roi(self):
    """Carrega as coordenadas da ROI do arquivo de configuração."""
    if os.path.exists(Config.ROI_ARQUIVO):
        try:
            with open(Config.ROI_ARQUIVO, "r") as f:
                line = f.readline()
                if line:
                    # Converte string "x,y,w,h" para tupla de inteiros
                    self.roi_coords = tuple(map(int, line.split(",")))
                    print(f"[ROI] Carregada: {self.roi_coords}")
        except Exception as e:
            print(f"[ROI] Erro ao carregar: {e}")

def salvar_roi(self, coords):
    """
    Salva as coordenadas da ROI em arquivo.

    Args:
        coords (tuple): Tupla com (x, y, largura, altura)
    """
    with open(Config.ROI_ARQUIVO, "w") as f:
        f.write(f"{coords[0]},{coords[1]},{coords[2]},{coords[3]}")
    self.roi_coords = coords
    print(f"[ROI] Salva: {coords}")

def definir_roi_manual(self):
    """
    Interface interativa para definição manual da ROI.
    Abre a câmera e permite ao usuário selecionar a área de interesse.
    """
    # Inicializa captura de vídeo
    cap = cv2.VideoCapture(Config.CAMERA_INDEX)
    if not cap.isOpened():
        messagebox.showerror("Erro", f"Câmera {Config.CAMERA_INDEX} não encontrada!")
        return

    print("\n--- DEFINIÇÃO MANUAL DA ROI ---")
    print("[S] Congelar e Selecionar nova área")

```

```

print("[ENTER] Confirmar/Salvar a seleção")
print("[Q] Sair / Cancelar")

selected_roi = None

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Desenha a ROI atual se já existir (para referência)
    if self.roi_coords:
        rx, ry, rw, rh = self.roi_coords
        cv2.rectangle(frame, (rx, ry), (rx+rw, ry+rh), (255, 0, 0), 2)
        cv2.putText(frame, "AREA ATUAL", (rx, ry-10),
                     cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,0,0), 2)

    # Instruções na tela
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(frame, "[S] SELECIONAR NOVA AREA", (10, 30), font, 0.6, (0,255,255), 2)
    cv2.putText(frame, "[ENTER] SALVAR SELECAO", (10, 60), font, 0.6, (0,255,255), 2)
    cv2.putText(frame, "[Q] SAIR", (10, 90), font, 0.6, (0,255,255), 2)

    cv2.imshow("DEFINIR AREA", frame)
    k = cv2.waitKey(1)

    # Tecla 'S': abre ferramenta de seleção do OpenCV
    if k == ord('s') or k == ord('S'):
        selected_roi = cv2.selectROI("DEFINIR AREA", frame,
                                     fromCenter=False, showCrosshair=True)
        break

    # Tecla 'Q': sai sem salvar
    elif k == ord('q') or k == ord('Q'):
        break

    # Finaliza captura e fecha janelas
    cap.release()
    cv2.destroyAllWindows()

    # Salva apenas se a ROI tiver dimensões válidas
    if selected_roi and selected_roi[2] > 0 and selected_roi[3] > 0:

```

```

        self.salvar_roi(selected_roi)
        messagebox.showinfo("Sucesso", "Área de monitoramento salva!")

# -----
# Gerenciamento do Dataset
# -----

def limpar_dataset(self):
    """
    Apaga todas as imagens dos diretórios de treinamento.
    Útil para reiniciar o processo de coleta de dados.

    Returns:
        int: Número de arquivos deletados
    """
    pastas = [Config.PATH_OBJETOS, Config.PATH_NAO_OBJETOS]
    count = 0

    for pasta in pastas:
        if os.path.exists(pasta):
            for arquivo in os.listdir(pasta):
                caminho = os.path.join(pasta, arquivo)
                try:
                    if os.path.isfile(caminho):
                        os.unlink(caminho) # Deleta o arquivo
                        count += 1
                except Exception as e:
                    print(f"Erro ao deletar {caminho}: {e}")

    print(f"[DATASET] {count} arquivos deletados.")
    return count

# -----
# Extração de Características
# -----

def extrair_features(self, img):
    """
    Extrai vetor de características da imagem usando histogramas.

    A característica é composta por:
    - Histograma de intensidade (escala de cinza) com 256 bins

```

- Histograma do canal Hue (matiz) no espaço HSV com 180 bins

Total: $256 + 180 = 436$ características

Args:

img (numpy.ndarray): Imagem BGR de entrada

Returns:

numpy.ndarray: Vetor de características normalizado (436 dimensões)

"""

Verifica se a imagem é válida

if img is None or img.size == 0:

return np.zeros(436)

try:

Redimensiona para tamanho fixo (reduz complexidade computacional)

img = cv2.resize(img, (64, 64))

Converte para escala de cinza e HSV

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

Calcula histogramas

h_gray = cv2.calcHist([gray], [0], None, [256], [0, 256]) # Intensidade

h_hue = cv2.calcHist([hsv], [0], None, [180], [0, 180]) # Matiz

Normaliza histogramas para soma = 1 (invariância a brilho)

cv2.normalize(h_gray, h_gray)

cv2.normalize(h_hue, h_hue)

Concatena os dois histogramas em um único vetor

return np.concatenate((h_gray, h_hue)).flatten()

except Exception as e:

print(f"[FEATURES] Erro na extração: {e}")

return np.zeros(436)

Treinamento do Modelo

def treinar(self):

"""

Treina o classificador KNN com as imagens do dataset.

Etapas:

1. Cria diretórios se não existirem
2. Carrega todas as imagens dos diretórios
3. Extrai características de cada imagem
4. Divide em treino e teste
5. Padroniza os dados com StandardScaler
6. Treina o KNN
7. Avalia acurácia no conjunto de teste
8. Serializa modelo e scaler para uso posterior

Returns:

```

    str: Mensagem de resultado com acurácia
    """"

# Garante que os diretórios existam
Path(Config.PATH_OBJETOS).mkdir(parents=True, exist_ok=True)
Path(Config.PATH_NAO_OBJETOS).mkdir(parents=True, exist_ok=True)

# Lista todas as imagens nos diretórios
lista_obj = list(Path(Config.PATH_OBJETOS).glob("*.png"))
lista_nobj = list(Path(Config.PATH_NAO_OBJETOS).glob("*.png"))

# Verifica se há imagens suficientes
if not lista_obj or not lista_nobj:
    return "Erro: Capture imagens das duas classes antes de treinar!"

# Extrai características e rótulos
feats, labels = [], []

for p in lista_obj:
    img = cv2.imread(str(p))
    feats.append(self.extrair_features(img))
    labels.append(1) # Classe 1: objeto presente

for p in lista_nobj:
    img = cv2.imread(str(p))
    feats.append(self.extrair_features(img))
    labels.append(0) # Classe 0: fundo vazio

# Converte para arrays numpy
X = np.array(feats)
```

```

y = np.array(labels)

# Define número de vizinhos (k) baseado no tamanho do dataset
# k pequeno para datasets pequenos, evitando overfitting
k_val = 3 if len(X) > 5 else 1

# Divide em treino (80%) e teste (20%)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Padroniza os dados (fundamental para KNN baseado em distância)
self.scaler = StandardScaler().fit(X_train)
X_train_scaled = self.scaler.transform(X_train)

# Treina o classificador KNN
self.knn = KNeighborsClassifier(n_neighbors=k_val).fit(X_train_scaled, y_train)

# Avalia no conjunto de teste (se houver)
if len(X_test) > 0:
    X_test_scaled = self.scaler.transform(X_test)
    acc = self.knn.score(X_test_scaled, y_test)
else:
    acc = 1.0 # Caso não haja dados de teste (dataset muito pequeno)

# Serializa modelo e scaler para uso futuro
joblib.dump(self.knn, Config.MODELO_ARQUIVO)
joblib.dump(self.scaler, Config.SCALER_ARQUIVO)

print(f"[TREINO] Modelo salvo com k={k_val}. Acurácia: {acc*100:.1f}%")
return f"Treino IA Concluído! Acurácia: {acc*100:.1f}%"

# -----
# Carregamento do Modelo
# -----

def carregar_modelo(self):
    """Carrega modelo KNN e scaler previamente treinados dos arquivos."""
    if os.path.exists(Config.MODELO_ARQUIVO) and os.path.exists(Config.SCALER_ARQUIVO):
        self.knn = joblib.load(Config.MODELO_ARQUIVO)
        self.scaler = joblib.load(Config.SCALER_ARQUIVO)
        print("[MODELO] Carregado com sucesso.")

```

```

else:
    print("[MODELO] Nenhum modelo encontrado. Execute o treinamento primeiro.")

# =====
# INTERFACE GRÁFICA (Tkinter)
# =====

class InterfaceApp:
    """
    Interface gráfica principal do sistema.
    Gerencia a interação do usuário com as funcionalidades.
    """

    def __init__(self, root):
        """
        Inicializa a interface gráfica.

        Args:
            root (tk.Tk): Janela principal do Tkinter
        """
        self.root = root
        self.root.title("Sistema de Inspeção Visual - Indústria 4.0")
        self.root.geometry("450x600")

        # Instancia o núcleo de visão computacional
        self.vision = VisionSystem()

        # Título da aplicação
        tk.Label(root, text="Controle de Qualidade Industrial",
                 font=("Arial", 14, "bold")).pack(pady=15)

        # Estilo padrão para botões
        btn_style = {"font": ("Arial", 11), "height": 2, "width": 30}

        # Botões das etapas do processo
        tk.Button(root, text="1. Definir Área (ROI)",
                  command=self.definir_area, **btn_style, bg="#dddddd").pack(pady=5)

        tk.Button(root, text="2. Capturar Imagens",
                  command=self.capturar, **btn_style, bg="#dddddd").pack(pady=5)

```

```

tk.Button(root, text="3. Treinar IA",
          command=self.treinar, **btn_style, bg="#ffcc00").pack(pady=5)

tk.Button(root, text="4. INICIAR PRODUÇÃO",
          command=self.producao, font=("Arial", 12, "bold"),
          height=3, width=30, bg="#4CAF50", fg="white").pack(pady=20)

# Barra de status na parte inferior
self.lbl_status = tk.Label(root, text="Sistema Pronto.",
                           bd=1, relief=tk.SUNKEN, anchor=tk.W)
self.lbl_status.pack(side=tk.BOTTOM, fill=tk.X)

def definir_area(self):
    """Callback para o botão 'Definir Área'."""
    self.vision.definir_roi_manual()

# -----
# Captura de Imagens para Treinamento
# -----

def capturar(self):
    """
    Modo de captura de imagens para composição do dataset.
    Permite ao usuário capturar exemplos positivos (objeto) e negativos (fundo).
    """

    # Verifica se a ROI já foi definida
    if not self.vision.roi_coords:
        messagebox.showwarning("Aviso", "Defina a Área (Passo 1) primeiro!")
        return

    # Garante que os diretórios existam
    Path(Config.PATH_OBJETOS).mkdir(parents=True, exist_ok=True)
    Path(Config.PATH_NAO_OBJETOS).mkdir(parents=True, exist_ok=True)

    # Inicializa captura de vídeo
    cap = cv2.VideoCapture(Config.CAMERA_INDEX)

    print("\n--- MODO CAPTURA ---")
    print("[O] Salvar como OBJETO (peça presente)")
    print("[N] Salvar como NÃO OBJETO (fundo)")
    print("[C] LIMPAR TODO O DATASET")
    print("[Q] Sair")

```

```

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Faz uma cópia do frame original (sem anotações) para salvar
    clean_frame = frame.copy()

    # Desenha a ROI na tela
    rx, ry, rw, rh = self.vision.roi_coords
    cv2.rectangle(frame, (rx, ry), (rx+rw, ry+rh), (255, 0, 0), 2)

    # Conta quantas imagens já foram capturadas
    n_obj = len(os.listdir(Config.PATH_OBJETOS))
    n_nobj = len(os.listdir(Config.PATH_NAO_OBJETOS))

    # Instruções na tela
    cv2.putText(frame, f"[O] OBJETO: {n_obj}", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
    cv2.putText(frame, f"[N] FUNDO: {n_nobj}", (10, 60),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
    cv2.putText(frame, "[C] LIMPAR TUDO", (10, 90),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,255), 2)
    cv2.putText(frame, "[Q] SAIR", (10, 120),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)

    cv2.imshow("Captura", frame)
    k = cv2.waitKey(1)

    # Tecla 'Q': sai do modo captura
    if k == ord('q'):
        break

    # Tecla 'O': salva região da ROI como objeto (classe 1)
    elif k == ord('o'):
        crop = clean_frame[ry:ry+rh, rx:rx+rw]
        timestamp = datetime.now().strftime('%H%M%S%f')
        fname = f"{Config.PATH_OBJETOS}/obj_{timestamp}.png"
        cv2.imwrite(fname, crop)
        print(f"Objeto salvo: {fname}")

```

```

# Tecla 'N': salva região da ROI como não-objeto (classe 0)
elif k == ord('n'):
    crop = clean_frame[ry:ry+rh, rx:rx+rw]
    timestamp = datetime.now().strftime('%H%M%S%f')
    fname = f"{Config.PATH_NAO_OBJETOS}/bg_{timestamp}.png"
    cv2.imwrite(fname, crop)
    print(f"Fundo salvo: {fname}")

# Tecla 'C': limpa todo o dataset
elif k == ord('c') or k == ord('C'):
    self.vision.limpar_dataset()
    print("--- DATASET LIMPO ---")

# Finaliza captura
cap.release()
cv2.destroyAllWindows()

# -----
# Treinamento do Modelo
# -----

def treinar(self):
    """Callback para o botão 'Treinar IA'."""
    resultado = self.vision.treinar()
    messagebox.showinfo("Resultado do Treinamento", resultado)

# -----
# Modo Produção (Classificação em Tempo Real)
# -----

def producao(self):
    """
    Modo de operação contínua com classificação em tempo real.
    Aplica histerese e filtro temporal para estabilidade.
    """
    # Verifica se o modelo foi treinado
    if not self.vision.knn:
        messagebox.showwarning("Erro", "Treine o modelo antes de iniciar produção!")
        return

# Inicializa captura de vídeo
cap = cv2.VideoCapture(Config.CAMERA_INDEX)

```

```

# Buffer para filtro temporal (média móvel das últimas predições)
buffer = []
TAMANHO_BUFFER = 5

# Estado atual da detecção
estado_atual_detectado = False
last_action = None # Último comando enviado ao CLP

# Temporizador para manter braço acionado por um período mínimo
tempo_liberacao_braço = 0
DELAY_SEGURAR = 0.3 # Tempo em segundos para manter o braço ativo

# Verifica se a ROI está definida
if not self.vision.roi_coords:
    messagebox.showerror("Erro", "Área de ROI não definida!")
    return

rx, ry, rw, rh = self.vision.roi_coords

print("\n--- MODO PRODUÇÃO INICIADO ---")
print("Pressione 'Q' para sair.")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Extraí região de interesse
    roi_img = frame[ry:ry+rh, rx:rx+rw]

    # Extraí características e classifica
    feats = self.vision.extrair_features(roi_img)
    feats_scaled = self.vision.scaler.transform([feats])
    pred = self.vision.knn.predict(feats_scaled)[0]

    # Atualiza buffer para filtro de média móvel
    buffer.append(pred)
    if len(buffer) > TAMANHO_BUFFER:
        buffer.pop(0)

    # Calcula confiança média (proporção de detecções positivas)
    avg = sum(buffer) / len(buffer)

```

```

# =====
# LÓGICA DE HISTERESE COM TEMPORIZADOR
# =====
# Objetivo: evitar oscilações rápidas e garantir tempo mínimo
# de atuação do braço pneumático.
# =====

# 1. Se a confiança média ultrapassa 80%, considera objeto detectado
if avg > 0.8:
    estado_atual_detectado = True
    # Renova o temporizador (mantém braço ativo por mais DELAY_SEGURAR segundos)
    tempo_liberacao_braco = time.time() + DELAY_SEGURAR

# 2. Se o estado está ativo, verifica se já pode desligar
if estado_atual_detectado:
    # Desliga apenas se confiança cair abaixo de 20% E o temporizador expirou
    if avg < 0.2 and time.time() > tempo_liberacao_braco:
        estado_atual_detectado = False

# =====
# ENVIO DE COMANDOS AO CLP
# =====
if estado_atual_detectado:
    # Se está em período de espera (confiança baixa mas temporizador ativo)
    if avg < 0.2:
        label = f"DELAY... ({int(tempo_liberacao_braco - time.time())}s)"
        color = (0, 165, 255) # Laranja
    else:
        label = "OBJETO DETECTADO"
        color = (0, 0, 255) # Vermelho

    # Destaca a ROI com a cor correspondente
    cv2.rectangle(frame, (rx, ry), (rx+rw, ry+rh), color, 3)

    # Envia comando de rejeição se não foi enviado antes
    if last_action != 1:
        FestoController.enviar_comando(Config.CMD_BRACO_ESTEIRAS)
        last_action = 1

else:
    label = "MONITORANDO..."
    color = (0, 255, 0) # Verde

```

```

cv2.rectangle(frame, (rx, ry), (rx+rw, ry+rh), color, 2)

# Envia comando de esteiras normais se não foi enviado antes
if last_action != 0:
    FestoController.enviar_comando(Config.CMD_2_ESTEIRAS)
    last_action = 0

# Exibe confiança na tela
cv2.putText(frame, f"Confiança: {int(avg*100)}%",
            (10, frame.shape[0]-20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,255,0), 1)

# Exibe rótulo do estado atual
cv2.putText(frame, label, (rx, ry-10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)

cv2.imshow("PRODUCAO - IA", frame)

# Tecla 'Q' para sair
if cv2.waitKey(1) == ord('q'):
    # Desliga todos os atuadores antes de sair
    FestoController.enviar_comando(Config.CMD_DESLIGAR_TUDO)
    break

# Finaliza captura
cap.release()
cv2.destroyAllWindows()

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

if __name__ == "__main__":
    """Ponto de entrada da aplicação."""
    root = tk.Tk()
    app = InterfaceApp(root)
    root.mainloop()

```

APÊNDICE B – GUIA DE OPERAÇÃO E INTEGRAÇÃO COM CLP

B.1 REQUISITOS DE HARDWARE E SOFTWARE

Hardware (Configuração Mínima Recomendada):

- Processador: Intel Core i5 (8ª geração) ou equivalente AMD;
- Memória RAM: 8 GB;
- Armazenamento: 128 GB SSD;
- Sistema de aquisição de imagem: câmera USB com resolução mínima de 720p (recomendado: 1080p ou superior);
- Conexão de rede: interface Ethernet para comunicação com CLP;
- Periféricos: teclado e mouse para operação do sistema.

Hardware (Configuração Específica do Projeto):

- Computador: Notebook Samsung Book2 com processador Intel Core i7;
- Câmera: Anker C200 (resolução 2K);
- Interface de rede: conexão direta via cabo UTP (configuração *crossover*);
- Controlador Industrial: CLP Festo FC640 configurado com endereço IP 192.168.0.1;
- Bancada: sistema didático com esteira e atuadores pneumáticos.

Software:

- Sistema operacional: Windows 11 ou Linux Ubuntu 20.04 LTS;
- Ambiente de programação: Python 3.8 ou superior;
- Bibliotecas Python essenciais:
 - opencv-python (processamento de imagens);
 - numpy (operações com arrays numéricos);
 - scikit-learn (algoritmo KNN e pré-processamento);
 - requests (comunicação HTTP/REST);
 - joblib (serialização do modelo treinado);
 - tkinter (interface gráfica, já incluso no Python).

Configurações de Rede:

- Endereçamento IP: configuração estática na sub-rede 192.168.0.0/24;
- Conexão CLP: acesso ao CLP no endereço 192.168.0.1 via porta HTTP padrão (80);
- Configuração do host: endereço IP do computador na mesma faixa (ex.: 192.168.0.100).

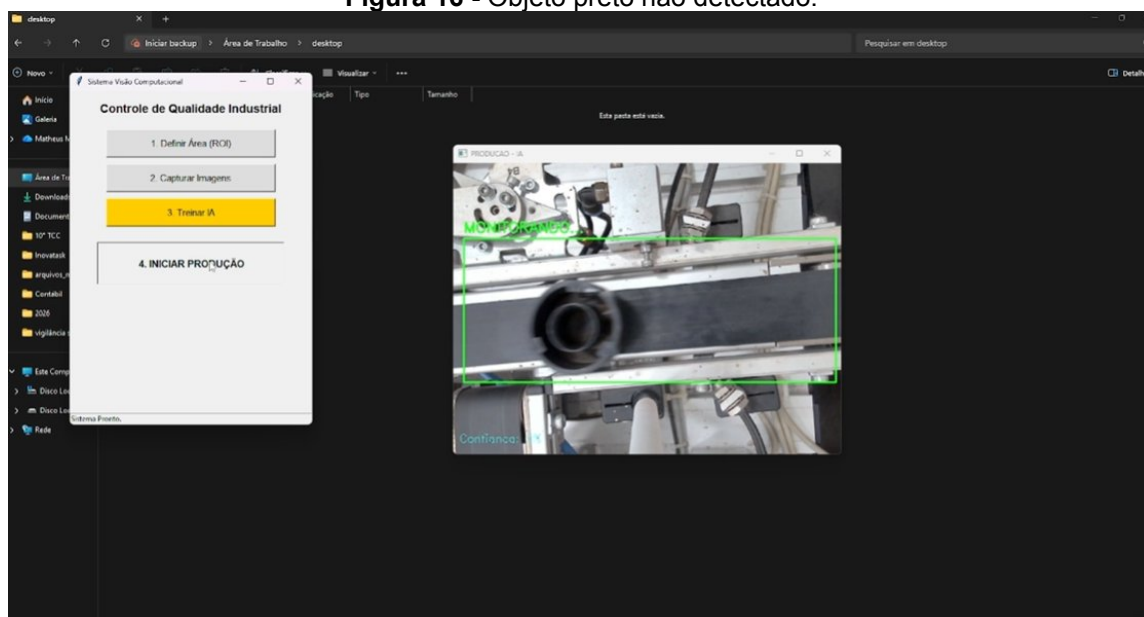
Configurações do Sistema:

- Permissões: privilégios de administrador para instalação de pacotes;
 - Drivers: controladores atualizados para a câmera USB;
- Ambiente de desenvolvimento: IDE ou editor de código configurado para Python.

B.2 GUIA DE OPERAÇÃO

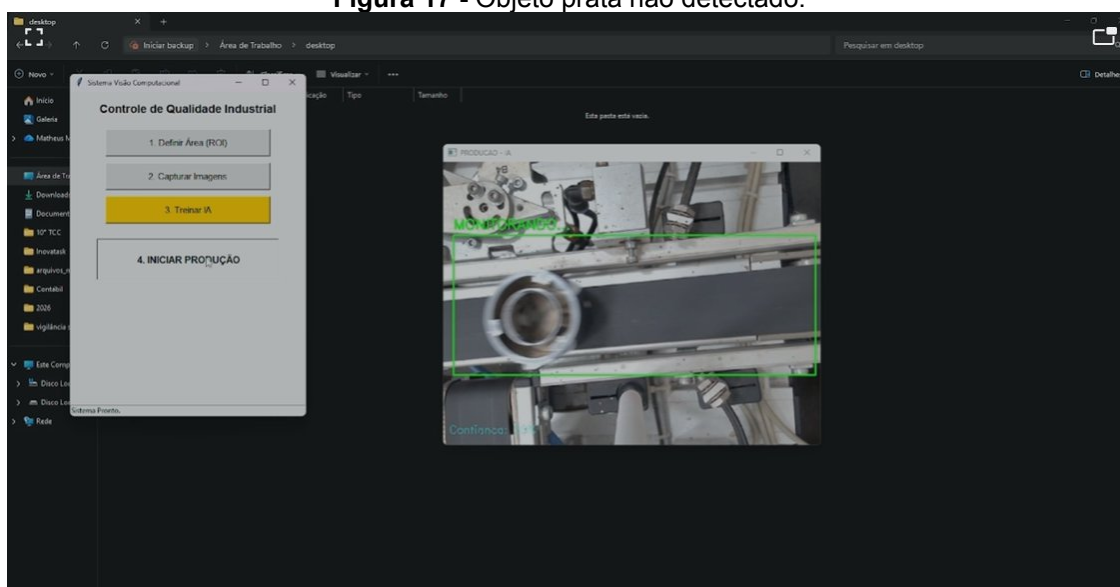
1. Executar o script main.py;
2. Primeiramente, clicar no botão "1. Definir Área (ROI)" e selecionar a região de interesse na câmera;
3. Selecionar a opção "2. Capturar Imagens";
4. Com a câmera exibindo o vídeo, pressionar a tecla "O" para registrar imagens com a peça (Objeto);
5. Pressionar a tecla "N" para registrar imagens sem a peça (Não Objeto);
6. Opcional) Pressionar a tecla "C" para limpar as imagens anteriores e reiniciar a coleta;
7. Pressionar a tecla "Q" para finalizar a captura. As imagens serão salvas automaticamente nos diretórios dataset/objetos e dataset/nao_objetos;
8. Clicar no botão "3. Treinar IA" para treinar o classificador KNN;
9. Clicar no botão "4. INICIAR PRODUÇÃO" para iniciar a classificação em tempo real.

Figura 16 - Objeto preto não detectado.



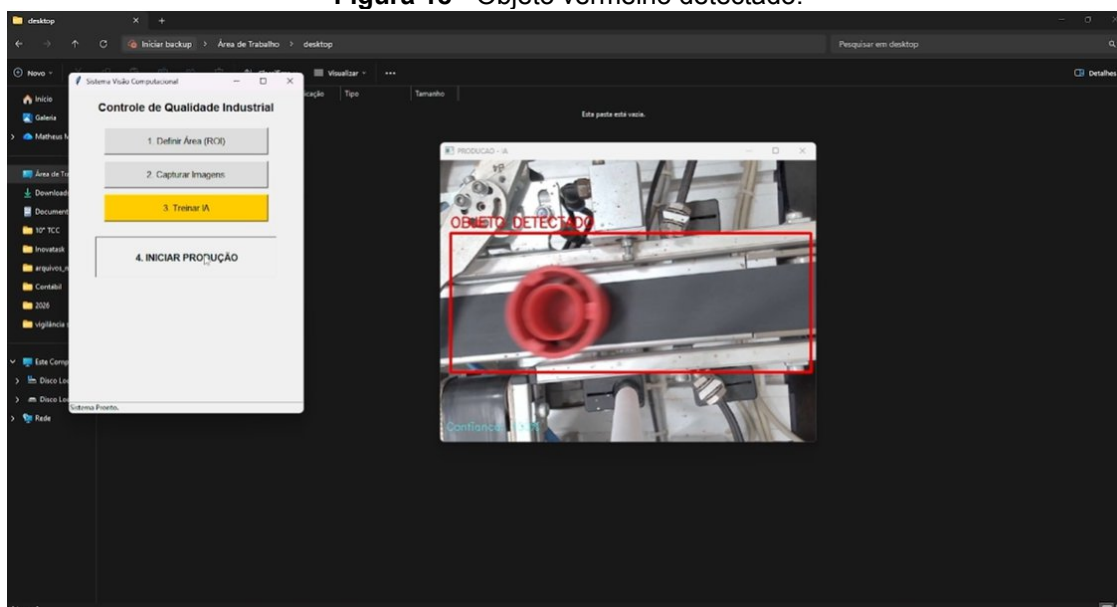
Fonte: Elaborada pelos autores (2026).

Figura 17 - Objeto prata não detectado.



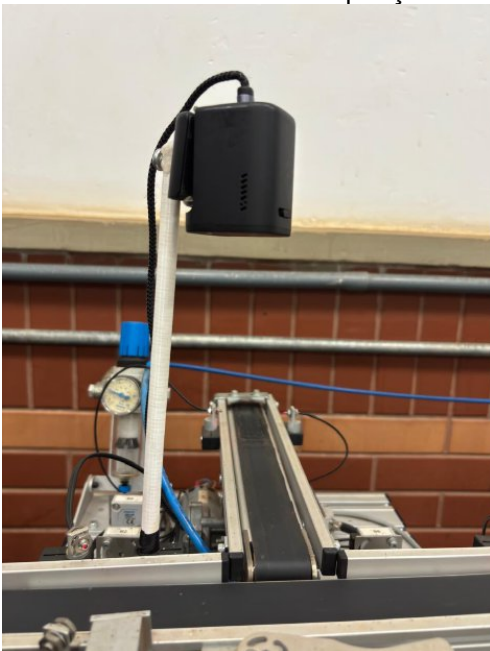
Fonte: Elaborada pelos autores (2026).

Figura 18 - Objeto vermelho detectado.



Fonte: Elaborada pelos autores (2026).

Figura 19 - Câmera Anker C200 utilizada na aquisição de imagens da bancada.



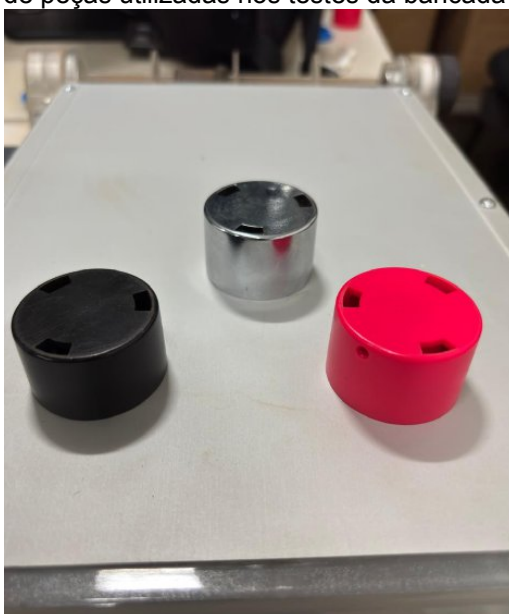
Fonte: Elaborada pelos autores (2026).

Figura 20 - Medidor pneumático em operação.



Fonte: Elaborada pelos autores (2026).

Figura 21 - Conjunto de peças utilizadas nos testes da bancada (preto, prata e vermelho).



Fonte: Elaborada pelos autores (2026).

Quadro 7 - Peças, cores e funções na bancada experimental.

Cor da Peça	Função / Observação
Preto	Componente móvel utilizado para calibração e testes de fundo
Prata	Componente móvel utilizado para calibração e testes de fundo
Vermelho	Objeto-alvo da classificação (classe positiva)

Fonte: Elaborada pelos autores (2026).

Figura 22 - Bancada Festo FC640 em funcionamento com suas conexões integradas.



Fonte: Elaborada pelos autores (2026).

B.3 MODO DE TREINAMENTO E CLASSIFICAÇÃO

1. Executar o script main.py;
2. Seguir os passos B.2.1 a B.2.8 para definição da ROI, captura de imagens e treinamento;
3. Verificar a acurácia exibida na mensagem de resultado do treinamento;
4. Iniciar o modo produção conforme passo B.2.9;
5. A janela da webcam exibirá o texto "OBJETO DETECTADO" ou "MONITORANDO..." conforme a classificação.

B.4 INTEGRAÇÃO COM CLP

A integração entre o sistema de visão computacional e o CLP Festo FC640 é realizada através de requisições HTTP GET para a interface web do controlador. O sistema implementa um cliente HTTP em Python que envia comandos para endpoints específicos, utilizando parâmetros na URL para controlar os atuadores da bancada.

- Endpoints e Parâmetros

A comunicação entre o sistema de visão e o CLP é realizada por meio de requisições HTTP, utilizando comandos padronizados para o controle dos atuadores. Para evitar a duplicidade de informações, os detalhes de cada comando, bem como suas respectivas URLs e funções operacionais, encontram-se descritos anteriormente no **Quadro 5**, cujos parâmetros são replicados nesta etapa do processo.

- Exemplos de Requisições HTTP

Exemplo 1: Comando para ativar braço e esteiras (rejeição):

```
GET /main.htm?ci:maw0=26 HTTP/1.1
```

```
Host: 192.168.0.1
```

```
User-Agent: Python-Vision-System/1.0
```

```
Connection: close
```

Exemplo 2: Comando para ativar apenas a esteira 1:

```
GET /main.htm?ci:maw0=16 HTTP/1.1
```

```
Host: 192.168.0.1
```

```
User-Agent: Python-Vision-System/1.0
```

```
Connection: close
```

Exemplo 3: Comando para desligar todas as funções:

```
GET /main.htm?ci:maw0=0 HTTP/1.1
```

```
Host: 192.168.0.1
```

```
User-Agent: Python-Vision-System/1.0
```

```
Connection: close
```

Código de Implementação Python (Classe CLPClient)

"""

Módulo de comunicação com CLP Festo FC640 via HTTP.

Implementa cliente com controle de duplicação, retentativas e modo de segurança.

"""

import requests

import time

import hashlib

import logging

from typing import Optional, Dict

class CLPClient:

"""

Cliente para comunicação com CLP Festo via requisições HTTP GET.

Attributes:

base_url (str): URL base do CLP (padrão: http://192.168.0.1)

last_command_hash (str): Hash do último comando para evitar duplicatas

timeout (float): Tempo máximo de espera por resposta (segundos)

max_retries (int): Número máximo de tentativas em caso de falha

logger (logging.Logger): Logger para registro de eventos

"""

def __init__(self, base_url: str = "http://192.168.0.1"):

"""

Inicializa o cliente CLP.

Args:

base_url: URL base do CLP (sem o path)

"""

self.base_url = base_url.rstrip('/')

self.last_command_hash = None

self.timeout = 0.5 # segundos (timeout curto para não travar o sistema)

```

self.max_retries = 3
self.logger = logging.getLogger(__name__)

def _generate_command_hash(self, codigo: int) -> str:
    """
    Gera hash único para o comando baseado no código e timestamp.

    Utilizado para evitar o reenvio de comandos duplicados em sequência.

    Args:
        codigo (int): Código numérico do comando (0, 16, 18, 26)

    Returns:
        str: Hash MD5 do comando + timestamp
    """
    # Combina código com timestamp para garantir unicidade
    timestamp = int(time.time() * 1000) # milissegundos para maior precisão
    data = f"{codigo}:{timestamp}"

    # Gera hash MD5 (16 bytes) e retorna como string hexadecimal
    return hashlib.md5(data.encode()).hexdigest()

def enviar_comando(self, codigo: int) -> Dict:
    """
    Envia comando para o CLP com controle de duplicação e retentativas.

    Args:
        codigo (int): Código do comando (0, 16, 18, 26)

    Returns:
        Dict: Dicionário com status da operação
    """
    # Gera hash para controle de duplicação
    cmd_hash = self._generate_command_hash(codigo)

```

```

# Evita reenvio do mesmo comando em sequência (otimização de rede)
if cmd_hash == self.last_command_hash:
    self.logger.info(f"Comando duplicado ignorado: {codigo}")
    return {"status": "skipped", "message": "Duplicate command"}

# Constrói URL completa com parâmetro
url = f"{self.base_url}/main.htm?ci:maw0={codigo}"

self.logger.info(f"Enviando comando {codigo} para {url}")

# Tenta enviar com retentativas (backoff exponencial)
for tentativa in range(self.max_retries):
    try:
        # Envia requisição GET com timeout
        response = requests.get(url, timeout=self.timeout)

        # Verifica se a requisição foi bem-sucedida
        if response.status_code == 200:
            self.last_command_hash = cmd_hash
            self.logger.info(f"Comando {codigo} enviado com sucesso")
            return {
                "status": "success",
                "codigo": codigo,
                "hash": cmd_hash
            }
        else:
            self.logger.warning(
                f"Tentativa {tentativa+1}: HTTP {response.status_code}"
            )

    except requests.exceptions.Timeout:
        self.logger.error(f"Tentativa {tentativa+1}: Timeout após {self.timeout}s")
    except requests.exceptions.ConnectionError:
        self.logger.error(f"Tentativa {tentativa+1}: Erro de conexão")

```

```

except Exception as e:
    self.logger.error(f"Tentativa {tentativa+1}: Erro inesperado: {e}")

    # Aguarda antes de retentar (backoff exponencial: 1s, 2s, 4s...)
    if tentativa < self.max_retries - 1:
        wait_time = 2 ** tentativa # 1, 2, 4 segundos
        self.logger.info(f"Aguardando {wait_time}s antes de retentar...")
        time.sleep(wait_time)

    # Se chegou aqui, todas as tentativas falharam
    self.logger.error(f"Falha ao enviar comando {codigo} após {self.max_retries}
tentativas")

    # Ativa modo de segurança em caso de falha crítica
    self._activate_safety_mode()

    return {
        "status": "error",
        "message": f"Falha após {self.max_retries} tentativas",
        "codigo": codigo
    }

def _activate_safety_mode(self):
    """
    Ativa modo de segurança quando o CLP não responde.

    Em um sistema real, esta função deveria:
    1. Parar a esteira
    2. Retrair atuadores para posição segura
    3. Acionar alarme visual/sonoro
    4. Registrar evento crítico para auditoria
    """
    self.logger.critical("ATIVANDO MODO DE SEGURANÇA")

```

Em um ambiente de produção, aqui seria acionado um desligamento de emergência

```
# Como é um protótipo didático, apenas registramos o evento
print("\n*** MODO DE SEGURANÇA ATIVADO ***")
print("Verifique a comunicação com o CLP e reinicie o sistema.")
```

```
# Registrar em log de auditoria
audit_logger = logging.getLogger('audit')
```

```
audit_logger.critical("MODO_SEGURANCA_ATIVADO|Falha_comunicacao_CLP")
```

- Práticas de Robustez Implementadas

Confirmação de Recebimento (ACK):

```
def send_command_with_ack(self, codigo: int) -> bool:
```

```
    """
```

Envia comando e verifica confirmação implícita.

Como o CLP Festo não retorna JSON, a confirmação é feita pelo código HTTP 200 e pela verificação de estado posterior.

```
    """
```

```
    response = self.enviar_comando(codigo)
```

```
    if response.get('status') == 'success':
```

```
        self.logger.info(f"Comando {codigo} confirmado (HTTP 200)")
```

```
        return True
```

```
    return False
```

- Timeout e Retentativas com Backoff Exponencial

O método `enviar_comando` já implementa timeout configurável e retentativas com backoff exponencial (1s, 2s, 4s). Esta estratégia:

- Evita sobrecarga da rede em caso de falhas temporárias;
- Aumenta a chance de sucesso em condições instáveis;
- Não bloqueia o sistema principal por longos períodos.

- Sistema de Logging Detalhado:

```
def setup_logging():
    """Configura sistema de logging para auditoria e diagnóstico."""
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        handlers=[
            logging.FileHandler('clp_integration.log'),
            logging.StreamHandler()
        ]
    )

    # Logger específico para auditoria (formato simplificado para análise)
    audit_logger = logging.getLogger('audit')
    audit_handler = logging.FileHandler('audit_trail.log')
    audit_handler.setFormatter(logging.Formatter(
        '%(asctime)s|%(levelname)s|COMANDO:%(message)s'
    ))
    audit_logger.addHandler(audit_handler)
```

- Modo de Fallback Seguro:

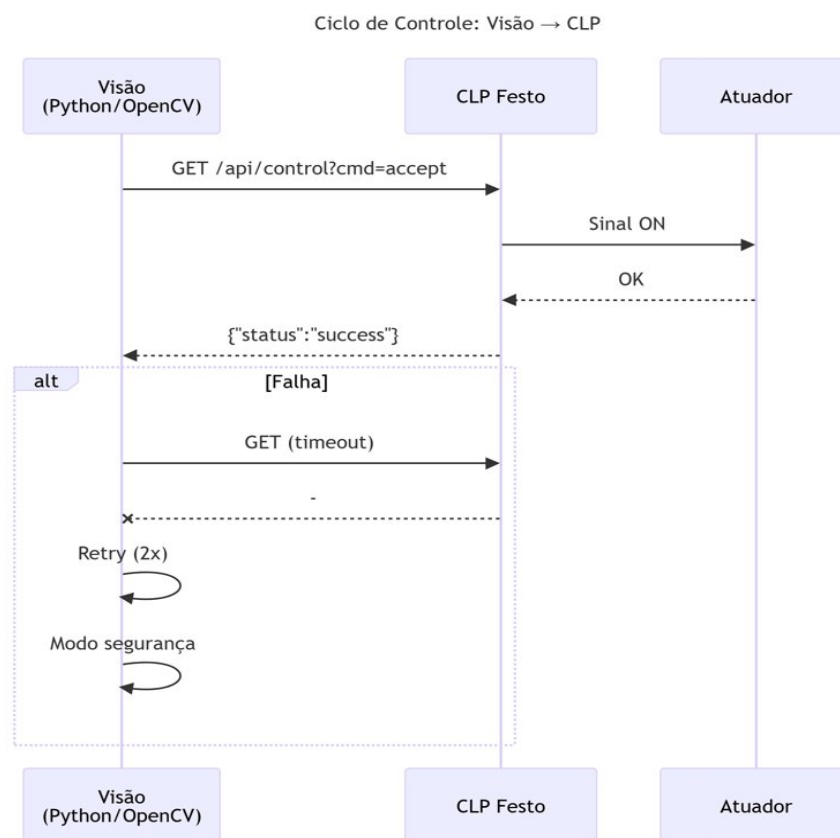
O método `_activate_safety_mode()` é chamado quando todas as tentativas de comunicação falham. Em um sistema de produção, este modo deveria:

1. Parar imediatamente a esteira;
2. Retrair atuadores para posição segura;
3. Acionar alarme visual e sonoro;
4. Registrar o evento para análise posterior;

5. Aguardar intervenção do operador.

No protótipo didático, a função registra o evento e exibe uma mensagem de alerta.

Figura 23 - Diagrama de sequência da integração entre sistema de visão computacional e CLP Festo FC640.



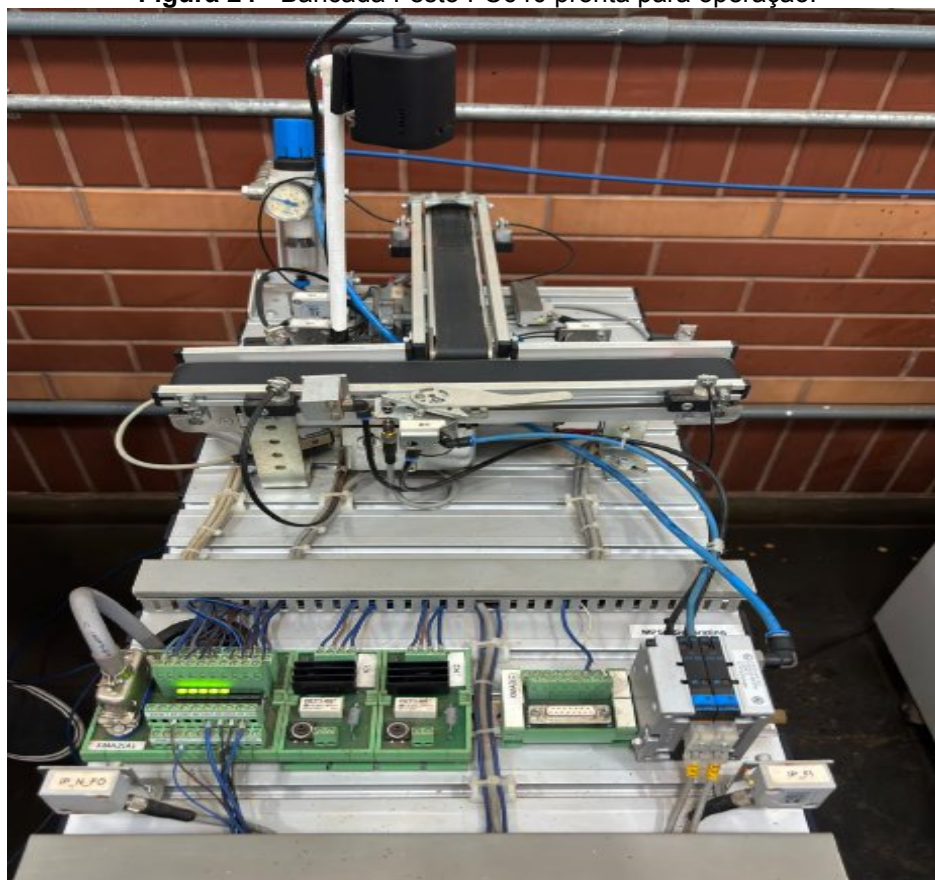
Fonte: Elaborada pelos autores (2026).

a. Descrição do fluxo:

1. Sistema de visão processa a imagem e toma a decisão de classificação;
2. Envia comando via requisição HTTP GET ao CLP;
3. CLP recebe o comando, executa a lógica Ladder e aciona o atuador correspondente;
4. Confirmação (HTTP 200) retorna ao sistema de visão;
5. Em caso de timeout ou falha, o sistema ativa o modo de segurança após retentativas.

A seguir na Figura 24, bancada de testes montada e pronta para a captura das imagens, com todos os componentes integrados.

Figura 24 - Bancada Festo FC640 pronta para operação.



Fonte: Elaborada pelos autores (2026).

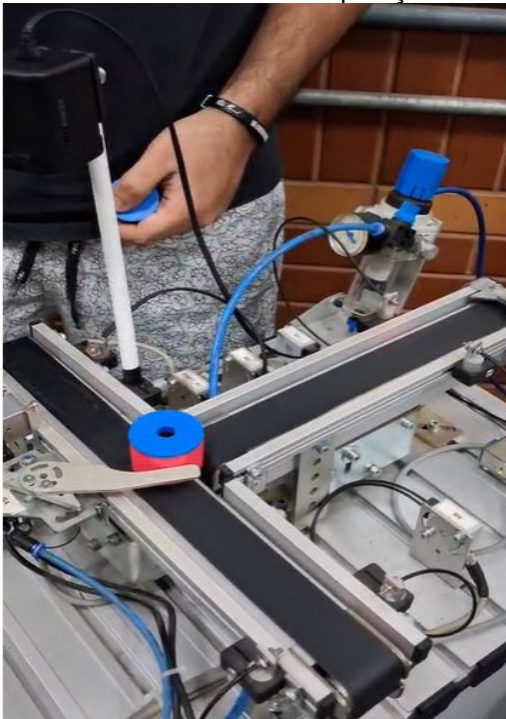
B.5 SEGURANÇA E CONFIABILIDADE

- Sensores de fim de curso: para evitar que os atuadores ultrapassem os limites mecânicos;
- Botões de parada de emergência: conectados diretamente ao CLP para desligamento imediato;
- Rotinas de emergência no CLP: programadas em Ladder para garantir estados seguros em caso de falha;
- Intertravamentos de segurança: evitam que comandos conflitantes sejam executados simultaneamente.

O sistema de visão computacional deve ser tratado como elemento auxiliar de decisão, e o CLP deve manter estados seguros em caso de:

- Falha de comunicação com o computador de visão;
- Desligamento inesperado do notebook;
- Timeout nas requisições HTTP;
- Detecção de comandos inválidos ou fora de sequência.
- Recomendações de segurança para operação:
 - Sempre verificar o estado dos atuadores antes de iniciar o modo produção;
 - Manter o botão de emergência acessível durante toda a operação;
 - Realizar testes iniciais sem a presença de peças para validar a lógica;
 - Monitorar os logs do sistema para identificar padrões de falha;
 - Implementar um *watchdog* de software que monitore continuamente a comunicação com o CLP.

Figura 25 - Bancada Festo FC640 em operação durante os testes.



Fonte: Elaborada pelos autores (2026).

A Figura 25 apresenta o sistema em funcionamento, com a câmera posicionada sobre a esteira e os atuadores respondendo às classificações em tempo real.