

,
,
UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

FERNANDO HENRIQUE PIMENTEL RONDON DE ASSIS

Desenvolvimento de uma Planta Didática com Sensor
Inteligente para Classificação Automática de Materiais em
Tempo Real Aplicada à Indústria 4.0

São Carlos
2025

FERNANDO HENRIQUE PIMENTEL RONDON DE ASSIS

Desenvolvimento de uma Planta Didática com Sensor
Inteligente para Classificação Automática de Materiais em
Tempo Real Aplicada à Indústria 4.0

Monografia apresentada ao Curso
de Engenharia Elétrica com
ênfase em eletrônica, da Escola
de Engenharia de São Carlos
da Universidade de São Paulo,
como parte dos requisitos para
obtenção do Título de Engenheiro
Eletricista

Orientador(a): Prof. Dr. Fabio
Romano Lofrano Dotto

São Carlos
2025

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

A848d

Assis, Fernando Henrique Pimentel Rondon de
Desenvolvimento de uma Planta Didática com Sensor Inteligente para Classificação Automática de Materiais em Tempo Real Aplicada à Indústria 4.0 / Fernando Henrique Pimentel Rondon de Assis; orientador Fabio Romano Lofrano Dotto. São Carlos, 2025.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2025.

1. Indústria 4.0. 2. Visão computacional embarcada. 3. Sistemas embarcados. 4. Automação Industrial. 5. Sensor Inteligente. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fernando Henrique Pimentel Rondon de Assis

Título: “Desenvolvimento de uma Planta Didática com Sensor Inteligente para Classificação Automática de Materiais em Tempo Real Aplicada à Indústria 4.0”

**Trabalho de Conclusão de Curso defendido e aprovado em
04/12/2025,**

com NOTA 9,5 (nove, cinco), pela Comissão Julgadora:

**Prof. Dr. Fabio Romano Lofrano Dotto - Orientador
SEL/EESC/USP**

Prof. Dr. Pedro de Oliveira Conceição Junior - SEL/EESC/USP

Mestre Giovanni Oliveira de Sousa - Doutorando EESC/USP

**Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior**

AGRADECIMENTOS

Agradeço primeiramente à minha família, pelo apoio e compreensão em todos os momentos desta jornada. Aos meus amigos agradeço pela presença constante, pela paciência nos momentos difíceis e pelas inúmeras conversas que me ajudaram a seguir em frente. Obrigado por acreditarem em mim e por tornarem essa caminhada mais leve e alegre. A todos que, de alguma forma, contribuíram para a realização deste trabalho, meu sincero muito obrigado.

*“O estudo, a busca da verdade e da beleza são domínios
em que nos é consentido sermos crianças por toda a vida.”*
Albert Einstein

RESUMO

ASSIS, F. **Desenvolvimento de uma Planta Didática com Sensor Inteligente para Classificação Automática de Materiais em Tempo Real Aplicada à Indústria 4.0.** 2025. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2025.

Este trabalho apresenta o desenvolvimento de uma planta didática integrada a um sensor inteligente para classificação automática de materiais em tempo real, com foco em baixo custo, modularidade e interoperabilidade com sistemas industriais legados. O projeto foi motivado pela necessidade de aproximar o ensino superior brasileiro das tecnologias emergentes da Indústria 4.0, especialmente no contexto de laboratórios acadêmicos que ainda utilizam equipamentos convencionais de automação. A solução proposta integra sensores industriais, uma esteira transportadora, um Controlador Lógico Programável (CLP) e um módulo embarcado baseado em microcontrolador equipado com câmera e algoritmos de visão computacional. Para garantir a comunicação segura com sistemas de automação operando em 24 V, foi projetado um circuito de interface com isolamento galvânico utilizando optoacopladores, abrangendo canais digitais e analógicos. O funcionamento dos circuitos foi previamente validado por simulações e posteriormente prototipado em bancada, seguido do desenvolvimento de uma Placa de Circuito Impresso (PCB) posteriormente fabricada. Os resultados demonstram a viabilidade técnica e didática da solução, permitindo sua aplicação no ensino de automação, sistemas embarcados e integração industrial. Além disso, a planta desenvolvida possibilita futuras expansões, adaptações e desenvolvimento de projetos como iniciações científicas e novos Trabalhos de Conclusão de Curso, contribuindo para a modernização do laboratório e para o alinhamento do ambiente acadêmico às exigências tecnológicas da Indústria 4.0 no Brasil.

Palavras-chave: Indústria 4.0. Visão computacional embarcada. Sistemas embarcados. Automação industrial. Sensor inteligente.

ABSTRACT

ASSIS, F. **Development of a Didactic Plant with an Intelligent Sensor for Real-Time Material Classification Applied to Industry 4.0.** 2025. Undergraduate Thesis – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2025.

This work presents the development of a low-cost didactic manufacturing cell integrated with an intelligent sensor for real-time material classification, aiming to bring academic training closer to the emerging technologies of Industry 4.0. The proposed system integrates industrial sensors, a conveyor, a Programmable Logic Controller (PLC), and an embedded microcontroller module equipped with a camera and computer vision algorithms. To ensure electrical safety and interoperability with legacy 24 V automation systems, a galvanically isolated interface circuit was designed using optocouplers for both digital and analog channels. The circuits were initially validated through simulation, prototyped on a breadboard, and subsequently consolidated into a Printed Circuit Board (PCB) and manufactured. Results confirm the technical and educational feasibility of the solution, enabling its use in teaching industrial automation and embedded systems. The developed plant also supports future expansions, research projects, and undergraduate theses, contributing to the modernization of the laboratory and to the alignment of academic infrastructure with Industry 4.0 demands in Brazil.

Keywords: Industry 4.0. Embedded computer vision. Embedded systems. Industrial automation. Intelligent sensor.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura do modelo ISA 95	26
Figura 2 – Arquitetura do modelo RAMI 4.0	27
Figura 3 – Pilares da I4.0	28
Figura 4 – Computação de borda	29
Figura 5 – <i>Mecanismo simplificado do optoacoplador.</i>	32
Figura 6 – Topologia de filtro passa-baixa passivo	33
Figura 7 – Filtro passa-baixa RC passivo de segunda ordem	34
Figura 8 – Placa de desenvolvimento ESP32-S3 utilizada no projeto	38
Figura 9 – Planta didática original em diferentes vistas	39
Figura 10 – Circuito de E/S proposto	41
Figura 11 – Circuitos de entrada e saída para simulação	41
Figura 12 – Circuito de entrada digital isolada.	42
Figura 13 – Circuito de saída digital isolada.	42
Figura 14 – Circuito de saída analógica isolada.	43
Figura 15 – Renderizações tridimensionais da PCB projetada.	44
Figura 16 – Diagrama de blocos para o funcionamento geral do software	46
Figura 17 – Resultado das simulações da entrada digital isolada.	55
Figura 18 – Resultado das simulações da saída digital isolada.	56
Figura 19 – Simulação de ponto de operação para ambos os circuitos em ambos estados.	56
Figura 20 – Prototipagem do circuito de interface montado em protoboard.	57
Figura 21 – Sinal observado na entrada digital isolada.	58
Figura 22 – Forma de onda obtida na saída digital isolada.	59
Figura 23 – Sinais PWM filtrados para diferentes ciclos ativos.	63
Figura 24 – Placa fabricada antes da montagem dos componentes	64
Figura 25 – Placa finalizada.	65
Figura 26 – Montagem final da planta didática sobre base rígida.	66
Figura 27 – Bloco e para interpretação do sinal recebido.	67

LISTA DE TABELAS

Tabela 1 – Pilares Tecnológicos da Indústria 4.0	28
--	----

LISTA DE ABREVIATURAS E SIGLAS

I4.0	Indústria 4.0
I3.0	Indústria 3.0
IoT	Internet of Things
IIoT	Industrial Internet of Things
IT	Information Technology
OT	Operational Technology
RAMI 4.0	Reference Architectural Model for Industry 4.0
SCADA	Supervisory Control and Data Acquisition
OPC-UA	Open Platform Communications – Unified Architecture
MQTT	Message Queuing Telemetry Transport
VPU	Vision Processing Unit
NPU	Neural Processing Unit
GPU	Graphics Processing Unit
IA	Inteligência Artificial
E/S	Entrada e Saída
LED	Light Emitting Diode
Hz	Hertz
DC	Corrente Contínua (Direct Current)
dB	Decibel
PWM	Pulse Width Modulation
CLP	Controlador Lógico Programável
PCB	Printed Circuit Board
OLED	Organic Light Emitting Diode
I2C	Inter-Integrated Circuit

CI	Circuito Integrado
HMI	Human-Machine Interface
SMD	Surface-Mount Device
PID	Proporcional, Integral e Derivativo

SUMÁRIO

1	INTRODUÇÃO	21
1.1	Objetivos	22
1.1.1	Geral	22
1.1.2	Específicos	23
2	FUNDAMENTAÇÃO TEÓRICA	25
2.1	A Quarta Revolução Industrial	25
2.1.1	Introdução à Quarta Revolução Industrial	25
2.1.2	Da Automação Hierárquica à Produção Ciberfísica	26
2.1.3	Pilares Tecnológicos Fundamentais da Indústria 4.0	27
2.2	Computação de Borda	28
2.2.1	Microcontroladores	30
2.2.2	Optoacopladores e Isolamento Galvânico	30
2.2.2.1	O Mecanismo do Optoacoplador	31
2.3	Filtros	32
2.3.1	Filtro Passa-Baixa	32
2.3.2	Filtro RC Passa-Baixa	33
2.3.3	Conversão de PWM para Sinal Analógico	34
3	MATERIAIS E MÉTODOS	37
3.1	ESP32	37
3.1.0.1	Simulação e Validação dos Circuitos	39
3.2	Projeto do Circuito de Interface e Integração com a Planta de Auto- mação	39
3.2.1	Integração do Sistema	40
3.2.2	E/S do Sensor de Classificação	40
3.2.2.1	Simulação e Validação dos Circuitos	40
3.2.2.2	Entrada Digital Isolada	40
3.2.2.3	Saída Digital Isolada	41
3.2.2.4	Saída Analógica	41
3.3	Desenvolvimento e Projeto da Placa de Circuito Impresso (PCB) . .	42
3.4	Arquitetura de Software	44
3.4.1	Visão Geral e Principais Módulos	45
3.4.2	Resumo do Fluxo de Execução	45
3.4.3	Inicialização do Sistema	45
3.4.4	Configuração e Controle da Câmera	47
3.4.5	Inferência com Edge Impulse	48

3.4.6	Máquina de Estados e Menus	50
3.4.7	HMI: OLED e Botões	51
3.4.8	Trigger por Entrada e Check de Botões	52
3.4.9	Controle de Saídas e Comunicação TCP	53
4	RESULTADOS	55
4.1	Simulações	55
4.2	Prototipagem	57
4.2.0.1	Entrada Digital Isolada	57
4.2.0.2	Saída Digital Isolada	58
4.2.0.3	Saída Analógica Filtrada	58
4.3	Fabricação	58
4.4	Montagem Final da Planta Didática	60
4.5	Validação da Montagem e Operação da Planta	60
5	CONCLUSÃO	69
	REFERÊNCIAS	71
	APÊNDICES	73
	APÊNDICE C – Código-Fonte do Firmware do Sistema	82
	APÊNDICE D – Código-Fonte do Bloco de Interpretação	124

1 INTRODUÇÃO

A I4.0 representa uma mudança significativa no paradigma da manufatura, marcada pela integração de tecnologias digitais, conectividade e sistemas ciberfísicos capazes de operar com autonomia. Desde sua introdução como estratégia industrial na Alemanha, a I4.0 vem remodelando cadeias produtivas, modelos de negócio e processos de tomada de decisão ao incorporar elementos como Internet das Coisas (IoT), análise avançada de dados e integração entre máquinas e infraestruturas computacionais (FOLGADO et al., 2024; GONÇALVES, 2024). Nesse cenário, os sistemas de produção passam a utilizar dados em tempo real para otimizar suas operações, antecipar falhas e promover maior flexibilidade na manufatura.

A evolução tecnológica que sustenta esse novo paradigma depende diretamente da convergência entre a Tecnologia da Informação (IT) e a Tecnologia Operacional (OT). Enquanto a IT se concentra na manipulação, análise e comunicação de dados, a OT atua no controle de processos físicos, tradicionalmente compostos por sistemas legados e tensões industriais padronizadas, como 24 V. A integração entre esses dois mundos é indispensável para o funcionamento de sistemas ciberfísicos e fábricas inteligentes (DERA; TALAŠKA; DŁUGOSZ, 2026; KAHVECI et al., 2022). Entretanto, barreiras como protocolos proprietários, diferenças estruturais e requisitos rigorosos de confiabilidade continuam a representar obstáculos para a adoção plena da I4.0, especialmente em países emergentes.

Nesse contexto, tecnologias de computação distribuída têm ganhado destaque, particularmente a Computação de Borda (*Edge Computing*). Esse paradigma desloca o processamento de dados da nuvem para dispositivos localizados próximos às fontes de informação, reduzindo latência, consumo de banda e dependência de infraestruturas remotas (CAO et al., 2020; KAHVECI et al., 2022). Em aplicações industriais críticas, a capacidade de responder em tempo hábil é essencial, especialmente em tarefas como inspeção visual, controle de robôs colaborativos e manutenção preditiva (SHARMA; TOMAR; KUSHWAH, 2024; LEE et al., 2020).

Entre esses sensores, aqueles baseados em visão computacional embarcada destacam-se por sua capacidade de extrair características complexas de objetos sem contato físico, sendo amplamente aplicados em inspeção de qualidade, classificação e rastreamento. Plataformas de baixo custo têm viabilizado o desenvolvimento de soluções embarcadas de visão computacional e inferência de modelos de aprendizado de máquina, potencializando aplicações de IA em ambientes industriais e educacionais (PETROSINO et al., 2021; DERA; TALAŠKA; DŁUGOSZ, 2026).

Apesar desses avanços, a integração de sensores inteligentes com sistemas industriais de 24 V requer interfaces elétricas adequadas e isolamento galvânico garantindo proteção ao hardware embarcado e preservação da segurança operacional (DERA; TALAŠKA; DŁUGOSZ, 2026).

Diante desse cenário, este trabalho propõe o Desenvolvimento de uma Planta Didática com Sensor Inteligente para Classificação Automática de Materiais em Tempo Real, utilizando visão computacional embarcada, técnicas de classificação de imagens e uma interface eletrônica isolada capaz de se comunicar com sistemas industriais. A planta didática visa aproximar estudantes e pesquisadores das tecnologias da Indústria 4.0, oferecendo um ambiente modular, acessível e alinhado às demandas atuais da automação industrial (GONÇALVES, 2024).

A solução proposta integra hardware dedicado, software embarcado, comunicação industrial e sistemas pneumáticos e mecânicos, constituindo uma célula completa de inspeção e separação de peças. Ao unir conceitos de visão computacional, computação de borda, isolamento galvânico e automação convencional, o projeto contribui tanto para a modernização do ensino quanto para a compreensão prática dos desafios da convergência IT/OT.

Diversos pesquisadores vêm desenvolvendo projetos didáticos que integram automação e microcontroladores no ensino de conceitos da Indústria 4.0. Santos (2024) propõem uma planta didática modular acessível remotamente, permitindo que estudantes realizem experimentos de controle PID via internet. Pugliese (2022) modelaram e construíram uma planta de baixo custo com microcontrolador e sensores ultrassônicos para ensino de sistemas dinâmicos multivariáveis. Pajpach (2022) criaram um kit educacional de baixo custo baseado, aprendizado de máquina e visão computacional para ensinar inspeção de qualidade em tarefas da Indústria 4.0.

Outros trabalhos focam diretamente a integração de IoT à automação. Grandinetti (2022) desenvolveu hardware que aciona remotamente CLPs via rede sem fio, permitindo controlar um motor trifásico por navegador. Matias (2022) elaboraram uma interface de programação didática para IoT, disponibilizando códigos e tutoriais web para apoiar estudantes no aprendizado da programação de dispositivos conectados para a Indústria 4.0.

Esses exemplos demonstram a tendência crescente de integrar plataformas acessíveis e IoT ao ensino de automação, preparando alunos para ambientes industriais inteligentes.

Nos capítulos seguintes, são detalhados os fundamentos teóricos, a metodologia de projeto, o desenvolvimento eletrônico e computacional, além da validação experimental da planta didática proposta.

1.1 Objetivos

1.1.1 Geral

Projetar e desenvolver uma planta didática com sensor inteligente para classificação de materiais em tempo real, visando baixo custo e integração com sistemas legados, aproximando tais sistemas da I4.0. Com isso, espera-se atualizar o currículo de ensino para

aproximar o que é lecionado com a realidade das indústrias de ponta no exterior, para, por fim, trazer essa tecnologia de forma madura ao Brasil.

1.1.2 Específicos

Modernizar o laboratório de automação e controle de processos industriais com tecnologias mais próximas à realidade atual das indústrias de ponta e assim beneficiar os alunos do curso com uma aparelhagem mais moderna.

Permitir a adição de novas modificações à planta e ao sensor para outras tarefas, estimulando atividades como iniciações científicas e trabalhos de conclusão de curso.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 A Quarta Revolução Industrial

2.1.1 Introdução à Quarta Revolução Industrial

A Quarta Revolução Industrial representa uma transformação nos paradigmas dos processos de produção e manufatura. O conceito foi formalmente introduzido na Feira de Hannover, Alemanha, em 2011, como um componente central de uma estratégia de alta tecnologia do governo alemão, visando aumentar a competitividade de sua indústria de manufatura (SHARMA; TOMAR; KUSHWAH, 2024; GONÇALVES, 2024; FOLGADO et al., 2024).

O objetivo central da I4.0 é a criação de fábricas inteligentes (*Smart Factories*), fundamentadas na integração de Sistemas Ciberfísicos que conectam o mundo físico dos processos de produção com o mundo digital da computação e comunicação. A base conceitual desta revolução, conforme detalhado por Kahveci et al. (2022), deriva da necessidade de transformar os sistemas de produção existentes em sistemas autoconscientes (*self-aware*) e autoaprendizes (*self-learning*). Tais sistemas são projetados não apenas para otimizar operações, mas também para antecipar falhas e ajustar-se dinamicamente a novos requisitos de produção e mudanças não previstas. (KAHVECI et al., 2022; FOLGADO et al., 2024; GONÇALVES, 2024).

Para alcançar o estado de fábrica inteligente, a evolução dos sistemas produtivos adota um conjunto de princípios fundamentais de design:

- **Interoperabilidade e Interconexão:** dizem respeito à capacidade de máquinas, sensores, softwares e sistemas industriais se conectarem e trocarem informações de forma padronizada. A interoperabilidade busca garantir que dispositivos distintos operem de maneira integrada. Já a interconexão refere-se ao estabelecimento efetivo desses vínculos por meio de redes e protocolos compartilhados, possibilitando fluxo contínuo de dados. Nesse contexto, a IoT atua como tecnologia habilitadora ao oferecer protocolos padronizados, como MQTT e OPC-UA, que permitem a comunicação entre equipamentos de diferentes fabricantes. Assim, conforme destacado por Lelli (2019), a IoT amplia a interoperabilidade e sustenta a formação de sistemas ciberfísicos conectados, essenciais à I4.0.
- **Transparência da Informação e Virtualização:** A criação de uma cópia virtual da fábrica (Gêmeo Digital), que permite o monitoramento e a simulação de processos, enriquecida por dados de sensores do mundo físico
- **Descentralização e Decisões Autônomas:** A habilidade dos sistemas ciberfísicos de tomar decisões de forma autônoma e executar tarefas com o mínimo de intervenção

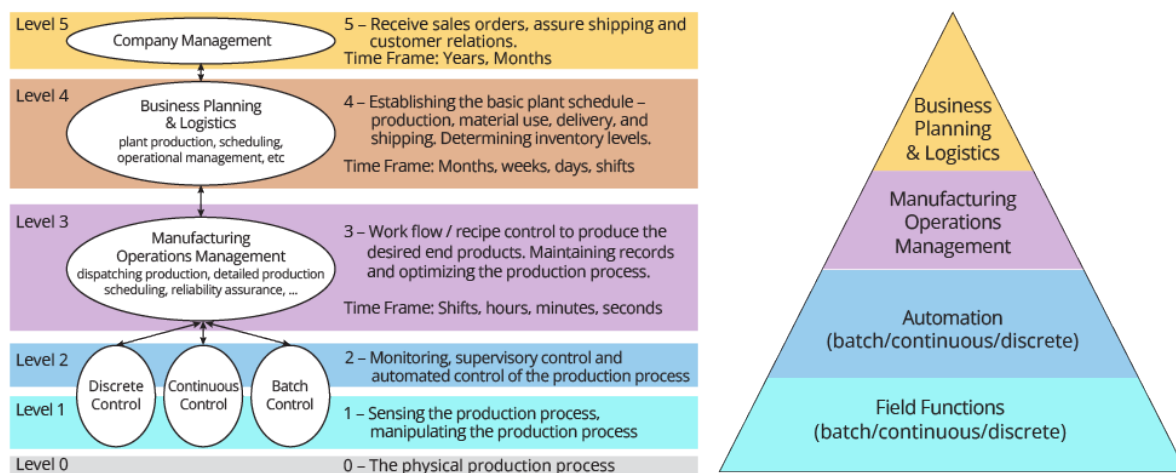
humana

- **Capacidade em Tempo Real:** A coleta e análise de dados de forma instantânea, permitindo reações imediatas a falhas ou mudanças na produção
- **Modularidade:** A capacidade de adaptar a fábrica a diversos requisitos, reconfigurando módulos de produção.

2.1.2 Da Automação Hierárquica à Produção Ciberfísica

A I4.0 representa uma ruptura fundamental com o modelo de automação estabelecido pela Indústria 3.0, definida pela Pirâmide da Automação, uma arquitetura de controle rigidamente hierárquica e centralizada, formalizada pela norma ISA-95 ilustrada pela Figura 1. Este modelo cria silos de informação segregados por níveis: Nível 1 (Sensores e Atuadores), Nível 2 (Controle de Processo), Nível 3 (Operações de Manufatura), e Nível 4 (Planejamento de Negócios) (FOLGADO et al., 2024).

Figura 1 – Arquitetura do modelo ISA 95

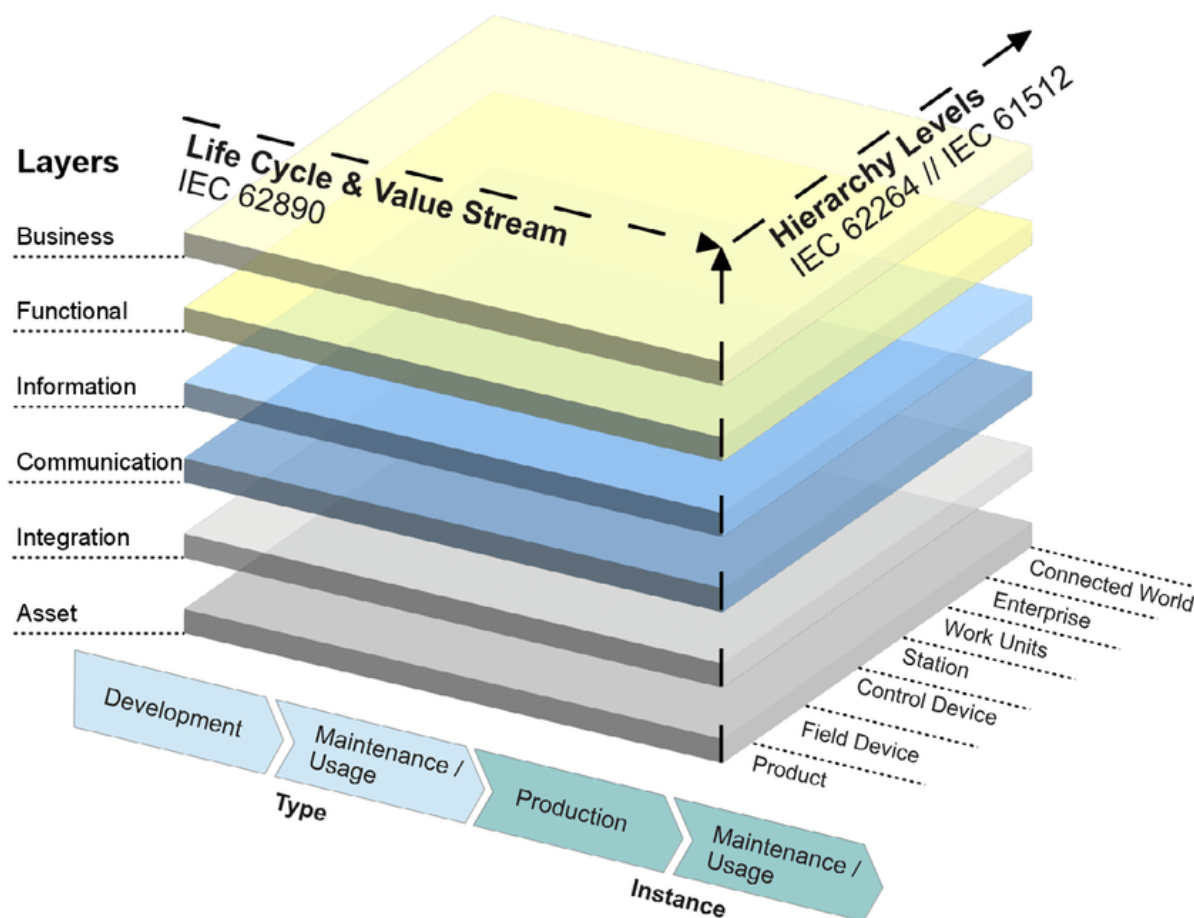


Fonte: (International Society of Automation, 2025)

Em contraste, a I4.0 abandona essa estrutura hierárquica em favor de arquiteturas de rede descentralizadas, interoperáveis e orientadas a serviços. O RAMI 4.0, representado na Figura 2, é frequentemente citado como o novo padrão que substitui a pirâmide, permitindo a comunicação horizontal, máquina-a-máquina, e vertical, chão de fábrica à nuvem, de forma fluida e em tempo real (FOLGADO et al., 2024).

O desafio central dessa transição é a convergência da Tecnologia da Informação, que gerencia dados e computação, e da Tecnologia Operacional, que controla os processos físicos, para o modelo da I4.0 a integração entre IT e OT é condição necessária. A principal barreira para esta convergência, identificada por Kahveci et al. (2022), é o legado da I3.0:

Figura 2 – Arquitetura do modelo RAMI 4.0



Fonte: (LINDNER et al., 2023)

a predominância de protocolos de comunicação de chão de fábrica proprietários, como Profibus e Modbus, que impedem a conectividade aberta.

A Pirâmide da Automação foi projetada para um cenário de escassez de dados, onde apenas informações críticas de controle eram transmitidas verticalmente (FOLGADO et al., 2024). A I4.0, impulsionada pela IoT, gera uma abundância de dados no nível de campo, que agora precisam ser acessados simultaneamente por múltiplos sistemas (plataformas em nuvem, ferramentas de IA na borda, sistemas SCADA). Essa necessidade de acesso ubíquo e descentralizado aos dados quebra a lógica hierárquica da pirâmide, forçando a convergência IT/OT através de padrões abertos de comunicação, como OPC-UA e MQTT (KAHVECI et al., 2022).

2.1.3 Pilares Tecnológicos Fundamentais da Indústria 4.0

A I4.0 é viabilizada por um conjunto de tecnologias que, quando integradas, formam a arquitetura da fábrica inteligente (JAVAID et al., 2022; KAHVECI et al., 2022; SHARMA; TOMAR; KUSHWAH, 2024). Estes pilares vistos na Tabela 1 e Figura 3 descrevem o fluxo de dados e decisão.

Figura 3 – Pilares da I4.0



Fonte: (Tecnicon, 2025)

Tabela 1 – Pilares Tecnológicos da Indústria 4.0

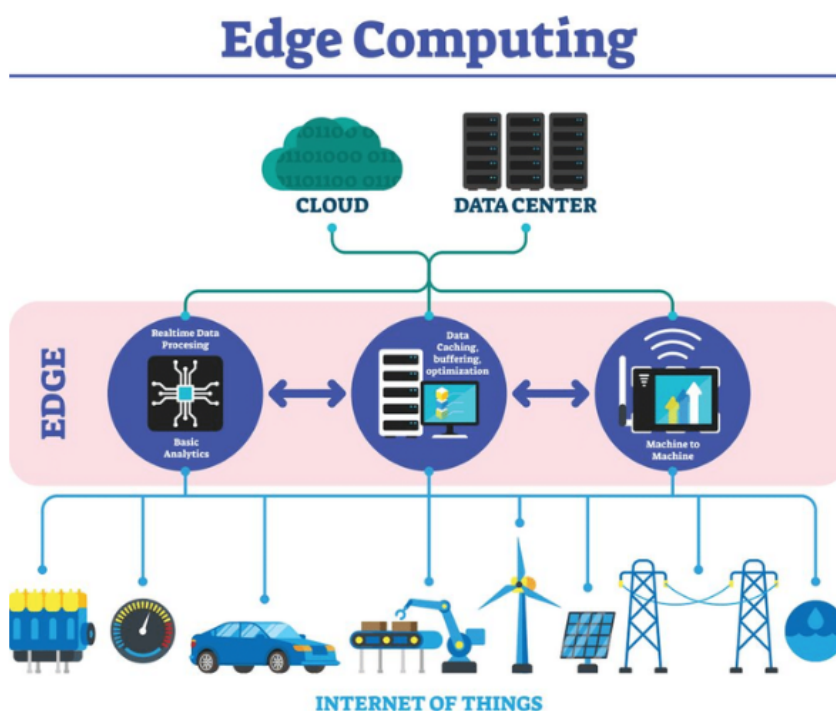
Pilar Tecnológico	Função Primária na I4.0
Internet das Coisas (IoT/IIoT)	Aquisição de Dados (Sensores)
Big Data & Analytics	Análise de Valor (Produtividade, Qualidade)
IA & Machine Learning (ML)	Decisão Autônoma (Aprendizagem)
Computação em Nuvem	Infraestrutura (Armazenamento, Analytics Off-site)
Cibersegurança	Proteção (Sistemas Conectados IT/OT)
Simulação / Gêmeos Digitais	Virtualização (Otimização, Transparência)
Robótica Avançada	Execução Física (Automação Flexível)
Manufatura Aditiva	Produção Flexível (Customização)

Fonte: Elaborado pelo autor com base em Javaid et al. (2022), Kahveci et al. (2022), Folgado et al. (2024) e Sharma, Tomar e Kushwah (2024).

2.2 Computação de Borda

A Computação de Borda é formalmente definida como um paradigma de computação distribuída que posiciona o processamento computacional e o armazenamento de dados estrategicamente próximos às fontes de origem desses dados (KAHVECI et al., 2022; CAO et al., 2020), ilustrado pela Figura 4. No contexto da I4.0 implica em mover a capacidade de computação do *datacenter* centralizado, computação de nuvem (*cloud computing*), para o chão de fábrica, integrando-a em *gateways* IIoT, controladores, ou nos próprios sensores e máquinas (DERA; TALAŚKA; DŁUGOSZ, 2026).

Figura 4 – Computação de borda



Fonte: (Embarcados, 2025)

A arquitetura *cloud computing*, onde os dados do chão de fábrica são enviados à nuvem para processamento, possui riscos quando associada à I4.0, que são resolvidos pela computação de borda:

Latência (Tempo de Resposta): A I4.0 exige reações em tempo real, frequentemente na ordem de milissegundos, para aplicações como controle de robôs colaborativos (SHARMA; TOMAR; KUSHWAH, 2024), detecção de falhas em ferramentas de corte ou inspeção visual de qualidade em alta velocidade (LEE et al., 2020). O tempo de trânsito dos dados é muito longo, limitando o controle em tempo real via nuvem. Ao processar localmente mitiga-se esta limitação, reduzindo a latência e garantindo decisões rápidas (DERA; TALAŚKA; DŁUGOSZ, 2026).

Largura de Banda: A IIoT gera um grande volume de dados, e a transmissão contínua para a nuvem é logisticamente complexa e financeiramente proibitiva (CAO et al., 2020). Realiza-se, portanto, o pré-processamento e filtragem no local, enviando à nuvem apenas dados relevantes, reduzindo o volume de dados transmitidos (KAHVECI et al., 2022).

Segurança, Privacidade e Soberania dos Dados: A computação de borda melhora a privacidade ao manter dados críticos em redes locais, processando-os sem depender de servidores externos (CAO et al., 2020).

A computação de borda não é uma antagonista da Nuvem, visto que a I4.0 opera em um espectro computacional com ambas as arquiteturas (CAO et al., 2020). Nesse modelo híbrido, a Nuvem realiza análises de *big data* em grandes quantidades, enquanto a Borda

foca em inferências locais e processamento em tempo real (KAHVECI et al., 2022).

2.2.1 Microcontroladores

Microcontroladores são dispositivos eletrônicos integrados que reúnem, em um único chip, unidade de processamento, memória e interfaces de entrada e saída. Sua função principal é executar rotinas de controle e processamento de dados de forma autônoma, permitindo o desenvolvimento de sistemas embarcados compactos, eficientes e dedicados a tarefas específicas. Esses dispositivos tornaram-se fundamentais em aplicações modernas de automação, IoT e computação de borda, dada sua capacidade de operar próximo às fontes de dados e atuar diretamente sobre variáveis do processo.

A adoção de microcontroladores em ambientes industriais acompanha a transição descrita pela I4.0, em que sensores, atuadores e máquinas passam a gerar maiores volumes de dados e demandar respostas em tempo real. Nesse contexto, microcontroladores desempenham papéis como: aquisição e pré-processamento de dados, execução de algoritmos de decisão, controle de atuadores e comunicação com sistemas mais complexos, como CLPs ou plataformas SCADA.

Contudo, para que esses dispositivos operem de forma segura em ambientes industriais, é necessário mitigar as diferenças entre o domínio de baixa tensão de suas E/S, tipicamente 3,3 V ou 5 V, e o domínio de 24 V característico de equipamentos de automação. Essa diferença de níveis e o ambiente eletricamente ruidoso exigem o uso de técnicas de isolamento, como optoacopladores, a serem discutidas adiante, garantindo a proteção do hardware embarcado e mantendo a integridade dos sinais.

2.2.2 Optoacopladores e Isolamento Galvânico

A Quarta Revolução Industrial é definida pela convergência de IT e OT (LEE et al., 2020), essa convergência visa criar uma "plataforma de análise de *big data* de ponta a ponta" que utiliza dados em tempo real para apoiar a tomada de decisão em vários estágios dos processos empresariais (KAHVECI et al., 2022). Este novo paradigma é construído sobre dois domínios.

O Inteligente (IT/computação de borda): Este é o domínio da tecnologia orientada a dados. É caracterizado por análises avançadas, plataformas em nuvem e uma tendência significativa para plataformas de hardware modulares, de baixo custo e de código aberto (DERA; TALAŠKA; DŁUGOSZ, 2026; KAHVECI et al., 2022). A análise das arquiteturas modernas da Indústria 4.0 revela uma clara preferência por sistemas embarcados eficientes em termos de custo, como os baseados em *ESP32*, *Arduino* e *Raspberry Pi* (DERA; TALAŠKA; DŁUGOSZ, 2026). Estas plataformas servem como anfitriãs para os módulos de computação sensíveis e de alto valor — VPU, NPU e GPU embarcadas — que executam

a inferência de IA e machine learning em tempo real na borda (PETROSINO et al., 2021). Estes componentes sofisticados operam em baixas tensões DC, tipicamente 3,3V ou 5V.

O da Fábrica (OT): Este é o domínio físico de máquinas de alta potência, equipamentos legados e controladores industriais proprietários (FOLGADO et al., 2024; DERA; TALAŠKA; DŁUGOSZ, 2026). Este ambiente é eletricamente hostil, e sua tensão de sinal padrão é, quase universalmente, 24V. Este domínio está repleto de cargas indutivas de alta energia, como motores, que introduzem ruído ou picos de tensão significativos no sistema elétrico (DERA; TALAŠKA; DŁUGOSZ, 2026).

Isso cria o conflito central de hardware da Indústria 4.0: o Problema 3,3V vs. 24V (DERA; TALAŠKA; DŁUGOSZ, 2026). Uma conexão direta entre o pino de E/S de 3,3V de um microcontrolador ESP32 e um sensor industrial de 24V é um evento destrutivo. Esta divisão exige uma interface que possa tanto traduzir os níveis de tensão quanto proteger o lado sensível de baixa tensão.

O isolamento galvânico é a separação física de dois ou mais circuitos sem um caminho condutivo direto, eliminando possíveis danos em equipamentos sensíveis. Ele garante que os circuitos sejam eletricamente independentes, operando em tensões vastamente diferentes ou estejam conectados a diferentes potenciais de terra. Assim, isola-se fisicamente os circuitos OT e IT, garantindo a proteção dos sistemas embarcados.

2.2.2.1 O Mecanismo do Optoacoplador

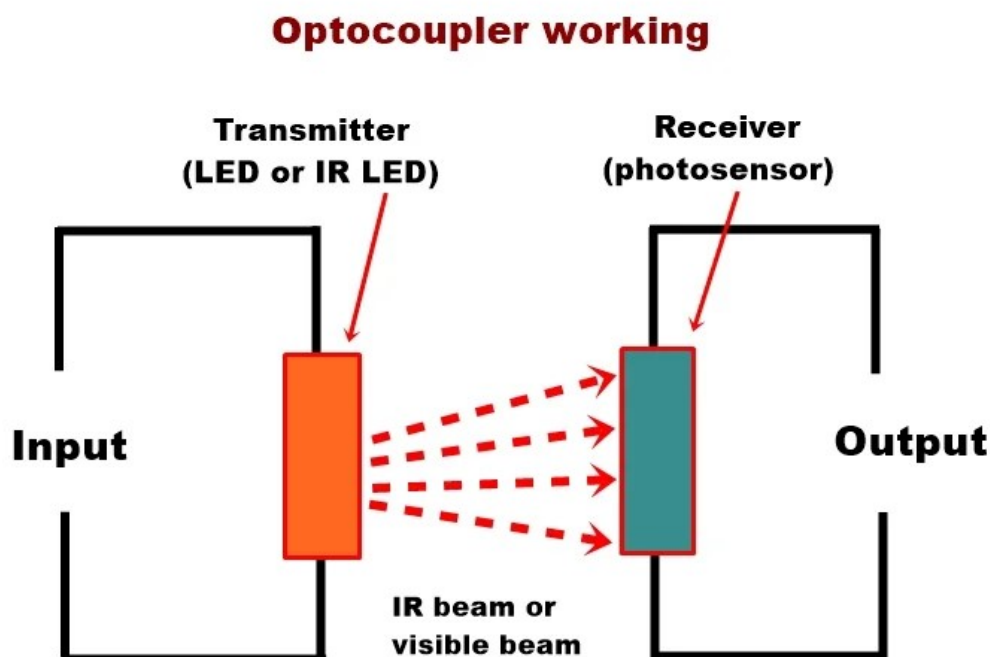
O optoacoplador é o responsável pela interação OT/IT de maneira isolada, esse componente funciona como uma barreira física transponível apenas ao sinal que se deseja transmitir, protegendo o hardware de baixa tensão do ambiente industrial de alta tensão, garantindo assim uma operação segura e confiável (DERA; TALAŠKA; DŁUGOSZ, 2026). Ele funciona convertendo o sinal elétrico em um meio não elétrico — como luz, um campo magnético ou um campo capacitivo — e depois convertendo-o de volta para um sinal elétrico do outro lado de uma barreira dielétrica, como visto na Figura 5. Sua operação consiste em três partes principais:

Estágio de Entrada (Transmissor): O sinal atravessa um resistor limitador de corrente para acionar este LED. Isso efetivamente converte o estado elétrico ligado em um fluxo de fótons.

A Barreira Dielétrica: Uma lacuna física não condutiva que separa o LED do receptor. Esta barreira é o núcleo do isolamento, fornecendo uma alta rigidez dielétrica capaz de suportar milhares de volts.

Estágio de Saída (Receptor): Um dispositivo fotossensível. Quando os fótons do LED atingem a base do dispositivo, permitindo que a corrente flua no circuito isolado.

Figura 5 – Mecanismo simplificado do optoacoplador.



Fonte: (CircuitsPedia, 2025)

2.3 Filtros

Sinais elétricos raramente são puros, frequentemente a informação de interesse está contaminada por componentes indesejados, como ruído de alta frequência, interferência eletromagnética ou ruído da rede elétrica (50/60Hz). A instrumentação de precisão depende da capacidade de medir e processar com exatidão os sinais provenientes de sensores e manter a integridade do sinal é, portanto, um objetivo primordial (SEDRA; SMITH, 2014).

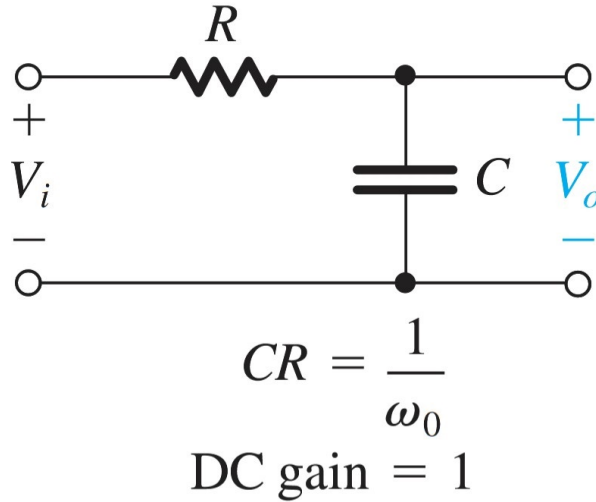
Um filtro é um sistema que atua sobre um sinal, modificando seletivamente partes de seu espectro de frequências. Ao atenuar as frequências indesejadas e permitir a passagem das frequências desejadas, o filtro isola a informação útil.

2.3.1 Filtro Passa-Baixa

Os filtros podem ser categorizados em dois tipos principais: passivos e ativos.

- **Filtros Passivos:** Utilizam apenas componentes passivos (Resistores, Capacitores, Indutores). São simples, robustos e econômicos, e não requerem fonte de alimentação externa, mas não podem amplificar o sinal, ou seja, não possuem ganho.
- **Filtros Ativos:** Utilizam componentes ativos como amplificadores operacionais, além de resistores e capacitores. Eles podem fornecer ganho e oferecer maior controle sobre o sinal.

Figura 6 – Topologia de filtro passa-baixa passivo



Fonte: (SEDRA; SMITH, 2014)

Embora filtros RLC (Resistor–Indutor–Capacitor) sejam eficazes, em frequências mais baixas, os indutores necessários tornam-se fisicamente grandes e caros.

2.3.2 Filtro RC Passa-Baixa

A topologia de um filtro passa-baixa RC é construída com um resistor R em série com o sinal de entrada e um capacitor C em paralelo com a saída como visto na figura 6.

O funcionamento do filtro baseia-se na propriedade da reatância capacitiva (X_C), que é a oposição que um capacitor oferece à passagem de corrente alternada. Essa reatância é inversamente proporcional à frequência (f) do sinal, conforme a equação 2.1 (SEDRA; SMITH, 2014):

$$X_C = \frac{1}{2\pi fC} \quad (2.1)$$

Em que X_c é a reatância, f é a frequência do sinal e C é a capacitância do capacitor. O comportamento do filtro pode ser entendido analisando os extremos de frequência:

- **Para Frequências Baixas e DC ($f \rightarrow 0$):** A reatância capacitiva (X_C) tende ao infinito. O capacitor comporta-se como um circuito aberto. Como o capacitor bloqueia o fluxo de corrente para o terra, o sinal de entrada passa quase integralmente para a saída ($V_{out} \approx V_{in}$).
- **Para Frequências Altas ($f \rightarrow \infty$):** A reatância capacitiva (X_C) tende a zero. O capacitor comporta-se como um curto-circuito para o terra, desviando os sinais de alta frequência, fazendo com que a tensão na saída caia para perto de zero ($V_{out} \approx 0$).

O ponto de transição entre a banda passante e a banda de atenuação é chamado de **frequência de corte** (f_c). É definida como a frequência na qual a potência do sinal de saída é reduzida pela metade, o que corresponde a uma atenuação de -3dB, ou $\frac{1}{\sqrt{2}}$ da amplitude da tensão de entrada.

A frequência de corte de um filtro RC simples é calculada pela equação 2.2 (SEDRA; SMITH, 2014):

$$f_c = \frac{1}{2\pi RC} \quad (2.2)$$

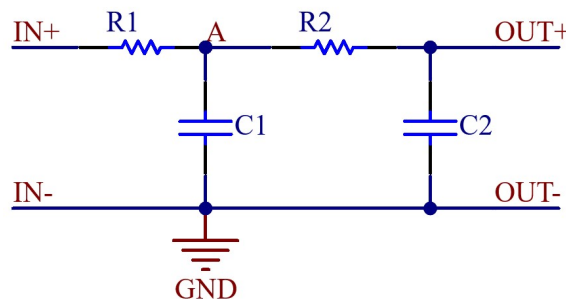
Em que f_c é a frequência de corte, R é a resistência e C é a capacitância.

Para um filtro RC de ordem 2, como visto na figura 7 com $R_1 = R_2 = R$ e $C_1 = C_2 = C$, possui frequência de corte dada pela equação 2.3:

$$f_{c2} = \sqrt{\frac{\sqrt{53} - 7}{2}} f_c \quad (2.3)$$

Em que R é a resistência dos resistores, C é a capacitância dos capacitores, f_c é a frequência de corte do filtro RC de primeira ordem e f_{c2} é a frequência de corte do filtro RC de segunda ordem.

Figura 7 – Filtro passa-baixa RC passivo de segunda ordem



Fonte: Elaborado pelo autor

2.3.3 Conversão de PWM para Sinal Analógico

Uma aplicação prática de filtros passa-baixa é a conversão de um sinal PWM em um sinal analógico.

Muitos microcontroladores não possuem um Conversor Digital–Analógico verdadeiro. Em vez disso, eles simulam uma saída analógica gerando um sinal digital de alta frequência (PWM). O valor analógico é codificado na proporção de tempo em que o sinal fica ligado, conhecido como *duty cycle* (SEDRA; SMITH, 2014).

Este sinal PWM não é uma tensão DC pura ele é, na verdade, um sinal composto por dois componentes:

1. O valor médio (DC), que é a tensão analógica desejada.
2. Um ruído de alta frequência devido a própria frequência de chaveamento do PWM e seus harmônicos.

Uma solução é usar um filtro passa-baixa RC. Definindo-se os valores de R e C para uma frequência de corte (f_c) muito abaixo da frequência de chaveamento do PWM. O filtro, então atenua fortemente o componente de alta frequência, ruído do chaveamento, desviando-o para o terra e deixa passar o componente de baixa frequência, valor DC médio, para a saída.

O capacitor, nesse arranjo, efetivamente calcula a média da onda quadrada digital. A tensão em seus terminais se estabiliza no valor DC correspondente ao *duty cycle* do PWM, transformando um sinal digital pulsado em uma tensão analógica utilizável (SEDRA; SMITH, 2014).

3 MATERIAIS E MÉTODOS

Neste capítulo serão descritos os materiais utilizados e os métodos utilizados para o projeto da planta educacional proposta.

3.1 ESP32

A família ESP32, desenvolvida pela Espressif Systems, é composta por microcontroladores que integram conectividade sem fio, múltiplos periféricos e baixo consumo, características que a tornam amplamente utilizada em aplicações de IoT, automação e computação na borda. De forma geral, os módulos ESP32 oferecem processador dual-core, conexões Wi-Fi e Bluetooth, diversas GPIOs configuráveis, conversores analógicos, temporizadores, interfaces serializadas e suporte a protocolos modernos de comunicação. Essas características tornam a plataforma adequada para a implementação de sensores inteligentes e módulos de aquisição de dados.

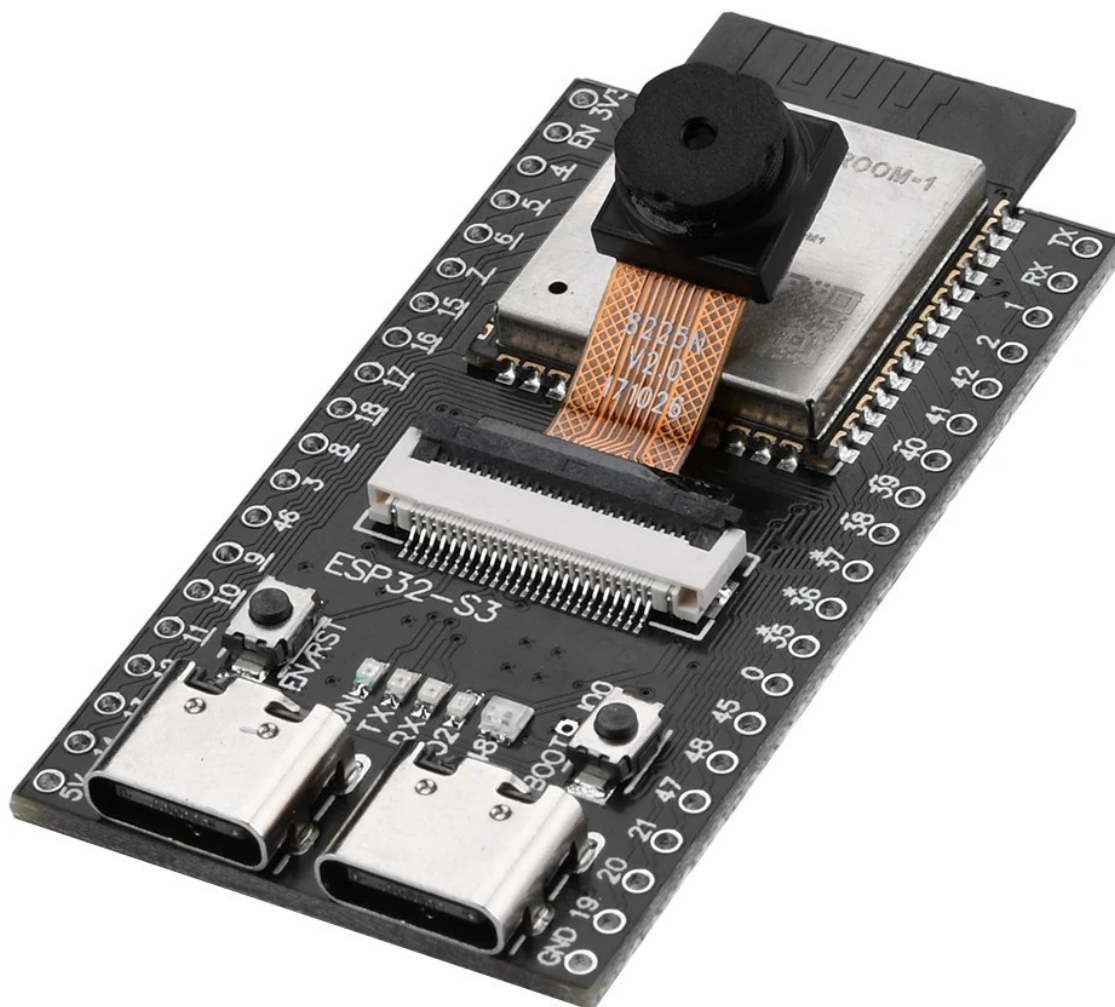
Entre os modelos da família, a versão ESP32-S3, como vista na Figura 8, foi selecionada para este projeto por disponibilizar recursos adicionais relevantes à tarefa de classificação automática de materiais. Esse microcontrolador oferece processamento dual-core de até 240 MHz, suporte aprimorado a operações vetoriais e desempenho superior em aplicações de visão computacional embarcada. Além disso, sua arquitetura facilita a integração com módulos de câmera, permitindo a captura e o pré-processamento de imagens com baixa latência.

Para o funcionamento da planta didática, foram utilizados somente os recursos essenciais da ESP32-S3, conforme listado a seguir:

- **Captura de Imagens:** suporte nativo a sensores de câmera compatíveis, permitindo aquisição contínua de quadros para inferência.
- **Processamento Local:** capacidade de executar o modelo de classificação gerado diretamente no microcontrolador, reduzindo latência e dependência de serviços externos.
- **GPIOs Digitais:** utilizadas para comunicação com o circuito isolado de entrada e saída, permitindo acionar atuadores ou detectar sinais do sistema.
- **Saída PWM:** responsável pela geração do sinal que, após filtragem, compõe a saída analógica proporcional à classe identificada pelo algoritmo.
- **Interface I2C:** empregada para comunicação com a tela OLED utilizada na HMI do sensor inteligente.
- **Conexão Wi-Fi:** utilizada para transmissão dos resultados de classificação ao CLP por meio de comunicação TCP/IP.

A escolha da ESP32-S3 também considerou sua memória interna suficiente para armazenar quadros de imagem e buffers intermediários necessários ao modelo de inferência. Dessa forma, o microcontrolador opera como um sensor inteligente completo, capaz de capturar imagens, processá-las, tomar decisões e comunicar-se com o sistema de automação industrial, atendendo às necessidades da planta de classificação desenvolvida.

Figura 8 – Placa de desenvolvimento ESP32-S3 utilizada no projeto



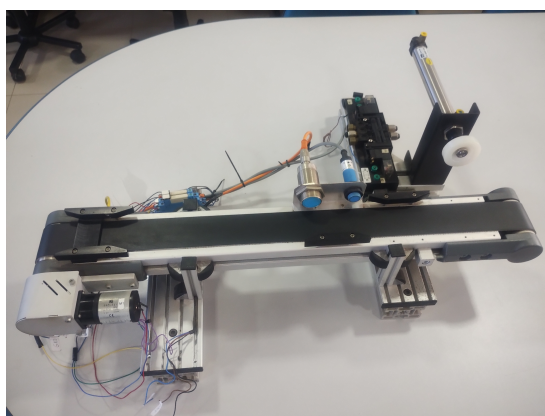
Fonte: (ALIEXPRESS, 2025)

3.1.0.1 Simulação e Validação dos Circuitos

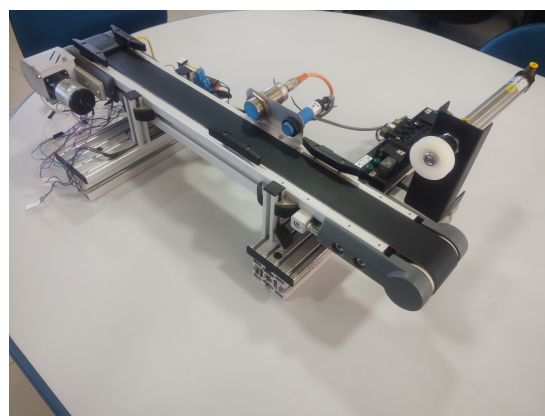
3.2 Projeto do Circuito de Interface e Integração com a Planta de Automação

Este capítulo comenta a planta didática utilizada como base para o desenvolvimento deste trabalho. A Figura 9 ilustra diferentes vistas da esteira em seu estado inicial, permitindo visualizar a disposição dos elementos mecânicos e elétricos do sistema antes das modificações realizadas.

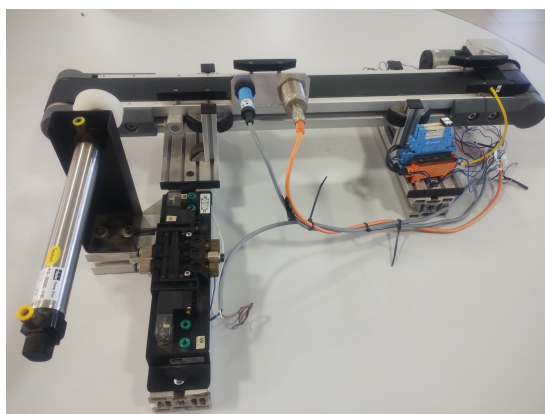
Figura 9 – Planta didática original em diferentes vistas



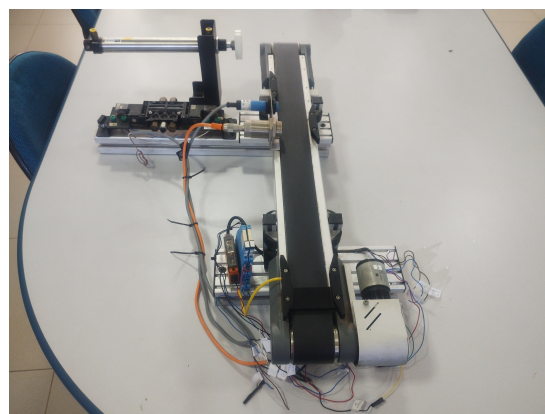
(a) Vista 1



(b) Vista 2



(c) Vista 3



(d) Vista 4

Fonte: Elaborado pelo autor

Para o desenvolvimento desse trabalho, duas plantas didáticas de esteira presentes no Laboratório de Automação da Escola de Engenharia de São Carlos (EESC) serão reaproveitadas, os componentes eletromecânicos forneceram uma base adequada para o desenvolvimento do sistema proposto. Cada planta é composta por uma esteira transportadora, um pistão pneumático acionado por duas saídas digitais e três sensores industriais: um sensor capacitivo, um sensor indutivo e um sensor óptico. Todos os atuadores e sensores são interligados a um módulo controlador, responsável pela conversão dos sinais de entrada e saída e pela comunicação com um computador via interface RS-485.

O objetivo do projeto é compor ambas as esteiras em uma célula de transporte e classificação automática de peças por visão computacional controlada por um CLP. A primeira esteira realiza a inspeção das peças e a tomada de decisão sobre seu destino, utilizando seus sensores originais e um pistão pneumático que atua em conjunto com o sistema de visão para direcionar as peças conforme o resultado da classificação. Além dos sensores já presentes, deseja-se adicionar um novo sensor embarcado de classificação, composto por um microcontrolador com câmera e circuito auxiliar desenvolvido neste projeto. A segunda esteira atua como linha de desvio, recebendo as peças conforme a classificação realizada pela primeira.

3.2.1 Integração do Sistema

Para interconectar os componentes da planta propõe-se um circuito de interface dedicado, responsável pela comunicação entre o CLP, os sensores e os atuadores de 24 V. Esse circuito possui bornes de parafuso para facilitar as conexões, distribuir as linhas de 24 V e terra, e garantir acesso rápido a todos os sinais de entrada e saída do CLP. O circuito final será projetado em uma PCB, essa abordagem modular simplifica o cabeamento, manutenção e futuras expansões do sistema.

O isolamento entre o domínio industrial (24 V) e o domínio lógico do microcontrolador é obtido por meio da separação física dos circuitos e uso de optoacopladores, que protegem o microcontrolador contra ruídos e surtos elétricos. A integração com o CLP permite que os sinais digitais e analógicos do sistema de visão sejam monitorados ou utilizados como variáveis de controle em processos automatizados mais amplos.

3.2.2 E/S do Sensor de Classificação

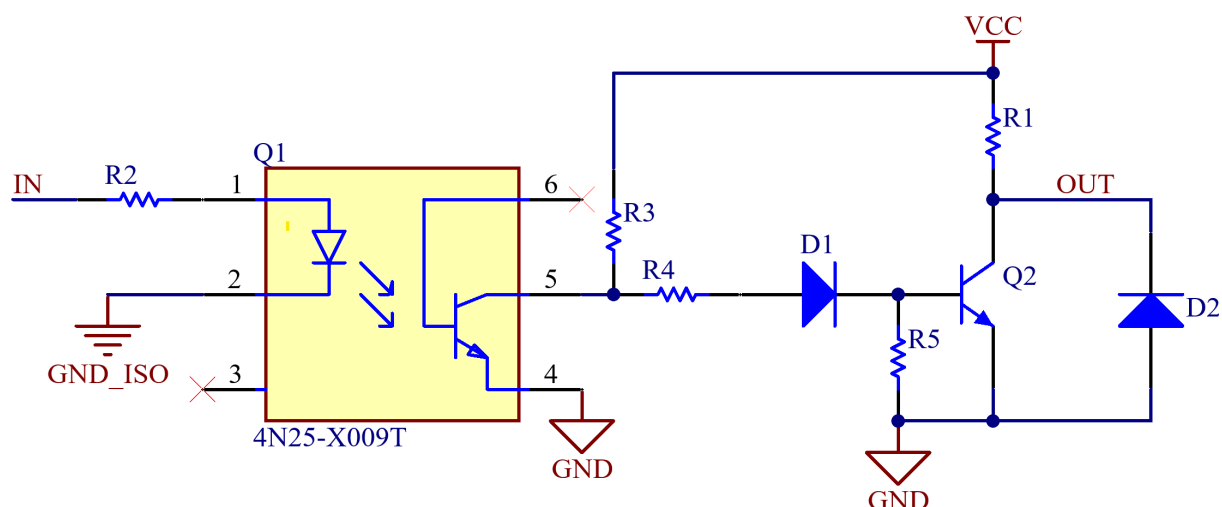
3.2.2.1 Simulação e Validação dos Circuitos

Para maior robustez dos sinais de entrada e saída propõe-se o circuito de chaveamento visto na Figura 10. Antes da prototipagem física, os circuitos de entrada e saída digital da Figura 11 foram simulados para validação de sua resposta elétrica e definição de valores adequados de resistores e componentes auxiliares. Nota-se que chaves foram utilizadas no lugar dos optoacopladores devido a disponibilidade de modelos de simulação. O protótipo conceitual visto na Figura 11) foi composto por dois circuitos: um de entrada isolada de 24 V e saída isolada de 3,3 V e outro com entrada isolada de 3,3 V e saída isolada de 24 V.

3.2.2.2 Entrada Digital Isolada

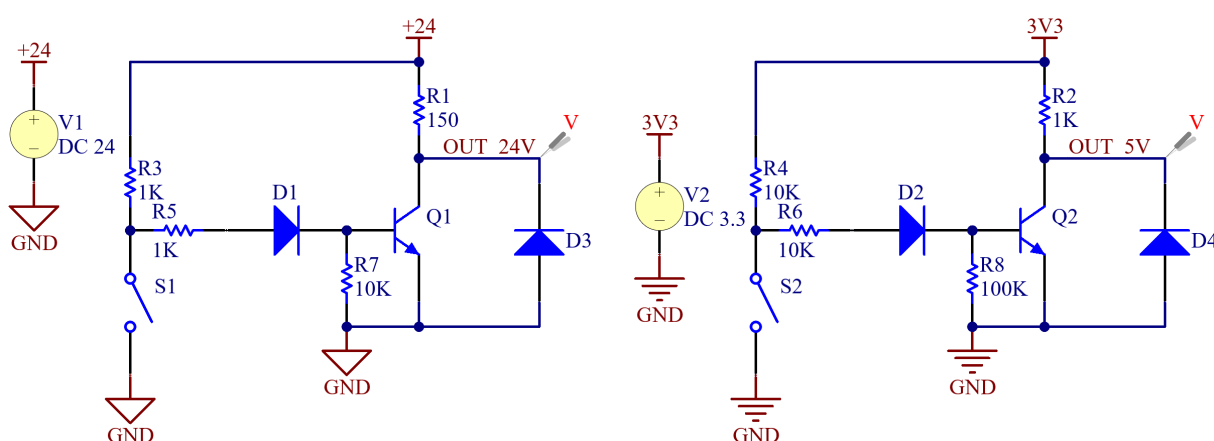
A entrada digital utiliza um optoacoplador do tipo 4N25 para isolar o domínio de 24 V do domínio lógico do microcontrolador. No lado industrial, um resistor limitador de corrente protege o LED do optoacoplador, enquanto no lado lógico o fototransistor atua com um

Figura 10 – Circuito de E/S proposto



Fonte: Elaborado pelo autor

Figura 11 – Circuitos de entrada e saída para simulação



Fonte: Elaborado pelo autor

resistor de *pull-up*, fornecendo nível alto quando inativo e nível baixo quando o sinal industrial está ativo, como visto na Figura 12.

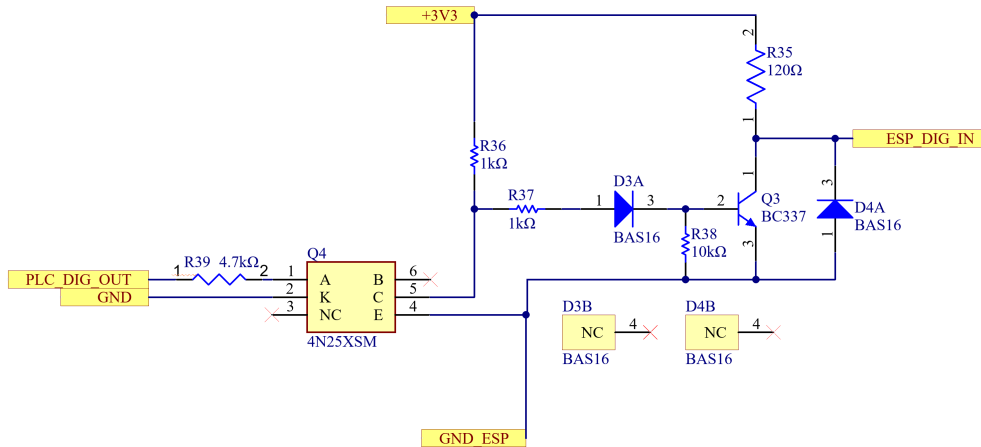
3.2.2.3 Saída Digital Isolada

A saída digital segue a mesma topologia da entrada, mas opera no sentido inverso, como visto na Figura 13.

3.2.2.4 Saída Analógica

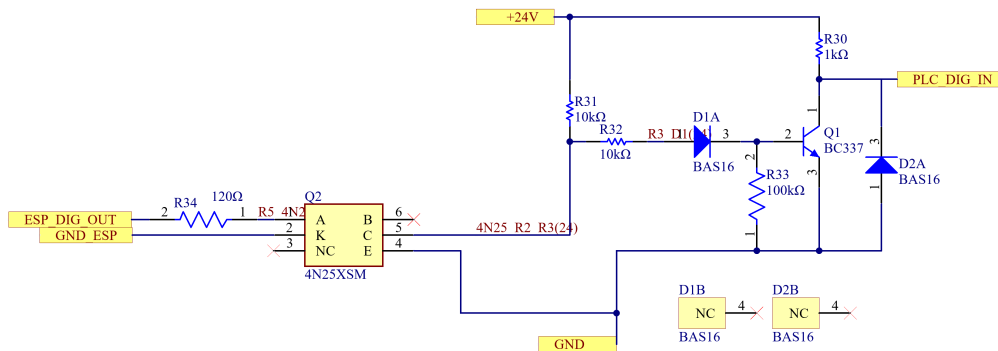
O canal analógico foi projetado para converter o sinal PWM gerado pela *ESP32-S3* e transmitido via optoacoplador de comutação rápida. O circuito utiliza um filtro RC de segunda ordem, seguido de um estágio de *buffer* com amplificador operacional, a frequência de corte do filtro é dada pela Equação 3.1, que está bem abaixo da frequência de chaveamento do PWM, conforme projetado.

Figura 12 – Circuito de entrada digital isolada.



Fonte: Elaborado pelo autor

Figura 13 – Circuito de saída digital isolada.



Fonte: Elaborado pelo autor

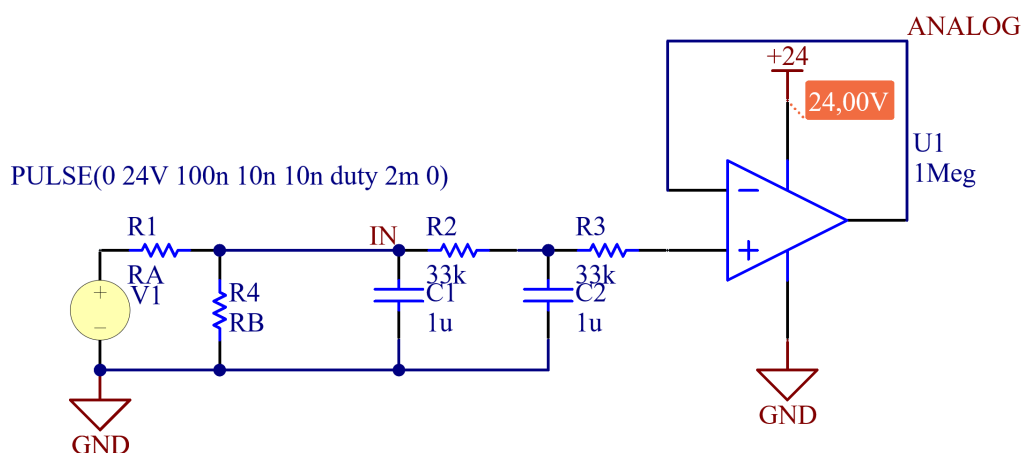
$$f_c = \sqrt{\frac{\sqrt{53} - 7}{2}} \frac{1}{2\pi * 33k * 1u} \approx 3.61Hz \quad (3.1)$$

Um potenciômetro é adicionado para realizar o ajuste fino da tensão do PWM de forma que a saída analógica máxima não supere os 10V. O amplificador operacional, configurado como seguidor de tensão, fornece baixa impedância de saída e estabilidade mesmo sob carga. O circuito proposto pode ser visto na Figura 14

3.3 Desenvolvimento e Projeto da Placa de Circuito Impresso (PCB)

O projeto final de *hardware* foi consolidado em uma única PCB, que integra tanto o módulo de interface quanto o módulo de sensoriamento embarcado. O desenvolvimento do *layout* foi realizado no software *Altium Designer*. As Figuras 15a e 15b apresentam as renderizações tridimensionais da face superior e inferior da PCB, nelas é possível observar a disposição dos componentes e a separação física entre os domínios de 24 V e 3,3/5 V.

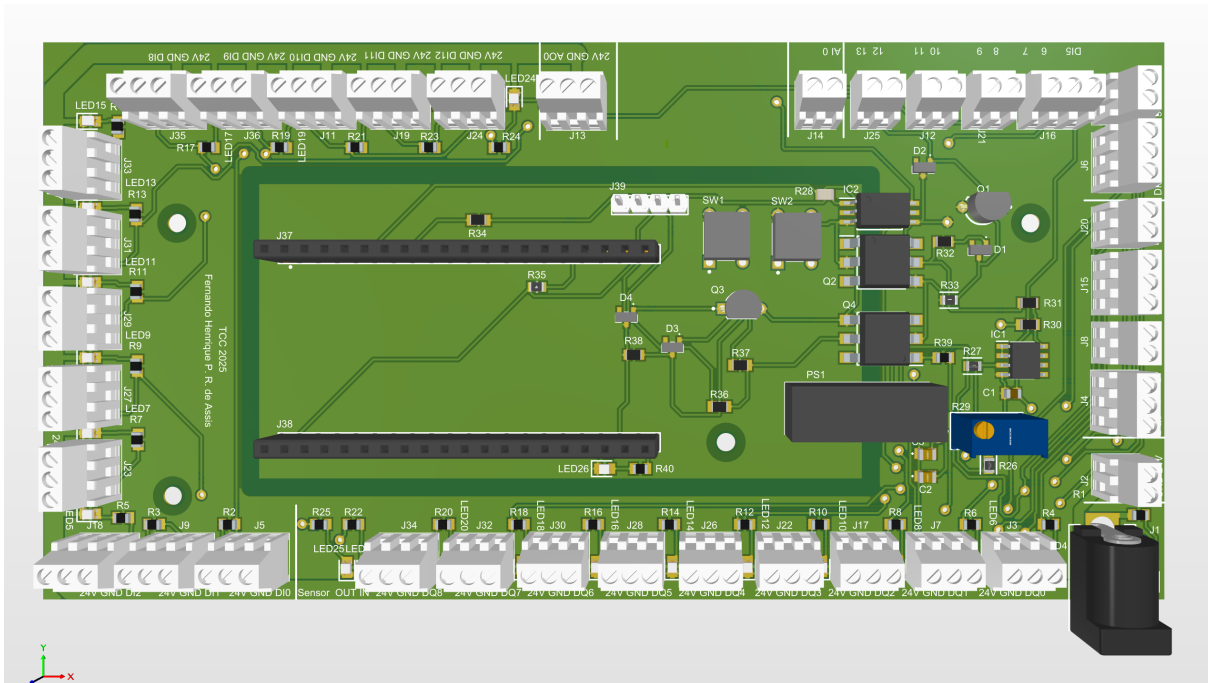
Figura 14 – Circuito de saída analógica isolada.



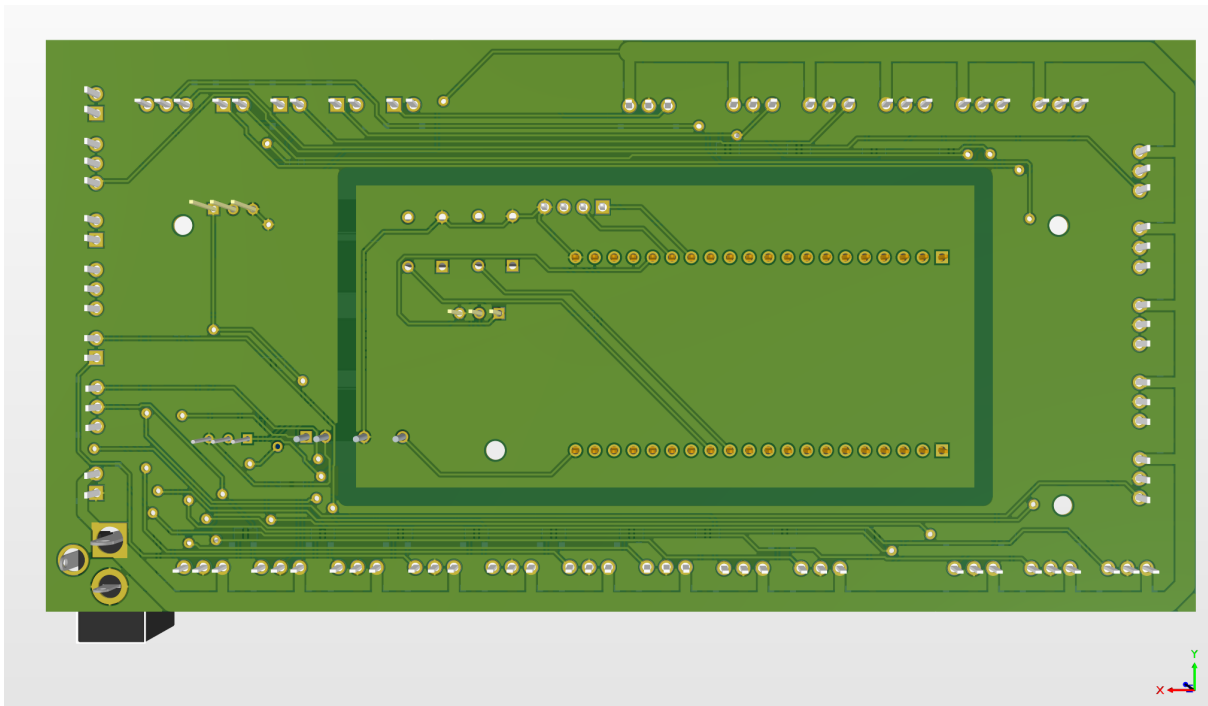
Fonte: Elaborado pelo autor

A placa foi projetada em duas camadas para otimizar custo e simplicidade de fabricação. A face superior concentra todos os componentes, enquanto a parte inferior é dedicada exclusivamente ao roteamento da placa.

Figura 15 – Renderizações tridimensionais da PCB projetada.



(a) Renderização 3D da face superior da PCB.



(b) Renderização 3D da face inferior da PCB.

Fonte: Elaborado pelo autor

3.4 Arquitetura de Software

Esta seção descreve a arquitetura de software implementada no trabalho, documentando as funções, estados e fluxos que compõem o sistema: inicialização, captura de imagem,

inferência (*Edge Impulse*), máquina de estados, HMI (OLED e botões), acionamento de saídas e comunicação TCP com PLC.

3.4.1 Visão Geral e Principais Módulos

O firmware é modularizado nas seguintes camadas funcionais:

- **Inicialização:** configuração de pinos, periféricos (camera, PWM, display) e carregamento de parâmetros persistentes.
- **Aquisição de imagem:** interação com `esp_camera` e buffers usados pela inferência.
- **Inferência:** preparação do sinal e chamada ao classificador gerado pelo Edge Impulse.
- **Máquina de estados e menus:** navegação por menus no OLED, seleção de modos e controle por botões.
- **Saídas e integração:** controle de saída digital, PWM proporcional por classe e envio de resultados via TCP ao PLC.
- **Funções utilitárias:** persistência de IP em Preferences, parse de comandos seriais e rotinas de reconexão.

3.4.2 Resumo do Fluxo de Execução

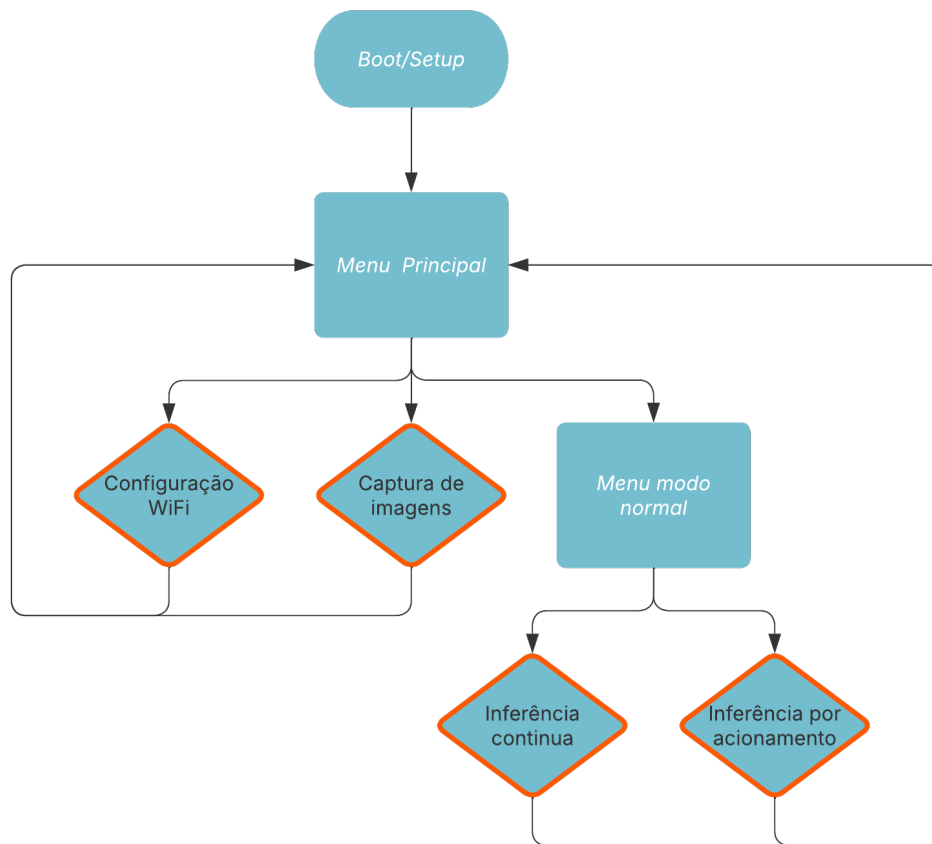
O fluxo de funcionamento do software segue o diagrama visto na Figura 16 como descrito:

1. **Boot / setup:** Periféricos e parâmetros carregados.
2. **Menu Principal:** Permite configurar Wi-Fi, iniciar modo de inferência e iniciar modo de coleta de imagens.
3. **Operação:** Operação normal dependendo do modo escolhido.
4. **Retorno ao estado de espera:** O sistema volta ao menu principal para aguardar nova seleção de modo.

3.4.3 Inicialização do Sistema

A função `setup()` inicializa hardware e variáveis críticas, carregando o IP do PLC salvo e preparando os periféricos. Trecho essencial da função:

Figura 16 – Diagrama de blocos para o funcionamento geral do software



Fonte: Elaborado pelo autor

```

void setup() {
  Wire.begin(45, 48);
  Serial.begin(115200);
  loadPLCIPFromPreferences();
  Serial.println("\nIniciando...");
  pinMode(SELECT_BTN_PIN, INPUT_PULLUP);
  pinMode(NAV_BTN_PIN, INPUT_PULLUP);
  pinMode(INPUT_TRIGGER_PIN, INPUT_PULLDOWN);
  pinMode(DIGITAL_OUTPUT_PIN, OUTPUT);
  digitalWrite(DIGITAL_OUTPUT_PIN, LOW);
  ledcSetup(PWM_CHANNEL, PWM_FREQUENCY, PWM_RESOLUTION);
  ledcAttachPin(PWM_OUTPUT_PIN, PWM_CHANNEL);
  ledcWrite(PWM_CHANNEL, 0);
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  display.clearDisplay();
  display.drawBitmap(0,0,startup_logo,128,64,SSD1306_WHITE);
}
  
```

```

display.display();
delay(LOGO_DISPLAY_TIME);
display.clearDisplay();
display.setTextColor(SSD1306_WHITE);
display.setTextSize(1);
WiFi.mode(WIFI_STA);
Serial.println("Inicializando_cmera...");
display.clearDisplay(); display.println(F("Inicializando_camera...
"));
display.display();
if (!ei_camera_init()) {
    display.clearDisplay(); display.println(F("Erro_na_camera!"));
    display.display();
    delay(2000);
} else {
    Serial.println("Cmera_inicializada");
}
configMode = true; configMenuIndex = 0;
updateConfigMenu();
memset(tcpBuffer, 0, MAX_ARRAY_SIZE);
memset(lastSentTcpBuffer, 0, MAX_ARRAY_SIZE);
isTcpConnected = false;
}

```

3.4.4 Configuração e Controle da Câmera

A inicialização do sensor é encapsulada em `ei_camera_init()`, que chama a API do `esp_camera` com a estrutura `camera_config`. A função marca o estado de inicialização para evitar reinicializações desnecessárias:

```

bool ei_camera_init(void) {
    if (is_initialised) return true;
    esp_err_t err = esp_camera_init(&camera_config);
    if (err != ESP_OK) {
        Serial.printf("Camera_init_failed_with_error_0x%x\n", err);
        return false;
    }
    sensor_t *s = esp_camera_sensor_get();
    is_initialised = true;
    return true;
}

```

A captura e o redimensionamento/crop para o formato utilizado pelo classificador ficam em `ei_camera_capture(...)`, que obtém `camera_fb_t` e converte para RGB888:

```
bool ei_camera_capture(uint32_t img_width, uint32_t img_height,
    uint8_t *out_buf) {
    bool do_resize = false;
    if (!is_initialised) {
        Serial.println("ERR:_Camera_is_not_initialized");
        return false;
    }
    camera_fb_t *fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Failed_to_get_camera_frame_buffer");
        return false;
    }
    bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT_JPEG,
        snapshot_buf);
    esp_camera_fb_return(fb);
    if (!converted) {
        Serial.println("Convert_failed");
        return false;
    }
    if (img_width != EI_CAMERA_RAW_FRAME_BUFFER_COLS || img_height !=
        EI_CAMERA_RAW_FRAME_BUFFER_ROWS) {
        ei::image::processing::crop_and_interpolate_rgb888(snapshot_buf,
            EI_CAMERA_RAW_FRAME_BUFFER_COLS,
            EI_CAMERA_RAW_FRAME_BUFFER_ROWS,
            out_buf ? out_buf : snapshot_buf,
            img_width, img_height);
    } else {
        if (out_buf) memcpy(out_buf, snapshot_buf,
            EI_CAMERA_FRAME_BYTE_SIZE * EI_CAMERA_RAW_FRAME_BUFFER_COLS *
            EI_CAMERA_RAW_FRAME_BUFFER_ROWS);
    }
    return true;
}
```

3.4.5 Inferência com Edge Impulse

A função `runInference()` realiza o pré-processamento, chamada ao classificador e pós-processamento (decisão, preparo do payload TCP e acionamento de saídas). Abaixo

está o trecho essencial responsável por executar a inferência e interpretar resultados adaptado do original:

```
void runInference() {
    objectDetected = false;
    currentClassIndex = -1;
    digitalWrite(DIGITAL_OUTPUT_PIN, LOW);
    ledcWrite(PWM_CHANNEL, 0);
    if (!ei_camera_capture(EI_CLASSIFIER_IMAGE_HEIGHT,
        EI_CLASSIFIER_IMAGE_WIDTH, snapshot_buf)) {
        Serial.println("Failed_to_capture_image_for_inference");
        return;
    }
    signal_t signal;
    signal.total_length = EI_CLASSIFIER_IMAGE_WIDTH *
        EI_CLASSIFIER_IMAGE_HEIGHT * EI_CAMERA_FRAME_BYTE_SIZE;
    signal.get_data = ei_camera_get_data;
    ei_impulse_result_t result;
    run_classifier(&signal, &result, debug_nn);
    #if EI_CLASSIFIER_OBJECT_DETECTION == 1
        recognitionLabel = "";
        recognitionConfidence = 0.0;
        String tcpData = "";
        for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
            ei_impulse_result_bounding_box_t bb = result.bounding_boxes[i];
            if (bb.value == 0) continue;
            if (bb.value > recognitionConfidence) {
                recognitionLabel = String(bb.label);
                recognitionConfidence = bb.value;
                objectDetected = true;
                currentClassIndex = 0;
                digitalWrite(DIGITAL_OUTPUT_PIN, HIGH);
            }
        }
    #else
        float highestConfidence = 0.0;
        int bestIdx = -1;
        for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
            if (result.classification[i].value > highestConfidence) {
                highestConfidence = result.classification[i].value;
                bestIdx = i;
            }
        }
    #endif
}
```

```

}
if (bestIdx >= 0) {
    recognitionLabel = String(ei_classifier_inferencing_categories[
bestIdx]);
    recognitionConfidence = highestConfidence;
    messageText = recognitionLabel + "_" + String(int(
recognitionConfidence * 100)) + "%";
    if (recognitionConfidence > 0.5) {
        objectDetected = true;
        currentClassIndex = bestIdx;
        digitalWrite(DIGITAL_OUTPUT_PIN, HIGH);
    }
} else {
    messageText = "Classificacao_falhou";
}
#endif
if (currentClassIndex >= 0) {
    int pwmDutyCycle = ((1 << PWM_RESOLUTION) - 1) / (
EI_CLASSIFIER_LABEL_COUNT + 1) * (currentClassIndex + 1);
    ledcWrite(PWM_CHANNEL, pwmDutyCycle);
}
if (wifiConnected && isTcpConnected) {
    String payload = recognitionLabel + ":" + String(
recognitionConfidence * 100, 1);
    sendDataToTCP(payload);
}
}

```

3.4.6 Máquina de Estados e Menus

A navegação e operação do sistema são implementadas como uma máquina de estados. O `loop()` dirige o comportamento por estados e flags (modo de configuração, coleta de imagens, modo normal, modo por entrada, etc.):

```

void loop() {
    if (!configMode && !imageCollectionMode) {
        if (WiFi.status() == WL_CONNECTED && !wifiConnected) {
            wifiConnected=true; updateDisplay(); }
        else if (WiFi.status() != WL_CONNECTED && wifiConnected) {
            wifiConnected=false; updateDisplay(); }
    }
}

```

```

checkButtons();
if (!configMode && !imageCollectionMode && wifiConnected) {
    checkTcpConnection();
    if (!isTcpConnected) connectToTCP();
}
if (!configMode && !normalModeSubmenu && !imageCollectionMode) {
    checkInputTrigger();
    bool shouldRun = inputBasedMode ? triggerPressed
                                   : (millis() - lastInferenceTime
>= INFERENCE_INTERVAL);
    if (shouldRun) {
        runInference();
        updateDisplay();
        lastInferenceTime = millis();
        triggerPressed = false;
    }
}
if (!configMode && !imageCollectionMode) checkForCommands();
delay(10);
}

```

3.4.7 HMI: OLED e Botões

A interação com o usuário é feita via display OLED e dois botões. A rotina que atualiza o menu principal no OLED está em `updateConfigMenu()`:

```

void updateConfigMenu() {
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 2);
    display.print(F("Menu_de_Configuracao"));
    display.drawLine(0, 12, SCREEN_WIDTH, 12, SSD1306_WHITE);
    for (int i = 0; i < CONFIG_MENU_ITEMS; i++) {
        display.setCursor(10, 16 + i * 10);
        if (i == configMenuIndex) display.print(F(">_"));
        else display.print(F("_"));
        display.println(CONFIG_MENU_OPTIONS[i]);
    }
    display.drawLine(0, 50, SCREEN_WIDTH, 50, SSD1306_WHITE);
    display.setCursor(2, 54);
    display.print(F("Nav:_Mudar:_Sel:_OK"));
}

```

```
display.display();
}
```

3.4.8 Trigger por Entrada e Check de Botões

A detecção de trigger externo (borda de subida) é tratada por `checkInputTrigger()`:

```
void checkInputTrigger() {
    if (!inputBasedMode) return;
    bool currentTriggerState = digitalRead(INPUT_TRIGGER_PIN);
    if (lastTriggerState == LOW && currentTriggerState == HIGH) {
        triggerPressed = true;
    }
    lastTriggerState = currentTriggerState;
}
```

A rotina de leitura dos botões, com distinção entre pressionamento curto e longo (10s para abrir o menu de configuração), está implementada em `checkButtons()` (trecho condensado abaixo):

```
void checkButtons() {
    if (digitalRead(SELECT_BTN_PIN) == LOW) {
        if (!selectBtnPressed) {
            selectBtnPressed = true;
            buttonPressedTime = millis();
        } else {
            if (!configMode && !imageCollectionMode && millis() -
buttonPressedTime >= 10000) {
                configMode = true;
                configMenuIndex = 0;
                updateConfigMenu();
                selectBtnPressed = false;
                waitForButtonRelease = true;
            }
        }
    } else {
        if (selectBtnPressed) {
            if (configMode && !waitForButtonRelease && (millis() -
buttonPressedTime < 10000)) {
                if (normalModeSubmenu) processNormalModeSelection();
                else processConfigMenuSelection();
            }
        }
    }
}
```

```

        selectBtnPressed = false;
    }
}
bool navBtnReading = digitalRead(NAV_BTN_PIN) == LOW;
if (navBtnReading && (millis() - navBtnLastPresTime) > 200) {
    if (!configMode) {
        configMenuIndex = (configMenuIndex + 1) % CONFIG_MENU_ITEMS;
        updateConfigMenu();
    }
    navBtnLastPresTime = millis();
}
}

```

3.4.9 Controle de Saídas e Comunicação TCP

A conexão TCP ao PLC é gerida por `connectToTCP()`, que tenta conectar periodicamente e marca o estado `isTcpConnected`:

```

void connectToTCP() {
    if (isTcpConnected) return;
    static unsigned long lastConnectionAttempt = 0;
    if (millis() - lastConnectionAttempt < 5000) return;
    lastConnectionAttempt = millis();
    if (tcpClient.connected()) {
        tcpClient.stop();
        delay(100);
    }
    if (tcpClient.connect(plc_ip, PLC_PORT)) {
        isTcpConnected = true;
        lastTcpActivityTime = millis();
    } else {
        isTcpConnected = false;
    }
}

```

Para enviar resultados é usada `sendDataToTCP(String data)`, que encapsula os dados entre STX/ETX e os envia:

```

void sendDataToTCP(String data) {
    if (!isTcpConnected || !tcpClient.connected()) {
        isTcpConnected = false;
        return;
    }
}

```

```

}
if (data.length() > MAX_ARRAY_SIZE - 5) data = data.substring(0,
    MAX_ARRAY_SIZE - 5);
memset(tcpBuffer, 0, MAX_ARRAY_SIZE);
tcpBuffer[0] = 0x02;
int stringLength = data.length();
for (int i = 0; i < stringLength; i++) tcpBuffer[i + 1] = data.
    charAt(i);
tcpBuffer[stringLength + 1] = 0x03;
int actualDataLength = stringLength + 2;
int bytesSent = tcpClient.write(tcpBuffer, actualDataLength);
if (bytesSent == actualDataLength) {
    memcpy(lastSentTcpBuffer, tcpBuffer, MAX_ARRAY_SIZE);
    hasValidLastTcpData = true;
    lastTcpActivityTime = millis();
} else {
    if (!tcpClient.connected()) {
        isTcpConnected = false;
        tcpClient.stop();
    }
}
}

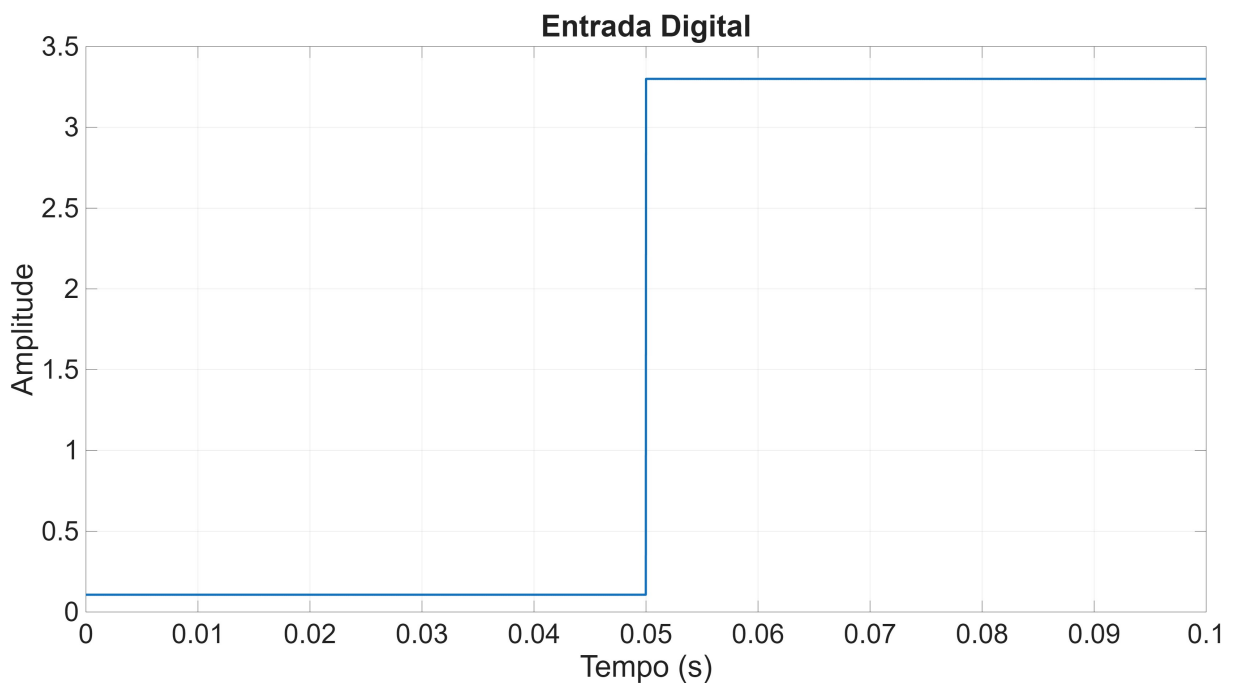
```

4 RESULTADOS

4.1 Simulações

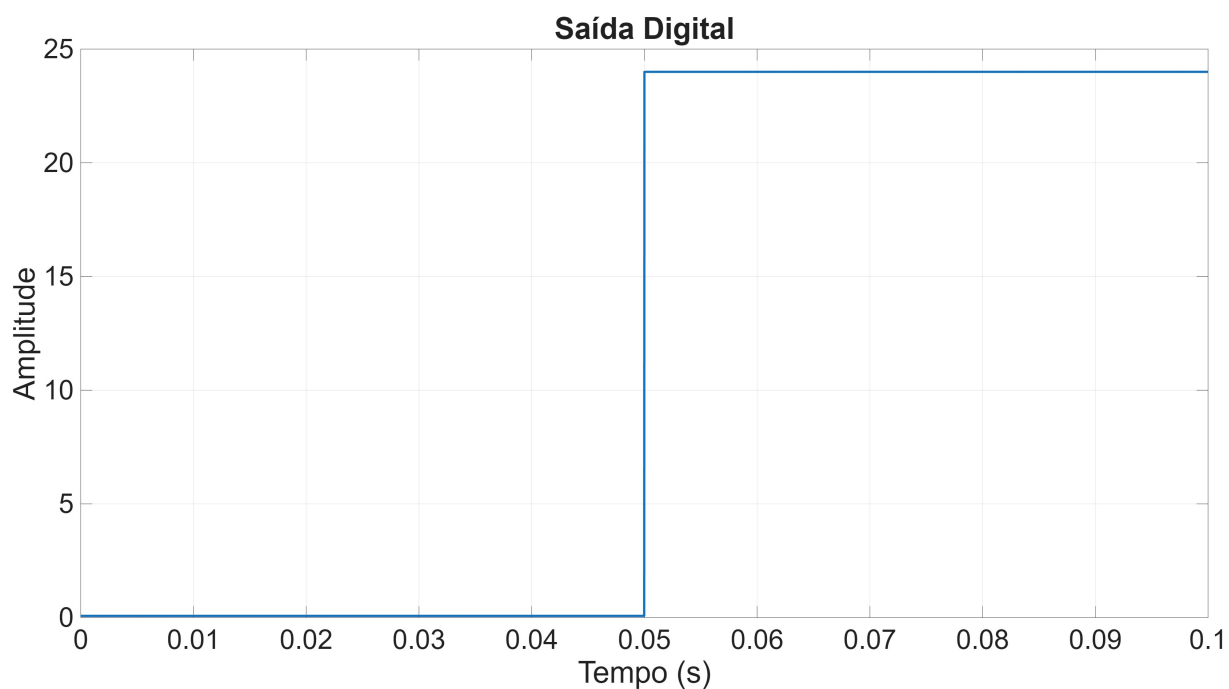
Ambos os circuitos digitais propostos foram testados em simulação de transiente com duração de 100ms. Durante as simulações, foram analisados os estados de chave aberta e fechada, que equivalem aos estados desativado e ativado do LED do optoacoplador, com a comutação da chave ocorrendo aos 50ms. Simulou-se também o ponto de operação de ambos os circuitos em ambos os estados. Os resultados podem ser vistos nas Figuras 17, 18 e 19.

Figura 17 – Resultado das simulações da entrada digital isolada.



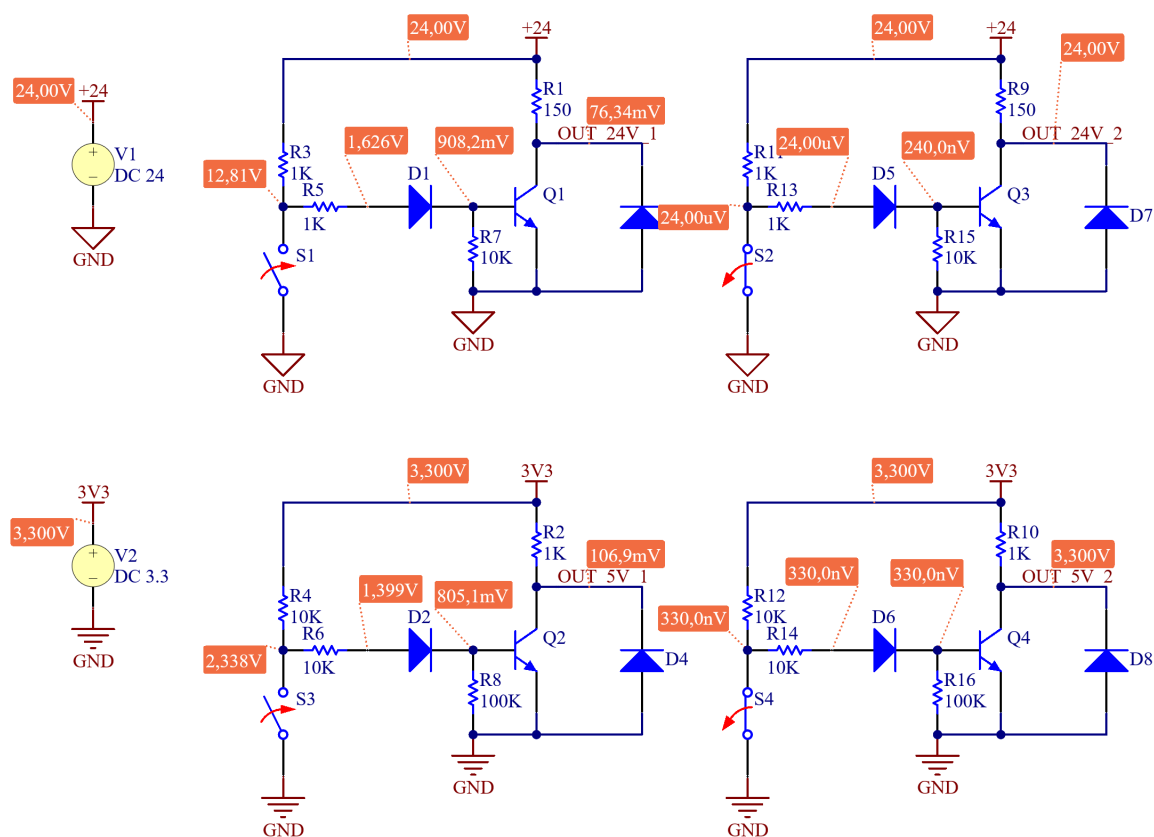
Fonte: Elaborado pelo autor

Figura 18 – Resultado das simulações da saída digital isolada.



Fonte: Elaborado pelo autor

Figura 19 – Simulação de ponto de operação para ambos os circuitos em ambos estados.



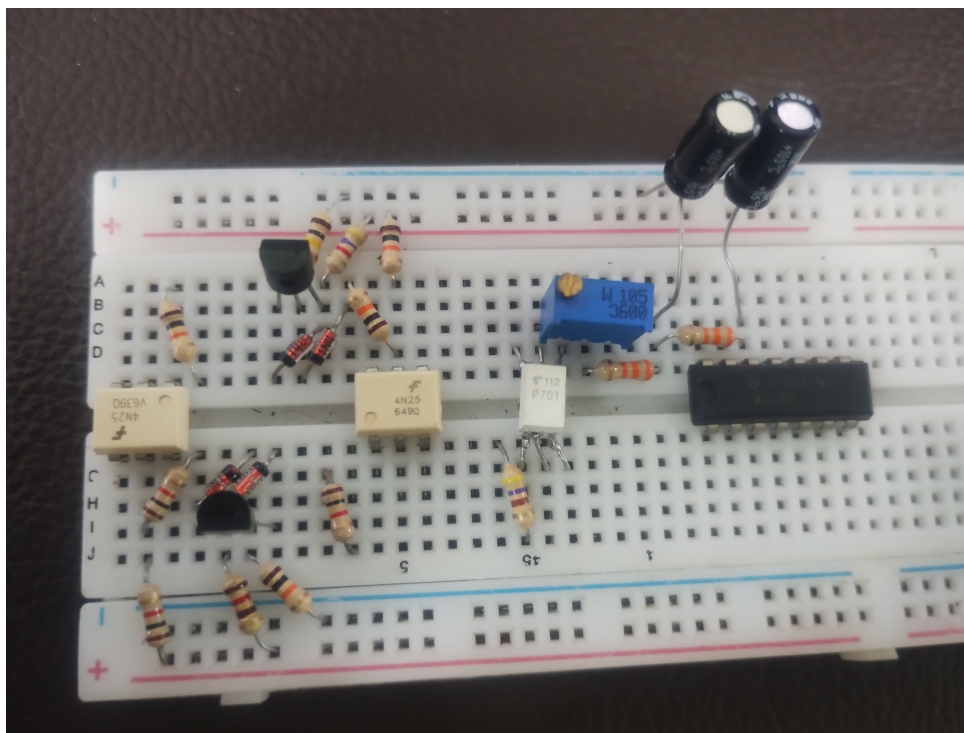
Fonte: Elaborado pelo autor

4.2 Prototipagem

Antes da fabricação da PCB foi realizada a etapa de prototipagem dos circuitos de entrada, saída digital e saída analógica, com o objetivo de validar o funcionamento dos circuitos projetados e comparar os resultados experimentais com as simulações.

O circuito foi montado em uma *protoboard* conforme visto na Figura 20, utilizando os mesmos valores de resistores, capacitores e optoacopladores definidos na etapa de simulação. Nesta montagem, as interligações entre os componentes foram propositalmente deixadas sem *jumpers* em alguns trechos, a fim de facilitar a visualização dos componentes.

Figura 20 – Prototipagem do circuito de interface montado em protoboard.



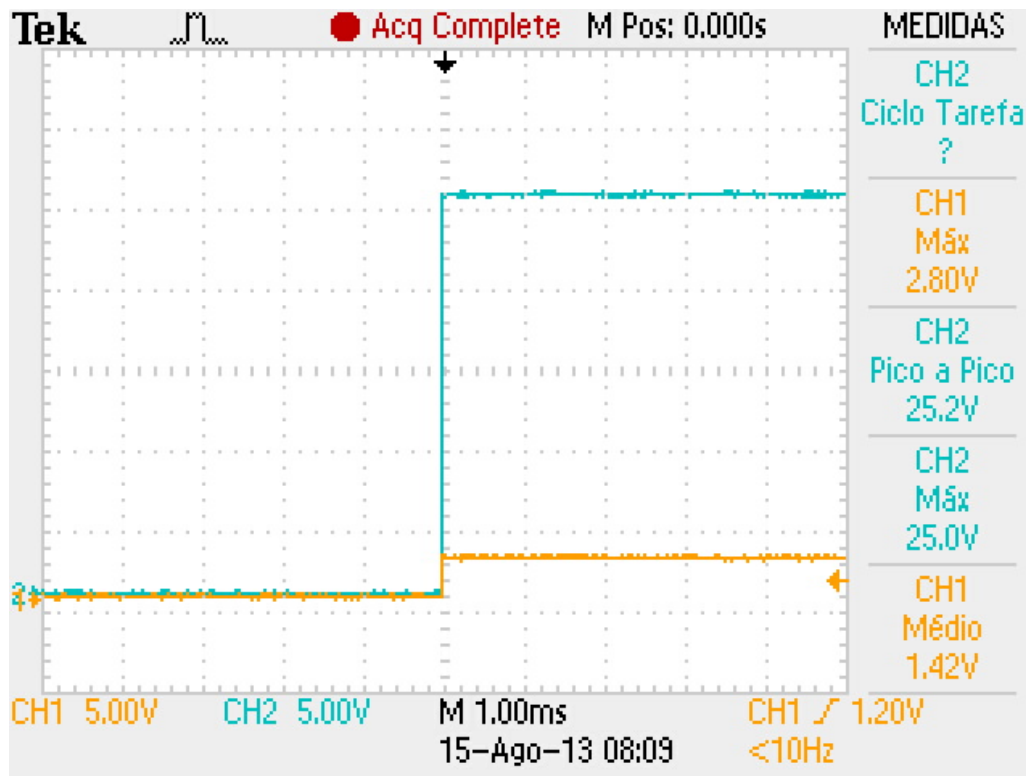
Fonte: Elaborado pelo autor

Foram testados três subsistemas principais: a entrada digital isolada, a saída digital isolada e o canal analógico. Em todos os casos, as medições foram realizadas com o auxílio de um osciloscópio registrando os sinais nos pontos de entrada e saída de cada estágio.

4.2.0.1 Entrada Digital Isolada

A Figura 21 apresenta o resultado da medição do circuito de entrada digital isolada. Observa-se a forma de onda correspondente à comutação entre os níveis de 0 V e 3,3 V no domínio lógico, conforme o acionamento da entrada de 24 V.

Figura 21 – Sinal observado na entrada digital isolada.



Fonte: Elaborado pelo autor

4.2.0.2 Saída Digital Isolada

O comportamento da saída digital isolada pode ser visto na Figura 22. O sinal de controle da *ESP32-S3* é refletido na saída de 24 V com transições rápidas e níveis bem definidos.

4.2.0.3 Saída Analógica Filtrada

Para o canal analógico, os testes foram conduzidos com sinais PWM de diferentes ciclos de trabalho, representando distintos valores analógicos. As Figuras 23a, 23b e 23c mostram as respostas obtidas para *duty cycles* de 30%, 50% e 100%, respectivamente.

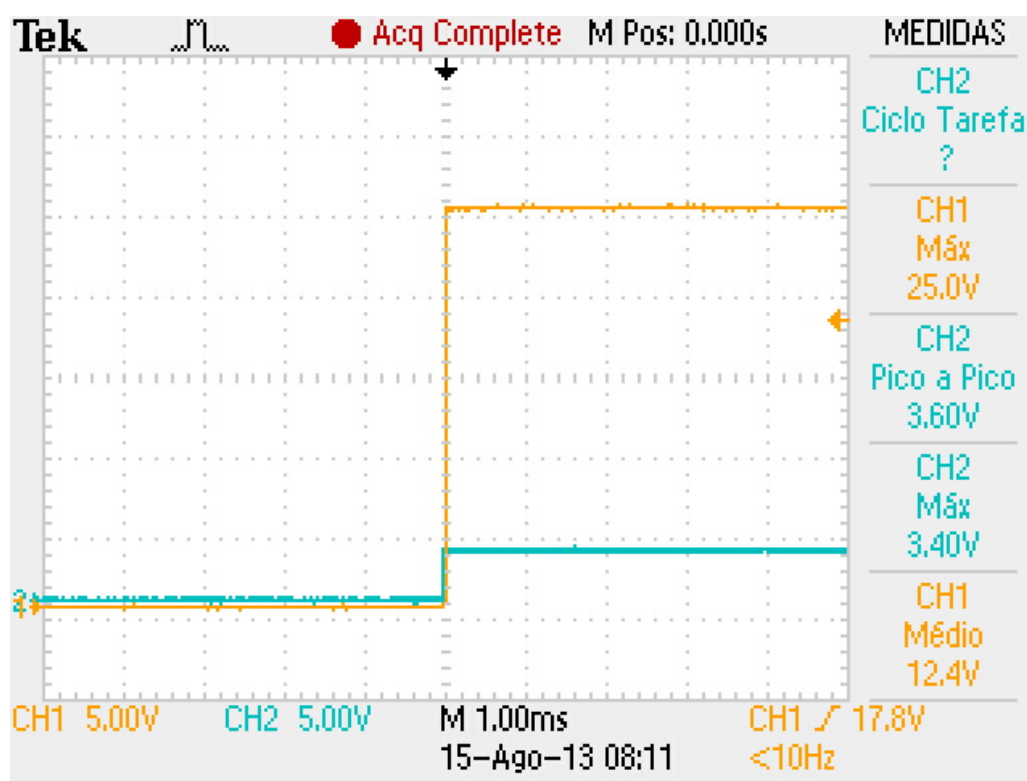
As medições indicaram um comportamento analógico proporcional ao ciclo de trabalho aplicado e ajustado dentro dos limites de *input* do CLP. O *ripple* na saída manteve-se inferior a 200 mV_{pp} , resultado suficiente para a aplicação desejada.

Os resultados obtidos em *proto board* confirmam a viabilidade do projeto eletrônico e a fidelidade entre as simulações e os testes práticos, validando a topologia adotada antes da produção da PCB.

4.3 Fabricação

Após a finalização do layout, os arquivos *Gerber* e a furação *NC drill* foram exportados diretamente do *Altium Designer* para envio ao serviço de manufatura. O procedimento de

Figura 22 – Forma de onda obtida na saída digital isolada.



Fonte: Elaborado pelo autor

submissão consistiu no carregamento dos arquivos no portal do fabricante, seguido pela seleção dos parâmetros de produção, tais como número de camadas, espessura do cobre, cor da máscara de solda e acabamento superficial.

Após a aprovação da cotação, a fabricação foi iniciada e as placas foram entregues conforme solicitado. A Figura 24 apresenta a PCB recém-fabricada, sem a montagem dos componentes, permitindo observar o roteamento, serigrafia, máscara de solda e trilhas expostas para inspeção visual preliminar.

Com a PCB fabricada, iniciou-se o processo de montagem dos componentes. Os dispositivos SMD, com exceção dos LEDs, foram soldados utilizando um forno de refusão, garantindo maior uniformidade na deposição de solda e reduzindo o risco de falhas. Os LEDs SMD e todos os componentes *through hole* foram posteriormente montados de forma manual, seguindo as boas práticas de soldagem.

O processo incluiu inspeção visual criteriosa, testes de continuidade em trilhas críticas e validação da integridade do isolamento entre os domínios de 24 V e 3,3/5 V, assegurando que não houvesse acoplamentos ou fugas indesejadas que pudessem comprometer a operação do sistema.

As Figuras 25a e 25b apresentam diferentes vistas da placa já montada. É importante destacar que tanto o módulo *ESP32-S3* quanto o display OLED não aparecem nas imagens, pois são conectados apenas na etapa final da montagem do sistema. Ambos utilizam

encaixes modulares na própria PCB, facilitando futuras manutenções e atualizações.

4.4 Montagem Final da Planta Didática

Após a conclusão do projeto eletrônico e mecânico, procedeu-se à montagem final da planta didática sobre uma base de madeira quadrada com 85 cm de lado. Essa base serviu como plataforma estrutural para a fixação das duas esteiras transportadoras, do sistema pneumático e do conjunto de controle.

As duas esteiras foram instaladas em um posicionamento para facilitar a transferência das peças classificadas. Para assegurar uma transição suave entre as esteiras, foi projetada uma rampa de ligação. Esse componente, produzido em impressora 3D, foi instalado imediatamente à frente dos pistões pneumáticos da primeira e segunda esteira. Sua geometria foi desenvolvida para guiar as peças ao serem impulsionadas pelo pistão, garantindo alinhamento adequado durante a transferência. Além disso os sensores capacitivos e indutivos foram instalados com uma altura maior em relação a esteira, permitindo a classificação por tamanho. As Figuras 26a e 26b apresentam a montagem completa.

A montagem final proporcionou uma célula didática modular, com clara separação entre subsistemas mecânicos, pneumáticos e eletrônicos.

4.5 Validação da Montagem e Operação da Planta

Com todos os módulos eletrônicos, mecânicos e pneumáticos integrados, realizou-se a validação completa da planta didática por meio de um roteiro operacional descrito em mais detalhes adiante. Essa validação teve como objetivo confirmar a operação coordenada entre o sistema embarcado de visão computacional, os sensores industriais, o CLP, os atuadores pneumáticos e ambas as esteiras transportadoras.

A planta foi configurada para operar com três destinos finais possíveis para cada peça processada:

1. **Saída 1 — Final da Esteira 1:** destino quando a peça não é desviada para a segunda esteira.
2. **Saída 2 — Ejeção na Esteira 2 (pistão de ejeção):** pistão localizado imediatamente antes do final da esteira 2.
3. **Saída 3 — Final da Esteira 2:** destino final quando a peça trafega pela segunda esteira sem sofrer ejeção pelo pistão dessa esteira.

O ciclo operacional adotado na validação segue os passos abaixo:

1. **Detecção inicial:** o processo inicia-se com a detecção de um objeto pelo sensor óptico instalado na primeira esteira, provocando a ativação do sequenciamento de transporte e inspeção.
2. **Transporte na Esteira 1:** a primeira esteira é ligada, conduzindo a peça ao longo do percurso. A peça é monitorada pelos sensores capacitivo e indutivo da esteira 1, que fornecem confirmação de presença, classificação de material plástico ou metálico e auxiliam no sincronismo do pistão de desvio.
3. **Posicionamento em frente ao pistão de desvio (pistão 1):** ao aproximar-se da região de desvio, a peça é posicionada em frente ao pistão 1, aguardando a decisão de classificação proveniente do sistema de visão embarcada.
4. **Decisão de roteamento:**
 - **Destino Saída 1:** se a classificação indicar que a peça deve permanecer na esteira 1, o pistão 1 permanece inativo e a esteira continua o transporte até o final (Saída 1).
 - **Destino Saída 2 ou Saída 3:** se a classificação indicar que a peça deve ser enviada à segunda esteira, o pistão 1 é acionado para desviar a peça para a esteira 2.
5. **Transporte e tratamento na Esteira 2:** após o desvio, a segunda esteira é colocada em movimento e conduz a peça ao longo de seu percurso. Os sensores capacitivo e indutivo da esteira 2 realizam a classificação em função da altura da peça, além das mesmas funções que seus equivalentes na esteira 1.
6. **Ejeção ou finalização na Esteira 2:** quando a peça alcança a região imediatamente anterior ao pistão de ejeção montado na esteira 2, é executada a ação correspondente à classificação:
 - **Saída 2 (ejeção):** se a classificação indicar ejeção, o pistão de ejeção na esteira 2 é acionado, removendo a peça para o destino correspondente à Saída 2.
 - **Saída 3 (final da esteira 2):** caso contrário, a peça segue até o final da esteira 2 e é coletada na Saída 3.

O bloco funcional criado no software *Tia Portal*, ilustrado na Figura 27, foi projetado para interpretar corretamente os dados enviados pelo sensor inteligente embarcado tanto no modo contínuo de operação quanto no modo de input. No modo contínuo, o sensor transmite sucessivas classificações via TCP a cada ciclo de captura, o que motivou a implementação de um mecanismo de análise por frequência dentro do bloco. Nesse modo, cada detecção válida é armazenada em um histórico circular, permitindo calcular a classe mais recorrente

e sua confiança estatística a partir das últimas amostras recebidas, filtrando eventuais classificações incorretas. Já para o modo sob demanda o bloco trabalha interpretando e disponibilizando diretamente a última detecção recebida, sem realizar agregação estatística ou histórico prolongado. Essa arquitetura híbrida permite que o bloco atue como uma camada de interpretação confiável entre os dados brutos recebidos pelo bloco *TRCV_C* e a lógica de controle da planta, adaptando-se automaticamente ao tipo de operação do sensor e garantindo respostas coerentes e sincronizadas com o ciclo de controle. O código do bloco se encontra no Apêndice D.

Durante a validação experimental, o roteiro foi executado repetidamente, cobrindo diversos cenários de classificação e roteamento. Os resultados observados foram:

Os testes de validação concentraram-se principalmente na atuação do sensor inteligente embarcado, uma vez que os demais elementos da planta (sensores industriais, esteiras e atuadores) já são dispositivos consolidados no ambiente industrial e possuem comportamento amplamente documentado. Assim, o objetivo desta etapa foi verificar a capacidade do sistema de visão baseado na *ESP32* em operar de forma confiável, estável e sincronizada com o CLP e com o fluxo físico das peças.

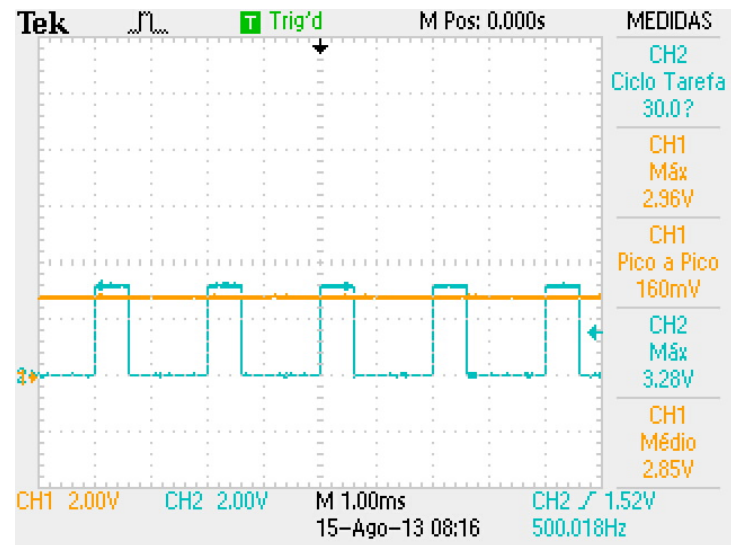
Os principais resultados observados foram:

- O algoritmo de inferência executado na *ESP32* apresentou classificação consistente das peças, com uma confiança próxima a 80% para todas as peças.
- As três modalidades de saída do sensor inteligente mostraram-se coerentes entre si e sincronizadas com o ciclo do processo. Para a saída TCP foi utilizado o bloco de recebimento *TRCV_C* e um bloco para interpretar os dados recebidos detalhado mais a frente e visto na Figura 27
- O tempo total de processamento da câmera, inferência e disponibilização das saídas manteve-se dentro da janela operacional necessária para permitir que a peça fosse posicionada corretamente em frente ao pistão de desvio.
- A integração lógica entre o sensor inteligente e o CLP mostrou robustez: a decisão enviada pela *ESP32-S3* foi corretamente interpretada pelo CLP.

Esses resultados demonstram que o sensor inteligente proposto, integrando visão computacional embarcada, interpretação de classes, comunicação industrial e comandos síncronos ao processo, foi capaz de operar como elemento central da lógica da planta, coordenando o roteamento físico das peças com precisão e confiabilidade.

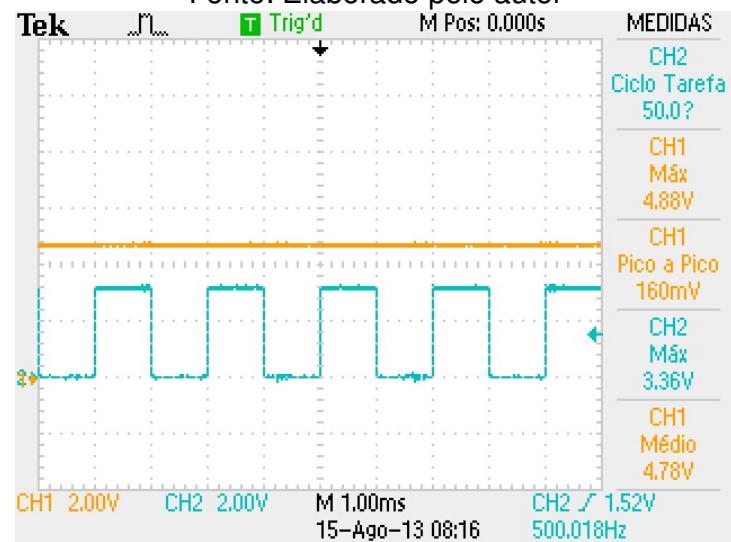
A planta montada, portanto, atende aos objetivos previstos e opera de maneira confiável conforme o roteiro experimental. Os testes validaram a integração entre visão embarcada, lógica de decisão, acionamento pneumático e transporte mecânico, confirmando a viabilidade do sistema como célula de inspeção e separação automatizada.

Figura 23 – Sinais PWM filtrados para diferentes ciclos ativos.



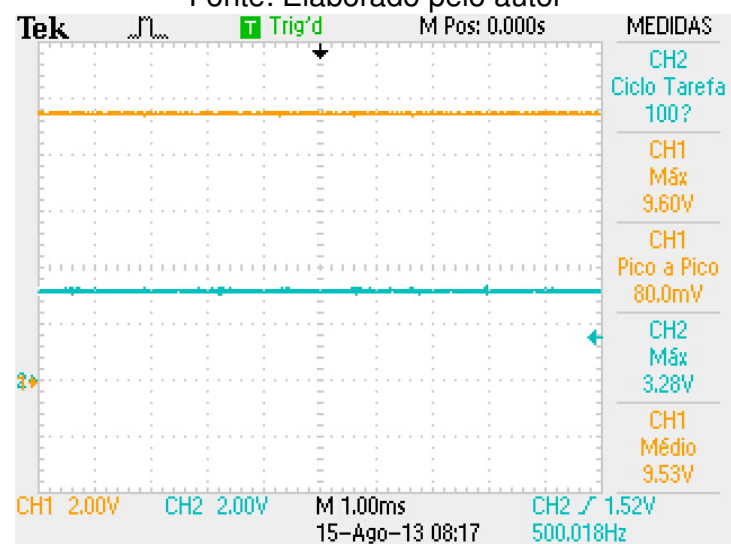
(a) Sinal PWM filtrado para *duty cycle* de 30%.

Fonte: Elaborado pelo autor



(b) Sinal PWM filtrado para *duty cycle* de 50%.

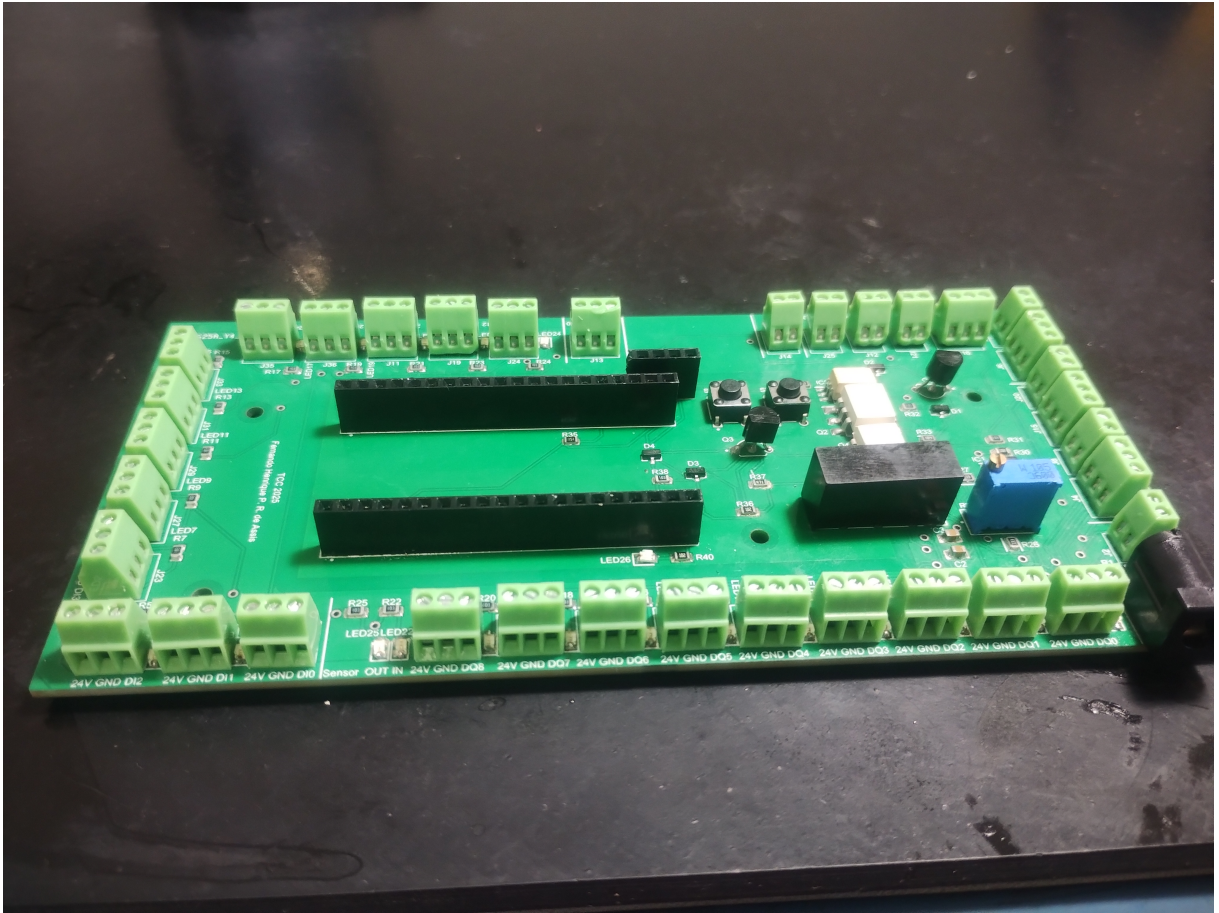
Fonte: Elaborado pelo autor



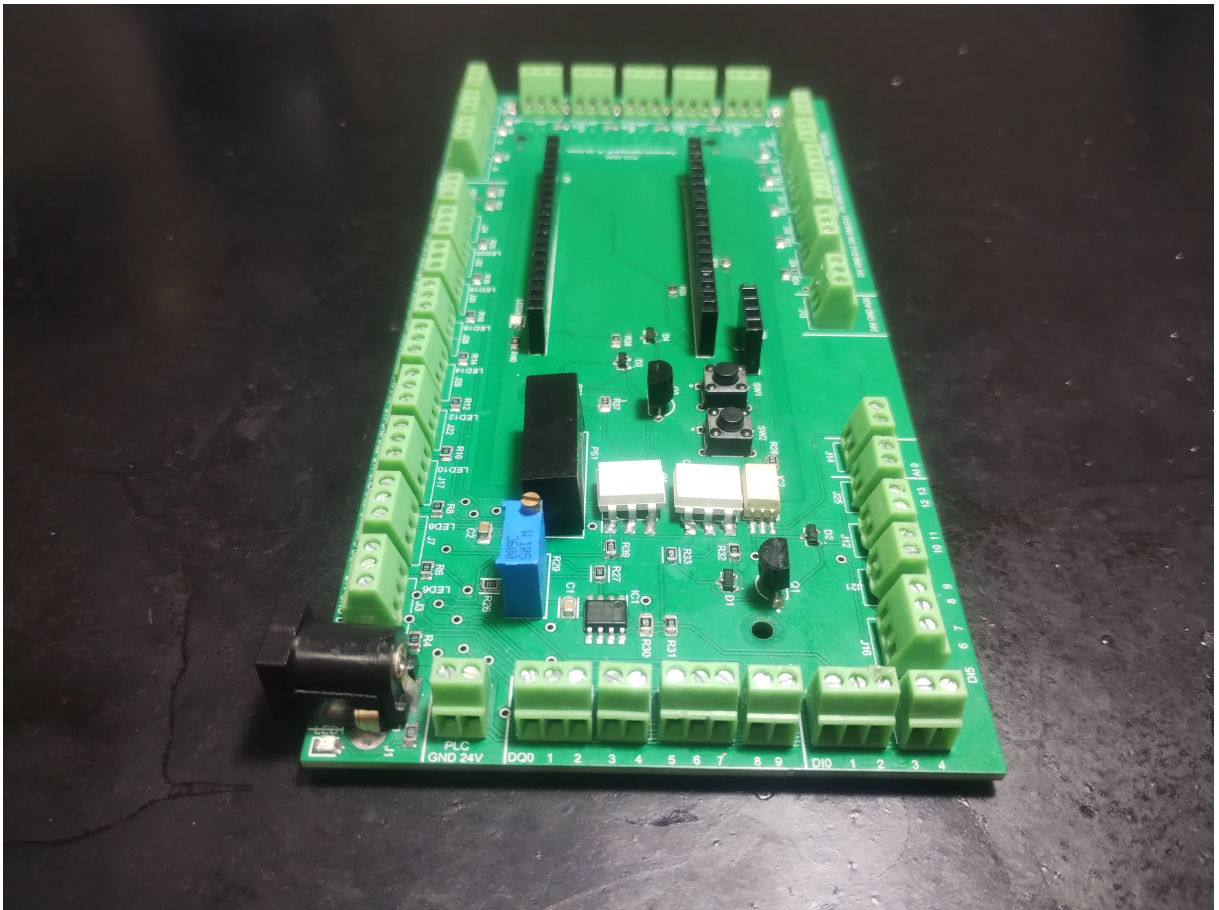
(c) Sinal PWM filtrado para *duty cycle* de 100%.

Fonte: Elaborado pelo autor

Figura 25 – Placa finalizada.

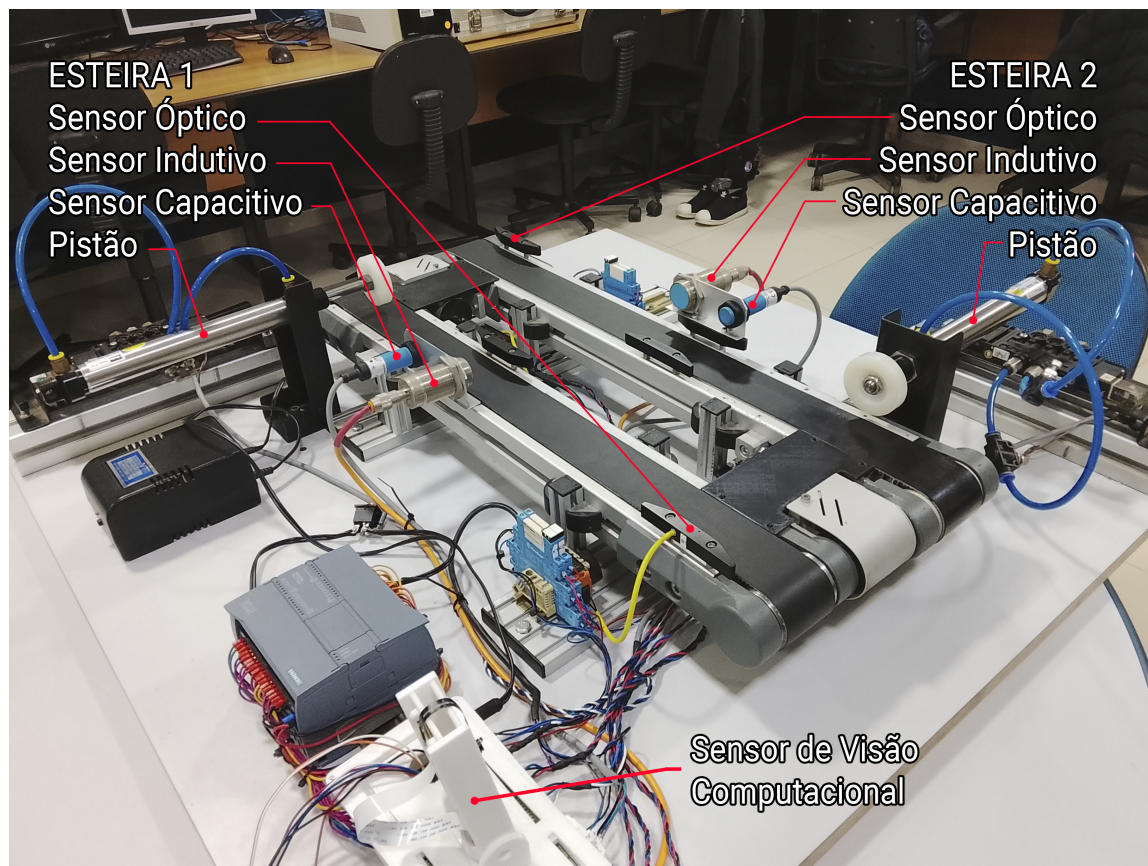


(a) Vista frontal

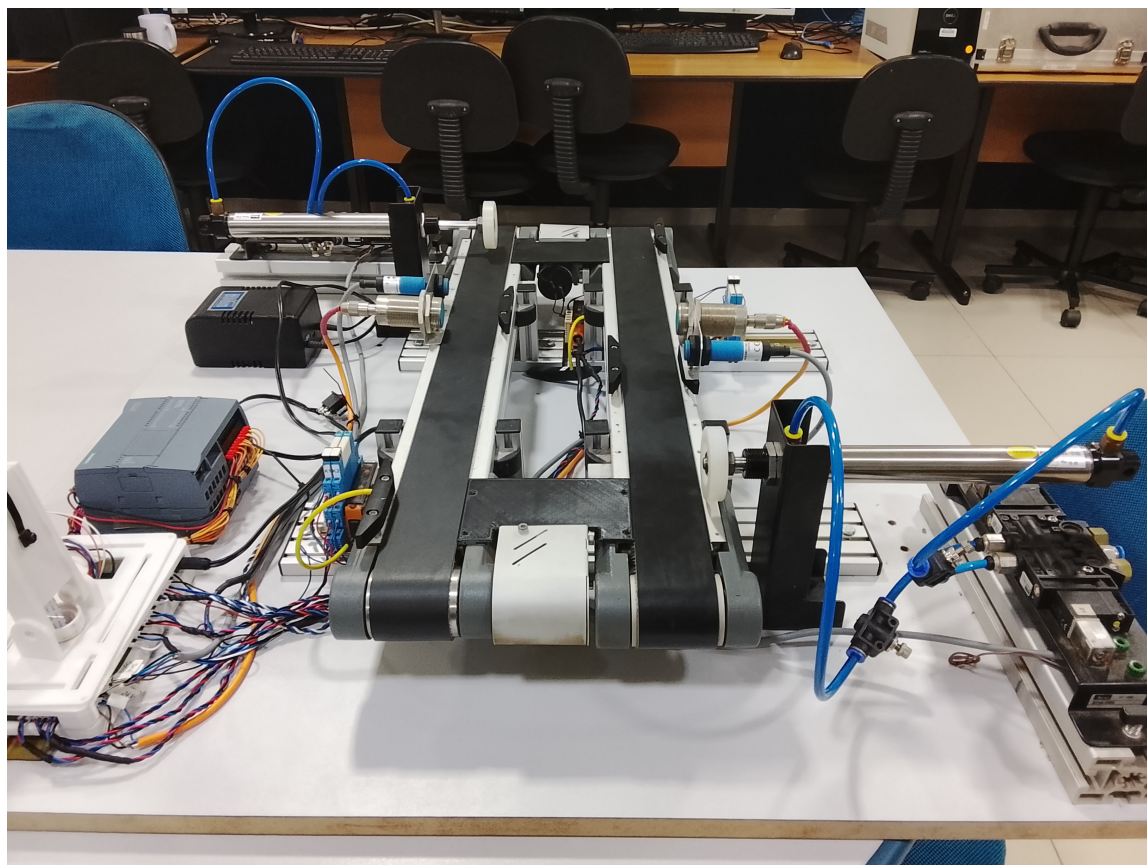


(b) Vista lateral

Figura 26 – Montagem final da planta didática sobre base rígida.

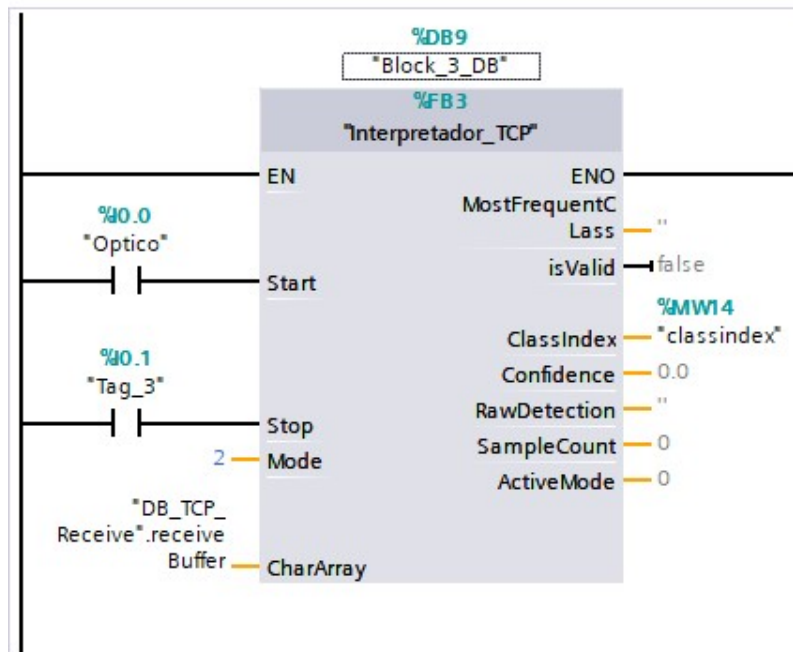


(a) Vista geral da planta e posicionamento dos componentes.



(b) Vista lateral.

Figura 27 – Bloco e para interpretação do sinal recebido.



Fonte: Elaborado pelo autor

5 CONCLUSÃO

A solução proposta demonstrou que dispositivos de baixo custo, como a *ESP32-S3*, podem executar modelos de inteligência artificial em tempo hábil com desempenho satisfatório, alinhados às tendências observadas em estudos recentes de computação embarcada e *edge AI*. A integração segura com sistemas industriais legados foi garantida por meio de um circuito dedicado de interface, composto por canais isolados para entrada e saída digital e uma saída analógica filtrada, conforme práticas recomendadas para ambientes industriais. A utilização de optoacopladores e filtros analógicos aliados ao isolamento galvânico permitiu a conexão confiável ao CLP e aos atuadores de 24 V.

O processo de desenvolvimento do hardware, desde a simulação dos circuitos, prototipagem em bancada, elaboração do layout no *Altium Designer*, envio dos arquivos para fabricação e posterior montagem, resultou em uma plataforma compacta, modular e reproduzível. Os testes realizados validaram a operação correta de todos os canais de comunicação e confirmaram o funcionamento integrado entre o sistema embarcado, os sensores industriais e a lógica de automação da planta.

Além da implementação técnica, o trabalho reforça a relevância das tecnologias habilitadoras da Indústria 4.0 no ambiente educacional, oferecendo uma ferramenta prática para ensino de automação, sistemas ciberfísicos e inteligência artificial aplicada. A planta didática desenvolvida contribui para reduzir o distanciamento tecnológico entre os alunos e as demandas industriais contemporâneas, especialmente no cenário brasileiro, onde a adoção da I4.0 ainda avança de forma gradual.

Como perspectivas de continuidade, destacam-se: (i) aprimoramento do modelo de visão computacional, (ii) integração de comunicação industrial padronizada, como MQTT ou OPC-UA, ampliando a interoperabilidade, e (iii) desenvolvimento de um módulo didático complementar para monitoramento remoto. A expansão dessas funcionalidades reforçará a aplicabilidade da plataforma em ambientes educacionais, alinhando-se às tendências emergentes da manufatura inteligente.

Dessa forma, conclui-se que os objetivos estabelecidos foram plenamente alcançados. O projeto demonstrou a viabilidade técnica de desenvolver uma planta didática de baixo custo com sensor inteligente para classificação de materiais em tempo real, integrando-se de maneira segura e eficiente a sistemas industriais legados. Além disso, o trabalho contribuiu diretamente para a modernização do laboratório de automação, aproximando o ambiente acadêmico das tecnologias emergentes da Indústria 4.0. A arquitetura modular proposta também cumpre o objetivo de possibilitar futuras expansões, favorecendo o desenvolvimento de novas aplicações, iniciações científicas e trabalhos de conclusão de curso.

REFERÊNCIAS

- ALIEXPRESS. *Módulo ESP32-S3 com câmera — imagem do produto*. 2025. Acesso em: 09 dez. 2025. Disponível em: <https://pt.aliexpress.com/item/1005006485517423.html>.
- CAO, K. et al. An overview on edge computing research. *IEEE Access*, v. 8, p. 85714–85728, 2020.
- CircuitsPedia. *Optocoupler / Opto-Isolator – Working and Uses*. 2025. <https://circuitspedia.com/optocoupler-opto-coupler-opto-isolator-working-and-uses/>. Acesso em 09 de dezembro de 2025.
- DERA, P.; TALAŚKA, T.; DŁUGOSZ, R. A new, cost-efficient modular sensor platform for IoT and predictive maintenance in industrial applications. *Journal of Computational and Applied Mathematics*, Elsevier B.V., v. 473, p. 116777, 2026.
- Embarcados. *Computação na Borda (Edge Computing): o futuro do processamento de dados distribuídos*. 2025. <https://embarcados.com.br/computacao-na-borda-edge-computing-o-futuro-do-processamento-de-dados-distribuidos/>. Acesso em 09 de dezembro de 2025.
- FOLGADO, F. J. et al. Review of industry 4.0 from the perspective of automation and supervision systems: Definitions, architectures and recent trends. *Electronics*, v. 13, n. 4, p. 782, 2024.
- GONÇALVES, D. B. INDÚSTRIA 4.0 no BRASIL: ANÁLISE DA CONTRIBUIÇÃO DO CENTRO DE EXCELÊNCIA EM TECNOLOGIA 4.0. *Latin American Journal of Business Management*, v. 15, n. 2, p. 249–267, jul-dec 2024. ISSN 2178-4833.
- GRANDINETTI, J. e. a. Desenvolvimento de hardware baseado em esp32 para aplicações em automação industrial e iot. *Anais do CBA*, 2022.
- International Society of Automation. *ISA-95 Automation Pyramid Diagram*. 2025. <https://www.isa.org/standards-and-publications/isa-standards/isa-95-standard>. Imagem acessada em 09 de dezembro de 2025.
- JAVOID, M. et al. Enabling flexible manufacturing system (FMS) through the applications of industry 4.0 technologies. *Internet of Things and Cyber-Physical Systems*, Elsevier B.V. on behalf of KeAi Communications Co., Ltd., v. 2, p. 49–62, 2022.
- KAHVECI, S. et al. An end-to-end big data analytics platform for IoT-enabled smart factories: A case study of battery module assembly system for electric vehicles. *Journal of Manufacturing Systems*, Elsevier Ltd on behalf of The Society of Manufacturing Engineers, v. 63, p. 214–223, 2022.
- LEE, J. et al. Industrial AI and predictive analytics for smart manufacturing systems. In: *Smart Manufacturing*. [S.l.]: Elsevier Inc., 2020. cap. 8, p. 213–243.
- LELLI, F. Interoperability of smart manufacturing systems. *ACM Computing Surveys*, 2019.
- LINDNER, M. et al. Digital twins in manufacturing: A rami 4.0 compliant concept. *Sci*, v. 5, n. 4, p. 40, 2023. Disponível em: <https://doi.org/10.3390/sci5040040>.

MATIAS, R. e. a. Ambiente de programação para ensino de iot com esp32. *Revista Ensino de Engenharia*, 2022.

PAJPACH, T. e. a. Low-cost educational kit with ai and computer vision for industry 4.0 training. *Procedia Computer Science*, v. 200, p. 541–550, 2022.

PETROSINO, L. et al. Image sensors and VPU acceleration for data analysis and classification. In: *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*. [S.l.]: IEEE, 2021. p. 392–396.

PUGLIESE, L. e. a. Desenvolvimento de uma planta didática de baixo custo baseada em arduino para controle de processos. *Encontro Nacional de Modelagem Computacional*, 2022.

SANTOS, A. e. a. Estudo de uma planta didática com acesso remoto para controle de processos via internet. *Anais do CONEMI*, 2024.

SEDRA, A. S.; SMITH, K. C. *Microelectronic Circuits*. 7th. ed. New York: Oxford University Press, 2014. ISBN 978-0-19-933913-6.

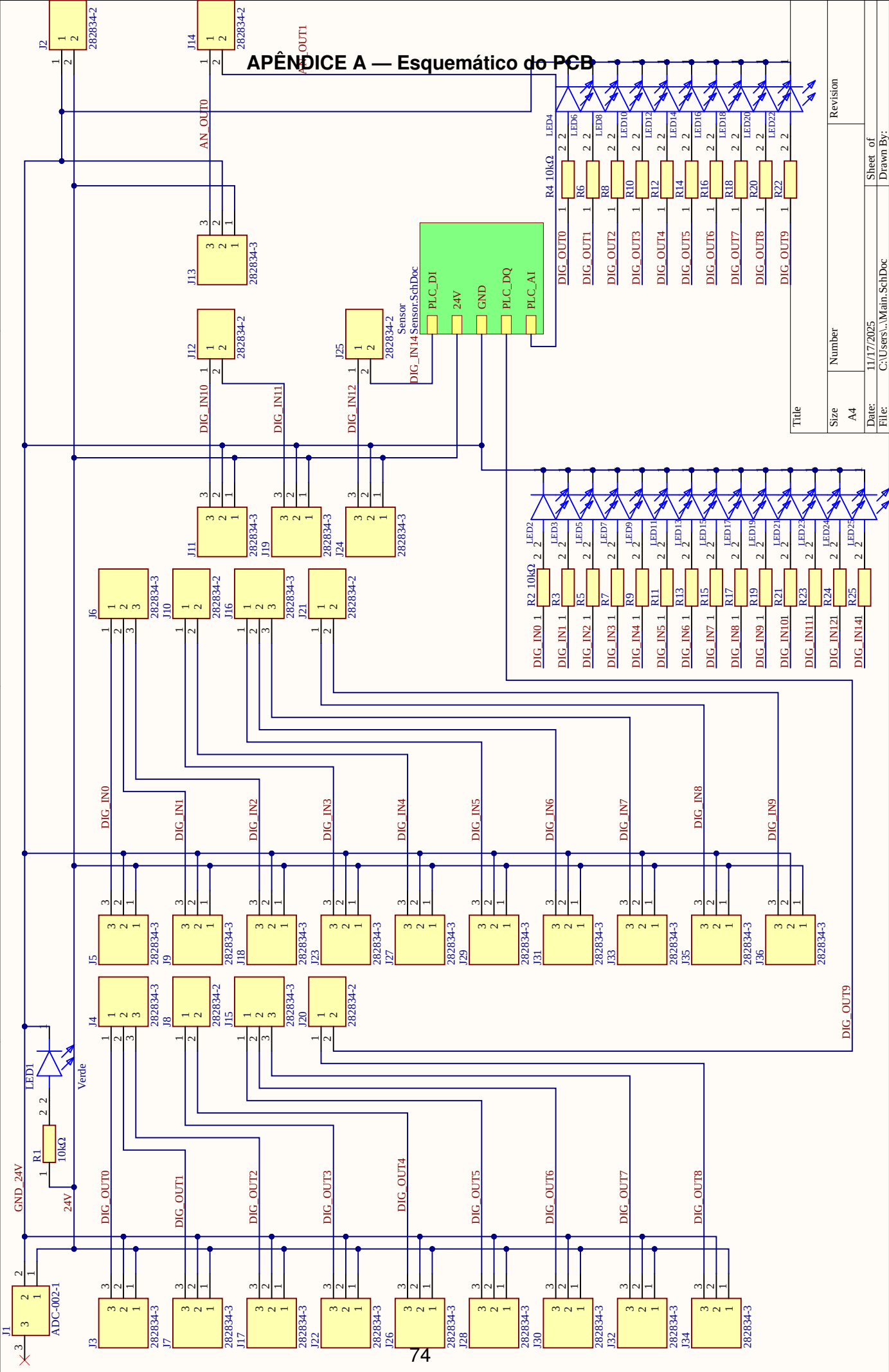
SHARMA, D.; TOMAR, G. S.; KUSHWAH, S. S. Embracing industry 4.0: A comprehensive analysis. In: *2024 16th IEEE International Conference on Computational Intelligence and Communication Networks (CICN)*. [S.l.]: IEEE, 2024. p. 1075–1079.

Tecnicon. *Imagem sobre os pilares da Indústria 4.0*. 2025. https://www.tecnicon.com.br/blog/476-4_exemplos_praticos_da_adocao_da_industria_4_0_nas_fabricas. Imagem acessada em 09 de dezembro de 2025.

APÊNDICES

APÊNDICE A – Esquemático do PCB

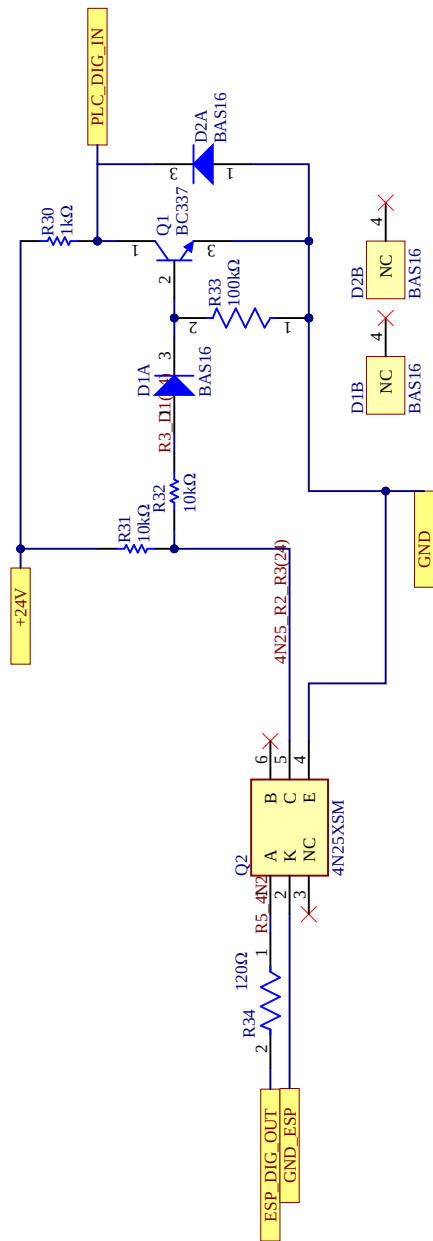
APÊNDICE A — Esquemático do PCB



Title		Revision	
Size	Number		
A4			
Date:	11/17/2025	Sheet of	
File:	C:\Users\Main\SchDoc	Drawn By:	

4

APÊNDICE A — Esquemático do PCB



Title

Revision

Number

Size

A4

Sheet of

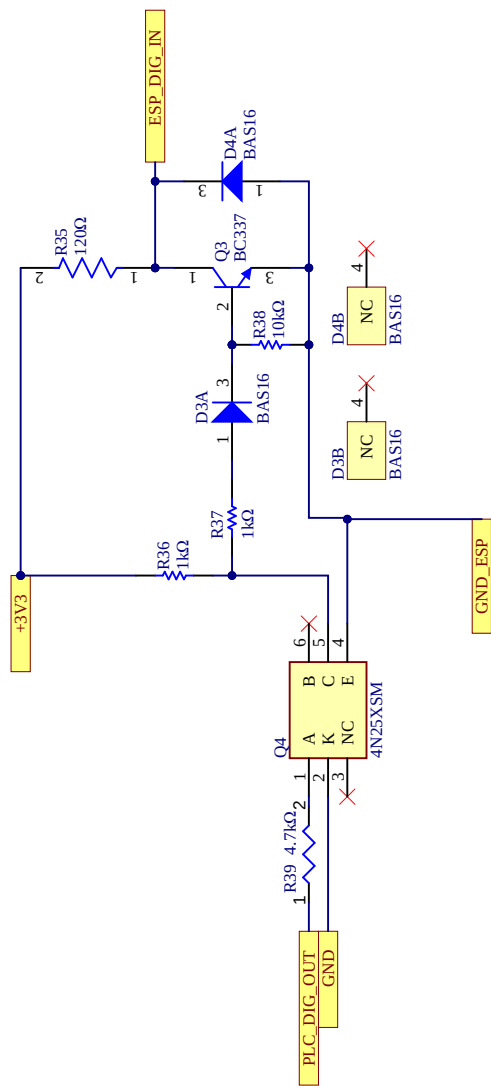
Date:

11/17/2025

Drawn By:

C:\Users\LDIG_ESP-PLC\SchDoc

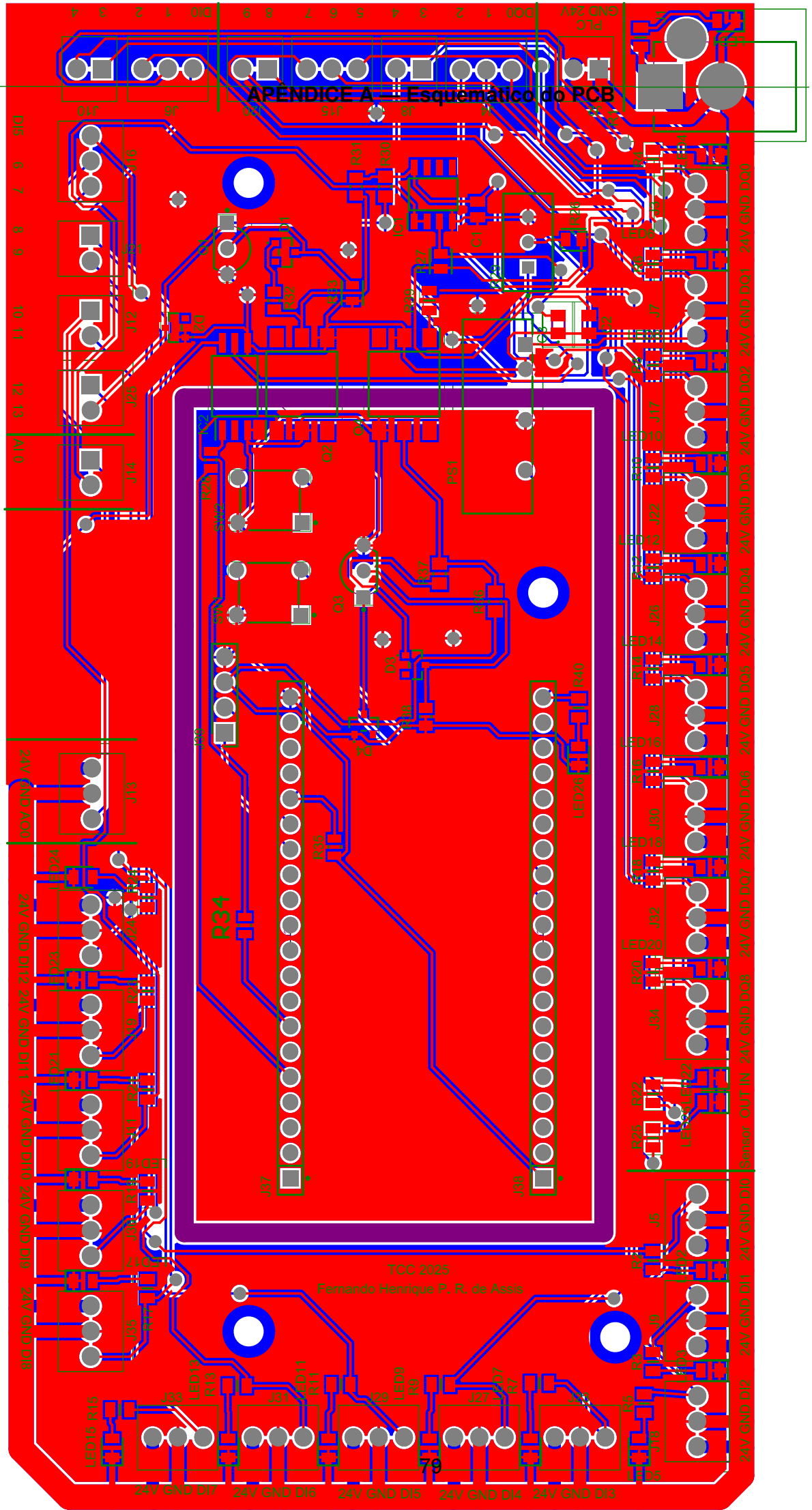
APÊNDICE A — Esquemático do PCB



Title		Revision	
Size	Number		
A4			
Date:	11/17/2025	Sheet of	
File:	C:\Users\LDIG\PLC-ESP\SchDoc	Drawn By:	

Title			
Size	Number	Revision	
A4			
Date:	11/17/2025	Sheet of	
File:	C:\Users\LANALOG	ESP-PLC_SchDoc	Drawn By:

APÊNDICE A — Esquemático do PCB



APÊNDICE B – Lista de Materiais

APÊNDICE B — Lista de Materiais

Comentário	Descrição	Designador	Footprint	Código Mouser	Quantidade
0.1uF	0805 0.1uF	C1	CAPC2012X140N	08051C104KAT2A	1
1uF	0805 1uF	C2, C3	C0805	C0805C105K5RACTU	2
BAS16	High-speed Switching Diode	D1, D2, D3, D4	NXP-SOT23-3_V	CMP-0648-00001-1	4
LM358DT	Integrated Circuit	IC1	SOIC127P600X175-8N	LM358DT	1
TLP701_TP/F_	Integrated Circuit	IC2	SOIC127P970X425-6N	TLP701_TP/F_	1
ADC-002-1	Connector	J1	ADC0021	ADC-002-1	1
282834-2	Connector	J2, J8, J10, J12, J14, J20, J21, J25	282834-2_1	282834-2	8
282834-3	Connector	J3, J4, J5, J6, J7, J9, J11, J13, J15, J16, J17, J18, J19, J22, J23, J24, J26, J27, J28, J29, J30, J31, J32, J33, J34, J35, J36	2828343	282834-3	27
SSW-120-01-G-S	SSW-120-01-G-S SSW - Sam	J37, J38	FP-SSW-120-01-G-S-MFG	CMP-00575-00296075-1	2
68001-204HLF	Connector	J39	HDRV4W67P0X254_1X4_1016X241X858P	68001-204HLF	1
LED Verde	LED Verde 0805	LED1, LED26	LEDC2012X80N	HSMG-C170	2
LED Vermelho	LED Vermelho 0805	LED2, LED3, LED4, LED5, LED6, LED7, LED8, LED9, LED10, LED11, LED12, LED13, LED14, LED15, LED16, LED17, LED18, LED19, LED20, LED21, LED22, LED23, LED24, LED25	LEDC2012X80N	HSMH-H170	24
HLK-B2405S-2WR3	DC/DC 24V-5V	PS1	HLKB2405S2WR3	HLK-B2405S-2WR3	1
BC337	Amplifier Transistor NPN	Q1, Q3	ONSC-TO-92-3-29-11_AM_BL	CMP-1048-01576-1	2
4N25XSM	Phototransistor	Q2, Q4	SOT254P1016X460-6N	4N25XSM	2
10kΩ	0805 10kΩ	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25	RESC2012X60N	CRCW080510K0FKEAC	25
33kΩ	0805 33kΩ	R26, R27	FP-CRGCQ0805-IPC_C	CMP-032111-001283-1	2
470Ω	0805 470Ω	R28	RESC2013X50X301L20T20	CMP-2001-03932-1	1
1MΩ	TRIMMER 1MΩ	R29	FP-3296W-MFG	CMP-07234-000060-1	1
1kΩ	0805 1kΩ	R30, R36, R37	RESC2013X60X35ML10T20	CMP-2100-03671-1	3
10kΩ	0805 10kΩ	R31, R32, R38	RESC2013X60X35NL10T20	CMP-1659-00013-4	3
100kΩ	0805 100kΩ	R33	FP-RC0805-0_6-IPC_C	CMP-2100-03668-5	1
120Ω	0805 120Ω	R34, R35	FP-CRCW0805-e3-IPC_C	CMP-02407-007854-1	2
4.7kΩ	0805 4.7kΩ	R39	RESC2012X50N	CRCW08054K70FKEA	1
1kΩ	0805 1kΩ	R40	RESC2012X60N	CRCW08051K00FKEAHP	1
1825910-6	Tact Switch	SW1, SW2	TECO-1825910-6_V	CMP-1684-00021-1	2

APÊNDICE C – Código-Fonte do Firmware do Sistema

Listing 5.1 – Código-fonte completo do firmware do sistema ESP32

```

1  /* Edge Impulse Arduino examples
2   * Combined with OLED display functionality and Image Collection
3   * Copyright (c) 2022 EdgeImpulse Inc.
4   */
5
6  // These sketches are tested with 2.0.4 ESP32 Arduino Core
7  // https://github.com/espressif/arduino-esp32/releases/tag/2.0.4
8
9  /* Includes
   -----
    */
10 #include <fernandohprausp-project-1_inferencing.h>
11 #include "edge-impulse-sdk/dsp/image/image.hpp"
12 #include "esp_camera.h"
13
14 // OLED and WiFi includes
15 #include <SPI.h>
16 #include <Wire.h>
17 #include <Adafruit_GFX.h>
18 #include <Adafruit_SSD1306.h>
19 #include <WiFiManager.h> // https://github.com/tzapu/WiFiManager
20 #include <WiFi.h>
21 #include <string.h> // For memset function
22
23
24 // Image collection includes
25 #include <eloquent_esp32cam.h>
26 #include <eloquent_esp32cam/extra/esp32/wifi/sta.h>
27 #include <eloquent_esp32cam/viz/image_collection.h>
28
29 // PLC Config
30 #include <Preferences.h>
31
32 Preferences preferences;
33 using eloq::camera;
34 using eloq::wifi;
35 using eloq::viz::collectionServer;
36
37 // TCP/PLC connection parameters

```

```

38 #define PLC_PORT 4500
39 #define MAX_ARRAY_SIZE 100
40 #define PLC_CONNECTION_MODE 2 // 1=Raw TCP, 2=With Length header,
    3=Both methods
41
42 // Definies do Display OLED
43 #define SCREEN_WIDTH 128
44 #define SCREEN_HEIGHT 64
45 #define OLED_RESET -1 // Compartilha o pino de reset do ESP32
46 #define SCREEN_ADDRESS 0x3C
47 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
    OLED_RESET);
48
49 // Pinos dos botes
50 #define SELECT_BTN_PIN 3 // Select
51 #define NAV_BTN_PIN 46 //Nav
52
53 // Camera model pins
54 #define PWDN_GPIO_NUM -1
55 #define RESET_GPIO_NUM -1
56 #define XCLK_GPIO_NUM 15
57 #define SIOD_GPIO_NUM 4
58 #define SIOC_GPIO_NUM 5
59
60 #define Y2_GPIO_NUM 11
61 #define Y3_GPIO_NUM 9
62 #define Y4_GPIO_NUM 8
63 #define Y5_GPIO_NUM 10
64 #define Y6_GPIO_NUM 12
65 #define Y7_GPIO_NUM 18
66 #define Y8_GPIO_NUM 17
67 #define Y9_GPIO_NUM 16
68
69 #define VSYNC_GPIO_NUM 6
70 #define HREF_GPIO_NUM 7
71 #define PCLK_GPIO_NUM 13
72
73 /* Constant defines
    ----- */
74 #define EI_CAMERA_RAW_FRAME_BUFFER_COLS 320
75 #define EI_CAMERA_RAW_FRAME_BUFFER_ROWS 240

```

```

76 #define EI_CAMERA_FRAME_BYTE_SIZE 3
77
78 // WiFi credentials for image collection mode
79 #define WIFI_SSID "SSID"
80 #define WIFI_PASS "PASSWORD"
81 #define HOSTNAME "esp32cam"
82
83 #define DIGITAL_OUTPUT_PIN 40 // Digital output pin for object
    detection
84 #define PWM_OUTPUT_PIN 42 // PWM output pin for class
    indication
85 #define PWM_FREQUENCY 500 // PWM frequency in Hz
86 #define PWM_RESOLUTION 14 // 14-bit PWM resolution
87 #define PWM_CHANNEL 1 // PWM channel to use
88
89
90 #define LOGO_DISPLAY_TIME 3000
91
92
93
94 // Normal Mode submenu variables
95 bool normalModeSubmenu = false;
96 int normalModeMenuIndex = 0;
97 const int NORMAL_MODE_MENU_ITEMS = 2;
98 const String NORMAL_MODE_MENU_OPTIONS[NORMAL_MODE_MENU_ITEMS] = {
99     "Continuous_Mode",
100     "Input-based_Mode"
101 };
102
103
104 bool objectDetected = false;
105 int currentClassIndex = -1;
106 uint16_t pwmDutyCycle = 0;
107
108 // Input-based mode variables
109 bool inputBasedMode = false;
110 #define INPUT_TRIGGER_PIN 47 // Define the pin for digital input
    trigger
111 bool lastTriggerState = HIGH;
112 bool triggerPressed = false;
113

```

```

114  /* Variveis de controle */
115  // Botes
116  unsigned long buttonPressedTime = 0;
117  bool selectBtnPressed = false;
118  bool navBtnPressed = false;
119  bool navBtnState = false;
120  unsigned long navBtnLastDebounceTime = 0;
121  unsigned long debounceDelay = 50;
122  bool waitForButtonRelease = false;
123  bool navBtnPressedInSubmenu = false;
124  unsigned long navBtnLastPressTime = 0;
125  const unsigned long NAV_BTN_DEBOUNCE_DELAY = 200;  // 200ms debounce
               for menu navigation
126
127  // Estados do sistema
128  bool wifiConnected = false;
129  bool configMode = false;
130  bool imageCollectionMode = false;
131  String messageText = "Iniciando_ML...";
132  String recognitionLabel = "";
133  float recognitionConfidence = 0.0;
134  unsigned long lastInferenceTime = 0;
135  const unsigned long INFERENCE_INTERVAL = 500;  // 0.5 segundos entre
               inferncias
136
137  // HUD de Configurao
138  int configMenuIndex = 0;
139  const int CONFIG_MENU_ITEMS = 3;  // WiFi Config, Normal Mode, Image
               Collection
140  const String CONFIG_MENU_OPTIONS[CONFIG_MENU_ITEMS] = {
141      "WiFi_Config",
142      "Normal_Mode",
143      "Image_Collection"
144  };
145
146  // TCP/PLC connection variables
147  WiFiClient tcpClient;
148  bool isTcpConnected = false;
149  unsigned long lastTcpActivityTime = 0;
150  const unsigned long TCP_CONNECTION_TIMEOUT = 30000;  // 30 seconds
151  uint8_t tcpBuffer[MAX_ARRAY_SIZE];

```

```

152 uint8_t lastSentTcpBuffer[MAX_ARRAY_SIZE];
153 bool hasValidLastTcpData = false;
154 IPAddress plc_ip(192, 168, 1, 100);  //
155
156 #define PREFS_NAMESPACE "plc_config"
157 #define PREFS_PLC_IP_KEY "plc_ip"
158
159 // Nome do AP para configuracao WiFi
160 #define AP_NAME "OnDemandAP"
161
162 unsigned long lastDisplayUpdate = 0;
163
164 // Timeout para o portal de configuracao (em segundos)
165 int configTimeout = 120;
166
167 // cones para indicador de WiFi (8x8 pixels)
168 static const unsigned char PROGMEM wifi_connected_icon[] = {
169     0x00, 0x7e, 0x81, 0x3c, 0x42, 0x18, 0x00, 0x00
170 };
171
172 static const unsigned char PROGMEM wifi_disconnected_icon[] = {
173     0x01, 0x45, 0x29, 0x11, 0x28, 0x45, 0x00, 0x00
174 };
175
176
177 static const unsigned char PROGMEM startup_logo[] = {
178     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0
179     xff, 0xff, 0xff, 0xff, 0xff, 0xff,
180     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
181     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
182     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
183     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
184     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
185     0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff

```

```

, 0xff, 0xff, 0xff, 0xff, 0xff,
186 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
, 0xff, 0xff, 0xff, 0xff, 0xff,
187 0x80, 0x00, 0x00, 0x00, 0x00, 0x3f, 0x00, 0x00, 0x00, 0x03, 0xff
, 0x80, 0x00, 0x00, 0x00, 0xff,
188 0x80, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x00, 0xfe
, 0x00, 0x00, 0x00, 0x00, 0x3f,
189 0x80, 0x00, 0x00, 0x00, 0x00, 0x38, 0x00, 0x00, 0x00, 0x00, 0x3c
, 0x00, 0x00, 0x00, 0x00, 0x1f,
190 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xf0, 0x7f, 0xff, 0xff, 0xf8, 0x18
, 0x3f, 0xff, 0xff, 0xfe, 0x0f,
191 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xe1, 0xff, 0xff, 0xff, 0xfe, 0x00
, 0x7f, 0xff, 0xff, 0xff, 0x07,
192 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xc3, 0xff, 0xff, 0xff, 0xff, 0x00
, 0xff, 0xff, 0xff, 0xff, 0xc3,
193 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xc7, 0xff, 0xff, 0xff, 0xff, 0x81
, 0xff, 0xff, 0xff, 0xff, 0xc3,
194 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x87, 0xff, 0xff, 0xff, 0xff, 0x83
, 0xff, 0xff, 0xff, 0xff, 0xe1,
195 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xff, 0xff, 0xff, 0xc3
, 0xff, 0xff, 0xff, 0xff, 0xe1,
196 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xff, 0xff, 0xff, 0xc3
, 0xff, 0xff, 0xff, 0xff, 0xf1,
197 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xfe, 0x3f, 0xff, 0xe7
, 0xff, 0xfe, 0xbf, 0xff, 0xf1,
198 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xfc, 0x1f, 0xff, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
199 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xfe, 0x0f, 0xff, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
200 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xff, 0x07, 0xff, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
201 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xff, 0x83, 0xff, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
202 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x8f, 0xff, 0xff, 0xc1, 0xff, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
203 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x87, 0xff, 0xff, 0xe0, 0xff, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
204 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0x83, 0xff, 0xff, 0xf0, 0x7f, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,
205 0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xc1, 0xff, 0xff, 0xfc, 0x3f, 0xff
, 0xff, 0xfe, 0x3f, 0xff, 0xf1,

```

```

206      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xe0, 0xff, 0xff, 0xfe, 0x0f, 0xff
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
207      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xf0, 0x7f, 0xff, 0xff, 0x07, 0xff
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
208      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xf8, 0x3f, 0xff, 0xff, 0x83, 0xff
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
209      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xfc, 0x1f, 0xff, 0xff, 0xc1, 0xff
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
210      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xfe, 0x0f, 0xff, 0xff, 0xe0, 0xff
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
211      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0x87, 0xff, 0xff, 0xf0, 0x7f
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
212      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xc1, 0xff, 0xff, 0xf8, 0x3f
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
213      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xe0, 0xff, 0xff, 0xfc, 0x1f
      , 0xff, 0xfe, 0x3f, 0xff, 0xf1,
214      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xf0, 0x7f, 0xff, 0xff, 0x0f
      , 0xff, 0xfe, 0x1f, 0xff, 0xe1,
215      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xf8, 0x3f, 0xff, 0xff, 0x87
      , 0xff, 0xfe, 0x00, 0x00, 0x01,
216      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xfc, 0x1f, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x00, 0x00, 0x01,
217      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xfe, 0x0f, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x00, 0x00, 0x01,
218      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xff, 0x07, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
219      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xff, 0x83, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
220      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xff, 0xe1, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
221      0x8f, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0xff, 0xf0, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
222      0x8f, 0xff, 0xff, 0xff, 0xff, 0xe3, 0xff, 0xf9, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
223      0x8f, 0xff, 0xff, 0xff, 0xff, 0xc3, 0xff, 0xff, 0xff, 0xff, 0xc3
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
224      0x87, 0xff, 0xff, 0xff, 0xff, 0xc3, 0xff, 0xff, 0xff, 0xff, 0xc7
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
225      0x87, 0xff, 0xff, 0xff, 0xff, 0xc1, 0xff, 0xff, 0xff, 0xff, 0x87
      , 0xff, 0xfe, 0x3f, 0xff, 0xff,
226      0xc3, 0xff, 0xff, 0xff, 0xff, 0x81, 0xff, 0xff, 0xff, 0xff, 0x87

```



```

    , 0xff, 0xfe, 0x3f, 0xff, 0xff,
227    0xc3, 0xff, 0xff, 0xff, 0xff, 0x00, 0xff, 0xff, 0xff, 0xff, 0x0f
    , 0xff, 0xfe, 0x3f, 0xff, 0xff,
228    0xe0, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x7f, 0xff, 0xff, 0xfe, 0x0f
    , 0xff, 0xfe, 0x3f, 0xff, 0xff,
229    0xf0, 0x7f, 0xff, 0xff, 0xf8, 0x18, 0x1f, 0xff, 0xff, 0xf8, 0x1f
    , 0xff, 0xfe, 0x3f, 0xff, 0xff,
230    0xf8, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x3c
    , 0x00, 0x00, 0x3f, 0xff, 0xff,
231    0xfc, 0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00, 0x00, 0x00, 0xfc
    , 0x00, 0x00, 0x3f, 0xff, 0xff,
232    0xff, 0x00, 0x00, 0x00, 0x03, 0xff, 0xc0, 0x00, 0x00, 0x03, 0xfc
    , 0x00, 0x00, 0x3f, 0xff, 0xff,
233    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
234    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
235    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
236    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
237    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
238    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
239    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
240    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
241    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff,
242    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
    , 0xff, 0xff, 0xff, 0xff, 0xff
243 };
244
245
246 /* Private variables for camera
    ----- */
247 static bool debug_nn = false; // Set this to true to see e.g.
    features generated from the raw signal
248 static bool is_initialised = false;

```

```

249 uint8_t *snapshot_buf; //points to the output of the capture
250
251 static camera_config_t camera_config = {
252     .pin_pwdn = PWDN_GPIO_NUM,
253     .pin_reset = RESET_GPIO_NUM,
254     .pin_xclk = XCLK_GPIO_NUM,
255     .pin_sscb_sda = SIOD_GPIO_NUM,
256     .pin_sscb_scl = SIOC_GPIO_NUM,
257
258     .pin_d7 = Y9_GPIO_NUM,
259     .pin_d6 = Y8_GPIO_NUM,
260     .pin_d5 = Y7_GPIO_NUM,
261     .pin_d4 = Y6_GPIO_NUM,
262     .pin_d3 = Y5_GPIO_NUM,
263     .pin_d2 = Y4_GPIO_NUM,
264     .pin_d1 = Y3_GPIO_NUM,
265     .pin_d0 = Y2_GPIO_NUM,
266     .pin_vsync = VSYNC_GPIO_NUM,
267     .pin_href = HREF_GPIO_NUM,
268     .pin_pclk = PCLK_GPIO_NUM,
269
270     //XCLK 20MHz or 10MHz for OV2640 double FPS (Experimental)
271     .xclk_freq_hz = 20000000,
272     .ledc_timer = LEDC_TIMER_0,
273     .ledc_channel = LEDC_CHANNEL_0,
274
275     .pixel_format = PIXFORMAT_JPEG, //YUV422, GRAYSCALE, RGB565, JPEG
276     .frame_size = FRAMESIZE_QVGA, //QQVGA-UXGA Do not use sizes
        above QVGA when not JPEG
277
278     .jpeg_quality = 12, //0-63 lower number means higher quality
279     .fb_count = 1, //if more than one, i2s runs in continuous
        mode. Use only with JPEG
280     .fb_location = CAMERA_FB_IN_PSRAM,
281     .grab_mode = CAMERA_GRAB_WHEN_EMPTY,
282 };
283
284 /* WiFiManager callback functions
        ----- */
285 // Callback para quando o WiFiManager falha em conectar durante a
        configuracao

```

```

286 void configModeCallback(WiFiManager *myWiFiManager) {
287     Serial.println("Entrou_no_modulo_de_configuracao");
288     configMode = true;
289
290     display.clearDisplay();
291     display.setCursor(0, 0);
292     display.println(F("Modo_de_Configuracao"));
293     display.println(F("Conecte_ao_AP:"));
294     display.println(F("_" AP_NAME));
295     display.println(F("Acesse:_192.168.4.1"));
296     display.display();
297 }
298
299 /* Function declarations
    ----- */
300 bool ei_camera_init(void);
301 void ei_camera_deinit(void);
302 bool ei_camera_capture(uint32_t img_width, uint32_t img_height,
    uint8_t *out_buf);
303 static int ei_camera_get_data(size_t offset, size_t length, float *
    out_ptr);
304 bool connectToSavedWiFi();
305 void startWiFiConfig();
306 void updateDisplay();
307 void updateConfigMenu();
308 void checkButtons();
309 void processConfigMenuSelection();
310 void startImageCollectionMode();
311 void runInference();
312 void checkForCommands();
313 void connectToTCP();
314 void checkTcpConnection();
315 void sendDataToTCP(String data);
316 void processNormalModeSelection();
317 void loadPLCIPFromPreferences();
318 void savePLCIPToPreferences();
319 void setPLCIPFromSerial(String ipString);
320 bool isValidIP(String ip);
321
322
323 void connectToTCP() {

```

```

324  if (isTcpConnected) return;
325
326  // Add delay between connection attempts to prevent overwhelming
    the server
327  static unsigned long lastConnectionAttempt = 0;
328  if (millis() - lastConnectionAttempt < 5000) { // Wait 5 seconds
    between attempts
329      return;
330  }
331  lastConnectionAttempt = millis();
332
333  Serial.println("Attempting_to_connect_to_TCP_server/PLC...");
334
335  // Stop any existing connection first
336  if (tcpClient.connected()) {
337      tcpClient.stop();
338      delay(100);
339  }
340
341  if (tcpClient.connect(plc_ip, PLC_PORT)) {
342      Serial.println("Connected_to_TCP_server/PLC!");
343      isTcpConnected = true;
344      lastTcpActivityTime = millis();
345  } else {
346      Serial.println("Connection_to_TCP_server/PLC_failed._Will_retry_
    in_5_seconds...");
347      tcpClient.stop(); // Ensure clean state
348  }
349 }
350
351 void checkTcpConnection() {
352     // Check if client is still connected
353     if (isTcpConnected) {
354         if (!tcpClient.connected()) {
355             Serial.println("Connection_to_TCP_server/PLC_lost.");
356             isTcpConnected = false;
357             tcpClient.stop();
358             return;
359         }
360
361         // Check for timeout (optional - remove if not needed)

```

```

362     if (millis() - lastTcpActivityTime > TCP_CONNECTION_TIMEOUT) {
363         Serial.println("TCP_connection_timeout._Disconnecting...");
364         isTcpConnected = false;
365         tcpClient.stop();
366         return;
367     }
368 }
369 }
370
371
372 void sendDataToTCP(String data) {
373     if (!isTcpConnected || !tcpClient.connected()) {
374         Serial.println("Not_connected_to_TCP_server._Cannot_send_data.")
375         ;
376         isTcpConnected = false;  // Ensure flag is consistent
377         return;
378     }
379     // Limit string length
380     if (data.length() > MAX_ARRAY_SIZE - 5) {
381         data = data.substring(0, MAX_ARRAY_SIZE - 5);
382         Serial.println("TCP_data_truncated_to_" + String(MAX_ARRAY_SIZE
383         - 5) + "_characters");
384     }
385     uint16_t stringLength = data.length();
386
387     // Create fixed-length buffer
388     memset(tcpBuffer, 0, MAX_ARRAY_SIZE);
389
390     // Add START marker
391     tcpBuffer[0] = 0x02;  // STX
392
393     // Copy data
394     for (int i = 0; i < stringLength; i++) {
395         tcpBuffer[i + 1] = data.charAt(i);
396     }
397
398     // Add END marker
399     tcpBuffer[stringLength + 1] = 0x03;  // ETX
400

```

```

401 // Send data - try to send only the actual data length + headers
    instead of full buffer
402 int actualDataLength = stringLength + 2; // STX + data + ETX
403 int bytesSent = tcpClient.write(tcpBuffer, actualDataLength);
404
405 if (bytesSent == actualDataLength) {
406     Serial.print("TCP_data_sent_successfully_");
407     Serial.print(stringLength);
408     Serial.print("_chars:_");
409     Serial.println(data);
410
411     // Store as last valid data
412     memcpy(lastSentTcpBuffer, tcpBuffer, MAX_ARRAY_SIZE);
413     hasValidLastTcpData = true;
414     lastTcpActivityTime = millis();
415 } else {
416     Serial.print("Error_sending_TCP_data._Only_");
417     Serial.print(bytesSent);
418     Serial.print("_of_");
419     Serial.print(actualDataLength);
420     Serial.println("_bytes_sent.");
421
422     // Check if connection is still valid
423     if (!tcpClient.connected()) {
424         Serial.println("TCP_connection_lost_during_send._Marking_as_
disconnected.");
425         isConnected = false;
426         tcpClient.stop();
427     }
428 }
429 }
430
431 /**
432  * @brief      Arduino setup function
433  */
434 void setup() {
435     Wire.begin(45, 48);
436     // Inicializa o Serial
437     Serial.begin(115200);
438     loadPLCIPFromPreferences();
439     Serial.println("\nIniciando...");

```

```

440
441 // Configura os pinos dos botes
442 pinMode(SELECT_BTN_PIN, INPUT_PULLUP);
443 pinMode(NAV_BTN_PIN, INPUT_PULLUP);
444 pinMode(INPUT_TRIGGER_PIN, INPUT_PULLDOWN); // Pull-down resistor
445
446 // Configure output pins
447 pinMode(DIGITAL_OUTPUT_PIN, OUTPUT);
448 digitalWrite(DIGITAL_OUTPUT_PIN, LOW);
449
450 // Configure PWM
451 ledcSetup(PWM_CHANNEL, PWM_FREQUENCY, PWM_RESOLUTION);
452 ledcAttachPin(PWM_OUTPUT_PIN, PWM_CHANNEL);
453 ledcWrite(PWM_CHANNEL, 0);
454
455 // Inicializa o display OLED
456 if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
457     Serial.println(F("Falha ao inicializar o display SSD1306"));
458     for (;;)
459         ; // Loop infinito se o display não inicializar
460 }
461
462 // Exibe mensagem de inicialização
463 // Display startup logo
464 display.clearDisplay();
465 display.drawBitmap(0, 0, startup_logo, 128, 64, SSD1306_WHITE);
466 display.display();
467 delay(LOGO_DISPLAY_TIME); // Show logo for defined time
468
469
470 // Reset display settings after logo
471 display.clearDisplay();
472 display.setTextColor(SSD1306_WHITE);
473 display.setTextSize(1);
474
475
476 // Configura o WiFi em modo estático, mas não tenta conectar ainda
477 WiFi.mode(WIFI_STA);
478
479 // Inicializa a câmera
480 Serial.println("Inicializando câmera...");

```

```

481 display.clearDisplay();
482 display.setCursor(0, 0);
483 display.println(F("Iniciando_camera..."));
484 display.display();
485
486 if (ei_camera_init() == false) {
487     Serial.println("Falha ao inicializar a camera!");
488     display.clearDisplay();
489     display.setCursor(0, 0);
490     display.println(F("Erro na camera!"));
491     display.display();
492     delay(2000);
493 } else {
494     Serial.println("Cmera inicializada");
495 }
496
497 // Vai direto para o menu de configuracao em vez de tentar conectar
   ao WiFi
498 configMode = true;
499 configMenuIndex = 0;
500 updateConfigMenu();
501
502 Serial.println("Modo de configuracao iniciado. Aguardando seleo...")
   ;
503
504 // Initialize TCP connection variables
505 memset(tcpBuffer, 0, MAX_ARRAY_SIZE);
506 memset(lastSentTcpBuffer, 0, MAX_ARRAY_SIZE);
507 isConnected = false;
508 }
509
510 /**
511  * @brief      Main loop
512  */
513 void loop() {
514     // Verifica o status da conexo WiFi (apenas se no estiver no modo
   de configurao)
515     if (!configMode && !imageCollectionMode) {
516         if (WiFi.status() == WL_CONNECTED && !wifiConnected) {
517             wifiConnected = true;
518             updateDisplay();

```



```

519     } else if (WiFi.status() != WL_CONNECTED && wifiConnected) {
520         wifiConnected = false;
521         updateDisplay();
522     }
523 }
524
525 // Verifica os botes
526 checkButtons();
527
528 if (!configMode && !imageCollectionMode && wifiConnected) {
529     checkTcpConnection();
530     if (!isTcpConnected) {
531         connectToTCP();
532     }
533 }
534
535 // Executa a inferncia baseada no modo selecionado
536 if (!configMode && !normalModeSubmenu && !imageCollectionMode) {
537     // Check for input trigger in input-based mode
538     checkInputTrigger();
539
540     bool shouldRunInference = false;
541
542     if (inputBasedMode) {
543         // Run inference only when trigger is pressed
544         if (triggerPressed) {
545             shouldRunInference = true;
546             triggerPressed = false; // Reset trigger
547         }
548     } else {
549         // Continuous mode - run inference periodically
550         if (millis() - lastInferenceTime >= INFERENCE_INTERVAL) {
551             shouldRunInference = true;
552         }
553     }
554
555     if (shouldRunInference) {
556         runInference();
557         updateDisplay();
558         lastInferenceTime = millis();
559     }

```

```

560     }
561
562     // Verifica se h comandos pela serial (sem sobrescrever a
        classificacao)
563     if (!configMode && !imageCollectionMode) {
564         checkForCommands();
565     }
566
567     delay(10); // Pequeno delay para evitar problemas de debounce e
        consumo de CPU
568 }
569
570 /**
571  * @brief      Check button states and handle long presses
572  */
573 void checkButtons() {
574     // Verifica o boto de seleo (antigo boto)
575     if (digitalRead(SELECT_BTN_PIN) == LOW) {
576         if (!selectBtnPressed) {
577             selectBtnPressed = true;
578             buttonPressedTime = millis();
579             Serial.println("Boto_de_seleo_pressionado");
580         } else {
581             // Verifica se o boto foi pressionado por 10 segundos e no
                estamos no modo de configurao
582             if (!configMode && !imageCollectionMode && millis() -
                buttonPressedTime >= 10000) {
583                 Serial.println("Boto_pressionado_por_10_segundos_-_Iniciando
                    _menu_de_configurao");
584                 configMode = true;
585                 configMenuIndex = 0;
586                 updateConfigMenu();
587                 selectBtnPressed = false; // Reseta o estado do boto
588                 waitForButtonRelease = true; // Flag para esperar que o
                    boto seja solto
589             }
590         }
591     } else {
592         if (selectBtnPressed) {
593             // Se estiver no modo de configurao e o boto for solto antes
                de 10s, seleciona a opo

```

```

594     if (configMode && !waitForButtonRelease && (millis() -
buttonPressedTime < 10000)) {
595         if (normalModeSubmenu) {
596             processNormalModeSelection();
597         } else {
598             processConfigMenuSelection();
599         }
600     }
601     selectBtnPressed = false; // Boto foi solto
602
603     // Se estvamos esperando que o boto fosse solto aps entrar no
modo de configurao
604     if (waitForButtonRelease) {
605         waitForButtonRelease = false; // Limpa a flag agora que o
boto foi solto
606     }
607 }
608 }
609
610 // Verifica o boto de navegao (novo boto) - Updated debounce logic
611 int navBtnReading = digitalRead(NAV_BTN_PIN);
612
613 // Se a leitura mudou devido a rudo ou presso
614 if (navBtnReading != navBtnState) {
615     navBtnLastDebounceTime = millis();
616 }
617
618 if ((millis() - navBtnLastDebounceTime) > debounceDelay) {
619     // Se a leitura foi estvel por tempo suficiente
620     if (navBtnReading != navBtnPressed) {
621         navBtnPressed = navBtnReading;
622
623         // Se o boto foi pressionado (LOW) e estamos no modo de
configurao
624         if (navBtnPressed == LOW && configMode) {
625             // Additional debounce check for menu navigation
626             if (millis() - navBtnLastPressTime > NAV_BTN_DEBOUNCE_DELAY)
{
627                 if (normalModeSubmenu) {
628                     normalModeMenuIndex = (normalModeMenuIndex + 1) %
NORMAL_MODE_MENU_ITEMS;

```

```

629         Serial.println("Navegando_para_opo_normal:_ " +
NORMAL_MODE_MENU_OPTIONS[normalModeMenuIndex]);
630         updateNormalModeMenu();
631     } else {
632         configMenuIndex = (configMenuIndex + 1) %
CONFIG_MENU_ITEMS;
633         Serial.println("Navegando_para_opo:_ " +
CONFIG_MENU_OPTIONS[configMenuIndex]);
634         updateConfigMenu();
635     }
636     navBtnLastPressTime = millis();
637 }
638 }
639 }
640 }
641
642 navBtnState = navBtnReading;
643 }
644
645 /**
646  * @brief      Process the selected menu option
647  */
648 void processConfigMenuSelection() {
649     Serial.println("Selecionando_opo:_ " + CONFIG_MENU_OPTIONS[
configMenuIndex]);
650
651     switch (configMenuIndex) {
652     case 0: // WiFi Config
653         startWiFiConfig();
654         break;
655
656     case 1: // Normal Mode - Now opens submenu
657         normalModeSubmenu = true;
658         normalModeMenuIndex = 0;
659         updateNormalModeMenu();
660         break;
661
662     case 2: // Image Collection
663         // Tenta conectar ao WiFi salvo antes de entrar no modo de
coleta
664         if (!wifiConnected) {

```

```

665     display.clearDisplay();
666     display.setCursor(0, 0);
667     display.println(F("Conectando_ao_WiFi..."));
668     display.display();
669
670     wifiConnected = connectToSavedWiFi();
671 }
672
673     startImageCollectionMode();
674     break;
675 }
676 }
677
678 /**
679  * @brief      Start image collection server
680  */
681 void startImageCollectionMode() {
682     display.clearDisplay();
683     display.setCursor(0, 0);
684     display.println(F("Modo_de_Colecao"));
685     display.println(F("de_Imagens"));
686     display.println(F("Iniciando..."));
687     display.display();
688
689     // Desabilita o modo de configuracao e ativa o modo de coleta de
        imagens
690     configMode = false;
691     imageCollectionMode = true;
692
693     // Verificar se j est conectado ao WiFi
694     if (WiFi.status() != WL_CONNECTED) {
695         display.clearDisplay();
696         display.setCursor(0, 0);
697         display.println(F("Conectando_ao_WiFi..."));
698         display.println(WIFI_SSID);
699         display.display();
700
701         // Tenta conectar com as credenciais definidas
702         WiFi.begin(WIFI_SSID, WIFI_PASS);
703
704         // Espera pela conexo com timeout

```

```

705     int timeout = 0;
706     while (WiFi.status() != WL_CONNECTED && timeout < 20) {
707         delay(500);
708         timeout++;
709         display.print(".");
710         display.display();
711     }
712
713     if (WiFi.status() != WL_CONNECTED) {
714         display.clearDisplay();
715         display.setCursor(0, 0);
716         display.println(F("Falha_ao_conectar"));
717         display.println(F("Retornando_ao_menu..."));
718         display.display();
719         delay(2000);
720         configMode = true;
721         imageCollectionMode = false;
722         updateConfigMenu();
723         return;
724     }
725 }
726
727 // Inicializa o servidor de coleta de imagens
728 camera.pinout.wroom_s3();
729 camera.brownout.disable();
730 camera.resolution.face(); // 240x240
731 camera.quality.high();
732
733 // Reinicializa a cmera para o modo de coleta de imagens
734 ei_camera_deinit();
735
736 bool cameraInitSuccess = camera.begin().isOk();
737 bool serverInitSuccess = false;
738
739 if (cameraInitSuccess) {
740     serverInitSuccess = collectionServer.begin().isOk();
741 }
742
743 display.clearDisplay();
744 display.setCursor(0, 0);
745

```

```

746  if (cameraInitSuccess && serverInitSuccess) {
747      display.println(F("Servidor_Iniciado!"));
748      display.println(collectionServer.address());
749      display.println(F("Pressione_Sel_10s"));
750      display.println(F("para_sair"));
751  } else {
752      display.println(F("Falha_ao_iniciar"));
753      display.println(F("servidor_de_imagens"));
754      display.println(F("Retornando_ao_menu..."));
755      delay(3000);
756      configMode = true;
757      imageCollectionMode = false;
758      updateConfigMenu();
759
760      // Reinicializa a cmera para o modo normal
761      if (ei_camera_init() == false) {
762          messageText = "Erro_ao_reiniciar_cam";
763      }
764      return;
765  }
766
767  display.display();
768
769  // Loop para o modo de coleta de imagens, continuando a verificar
o boto de sada
770  while (imageCollectionMode) {
771      // Verifica se o boto de seleo est pressionado por 10s para sair
772      if (digitalRead(SELECT_BTN_PIN) == LOW) {
773          if (!selectBtnPressed) {
774              selectBtnPressed = true;
775              buttonPressedTime = millis();
776          } else if (millis() - buttonPressedTime >= 10000) {
777              // Sair do modo de coleta de imagens
778              selectBtnPressed = false;
779              imageCollectionMode = false;
780              configMode = true;
781
782              display.clearDisplay();
783              display.setCursor(0, 0);
784              display.println(F("Saindo_do_modode"));
785              display.println(F("coleta_de_imagens..."));

```

```

786     display.display();
787     delay(1000);
788
789     // Reinicializa a camera para o modo normal
790     if (ei_camera_init() == false) {
791         messageText = "Erro_ao_reiniciar_cam";
792     }
793
794     updateConfigMenu();
795     break;
796 }
797 } else {
798     selectBtnPressed = false;
799 }
800
801     delay(100);
802 }
803 }
804
805 /**
806  * @brief      Update the configuration menu on display
807  */
808 void updateConfigMenu() {
809     display.clearDisplay();
810
811     // --- HEADER SECTION (0-12px) ---
812     display.setTextSize(1);
813     display.setCursor(0, 2);
814     display.print(F("Menu_de_Configuracao"));
815
816     // Desenha uma linha de separao
817     display.drawLine(0, 12, SCREEN_WIDTH, 12, SSD1306_WHITE);
818
819     // --- MAIN CONTENT (14-48px) ---
820     // Mostra todas as opes, destacando a selecionada
821     for (int i = 0; i < CONFIG_MENU_ITEMS; i++) {
822         display.setCursor(10, 16 + i * 10);
823
824         if (i == configMenuIndex) {
825             display.print(F(">_"));
826         } else {

```



```

827     display.print(F("_"));
828 }
829     display.println(CONFIG_MENU_OPTIONS[i]);
830 }
831
832 // --- FOOTER SECTION (50-64px) ---
833 display.drawLine(0, 50, SCREEN_WIDTH, 50, SSD1306_WHITE);
834 display.setCursor(2, 54);
835 display.print(F("Nav: _Mudar _Sel: _OK"));
836
837 display.display();
838 }
839
840 /**
841  * @brief      Run inference and update display with results
842  */
843 void runInference() {
844     // Aloca buffer para a imagem
845     snapshot_buf = (uint8_t *)malloc(EI_CAMERA_RAW_FRAME_BUFFER_COLS *
846         EI_CAMERA_RAW_FRAME_BUFFER_ROWS * EI_CAMERA_FRAME_BYTE_SIZE);
847
848     // Verifica se a alocao foi bem-sucedida
849     if (snapshot_buf == nullptr) {
850         Serial.println("ERRO: _Falha_ao_alocar_buffer!");
851         messageText = "Erro: _Sem_memoria!";
852         updateDisplay();
853         return;
854     }
855
856     // Prepara o sinal para o classificador
857     ei::signal_t signal;
858     signal.total_length = EI_CLASSIFIER_INPUT_WIDTH *
859         EI_CLASSIFIER_INPUT_HEIGHT;
860     signal.get_data = &ei_camera_get_data;
861
862     // Captura a imagem
863     if (ei_camera_capture((size_t)EI_CLASSIFIER_INPUT_WIDTH, (size_t)
864         EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf) == false) {
865         Serial.println("Falha_ao_capturar_imagem");
866         messageText = "Erro: _Captura_falhou";
867         updateDisplay();
868     }
869 }

```

```

865     free(snapshot_buf);
866     snapshot_buf = NULL;
867     yield();
868     return;
869 }
870 yield();
871
872 // Executa o classificador
873 ei_impulse_result_t result = { 0 };
874 EI_IMPULSE_ERROR err = run_classifier(&signal, &result, debug_nn);
875
876 if (err != EI_IMPULSE_OK) {
877     Serial.printf("ERRO: Falha ao executar classificador (%d)\n",
878 err);
879     messageText = "Erro: Classificador";
880     updateDisplay();
881     free(snapshot_buf);
882     snapshot_buf = NULL;
883     yield();
884     return;
885 }
886
887 // Exibe os resultados da predição no serial
888 Serial.printf("Predictions (DSP: %d ms., Classification: %d ms.,
889 Anomaly: %d ms.): \n",
890 result.timing.dsp, result.timing.classification,
891 result.timing.anomaly);
892
893 // Reset outputs at the beginning of each inference
894 objectDetected = false;
895 currentClassIndex = -1;
896 digitalWrite(DIGITAL_OUTPUT_PIN, LOW);
897 ledcWrite(PWM_CHANNEL, 0);
898
899 #if EI_CLASSIFIER_OBJECT_DETECTION == 1
900 // Deteco de objetos
901 Serial.println("Object detection bounding boxes:");
902 recognitionLabel = "";
903 recognitionConfidence = 0.0;
904 String tcpData = ""; // String for TCP transmission
905

```

```

903  for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
904      ei_impulse_result_bounding_box_t bb = result.bounding_boxes[i];
905      if (bb.value == 0) {
906          continue;
907      }
908      Serial.printf("_%s_(%.2f%%)_[x:_%u,_y:_%u,_width:_%u,_height:_%u_]\\r\\n",
909                  bb.label, bb.value * 100,
910                  bb.x, bb.y, bb.width, bb.height);
911
912      // Guarda a deteco com maior confiana
913      if (bb.value > recognitionConfidence) {
914          recognitionLabel = String(bb.label);
915          recognitionConfidence = bb.value;
916
917          // Set digital output high when object is detected
918          objectDetected = true;
919          digitalWrite(DIGITAL_OUTPUT_PIN, HIGH);
920
921          // Find class index for PWM (you may need to adjust this based
922          on your labels)
923          // This assumes your classes are numbered or you have a
924          specific mapping
925          for (int j = 0; j < EI_CLASSIFIER_LABEL_COUNT; j++) {
926              if (String(ei_classifier_inferencing_categories[j]) == bb.
927                  label) {
928                  currentClassIndex = j;
929                  break;
930              }
931          }
932
933          // Prepare TCP data with label and bounding box coordinates
934          if (tcpData.length() > 0) tcpData += ";"; // Separator for
935          multiple detections
936          tcpData += String(bb.label) + ":" + String(bb.value * 100, 1) +
937                  ":" + String(bb.x) + ":" + String(bb.y) + ":" + String(bb.width)
938                  + ":" + String(bb.height);
939      }
940
941      if (recognitionLabel == "") {

```

```

937     messageText = "-";
938     tcpData = "NO_DETECTION";
939 } else {
940     messageText = recognitionLabel + "_" + String(int(
        recognitionConfidence * 100)) + "%";
941 }
942
943 // Send data via TCP if connected and there's data to send
944 if (wifiConnected && !configMode && !imageCollectionMode &&
        tcpData.length() > 0) {
945     sendDataToTCP(tcpData);
946 }
947
948 #else
949     // Classificao
950     Serial.println("Predictions:");
951
952     // Encontra a classe com maior confiana
953     float highestConfidence = 0;
954     int bestIdx = -1;
955
956     for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
957         Serial.printf("_%s: %.5f\r\n",
            ei_classifier_inferencing_categories[i], result.classification[i]
            .value);
958         if (result.classification[i].value > highestConfidence) {
959             highestConfidence = result.classification[i].value;
960             bestIdx = i;
961         }
962     }
963
964     String tcpData = ""; // String for TCP transmission
965
966     if (bestIdx >= 0) {
967         recognitionLabel = String(ei_classifier_inferencing_categories[
            bestIdx]);
968         recognitionConfidence = highestConfidence;
969         messageText = recognitionLabel + "_" + String(int(
            recognitionConfidence * 100)) + "%";
970
971         // Set digital output high when object is detected with

```

```

sufficient confidence
972     if (recognitionConfidence > 0.5) { // Adjust threshold as
needed
973         objectDetected = true;
974         currentClassIndex = bestIdx;
975         digitalWrite(DIGITAL_OUTPUT_PIN, HIGH);
976     }
977
978     // Prepare TCP data for classification (without bounding box
coordinates)
979     tcpData = recognitionLabel + ":" + String(recognitionConfidence
* 100, 1);
980 } else {
981     messageText = "Classificacao_falhou";
982     tcpData = "CLASSIFICATION_FAILED";
983 }
984
985 // Send data via TCP if connected
986 if (wifiConnected && !configMode && !imageCollectionMode &&
tcpData.length() > 0) {
987     sendDataToTCP(tcpData);
988 }
989 #endif
990
991 // Set PWM based on detected class
992 if (objectDetected && currentClassIndex >= 0) {
993     // Calculate duty cycle: (2^14-1)/(EI_CLASSIFIER_LABEL_COUNT+1)
* (classIndex + 1)
994     uint16_t maxDuty = (1 << PWM_RESOLUTION) - 1; // 2^14 - 1 =
16383
995     pwmDutyCycle = (maxDuty / (EI_CLASSIFIER_LABEL_COUNT + 1)) * (
currentClassIndex + 1);
996     ledcWrite(PWM_CHANNEL, pwmDutyCycle);
997
998     Serial.printf("Class_%d_detected,_PWM_duty_cycle_set_to_%d/%d\n"
,
999                 currentClassIndex, pwmDutyCycle, maxDuty);
1000 }
1001
1002 // Mostra anomalia se existir
1003 #if EI_CLASSIFIER_HAS_ANOMALY

```

```

1004     Serial.printf("Anomaly_prediction: %.3f\r\n", result.anomaly);
1005     if (result.anomaly > 0.5) {
1006         messageText += "\n(Anomalia!)";
1007     }
1008 #endif
1009
1010     // Atualiza o display com o resultado
1011     yield();
1012     updateDisplay();
1013
1014     // Libera o buffer
1015     free(snapshot_buf);
1016     snapshot_buf = NULL;
1017 }
1018
1019 /**
1020  * @brief   Setup image sensor & start streaming
1021  *
1022  * @retval  false if initialisation failed
1023  */
1024 bool ei_camera_init(void) {
1025     if (is_initialised) return true;
1026
1027     //initialize the camera
1028     esp_err_t err = esp_camera_init(&camera_config);
1029     if (err != ESP_OK) {
1030         Serial.printf("Camera_init_failed_with_error_0x%x\n", err);
1031         return false;
1032     }
1033
1034     sensor_t *s = esp_camera_sensor_get();
1035     is_initialised = true;
1036     return true;
1037 }
1038
1039 /**
1040  * @brief   Stop streaming of sensor data
1041  */
1042 void ei_camera_deinit(void) {
1043     //deinitialize the camera
1044     esp_err_t err = esp_camera_deinit();

```

```

1045
1046     if (err != ESP_OK) {
1047         Serial.println("Camera_deinit_failed");
1048         return;
1049     }
1050
1051     is_initialised = false;
1052     return;
1053 }
1054
1055 /**
1056  * @brief          Capture, rescale and crop image
1057  *
1058  * @param[in]  img_width      width of output image
1059  * @param[in]  img_height     height of output image
1060  * @param[in]  out_buf        pointer to store output image, NULL may
1061                               be used
1062                               if ei_camera_frame_buffer is to be used
1063                               for capture and resize/cropping.
1064  *
1065  * @retval      false if not initialised, image captured, rescaled or
1066                 cropped failed
1067  */
1068 bool ei_camera_capture(uint32_t img_width, uint32_t img_height,
1069                        uint8_t *out_buf) {
1070     bool do_resize = false;
1071
1072     if (!is_initialised) {
1073         Serial.println("ERR:_Camera_is_not_initialized");
1074         return false;
1075     }
1076
1077     camera_fb_t *fb = esp_camera_fb_get();
1078
1079     if (!fb) {
1080         Serial.println("Camera_capture_failed");
1081         return false;
1082     }
1083
1084     bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT_JPEG,

```

```

    snapshot_buf);
1082
1083     esp_camera_fb_return(fb);
1084
1085     if (!converted) {
1086         Serial.println("Conversion_failed");
1087         return false;
1088     }
1089
1090     if ((img_width != EI_CAMERA_RAW_FRAME_BUFFER_COLS)
1091         || (img_height != EI_CAMERA_RAW_FRAME_BUFFER_ROWS)) {
1092         do_resize = true;
1093     }
1094
1095     if (do_resize) {
1096         ei::image::processing::crop_and_interpolate_rgb888(
1097             out_buf,
1098             EI_CAMERA_RAW_FRAME_BUFFER_COLS,
1099             EI_CAMERA_RAW_FRAME_BUFFER_ROWS,
1100             out_buf,
1101             img_width,
1102             img_height);
1103     }
1104
1105     return true;
1106 }
1107
1108 static int ei_camera_get_data(size_t offset, size_t length, float *
    out_ptr) {
1109     // we already have a RGB888 buffer, so recalculate offset into
    pixel index
1110     size_t pixel_ix = offset * 3;
1111     size_t pixels_left = length;
1112     size_t out_ptr_ix = 0;
1113
1114     while (pixels_left != 0) {
1115         // Swap BGR to RGB here
1116         // due to https://github.com/espressif/esp32-camera/issues/379
1117         out_ptr[out_ptr_ix] = (snapshot_buf[pixel_ix + 2] << 16) + (
            snapshot_buf[pixel_ix + 1] << 8) + snapshot_buf[pixel_ix];
1118

```



```

1119     // go to the next pixel
1120     out_ptr_ix++;
1121     pixel_ix += 3;
1122     pixels_left--;
1123 }
1124 // and done!
1125 return 0;
1126 }
1127
1128 bool connectToSavedWiFi() {
1129     // Tenta conectar usando as credenciais salvas
1130     WiFiManager wm;
1131
1132     // Define um timeout curto para a tentativa de conexo
1133     wm.setConnectTimeout(10);
1134
1135     // Tenta conectar (no inicia o portal de configurao se falhar)
1136     bool connected = wm.autoConnect(AP_NAME);
1137
1138     if (connected) {
1139         Serial.println("Conectado_ao_WiFi_usando_credenciais_salvas");
1140         return true;
1141     } else {
1142         Serial.println("Falha_ao_conectar_usando_credenciais_salvas");
1143         return false;
1144     }
1145 }
1146
1147 void startWiFiConfig() {
1148     // Ativa o modo de configurao
1149     configMode = true;
1150
1151     // Atualiza o display com informaes do modo de configurao
1152     display.clearDisplay();
1153     display.setCursor(0, 0);
1154     display.println(F("Modo_de_Configuracao"));
1155     display.println(F("Conecte_ao_AP:"));
1156     display.println(F("_" AP_NAME));
1157     display.println(F("Acesse:_192.168.4.1"));
1158     display.display();
1159

```

```

1160   WiFiManager wm;
1161
1162   // Configura callbacks para diferentes estados
1163   wm.setAPCallback(configModeCallback);
1164   wm.setBreakAfterConfig(true); // Importante: impede loops de
      reconexo
1165
1166   // Define o timeout do portal de configuracao
1167   wm.setConfigPortalTimeout(configTimeout);
1168
1169   // Inicia o portal de configuracao com o nome consistente
1170   bool connected = wm.startConfigPortal(AP_NAME);
1171
1172   // Aps sair do portal de configuracao
1173   configMode = false;
1174
1175   if (connected) {
1176       Serial.println("Conectado_ao_WiFi_aps_configuracao");
1177       wifiConnected = true;
1178
1179       // Exibe mensagem de sucesso
1180       display.clearDisplay();
1181       display.setCursor(0, 0);
1182       display.println(F("Conexao_bem-sucedida!"));
1183       display.display();
1184       delay(2000);
1185   } else {
1186       Serial.println("Falha_ao_conectar_ou_timeout_attingido");
1187       wifiConnected = false;
1188
1189       // Exibe mensagem de falha
1190       display.clearDisplay();
1191       display.setCursor(0, 0);
1192       display.println(F("Falha_na_conexao"));
1193       display.println(F("ou_timeout_attingido."));
1194       display.println(F("Reiniciando..."));
1195       display.display();
1196       delay(2000);
1197   }
1198 } // Missing closing bracket added here
1199

```

```

1200 // Atualiza o display para
1201 void updateDisplay() {
1202     // No atualiza se estiver no modo de configuracao
1203     if (configMode || normalModeSubmenu) return;
1204
1205     display.clearDisplay();
1206
1207     // --- HEADER SECTION (0-12px) ---
1208     // Desenha o indicador de WiFi no canto superior direito
1209     if (wifiConnected) {
1210         display.drawBitmap(SCREEN_WIDTH - 9, 2, wifi_connected_icon, 8,
1211             8, SSD1306_WHITE);
1212     } else {
1213         display.drawBitmap(SCREEN_WIDTH - 9, 2, wifi_disconnected_icon,
1214             8, 8, SSD1306_WHITE);
1215     }
1216
1217     // Exibe o status de conexo na primeira linha
1218     display.setTextSize(1);
1219     display.setCursor(0, 2);
1220     if (wifiConnected) {
1221         display.print(F("WiFi_conectado"));
1222     } else {
1223         display.print(F("WiFi_desconectado"));
1224     }
1225
1226     // Desenha uma linha de separao
1227     display.drawLine(0, 12, SCREEN_WIDTH, 12, SSD1306_WHITE);
1228
1229     // --- MAIN CONTENT (14-48px) ---
1230     // Exibe a mensagem de classificao com tamanho maior
1231     display.setTextSize(1);
1232
1233     // Calcula a posio para centralizar o texto
1234     int16_t x1, y1;
1235     uint16_t w, h;
1236     display.getTextBounds(messageText, 0, 0, &x1, &y1, &w, &h);
1237     int x = (SCREEN_WIDTH - w) / 2;
1238     if (x < 0) x = 0; // Evita posio negativa
1239
1240     display.setCursor(x, 28);

```

```

1239     display.println(messageText);
1240
1241     // --- FOOTER SECTION (50-64px) ---
1242     display.setTextSize(1);
1243     display.drawLine(0, 50, SCREEN_WIDTH, 50, SSD1306_WHITE);
1244
1245     // Indicador de presso do boto
1246     display.setCursor(2, 54);
1247
1248     // --- FOOTER SECTION (50-64px) ---
1249     display.setTextSize(1);
1250     display.drawLine(0, 50, SCREEN_WIDTH, 50, SSD1306_WHITE);
1251
1252     // Indicador de presso do boto
1253     display.setCursor(2, 54);
1254
1255     if (selectBtnPressed) {
1256         // Calcula o tempo de presso
1257         unsigned long pressTime = millis() - buttonPressedTime;
1258
1259         // Mostra uma barra de progresso
1260         display.print(F("Config:_"));
1261         int barLength = min(10, (int)(pressTime / 1000));
1262         for (int i = 0; i < 10; i++) {
1263             if (i < barLength) {
1264                 display.print(F("#"));
1265             } else {
1266                 display.print(F("-"));
1267             }
1268         }
1269     } else {
1270         // Show current mode
1271         if (inputBasedMode) {
1272             display.print(F("Modo:_Entrada"));
1273         } else {
1274             display.print(F("Modo:_Continuo"));
1275         }
1276     }
1277
1278     // Atualiza o display
1279     display.display();

```

```

1280 }
1281
1282
1283
1284
1285
1286 // Função para verificar se há comandos pela serial, sem alterar a
      classificação exibida
1287 void checkForCommands() {
1288     if (Serial.available() > 0) {
1289         String command = Serial.readStringUntil('\n');
1290         command.trim();
1291
1292         if (command.startsWith("CMD:")) {
1293             String cmd = command.substring(4);
1294
1295             if (cmd == "FORCE_INFERENCE") {
1296                 Serial.println("Comando_recebido:_Forando_nova_inferencia");
1297                 runInference();
1298                 updateDisplay();
1299             } else if (cmd == "WIFI_CONFIG") {
1300                 Serial.println("Comando_recebido:_Iniciando_configuracao_WiFi"
1301 );
1302                 startWiFiConfig();
1303             } else if (cmd == "STATUS") {
1304                 Serial.println("Status_atual:");
1305                 Serial.println("-_WiFi:_ " + String(wifiConnected ? "
1306 Conectado" : "Desconectado"));
1307                 Serial.println("-_PLC_IP:_ " + plc_ip.toString());
1308                 Serial.println("-_PLC_Connected:_ " + String(isTcpConnected ?
1309 "Sim" : "No"));
1310                 Serial.println("-_ultima_classificacao:_ " + recognitionLabel);
1311                 Serial.println("-_Confiança:_ " + String(recognitionConfidence
1312 ));
1313             } else if (cmd == "GET_PLC_IP") {
1314                 Serial.print("Current_PLC_IP:_");
1315                 Serial.println(plc_ip.toString());
1316             } else {
1317                 Serial.println("Comando_desconhecido:_ " + cmd);
1318             }
1319         } else if (command.startsWith("SET_PLC_IP:")) {

```

```

1316     String newIP = command.substring(11);  // Remove "SET_PLC_IP:"
1317     setPLCIPFromSerial(newIP);
1318 } else {
1319     Serial.println("Mensagem_recebida_(ignorada_para_preservar_
classificacao):_" + command);
1320 }
1321 }
1322 }
1323
1324 void processNormalModeSelection() {
1325     Serial.println("Selecioneando_modos_normal:_" +
NORMAL_MODE_MENU_OPTIONS[normalModeMenuIndex]);
1326
1327     switch (normalModeMenuIndex) {
1328     case 0:  // Continuous Mode
1329         configMode = false;
1330         normalModeSubmenu = false;
1331         imageCollectionMode = false;
1332         inputBasedMode = false;
1333
1334         // Tenta conectar ao WiFi salvo antes de entrar no modo normal
1335         if (!wifiConnected) {
1336             display.clearDisplay();
1337             display.setCursor(0, 0);
1338             display.println(F("Conectando_ao_WiFi..."));
1339             display.display();
1340
1341             wifiConnected = connectToSavedWiFi();
1342         }
1343
1344         messageText = "Modo_Contínuo";
1345         updateDisplay();
1346         break;
1347
1348     case 1:  // Input-based Mode
1349         configMode = false;
1350         normalModeSubmenu = false;
1351         imageCollectionMode = false;
1352         inputBasedMode = true;
1353
1354         // Tenta conectar ao WiFi salvo antes de entrar no modo normal

```

```

1355     if (!wifiConnected) {
1356         display.clearDisplay();
1357         display.setCursor(0, 0);
1358         display.println(F("Conectando_ao_WiFi..."));
1359         display.display();
1360
1361         wifiConnected = connectToSavedWiFi();
1362     }
1363
1364     messageText = "Aguardando_entrada...";
1365     updateDisplay();
1366     break;
1367 }
1368 }
1369
1370 void updateNormalModeMenu() {
1371     display.clearDisplay();
1372
1373     // --- HEADER SECTION (0-12px) ---
1374     display.setTextSize(1);
1375     display.setCursor(0, 2);
1376     display.print(F("Selecionar_Modo_Normal"));
1377
1378     // Desenha uma linha de separao
1379     display.drawLine(0, 12, SCREEN_WIDTH, 12, SSD1306_WHITE);
1380
1381     // --- MAIN CONTENT (14-48px) ---
1382     // Mostra todas as opes, destacando a selecionada
1383     for (int i = 0; i < NORMAL_MODE_MENU_ITEMS; i++) {
1384         display.setCursor(10, 20 + i * 12);
1385
1386         if (i == normalModeMenuIndex) {
1387             display.print(F(">_"));
1388         } else {
1389             display.print(F("_"));
1390         }
1391         display.println(NORMAL_MODE_MENU_OPTIONS[i]);
1392     }
1393
1394     // --- FOOTER SECTION (50-64px) ---
1395     display.drawLine(0, 50, SCREEN_WIDTH, 50, SSD1306_WHITE);

```

```

1396     display.setCursor(2, 54);
1397     display.print(F("Nav:  Mudar Sel: OK"));
1398
1399     display.display();
1400 }
1401
1402
1403 void checkInputTrigger() {
1404     if (!inputBasedMode) return;
1405
1406     bool currentTriggerState = digitalRead(INPUT_TRIGGER_PIN);
1407
1408     // Detect falling edge (button press)
1409     if (lastTriggerState == LOW && currentTriggerState == HIGH) {
1410         triggerPressed = true;
1411         Serial.println("Input_trigger_detected_-_Running_inference");
1412     }
1413
1414     lastTriggerState = currentTriggerState;
1415 }
1416
1417 void loadPLCIPFromPreferences() {
1418     preferences.begin(PREFS_NAMESPACE, false);
1419
1420     // Check if IP is stored
1421     if (preferences.isKey(PREFS_PLC_IP_KEY)) {
1422         String storedIP = preferences.getString(PREFS_PLC_IP_KEY, "");
1423
1424         if (storedIP.length() > 0) {
1425             // Parse the stored IP string
1426             int parts[4];
1427             int partIndex = 0;
1428             int lastDot = -1;
1429
1430             for (int i = 0; i <= storedIP.length(); i++) {
1431                 if (i == storedIP.length() || storedIP[i] == '.') {
1432                     String part = storedIP.substring(lastDot + 1, i);
1433                     parts[partIndex++] = part.toInt();
1434                     lastDot = i;
1435
1436                     if (partIndex >= 4) break;

```



```

1437     }
1438 }
1439
1440     if (partIndex == 4) {
1441         plc_ip = IPAddress(parts[0], parts[1], parts[2], parts[3]);
1442         Serial.print("PLC_IP_loaded_from_preferences:_");
1443         Serial.println(plc_ip.toString());
1444     }
1445 }
1446 } else {
1447     Serial.print("Using_default_PLC_IP:_");
1448     Serial.println(plc_ip.toString());
1449 }
1450
1451 preferences.end();
1452 }
1453
1454 void savePLCIPToPreferences() {
1455     preferences.begin(PREFS_NAMESPACE, false);
1456     String ipString = plc_ip.toString();
1457     preferences.putString(PREFS_PLC_IP_KEY, ipString);
1458     preferences.end();
1459
1460     Serial.print("PLC_IP_saved_to_preferences:_");
1461     Serial.println(ipString);
1462 }
1463
1464 bool isValidIP(String ip) {
1465     int parts[4];
1466     int partIndex = 0;
1467     int lastDot = -1;
1468
1469     // Parse IP string
1470     for (int i = 0; i <= ip.length(); i++) {
1471         if (i == ip.length() || ip[i] == '.') {
1472             if (partIndex >= 4) return false; // Too many parts
1473
1474             String part = ip.substring(lastDot + 1, i);
1475             if (part.length() == 0) return false; // Empty part
1476
1477             int value = part.toInt();

```

```

1478     if (value < 0 || value > 255) return false; // Out of range
1479
1480     parts[partIndex++] = value;
1481     lastDot = i;
1482 }
1483 }
1484
1485 return (partIndex == 4); // Must have exactly 4 parts
1486 }
1487
1488 void setPLCIPFromSerial(String ipString) {
1489     ipString.trim();
1490
1491     if (!isValidIP(ipString)) {
1492         Serial.println("ERROR:_Invalid_IP_address_format._Use_format:_
1493         XXX.XXX.XXX.XXX");
1494         Serial.println("Example:_SET_PLC_IP:192.168.1.100");
1495         return;
1496     }
1497
1498     // Parse and set the new IP
1499     int parts[4];
1500     int partIndex = 0;
1501     int lastDot = -1;
1502
1503     for (int i = 0; i <= ipString.length(); i++) {
1504         if (i == ipString.length() || ipString[i] == '.') {
1505             String part = ipString.substring(lastDot + 1, i);
1506             parts[partIndex++] = part.toInt();
1507             lastDot = i;
1508
1509             if (partIndex >= 4) break;
1510         }
1511     }
1512
1513     // Disconnect if currently connected
1514     if (isTcpConnected) {
1515         Serial.println("Disconnecting_from_current_PLC...");
1516         tcpClient.stop();
1517         isTcpConnected = false;
1518     }

```

```
1518
1519     // Update the IP
1520     plc_ip = IPAddress(parts[0], parts[1], parts[2], parts[3]);
1521
1522     // Save to preferences
1523     savePLCIPToPreferences();
1524
1525     Serial.print("PLC_IP_address_updated_to: ");
1526     Serial.println(plc_ip.toString());
1527     Serial.println("Will attempt to connect on next cycle...");
1528 }
```

APÊNDICE D – Código-Fonte do Bloco de Interpretação

Listing 5.2 – Código SFC/SCL do bloco de Interpretação

```

1  VAR_INPUT
2      Mode : Int;    // 1 = Modo Frequencia (com Start/Stop), 2 = Modo
    Direto
3      Start : Bool;
4      Stop : Bool;
5      CharArray : Array[0..99] of Char;
6  END_VAR
7
8  VAR_OUTPUT
9      MostFrequentClass : String;
10     ClassIndex : Int;
11     Confidence : Real;
12     SampleCount : Int;
13     IsValid : Bool;
14     RawDetection : String;
15     ActiveMode : Int;
16 END_VAR
17
18 VAR
19     DetectionHistory : Array[0..19] of String;
20     DetectionIndex : Int := 0;
21     ClassCount : Array[0..9] of Int;
22     ClassNames : Array[0..9] of String;
23     UniqueClasses : Int := 0;
24     AnalysisActive : Bool := FALSE;
25     LastDetection : String := '';
26     MaxCount : Int;
27     MaxIndex : Int;
28     i : Int;
29     j : Int;
30     ClassFound : Bool;
31     TotalSamples : Int;
32     startMarker : Int := -1;
33     endMarker : Int := -1;
34     Len : Int := 0;
35     DirectClassIndex : Int := -1;
36 END_VAR
37
38 VAR_TEMP
39     ExtractedLabel : String;
40     ColonPos : Int;
41     DetectionString : String;
42     ConfidenceStr : String;
43     ConfidenceValue : Real;

```

```

44     SecondColonPos : Int;
45     END_VAR
46
47 BEGIN
48     //
49     =====
50     // Atualiza modo ativo
51     //
52     =====
53     #ActiveMode := #Mode;
54     IF (#Mode <> 1) AND (#Mode <> 2) THEN
55         #ActiveMode := 1;
56     END_IF;
57     //
58     =====
59     // Extrao da String do Array de Caracteres
60     //
61     =====
62     #startMarker := -1;
63     #endMarker := -1;
64     #DetectionString := '';
65     FOR #i := 0 TO 99 DO
66         IF (#startMarker = -1) AND (#CharArray[#i] = '$02') THEN
67             #startMarker := #i;
68         END_IF;
69         IF (#startMarker <> -1) AND (#endMarker = -1) AND (#CharArray[#i] = '
$03') THEN
70             #endMarker := #i;
71         END_IF;
72     END_FOR;
73     IF (#startMarker <> -1) AND (#endMarker <> -1) AND (#endMarker > #
startMarker) THEN
74         #Len := #endMarker - #startMarker - 1;
75         FOR #j := (#startMarker + 1) TO (#endMarker - 1) DO
76             #DetectionString := CONCAT(IN1 := #DetectionString,
77                                     IN2 := CHAR_TO_STRING(#CharArray[#j]))
78         ;
79     END_FOR;
80 END_IF;

```

```

80
81 #RawDetection := #DetectionString;
82
83 //
=====
84 // MODO 2: TRANSMISSO DIRETA
85 //
=====

86 IF #ActiveMode = 2 THEN
87     #AnalysisActive := FALSE;
88     #SampleCount := 0;
89
90     IF (#DetectionString <> '') AND (#DetectionString <> 'NO_DETECTION')
    THEN
91         #ColonPos := FIND(IN1 := #DetectionString, IN2 := ':');
92
93         IF #ColonPos > 0 THEN
94             #ExtractedLabel := LEFT(IN := #DetectionString, L := #ColonPos
            - 1);
95             #ConfidenceStr := MID(IN := #DetectionString, L := 10, P := #
            ColonPos + 1);
96             #SecondColonPos := FIND(IN1 := #ConfidenceStr, IN2 := ':');
97
98             IF #SecondColonPos > 0 THEN
99                 #ConfidenceStr := LEFT(IN := #ConfidenceStr, L := #
            SecondColonPos - 1);
100             END_IF;
101
102             #ConfidenceValue := STRING_TO_REAL(#ConfidenceStr);
103         ELSE
104             #ExtractedLabel := #DetectionString;
105             #ConfidenceValue := 0.0;
106         END_IF;
107
108         #ClassFound := FALSE;
109         #DirectClassIndex := -1;
110
111         FOR #i := 0 TO 9 DO
112             IF #ClassNames[#i] = #ExtractedLabel THEN
113                 #DirectClassIndex := #i;
114                 #ClassFound := TRUE;
115                 EXIT;
116             END_IF;
117         END_FOR;
118

```

```

119     IF NOT #ClassFound THEN
120         FOR #i := 0 TO 9 DO
121             IF #ClassNames[#i] = '' THEN
122                 #ClassNames[#i] := #ExtractedLabel;
123                 #DirectClassIndex := #i;
124                 EXIT;
125             END_IF;
126         END_FOR;
127     END_IF;
128
129     #MostFrequentClass := #ExtractedLabel;
130     #ClassIndex := #DirectClassIndex;
131     #Confidence := #ConfidenceValue;
132     #IsValid := TRUE;
133
134 ELSE
135     #MostFrequentClass := '';
136     #ClassIndex := -1;
137     #Confidence := 0.0;
138     #IsValid := FALSE;
139 END_IF;
140
141 RETURN;
142 END_IF;
143
144 //
=====
145 // MODO 1: ANLISE POR FREQUENCIA
146 //
=====
147 IF #Stop THEN
148     #AnalysisActive := FALSE;
149     #DetectionIndex := 0;
150     #UniqueClasses := 0;
151     #SampleCount := 0;
152     #MostFrequentClass := '';
153     #ClassIndex := -1;
154     #Confidence := 0.0;
155     #IsValid := FALSE;
156     #LastDetection := '';
157
158     FOR #i := 0 TO 19 DO
159         #DetectionHistory[#i] := '';
160     END_FOR;
161

```

```

162     FOR #i := 0 TO 9 DO
163         #ClassCount[#i] := 0;
164         #ClassNames[#i] := '';
165     END_FOR;
166
167     RETURN;
168 END_IF;
169
170 IF #Start THEN
171     #AnalysisActive := TRUE;
172 END_IF;
173
174 IF #AnalysisActive AND (#DetectionString <> '') AND (#DetectionString <> '
    NO_DETECTION')
    AND (#DetectionString <> #LastDetection) THEN
175
176     #ColonPos := FIND(IN1 := #DetectionString, IN2 := ':');
177
178     IF #ColonPos > 0 THEN
179         #ExtractedLabel := LEFT(IN := #DetectionString, L := #ColonPos -
180 1);
181     ELSE
182         #ExtractedLabel := #DetectionString;
183     END_IF;
184
185     #DetectionHistory[#DetectionIndex] := #ExtractedLabel;
186     #DetectionIndex := #DetectionIndex + 1;
187     IF #DetectionIndex >= 20 THEN
188         #DetectionIndex := 0;
189     END_IF;
190
191     #LastDetection := #DetectionString;
192
193     FOR #i := 0 TO 9 DO
194         #ClassCount[#i] := 0;
195     END_FOR;
196
197     #UniqueClasses := 0;
198     #TotalSamples := 0;
199
200     FOR #i := 0 TO 19 DO
201         IF #DetectionHistory[#i] <> '' THEN
202             #TotalSamples := #TotalSamples + 1;
203             #ClassFound := FALSE;
204
205             FOR #j := 0 TO (#UniqueClasses - 1) DO
206                 IF #ClassNames[#j] = #DetectionHistory[#i] THEN

```



```

207         #ClassCount[#j] := #ClassCount[#j] + 1;
208         #ClassFound := TRUE;
209         EXIT;
210     END_IF;
211 END_FOR;
212
213     IF NOT #ClassFound AND (#UniqueClasses < 10) THEN
214         #ClassNames[#UniqueClasses] := #DetectionHistory[#i];
215         #ClassCount[#UniqueClasses] := 1;
216         #UniqueClasses := #UniqueClasses + 1;
217     END_IF;
218 END_IF;
219 END_FOR;
220
221 #MaxCount := 0;
222 #MaxIndex := -1;
223
224 FOR #i := 0 TO (#UniqueClasses - 1) DO
225     IF #ClassCount[#i] > #MaxCount THEN
226         #MaxCount := #ClassCount[#i];
227         #MaxIndex := #i;
228     END_IF;
229 END_FOR;
230
231 IF #MaxIndex >= 0 AND #TotalSamples > 0 THEN
232     #MostFrequentClass := #ClassNames[#MaxIndex];
233     #ClassIndex := #MaxIndex;
234     #Confidence := INT_TO_REAL(#MaxCount) / INT_TO_REAL(#TotalSamples)
235     * 100.0;
236     #SampleCount := #TotalSamples;
237     #IsValid := TRUE;
238 ELSE
239     #MostFrequentClass := '';
240     #ClassIndex := -1;
241     #Confidence := 0.0;
242     #SampleCount := 0;
243     #IsValid := FALSE;
244 END_IF;
245 END_IF;
246 END_FUNCTION_BLOCK

```