Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Mecatrónica

Trabalho de Projeto

Desenvolvimento de protocolo de comunicação entre PLCs através de microcontroladores

Nuno Miguel dos Santos Alas

Orientador(es) | João Manuel Figueiredo

Leonardo Andrade

Évora 2025

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Mecatrónica

Trabalho de Projeto

Desenvolvimento de protocolo de comunicação entre PLCs através de microcontroladores

Nuno Miguel dos Santos Alas

Orientador(es) | João Manuel Figueiredo

Leonardo Andrade

Évora 2025

O trabalho de projeto foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	Frederico José Grilo (Universidade de Évora)
Vogais	|	João Manuel Figueiredo (Universidade de Évora) (Orientador) Mouhaydine Tlemcani (Universidade de Évora) (Arguente)

Évora 2025

Agradecimentos

A elaboração deste protejo é o culminar de um longo caminho de aprendizagem e crescimento.

Nem sempre foi um caminho fácil ou linear, mas nunca foi solitário. Foram várias as pessoas que comigo percorreram este caminho, que me desafiaram a ir sempre mais além e ajudaram a manter o foco no objetivo a que me tinha proposto.

Chegou agora o momento de, a todas, expressar o meu sincero agradecimento.

Em primeiro lugar, quero expressar os meus sinceros agradecimentos ao Professor Doutor João Figueiredo, por me ter aceitado como seu orientando. Pelos valiosos ensinamentos que me transmitiu ao longo deste percurso académico, mas também pela orientação sempre que me afastava do objetivo e pela disponibilidade que sempre demonstrou. A sua experiência e conhecimento foram fundamentais para a conclusão deste trabalho.

Seguidamente quero endereçar um agradecimento ao meu coorientador, ao Professor Leonardo Andrade. Pelo apoio que me dispensou sempre que o procurei.

Por fim, quero ainda expressar um agradecimento à minha família e amigos por todo o apoio e acompanhamento nesta importante etapa da minha vida pessoal e académica.

Resumo

Este projeto apresenta o desenvolvimento de um protocolo de comunicação entre *Programmable Logic Controller* (PLC) Siemens utilizando microcontroladores ESP32 como interfaces de comunicação, implementando uma arquitetura *Internet of Things* (IoT) industrial completa. O sistema desenvolvido integra um PLC LOGO! e um PLC S7-314 através de um protocolo personalizado de sete bits e comunicação *Message Queing Telemetry Transport* (MQTT), permitindo a troca de dados em tempo real. De modo a testar a sua aplicabilidade em contexto pratico, o mesmo foi aplicado num sistema de visão. Sistema este que simula uma linha de controlo de qualidade de peças.

A solução implementada inclui um sistema de monitorização com um *dashboard web* responsivo, uma base de dados PostgreSQL para armazenamento histórico, e integração com sistema de visão computacional para controlo de qualidade automatizado. Os componentes principais incluem dois microcontroladores ESP32, dois PLC industriais, um *broker* MQTT Eclipse Mosquitto para comunicação, serviços *backend* para processamento de dados, e interfaces web para monitorização. O sistema de visão computacional, hospedado num Raspberry Pi, utiliza a biblioteca Snap7 para comunicação direta com o PLC, oferecendo inspeção dimensional automatizada. O sistema utiliza tecnologias modernas como FastAPI, React, Docker e OpenCV, possibilitando assim, uma plataforma escalável e modular.

A arquitetura desenvolvida demonstra a viabilidade de soluções IoT de baixo custo para modernização de sistemas industriais, oferecendo uma alternativa económica às soluções comerciais tradicionais. O protocolo de comunicação personalizado permite flexibilidade na integração de diferentes sistemas, contribuindo para a evolução da Indústria 4.0.

Palavras-chave: *PLC, ESP32, MQTT, IoT Industrial, Protocolo de Comunicação, Indústria 4.0, Visão Computacional*

Abstract

Development of a communication protocol between PLCs through microcontrollers

This project presents the development of a communication protocol between Siemens Programmable Logic Controllers (PLCs) using ESP32 microcontrollers as communication interfaces, implementing a complete industrial Internet of Things (IoT) architecture. The developed system integrates a LOGO! PLC and an S7-314 PLC through a custom seven-bit protocol and Message Queuing Telemetry Transport (MQTT) communication, enabling real-time data exchange. To test its applicability in a practical context, it was applied to a vision system. This system simulates a parts quality control line.

The implemented solution includes a monitoring system with a responsive web dashboard, a PostgreSQL database for historical storage, and integration with a computer vision system for automated quality control. The main components include two ESP32 microcontrollers, two industrial PLC, an Eclipse Mosquitto MQTT broker for communication, backend services for data processing, and web interfaces for monitoring. The computer vision system, hosted on a Raspberry Pi, uses the Snap7 library for direct communication with PLC, offering automated dimensional inspection. The system uses modern technologies such as FastAPI, React, Docker, and OpenCV, creating a scalable and modular platform.

The developed architecture shows the feasibility of low-cost IoT solutions for industrial system modernization, offering an economical alternative to traditional commercial solutions. The customized communication protocol allows flexibility in the integration of different systems, contributing to the evolution of Industry 4.0.

Keywords: *PLC, ESP32, MQTT, Industrial IoT, Communication Protocol, Industry 4.0, Computer Vision*

Lista de Figuras

Índice

Capítulo 1: Introdução

1.1 Introdução

Desde os primórdios da automação, quando máquinas comunicavam através de ligações diretas e simples, assistimos a uma transformação radical. Esta evolução insere-se no contexto da quarta revolução industrial, ou Indústria 4.0, conceito introduzido pelo governo alemão em janeiro de 2011 como uma "iniciativa estratégica" pela Aliança de Pesquisa em Ciência da Indústria (Andrade, L. H. S, 2021)[1]. O panorama atual caracteriza-se por infraestruturas de rede sofisticadas que interligam equipamentos dispersos por toda a instalação fabril. Protocolos como Profibus, Profinet e EtherNet/IP conquistaram uma fatia substancial do mercado industrial. Contudo, quando tentamos que equipamentos de marcas distintas trabalharem em harmonia, surgem obstáculos técnicos que persistem até hoje. Esta fragmentação tecnológica obriga frequentemente as empresas a adotar soluções proprietárias ou investir em *gateways* dispendiosos. Por outro lado, o mundo do IoT trouxe-nos alternativas interessantes. O MQTT destaca-se como uma dessas inovações, funcionando segundo um princípio elegante: alguns dispositivos publicam informação enquanto outros se subscrevem para a receber. Esta abordagem revelou-se particularmente valiosa em ambientes industriais onde a informação precisa de fluir entre múltiplos pontos. O que torna o MQTT especialmente atrativo é a sua leveza computacional. Enquanto outros protocolos exigem processadores robustos e memória abundante, este protocolo consegue operar eficazmente mesmo em microcontroladores modestos, abrindo possibilidades para soluções de baixo custo, mas elevada funcionalidade. Esta característica é particularmente relevante no contexto dos Sistemas Ciber-Físicos, onde "a maioria desses sistemas embebidos são atualmente 'caixas' fechadas que não expõem a capacidade de computação ao exterior" (Andrade, L. H. S, 2021, p. 7) [1], sendo necessário maximizar a eficiência dos recursos disponíveis.

1.2 Contextualização

As indústrias de hoje enfrentam um dilema complexo: como conseguir colocar equipamentos antigos a "conversarem" com tecnologia de ponta? Esta questão tornou-se central no processo de digitalização industrial, especialmente quando tentamos unir controladores de diferentes origens numa única rede coesa. Portugal e a Europa dependem fortemente dos sistemas Siemens, as famílias LOGO! e S7 dominam fabricas por todo o continente. Mas aqui surge o problema. Quando queremos que estes equipamentos trabalhem em conjunto para criar sistemas de monitorização avançados, as barreiras técnicas multiplicam-se. O mercado atual oferece soluções, mas de custo elevado. Não falando apenas do investimento inicial em hardware, há também os custos recorrentes de licenciamento de software proprietário. Por vezes, estas soluções raramente se adaptam às necessidades específicas de cada instalação. Felizmente, a tecnologia não pára. Microprocessadores como o ESP32 revolucionaram o panorama, oferecem poder de processamento impressionante a preços acessíveis, Wi-Fi incorporado e capacidade para dialogar com múltiplos protocolos industriais. Esta convergência entre IoT e automação está a abrir portas que antes pareciam fechadas, permitindo criar pontes de comunicação personalizadas e economicamente viáveis entre equipamentos industriais de qualquer idade ou procedência.

1.3 Motivação e objetivos

O presente projeto surge da necessidade de desenvolver uma solução viável e prática para permitir a comunicação sem fio entre dois PLC. Uma das dificuldades que se encontrou no laboratório de automação industrial da universidade de Évora, foi a grande quantidade de fios entrelaçados e às vezes não identificados, o que tornava a programação dos PLC mais desafiante. Durante a Unidade Curricular de Programação e Sistemas Inteligentes, houve a oportunidade de conhecer o microcontrolador ESP32 e explorar as suas várias funcionalidades, das quais, entradas e saídas digitais, bem como as suas capacidades de conectividade Wi-Fi. Esta solução visa não apenas facilitar a interoperabilidade, mas também servir de base para sistemas descentralizados de controlo e monitorização industrial, com potencial para integração com *dashboards* em tempo real com armazenamento numa base de dados.

O objetivo principal deste projeto é o desenvolvimento de um protocolo de comunicação personalizado entre PLC Siemens utilizando microcontroladores ESP32 como interfaces de comunicação. Criando uma solução modular, escalável e económica para integração de sistemas de automação industrial com aplicação prática numa linha simulada de controlo de qualidade através de um sistema de captura e processamento de imagem.

Capítulo 2: Conceptualização

2.1 Introdução

Este capítulo apresenta a estrutura de conceptualização e design do projeto, seguida de uma descrição sistemática da arquitetura de hardware e ambiente de software utilizado ao longo do seu desenvolvimento.

2.2 Design do projeto

A arquitetura do sistema desenvolvido, ilustrada na Figura 1, apresenta uma solução integrada que interliga componentes de automação industrial tradicional com tecnologias contemporâneas de IoT. O design conceptual organiza-se em quatro partes distintas: física, comunicação, processamento, apresentação.

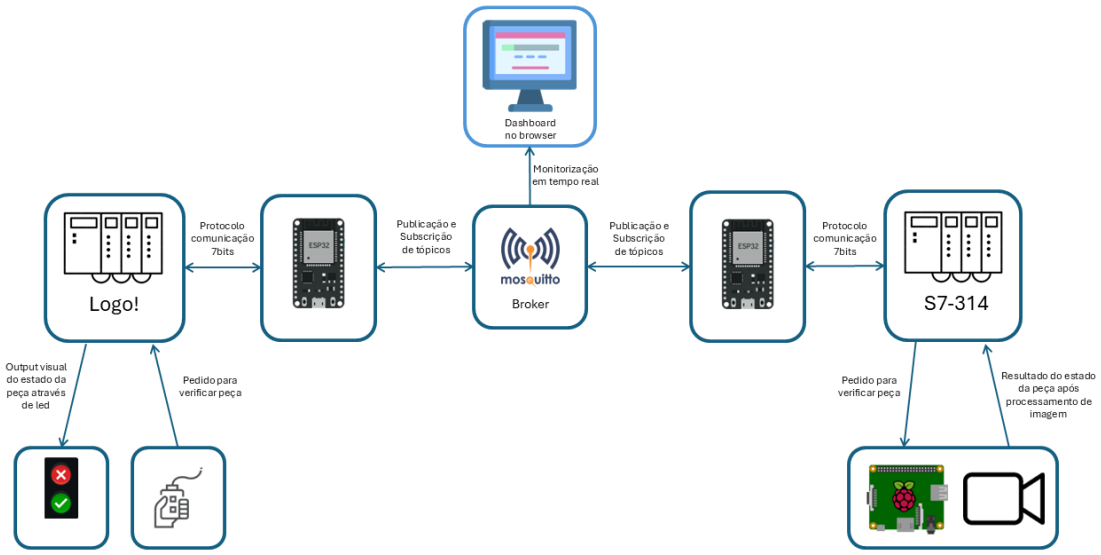


Figura 1 Conceito do projeto

A parte física é constituída por dois PLC Siemens, PLC LOGO! 24CE e um PLC S7-314. Estes dispositivos representam diferentes escalas de aplicação industrial, sendo o LOGO! orientado para tarefas de automação simples e o S7-314 destinado a aplicações de maior complexidade. A escolha destes dois PLCs distintos permite demonstrar a versatilidade do protocolo desenvolvido na integração de sistemas heterogéneos.

A comunicação é implementada através de dois ESP32, cada um associado a um PLC. Os EPS32 atuam como *gateways* de comunicação, realizando a conversão de sinais digitais entre os níveis lógicos dos PLCs (24V DC) e os níveis dos microcontroladores(3.3V). Esta conversão é possível devido à utilização de ULN2003A. Os ESP32 estabelecem uma rede Wi-Fi, eliminando a necessidade de cabos entre os PLCs. A comunicação entre os microcontroladores é gerida através do protocolo MQTT. Um broker MQTT, implementado através do software Eclipse Mosquitto, coordena a troca de mensagens entre os diversos componentes do sistema, funcionando como intermediário central que recebe, filtra e distribui as mensagens publicadas pelos diferentes clientes.

O processamento integra um Raspberry Pi 3 Model B, responsável pela implementação do sistema de visão computacional. Este executa algoritmos de processamento de imagem desenvolvidos com a biblioteca OpenCV, realizando inspeção dimensional de peças. A comunicação entre o Raspberry Pi e o PLC S7-314 é estabelecida através da biblioteca Snap7, que implementa o protocolo de comunicação S7 da Siemens, permitindo leitura e escrita direta na memória do PLC.

A apresentação é realizada através de uma aplicação web desenvolvida, composta por microsserviços containerizados através do Docker. Esta aplicação divide-se em três componentes. Um *frontend* desenvolvido em React para visualização de dados em tempo real, um backend API implementado em FastAPI para gestão de requisitos HTTP, e um backend MQTT que subscreve aos tópicos e insere os dados numa base de dados PostgreSQL.

2.3 Hardware

2.3.1 CPU S7-314 com CP 343 - Lean, SM321 DI 32xDC24V e SM322 DO 32xDC24V/0.5A

Este conjunto representa uma solução robusta da Siemens para automação industrial de média complexidade. A CPU S7-314[1] oferece capacidade de processamento adequada para aplicações que exigem controlo preciso e confiável. O módulo de comunicação CP 343-Lean[3] permite conectividade Ethernet, essencial para integração em redes industriais modernas. O sistema é complementado por módulos de entrada SM321[4], que conseguem processar 32 sinais digitais a 24V DC, e módulos de saída SM322[5], capazes de fornecer 32 saídas digitais com corrente máxima de 0,5A por canal. Esta configuração torna-se ideal para aplicações que necessitam de múltiplas entradas e saídas digitais, por exemplo: linhas de produção, sistemas de transporte ou equipamentos de empacotamento. Neste projeto, esta rack S7-314, foi utilizada para processamento de dados.

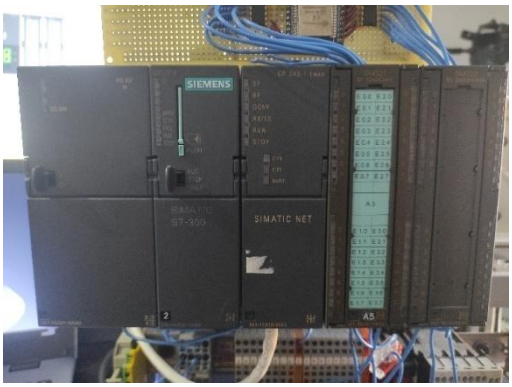


Figura 2 Rack CPU S7-314

2.3.2 LOGO! 24CE

O LOGO! 24CE[6] é um controlador lógico compacto da Siemens, concebido para aplicações de pequena escala que não justificam um PLC tradicional. Alimentado a 24V DC, integra entradas e saídas digitais num formato extremamente compacto, com programação através de software gráfico intuitivo. Este dispositivo é particularmente útil em aplicações simples tais como controlo de iluminação, sistemas de ventilação básicos, ou automatismos de portões. A sua simplicidade de instalação e programação, combinada com a fiabilidade típica dos produtos Siemens, torna-o ideal para pequenos projetos de automação ou como complemento a sistemas maiores para funções auxiliares específicas.

Neste projeto, o LOGO! é utilizado para requisição de imagens, através de um interruptor, e para output do estado da peça após o processamento de imagem, através de um led tricolor.

Figura 3 LOGO! 24CE

2.3.3 ESP32-WROOM-32

O ESP32-WROOM-32[7] é um microcontrolador de baixo custo, desenvolvido pela Espressif Systems. Ele integra um processador dual-core, conectividade Wi-Fi e Bluetooth, o que o faz ser para muitos a escolha ideal para projetos de IoT e automação. As suas capacidades incluem múltiplas interfaces de comunicação (SPI, I2C, UART), conversores analógico-digital e suporte para vários protocolos de rede. No contexto industrial, este dispositivo serve como elemento de ligação entre os equipamentos tradicionais e os modernos sistemas de monitorização, permitindo a implementação de soluções de conectividade personalizadas a custos reduzidos. Neste projeto, os ESP32 são utilizados como interface de comunicação entre os PLCs e para monitorização de dados em tempo real.

Figura 4 ESP32-WROOM-32 Dev Board

2.3.4 ULN2003A

O ULN2003A[8] é um circuito integrado que atua como uma interface entre sistemas de baixa potência e cargas que exigem correntes mais altas. Ele consiste em sete pares Darlington com diodos de proteção integrados, permitindo controlar cargas indutivas, como relés, motores de passo ou solenoides. Cada canal pode suportar até 500mA de corrente contínua, tornando-se essencial quando microcontroladores como o ESP32 precisam acionar dispositivos que consomem mais energia do que as suas saídas podem fornecer. Na automação industrial, este componente é frequentemente usado como um driver para motores de passo, uma interface para relés de potência, ou para acionar indicadores de luz de alta corrente. Neste projeto, o ULN2003a é utilizado com o um deslocador de tensão e energia entre os microcontroladores ESP32 de baixa potência e sinais/cargas de nível industrial.

Figura 5 ULN2003A

2.3.5 Raspberry Pi 3 Model B

O Raspberry Pi 3 Model B[9] é um computador de placa única que oferece capacidades de processamento superiores aos microcontroladores tradicionais, utilizando um sistema operacional totalmente baseado em Linux. Equipado com processador quad-core ARM Cortex-A53, 1GB de RAM, conectividade Wi-Fi, Bluetooth e Ethernet, fornece uma plataforma versátil para aplicações que exigem maior poder de computação. Na automação industrial, muitas vezes serve como um *gateway* de comunicação, servidor local para interfaces web ou sistema de aquisição e análise de dados. As suas múltiplas portas GPIO permitem interface direta com sensores e atuadores, enquanto a capacidade de executar linguagens de programação como Python, Node.js ou C++ oferece flexibilidade no desenvolvimento de soluções personalizadas. É particularmente valioso em projetos que requerem processamento de imagem, base de dados local, ou interfaces gráficas avançadas. Neste projeto, o Raspberry Pi foi utilizado para processamento de imagem e comunicação com o PLC S7-314.

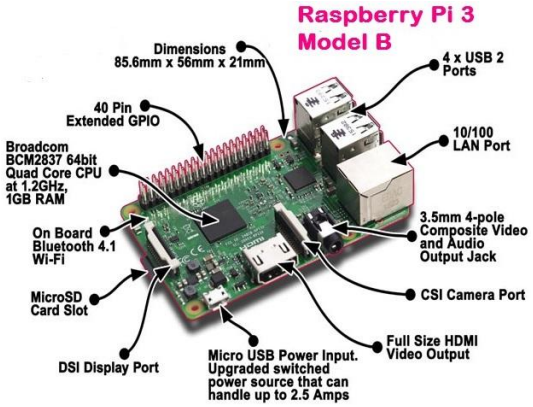


Figura 6 Raspberry Pi 3 Model B

2.4 Software

2.4.1 Totally Integrated Automation Portal v.16

Esta é a principal plataforma de software de engenharia da Siemens que serve como um ambiente de desenvolvimento abrangente para projetos de automação industrial. O *Portal de Automação Totalmente Integrado* (TIA Portal)[10] integra várias disciplinas de engenharia numa única interface, permitindo que os engenheiros configurem hardware, programem PLC (utilizando o STEP 7), projetem interfaces homem-máquina (WinCC), configurem sistemas de segurança (Safety Integrated) e gerenciem aplicações de controlo de movimento. O software suporta todo o ciclo de vida da automação, desde o projeto inicial até o comissionamento e manutenção. É particularmente valioso para projetos industriais de grande escala, onde vários componentes de automação precisam trabalhar juntos sem problemas.

2.4.2 LOGO!Soft Comfort v8.0[11]

Este software de programação é projetado para Siemens LOGO!, que são controladores lógicos programáveis compactos usados em pequenas aplicações de automação. O software tem uma interface de programação gráfica intuitiva que usa blocos de funções em vez da lógica ladder tradicional, tornando-o acessível a técnicos e engenheiros que podem não ter uma vasta experiência em programação PLC. O software inclui recursos de simulação, permitindo que os utilizadores testem os seus programas antes de enviá-los para o módulo. É frequentemente usado na automação de edifícios, controlo de pequenas máquinas e ambientes educacionais.

2.4.3 Visual Studio Code

Editor gratuito de código aberto da Microsoft que se tornou uma das ferramentas de desenvolvimento mais populares na comunidade de programação. O VS Code[12] oferece um ambiente leve e rico em recursos com realce de sintaxe, autocompletar código inteligente (IntelliSense), suporte a *debug* e um terminal integrado. A sua extensibilidade num vasto mercado de extensões torna-o adequado para praticamente qualquer linguagem de programação ou estrutura de desenvolvimento. O editor inclui integração Git, edição de vários cursores e temas personalizáveis. A sua disponibilidade multiplataforma (Windows, macOS, Linux) e comunidade de desenvolvimento ativa tornaram-no numa escolha ideal para programadores web, cientistas de dados e engenheiros de software em vários domínios.

2.4.4 Thonny

Um IDE Python projetado com iniciantes e ambientes educacionais em mente. Thonny[13] fornece uma interface limpa e organizada que se concentra em ajudar os utilizadores a entender como o código Python é executado. O IDE inclui um interpretador Python integrado, uma ferramenta de *debug* simples e um inspetor de variáveis. Também inclui ferramentas que tornam o processo de programar microcontroladores, que tenham instalados como seu firmware MicroPython, muito mais simples.

2.4.5 Ubuntu Desktop

Uma das distribuições Linux mais populares em todo o mundo. Baseado no Debian, o Ubuntu[14] fornece um sistema operacional seguro e de código aberto com o ambiente de desktop GNOME. A distribuição inclui um repositório de software abrangente com milhares de aplicações, excelente compatibilidade de hardware e um forte suporte da comunidade. Ubuntu é amplamente utilizado em ambientes de desenvolvimento, servidores e como uma alternativa a sistemas operativos convencionais como Windows ou macOS.

2.4.6 Raspberry Pi OS

O sistema operacional[15] oficial para computadores de placa única Raspberry Pi, baseado no Debian Linux, mas otimizado para a arquitetura ARM do Pi e recursos limitados. O sistema operacional inclui um ambiente de desktop com aplicações essenciais, ferramentas de programação e software educacional. Ele foi projetado para ser leve e funcional, apoiando o papel do Pi na educação, projetos de hobby e aplicações de IoT. O sistema inclui bibliotecas GPIO para interface de hardware, suporte para câmara e recursos de rede.

2.4.7 Docker

Uma plataforma de conteinerização que transformou a maneira como as aplicações são desenvolvidas, implantadas e gerenciadas. O Docker[16] permite que os programadores “empacotem” as aplicações juntamente com as suas dependências, bibliotecas e arquivos de configuração em containers leves e portáteis. Esses containers podem ser executados em diferentes ambientes, desde portáteis de desenvolvimento a servidores de produção, resolvendo o problema "Mas no meu computador funciona…". Tornou-se essencial para práticas modernas de DevOps, arquitetura de microsserviços e desenvolvimento nativo na *cloud*, permitindo ciclos de implementação mais rápidos e utilização mais eficiente de recursos.

2.4.8 Python

Uma linguagem de programação de alto nível, reconhecida pela sua legibilidade e simplicidade do código. A sintaxe do Python[17] é projetada para ser intuitiva e próxima da linguagem natural, tornando-a numa escolha excelente para iniciantes, mas permanecendo poderosa o suficiente para aplicações complexas. A extensa biblioteca padrão do Python e o vasto ecossistema de pacotes de terceiros tornam-no adequado para uma variedade de aplicações, incluindo desenvolvimento *web* (Django, Flask), ciência de dados (pandas, NumPy), *machine learning* (scikit-learn, TensorFlow), *scripts* de automação e computação científica.

2.4.9 JavaScript

No desenvolvimento *web*, JavaScript[18] permite interfaces de utilizador interativas, atualizações de conteúdo dinâmico e aplicações complexas do lado do cliente. Com a introdução do Node.js, o JavaScript expandiu-se para o desenvolvimento do lado do servidor, permitindo o desenvolvimento *full-stack* com uma única linguagem. A linguagem suporta estilos de programação funcionais e orientados a objetos, apresenta digitação dinâmica e inclui recursos modernos, como *async/await* para lidar com operações assíncronas. A natureza orientada a eventos do JavaScript e o extenso ecossistema de bibliotecas e *frameworks* (React, Vue, Angular) fizeram dele uma das linguagens de programação mais utilizadas no mundo.

2.4.10 Vite

Uma ferramenta de construção moderna que representa um avanço significativo nas ferramentas de desenvolvimento *web*, criadas pela equipe de Vue.js, mas agnóstica ao *framework*. O Vite[19] suporta TypeScript pronto para utilização, inclui um sistema de plug-ins para estender a funcionalidade e funciona perfeitamente com estruturas populares como React, Vue e Svelte. A sua velocidade e melhorias na experiência do programador tornam-no cada vez mais popular para o desenvolvimento de aplicações web modernas.

2.4.11 KiCad 8.0

Software de automação de projeto eletrónico de código aberto (EDA) que fornece ferramentas de nível profissional para projetar circuitos eletrónicos e placas de circuito impresso. O KiCad[20] inclui um editor esquemático para criar diagramas de circuitos, um editor de layout PCB para projetar layouts de placas, um visualizador 3D para visualizar placas acabadas e ferramentas para gerar arquivos de fabricação. O software suporta várias bibliotecas de componentes, criação de *footprints* personalizadas e verificação de regras de design para garantir a viabilidade de fabricação.

2.4.12 MQTT

O MQTT[20] (*Message Queuing Telemetry Transport*) é um protocolo de mensagens leve, de publicação-subscrição, projetado para dispositivos restritos e redes de baixa largura de banda, alta latência ou não confiáveis. Originalmente desenvolvido pela IBM em 1999, tornou-se um protocolo padrão para aplicações de IoT. O protocolo funciona com um princípio simples: um *broker* central atua como um centro de mensagens, enquanto os clientes podem publicar mensagens para tópicos específicos ou se subscreverem em tópicos para receber mensagens. Isso torna possível distinguir entre emissores e recetores, tornando o sistema altamente escalável e flexível.

Capítulo 3: Implementação

3.1 Introdução

O presente capítulo aborda a implementação técnica do projeto. Os processos de configuração de hardware e desenvolvimento de *software* são detalhados, englobando tanto a montagem física de componentes quanto a implementação algorítmica.

3.2 Protocolo de Comunicação

O protocolo de comunicação desenvolvido baseia-se numa estrutura de sete bits que codifica informações de estado e comandos entre os PLCs através dos microcontroladores ESP32. Esta abordagem minimalista foi concebida para otimizar a utilização dos recursos limitados dos microcontroladores, mantendo simultaneamente a clareza e robustez necessárias para aplicações industriais.

A figura 7 ilustra o protótipo físico desenvolvido, onde o ESP32 é montado numa placa de desenvolvimento que integra o circuito integrado ULN2003A. Este componente é essencial para a arquitetura do sistema, funcionando como interface de potência entre os sinais do ESP32 e os sinais industriais dos PLCs. A escolha de sete bits deve-se a esses mesmos sete pares Darlington que constituem o integrado.

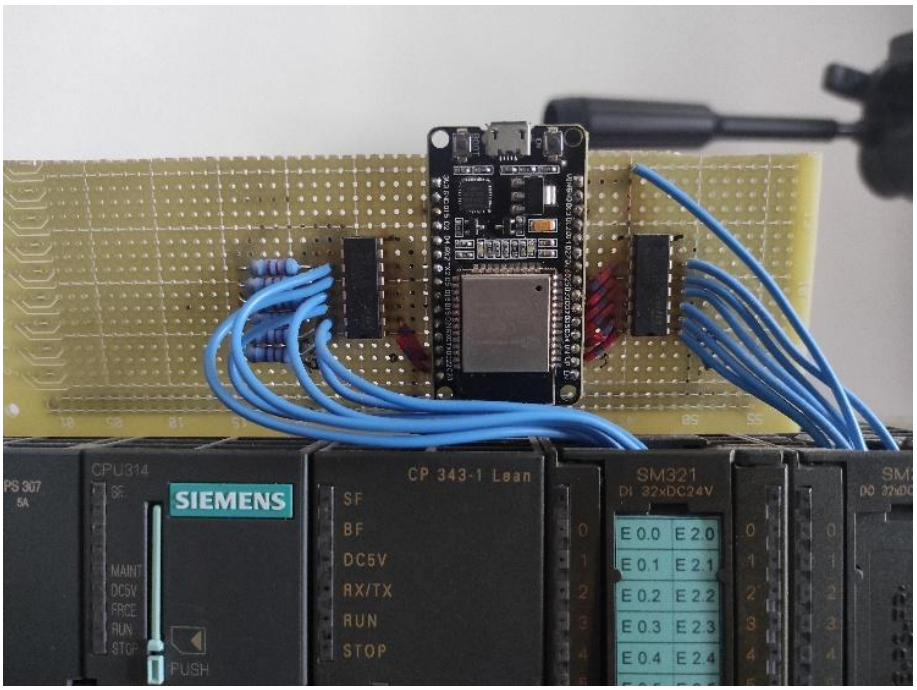


Figura 7 Protótipo da placa ESP32 com ULN2003a

A figura 8 demonstra o esquema elétrico do protótipo acima apresentado.

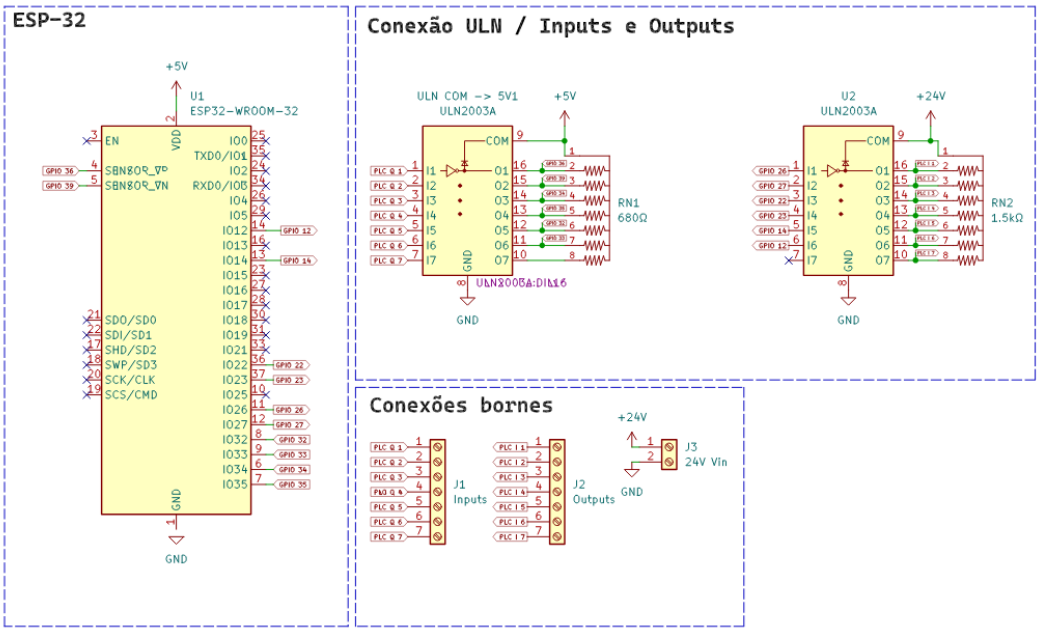


Figura 8 Esquema elétrico do protótipo

3.3 Contextualização Prática

De modo a testar este protocolo, implementou se o mesmo num ambiente simulado de controlo de qualidade que integra os diversos componentes descritos na arquitetura conceptual. Este cenário de aplicação demonstra a viabilidade do protocolo desenvolvido num contexto que replica condições operacionais industriais reais.

O processo inicia-se no PLC LOGO!. Quando o utilizador aciona este interruptor, o LOGO! ativa uma saída digital que é monitorizada pelo ESP32 associado. Esta ação desencadeia uma sequência de eventos coordenados através do protocolo de comunicação desenvolvido. O ESP32 ligado ao LOGO! deteta o flanco negativo no pino de entrada correspondente, identifica o padrão dentro do protocolo e publica uma mensagem MQTT direcionada ao ESP32 associado ao PLC S7-314.

O segundo microcontrolador, ao receber a mensagem através da subscrição MQTT, ativa uma saída digital que aciona uma entrada no PLC S7-314. Este PLC, interpreta o sinal como comando para iniciar o processo de inspeção de uma peça. O PLC S7-314 comunica então com o Raspberry Pi através do protocolo S7, utilizando a biblioteca Snap7 para estabelecer uma conexão direta via Ethernet. A instrução específica

a escrita numa posição de memória do PLC sinaliza ao sistema de visão computacional que uma nova peça está pronta para inspeção.

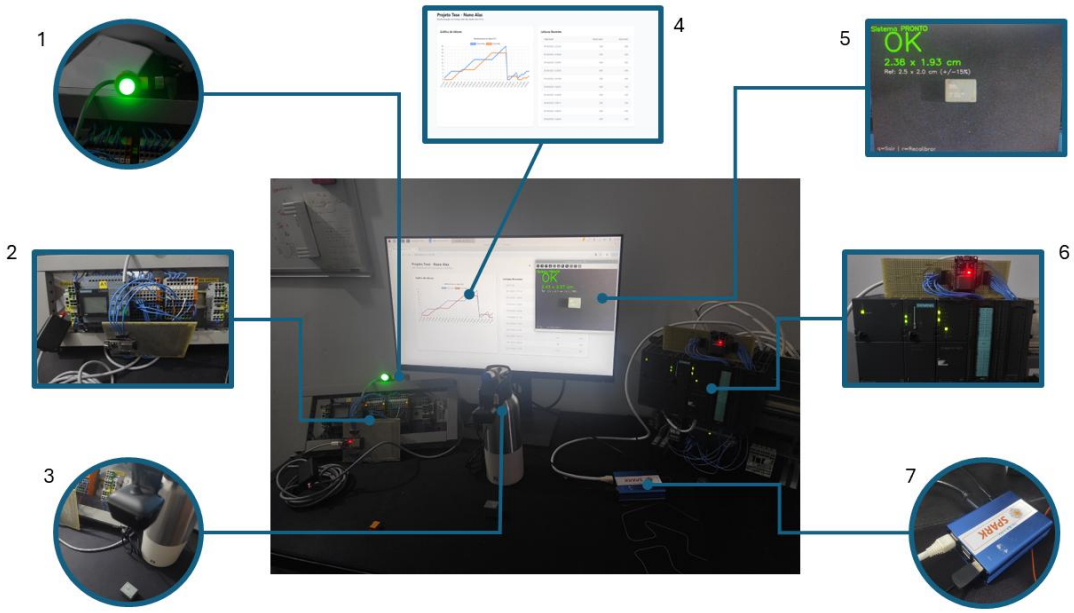
O Raspberry Pi, monitoriza continuamente a memória do PLC através de leituras cíclicas. Quando deteta a *flag* de solicitação, o sistema de visão ativa a câmara USB conectada e captura uma imagem da peça posicionada no campo de visão. O processamento desta imagem é realizado através de algoritmos implementados com OpenCV , biblioteca cv2, que executam uma sequência de operações de processamento digital de imagem: conversão para escala de cinzentos, aplicação de filtros para redução de ruído, segmentação através de técnicas de *thresholding*, deteção de contornos e cálculo de dimensões físicas através de calibração previamente estabelecida.

As dimensões medidas são comparadas com especificações de tolerância predefinidas. Se as medições se enquadram dentro dos limites aceitáveis, o resultado "OK" é escrito numa memória do PLC S7-314. Caso contrário, o resultado "NOK" é registado numa outra memoria do mesmo PLC. O PLC S7-314, ao ler este resultado da sua memória, ativa padrões específicos de saída de acordo com o protocolo de sete bits desenvolvido. O ESP32 associado deteta estes padrões, incrementa os contadores apropriados e publica as informações atualizadas via MQTT.

Paralelamente, o ESP associado ao PLC S7-314 envia também um sinal de retorno ao ESP associado ao PLC LOGO!, indicando o resultado da inspeção. O PLC LOGO! ativa então um LED tricolor conectado às suas saídas: verde para peças OK, vermelho para NOK.

No decorrer deste processo, está um cliente MQTT a escutar os resultados das inspeções, colocando-os numa base de dados onde estes são demonstrados ao utilizador, em tempo real, através de um *dashboard*.

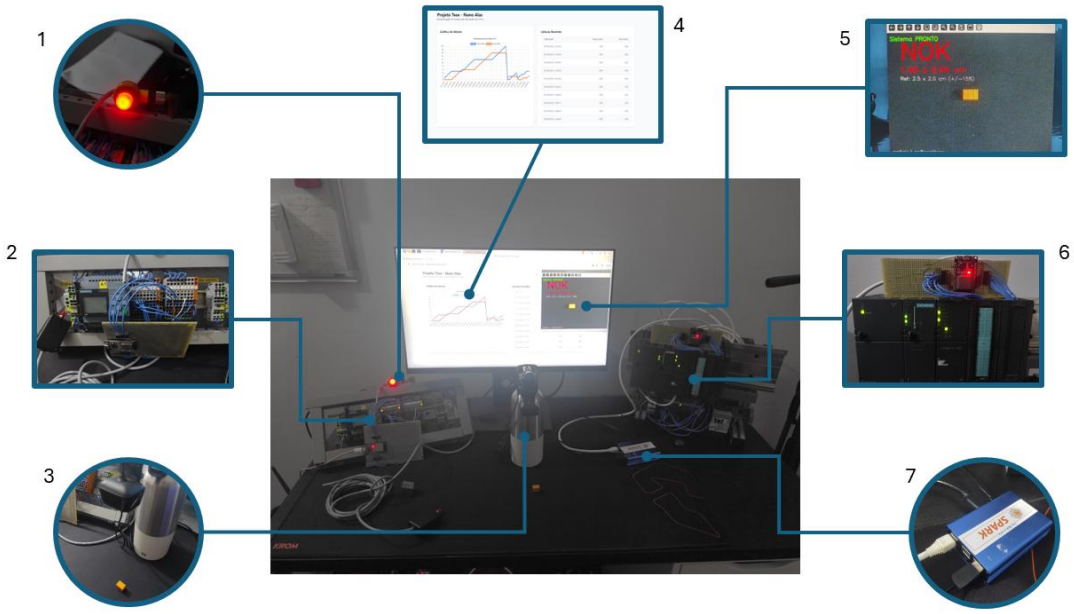
As Figuras 10 e 11 documentam a implementação física do sistema em funcionamento. Na Figura 10, observa-se uma situação onde a peça inspecionada apresenta dimensões dentro das especificações, resultando na ativação do LED verde no painel do LOGO! e o incremento do contador de peças OK visível no *dashboard*.



1. LED de cor Verde a indicar peça OK
2. LOGO!24CE com ESP32 e o interruptor
3. WebCam apontada para a peça
4. Dashboard a demonstrar os dados em tempo real
5. Interface do programa de processamento de imagem com o resultado OK e respetivas medidas da peça
6. S7-314 com ESP32
7. Raspberry Pi

Figura 9Projeto real com resultado "OK"

A Figura 11 ilustra o cenário oposto, onde dimensões fora de tolerância resultam em LED vermelho e registo como peça NOK.



1. LED de cor Vermelha a indicar peça NOK
2. LOGO!24CE com ESP32 e o interruptor
3. WebCam apontada para a peça
4. Dashboard a demonstrar os dados em tempo real
5. Interface do programa de processamento de imagem com o resultado NOK e respetivas medidas da peça
6. S7-314 com ESP32
7. Raspberry Pi

Figura 10 Projeto real com resultado "NOK"

Para a concretização deste projeto, recorreu-se aos seguintes recursos:

3.3.1 ESP32

Ambos os ESP32 atuam como monitores e detetores de eventos, comunicando entro si os dados através da publicação e subscrição dos mesmos enquanto tópicos de MQTT.

3.3.1.1 Biblioteca umqtt

A biblioteca umqtt[22] é uma implementação leve do protocolo MQTT para micropython. Responsável pela criação de tópicos, clientes e conexão com o broker.

Classe e Métodos utilizados:

- Classe MQTTClient(client_id, server, port, user, password), para construir o cliente;
- .set_callback(callback_function) Define a função callback que é executada quando uma mensagem é recebida;
- .connect() para conectar ao broker;
- .subscribe(topic) para subscrever ao tópico;
- .publish(topic, msg) para publicar a mensagem ao tópico definido;
- .check_msg() para verificar se há mensagens pendentes no broker.

3.3.1.2 Biblioteca json

Biblioteca padrão Python para codificação e descodificação de dados no formato JSON (JavaScript Object Notation). JSON é o formato de dados mais utilizado em APIs web e sistemas IoT devido à sua legibilidade e suporte universal.

Métodos utilizados:

- json.dumps(obj), converte um objeto Python numa string JSON.

3.3.1.3 Main do ESP32

O loop de execução principal une tudo. Depois de estabelecer a conexão MQTT, entra num loop infinito que verifica as mensagens recebidas, deteta alterações nos flancos negativos dos pinos de entrada e publica atualizações dos dados. O intervalo de suspensão de 50 milissegundos equilibra a capacidade de resposta com a eficiência do processador. Esse tempo evita a falta de alterações rápidas de entrada, evitando o uso excessivo da CPU.

Funções:

- mqtt_callback(topic, msg), esta função é executada sempre que uma mensagem chega aos tópicos subscritos. Ela descodifica a mensagem recebida e a compara-a com o protocolo de comunicação.
- connect_mqtt(), estabelece a conexão com o broker MQTT. Inicia a conexão e subscreve o tópico 'esp32_LOGO!/'. A função devolve o

objeto cliente , sendo este utilizado noutras partes do programa para publicar dados. A subscrição permite que este ESP32 receba dados do outro dispositivo.

- detect_falling_edges(), monitoriza todos os seis pinos de entrada continuamente. Compara os estados de pinos atuais com valores armazenados anteriormente para identificar transições de flanco negativo (alterações de *HIGH* para *LOW*). Em sistemas lógicos negativos com matrizes de Darlington, estes flancos representam sinais ativos. A função devolve dois valores: uma lista de índices de pinos onde os flancos ocorreram e a matriz completa de estados atuais. Após o processamento, ele atualiza a variável previous_states para se preparar para o próximo ciclo de verificação.
- publicar_contadores(client, falling_edges, current_states), só processa dados quando pelo menos um flanco negativo nos pins de entrada é detetado, evitando tráfego MQTT desnecessário. Os estados de pinos atuais são convertidos numa string separada por vírgulas e comparados com os padrões do protocolo. Quando o padrão OK aparece, ele incrementa o contador OK e publica o padrão para 'esp32_s7_314/'. O padrão NOK incrementa o contador NOK de forma semelhante. Se o padrão RESET_DIA for detetado, ambos os contadores retornarão a zero. Depois de atualizar os contadores, a função constrói um objeto JSON contendo ambos os contadores (pecas_boas e pecas_mas) e publica-o no tópico 'plc/data'.

3.3.2 Dashboard

O *dashboard* de monitorização, ilustrado na Figura 11, apresenta em tempo real os resultados do sistema de inspeção de qualidade implementado. Os valores correspondentes às peças OK são representados pela métrica "Peças Boas", enquanto as peças NOK são categorizadas como "Peças Más". Para além destas duas variáveis, o sistema regista e apresenta o *timestamp* associado a cada aquisição de dados.

A interface encontra-se dividida em dois componentes principais de visualização. No lado esquerdo, um gráfico de linhas temporais que apresenta a evolução das métricas,

utilizando a cor azul para representar as "Peças Boas" e a cor laranja para as "Peças Más". Esta representação gráfica permite uma análise visual intuitiva da tendência da qualidade e quantidade ao longo do tempo, facilitando a identificação imediata de padrões anómalos ou variações no desempenho do processo. No lado direito, uma tabela de dados apresenta o histórico cronológico das leituras mais recentes, organizadas por *timestamp*.

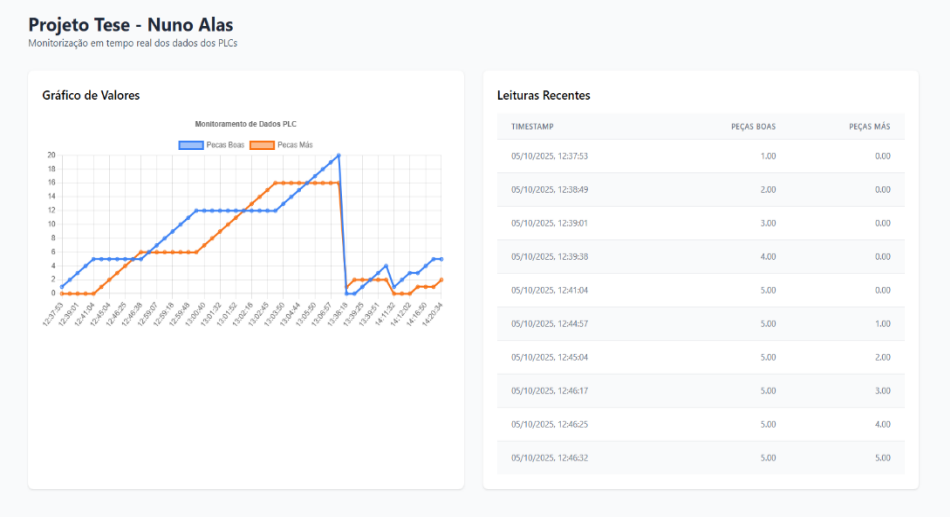


Figura 11Dashboard para monitorização em tempo real

3.3.2.1 Docker-compose[23]

Uma aplicação *web* é composta por um *stack,* e esse *stack* geralmente é composto por conjunto de serviços, *fronted*, *backend* e uma base de dados. O ficheiro docker-compose.yml contem a configuração desse mesmo *stack.* Para alem do stack, também foi criado um serviço para o broker MQTT.

Serviços:

- Base de dados: PostgreSQL[24];
- MQTT Broker: Eclipse-Mosquitto;
- Backend API: na diretoria ./backend-api contem um container com uma imagem de Python versão 3.9 – slim;
- Backend MQTT: na diretoria ./backend-mqtt contem um container com uma imagem de Python versão 3.9 – slim;

- Frontend: na diretoria ./frontend contem um container com uma imagem de Node.js versão 18 – alpine.

O ficheiro docker-compose também permite com que estes microsserviços comuniquem entre si , configurando as portas de comunicação de cada um.

3.3.2.2 Backend API

Esta aplicação Python representa um serviço de *backend* projetado para monitorização em tempo real e recolha de dados. A implementação utiliza FastAPI[25] como *framework*, criando uma REST API que facilita o armazenamento e recuperação de dados.

3.3.2.2.1 Framework Fastapi

API Endpoints:

- GET/ - Verifica o estado da aplicação. Devolve uma simples mensagem em JSON a confirmar que a API está operacional;
- GET/health - Verifica o estado da conexão entre a aplicação e a base de dados;
- GET/data - Consulta a base de dados;
- POST/data - Insere novos dados na base de dados.

3.3.2.2.2 Biblioteca psycopg2[26]

Esta biblioteca fornece a interface Python-PostgreSQL, essencialmente traduzindo instruções Python em operações de base de dados.

Métodos utilizados:

- .connect(dbname, user, password, host, port), estabelece a conexão à base de dados.
- .cursor(), cria um objeto para executar as instruções em SQL.
- .execute(), executa instruções em SQL

- .commit(), finaliza as transições, fazendo as alterações ficarem permanentes na base de dados.
- .close(), encerra a conexão com a base de dados.

3.3.2.2.3 Biblioteca uvicorn

Uvicorn[27] é uma interface assíncrona em Python para servidores web, frameworks e aplicações que suportam comunicação em tempo real. Utilizado para direcionar os pedidos HTTP aos *endpoints* da *framework*.

Método utilizado:

- uvicorn.run(app, host, port), inicia o servidor.

3.3.2.3 Backend MQTT

Esta aplicação Python serve como intermediário que faz a ponte entre as mensagens MQTT e a base de dados. O seu principal objetivo é recolher os dados em tempo real e colocá-los na base de dados. A implementação segue uma arquitetura orientada a eventos, respondendo a mensagens MQTT recebidas e armazenando os dados recebidos sem exigir sondagem contínua. Esta abordagem é particularmente eficiente para cenários de automação industrial em que os dados chegam em intervalos irregulares.

3.3.2.3.1 Biblioteca paho.mqtt.client

Paho MQTT[28] é a biblioteca oficial do Python para o protocolo MQTT, mantida pela Eclipse Foundation.

Métodos utilizados:

- .client(), cria uma instância de cliente MQTT. O objeto cliente gere a conexão com o broker, lida com a comunicação de rede e fornece métodos para publicar e subscrever tópicos;
- .subscribe(topic), informa ao broker quais os tópicos o cliente deseja receber mensagens

- .on_connect(callback_function), Define uma função callback que é acionada quando o cliente se conecta com sucesso ao broker;
- .on_message(callback_function), Define uma função callback que é acionada quando uma mensagem chega ao tópico subscrito;
- .connect(host, port), estabelece a conexão ao broker.
- .loop_forever(), cria um loop infinito que gere todos os métodos em cima referidos.

3.3.2.3.2 Main do Backend MQTT

Loop principal. Começa por tentar conectar à base de dados, verifica a existência de uma tabela estruturada para acolhimento dos mesmos. Caso não exista, cria-a. De seguida, é criado o cliente MQTT e é estabelecida a conexão ao broker.

Funções:

- create_database_table(), esta função estabelece uma conexão à base de dados e executa a instrução SQL "CREATE TABLE IF NOT EXISTS". Assegurando que existe sempre uma estrutura para colocar os dados;
- on_connect(client, userdata, flags, rc), função *callback* que é acionada quando a conexão é estabelecida. Subscreve ao tópico "plc/data".
- on_message(client, userdata, msg), função callback que é acionada quando existe mensagem no tópico. Descodifica a string JSON e coloca os dados dentro da base de dados.

3.3.2.4 Frontend

Programa baseado em React[29] para monitorização e visualização em tempo real de dados. Este serviço serve como interface gráfico da aplicação. A integração com o Chart.js[30] fornece recursos de visualização, enquanto o Axios[31] lida com a comunicação HTTP com a API do *backend*.

3.3.3 Raspberry Pi

Este programa Python implementa um sistema de visão computacional em tempo real para controlo de qualidade automatizado de peças, integrado com um PLC Siemens através do protocolo Snap7. O sistema captura imagens de uma câmara USB, mede as dimensões das peças, valida-as de acordo com as especificações e comunica os resultados ao PLC para controlo de automação industrial.

3.3.3.1 Biblioteca cv2(OpenCV)

OpenCV[32] é uma biblioteca de código aberto utilizada para processamento de imagem e visão computacional. A biblioteca cv2 representa a interface Python do OpenCV.

Métodos utilizados:

- .VideoCapture(index), inicia a conexão com câmara USB;
- .cvtColor(image, color_space), converte a imagem em diferentes espaços de cor;
- .GaussianBlur(image), aplica um filtro gaussiano, reduzindo o ruido através de suavização da imagem;
- .findContours(image, mode, method), deteta os contornos presentes na imagem;
- .contourArea(contour), calcula a área de um contorno;
- .putText(image, texto, position, font, scale, color, thickness), adiciona texto à imagem.

3.3.3.2 Biblioteca Snap7

Snap7[33] é uma biblioteca de código aberto que utiliza o protocolo de comunicação S7 da Siemens, para aceder à memória do PLC.

Métodos utilizados:

- .connect(ip, rack, slot), estabelece a conexão com o PLC através do endereço de ip e a localização física da cpu na *rack;*
- .db_read(db_number, start), permite a leitura da base de dados no PLC

- .db_write(db_number, start), permite a escrita na base de dados no PLC
- .get_connected(), devolve um boleano que indica a condição da conexão.
- .get_bool(bytearray), extrai um valor boleano específico de um *array* de bytes
- .set_bool(bytearray), define o valor de um bit específico dentro de um *array* de bytes
- .disconnect(), encerra a conexão com o PLC

3.3.3.3 Main do Raspberry Pi

O programa principal implementa uma arquitetura baseada em eventos que coordena a captura de imagens, processamento, comunicação com o PLC e visualização de resultados.

Funções:

- initialize_camera(), Inicializa a câmara USB conectada ao Raspberry Pi. Verifica se a câmara foi inicializada corretamente através do método isOpened(), devolvendo o objeto de captura.
- connect_to_plc(), Cria uma instância do cliente Snap7 e tenta conectar-se ao PLC. Após estabelecer a conexão, verifica o estado através de get_connected() e devolve o objeto cliente para utilização nas operações subsequentes de leitura e escrita.
- send_result_to_plc(result), Esta função recebe como parâmetro uma string ("OK" ou "NOK") e escreve o resultado correspondente em posições específicas de memória do PLC. A função garante que apenas um dos bits (OK ou NOK) está ativo de cada vez, evitando estados ambíguos.
- read_bool_from_plc(), Monitoriza continuamente uma posição específica de memória do PLC onde está armazenada a flag de requisição de inspeção. A função realiza leituras cíclicas através de db_read(), extraindo o valor booleano relevante com get_bool(). Quando deteta uma transição de estado da flag, de False para True, devolve True para sinalizar que uma nova inspeção foi solicitada.

- calibrate_camera(), Executa a calibração do sistema de visão para estabelecer a relação entre pixels e centímetros. O processo começa por capturar uma imagem de um objeto de referência com dimensões conhecidas, processa a imagem para detetar os contornos do objeto e calcula o fator de conversão (pixels por centímetro) dividindo as dimensões em pixels pelas dimensões físicas reais. Este fator é armazenado e utilizado posteriormente para converter todas as medições de pixels para centímetros.
- detect_piece_size(frame), A função recebe um frame capturado pela câmara e aplica sequencialmente: conversão para escala de cinzentos (cv2.cvtColor), suavização gaussiana para redução de ruído (cv2.GaussianBlur), binarização através de *threshold* adaptativo (cv2.threshold), e deteção de contornos (cv2.findContours). Filtra os contornos detetados por área mínima para eliminar ruído, identifica o maior contorno como sendo a peça a inspecionar, e calcula o retângulo delimitador (cv2.boundingRect) para extrair largura e altura em pixels. Aplica o fator de calibração para converter estas dimensões para centímetros e devolve os valores medidos juntamente com a imagem anotada.
- is_piece_ok(width_cm, height_cm), Compara as dimensões medidas com as especificações de tolerância predefinidas. Recebe como parâmetros a largura e altura, em centímetros, e verifica se ambas as dimensões se enquadram dentro dos limites aceitáveis (dimensão ± tolerância). Devolve True se a peça está OK (dentro das tolerâncias) ou False caso contrário. As especificações de tolerância podem ser configuradas como constantes no início do programa ou carregadas de um ficheiro de configuração para maior flexibilidade.
- monitor_plc_bool(), Implementa o loop principal de monitorização que aguarda requisições do PLC. Executa esta função continuamente, invocando a função read_bool_from_plc() em intervalos regulares. Quando deteta uma requisição, desencadeia toda a sequência de inspeção: captura de frame, processamento através de detect_piece_size(), validação através de is_piece_ok(), e envio do

resultado através de send_result_to_plc(). Após completar a inspeção, dá *reset* à flag no PLC para sinalizar que o processamento foi concluído e o sistema está pronto para a próxima requisição.

- display_loop(), Fornece feedback visual em tempo real do sistema de visão. Executa num *thread* separado, capturando continuamente frames da câmara e exibindo-os numa janela OpenCV. Quando uma inspeção é processada, sobrepõe na imagem o resultado da medição (dimensões calculadas e classificação OK/NOK). A janela pode ser fechada pressionando 'q', encerrando corretamente o programa.

3.4 Demonstração Prática do Sistema

Para uma visualização completa do funcionamento do sistema desenvolvido em ambiente operacional, foi preparado um vídeo demonstrativo, disponível em https://youtu.be/zlzrFeF9EAY, que documenta todos os componentes da arquitetura em funcionamento. O vídeo ilustra o processo completo desde o acionamento do interruptor no PLC LOGO!, passando pelo protocolo desenvolvido, comunicação através de MQTT entre os microcontroladores ESP32, a captura e processamento de imagem pelo Raspberry Pi, até à apresentação dos resultados no *dashboard* web em tempo real. Esta documentação visual complementa a descrição técnica apresentada ao longo deste documento, evidenciando a viabilidade e funcionalidade da solução desenvolvida num cenário que replica condições operacionais industriais reais.

Capítulo 4: Conclusão e Futuros Desenvolvimentos

4.1 Conclusão

O objetivo principal deste projeto foi alcançado, não só como o protocolo foi criado, como também foi implementado com sucesso em contexto pratico. Demonstrando assim a possibilidade da sua aplicação em situações reais. Também foi possível ver a sua aplicabilidade em elevar linhas de automação industrial mais antigas, às novas tecnologias da indústria 4.0 por um custo muito mais acessível.

4.2 Futuros Desenvolvimentos

A transição de uma placa de protótipo para uma placa de circuito impresso construída especificamente representa uma progressão lógica na maturação do sistema. Havendo duas possibilidades de design. Uma placa que acomoda módulos de desenvolvimento ESP32 amovíveis ou uma solução integrada com circuitos ESP32 incorporados. Cada abordagem apresenta compensações distintas em relação à modularidade, otimização de custos e complexidade de fabricação.

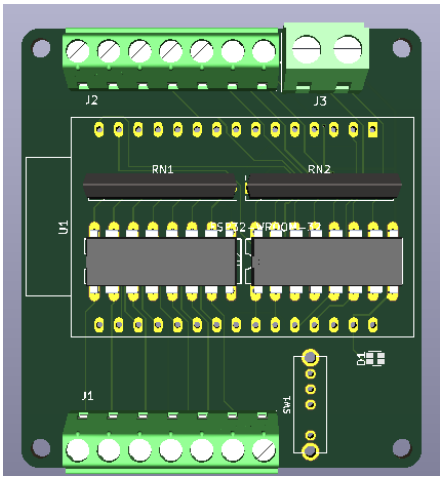


Figura 12 Proposta de PCB customizado para placa de desenvolvimento ESP32

Expandir as capacidades do sistema para incorporar canais analógicos de entrada e saída ampliaria significativamente a sua aplicabilidade em diversos cenários de deteção e atuação.

Uma migração do ambiente de programação atual para linguagens de nível inferior, como C ou Rust. Estas linguagens oferecem maior eficiência de execução, espaço de memória reduzido e características de desempenho determinísticas atributos

particularmente valiosos em sistemas embebidos com recursos limitados. Além disso, o Rust fornece garantias de segurança de memória sem sobrecarga de tempo de execução, potencialmente melhorando a confiabilidade do sistema.

O desenvolvimento de uma interface de configuração Web, acessível através do endereço IP do ESP32, permitiria a adaptação do protocolo sem exigir conhecimentos de programação. Facilitaria a rápida implantação em contextos operacionais variados, reduzindo ao mesmo tempo a barreira técnica à personalização do sistema.

A restrição atual de sete canais de entrada e saída sugere a necessidade de desenvolvimento de placas de expansão. Essa arquitetura escalável aumentaria a versatilidade do sistema em várias aplicações industriais.

A integração do *feed* da camera no *dashboard* forneceria recursos valiosos de verificação de dados. A confirmação visual em tempo real pode aumentar a confiança nas leituras dos sensores e facilitar a deteção de anomalias através da validação intermodal.

O *dashboard* em si, também beneficiaria de uma interface de personalização acessível ao utilizador. Permitiria a adaptação do painel a diversos cenários de monitorização sem a necessidade de modificações na base de código, melhorando assim a flexibilidade do sistema e reduzindo a complexidade da implantação.

Bibliografia

[1] Andrade, L. H. S. (2021). Contributo para a digitalização da indústria do mármore com identificação automática por sistemas RFID [Dissertação de mestrado, Universidade de Évora]. Universidade de Évora - Escola de Ciências e Tecnologia.

[2] Product data sheet 6ES7314-1AG14-0AB0. (n.d.). *SIMATIC S7-300, CPU 314* https://docs.rs-online.com/f610/0900766b8172fcf8.pdf

[3] Product data sheet 6GK7343-1CX10-0XE0. (n.d.). *Communications processor CP 343-1 Lean for connection of SIMATIC S7-300.* https://docs.rs-online.com/65b4/0900766b8172fccf.pdf

[4] Product data sheet 6ES7321-1BL00-0AA0. (n.d.). https://docs.rs-online.com/301b/0900766b8131a202.pdf

[5] Product data sheet 6ES7322-1BL00-0AA0. (n.d). https://docs.rs-online.com/4b10/0900766b8131a204.pdf

[6] Product data sheet 6ED1052-1CC08-0BA0. (n.d.). https://docs.rs-online.com/eb10/0900766b8164240a.pdf

[7] ESP32 Manual PDF. (n.d.). *ESP32 Manual PDF*. https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[8] ULN2003a Datasheet(pdf) – Texas Instruments. (n.d.). *High Voltage, High Current Darlington transistor array.* https://www.alldatasheet.com/datasheet-pdf/view/29520/TI/ULN2003A.html

[9] Product data sheet Raspberry Pi 3 Model B. (n.d.). https://us.rs-online.com/m/d/4252b1ecd92888dbb9d8a39b536e7bf2.pdf?srsltid=AfmBOopJ4yknKmYqG1arOxrvi-TA0Y8-jBV8q6yqWuFWaqD-HwY1GjM5

[10] Siemens TIA Docs. (n.d.). *Siemens TIA Docs*. https://docs.tia.siemens.cloud

[11] Siemens PDF Manual. (n.d.). *Siemens PDF Manual LOGO!Soft Comfort Online Help*.https://cache.industry.siemens.com/dl/files/807/100782807/att_924632/v1/Help_en-US_en-US.pdf

[12] Visual Studio Code Documentation (n.d.) https://code.visualstudio.com/docs

[13] Thonny Documentation (n.d.). *Thonny: The Beginner-Friendly Python Editor*. https://realpython.com/python-thonny

[14] Ubuntu Documentation (n.d.). *Guia do ambiente de trabalho Ubuntu*. https://help.ubuntu.com/lts/ubuntu-help/index.html

[15] Raspberry Pi OS Documentation (n.d.). https://www.raspberrypi.com/documentation/computers/os.html

[16] Docker Docs. (n.d.). *Docker Docs*. https://docs.docker.com

[17] Python Docs. (n.d.). *Python Docs*. https://docs.python.org/3/

[18] DevDocs JavaScript. (n.d.). *DevDocs JavaScript*. https://devdocs.io/javascript/

[19] Vite Guide. (n.d.). *Vite Guide*. https://vite.dev/guide/

[20] KiCad Docs. (n.d.). *KiCad Docs*. https://docs.kicad.org

[21] Mosquitto Docs. (n.d.). *Mosquitto documentation*. https://mosquitto.org/documentation/

[22] Umqtt Documentation. (n.d.). https://mpython.readthedocs.io/en/v2.2.1/library/mPython/umqtt.simple.html

[23] Docker Compose Docs. (n.d.). *Docker Compose Docs*. https://docs.docker.com/compose/

[24] PostgreSQL Docs. (n.d.). *PostgreSQL v18.0 Documentation*. https://www.postgresql.org/docs/

[25] FastAPI. (n.d.). *FastAPI*. https://fastapi.tiangolo.com

[26] Psycopg Docs. (n.d.). *Psycopg v2.9.10 – PostgreSQL database adapter for python.* https://www.psycopg.org/docs/

[27] Uvicorn Docs. (n.d.). *Web Server for Python.* https://www.uvicorn.org

[28] Eclipse paho-mqtt Docs. (n.d.). *Ecplipse paho-mqtt documentation*. https://eclipse.dev/paho/files/paho.mqtt.python/html/client.html

[29] React Learn. (n.d.). *React Learn*. https://react.dev/learn

[30] Chartjs Docs. (n.d.). *Chartjs documentation v4.5.0.* https://www.chartjs.org/docs/latest/

[31] Axios Docs. (n.d.). *Promise-based HTTP Client for node.js.* https://axios-http.com/docs/intro

[32] OpenCV Documentation. (n.d.). https://docs.opencv.org/3.0-beta/index.html

[33] Python-Snap7 Docs. (n.d.). *Welcome to python-snap7's documentation!* https://python-snap7.readthedocs.io/en/latest/

Anexos:

esp32_S7-314.py

```python
import network
import time
import machine
from umqtt.simple import MQTTClient
import json



MQTT_BROKER = 'tesedoalas.duckdns.org'
MQTT_PORT = 1883
MQTT_USER = None
MQTT_PASSWORD = None
MQTT_CLIENT_ID = 'esp32_s7_314'
MQTT_TOPIC_SUB = 'esp32_LOGO!/'
MQTT_TOPIC_PUB = ['plc/data','esp32_s7_314/']


input_pins = [machine.Pin(i, machine.Pin.IN) for i in [33, 32, 35, 34, 39, 36]]
output_pins = [machine.Pin(i, machine.Pin.OUT) for i in [12, 14, 27, 26, 22, 23]]

for pin in output_pins:
    pin.value(1)


previous_states = [pin.value() for pin in input_pins]







def mqtt_callback(topic, msg):
    states = msg.decode()
    print('Mensagem recebida:', topic, states)
    if states == PROTOCOLO_COMUNICACAO['GET_CAM']:
        output_pins[0].value(0)
        time.sleep(0.2)
        output_pins[0].value(1)



def connect_mqtt():
    client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER, port=MQTT_PORT, user=MQTT_USER, password=MQTT_PASSWORD)
    client.set_callback(mqtt_callback)
    client.connect()
    client.subscribe(MQTT_TOPIC_SUB)
    print(f'Inscrito no tópico {MQTT_TOPIC_SUB}')
    return client



def detect_falling_edges():
    global previous_states
    current_states = [pin.value() for pin in input_pins]
    falling_edges = []

    for i in range(len(input_pins)):

        if previous_states[i] == 1 and current_states[i] == 0:
            falling_edges.append(i)

    previous_states = current_states
    return falling_edges, current_states


```

```python
def publicar_contadores(client, falling_edges, current_states):
    global OK, NOK


    if not falling_edges:
        return

    estado_str = ','.join([str(state) for state in current_states])

    if estado_str == PROTOCOLO_COMUNICACAO['OK']:
        OK += 1
        client.publish(MQTT_TOPIC_PUB[1], PROTOCOLO_COMUNICACAO['OK'])
        print(f'OK detectado - Total: {OK}')
    elif estado_str == PROTOCOLO_COMUNICACAO['NOK']:
        NOK += 1
        client.publish(MQTT_TOPIC_PUB[1], PROTOCOLO_COMUNICACAO['NOK'])
        print(f'NOK detectado - Total: {NOK}')
    elif estado_str == PROTOCOLO_COMUNICACAO['RESET_DIA']:
        OK, NOK = 0, 0
        print('Contadores resetados')

    dados = {"pecas_boas": OK, "pecas_mas": NOK}
    mensagem = json.dumps(dados)
    client.publish(MQTT_TOPIC_PUB[0], mensagem)
    print(f'Mensagem publicada: {mensagem}')



def main():
    client = connect_mqtt()
    while True:
        client.check_msg()
        falling_edges, current_states = detect_falling_edges()
        publicar_contadores(client, falling_edges, current_states)
        time.sleep(0.05)


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

esp32_LOGO.py

```python
import network
import time
import machine
from umqtt.simple import MQTTClient


# Configurações MQTT
MQTT_BROKER = 'tesedoalas.duckdns.org'
MQTT_PORT = 1883
MQTT_USER = None
MQTT_PASSWORD = None
MQTT_CLIENT_ID = 'esp32_LOGO!'
MQTT_TOPIC_SUB = 'esp32_s7_314/'
MQTT_TOPIC_PUB = 'esp32_LOGO!/'


input_pins = [machine.Pin(i, machine.Pin.IN) for i in [33, 32, 35, 34, 39, 36]]
output_pins = [machine.Pin(i, machine.Pin.OUT) for i in [12, 14, 27, 26, 22, 23]]

for pin in output_pins:
    pin.value(1)


previous_states = [pin.value() for pin in input_pins]




def mqtt_callback(topic, msg):
    states = msg.decode()
    print('Mensagem recebida:', topic, states)
    if states == PROTOCOLO_COMUNICACAO['OK']:
        output_pins[0].value(0)
        time.sleep(0.2)
        output_pins[0].value(1)
    elif states == PROTOCOLO_COMUNICACAO['NOK']:
        output_pins[1].value(0)
        time.sleep(0.2)
        output_pins[1].value(1)



def connect_mqtt():
    client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER, port=MQTT_PORT, user=MQTT_USER, password=MQTT_PASSWORD)
    client.set_callback(mqtt_callback)
    client.connect()
    client.subscribe(MQTT_TOPIC_SUB)
    print(f'Inscrito no tópico {MQTT_TOPIC_SUB}')
    return client



def detect_falling_edges():
    global previous_states
    current_states = [pin.value() for pin in input_pins]
    falling_edges = []

    for i in range(len(input_pins)):
        # Detecta transição de 1 para 0 (borda de descida = sinal ativo)
        if previous_states[i] == 1 and current_states[i] == 0:
            falling_edges.append(i)

    previous_states = current_states
    return falling_edges, current_states



def publish_states(client, falling_edges, current_states):
    # Só publica se houver pelo menos uma borda de descida
    if not falling_edges:
        return

    message = ','.join([str(state) for state in current_states])
    client.publish(MQTT_TOPIC_PUB, message)
    print(f'Mensagem publicada: {message} (bordas detectadas nos pinos: {falling_edges})')



def main():
    client = connect_mqtt()
    while True:
        client.check_msg()
        falling_edges, current_states = detect_falling_edges()
        publish_states(client, falling_edges, current_states)
        time.sleep(0.05)  # Intervalo reduzido para melhor detecção de bordas


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

docker-compose.yml

```yaml
version: '3.8'

services:
  database:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: plc_database
      POSTGRES_USER:
      POSTGRES_PASSWORD:
    ports:
      - "5432:5432"
    restart: always
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U nallas -d plc_database"]
      interval: 5s
      timeout: 5s
      retries: 5

  mqtt-broker:
    image: eclipse-mosquitto
    ports:
      - "1883:1883"
    volumes:
      - ./mosquitto/config:/mosquitto/config
    restart: always

  backend-mqtt:
    build: ./backend-mqtt
    depends_on:
      database:
        condition: service_healthy
      mqtt-broker:
        condition: service_started
    environment:
      - POSTGRES_DB=plc_database
      - POSTGRES_USER=
      - POSTGRES_PASSWORD=
      - POSTGRES_HOST=database
      - MQTT_BROKER=mqtt-broker
      - MQTT_TOPIC=plc/data
    restart: always

  backend-api:
    build: ./backend-api
    ports:
      - "8000:8000"
    depends_on:
      database:
        condition: service_healthy
    environment:
      - POSTGRES_DB=plc_database
      - POSTGRES_USER=
      - POSTGRES_PASSWORD=
      - POSTGRES_HOST=database
    restart: always

  frontend:
    build: ./frontend
    ports:
      - "3001:80"
    depends_on:
      - backend-api
    restart: always

volumes:
  postgres_data:
```

backendapi.py

```python
import os
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import psycopg2
import psycopg2.extras
import time
from datetime import datetime
from pydantic import BaseModel
from typing import List, Optional



app = FastAPI(
    title="PLC Data API",
    description="API para monitoramento de dados PLC",
    version="1.0.0"
)


app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Permite todas as origens
    allow_credentials=True,
    allow_methods=["*"],  # Permite todos os métodos
    allow_headers=["*"],  # Permite todos os cabeçalhos
)

class PlcData(BaseModel):
    id: Optional[int] = None
    timestamp: str
    pecas_boas: int
    pecas_mas: int

class PlcDataInput(BaseModel):
    pecas_boas: int
    pecas_mas: int

def get_db_connection():
    """Get a database connection with retry mechanism"""
    max_retries = 5
    retry_count = 0

    while retry_count < max_retries:
        try:
            conn = psycopg2.connect(**DB_CONFIG)
            return conn
        except Exception as e:
            retry_count += 1
            print(f"Database connection attempt {retry_count}/{max_retries} failed: {e}")
            if retry_count >= max_retries:
                raise
            time.sleep(1)

@app.get("/")
async def root():
    return {"message": "PLC Monitoring API"}

@app.get("/data", response_model=List[PlcData])
async def get_data(limit: int = 50):
    try:
        conn = get_db_connection()
        cursor = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)

        cursor.execute('''
            SELECT id, timestamp, pecas_boas, pecas_mas
            FROM dados_plc
            ORDER BY timestamp DESC
            LIMIT %s
        ''', (limit,))

        results = cursor.fetchall()
        cursor.close()
        conn.close()


        return [
            {
                "id": row["id"],
                "timestamp": row["timestamp"].isoformat(),
                "pecas_boas": int(row["pecas_boas"]),
                "pecas_mas": int(row["pecas_mas"])
            }
            for row in results
        ]
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Database error: {str(e)}")

```

```python
@app.post("/data", response_model=PlcData)
async def post_data(data: PlcDataInput):
    try:
        conn = get_db_connection()
        cursor = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)

        cursor.execute('''
            INSERT INTO dados_plc (pecas_boas, pecas_mas)
            VALUES (%s, %s)
            RETURNING id, timestamp, pecas_boas, pecas_mas
        ''', (data.pecas_boas, data.pecas_mas))

        result = cursor.fetchone()
        conn.commit()
        cursor.close()
        conn.close()

        return {
            "id": result["id"],
            "timestamp": result["timestamp"].isoformat(),
            "pecas_boas": int(result["pecas_boas"]),
            "pecas_mas": int(result["pecas_mas"])
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Database error: {str(e)}")

@app.get("/health")
async def health_check():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT 1")
        cursor.close()
        conn.close()
        return {"status": "healthy", "database": "connected"}
    except Exception as e:
        raise HTTPException(status_code=503, detail=f"Health check failed: {str(e)}")


@app.on_event("startup")
async def startup_db_client():
    max_retries = 10
    retry_count = 0

    while retry_count < max_retries:
        try:
            conn = psycopg2.connect(**DB_CONFIG)
            conn.close()
            print("Database connection successful")

            # Create table if it doesn't exist
            conn = psycopg2.connect(**DB_CONFIG)
            cursor = conn.cursor()
            cursor.execute('''
                CREATE TABLE IF NOT EXISTS dados_plc (
                    id SERIAL PRIMARY KEY,
                    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    pecas_boas INT,
                    pecas_mas INT
                )
            ''')
            conn.commit()
            cursor.close()
            conn.close()
            print("Database table created/verified successfully")
            break
        except Exception as e:
            retry_count += 1
            print(f"Startup database connection attempt {retry_count}/{max_retries} failed: {e}")
            if retry_count >= max_retries:
                print("Failed to connect to the database during startup")
            time.sleep(2)

import uvicorn

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

backendmqtt.py

```python
import os
import paho.mqtt.client as mqtt
import psycopg2
from datetime import datetime
import json
import time




MQTT_BROKER = os.getenv('MQTT_BROKER', 'mqtt-broker')
MQTT_PORT = int(os.getenv('MQTT_PORT', 1883))
MQTT_TOPIC = os.getenv('MQTT_TOPIC', 'plc/data')


def create_database_table():
    """Create the table to store PLC data if it doesn't exist"""
    try:
        conn = psycopg2.connect(**DB_CONFIG)
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS dados_plc (
                id SERIAL PRIMARY KEY,
                timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                pecas_boas INT,
                pecas_mas INT
            )
        ''')
        conn.commit()
        cursor.close()
        conn.close()
        print("Database table created successfully")
    except Exception as e:
        print(f"Error creating database table: {e}")


def on_connect(client, userdata, flags, rc):
    """Callback when MQTT client connects"""
    print(f"Connected to MQTT Broker with result code {rc}")
    client.subscribe(MQTT_TOPIC)
    print(f"Subscribed to topic: {MQTT_TOPIC}")


def on_message(client, userdata, msg):
    """Callback when a message is received from the MQTT broker"""
    try:

        payload = json.loads(msg.payload.decode())
        print(f"Received message on topic {msg.topic}: {payload}")


        conn = psycopg2.connect(**DB_CONFIG)
        cursor = conn.cursor()


        cursor.execute('''
            INSERT INTO dados_plc (pecas_boas, pecas_mas)
            VALUES (%s, %s)
        ''', (
            payload.get('pecas_boas', 0),
            payload.get('pecas_mas', 0)
        ))


        conn.commit()
        cursor.close()
        conn.close()

        print(f"Data saved to database: {payload}")

    except Exception as e:
        print(f"Error processing message: {e}")
```

```python

def main():

    max_retries = 30
    retry_count = 0

    while retry_count < max_retries:
        try:
            # Try to connect to the database
            conn = psycopg2.connect(**DB_CONFIG)
            conn.close()
            print("Database connection successful")
            break
        except Exception as e:
            retry_count += 1
            print(f"Database connection attempt {retry_count}/{max_retries} failed: {e}")
            time.sleep(2)

    if retry_count >= max_retries:
        print("Failed to connect to the database after multiple attempts")
        return


    create_database_table()


    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message


    connected = False
    retry_count = 0

    while not connected and retry_count < max_retries:
        try:
            client.connect(MQTT_BROKER, MQTT_PORT, 60)
            connected = True
            print(f"Connected to MQTT broker at {MQTT_BROKER}:{MQTT_PORT}")
        except Exception as e:
            retry_count += 1
            print(f"MQTT connection attempt {retry_count}/{max_retries} failed: {e}")
            time.sleep(2)

    if not connected:
        print("Failed to connect to the MQTT broker after multiple attempts")
        return


    print("Starting MQTT client loop...")
    client.loop_forever()


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

frontend.jsx

```jsx
import { useState, useEffect } from 'react';
import axios from 'axios';
import { Line } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';


ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);

function App() {
  const [data, setData] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        setIsLoading(true);
        const response = await axios.get(`${import.meta.env.VITE_API_URL}/data`);
        const sortedData = [...response.data].sort((a, b) => {
          return new Date(a.timestamp) - new Date(b.timestamp);
        });
        setData(sortedData);
        setError(null);
      } catch (err) {
        setError('Failed to fetch data. Please try again later.');
        console.error('Error fetching data:', err);
      } finally {
        setIsLoading(false);
      }
    };


    fetchData();


    const interval = setInterval(fetchData, 5000);


    return () => clearInterval(interval);
  }, []);

  const chartData = {
    labels: data.map(item => new Date(item.timestamp).toLocaleTimeString()),
    datasets: [
      {
        label: 'Pecas Boas',
        data: data.map(item => item.pecas_boas),
        borderColor: 'rgb(59, 130, 246)',
        backgroundColor: 'rgba(59, 130, 246, 0.5)',
        tension: 0.1
      },
      {
        label: 'Pecas Más',
        data: data.map(item => item.pecas_mas),
        borderColor: 'rgb(249, 115, 22)',
        backgroundColor: 'rgba(249, 115, 22, 0.5)',
        tension: 0.1
      }
    ]
  };

  const chartOptions = {
    responsive: true,
    plugins: {
      legend: {
        position: 'top',
      },
      title: {
        display: true,
        text: 'Monitoramento de Dados PLC'
      },
    },
  };
```

```
  return (
    <div className="min-h-screen bg-gray-50">
      <div className="container mx-auto px-4 py-8">
        <header className="mb-8">
          <h1 className="text-3xl font-bold text-gray-800">Projeto Tese - Nuno Alas</h1>
          <p className="text-gray-600">Monitoramento em tempo real dos dados dos PLCs</p>
        </header>

        {isLoading && data.length === 0 ? (
          <div className="flex justify-center items-center h-64">
            <div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-blue-500"></div>
          </div>
        ) : error ? (
          <div className="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded relative" role="alert">
            <span className="block sm:inline">{error}</span>
          </div>
        ) : (
          <div className="grid grid-cols-1 lg:grid-cols-2 gap-8">
            <div className="bg-white rounded-lg shadow p-6">
              <h2 className="text-xl font-semibold mb-4">Gráfico de Valores</h2>
              <Line data={chartData} options={chartOptions} />
            </div>

            <div className="bg-white rounded-lg shadow p-6">
              <h2 className="text-xl font-semibold mb-4">Leituras Recentes</h2>
              <div className="overflow-x-auto">
                <table className="min-w-full divide-y divide-gray-200">
                  <thead className="bg-gray-50">
                    <tr>
                      <th scope="col" className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">
                        Timestamp
                      </th>
                      <th scope="col" className="px-6 py-3 text-right text-xs font-medium text-gray-500 uppercase tracking-wider">
                        Peças Boas
                      </th>
                      <th scope="col" className="px-6 py-3 text-right text-xs font-medium text-gray-500 uppercase tracking-wider">
                        Peças Más
                      </th>
                    </tr>
                  </thead>
                  <tbody className="bg-white divide-y divide-gray-200">
                    {data.slice(0, 10).map((item, index) => (
                      <tr key={index} className={index % 2 === 0 ? 'bg-white' : 'bg-gray-50'}>
                        <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500">
                          {new Date(item.timestamp).toLocaleString()}
                        </td>
                        <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500 text-right">
                          {item.pecas_boas.toFixed(2)}
                        </td>
                        <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500 text-right">
                          {item.pecas_mas.toFixed(2)}
                        </td>
                      </tr>
                    ))}
                  </tbody>
                </table>
              </div>
            </div>
          </div>
        )}
      </div>
    </div>
  );
}

export default App;
```

deteção_pecas.py

```python
import cv2
import numpy as np
import time
import snap7
from snap7.util import *
from threading import Thread
import os
import sys


# Configurações PLC
PLC_IP = "192.168.0.1"
PLC_RACK = 0
PLC_SLOT = 2
DB_NUMBER = 1
BYTE_OFFSET = 0
BIT_OK = 0
BIT_NOK = 1
BOOL_BIT = 2
POLLING_INTERVAL = 0.5

# Configurações Câmera
CAMERA_ID = 0
REFERENCE_WIDTH_CM = 2.5
REFERENCE_HEIGHT_CM = 2.0
TOLERANCE = 15
MIN_CONTOUR_AREA = 1000


camera = None
running = True
pixels_per_cm = 50
detection_ready = False
plc_client = None
last_plc_bool = False

def initialize_camera():

    global camera
    camera = cv2.VideoCapture(CAMERA_ID)

    if not camera.isOpened():
        raise Exception(f"Não foi possível abrir a câmera {CAMERA_ID}")

    camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    print(f"Câmera {CAMERA_ID} inicializada")
    return True

def connect_to_plc():

    global plc_client

    print(f"Tentando conectar ao PLC...")
    print(f"  IP: {PLC_IP}")
    print(f"  Rack: {PLC_RACK}")
    print(f"  Slot: {PLC_SLOT}")

    try:
        plc_client = snap7.client.Client()
        plc_client.connect(PLC_IP, PLC_RACK, PLC_SLOT)
        print(f"✓ Conectado ao PLC em {PLC_IP}")
        return True
    except Exception as e:
        print(f"✗ Erro ao conectar ao PLC: {e}")
        print("\nDicas de troubleshooting:")
        print("1. Verifique se o IP está correto")
        print("2. Teste: ping " + PLC_IP)
        print("3. Verifique RACK e SLOT")
        print("4. Certifique-se de que o PLC está ligado e acessível")
        print("5. Verifique se está na mesma rede")
        return False

def send_result_to_plc(result):

    global plc_client

    if plc_client is None or not plc_client.get_connected():
        if not connect_to_plc():
            return False

    try:
        data = plc_client.db_read(DB_NUMBER, BYTE_OFFSET, 1)

        if result == "OK":
            set_bool(data, 0, BIT_OK, True)
            set_bool(data, 0, BIT_NOK, False)
        elif result == "NOK":
            set_bool(data, 0, BIT_OK, False)
            set_bool(data, 0, BIT_NOK, True)
        else:  # NONE
            set_bool(data, 0, BIT_OK, False)
            set_bool(data, 0, BIT_NOK, False)

        plc_client.db_write(DB_NUMBER, BYTE_OFFSET, data)
        print(f"Enviado para PLC: {result}")
        return True
    except Exception as e:
        print(f"Erro ao enviar para PLC: {e}")
        plc_client = None
        return False
```

```python

def read_bool_from_plc():

    global plc_client

    if plc_client is None or not plc_client.get_connected():
        if not connect_to_plc():
            return False

    try:
        data = plc_client.db_read(DB_NUMBER, BYTE_OFFSET, 1)
        return get_bool(data, 0, BOOL_BIT)
    except Exception as e:
        print(f"Erro ao ler BOOL do PLC: {e}")
        plc_client = None
        return False

def calibrate_camera():

    global pixels_per_cm, MIN_CONTOUR_AREA, detection_ready

    print("===== CALIBRAÇÃO DA CÂMERA =====")
    print("Coloque a peça de referência (2.5 x 2.0 cm) sob a câmera")
    print("Pressione 'c' para calibrar ou 'q' para usar padrão")

    while True:
        ret, frame = camera.read()
        if not ret:
            continue

        # Add text to frame
        cv2.putText(frame, "Pressione 'c' para calibrar ou 'q' para padrao",
                    (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        # Try to show window - if it fails, fallback to auto-calibration
        try:
            cv2.imshow('Calibracao', frame)
            key = cv2.waitKey(1) & 0xFF
        except Exception as e:
            print(f"Não foi possível abrir janela de visualização: {e}")
            print("Usando calibração automática em 3 segundos...")
            time.sleep(3)
            key = ord('c')  # Force calibration

        if key == ord('c'):
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            _, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
            contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

            if contours:
                largest_contour = max(contours, key=cv2.contourArea)
                contour_area = cv2.contourArea(largest_contour)

                if contour_area > 500:
                    x, y, w, h = cv2.boundingRect(largest_contour)

                    pixels_per_cm_width = w / REFERENCE_WIDTH_CM
                    pixels_per_cm_height = h / REFERENCE_HEIGHT_CM
                    pixels_per_cm = (pixels_per_cm_width + pixels_per_cm_height) / 2

                    MIN_CONTOUR_AREA = (REFERENCE_WIDTH_CM * REFERENCE_HEIGHT_CM * pixels_per_cm * pixels_per_cm) * 0.2

                    print(f"Calibração concluída: {pixels_per_cm:.2f} pixels/cm")
                    print(f"Área mínima: {MIN_CONTOUR_AREA:.0f} pixels²")
                    try:
                        cv2.destroyWindow('Calibracao')
                    except:
                        pass
                    detection_ready = True
                    break
                else:
                    print("Peça muito pequena. Tente novamente.")
            else:
                print("Nenhuma peça detectada. Tente novamente.")
        elif key == ord('q'):
            print(f"Usando padrão: {pixels_per_cm:.2f} pixels/cm")
            MIN_CONTOUR_AREA = (REFERENCE_WIDTH_CM * REFERENCE_HEIGHT_CM * pixels_per_cm * pixels_per_cm) * 0.2
            try:
                cv2.destroyWindow('Calibracao')
            except:
                pass
            detection_ready = True
            break

def detect_piece_size(frame):

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    valid_contours = [c for c in contours if cv2.contourArea(c) > MIN_CONTOUR_AREA]

    if not valid_contours:
        return 0, 0, False

    largest_contour = max(valid_contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(largest_contour)

    width_cm = w / pixels_per_cm
    height_cm = h / pixels_per_cm

    return width_cm, height_cm, True
```

```
204
205 def is_piece_ok(width_cm, height_cm):
206
207     min_width = REFERENCE_WIDTH_CM * (1 - TOLERANCE/100)
208     max_width = REFERENCE_WIDTH_CM * (1 + TOLERANCE/100)
209     min_height = REFERENCE_HEIGHT_CM * (1 - TOLERANCE/100)
210     max_height = REFERENCE_HEIGHT_CM * (1 + TOLERANCE/100)
211
212     width_ok = min_width <= width_cm <= max_width
213     height_ok = min_height <= height_cm <= max_height
214
215     return width_ok and height_ok
216
217 def monitor_plc_bool():
218
219     global running, last_plc_bool
220
221     while running:
222         try:
223             current_bool = read_bool_from_plc()
224
225             # Detecção quando PLC sinaliza (transição 0→1)
226             if current_bool and not last_plc_bool:
227                 print("Sinal recebido do PLC - Iniciando detecção")
228
229                 if detection_ready:
230                     ret, frame = camera.read()
231                     if ret:
232                         width_cm, height_cm, piece_detected = detect_piece_size(frame)
233
234                         if piece_detected:
235                             result_ok = is_piece_ok(width_cm, height_cm)
236                             result_text = "OK" if result_ok else "NOK"
237
238                             send_result_to_plc(result_text)
239                             print(f"Resultado: {result_text} - Dimensões: {width_cm:.1f} x {height_cm:.1f} cm")
240                         else:
241                             send_result_to_plc("NONE")
242                             print("Nenhuma peça detectada")
243                     else:
244                         print("Erro ao capturar imagem")
245                 else:
246                     print("Sistema não calibrado")
247
248             last_plc_bool = current_bool
249             time.sleep(POLLING_INTERVAL)
250
251         except Exception as e:
252             print(f"Erro no monitoramento: {e}")
253             time.sleep(POLLING_INTERVAL)
254
255 def display_loop():
256
257     global running, detection_ready
258
259     print("\n===== JANELA DE VISUALIZAÇÃO =====")
260     print("Pressione 'q' na janela para encerrar o sistema")
261     print("Pressione 'r' para recalibrar\n")
262
263     last_detection_time = 0
264     last_result = None
265     last_width = 0
266     last_height = 0
267
268     while running:
269         try:
270             ret, frame = camera.read()
271             if not ret:
272                 time.sleep(0.1)
273                 continue
274
275             # Criar uma cópia para exibição
276             display_frame = frame.copy()
277
278             if detection_ready:
279                 # Fazer detecção contínua para feedback visual
280                 width_cm, height_cm, piece_detected = detect_piece_size(display_frame)
281
282                 if piece_detected:
283                     result_ok = is_piece_ok(width_cm, height_cm)
284                     result_text = "OK" if result_ok else "NOK"
285                     color = (0, 255, 0) if result_ok else (0, 0, 255)
286
287                     # Mostrar resultado grande e claro
288                     cv2.putText(display_frame, result_text, (50, 100),
289                                 cv2.FONT_HERSHEY_SIMPLEX, 3, color, 5)
290
291                     # Mostrar dimensões
292                     cv2.putText(display_frame, f"{width_cm:.2f} x {height_cm:.2f} cm",
293                                 (50, 150), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
294
295                     # Mostrar tolerâncias
296                     cv2.putText(display_frame, f"Ref: {REFERENCE_WIDTH_CM} x {REFERENCE_HEIGHT_CM} cm (+/-{TOLERANCE}%)",
297                                 (50, 180), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)
298
299                     last_result = result_text
300                     last_width = width_cm
301                     last_height = height_cm
302                     last_detection_time = time.time()
303                 else:
304                     # Sem peça detectada
305                     cv2.putText(display_frame, "Aguardando peca...", (50, 100),
306                                 cv2.FONT_HERSHEY_SIMPLEX, 1.5, (255, 255, 0), 3)
307
308                     # Mostrar último resultado por 3 segundos
309                     if last_result and (time.time() - last_detection_time) < 3:
310                         color = (0, 255, 0) if last_result == "OK" else (0, 0, 255)
311                         cv2.putText(display_frame, f"Ultimo: {last_result}", (50, 150),
312                                     cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
313                         cv2.putText(display_frame, f"{last_width:.2f} x {last_height:.2f} cm",
314                                     (50, 180), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
315
```

```
                # Status no canto
                cv2.putText(display_frame, "Sistema PRONTO", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
            else:
                # Sistema não calibrado
                cv2.putText(display_frame, "Sistema NAO CALIBRADO", (50, 100),
                            cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 255), 3)
                cv2.putText(display_frame, "Pressione 'r' para calibrar", (50, 150),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)

            # Instruções
            cv2.putText(display_frame, "q=Sair | r=Recalibrar", (10, display_frame.shape[0] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)

            # Mostrar frame
            cv2.imshow('Deteccao de Pecas - PLC', display_frame)

            # Verificar teclas
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                print("\nEncerrando sistema...")
                running = False
                break
            elif key == ord('r'):
                print("\nIniciando recalibração...")
                cv2.destroyWindow('Deteccao de Pecas - PLC')
                calibrate_camera()

        except Exception as e:
            print(f"Erro no display: {e}")
            time.sleep(0.1)

    cv2.destroyAllWindows()

def main():

    global camera, running

    try:
        initialize_camera()
        calibrate_camera()
        connect_to_plc()

        # Thread de monitoramento do PLC
        plc_thread = Thread(target=monitor_plc_bool)
        plc_thread.daemon = True
        plc_thread.start()

        print("\n===== SISTEMA INICIADO =====")
        print(f"Peça OK: {REFERENCE_WIDTH_CM} x {REFERENCE_HEIGHT_CM} cm (±{TOLERANCE}%)")
        print(f"Monitorizando DB{DB_NUMBER}.DBX{BYTE_OFFSET}.{BOOL_BIT}")
        print("A abrir janela de visualização...\n")

        # Loop de display (roda na thread principal)
        display_loop()

    except KeyboardInterrupt:
        print("\nA encerrar o programa...")
        running = False
    except Exception as e:
        print(f"Erro: {e}")
        running = False
    finally:
        running = False
        if camera is not None:
            camera.release()
        if plc_client is not None:
            plc_client.disconnect()
        try:
            cv2.destroyAllWindows()
        except:
            pass
        print("Sistema encerrado.")

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