UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia de Sorocaba
Campus de São João da Boa Vista

João Vitor Rocha Santana

Plataforma de Automação e Controle Open Source para Internet das Coisas Industrial

Sorocaba
2025

João Vitor Rocha Santana

Plataforma de Automação e Controle Open Source para Internet das Coisas Industrial

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica, junto ao Programa de Pós-Graduação em Engenharia Elétrica, Interunidades, entre o Instituto de Ciência e Tecnologia de Sorocaba e o Campus de São João da Boa Vista da Universidade Estadual Paulista “Júlio de Mesquita Filho”.

Orientador: Prof. Dr. Eduardo Paciência Godoy

Sorocaba
2025

S232p

Santana, João Vitor Rocha
Plataforma de automação e controle open source para Internet das Coisas Industrial / João Vitor Rocha Santana. -- Sorocaba, 2025
91 p.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Instituto de Ciência e Tecnologia, Sorocaba
Orientador: Eduardo Paciência Godoy

1. Automação industrial. 2. Internet das coisas. 3. Hardware. 4. Open source software. 5. Sistemas de controle inteligente. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

Impacto potencial desta pesquisa

O desenvolvimento de uma plataforma open source de automação e controle para aplicações de IIoT tem o potencial de impactar positivamente diversos aspectos da indústria. Essa iniciativa acelera a inovação ao permitir a contribuição de uma ampla comunidade de desenvolvedores, resultando em avanços mais rápidos e significativos na automação industrial. Além disso, essa plataforma reduz custos ao eliminar os altos custos de licenciamento de software proprietário, tornando a automação mais acessível para empresas de todos os portes. A flexibilidade e customização oferecidas por uma plataforma open source possibilitam a adaptação às necessidades específicas de cada empresa, enquanto a interoperabilidade aprimorada promove uma maior conectividade entre os sistemas. Por fim, o compartilhamento de conhecimento e colaboração facilitados por uma comunidade ativa de desenvolvedores que levam a soluções mais inovadoras e eficientes na automação industrial. Ainda, este trabalho é parte do programa de Pós-Graduação em Engenharia Elétrica da UNESP na categoria de Mestrado, envolvendo também parceria com empresas nacionais e internacionais e o apoio da Agência Unesp de Inovação (AUIN).

Potential impact of this research

The development of an open source automation and control platform for IIoT applications has the potential to positively impact several aspects of the industry. This initiative accelerates innovation by enabling contributions from a broad community of developers, resulting in faster and more significant advances in industrial automation. In addition, this platform reduces costs by eliminating the high licensing costs of proprietary software, making automation more accessible to companies of all sizes. The flexibility and customization offered by an open source platform allow adaptation to the specific needs of each company, while improved interoperability promotes greater connectivity between systems. Finally, knowledge sharing and collaboration facilitated by an active community of developers lead to more innovative and efficient solutions in industrial automation. Furthermore, this work is part of the UNESP Postgraduate Program in Electrical Engineering in the Master's category, also involving partnerships with national and international companies and the support of the Unesp Innovation Agency (AUIN).

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Sorocaba

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Plataforma de Automação e Controle Open Source para Internet das Coisas Industrial

AUTOR: JOÃO VITOR ROCHA SANTANA

ORIENTADOR: EDUARDO PACIÊNCIA GODOY

Aprovado como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica, área: Automação pela Comissão Examinadora:

unesp
UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
ICPEdu - Certificado Pessoal
Assinado de forma digital por Eduardo Paciencia Godoy:29137583808
Dados: 2025.05.27 15:40:39 -03'00'

Prof. Dr. EDUARDO PACIÊNCIA GODOY (Participaçao Presencial)
Departamento de Engenharia de Controle e Automacao / Instituto de Ciencia e Tecnologia UNESP Campus de Sorocaba

gov.br
Documento assinado digitalmente
GALDENORO BOTURA JUNIOR
Data: 27/05/2025 15:24:50-0300
Verifique em https://validar.iti.gov.br

Professor Titular GALDENORO BOTURA JUNIOR (Participaçao Presencial)
Programa de Pos-graduacao em Design / Faculdade de Arquitetura Artes Comunicacao e Design - Campus de Bauru

Firmato digitalmente da Dennis Brandao
Data: 27.05.2025 17:42:40 CEST
Organizzazione: UNIVERSITA' DEGLI STUDI DI BRESCIA/01773710171

Prof. Dr. DENNIS BRANDÃO (Participaçao Virtual)
Departamento de Engenharia Elétrica e da Computação / Universidade de Bréscia - Itália

Sorocaba, 27 de maio de 2025

AGRADECIMENTOS

Primeiramente a Deus por me guiar e me fortalecer ao longo de todo o processo

Aos meus pais, Maria e José por terem me ensinando a ser uma pessoa honesta e dedicada.

Ao Professor Eduardo Paciência Godoy pela confiança, amizade, dedicação e orientação durante todas as etapas do desenvolvimento deste trabalho.

Aos colegas que estiveram presente ao longo do desenvolvimento pelas trocas de experiências e suporte prestado.

“Sei que podes fazer todas as coisas; nenhum dos teus planos pode ser frustrado.”

Jó 42:2

RESUMO

Com o avanço da Indústria 4.0 e a crescente demanda por soluções flexíveis, conectadas e de baixo custo, este trabalho apresenta o desenvolvimento de uma plataforma de código aberto (open source) de automação e controle voltada para aplicações de Internet das Coisas Industrial (IIoT). A proposta integra hardware e software desenvolvidos para atender aos requisitos industriais de robustez, escalabilidade e interoperabilidade. O hardware, baseado no microcontrolador RP2040, oferece entradas e saídas digitais, analógicas, PWM, suporte a encoders, e interfaces de comunicação Ethernet, RS-232 e RS-485. Seu projeto priorizou modularidade, uso de componentes validados industrialmente e serialização de I/Os via protocolo SPI, possibilitando maior flexibilidade e expansão.No software, a plataforma adota duas abordagens complementares: OpenPLC, compatível com a norma IEC 61131-3, e MicroPython, utilizado por sua simplicidade, agilidade na prototipagem e facilidade de integração com sistemas IIoT. Foram realizadas adaptações no OpenPLC para suportar a serialização dos sinais, além do desenvolvimento de blocos específicos para leitura analógica e comunicação MQTT. A validação experimental foi realizada com dois estudos de caso: uma estação de separação de peças e um controle de velocidade de motor com PID em tempo real, ambos com visualização dos dados via Node-RED. Nos dois cenários a plataforma apresentou um resultado satisfatório e atingiu as expectativas e exigências propostas, além de demonstrar sua alta adaptabilidade a diferentes cenários. Com documentação aberta e disponibilizada publicamente, a plataforma desenvolvida promove o acesso democrático à automação industrial, oferecendo um recurso de alta aplicabilidade tanto para o meio acadêmico quanto para o setor produtivo, fortalecendo a inovação aberta e a customização de soluções industriais.

Palavras-chave: *Indústria 4.0, IIoT, Automação Industrial, Plataforma Open Source, OpenPLC, MicroPython.*

ABSTRACT

With the advancement of Industry 4.0 and the growing demand for flexible, connected, and low-cost solutions, this paper presents the development of an open source automation and control platform aimed at Industrial Internet of Things (IIoT) applications. The proposal integrates hardware and software developed to meet industrial requirements for robustness, scalability, and interoperability. The hardware, based on the RP2040 microcontroller, offers digital, analog, and PWM inputs and outputs, encoder support, and Ethernet, RS-232, and RS-485 communication interfaces. Its design prioritized modularity, the use of industrially validated components, and I/O serialization via SPI protocol, enabling greater flexibility and expansion. In software, the platform adopts two complementary approaches: OpenPLC, compatible with the IEC 61131-3 standard, and MicroPython, used for its simplicity, agility in prototyping, and ease of integration with IIoT systems. Adaptations were made to OpenPLC to support signal serialization, in addition to the development of specific blocks for analog reading and MQTT communication. Experimental validation was performed with two case studies: a parts separation station and a real-time PID motor speed control, both with data visualization via Node-RED. In both scenarios, the platform presented satisfactory results and met the proposed expectations and requirements, in addition to demonstrating its high adaptability to different scenarios. With open and publicly available documentation, the developed platform promotes democratic access to industrial automation, offering a highly applicable resource for both academia and the production sector, strengthening open innovation and the customization of industrial solutions.

Keywords: *Industry 4.0, IIoT, Industrial Automation, Open Source Platform, OpenPLC, MicroPython*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ADC	Conversor Analógico-Digital
CLP	Controlador Lógico Programável
CPS	Sistemas Ciberfísicos
CPU	*Central Processing Unit*
CI	Circuito Integrado
GASI	Grupo de Automação e Sistemas Integráveis
GPIO	*General Purpose Input/Output*
I4.0	*Industry 4.0*
IoT	*Internet of Things*
IIoT	*Industrial Internet of Things*
IEC	Comissão Eletrotécnica Internacional
IDE	*Integrated Development Environment*
MQTT	*Message Queuing Telemetry Transport*
OPC UA	*Open Platform Communication Unified Architecture*
PLC	*Programmable Logic Controller*
PWM	*Pulse Width Modulation*
PCB	*Printed Circuit Board*
SPI	Serial Peripheral Interface
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UART	*Universal Asynchronous Receiver/Transmitter*

SUMÁRIO

1. PREFÁCIO

Este trabalho é integrante do programa de Pós-Graduação em Engenharia Elétrica na categoria de Mestrado. Com a intermediação da Agência Unesp de Inovação (AUIN), foi firmada parceria com uma empresa externa, que por escolha dela terá o nome mantido em sigilo, para o desenvolvimento das atividades. A temática deste trabalho atende a uma demanda da empresa parceira no sentido de desenvolver soluções de automação e controle para aplicações industriais que atendam requisitos de conectividade para Internet das Coisas Industrial (IIoT). Desta forma, os estudos e aplicações realizados neste trabalho de Mestrado têm também a finalidade de desenvolver uma solução que atenda às necessidades da empresa parceira, mediante o acordo de cooperação firmado.

Além desta, outras empresas participaram como parceiras no desenvolvimento da solução, sendo elas a FACTS Engineering e a Autonomy Logic. A FACTS Engineering é uma empresa sediada nos Estados Unidos da América (EUA) e possui um foco na criação de produtos de automação que sejam fáceis de usar, de baixo custo, confiáveis e fáceis de adquirir. Já a Autonomy Logic, que também possui sede nos EUA, tem como seu produto principal o OpenPLC, um software aberto para programação de controladores lógico programável segundo a norma IEC 61131-3.

Neste contexto, as parcerias levantadas neste projeto têm como frente o desenvolvimento de uma solução de automação e controle que atenda às necessidades industriais, possua a confiabilidade necessária para utilização dela no chão de fábrica, atenda aos requisitos necessários para aplicações de IIoT e possa fomentar novos desenvolvimentos e parcerias através da adoção do open source.

2. INTRODUÇÃO

2.1. Justificativa

A Quarta Revolução Industrial, popularmente conhecida como Indústria 4.0 (I4.0), está provocando mudanças profundas nas práticas industriais e na forma como as operações são concebidas e executadas. Este movimento representa um avanço em relação às revoluções industriais anteriores, ao integrar tecnologias digitais em todos os aspectos da produção industrial. Entre as principais inovações dessa nova era, a Internet das Coisas Industrial (IIoT) destaca-se como uma das tecnologias habilitadoras mais transformadoras.

Ao conectar máquinas, dispositivos e sistemas em uma rede interconectada, a IIoT cria uma inteligência distribuída que transforma dados em tempo real em informações valiosas, permitindo uma tomada de decisão mais informada e uma eficiência operacional significativamente maior. Nesse sentido, a IIoT não apenas redefine a automação industrial, mas também impõe novos desafios e oportunidades para o setor.

A IIoT tem suas raízes na automação industrial tradicional, mas se diferencia pela capacidade de integrar tecnologia da informação (TI) e tecnologia operacional (TO) de forma abrangente. Seu desenvolvimento foi possibilitado pelo avanço nas tecnologias de sensores, comunicação sem fio, e computação em nuvem, que permitem a coleta massiva de dados e sua análise em tempo real. Esse nível de conectividade e inteligência operacional marca uma nova fase no ciclo de digitalização da indústria, trazendo benefícios como otimização de processos, maior eficiência energética, e uma gestão de recursos mais ágil e flexível.

Além disso, a IIoT permite uma abordagem preditiva e preventiva, reduzindo o tempo de inatividade de máquinas e aumentando a confiabilidade dos processos produtivos (SEHR et al., 2021). Assim, a automação industrial não apenas se torna mais inteligente, mas também mais responsiva às mudanças nas demandas do mercado.

Setores como manufatura, energia, saúde e logística já estão incorporando a IIoT em suas operações para aproveitar as vantagens da conectividade em tempo real e da análise avançada de dados. A capacidade de monitorar e ajustar processos em

tempo real tem se mostrado essencial para aumentar a eficiência, reduzir custos operacionais e otimizar o uso de recursos.

A crescente adoção de soluções baseadas em IIoT reflete a necessidade das indústrias de se adaptar a um ambiente operacional mais dinâmico e interconectado, no qual a automação desempenha um papel central. Em um cenário competitivo e em constante evolução, a busca por soluções de automação e controle flexíveis e integradas torna-se uma prioridade estratégica para empresas que desejam maximizar sua eficiência e competitividade global.

Nesse contexto, a interdependência entre a IIoT e os controladores industriais se torna evidente. A IIoT se apoia na conectividade de uma vasta gama de sensores e dispositivos para a coleta de dados operacionais em tempo real. Os controladores industriais, por sua vez, desempenham o papel fundamental de monitorar e controlar esses dispositivos, processando os dados coletados e atuando de forma a garantir a eficiência e a segurança dos processos produtivos.

Entre os controladores industriais, os Controladores Lógicos Programáveis (CLPs) destacam-se como uma tecnologia estabelecida e amplamente adotada na automação industrial. Os CLPs oferecem uma arquitetura de hardware modular, que facilita a instalação em diferentes ambientes e permite uma expansão simples através de módulos adicionais. Além disso, as normas de programação da IEC 61131-3 garantem a padronização e previsibilidade nos sistemas de automação, permitindo que os CLPs sejam utilizados em uma ampla variedade de aplicações industriais (SEHR et al., 2021).

No entanto, apesar de suas vantagens, os CLPs tradicionais enfrentam dificuldades para se adaptar às novas exigências da IIoT, especialmente em relação à conectividade, escalabilidade e segurança. Como observa Azarmipour et al. (2019), o ambiente de IIoT requer soluções de controle que sejam altamente adaptáveis e capazes de lidar com a complexidade crescente dos sistemas de automação. Os CLPs, projetados para operar em ambientes fechados e com um conjunto restrito de funções, muitas vezes não conseguem acompanhar a velocidade com que as tecnologias da IIoT estão evoluindo.

Além disso, a necessidade de integrar esses controladores com redes industriais modernas geralmente exige o uso de dispositivos externos e softwares proprietários, o que pode aumentar a complexidade e o custo de implementação (CONTIERI et al., 2023). Esses desafios sugerem que os CLPs, embora ainda sejam uma parte vital da

automação industrial, precisam evoluir para atender às demandas de conectividade e flexibilidade impostas pela IIoT.

Essa dificuldade de adaptação dos CLPs tradicionais também é destacada por Lundh et al. (2019), que observam que os CLPs são projetados para serem altamente especializados, o que limita sua capacidade de se integrar rapidamente a sistemas de automação mais amplos e dinâmicos. Wang et al. (2021) reforçam essa crítica, ao apontar que a integração de dados e a conectividade em tempo real são essenciais para a automação moderna, mas os CLPs tradicionais não foram desenvolvidos com essas capacidades em mente. O uso de sistemas fechados e proprietários também impede a flexibilidade e a customização necessárias para acompanhar o ritmo das inovações tecnológicas.

Diante dessas limitações, a indústria tem buscado alternativas aos CLPs tradicionais. De acordo com Jones e Smith (2020), há uma crescente demanda por soluções de automação mais flexíveis, escaláveis e econômicas, que possam integrar-se facilmente às novas tecnologias de IIoT. O desenvolvimento de controladores baseados em hardware aberto e software livre oferece uma alternativa promissora, permitindo que as indústrias implementem soluções de automação mais acessíveis e adaptáveis.

Lemos et al. (2021) argumentam que as indústrias que buscam maior flexibilidade e capacidade de integração rápida com novas tecnologias estão cada vez mais optando por soluções open source, que oferecem maior liberdade para customização tanto no hardware quanto no software. Esse tipo de solução permite que engenheiros ajustem os sistemas de controle de maneira ágil, integrando-os de forma eficiente aos sistemas de dados empresariais e aumentando a eficiência operacional.

A flexibilidade e a customização, dois dos principais benefícios das soluções open source, também são fundamentais para melhorar a experiência do cliente (Customer Experience - CX) em ambientes industriais. A transição para a Indústria 4.0 está promovendo uma valorização crescente da experiência do cliente, onde a customização de produtos e serviços se tornou um fator decisivo para o sucesso.

Parmentier et al. (2020) destacam que a customização não apenas melhora a eficiência operacional dos clientes, mas também aumenta sua satisfação e lealdade, já que os sistemas ajustados às necessidades específicas de cada cliente tendem a

oferecer resultados mais eficazes e alinhados com suas expectativas. Nesse contexto, as soluções open source se destacam por permitir um nível de personalização que os sistemas proprietários muitas vezes não conseguem atingir, devido à sua natureza fechada.

Além disso, a flexibilidade das plataformas open source permite que as indústrias se adaptem rapidamente a mudanças nas condições operacionais ou às novas demandas do mercado, sem a necessidade de substituir completamente os sistemas existentes (WANG et al., 2021). A capacidade de expandir e modificar os sistemas de automação de forma incremental é crucial em ambientes industriais dinâmicos, onde a inovação contínua é necessária para manter a competitividade.

Pereira e Lima (2022) observam que, em soluções proprietárias, a necessidade de customizações específicas é frequentemente limitada pelas funcionalidades predefinidas pelos fabricantes, o que pode ser um obstáculo significativo para empresas que buscam maior flexibilidade. Essa rigidez pode resultar em custos adicionais e no aumento do tempo de implementação, já que as modificações exigem adaptações complexas ou até a substituição de componentes. Em contraste, soluções open source ou sistemas baseados em plataformas modulares oferecem uma flexibilidade consideravelmente maior, permitindo que as empresas ajustem seus sistemas conforme as demandas específicas de seus processos. Além disso, esses sistemas podem ser facilmente atualizados ou expandidos à medida que novas tecnologias ou requisitos surgem, sem depender de um fornecedor exclusivo.

Diante desse cenário, o desenvolvimento de uma plataforma de automação e controle open source para aplicações de IIoT se justifica como uma solução eficiente e adaptável às necessidades industriais modernas. Ao oferecer flexibilidade, escalabilidade e custo-benefício, essa plataforma pode atender a um amplo espectro de aplicações industriais, ao mesmo tempo em que promove a inovação e a integração de novas tecnologias.

Müller et al. (2020) e Santos et al. (2020) defendem que o uso de hardware open source oferece vantagens significativas para as indústrias, permitindo que implementem soluções de automação customizadas e inovadoras. Essas soluções podem ser moldadas para atender às especificidades de cada operação, sem a necessidade de recorrer a fornecedores terceirizados para ajustes e atualizações constantes. O desenvolvimento de plataformas open source facilita ainda mais a integração de novos recursos e tecnologias, sem as limitações impostas por soluções

proprietárias. Isso significa que as empresas têm maior controle sobre seus sistemas, o que se traduz em maior agilidade e menores custos ao longo do tempo.

Portanto, o desenvolvimento de uma plataforma open source de automação e controle não apenas promove o acesso universal e a transparência, mas também proporciona uma personalização e flexibilidade incomparáveis. Esses aspectos são fundamentais para atender às necessidades e desafios da Indústria 4.0, onde a adaptabilidade e a inovação contínua são essenciais para a competitividade. Em um ambiente industrial cada vez mais dinâmico e interconectado, as empresas necessitam de soluções que possam evoluir rapidamente, sem comprometer a confiabilidade ou a eficiência operacional.

A proposta deste trabalho é exatamente desenvolver uma plataforma open source que atenda aos rigorosos requisitos do setor industrial, incluindo conectividade avançada e suporte a protocolos amplamente utilizados, como MQTT e Modbus. Dessa forma, busca-se fornecer uma solução eficiente, escalável e adaptável para plantas industriais, permitindo que as empresas implementem e mantenham sistemas de automação que atendam às suas necessidades específicas de forma contínua e sem as restrições das soluções fechadas tradicionais.

2.2. Objetivos

Este projeto de pesquisa objetivou o estudo e o desenvolvimento de uma Plataforma de Automação e Controle Open Source para IIoT com hardware e software open source, que possua um hardware flexível e customizável compatível com aplicações em plantas de manufatura, que esteja conforme a norma IEC 61131 e atenda às necessidades de conectividade para aplicações de Internet das Coisas Industrial.

2.3. Estrutura e Conteúdo

O presente trabalho está dividido em uma estrutura contendo cinco seções considerando esta introdução e desconsiderando as referências bibliográficas. O item 2 tratou de uma revisão bibliográfica que relacionava os principais tópicos acerca do tema como a Indústria 4.0, Internet das coisas industrial (IIoT) e Controladores. Além

disso, também foram apresentadas outras pesquisas e desenvolvimento de plataformas que já foram publicadas com base no objeto de estudo proposto.

O item 3 abordou o desenvolvimento da plataforma, fornecendo detalhes sobre a parte de hardware desenvolvida, onde são apresentados circuitos para proporcionar um entendimento completo da plataforma, bem como detalhes sobre o software utilizado. O capítulo 4 analisou os resultados dos testes realizados com a plataforma, incluindo sua submissão a um caso de estudo real de aplicação da IIoT.

Por fim, o item 5 destacou as conclusões derivadas do estudo realizado e aponta direções para trabalhos futuros. Essa estrutura visa oferecer uma visão abrangente do desenvolvimento da plataforma, seus resultados práticos e as perspectivas para o futuro do projeto.

3. REVISÃO BIBLIOGRÁFICA

3.1. Cenário atual da Indústria, IIoT e Controladores

A evolução industrial ao longo dos anos foi marcada por diferentes fases e culminou na quarta revolução industrial, também conhecida como Indústria 4.0. Este movimento, iniciado no início do século XXI, representou uma mudança significativa na forma como as indústrias operam. A terceira revolução industrial introduziu a eletrônica, telecomunicações e informática, permitindo automação total e a ascensão da robótica na indústria. No entanto, a Indústria 4.0 vai além, criando um cenário onde a interconexão é a chave. Nesse contexto, "redes inteligentes" emergem, conectando pessoas, dispositivos e máquinas digitais em uma teia complexa de comunicação, troca de dados e aprendizado mútuo (COLOMBO et al, 2021).

Essas "redes inteligentes" na Indústria 4.0 são frequentemente referidas como Sistemas Ciberfísicos (CPS). A crescente demanda por resultados eficientes e o rápido avanço tecnológico levam a um aumento expressivo no número de dispositivos e sistemas de processamento nas arquiteturas industriais. Essa evolução destaca a necessidade de criar CPS e componentes alinhados com as características essenciais da Indústria 4.0. Desta forma, torna-se necessário explorar constantemente novas abordagens, tecnologias e estratégias para maximizar o potencial dos CPS na Indústria 4.0 (PIVOTO et al, 2021).

A implementação da Indústria 4.0, apesar dos benefícios e avanços que pode trazer, de fato, enfrenta diversos desafios. Segundo Bajic (2021), entre os principais obstáculos estão as infraestruturas subdesenvolvidas em sistemas de fábricas, a alta complexidade nos sistemas e a falta de maturidade tecnológica. Em muitos casos, as instalações fabris existentes podem ter infraestruturas não preparadas para a integração completa da Indústria 4.0, exigindo investimentos significativos em atualizações. Além disso, Bajic destacou que os desafios na implementação da Indústria 4.0 ainda não foram totalmente superados, enfatizando a natureza em constante evolução dessa transformação. A colaboração entre diferentes setores da indústria, pesquisa e governos desempenha um papel crucial na superação desses obstáculos.

Juntamente com a Indústria 4.0, a Internet Industrial das Coisas (IIoT) despertou considerável interesse tanto na pesquisa acadêmica quanto na indústria. Empresas em todo o mundo estão implementando a IIoT em seus setores de produção, participando ativamente na revolução industrial em curso. A IIoT está redefinindo a conectividade e comunicação dos dispositivos de campo, resultando em maior eficiência e otimização de ativos em várias indústrias.

De acordo com Karmakar (2019), a evolução da IIoT é impulsionada por uma necessidade crescente de maior automação, comunicação eficiente, monitoramento em tempo real, autodiagnóstico e análise avançada. Esses elementos visam aumentar a produtividade e a eficiência de custos nas operações industriais. A IIoT proporciona uma infraestrutura que permite a coleta e a troca de dados entre dispositivos conectados, possibilitando uma tomada de decisões mais informada e rápida.

No entanto, conforme destacado por Khan (2020), mesmo atualmente, os sistemas da Internet Industrial das Coisas (IIoT) continuam a enfrentar desafios substanciais. Estes desafios abrangem áreas como interoperabilidade, segurança, escalabilidade, confiabilidade, gestão de recursos, segurança pública e gestão de dados. A interoperabilidade é crucial para a integração harmoniosa de diversos dispositivos e sistemas dentro da IIoT, enquanto a segurança é uma preocupação fundamental, dado o aumento da conectividade. A escalabilidade e confiabilidade dos sistemas são essenciais para lidar com o crescimento contínuo do número de dispositivos conectados. Esses desafios destacam a natureza dinâmica e em evolução da IIoT, requerendo esforços contínuos na pesquisa e implementação para superar essas barreiras e promover um ambiente IIoT mais robusto e eficiente.

Dentro do contexto da Indústria 4.0 e da Internet Industrial das Coisas (IIoT), os controladores desempenham um papel crucial. Eles são responsáveis por coletar dados, controlar os dispositivos dentro do sistema e atuam como ponte entre os componentes cibernéticos e físicos do sistema, garantindo coordenação e sincronização (JBAIR et al, 2018).

Os Controladores Lógicos Programáveis (CLPs) foram introduzidos para superar limitações dos sistemas tradicionais de controle industrial e utilizam unidades de software e hardware para controlar diversos sistemas de processos. Ao longo do tempo, esses controladores evoluíram, incorporando melhorias significativas em modularidade, capacidades computacionais e ferramentas de programação. Essas

melhorias expandiram consideravelmente a utilidade dos CLPs na tecnologia de controle numérico, permitindo sua adaptação e integração eficazes nos ambientes dinâmicos da Indústria 4.0 (NAMEKAR e YADAV, 2020).

No entanto, o atual ecossistema de Controladores Lógicos Programáveis enfrenta desafios significativos devido à crescente complexidade dos sistemas de automação e à necessidade de integração com a Internet e serviços sem fio. A tentativa de incorporar tecnologias modernas pode comprometer a segurança, tornando a verificação e simulação dos projetos de CLP complexas. Além disso, a natureza conservadora da indústria de automação industrial, guiada pela preocupação com a segurança e a estabilidade das linhas de produção, dificulta a introdução de inovações. O aumento da complexidade dos sistemas industriais e a demanda por personalização no mercado impõem pressão adicional sobre os CLPs para se adaptarem a essas exigências (SEHR et al, 2021).

A indústria, tradicionalmente focada na estabilidade e segurança das operações, enfrenta o dilema de equilibrar a busca por inovação com a manutenção dos padrões rigorosos de confiabilidade. Essa resistência à mudança e a adaptação mais lenta às tecnologias emergentes podem limitar a capacidade de aproveitar todo o potencial da revolução industrial em curso.

3.2. Plataformas de Automação

Com os avanços tecnológicos, surgem diversas alternativas que visam solucionar desafios e tornar a adoção e implementação de sistemas de automação mais acessível e ágil. Neste contexto, a presente revisão bibliográfica examina estudos e plataformas relevantes no campo da automação industrial e da Internet das Coisas Industrial (IIoT), com o intuito de estabelecer uma base analítica sólida para justificar o desenvolvimento da plataforma de automação proposta. Serão exploradas comparações entre soluções open source e proprietárias, além de análises de trabalhos acadêmicos que investigam alternativas de baixo custo e sistemas flexíveis voltados para controle e automação.

A Tabela 1 apresenta um resumo do foco de cada plataforma analisada, abordando suas configurações de hardware e software. Em seguida, realiza-se uma

discussão detalhada sobre as soluções observadas, ressaltando pontos chave para a análise comparativa.

Tabela 1 – Revisão Bibliográfica

Nome	Foco	Software	Hardware
Revolution PI	PLC baseado em Raspberry Pi	Possui imagem para utilização do software de sua escolha	Fechado, dependência de módulos da empresa
P1AM	PLC baseado em Arduino	ProductivityBlocks ou Arduino IDE	Fechado, dependência de módulos da empresa
Industrial Shields	PLC baseado em Arduino, ESP32 e Raspberry Pi	Arduino IDE ou Python para baseado em Raspberry Pi	Fechado, dependência de módulos da empresa
Arduino Industrial 101	Placa Industrial da Arduino	Arduino IDE	Apenas uma placa, falta funcionalidades
Arduino Opta Lite	PLC da Arduino	Arduino IDE ou Software pago para IEC 61131-3 (Potenta)	Schematic falta informação, não é replicável
Andrei et al (2020)	Transformação de um Raspberry Pi em um PLC	Aberto via OpenPLC	Não disponibiliza o desenvolvimento
Rout et al (2024)	Projeto de um controlador de baixo custo	Não apresenta software	Apenas protótipo, não possui uma placa e não disponibiliza os arquivos
Fernandez-Rodriguez et al (2021)	Plataforma aberta para controle industrial	Não disponível	Hardware modular e Aberto no git hub
OpenPLC	Linguagens da IEC 61313-3	100% aberto	Sem solução de Hardware

Vieira et al (2020)	Controlador industrial baseado em Raspberry Pi	Não possui desenvolvimento	Somente entradas e saídas digitais, não aberto

Fonte: Elaborado pelo autor

Ao analisar as referências selecionadas, observa-se uma lacuna significativa na disponibilidade de soluções 100% open source que ofereçam tanto hardware quanto software abertos, além de serem adequadas para aplicações industriais. Muitas plataformas de mercado que se apresentam como open source ainda mantêm limitações de hardware, restringindo as possibilidades de personalização e adaptações que sejam realmente flexíveis para o usuário final. Essas limitações refletem antigas práticas da indústria, onde o hardware oferecido não permite a customização necessária para atender às demandas específicas de cada aplicação.

No contexto acadêmico, as alternativas propostas, embora frequentemente inovadoras, ainda carecem da robustez e confiabilidade necessárias para suportar as rigorosas exigências do setor industrial. Além disso, muitas dessas soluções baseiam-se em placas como a Raspberry Pi, o que, embora forneça um ambiente de desenvolvimento rico e acessível para prototipagem, eleva significativamente o custo, tornando-se cerca de 10 vezes mais dispendioso em comparação com plataformas baseadas em microcontroladores. Esta dependência de hardware mais caro limita a viabilidade de implementação em escala e contrasta com a demanda por soluções econômicas e de alto desempenho que possam ser aplicadas na Indústria 4.0.

Assim, este projeto visa preencher essa lacuna, propondo uma solução integralmente open source que atenda às necessidades de controle de processos de manufatura e possua a conectividade e flexibilidade indispensáveis para aplicações industriais e de IIoT. A plataforma desenvolvida busca não apenas oferecer um sistema de automação economicamente viável, mas também adaptável e em linha com as exigências da Indústria 4.0.

4. DESENVOLVIMENTO

4.1. Proposta do Trabalho

O objetivo central deste trabalho é desenvolver uma plataforma de automação e controle open source que seja altamente versátil e capaz de atender às demandas específicas da Internet das Coisas Industrial. A proposta visa criar um hardware que, além de atender aos requisitos tradicionais de controle de processos industriais, seja compatível com tecnologias de conectividade e suporte a múltiplos protocolos industriais. Dessa forma, busca-se atender à crescente demanda por soluções flexíveis e integradas em ambientes industriais, onde a eficiência, a escalabilidade e a facilidade de implementação são essenciais.

Para reforçar o compromisso com a comunidade de desenvolvedores e com a transparência do projeto, toda a documentação e os materiais técnicos desenvolvidos ao longo deste trabalho foram disponibilizados em um repositório open source no GitHub. Essa iniciativa visa facilitar o acesso a informações detalhadas e promover a colaboração contínua. Os interessados podem acessar o repositório em: https://github.com/gasiepgodoy/Open-Source-PLC onde encontrarão todos os arquivos relevantes, incluindo esquemáticos, códigos-fonte, e instruções de implementação, com o objetivo de incentivar melhorias e adaptações de acordo com as necessidades de cada aplicação

Para o desenvolvimento do hardware proposto, foi necessário estabelecer um conjunto estruturado de requisitos que assegurasse tanto a abrangência funcional quanto a viabilidade técnica da solução. Esses requisitos foram definidos com base nas necessidades operacionais observadas em ambientes industriais e nas demandas de flexibilidade típicas de contextos acadêmicos e de pesquisa aplicada. A proposta visa garantir que a plataforma seja capaz de atender a diferentes aplicações em controle e automação, mantendo compatibilidade com dispositivos e protocolos amplamente utilizados. Para fins de organização e clareza, os requisitos foram classificados em duas categorias complementares: requisitos funcionais, que descrevem o comportamento esperado da plataforma e suas capacidades operacionais, e requisitos técnicos, que detalham as características físicas, elétricas e estruturais necessárias para garantir a confiabilidade, expansibilidade e compatibilidade da solução. Ambos os conjuntos de requisitos são apresentados a seguir.

4.1.1. Requsitos Funcionais

Os requisitos funcionais correspondem às funcionalidades e comportamentos mínimos que a plataforma deve apresentar para cumprir os objetivos definidos no projeto. Em sistemas de automação, esses requisitos estão diretamente relacionados à capacidade do hardware e do software de realizar tarefas como leitura e controle de sinais, comunicação entre dispositivos e execução de lógicas de controle em conformidade com normas técnicas. A definição clara desses requisitos é essencial para garantir a aplicabilidade, a reprodutibilidade e a expansão do sistema em diferentes contextos, sejam eles industriais, educacionais ou de pesquisa. Com base nessas premissas, a plataforma proposta deve atender aos seguintes requisitos funcionais:

- **Entradas e saídas industriais:** A plataforma deve dispor de entradas e saídas digitais e analógicas, fundamentais para o controle e o monitoramento de processos industriais. A presença desses canais de I/O permite a conexão direta com sensores, atuadores e dispositivos de campo, viabilizando aplicações práticas em automação de máquinas, processos e sistemas embarcados.
- **Controle de atuadores e leitura de sensores:** É requisito a inclusão de saídas PWM, que possibilitam o controle preciso de atuadores como motores elétricos. Além disso, a plataforma deve suportar a leitura de sinais de encoders, permitindo a detecção de posição e velocidade em tempo real, o que é especialmente útil em sistemas com retroalimentação (feedback), como os que utilizam controle PID.
- **Interfaces de comunicação:** A plataforma deve ser compatível com os principais padrões de comunicação utilizados na indústria, incluindo RS-232, RS-485 e Ethernet (W5500). Esses recursos garantem conectividade com equipamentos legados e modernos, além de permitir a integração com redes industriais e sistemas baseados em Internet das Coisas Industrial (IIoT), como aqueles que utilizam protocolo MQTT.
- **Processamento embarcado:** O núcleo computacional da plataforma deve utilizar um microcontrolador com suporte a todos os periféricos listados, com desempenho suficiente para operações em tempo real. A escolha por

um modelo com baixo custo, boa documentação e comunidade ativa é estratégica, pois garante viabilidade econômica sem comprometer a expansão, a manutenção e a customização do projeto.

- **Compatibilidade com linguagens normatizadas:** A plataforma deve oferecer suporte à norma IEC 61131-3, que define linguagens como Ladder Diagram (LD), Structured Text (ST) e Function Block Diagram (FBD), padronizando o desenvolvimento de aplicações industriais com foco em confiabilidade, segurança e legibilidade.
- **Alternativa moderna de programação:** Além das linguagens da IEC, a plataforma deve oferecer suporte a uma linguagem de alto nível moderna e de fácil uso, como o MicroPython. Essa alternativa permite maior flexibilidade, reduz a curva de aprendizado e facilita a integração com bibliotecas e ferramentas contemporâneas, como dashboards em Node-RED e sistemas de telemetria via MQTT.

4.1.2. Requisitos Técinicos

Os requisitos técnicos definem as características físicas, elétricas e estruturais que a plataforma deve apresentar para garantir seu funcionamento adequado em ambientes industriais e laboratoriais. Esses requisitos são essenciais para assegurar a compatibilidade com sistemas existentes, a confiabilidade da operação e a escalabilidade da solução em diferentes aplicações. A seguir, são apresentados os principais requisitos técnicos adotados para o desenvolvimento da plataforma proposta:

- **Alimentação padrão industrial:** A plataforma deve operar com tensão de alimentação de 24V em corrente contínua, facilitando sua integração em painéis industriais, onde esse padrão é amplamente utilizado. Essa escolha simplifica a instalação e evita a necessidade de fontes externas adicionais ou adaptações complexas.
- **Configuração de entradas e saídas (I/Os):** O sistema deve conter, no total, 19 entradas e 12 saídas, distribuídas da seguinte forma:
    - **12 entradas digitais de 24V**, isoladas por optoacopladores, garantindo proteção contra interferências e maior confiabilidade na leitura de sinais industriais.

- **4 entradas analógicas**, com conversores ADC programáveis, capazes de operar no intervalo de -10 V a +10 V, atendendo às faixas típicas de sensores industriais analógicos.
- **Entrada dedicada à leitura de encoders,** permitindo a detecção precisa de movimento, posição e velocidade.
- **8 saídas digitais acionadas por relés,** compatíveis com acionamentos industriais como contatoras, válvulas e sinalizadores.
- **4 saídas PWM**, utilizadas para controle de dispositivos que demandam modulação por largura de pulso, como motores e drivers.

- **Interfaces de comunicação industrial:** A plataforma deve dispor de múltiplas interfaces de comunicação, incluindo Ethernet (W5500), RS-232 e RS-485, compatíveis com protocolos industriais amplamente utilizados, como Modbus RTU e MQTT. Essa diversidade garante a interoperabilidade da plataforma com sistemas de supervisão, CLPs e redes IIoT.
- **Serialização de I/Os:** Um diferencial técnico importante da plataforma é a adoção da técnica de serialização das entradas e saídas. Essa abordagem permite que os sinais digitais e analógicos sejam manipulados de forma modular por meio de barramentos seriais, como SPI, possibilitando a expansão e reconfiguração das I/Os sem a necessidade de alteração do hardware principal. A serialização também facilita o processo de adaptação da plataforma a diferentes cenários de automação, contribuindo para sua escalabilidade e modularidade.

4.2. Projeto de Hardware

4.2.1 Projeto Eletroeletrônico

Após a definição dos requisitos operacionais e técnicos, deu-se início ao desenvolvimento do projeto eletroeletrônico da placa, contando com o apoio da FACTS Engineering, uma empresa especializada em desenvolvimento de hardware para automação, que possui soluções no mercado, como a linha *ProductivityOpen*,

premiada na 33ª edição da *Annual Plant Engineering Product of the Year* nos Estados Unidos (FACTS, 2023).

Para assegurar a confiabilidade e robustez do projeto, todos os circuitos foram desenvolvidos com base em produtos testados e validados pela FACTS, sendo o principal deles o P1AM (AUTOMATION, 2023). A escolha desses componentes consolidados permite que o projeto herde características de robustez e segurança, essenciais em ambientes industriais.

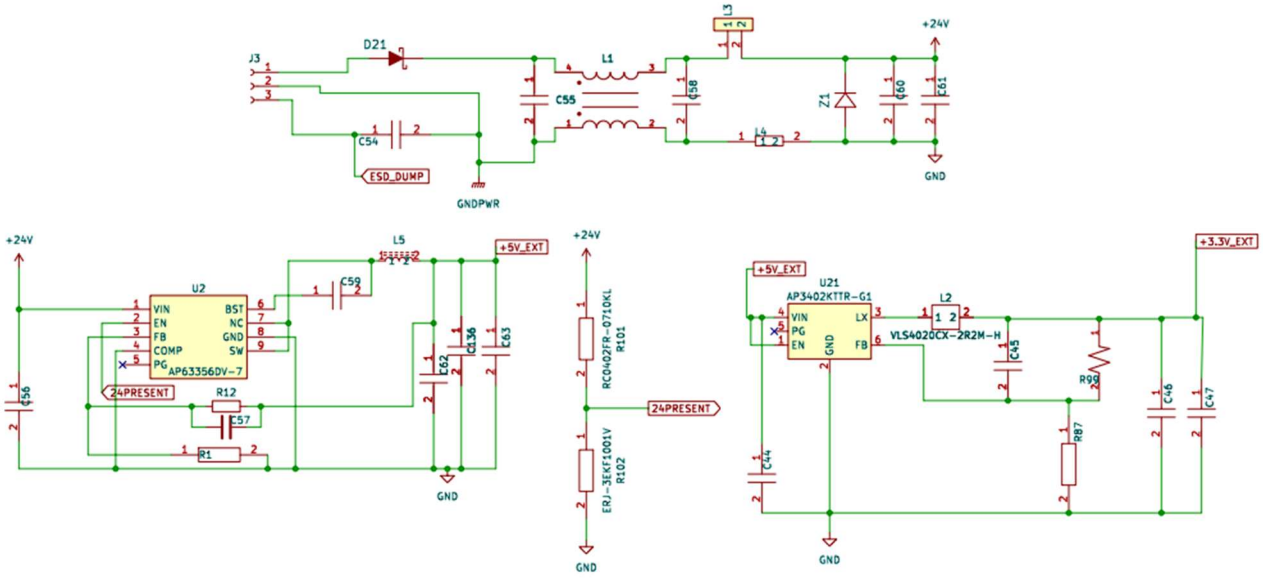
O desenvolvimento do hardware foi realizado com o KiCad, um software gratuito e open source amplamente utilizado para prototipagem eletrônica, alinhando-se ao compromisso do projeto com a acessibilidade e o baixo custo. O design dos circuitos foi dividido em 8 módulos, visando facilitar futuras atualizações e permitir que cada circuito possa ser modificado de forma independente. Essa modularidade traz flexibilidade e contribui para a escalabilidade do sistema, importante para atender às necessidades dinâmicas de aplicações industriais.

4.2.1.1 Alimentação Elétrica

As instalações industriais atuais geralmente adotam uma alimentação padrão de 24V em seus painéis, sendo assim, a plataforma foi projetada levando em consideração essa tensão de alimentação. No entanto, surge um desafio quando se considera que muitos componentes eletrônicos, como controladores e circuitos integrados (CI), operam com alimentação de 3.3V ou 5V. Assim, é necessária a implementação de conversores de tensão para adequar a alimentação aos requisitos específicos de cada componente na placa.

Para resolver essa discrepância de tensões, foram incorporados dois conversores de tensão no projeto. O primeiro conversor, o AP63356DV-7, desempenha a função de converter a tensão de 24V para 5V, proporcionando a alimentação adequada para os componentes que operam com essa faixa de tensão. Em seguida, o segundo conversor, o AP3402KTTR-G1, realiza a conversão de 5V para 3.3V, assegurando que os componentes que requerem essa tensão específica recebam a alimentação correta. A Figura 1, oferece uma representação visual dos circuitos desenvolvidos para essas conversões de tensão.

Figura 1 - Circuitos de Alimentação



Fonte: Elaborado pelo autor

4.2.1.2 Entradas Digitais

Uma das principais características de uma plataforma é a facilidade para acrescentar e retirar funcionalidades, desta forma é necessário que a modificação do número de entradas não seja de alta complexidade. Juntamente com esta característica, o número limitado de pinos do controlador torna-se um empecilho para um número elevado de entradas e conforme definido anteriormente a plataforma irá conter na sua versão base 12 entradas digitais.

Para superar esses desafios, a serialização das entradas se revela como uma solução estratégica e eficiente. Ao optar por esse método, além de garantir a facilidade na adição e remoção de funcionalidades na plataforma, contorna-se a limitação de pinos do controlador. A serialização é uma alternativa muito utilizada em controladores modulares, onde o número de entradas não é previamente definido e varia de acordo com a quantidade de módulos conectados, pois oferece a flexibilidade necessária para lidar com um número maior de entradas sem comprometer a complexidade do processo.

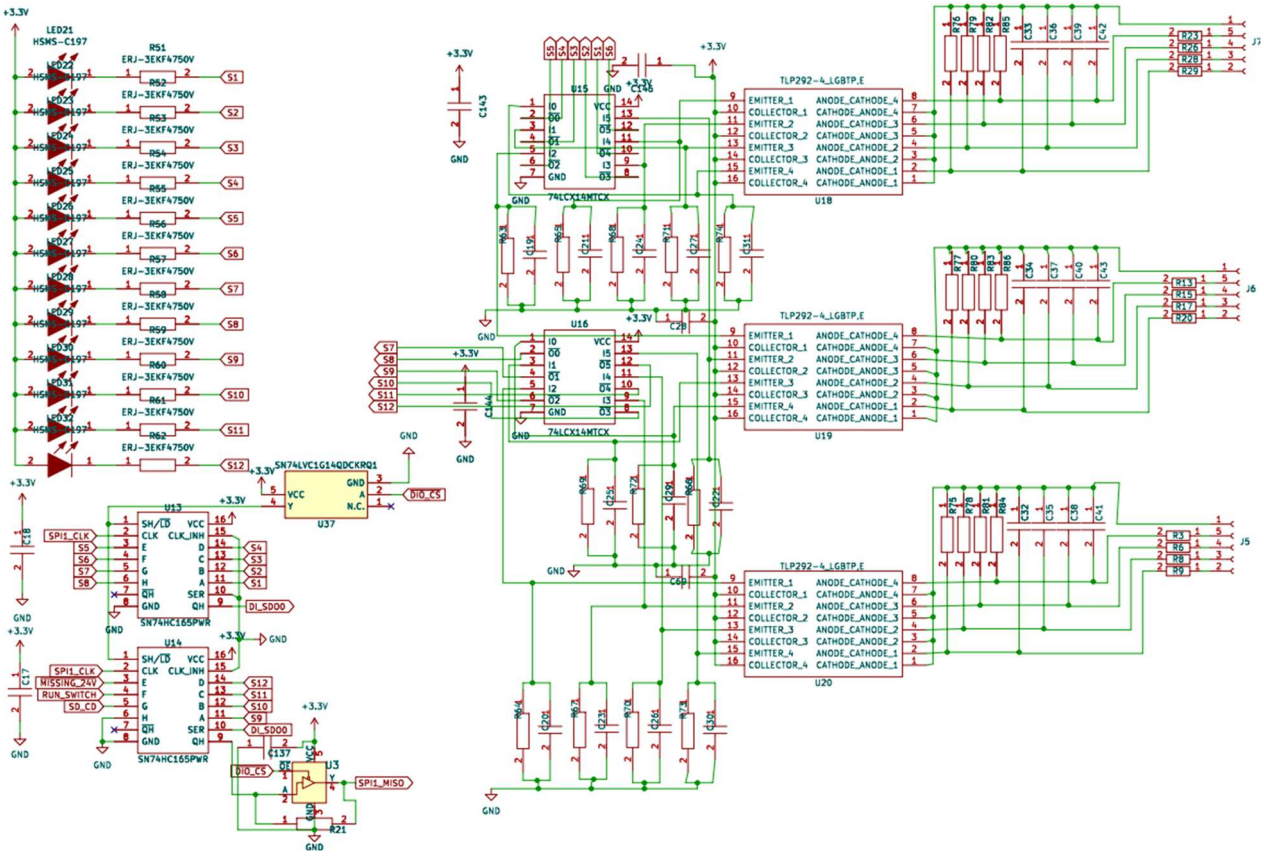
A escolha da serialização traz consigo a necessidade de selecionar um protocolo adequado, e nesse contexto, optou-se pela utilização do protocolo SPI (*Serial Peripheral Interface*). O SPI é uma forma de comunicação serial síncrona

especialmente projetada para curtas distâncias. Sua dinâmica envolve uma linha dedicada para sincronizar os dados entre o dispositivo que envia e o dispositivo que recebe. Além disso, o SPI opera em um modo mestre-escravo, permitindo a presença de apenas um mestre (controlador da comunicação) enquanto pode haver vários escravos (dispositivos controlados).

Este protocolo oferece diversas vantagens cruciais. A capacidade de comunicação em alta velocidade do SPI o torna particularmente apropriado para aplicações que exigem transferências rápidas de dados. A baixa sobrecarga do protocolo é uma característica fundamental, resultando em latência mínima e transferência eficiente de dados, eliminando a necessidade de esquemas de endereçamento complexos. Outro ponto relevante é a ampla adoção do SPI na indústria, com suporte integrado em muitos microcontroladores e dispositivos periféricos. Essa prevalência facilita a identificação de componentes compatíveis e contribui para a eficiência no desenvolvimento de soluções. Em resumo, a escolha do SPI se fundamenta em sua eficácia, velocidade, baixa sobrecarga e integração generalizada na indústria, consolidando-o como uma escolha sólida para a serialização em diversos contextos aplicados.

Para a aquisição das entradas digitais, utilizou-se o chip 74HC165PWR, que opera no modo *Parallel-in Serial-out* (entrada paralela e saída serial). Cada chip possui 8 entradas de sinal, desta forma para atender ao requisito de 12 entradas, utilizou-se de 2 desses chips. Adicionalmente, foram incorporados outros chips no circuito para aprimorar a proteção e funcionalidade do sistema. O TLP292-4 (LGBTR,E) foi empregado para proporcionar isolamento elétrico entre diferentes partes do circuito, garantindo uma maior robustez contra interferências e eventuais variações de voltagem. O 74LCX14MTCX também foi utilizado, desempenhando um papel fundamental na proteção do circuito contra transientes e picos de voltagem. A Figura 2 apresenta o circuito desenvolvido para esta funcionalidade.

Figura 2 - Circuito Entradas Digitais



Fonte: Elaborado pelo autor

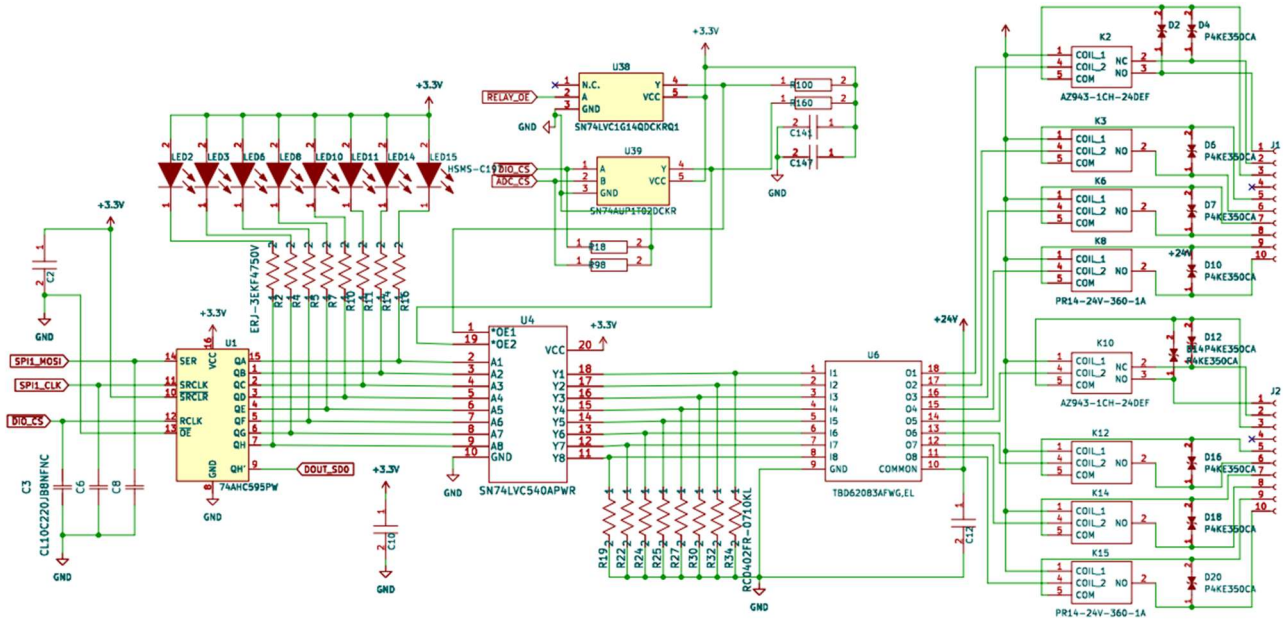
4.2.1.3 Saídas Digitais

Para implementar as saídas digitais na plataforma, conforme definido anteriormente, foi escolhida a utilização de relés, totalizando 8 saídas na versão base. Assim como nas entradas digitais, a serialização via SPI foi adotada para alcançar os objetivos previamente estabelecidos. A escolha de utilizar relés nas saídas digitais proporciona uma solução robusta e flexível para o controle de dispositivos externos, enquanto a serialização via SPI contribui para a eficiência geral do sistema, garantindo uma comunicação confiável e coordenada entre os componentes.

O chip escolhido para a implementação das saídas foi o 74AHC595PW, que opera como um *Serial-in Parallel-out* (entrada serial e saída em paralelo). Cada chip dessa série possui 8 saídas paralelas, necessitando de apenas um chip para atender aos requisitos. Juntamente com o 74AHC595PW, outros chips foram incorporados para a proteção e funcionalidade do circuito. O TBD62083AFWG,EL desempenha um

papel crucial no circuito, fornecendo isolamento e acionamento eficiente para os relés associados às saídas digitais. O SN74LVC540APWR também foi utilizado para garantir a integridade do sinal e proteger o circuito contra variações de voltagem. A Figura 3 pode ser consultada para visualizar o circuito desenvolvido para essa funcionalidade específica.

Figura 3 - Circuito Saídas a Relé



Fonte: Elaborado pelo autor

4.2.1.4 PWM

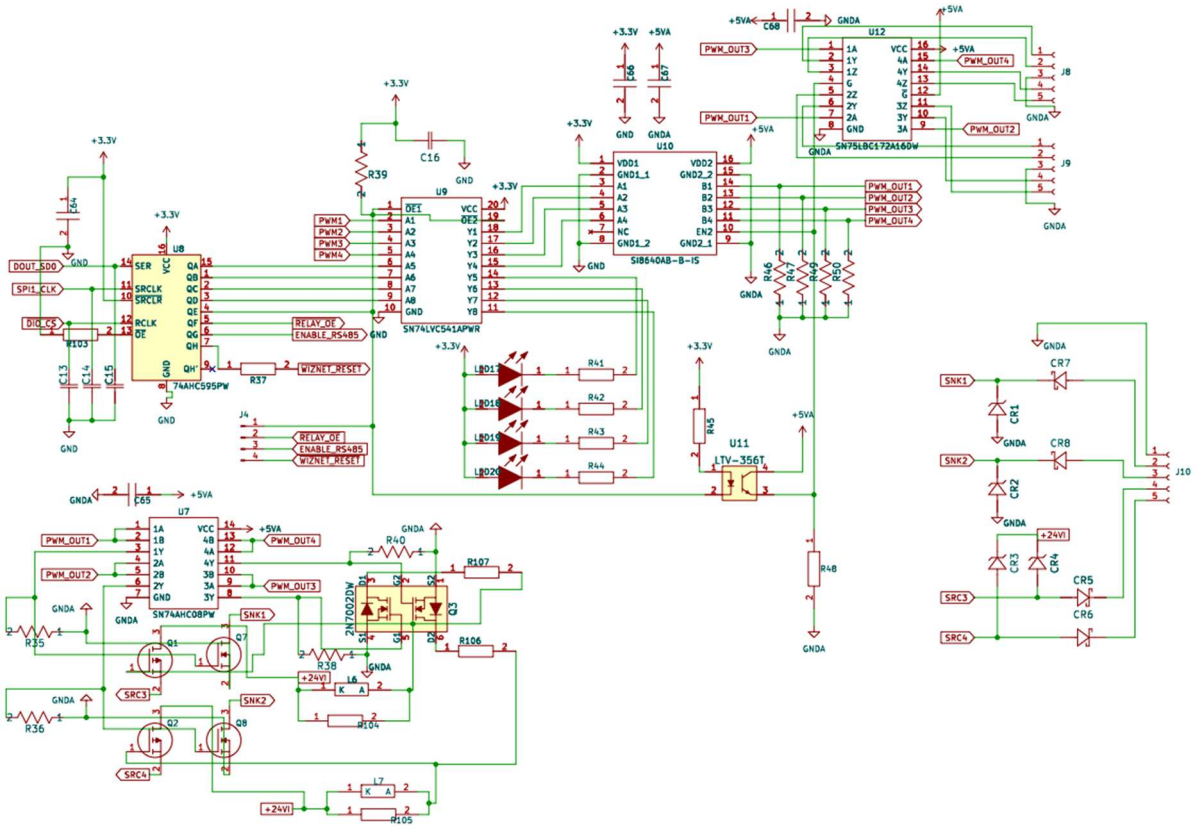
A plataforma traz como diferencial a presença de 4 saídas PWM (*Pulse Width Modulation*) destacando o compromisso com a inovação e a adaptabilidade da plataforma às demandas específicas da indústria. O PWM é amplamente utilizado na indústria devido à sua capacidade de controlar a potência de saída de dispositivos e motores de maneira eficiente, desta forma, essa funcionalidade específica proporciona maior flexibilidade no controle de sistemas e equipamentos.

O controlador escolhido para a plataforma já incorpora canais PWM internos, simplificando a implementação do circuito já que cada saída PWM pode ser diretamente controlada pelo controlador, eliminando a necessidade de componentes adicionais para gerar os sinais PWM. Para garantir a compatibilidade e a ampliação do range de saída das saídas PWM, foi adicionado o chip SI8640AB-B-IS para realizar o isolamento e permitir uma variação de 0 a 5V na saída, em vez do range padrão de

0 a 3.3V do controlador. Além disso, o SN75LBC172A16DW foi utilizado para possibilitar sinais inversos (PWM + e PWM -), expandindo as opções de controle.

Para uma implementação mais completa, foram incorporados dois circuitos de Sink (Dreno) para os PWM 1 e 2, e dois circuitos de Source (Fonte) para os PWM 3 e 4. Esses circuitos adicionais aprimoram a capacidade de acionamento das saídas PWM, oferecendo flexibilidade adicional para lidar com diferentes cargas. A Figura 4 fornece uma representação visual do circuito para essa funcionalidade específica.

Figura 4 - Circuito PWM



Fonte: Elaborado pelo autor

A presença do chip SN74LVC541APWR na Figura 4 indica sua utilização como um componente de proteção no circuito e está sendo controlado por uma das saídas do chip 74AHC595PW. Essa configuração foi escolhida estrategicamente para permitir a interrupção da utilização das saídas PWM através da comunicação serial, mesmo que essas saídas não estejam diretamente serializadas junto com as outras entradas e saídas. A capacidade de controlar as saídas PWM por meio da comunicação serial adiciona uma camada de flexibilidade ao sistema, possibilitando

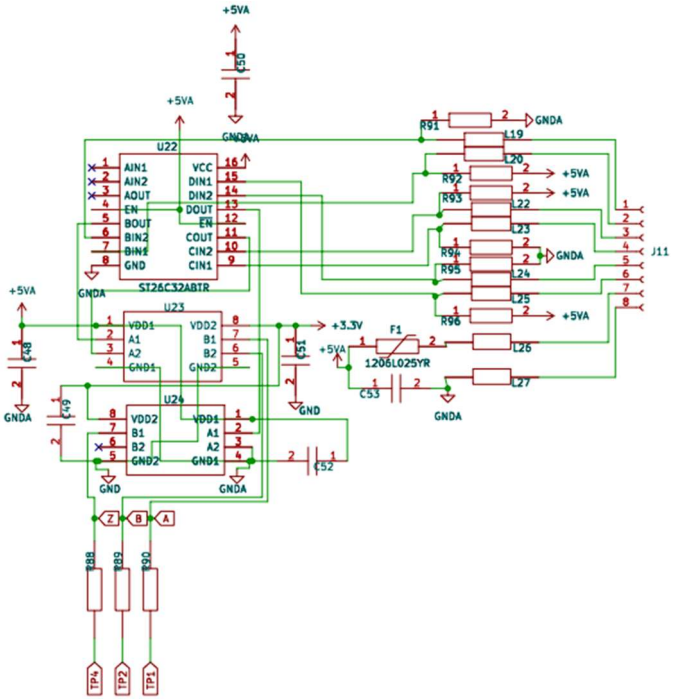
a ativação ou desativação dessas saídas conforme necessário. Isso pode ser útil em situações em que é desejável modificar dinamicamente a operação do sistema, otimizando o consumo de energia ou adaptando-se a diferentes condições de operação.

4.2.1.5 Encoder

A inclusão de uma entrada dedicada para encoders na plataforma é uma funcionalidade diferencial significativa. Os encoders são amplamente empregados em diversas aplicações industriais, desempenhando um papel crucial no controle de velocidade e posicionamento em sistemas automatizados. No caso específico da plataforma, a abordagem adotada para lidar com a leitura dos pulsos do encoder é direta e eficiente, sendo assim, as entradas do encoder estão conectadas diretamente aos pinos do controlador. Essa escolha sugere uma estratégia otimizada para atender às altas velocidades e à necessidade de leitura rápida dos pulsos, aproveitando a capacidade interna do controlador para processar esses sinais de maneira eficaz.

Para garantir a integridade do circuito e proporcionar isolamento necessário, foram incluídos os chips ST26C32ABTR e SI8620BB-B-ISR. O ST26C32ABTR desempenha um papel fundamental na comunicação de dados, enquanto o SI8620BB-B-ISR oferece isolamento, contribuindo para a segurança e robustez do circuito. A Figura 5 fornece uma representação visual do circuito desenvolvido para essa funcionalidade específica, destacando a simplicidade da conexão direta dos pinos do encoder ao controlador e a presença dos componentes de isolamento para garantir um desempenho confiável em ambientes industriais.

Figura 5 - Circuito Entrada Encoder



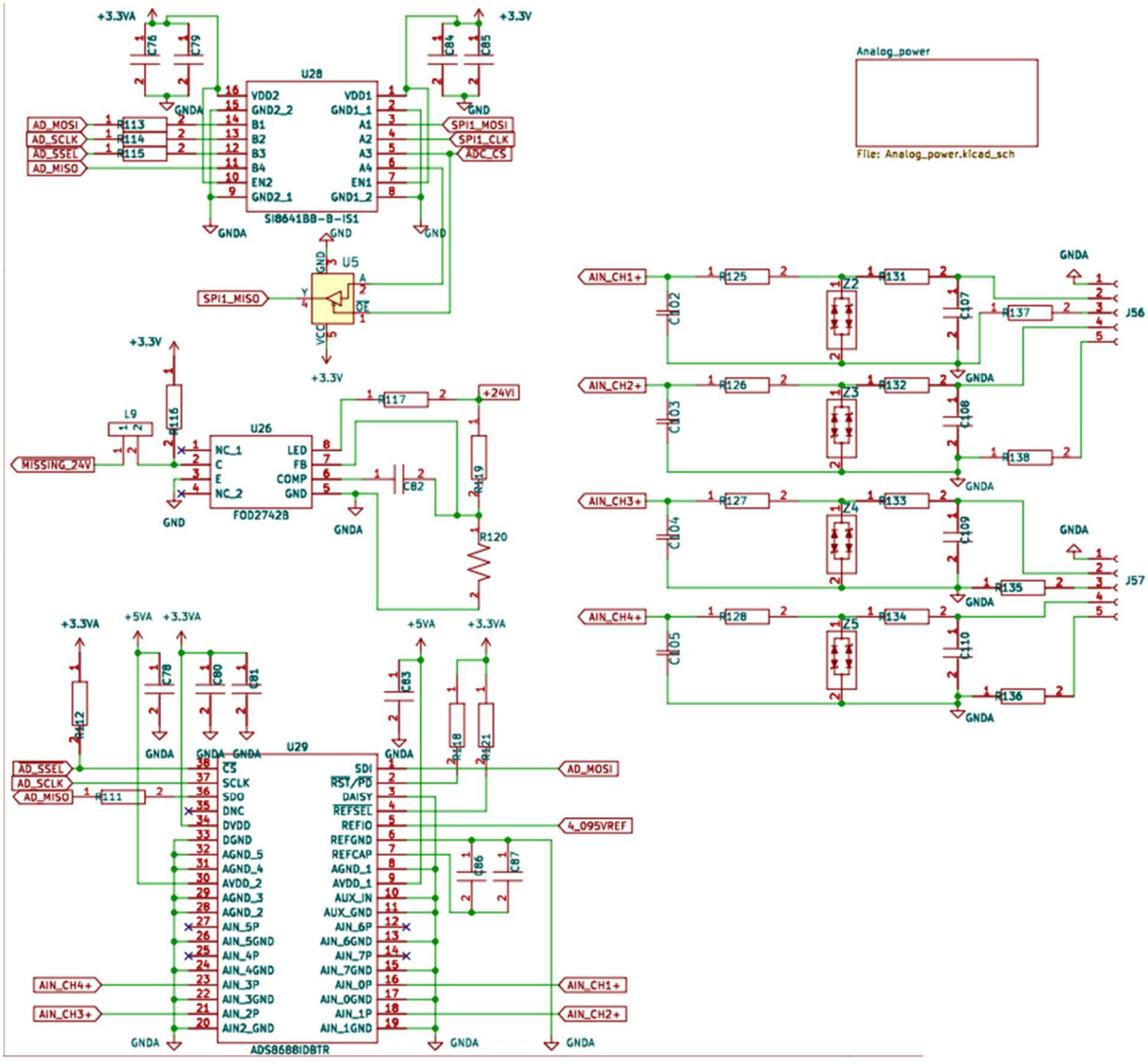
Fonte: Elaborado pelo autor

4.2.1.6 Entradas Analógicas

Conforme definido anteriormente, 4 entradas analógicas estão presentes na plataforma. Desta forma seria possível utilizar-se do conversor analógico-digital (ADC) interno do controlador. Contudo, optou-se pela incorporação de um chip ADS8688IDBTR (*analog-in serial-out*) na plataforma, de modo a garantir a busca por flexibilidade, robustez no design, além de permitir a utilização da serialização via SPI. Este chip ADC de 16 bits de resolução permite uma variação de tensão nas suas entradas de -10V a 10V, oferecendo precisão na leitura de sinais analógicos.

A Figura 6 fornece uma representação visual do circuito desenvolvido para essa funcionalidade.

Figura 6 - Circuito Entradas Analógicas



Fonte: Elaborado pelo autor

Juntamente com o chip ADC, foi projetado um circuito para a entrada que possibilita a leitura tanto de variação de tensão quanto de variação de corrente. Essa versatilidade é crucial em ambientes industriais, onde diferentes tipos de sensores podem estar presentes e precisam ser integrados de maneira eficiente. Para garantir o isolamento do circuito analógico e evitar interferências indesejadas no domínio digital, foi empregado o SI8641BB-B-IS1. Este chip desempenha um papel vital na prevenção de tensões analógicas prejudiciais ao circuito digital, assegurando uma leitura precisa e confiável dos sinais analógicos.

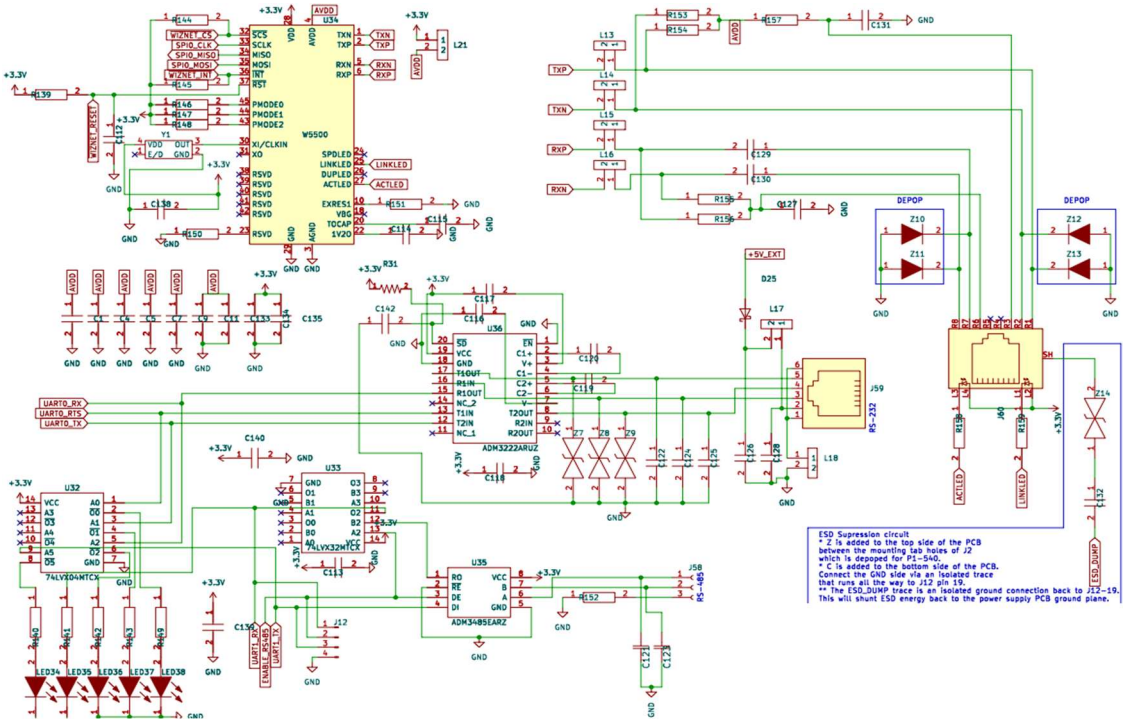
4.2.1.7 Comunicação

A inclusão de três opções de comunicação (Ethernet, RS-485 e RS-232) na versão inicial da plataforma reflete uma abordagem abrangente para atender às variadas necessidades de comunicação na indústria e em aplicações de IIoT. Cada uma dessas interfaces desempenha um papel crucial em diferentes cenários industriais.

Para possibilitar a comunicação via Ethernet, foi utilizado o chip W5500. Este chip é amplamente empregado no desenvolvimento de hardware e é conhecido por oferecer uma solução eficaz para conectar dispositivos à rede Ethernet. A presença dessa opção de comunicação é essencial para integrar a plataforma em redes industriais e permitir a comunicação eficiente com outros dispositivos na infraestrutura.

A comunicação RS-232 é uma interface serial comum na indústria. O chip ADM3222ARUZ foi escolhido como um CI de interface para habilitar a comunicação RS-232. A comunicação RS-485 é frequentemente utilizada em ambientes industriais devido à sua capacidade de suportar comunicação em longas distâncias e com vários dispositivos. O chip ADM3485EARZ foi selecionado como um CI de interface para possibilitar a comunicação RS-485. A Figura 7 oferece uma representação visual do circuito desenvolvido para suportar essas opções de comunicação.

Figura 7 - Circuitos de Comunicação



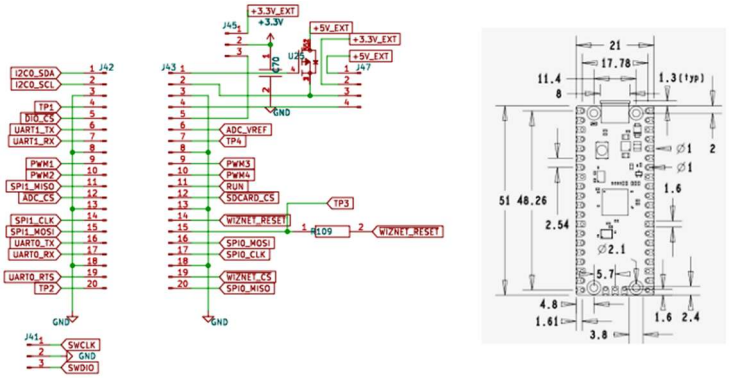
Fonte: Elaborado pelo autor

4.2.1.8 Unidade de processamento

Para o microcontrolador, após uma análise comparativa de diversos modelos, definiu-se pelo RP2040, desenvolvido pela Raspberry. O RP2040 conta com um processador dual-core Cortex-M0+, 264 KB de RAM, suporte para até 16 MB de memória Flash, 30 pinos GPIO, além de periféricos como 2 UARTs, 2 controladores SPI, 2 controladores I2C, 16 canais PWM e 8 máquinas de estado PIO. Essa escolha se deu pela combinação de alto desempenho e baixo custo, facilidade de uso, ampla documentação, suporte a várias linguagens e IDEs, baixo consumo energético e pelo fato de ser um microcontrolador open source, fatores que alinham o RP2040 aos objetivos e especificações do projeto.

Este controlador está presente nas placas Raspberry Pi Pico e Raspberry Pi Pico W. Desta forma, de modo a simplificar o desenvolvimento e futuras alterações, optou-se por utilizar a placa Raspberry Pi Pico ao invés do RP2040 direto na placa pois assim é necessário apenas os pinos na placa como mostra a Figura 8.

Figura 8 - Esquemático do controle

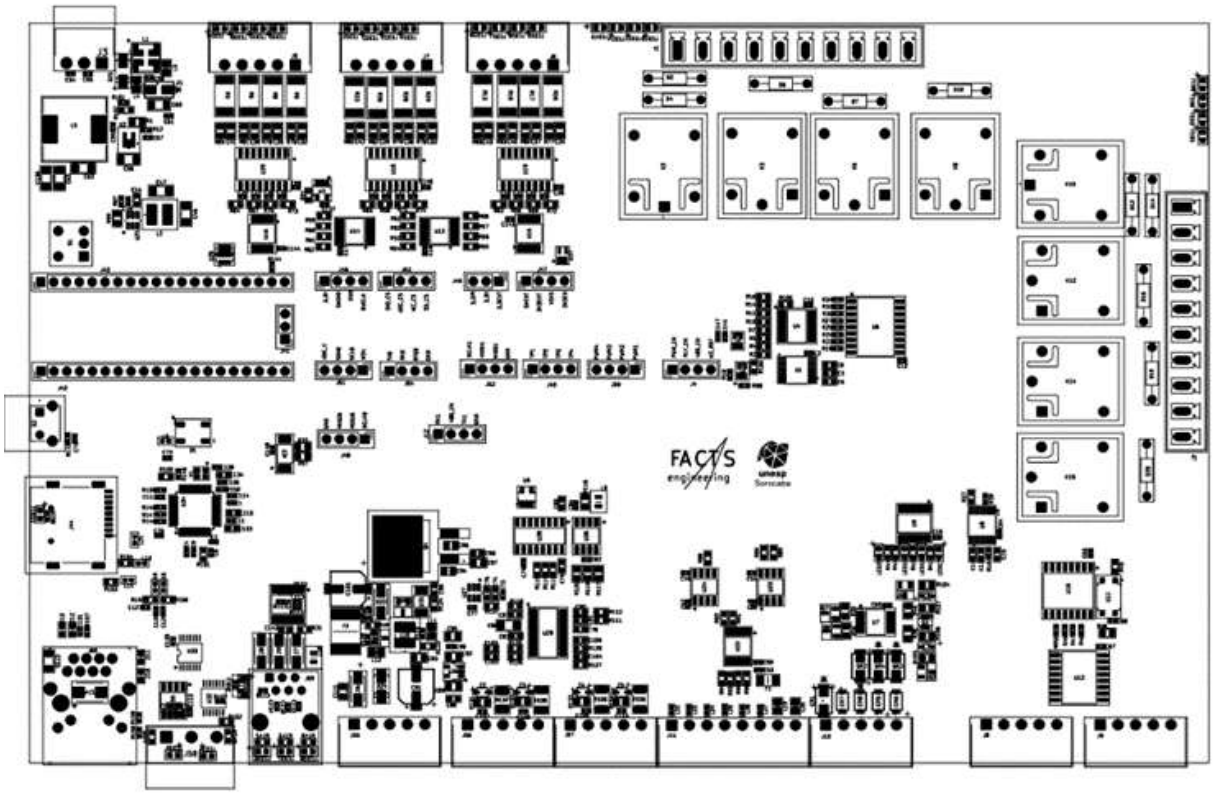


Fonte: Elaborado pelo autor

4.2.2. Projeto da Placa

O projeto da placa eletrônica iniciou-se com o desenvolvimento do layout e posicionamento dos componentes conforme os circuitos desenvolvidos anteriormente. Para isso, pensou-se na distribuição mais conveniente, de modo que, fosse possível ter fácil acesso a todos os componentes eletrônicos presentes no projeto. Desta forma, obteve-se o layout apresentado na Figura 9.

Figura 9 - Layout PCB

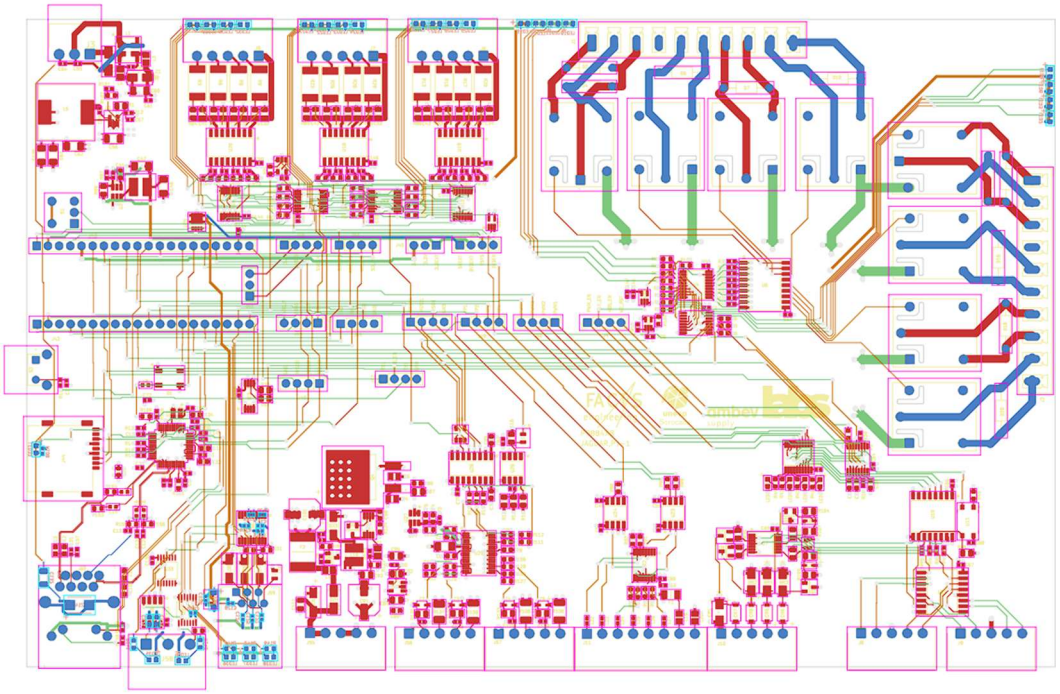


Fonte: Elaborado pelo autor

Nota-se um tamanho e forma não convencional ao de um controlador industrial padrão, porém isso se deve ao fato de que uma plataforma não representa a versão final de um produto e tem sua distribuição feita de maneira a facilitar testes e validações. Após o posicionamento dos componentes, iniciou-se o desenvolvimento das trilhas e dos layers da placa.

Devido à complexidade do projeto, optou-se por uma placa de 4 layers de modo a garantir o correto funcionamento de todos os circuitos e evitar problemas de interferências. Desta maneira obteve-se o resultado mostrado na Figura 10.

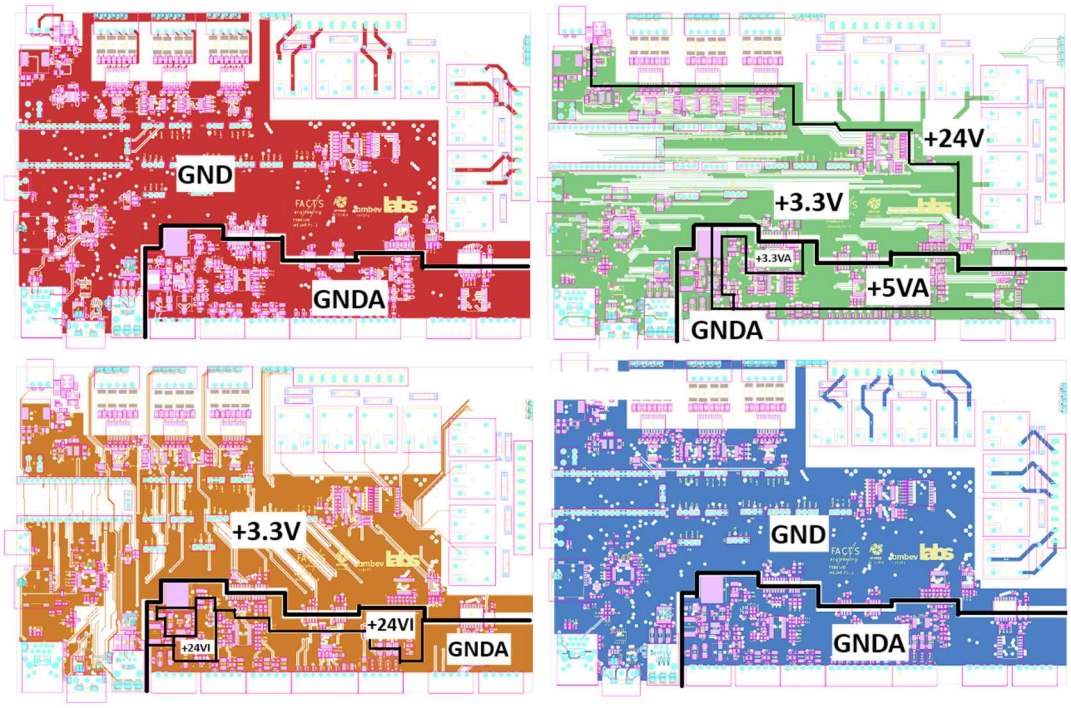
Figura 10 - Trilhas de cobre PCB



Fonte: Elaborado pelo autor

Em cada layer foram estipuladas zonas de cobre que ajudam a garantir o correto funcionamento dos circuitos e diminuem as necessidades de trilhas e conexões por toda a placa. A Figura 11 exemplifica as zonas presentes.

Figura 11 – Layouts PCB



Fonte: Elaborado pelo autor

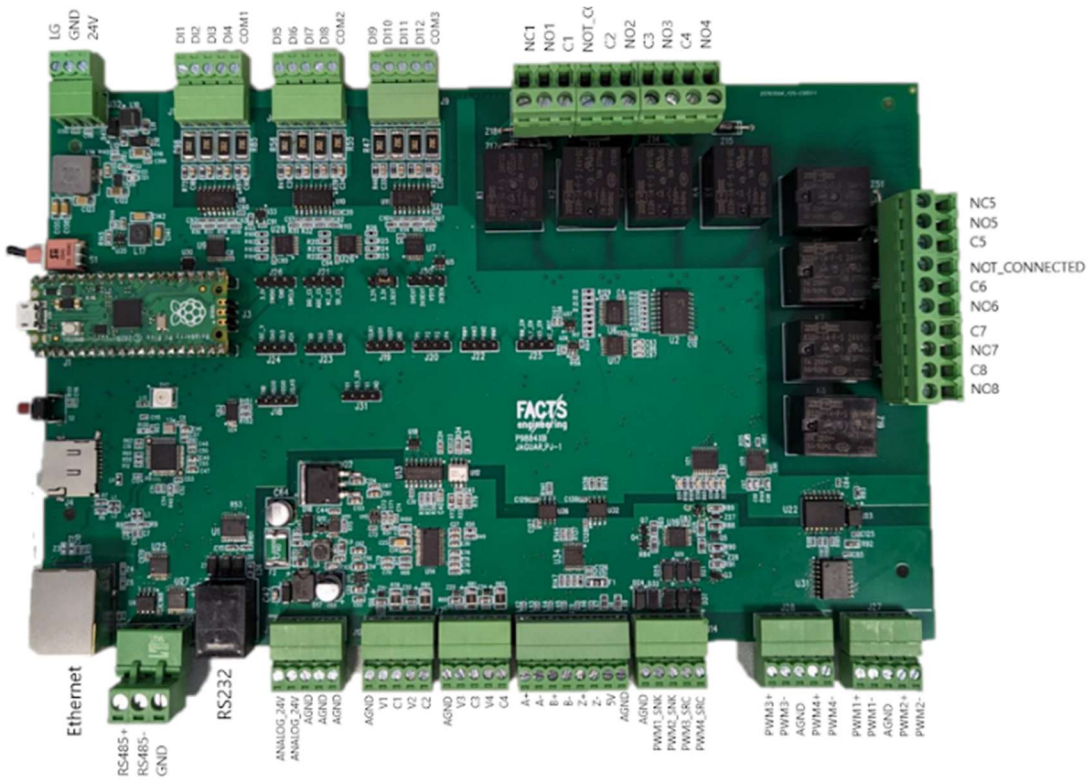
Ao todo, o projeto final contém 13 zonas de cobre, no Layer 1 (vermelho) estão presentes 2 zonas (GND e GNDA), o Layer 2 (verde) apresenta 5 zonas (+24V, +3.3V, +5VA, GNDA, +3.3VA), o Layer 3 (laranja) tem 4 zonas (+24V,GNDA, +3.3V) e por fim o Layer 4 (azul) traz 2 zonas (GND e GNDA).

4.2.3. Prototipagem da Placa

Finalizado o desenvolvimento foi possível fazer a prototipagem e fabricação das primeiras versões para os testes de validação. A prototipagem é uma etapa fundamental no desenvolvimento de produtos e sistemas, pois envolve a criação de versões iniciais ou modelos funcionais para avaliação e teste. Essa prática desempenha um papel crucial em diversas áreas, desde a indústria e engenharia até o design de produtos e desenvolvimento de software.

Inicialmente, 3 placas foram produzidas para os testes que permitiram avaliar o desempenho e a funcionalidade da plataforma em condições práticas. Conforme informado, para os protótipos priorizou-se a facilidade de acesso aos componentes, além da inclusão de pinos de teste para facilitar o processo de validação e busca por possíveis erros. O resultado da prototipagem e fabricação é mostrado na Figura 12.

Figura 12 - Placa Fabricada



Fonte: Elaborado pelo autor.

4.3. Projeto de Software: OpenPLC

No que se refere ao software, selecionou-se o OpenPLC, um projeto open source que permite a implementação das linguagens definidas na norma IEC 61131-3 em microcontroladores, proporcionando conformidade com os padrões tradicionais da indústria e atendendo aos requisitos essenciais de robustez e segurança.

O OpenPLC é um ambiente open source de programação de controladores voltado para o setor industrial e permite a utilização das linguagens previstas na IEC 61131-3 para programar desde controladores baseados em microcontroladores, como por exemplo projetos baseados em Arduino, até projetos embarcados, como por exemplo baseados em Linux. Graças a sua natureza open source o OpenPLC disponibiliza o acesso aos seus arquivos fontes e desta forma facilita a inclusão e desenvolvimento de novas placas, sendo assim, sua escolha facilita a integração da plataforma desenvolvida neste projeto com as normas e padrões adotados pela indústria [Autonomy, 2023].

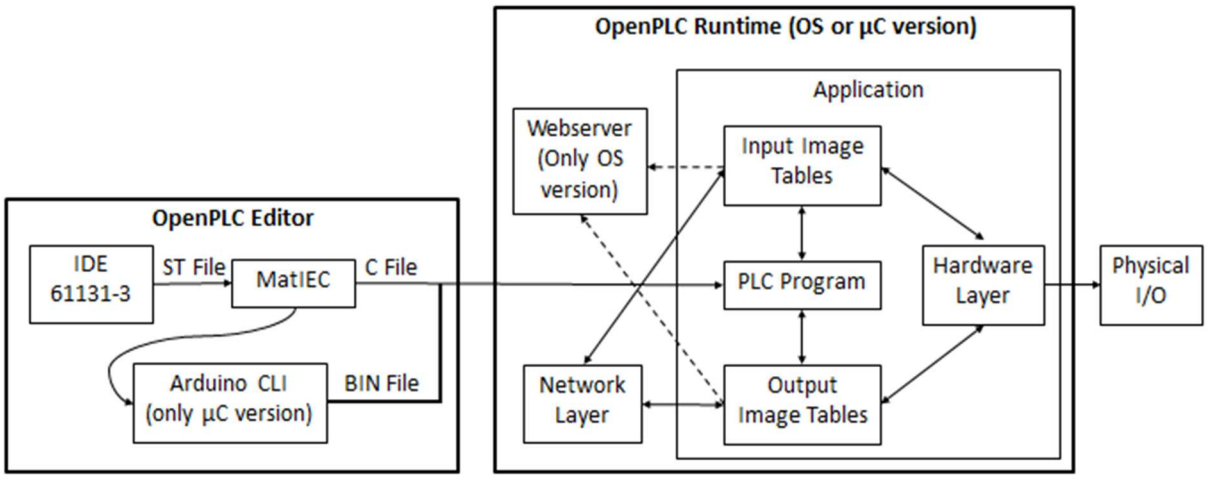
O OpenPLC é dividido em duas partes com funcionalidades bem definidas: o Editor é a interface responsável pelo desenvolvimento dos programas e a interface de maior contato com o usuário, enquanto o Runtime é a parte responsável pela execução dos programas. Ambas as partes funcionam em conjunto e são necessárias para o correto funcionamento do programa.

4.3.1. Arquitetura e Componentes

Conforme comentado anteriormente, o correto funcionamento do OpenPLC depende da integração de 2 partes: o Editor e o Runtime. O Editor é composto por uma IDE, onde independentemente da linguagem em que o programa foi desenvolvido, gera um arquivo em texto estruturado contendo todas as informações do programa e através do auxílio do MatIEC, um projeto criado em 2002 na Universidade do Porto, converte este arquivo em arquivos de linguagem C. Já o Runtime executa a aplicação no hardware. A arquitetura do OpenPLC está exemplificada na Figura 13.

Atualmente, o OpenPLC funciona de duas formas diferentes que dependem da aplicação final. Caso esteja sendo executado em um microcontrolador, o arquivo em C gerado pelo MatIEC é carregado na CLI do Arduino para que o mesmo possa ser compilado e gerar um arquivo binário para o microcontrolador correspondente. Entretanto, em casos em que são utilizados controladores baseados em OS (sistemas operacionais) é possível a utilização de um Webserver e o Runtime interpreta diretamente os arquivos em C.

Figura 13 - Arquitetura do OpenPLC



Fonte: Elaborado pelo autor.

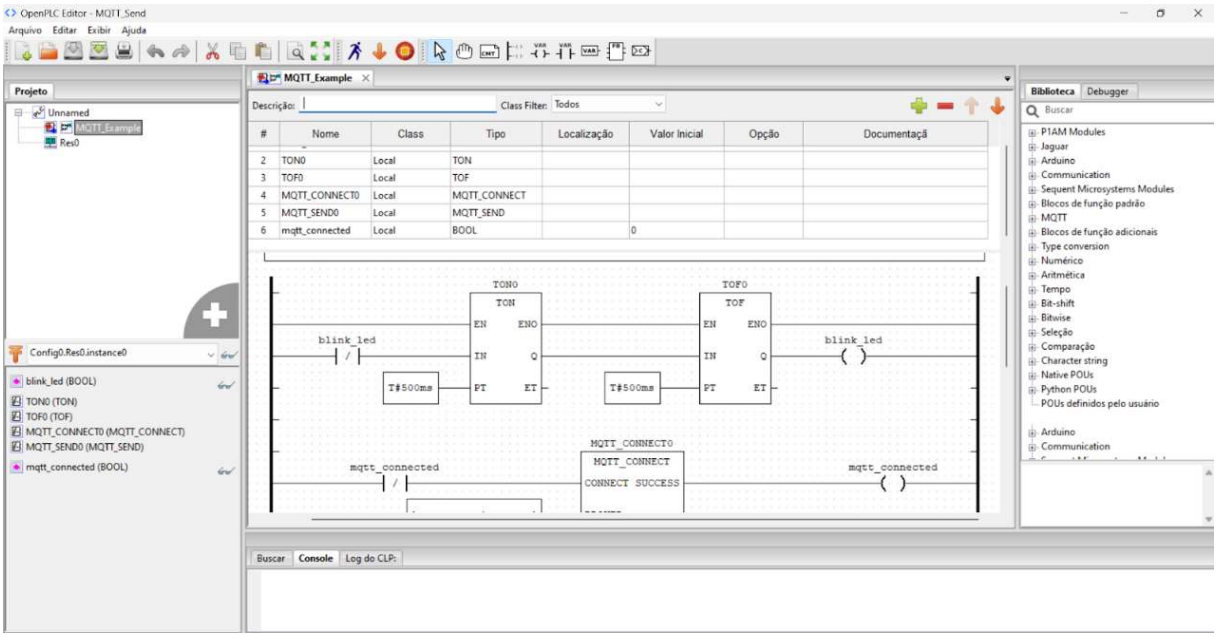
Nota-se que o Runtime dispõe de um bloco de aplicação próprio, isso ocorre devido ao fato de que o runtime do OpenPLC não conversar diretamente com os periféricos físicos do hardware. O Runtime utiliza de '*Image Tables*' para coletar e enviar dados e é necessário um '*Hardware Layer*' que leia as entradas e disponibilize as mesmas na '*Input Image Table*' e leia a '*Output Image Table*' e altere as saídas.

4.3.1.1 OpenPLC Editor

O OpenPLC Editor é um ambiente de desenvolvimento integrado (IDE) que permite criar, compilar e carregar programas baseados na IEC 61131-3 para o OpenPLC Runtime. O Editor é de simples utilização, suporta desenvolvimento em todas as cinco linguagens definidas na norma IEC e segue o padrão definido pela PLCopen TC6-XML.

PLCopen é uma associação mundial independente de fornecedores e produtos que define padrões internacionais para vários tópicos relacionados à programação de controle. O objetivo do sexto comitê técnico do PLCopen (TC6) é definir um formato de arquivo padrão, baseado em XML, para armazenar programas IEC 61131-3. A Figura 14 exemplifica a interface do OpenPLC Editor.

Figura 14 - Interface OpenPLC Editor



Fonte: OpenPLC Editor.

4.3.1.2 OpenPLC Runtime

O OpenPLC Runtime é o que permite a execução dos programas de PLC criados através do OpenPLC Editor. O Runtime foi projetado pensando na sua portabilidade, desta forma, seu núcleo foi escrito em C puro, permitindo que ele fosse portado para diversas plataformas de hardware. Para sistemas em SBC (*Single Board Computer*), que rodam sistemas operacionais embarcados, o Runtime possui um servidor web que permite a configurar o OpenPLC, fazer upload de novos programas para serem executados, além de outras funcionalidades de monitoramento e controle.

Contudo, para sistemas embarcados sem um sistema operacional completo, como ocorre em sistemas microcontrolados, estas configurações devem ser implantadas como parte do próprio programa PLC. Sendo assim, estas configurações e controles são feitas diretamente do OpenPLC Editor que ao fazer o upload do programa no hardware mescla uma versão do runtime OpenPLC especialmente criada para a plataforma de hardware selecionada.

Atualmente, o OpenPLC funciona de duas formas diferentes que dependem da aplicação final. Caso esteja sendo executado em um microcontrolador, o arquivo em C gerado pelo MatIEC é carregado na CLI do Arduino para que o mesmo possa ser compilado e gerar um arquivo binário para o microcontrolador correspondente.

Entretanto, em casos em que são utilizados controladores baseados em OS (sistemas operacionais) é possível a utilização de um Web server e o Runtime interpreta diretamente os arquivos em C.

4.3.2. Desenvolvimento de software

4.3.2.1. Adaptações Necessárias

Embora o OpenPLC já possua suporte ao Raspberry Pi Pico, a utilização de serialização na plataforma desenvolvida neste trabalho exigiu adaptações para garantir a compatibilidade total com os blocos existentes. Como a serialização altera a forma de acesso e endereçamento das entradas e saídas, foi necessário incluir a nova plataforma nos arquivos-fonte do OpenPLC, permitindo que os blocos previamente disponíveis pudessem ser ajustados e utilizados corretamente dentro do novo contexto. Dessa forma, não foi necessária a criação de novos blocos funcionais, mas sim a adaptação da infraestrutura do OpenPLC para integrar a abordagem adotada neste projeto.

A adição de uma nova plataforma ao OpenPLC segue um conjunto de etapas essenciais para a correta implementação. No caso de plataformas baseadas em Arduino, o primeiro passo consiste na coleta de informações detalhadas sobre a placa-alvo, garantindo que seja compatível com a IDE do Arduino e identificando o programador necessário para o upload do firmware. Em seguida, é necessário atualizar o arquivo hals.json, localizado no diretório editor/arduino/examples/Baremetal/. Esse arquivo contém definições das plataformas suportadas pelo OpenPLC, sendo necessário adicionar uma nova entrada correspondente à plataforma em questão. Para isso, pode-se tomar como base um modelo já existente, como o do Arduino Uno, ajustando os parâmetros conforme necessário.

Entre a atualização do hals.json e a modificação do builder.py – script responsável por gerenciar a compilação e o upload do firmware no OpenPLC – há um passo intermediário fundamental: a criação de um arquivo .cpp que define o funcionamento da nova plataforma dentro do sistema. Esse arquivo tem a função de implementar os métodos necessários para que o OpenPLC possa interagir corretamente com o hardware. O processo de criação envolve a identificação da pasta

de plataformas do OpenPLC, que normalmente está localizada no diretório OpenPLC_v3/utils/arduino/hardware/OpenPLC/. Dentro dessa estrutura, existem subpastas correspondentes às placas já suportadas, como uno e mega. Assim, deve-se criar uma nova pasta para a nova plataforma, nomeando-a de forma apropriada (por exemplo, my_custom_board).

Dentro dessa nova pasta, é necessário desenvolver dois arquivos principais: um arquivo de código-fonte .cpp e um cabeçalho .h correspondente. O arquivo .cpp deve conter a implementação das funções de inicialização do hardware, leitura de entradas digitais e analógicas, acionamento de saídas, gerenciamento de temporizações, entre outras funcionalidades essenciais para a integração da plataforma ao OpenPLC. Para isso, recomenda-se utilizar como referência os arquivos de outras placas já suportadas, como o uno.cpp. No arquivo .h, devem ser definidos os protótipos dessas funções, garantindo a modularidade e a correta estruturação do código.

Após a implementação dessas adaptações, os arquivos de configuração do OpenPLC devem ser ajustados para referenciar corretamente a nova plataforma e seus respectivos arquivos-fonte. O hals.json deve ser revisado para incluir a nova entrada, enquanto o builder.py precisará ser modificado para garantir a compilação do novo arquivo .cpp dentro do processo de construção do firmware. Essa etapa é fundamental para assegurar que a nova plataforma seja devidamente reconhecida pelo OpenPLC, permitindo sua integração e operação dentro do sistema de controle.

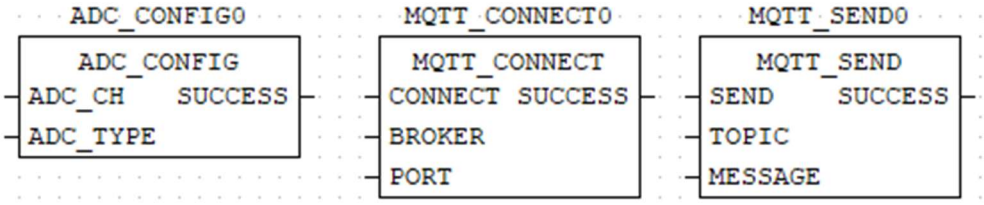
4.3.2.2 Desenvolvimento de Novos Blocos

A linguagem Ladder, atualmente, é a mais utilizada na programação de CLP’s, devido à sua simplicidade de identificação de parâmetros, fácil assimilação e por se basear em esquemas elétricos de contatos. A linguagem Ladder utiliza três elementos básicos para o controle de sistemas: contatos, bobinas e blocos funcionais, este último, permite a utilização de funções avançadas mesmo em uma linguagem simples visto que toda a lógica está oculta no bloco e o programador apenas necessita entender como utilizar o bloco. Desta forma o OpenPLC já possui alguns blocos com funções padrões e mais utilizadas, porém o OpenPLC permite a criação de novos blocos para caso haja necessidade de uma função mais específica ou o bloco necessário não estiver disponível no programa.

Para a plataforma desenvolvida, muitas das funcionalidades foram aproveitadas de outras placas, desta forma foi possível utilizar blocos já existentes no OpenPLC. Contudo, havia a necessidade da criação do bloco para leitura das entradas analógicas e para a comunicação MQTT. A criação de novos blocos no OpenPLC é feita através da modificação do código fonte do Editor e atualmente existem duas maneiras de realizar esta criação: utilizando o *OpenPLC import tool* ou de maneira manual. De modo a aprender melhor sobre o funcionamento do software e juntamente com o auxílio do criador do projeto, optou-se pelo modo manual de criação de blocos.

Primeiramente modificou-se o arquivo *definitions.py*, que contém todos os blocos presentes no Editor, para adicionar na seção "*Function Block Types definitions*" os blocos que seriam criados. Após a inclusão dos blocos, desenvolveu-se um arquivo .xml contendo as informações das variáveis presentes no bloco, juntamente com o seu tipo, para que o MatIEC possa interpretar corretamente o bloco. Conforme explicado anteriormente, o MatIEC compila os arquivos para linguagem C. Desta forma é necessário incluir o arquivo em c responsável pelo funcionamento do bloco nas configurações do MatIEC. Para isso, desenvolveram-se os códigos necessários para que o microcontrolador possa executar de forma correta o funcionamento do código e por fim, obtiveram-se os resultados exemplificados na Figura 15.

Figura 15 - Blocos desenvolvidos



Fonte: OpenPLC Editor.

Através do site do OpenPLC é possível encontrar um passo a passo mais detalhado para a criação com exemplos de como devem ser escritos e modificados os códigos (AUTONOMY, 2023). Vale ressaltar que ao atualizar o OpenPLC Editor, todo o código-fonte na pasta de instalação é substituído por uma cópia mais recente do repositório oficial no GitHub, desta forma os blocos criados a parte serão perdidos, para isso, recomenda-se realizar um backup e seguir novamente os passos para inclusão do bloco após a atualização.

4.4. Projeto de Software: MicroPython

Para ampliar a flexibilidade do projeto, o MicroPython foi definido como uma alternativa de linguagem. Devido à sua simplicidade e flexibilidade, o MicroPython facilita a integração com sistemas da IIoT, tornando o desenvolvimento de soluções de automação mais acessível.

De acordo com Miller et al. (2021), os clientes da indústria tendem a preferir soluções que ofereçam facilidade de uso e uma curva de aprendizado reduzida, especialmente em ambientes de controle e automação onde engenheiros e técnicos operam. O MicroPython, ao apresentar uma sintaxe simples e intuitiva, proporciona uma alternativa mais acessível às linguagens normatizadas pela IEC 61131-3, que, embora reconhecidas pela segurança e confiabilidade, requerem maior especialização e tempo de aprendizado. Estudos como o de Jones e García (2020) demonstram que o uso do MicroPython em sistemas de controle industrial pode reduzir em até 40% o tempo médio de desenvolvimento, devido à sua capacidade de prototipagem rápida e à ampla reutilização de bibliotecas prontas.

O MicroPython tem se destacado como uma solução promissora para a automação industrial, principalmente em aplicações que envolvem a Internet das Coisas (IoT), sistemas embarcados e prototipagem rápida. Sua simplicidade, aliada à flexibilidade e alta capacidade de integração, o coloca como uma opção ideal para indústrias que buscam soluções ágeis, escaláveis e conectadas. O uso de uma linguagem de alto nível, como o MicroPython, permite que engenheiros e desenvolvedores concentrem seus esforços em aspectos estratégicos, ao invés de dedicarem tempo excessivo à codificação complexa, o que acelera o ciclo de desenvolvimento.

Conforme Smith e Jones (2020) apontam, o emprego de linguagens como MicroPython na automação industrial pode reduzir consideravelmente o tempo necessário para o desenvolvimento de soluções, principalmente em fases de prototipagem e integração de novos dispositivos e sensores. Essa vantagem temporal é particularmente relevante em um ambiente industrial em constante evolução, onde a capacidade de resposta rápida a novas demandas é crucial para a competitividade. Gómez et al. (2020) ressaltam ainda que a facilidade de uso do MicroPython minimiza as barreiras associadas à criação de protótipos funcionais, permitindo modificações

em tempo real, sem a complexidade normalmente atribuída a linguagens de programação mais tradicionais, como as definidas pela IEC 61131-3.

4.4.1. Bibliotecas Padrão

O MicroPython inclui uma versão reduzida da biblioteca padrão do Python, especificamente adaptada para operar em dispositivos de baixo consumo de energia e com recursos limitados de memória. Essa adaptação torna o MicroPython ideal para sistemas embarcados e aplicações industriais que exigem eficiência energética, sem comprometer a funcionalidade. No Raspberry Pi Pico, o MicroPython oferece suporte a uma série de módulos otimizados, permitindo uma interface direta com o hardware e o controle de diversos periféricos.

Entre os módulos mais importantes, destaca-se o *machine*, que oferece uma interface para controlar os principais periféricos do microcontrolador RP2040, como os GPIOs (para entradas e saídas digitais), ADCs (conversores analógicos-digitais), PWMs (modulação por largura de pulso), e interfaces de comunicação UART, I2C e SPI. Esses recursos são essenciais para a automação industrial, possibilitando o controle preciso de sensores, atuadores e outros dispositivos conectados ao sistema.

O módulo *uos* fornece uma interface para o gerenciamento de sistemas de arquivos, o que é particularmente útil para o acesso e armazenamento de dados na memória flash do dispositivo. Em aplicações industriais, isso pode ser utilizado para registrar informações críticas sobre o funcionamento de máquinas ou para armazenar temporariamente parâmetros de operação.

Outro módulo crucial é o *utime*, que permite o gerenciamento de temporizações e delays, funcionalidades indispensáveis para o controle de processos em tempo real. Em automação industrial, a precisão no controle de tempo é essencial para garantir a sincronia entre diferentes etapas de produção e o desempenho eficiente de máquinas.

Por fim, o módulo *network* viabiliza a comunicação em rede para dispositivos que dispõem de conectividade Wi-Fi incorporada. Em sistemas industriais modernos, a conectividade é um fator-chave para a implementação de soluções de IoT, permitindo que dados de sensores e máquinas sejam transmitidos em tempo real para sistemas de monitoramento e controle remoto.

Essa combinação de bibliotecas compactas e específicas permite ao MicroPython fornecer uma plataforma poderosa e flexível para o desenvolvimento de soluções industriais, oferecendo controle direto sobre o hardware ao mesmo tempo em que simplifica a programação para engenheiros e desenvolvedores.

4.4.2. IDE

A IDE utilizada neste projeto foi a *Thonny*, uma ferramenta leve e acessível, ideal para o desenvolvimento em MicroPython, especialmente em plataformas como o Raspberry Pi Pico. Inicialmente projetada para ser uma interface simples e intuitiva para programadores Python, a *Thonny* ganhou ampla popularidade devido ao seu suporte nativo ao MicroPython, tornando-se uma ferramenta importante para iniciantes e para a prototipagem de sistemas embarcados.

Um dos principais atrativos da *Thonny* é seu foco na simplicidade, o que facilita o desenvolvimento em MicroPython, especialmente para aqueles que estão dando os primeiros passos na programação embarcada. A interface gráfica amigável separa claramente as janelas de código, saída do interpretador e ferramentas de depuração, permitindo uma interação eficiente e rápida com o microcontrolador. Para projetos industriais ou de IoT que requerem prototipagem rápida, essa interface intuitiva reduz a curva de aprendizado e acelera o desenvolvimento de soluções.

A conexão com a porta serial do Raspberry Pi Pico é automática, com o *Thonny* carregando o REPL (*Read-Evaluate-Print Loop*) diretamente, o que permite testar rapidamente comandos em MicroPython. Esse recurso possibilita a execução de pequenas porções de código em tempo real, uma vantagem significativa para a depuração e análise do comportamento do hardware, especialmente em projetos de controle industrial, onde ajustes em tempo real podem ser cruciais para o sucesso do sistema.

A IDE também oferece facilidade no carregamento e execução de código no microcontrolador, permitindo que scripts sejam gravados diretamente na memória flash do dispositivo. Após o carregamento, o código pode ser executado ou depurado de maneira eficiente, sem a necessidade de reinicializar o hardware manualmente, o que melhora o fluxo de trabalho do desenvolvedor e torna o ciclo de desenvolvimento mais ágil.

Além disso, o *Thonny* permite a instalação de pacotes e bibliotecas externas, facilitando a adição de funcionalidades ao projeto, como o suporte a sensores e atuadores externos, o que é crucial para sistemas de automação que dependem de uma ampla variedade de periféricos.

4.4.3 Desenvolvimento

Assim como ocorre no OpenPLC, a escolha de componentes amplamente utilizados no mercado permitiu o reaproveitamento de diversas bibliotecas de outros projetos, acelerando o desenvolvimento e ampliando as funcionalidades disponíveis na plataforma proposta. Esse fator contribui para a robustez e flexibilidade do sistema, ao reduzir a necessidade de desenvolver soluções do zero e possibilitar a integração com ferramentas já validadas na indústria.

O desenvolvimento no MicroPython apresenta vantagens significativas devido à sua abordagem orientada a objetos e sintaxe simplificada, semelhante à do Python. Essa característica facilita a criação de scripts modulares e intuitivos, eliminando a necessidade de reimplementação de funcionalidades básicas de controle, que seriam essenciais em linguagens normatizadas pela IEC 61131-3. Com o suporte às bibliotecas padrão do MicroPython, é possível implementar diretamente uma ampla gama de funcionalidades, sem a necessidade de customizações adicionais. No entanto, o uso dessa plataforma exige um conhecimento aprofundado sobre o endereçamento das entradas e saídas, a serialização dos sinais e a comunicação entre periféricos.

A complexidade inerente a essas configurações pode representar uma barreira para usuários que buscam integrar rapidamente a plataforma a sistemas industriais. O processo de configuração exige que o usuário manipule diretamente os pinos do microcontrolador, estabeleça a comunicação via SPI para acesso serializado das I/Os e lide com diferentes interfaces de comunicação, como Ethernet, RS-232 e RS-485. Essas etapas tornam o desenvolvimento mais suscetível a erros e aumentam o tempo necessário para implementação. Diante desse desafio, propõe-se o desenvolvimento de uma biblioteca em MicroPython que abstraia essas complexidades e forneça uma interface intuitiva para o usuário final.

A implementação dessa biblioteca permitirá que a seleção de entradas e saídas seja realizada por meio de identificações lógicas, eliminando a necessidade de conhecimento sobre a estrutura interna do hardware. Essa abordagem reduz significativamente o tempo de desenvolvimento e implementação de aplicações industriais, além de aumentar a confiabilidade do sistema, minimizando erros provenientes de configurações manuais.

4.4.3.1 Definição da Biblioteca

O desenvolvimento da biblioteca seguiu uma abordagem estruturada, baseada na análise dos requisitos funcionais da plataforma e na definição de uma interface de programação acessível. O primeiro passo consistiu na identificação das entradas e saídas disponíveis, considerando os diferentes tipos de periféricos suportados: 12 entradas digitais via SPI, 8 saídas digitais via SPI, 4 entradas analógicas via ADC externo, 4 saídas PWM diretas, 1 entrada dedicada para encoder e interfaces de comunicação ethernet, RS-232 e RS-485.

Sem o uso da biblioteca, a operação de cada funcionalidade exigiria a configuração manual de diversos parâmetros. A seguir, está um resumo das etapas necessárias para operar cada periférico diretamente:

- Entradas Digitais: Configuração do barramento SPI, leitura de bits do shift register e conversão para estados lógicos.
- Saídas Digitais: Escrita de estados lógicos no shift register via SPI e acionamento do latch para atualizar as saídas.
- Entradas Analógicas: Configuração do barramento SPI, envio de comandos ao ADC e conversão dos dados lidos.
- Saídas PWM: Configuração da frequência e do duty cycle no RP2040.
- Encoder: Configuração dos GPIOs com interrupção e leitura de pulsos.
- Interfaces de Comunicação: Inicialização dos protocolos e gerenciamento de transmissão e recepção de dados.

Para simplificar o uso da plataforma, a biblioteca foi projetada com uma estrutura modular baseada em classes, encapsulando as operações necessárias para cada tipo de periférico. Cada classe foi projetada para abstrair a configuração interna dos periféricos, permitindo que o usuário final interaja com o hardware sem precisar

manipular diretamente os registradores do microcontrolador ou configurar protocolos de comunicação manualmente.

4.4.3.2 Estrutura da Biblioteca

A biblioteca foi organizada em classes independentes, cada uma responsável por uma funcionalidade específica da plataforma de automação baseada no RP2040 (Raspberry Pi Pico). Essa estrutura modular garante que a biblioteca possa ser expandida conforme necessário, permitindo a inclusão de novas funcionalidades sem comprometer a organização e a usabilidade do código. Além disso, ao utilizar abstrações claras e bem definidas, a biblioteca reduz a complexidade do desenvolvimento, tornando a plataforma mais acessível a desenvolvedores e integradores de sistemas industriais. O código final da biblioteca pode ser encontrado em: https://github.com/gasiepgodoy/Open-Source-PLC.

4.4.3.2.1 Manipulação de Entradas e Saídas Digitais

A manipulação das entradas e saídas digitais foi implementada por meio da criação da classe DigitalIO, que gerencia automaticamente a comunicação via SPI e possibilita ao usuário interagir com os pinos de entrada e saída utilizando operadores de índice. O processo inicia-se com a inicialização da classe, onde são definidos os parâmetros do barramento SPI, a quantidade de entradas e saídas digitais, além da configuração de um registrador interno responsável por armazenar o estado das saídas.

A leitura das entradas digitais ocorre de maneira otimizada, por meio da aquisição simultânea de todos os bits das entradas via SPI. O valor obtido é então convertido para binário e processado de forma a extrair o estado da entrada específica desejada. Dessa forma, o usuário pode acessar diretamente o valor de uma entrada digital com um comando simples, como io[1].

Para a escrita nas saídas digitais, a implementação segue um fluxo estruturado. Primeiramente, o sistema avalia o estado desejado pelo usuário: se o valor definido for 1, a saída correspondente é ativada; caso seja 0, a saída é desativada. Após

essa verificação, o estado atualizado de todas as saídas é automaticamente transmitido via SPI, garantindo a coerência dos sinais enviados ao hardware. Essa abordagem simplifica a interação do usuário, permitindo que uma saída digital seja controlada diretamente por um comando como io[1] = 1. A Figura 16 apresenta a classe desenvolvida e sua estrutura.

Figura 16 - Classe entradas e saídas digitais

```
class DigitalIO:
    """ Classe para manipulação de entradas e saídas digitais via SPI. """
    def __init__(self, spi_id=0, sck=10, mosi=11, miso=8, dio_cs=3, adc_cs=9, num_inputs=12, num_outputs=8):
        self.spi = machine.SPI(spi_id, baudrate=50000, polarity=0, phase=0,
                               sck=machine.Pin(sck), mosi=machine.Pin(mosi), miso=machine.Pin(miso))
        self.dio_cs = machine.Pin(cs, machine.Pin.OUT)
        self.adc_cs = machine.Pin(adc_cs, machine.Pin.OUT)
        self.num_inputs = num_inputs
        self.num_outputs = num_outputs
        self.output_state = 0  # Estado inicial das saídas (todos os bits em 0)
        self.dio_cs.value(1)
        self.adc_cs.value(1)

    def __getitem__(self, pin):
        """ Lê o estado de uma entrada digital. """
        if 1 <= pin <= self.num_inputs:
            self.adc_cs.value(1)  # Garante que o ADC está desativado
            self.dio_cs.value(0)
            raw_data = self.spi.read(2)  # Lê os 12 bits das entradas digitais
            self.dio_cs.value(1)
            value = (raw_data[0] << 8 | raw_data[1]) >> (16 - self.num_inputs)
            return (value >> (self.num_inputs - pin)) & 1
        raise IndexError("Entrada digital fora do alcance")

    def __setitem__(self, pin, state):
        """ Define o estado de uma saída digital. """
        if 1 <= pin <= self.num_outputs:
            if state:
                self.output_state |= (1 << (self.num_outputs - pin))
            else:
                self.output_state &= ~(1 << (self.num_outputs - pin))

            self.adc_cs.value(1)  # Garante que o ADC está desativado
            self.dio_cs.value(0)
            self.spi.write(bytearray([self.output_state]))  # Envia novo estado das saídas
            self.dio_cs.value(1)
        else:
            raise IndexError("Saída digital fora do alcance")
```

Fonte: Thonny IDE.

4.4.3.2.2 Leitura de Entradas Analógicas

De maneira semelhante à abordagem adotada para as entradas digitais, a leitura das entradas analógicas foi implementada por meio da criação da classe AnalogInput. Inicialmente, o barramento SPI é configurado para garantir uma comunicação adequada com o conversor analógico-digital (ADC). Essa etapa é fundamental para assegurar que os dados adquiridos sejam precisos e correspondam corretamente aos valores de tensão analógica aplicados.

Após a configuração inicial, a biblioteca permite a seleção dinâmica do canal de entrada analógica a ser lido. Esse processo é realizado de forma automatizada, garantindo que a leitura do sinal seja feita corretamente via SPI. O valor obtido é então processado e convertido em um número digital correspondente à tensão analógica aplicada ao canal selecionado. Essa estrutura facilita a aquisição de dados, permitindo que o usuário obtenha leituras de forma simplificada e eficiente. A Figura 17 apresenta a classe desenvolvida e sua implementação.

Figura 17 – Classe entradas analógicas

```
class AnalogInput:
    """ Classe para leitura de entradas analógicas via ADS8688IDBTR. """
      def __init__(self, spi_id=0, sck=10, mosi=11, miso=8, dio_cs=3, adc_cs=9, num_inputs=12, num_outputs=8):
        self.spi = machine.SPI(spi_id, baudrate=50000, polarity=0, phase=0,
                               sck=machine.Pin(sck), mosi=machine.Pin(mosi), miso=machine.Pin(miso))
        self.adc_cs = machine.Pin(adc_cs, machine.Pin.OUT)
        self.dio_cs = machine.Pin(dio_cs, machine.Pin.OUT)
        self.dio_cs.value(1)
        self.adc_cs.value(1)

    def __getitem__(self, channel):
        """ Lê o valor do ADC para um canal específico. """
        if 1 <= channel <= 8:
            self.dio_cs.value(1)
            self.adc_cs.value(0)  # Ativa o ADC
            self.spi.write(bytearray([0xC0 | (channel - 1)]))  # Envia comando de leitura
            utime.sleep_us(10)
            value = self.spi.read(2)
            self.adc_cs.value(1)  # Desativa o ADC
            return (value[0] << 8) | value[1]
        raise IndexError("Canal ADC fora do alcance")
```

Fonte: Thonny IDE.

4.4.3.2.3 Geração de Sinais PWM

Ao contrário das classes anteriores, que dependem da comunicação via SPI para a leitura e escrita de sinais, a geração de sinais PWM no RP2040 é realizada diretamente pelo microcontrolador, simplificando significativamente a implementação. Para essa funcionalidade, foi desenvolvida a classe PWMOutput, que permite ao usuário configurar as saídas PWM de maneira intuitiva e eficiente.

A classe foi estruturada para possibilitar a definição do pino de saída PWM, além da configuração da frequência e do duty cycle, parâmetros essenciais para o controle de motores, LEDs e outros dispositivos que dependem de modulação por largura de pulso. Essa abordagem elimina a necessidade de manipulações diretas nos registradores do microcontrolador, proporcionando uma interface mais acessível

e reduzindo a complexidade para o usuário. A Figura 18 ilustra a implementação da classe e sua funcionalidade.

Figura 18 – Classe PWM

```
class PWMOutput:
    """ Classe para controle de saídas PWM. """
    def __init__(self, pin, freq=1000):
        self.pwm = machine.PWM(machine.Pin(pin))
        self.pwm.freq(freq)

    def set_duty_cycle(self, duty):
        """ Define o duty cycle do sinal PWM. """
        self.pwm.duty_u16(int(duty * 65535 / 100))
```

Fonte: Thonny IDE.

4.4.3.2.4 Leitura de Encoder

Semelhante à geração de sinais PWM, a leitura do encoder também é realizada diretamente pelo RP2040, sem a necessidade de comunicação serial. Para essa funcionalidade, foi desenvolvida a classe Encoder, responsável por capturar e processar os pulsos gerados pelo sensor acoplado ao eixo do motor.

A classe foi estruturada para atualizar automaticamente um contador sempre que um pulso for detectado na entrada A do encoder. A direção da rotação é determinada pela relação entre os sinais das entradas A e B, permitindo o incremento ou decremento do contador conforme o sentido de giro. Essa abordagem possibilita a obtenção precisa da posição e velocidade do motor sem necessidade de manipulações manuais nos registradores do microcontrolador. A Figura 19 ilustra a implementação da classe e sua funcionalidade.

Figura 19 – Classe Encoder

```
class Encoder:
    """ Classe para leitura de encoder incremental. """
    def __init__(self, pin_a, pin_b):
        self.pin_a = machine.Pin(pin_a, machine.Pin.IN, machine.Pin.PULL_UP)
        self.pin_b = machine.Pin(pin_b, machine.Pin.IN, machine.Pin.PULL_UP)
        self.count = 0
        self.pin_a.irq(trigger=machine.Pin.IRQ_RISING, handler=self._callback)

    def _callback(self, pin):
        """ Callback para contar pulsos do encoder. """
        if self.pin_b.value():
            self.count += 1
        else:
            self.count -= 1

    def read(self):
        """ Retorna a contagem do encoder. """
        return self.count
```

Fonte: Thonny IDE.

4.4.3.2.5 Comunicação via Ethernet

comunicação Ethernet na plataforma desenvolvida é realizada por meio do chip W5500, que opera utilizando o barramento SPI para a troca de dados. No entanto, diferentemente das classes criadas para o gerenciamento de entradas e saídas digitais e analógicas, a classe Ethernet não realiza a inicialização direta do SPI em sua implementação.

Essa decisão foi tomada visando proporcionar maior flexibilidade na adaptação do hardware. Enquanto as entradas e saídas digitais possuem uma serialização fixa, garantindo que a estrutura do SPI permaneça inalterada, a comunicação Ethernet pode variar conforme a necessidade do projeto, podendo envolver a troca do chip controlador ou a redefinição dos pinos utilizados.

Dessa forma, a classe Ethernet foi projetada para atuar exclusivamente na configuração da interface de rede, permitindo ao usuário definir o endereço IP, estabelecer a comunicação e realizar o envio e recebimento de dados de maneira eficiente. Essa abordagem modular possibilita a adaptação do código a diferentes configurações de hardware sem a necessidade de modificar a estrutura central da classe. A Figura 20 apresenta a classe desenovlida.

Figura 20 – Comunicação Ethernet

```
class Ethernet:
    """ Classe para comunicação via Ethernet utilizando W5500. """
    def __init__(self, spi, cs_pin):
        self.nic = network.WIZNET5K(spi, cs=machine.Pin(cs_pin))

    def connect(self, timeout=10):
        """ Conecta a rede via DHCP e retorna o IP obtido. """
        self.nic.active(True)  # Ativa a interface Ethernet
        self.nic.ifconfig('dhcp')  # Solicita IP via DHCP
         # Aguarda até obter um IP válido
        for _ in range(timeout):
            ip = self.nic.ifconfig()[0]
            if ip and ip != '0.0.0.0':
                return ip
            time.sleep(1)

    def send(self, data):
        """ Envia dados pela rede Ethernet. """
        self.nic.send(data)

    def receive(self):
        """ Recebe dados da rede Ethernet. """
        return self.nic.recv(1024)
```

Fonte: Thonny IDE.

4.4.3.2.6 Comunicação Serial

A plataforma desenvolvida oferece suporte às comunicações RS-232 e RS-485, ambas amplamente utilizadas na integração com CLPs, sensores e dispositivos industriais. Essas interfaces seriais possibilitam a troca de dados de forma robusta e confiável em ambientes industriais, garantindo a interoperabilidade com diferentes equipamentos.

A implementação dessas comunicações foi realizada de maneira semelhante, utilizando a configuração da porta UART do RP2040. Através da biblioteca desenvolvida, a inicialização da interface serial ocorre de forma parametrizável, permitindo que o usuário defina os pinos utilizados, a taxa de transmissão (baud rate) e outras configurações essenciais para o correto funcionamento da comunicação.

Além disso, foram implementadas funções dedicadas para o envio e recebimento de dados, garantindo que a transmissão ocorra de maneira eficiente e segura. A Figura 21 ilustra a estrutura da classe desenvolvida para a comunicação serial, destacando sua modularidade e facilidade de uso na integração com dispositivos industriais.

Figura 21 – Comunicação Serial

```
class RS232:
    """ Classe para comunicação via RS-232. """
    def __init__(self, uart_num=1, baudrate=9600, tx=4, rx=5):
        self.uart = machine.UART(uart_num, baudrate=baudrate, tx=machine.Pin(tx), rx=machine.Pin(rx))

    def send(self, data):
        """ Envia dados via RS-232. """
        self.uart.write(data)

    def receive(self):
        """ Recebe dados via RS-232. """
        return self.uart.read()

class RS485(RS232):
    """ Classe para comunicação RS-485, herdando de RS-232 e adicionando controle de transmissão. """
    def __init__(self, uart_num=1, baudrate=9600, tx=4, rx=5, de_re=6):
        super().__init__(uart_num, baudrate, tx, rx)
        self.de_re = machine.Pin(de_re, machine.Pin.OUT)

    def send(self, data):
        """ Envia dados via RS-485, ativando o transmissor. """
        self.de_re.value(1)
        utime.sleep_ms(1)
        self.uart.write(data)
        utime.sleep_ms(1)
        self.de_re.value(0)

    def receive(self):
        """ Recebe dados via RS-485. """
        return self.uart.read()
```

Fonte: Thonny IDE.

5. RESULTADOS

Nesta seção a plataforma desenvolvida foi testada e validada em dois cenários de aplicações utilizando tanto o OpenPLC quanto o MicroPython. Os testes operacionais e a validação são fases essenciais no ciclo de desenvolvimento de um produto ou sistema. Durante os testes operacionais, as funcionalidades foram verificadas em condições normais de operação para assegurar a presença e o correto funcionamento das características planejadas, além de confirmar se o produto atende aos requisitos estabelecidos no início do projeto, incluindo critérios funcionais, de desempenho.

5.1. Teste Operacional e Validação: OpenPLC

Para os primeiros testes operacionais da plataforma e a validação do seu funcionamento com o OpenPLC, utilizou-se como cenário uma estação Festo que simula um processo industrial localizada no laboratório da Unesp Sorocaba. A estação simula um processo de separação de peças segundo a respectiva cor e propõe-se a utilização de comunicação via MQTT para estabelecer um dashboard de visualização online, enquadrando-se assim em um projeto de IIoT.

Ao todo a estação conta com 4 sensores, 2 atuadores pneumáticos e 1 motor de corrente contínua e através da utilização de um sensor fotoelétrico e um sensor indutivo a estação separa as peças por meio da combinação desses sensores como mostra a Tabela 2.

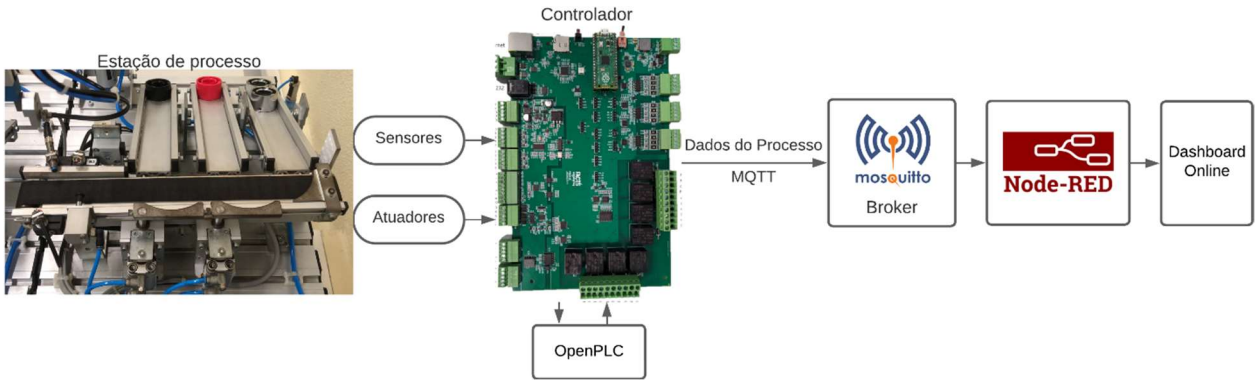
Tabela 2 - Identificação da cor da peça

Fotoelétrico	Indutivo	Cor
0	0	Preta
1	0	Vermelha
1	1	Prata

Fonte: Elaborado pelo autor

Desta forma visa conseguir controlar o processo de separação de forma correta e juntamente realizar e comunicar em tempo real a contagem de peças de cada cor para um dashboard remoto como exemplifica a Figura 16.

Figura 22 - Esquemático da Aplicação



Fonte: Elaborado pelo autor

Delimitando o cenário proposto, os testes foram divididos em três partes distintas. Inicialmente, foi realizada a validação da comunicação com o OpenPLC e a verificação do funcionamento das entradas e saídas digitais presentes no processo. Em seguida, desenvolveu-se a comunicação e disponibilização dos dados online. Por fim, validou-se o cenário como um todo, simulando um processo industrial integrado.

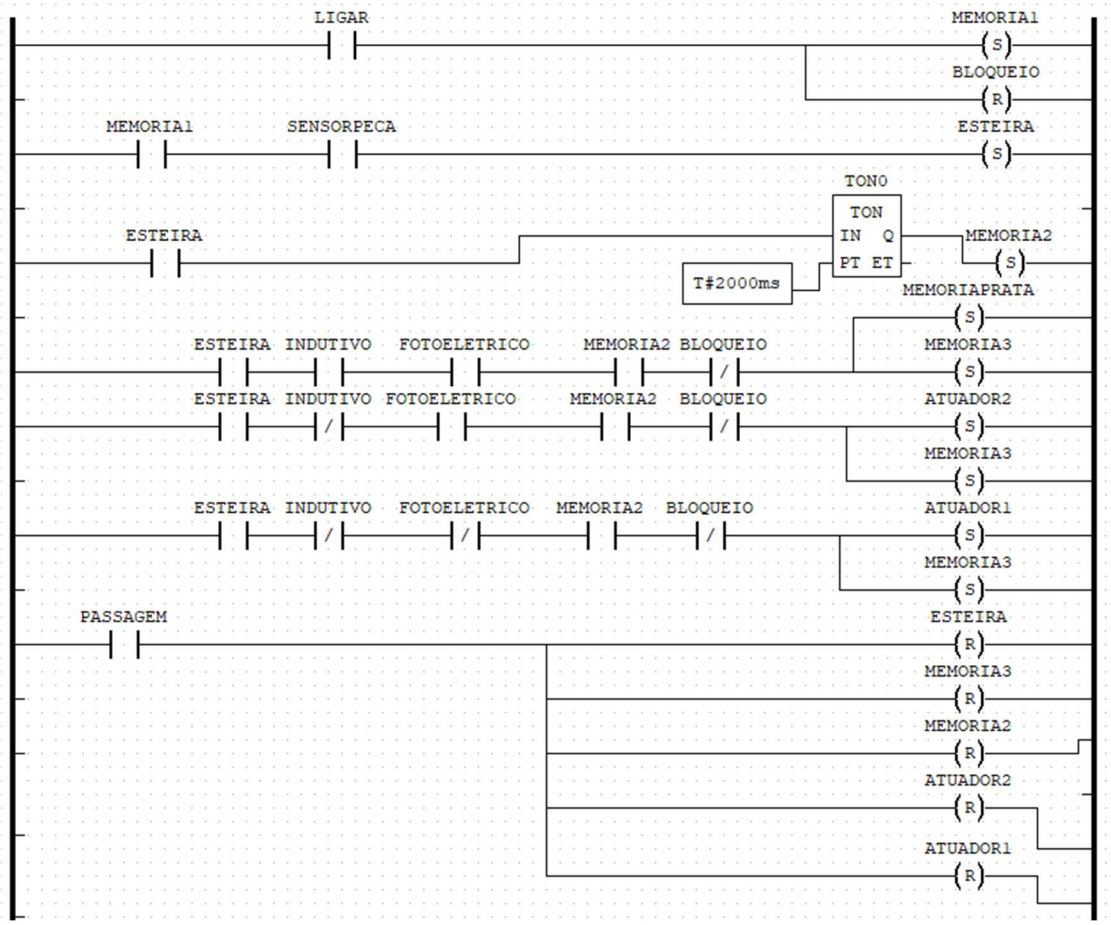
Para a primeira parte dos testes, a opção foi pelo desenvolvimento do código de controle do processo por meio da linguagem Ladder disponibilizada pelo OpenPLC. Essa escolha foi motivada pela ampla utilização do Ladder em plantas industriais e, além disso, a verificação direta do funcionamento das entradas e saídas no processo contribui para a redução do tempo demandado nesta fase.

Juntamente com o desenvolvimento do código definiu-se quais as entradas e saídas da placa seriam utilizadas e através do OpenPLC realizou-se o endereçamento das variáveis. Desta forma a Figura 17 demonstra o código de controle desenvolvido e a Figura 18 traz o endereçamento das entradas e saídas utilizadas.

Com o código desenvolvido iniciou-se a validação da aplicação juntamente com a estação de processos. Primeiramente validou-se a compilação e carregamento do código na plataforma através do OpenPLC onde o mesmo ocorreu sem falhas ou complicações e em seguida validou-se o correto funcionamento das entradas e saídas digitais presentes na plataforma. Nota-se que o processo utiliza apenas 7 entradas e 4 saídas, sendo assim, visando uma validação de todas as 12 entradas e 8 saídas simulou-se o processo por duas vezes alterando o endereçamento. Ao fim dos testes foi possível afirmar que todas as entradas e saídas da plataforma estavam

funcionando de maneira correta e a interpretação do código desenvolvido através do OpenPLC também ocorreu corretamente, validando então a primeira etapa dos testes.

Figura 23 - Programa de Automação Discreta



Fonte: Elaborado pelo autor

Figura 24 - Endereçamento das Entradas e Saídas no OpenPLC Editor

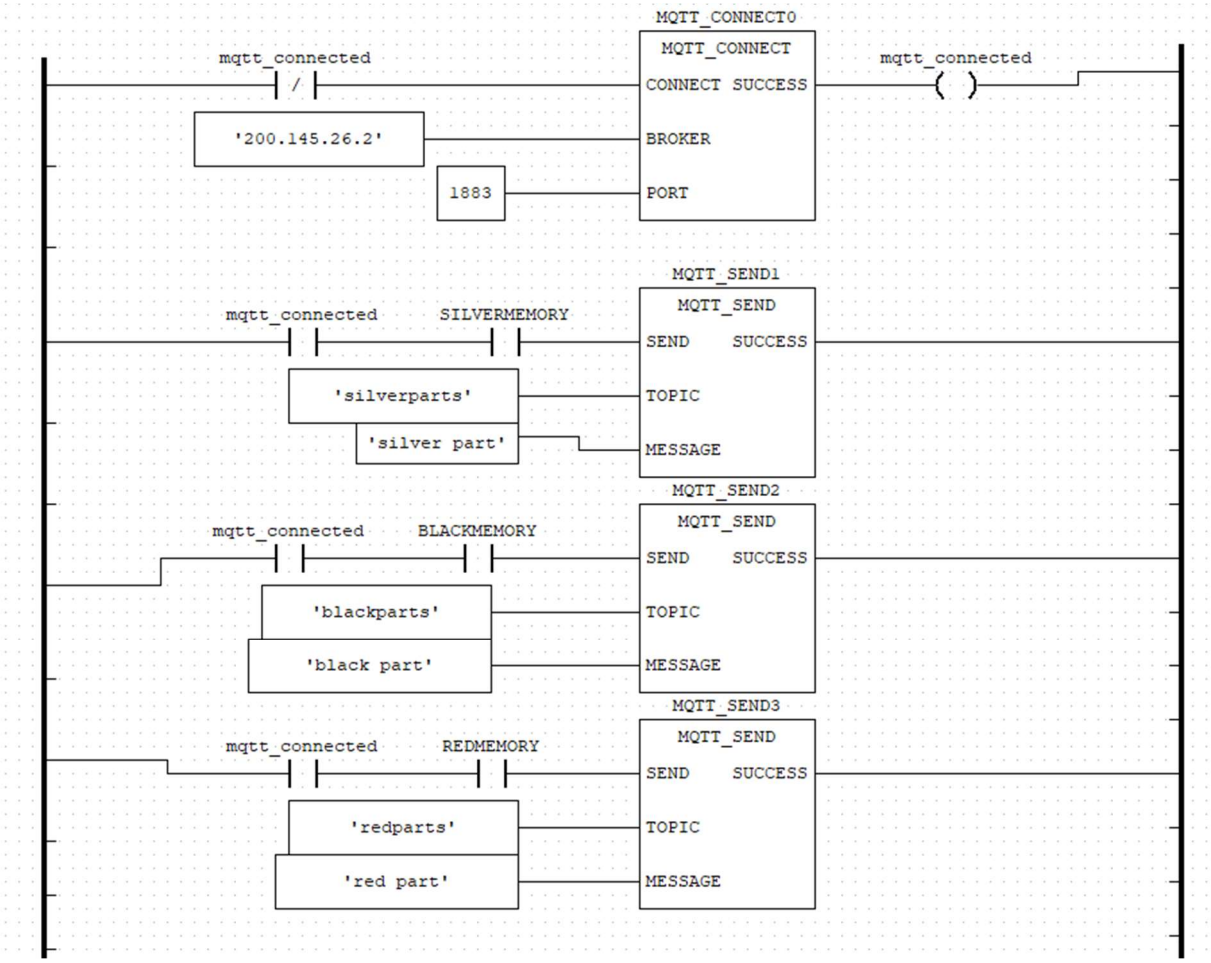
TON0	Local	TON	
LIGAR	Local	BOOL	%IX1.0
DESLIGAR	Local	BOOL	%IX1.1
RESET	Local	BOOL	%IX1.2
SENSORPECA	Local	BOOL	%IX0.0
INDUTIVO	Local	BOOL	%IX0.1
FOTOELETRIC(	Local	BOOL	%IX0.4
PASSAGEM	Local	BOOL	%IX0.5
ESTEIRA	Local	BOOL	%QX0.0
ATUADOR1	Local	BOOL	%QX0.1
ATUADOR2	Local	BOOL	%QX0.2
BLOQUEIO	Local	BOOL	%QX0.3
MEMORIA	Local	BOOL	
MEMORIA1	Local	BOOL	
MEMORIA2	Local	BOOL	
MEMORIA3	Local	BOOL	
MEMORIAPRA	Local	BOOL	
MEMORIAPRE	Local	BOOL	
MEMORIAVER	Local	BOOL	

Fonte: Elaborado pelo autor

Finalizando a primeira etapa iniciou-se o desenvolvimento da comunicação. Devido à escolha da utilização do protocolo MQTT para comunicação, definiu-se a utilização do Mosquitto como broker e do Node-RED como interface para visualização dos dados, optou-se pela utilização destas plataformas por ambas serem open source e já serem validadas em ambientes industriais.

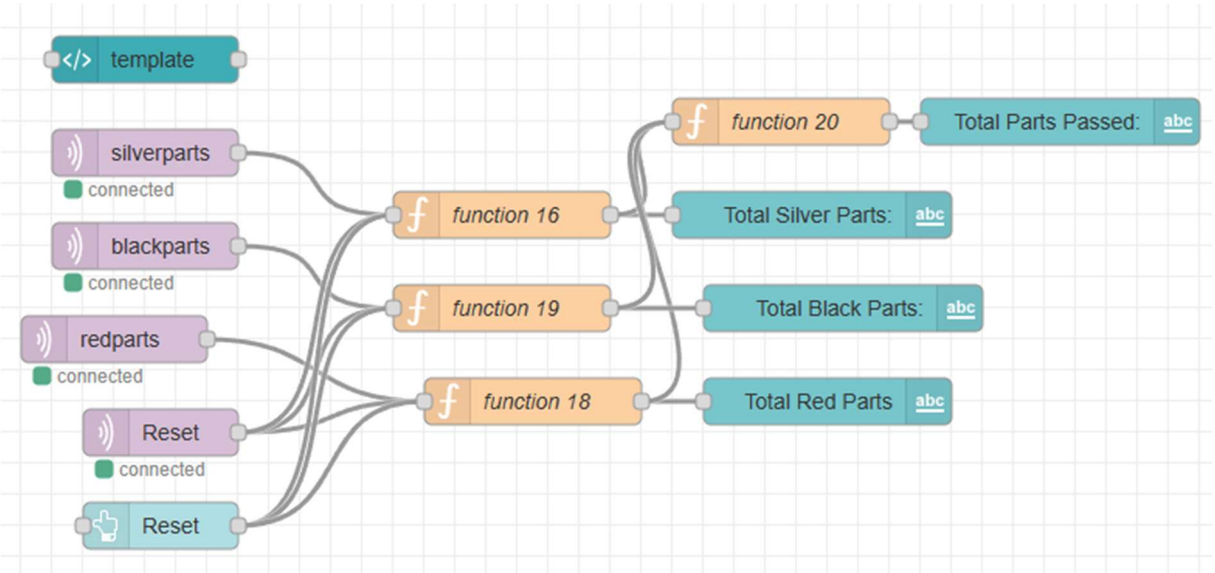
A Figura 19 traz a programação desenvolvida em Ladder através do OpenPLC para a comunicação. Além do programa presente no controlador, também é necessário o desenvolvimento do programa do Node-RED para a leitura e disponibilização dos dados mediante um dashboard online, sendo assim a Figura 20 demonstra o desenvolvimento realizado.

Figura 25 - Programa da Comunicação



Fonte: Elaborado pelo autor

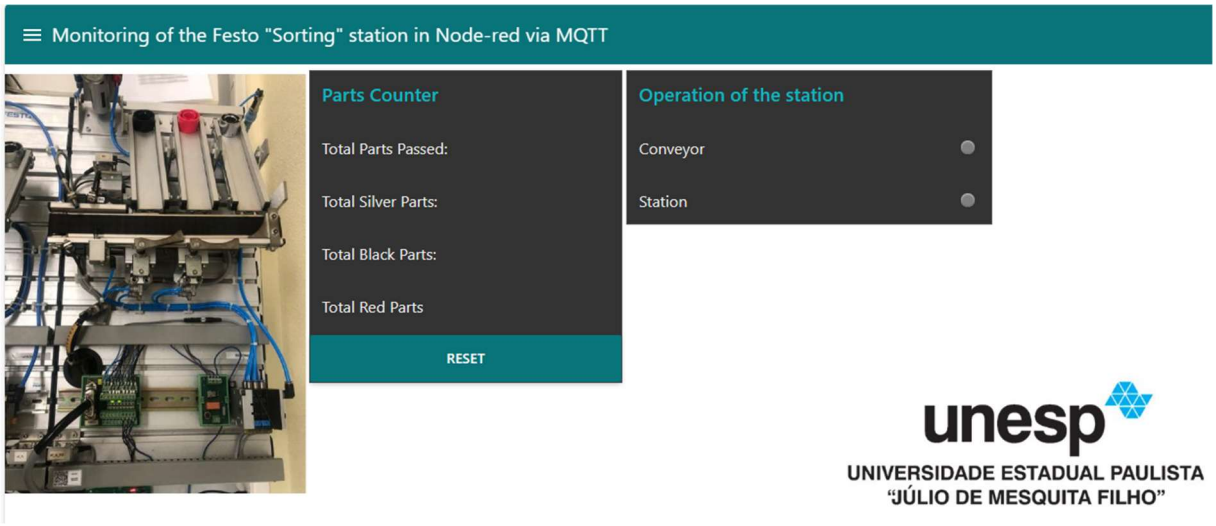
Figura 26 - Desenvolvimento Node-RED



Fonte: Elaborado pelo autor

Finalizado o desenvolvimento iniciou-se os testes da comunicação e validação da plataforma. Para isso, com o processo em funcionamento, analisou-se o envio das mensagens e a contagem e disponibilização dos dados no dashboard como mostra a Figura 21. Ao fim dos testes concluiu-se que a comunicação ocorreu de forma satisfatória, validando assim a mesma e garantindo a possibilidade da utilização da plataforma em cenários de IIoT.

Figura 27 - Dashboard de dados



Fonte: Elaborado pelo autor

Por fim, validou-se o conjunto das funcionalidades em um processo industrial através da utilização da estação de processos presente no laboratório. O vídeo está disponível em https://youtu.be/A4_A6H7zyto e apresenta os resultados e valida a utilização da plataforma no processo industrial.

Ao fim da validação é possível afirmar que a plataforma desenvolvida atende aos requisitos estabelecidos e se mostrou uma alternativa confiável e robusta para controle de processos em aplicações de IIoT. A utilização do OpenPLC se mostra uma excelente ferramenta para programação através das linguagens da IEC 61131-3, o que facilita na implementação em plantas industriais, além de permitir a implementação da comunicação de uma maneira simples e eficaz. A comunicação através do protocolo MQTT, juntamente com as outras opções de conectividade presentes na plataforma, mostra-se uma vantagem em relação a controladores

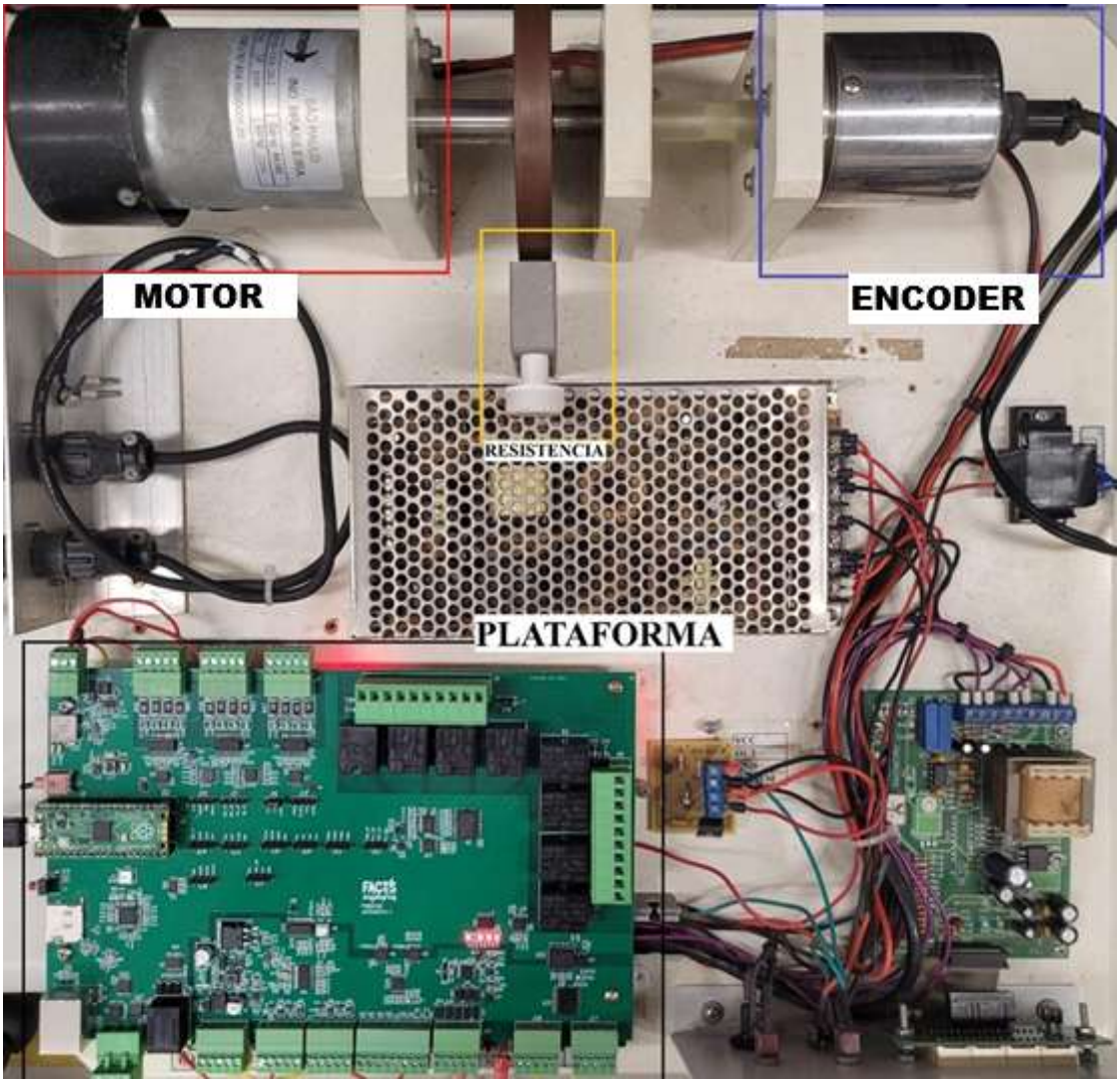
padrões que normalmente não trazem estas funcionalidades em suas versões de entrada.

5.2. Teste Operacional e Validação: MicroPython

Para validar a aplicabilidade do MicroPython na plataforma desenvolvida, foi estruturado um cenário de teste detalhado, projetado para avaliar as capacidades do sistema no controle de processos industriais. O experimento envolve a implementação de um controle PID de velocidade, demonstrando a capacidade da plataforma em aplicações que exigem precisão e resposta dinâmica.

A bancada experimental foi montada no laboratório e consiste em um motor de corrente contínua modelo M910-VER-K2 da Motron, este motor consiste em um motor de 24v com 43w de potência e 2200 RPM, um encoder responsável pela medição da velocidade e posição do motor modelo 5810-0351-0600 da hohner e uma resistência de carga variável, acoplada ao eixo do motor, que simula variações de carga. A Figura 28 ilustra a configuração física do experimento, destacando as conexões entre os componentes e a interface com a plataforma.

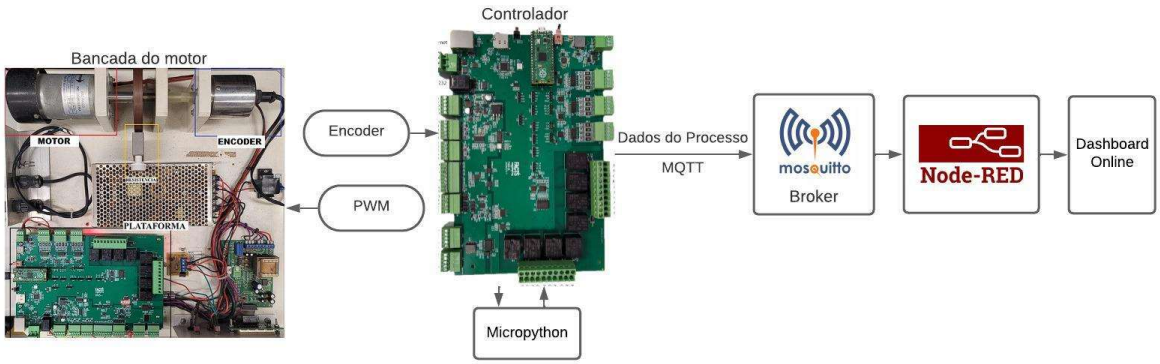
Figura 28 – Bancada de testes de controle de motor cc



Fonte: Elaborado pelo autor

O experimento tem como principal objetivo avaliar a eficiência da plataforma desenvolvida na compensação de variações de carga no motor, garantindo que a velocidade seja mantida dentro de uma faixa predefinida, mesmo diante de perturbações externas. Para alcançar esse objetivo, implementou-se um controle PID (Proporcional-Integral-Derivativo), que ajusta dinamicamente o sinal PWM enviado ao motor, garantindo a estabilização da velocidade desejada. A Figura 29 exemplifica a aplicação desenvolvida e o funcionamento esperado do teste.

Figura 29 – Esquemático da Aplicação



Fonte: Elaborado pelo autor

O código de controle foi desenvolvido em MicroPython, utilizando a biblioteca Jaguar, criada neste trabalho para abstração das entradas e saídas da plataforma. A estrutura do código contempla a leitura do encoder, para obtenção em tempo real da velocidade do motor, e a implementação do algoritmo PID, com ajuste dos coeficientes Kp, Ki e Kd. Com esses valores ajustados, o sistema realiza a modulação PWM para controlar a potência enviada ao motor, garantindo a resposta adequada a variações de carga.

Além do controle PID, foi implementada uma interface MQTT, que possibilita o envio de dados coletados para um sistema de monitoramento, permitindo o acompanhamento detalhado do comportamento do sistema e possibilitando ajustes conforme necessário. Essa funcionalidade possibilita a integração com o Node-RED, facilitando a visualização e análise das variáveis do experimento em tempo real.

O desenvolvimento do código inicia-se com a importação das bibliotecas essenciais para a execução do controle, sendo elas: utime (para manipulação de contagem de tempo e delays), machine (para configurações gerais de hardware), mqtt (para comunicação com o Node-RED via protocolo MQTT), Jaguar (biblioteca desenvolvida para a comunicação com a plataforma). Essas bibliotecas foram escolhidas por fornecerem uma base sólida para um controle preciso do motor, garantindo a captura confiável das leituras do encoder e a comunicação eficiente com o sistema de monitoramento.

Com as bibliotecas devidamente importadas, o próximo passo foi a configuração dos componentes do sistema. Para isso, definiu-se o pino responsável pelo PWM, realizou-se a configuração dos parâmetros Ethernet, e estipularam-se os termos proporcional, derivativo e integrativo do PID, além do setpoint de velocidade do motor. Também foi realizada a configuração do encoder, determinando o pino utilizado para a leitura dos pulsos gerados. A Figura 30 ilustra o código desenvolvido até essa etapa, apresentando a estrutura da configuração inicial do experimento e destacando os principais parâmetros envolvidos no controle.

Figura 30 – Inicio do código

```
import utime
from machine import SPI, Pin
from umqtt.simple import MQTTClient
from jaguar import DigitalIO, PWMOutput, Encoder, Ethernet

# Configuração do hardware
pwm_motor = PWMOutput(pin=6)  # Controle de velocidade do motor

# Configuração da comunicação Ethernet
spi = SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
eth = Ethernet(spi, cs_pin=17)
eth.connect()

# Configuração do PID
class PIDController:
    def __init__(self, setpoint, kp, ki, kd, integral_max):
        self.setpoint = setpoint
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.integral_max = integral_max
        self.prev_error = 0
        self.integral = 0

    def compute(self, feedback_value):
        error = self.setpoint - feedback_value
        self.integral += error
        # Limita a parte integral para evitar integral windup
        self.integral = min(max(self.integral, -self.integral_max), self.integral_max)
        derivative = error - self.prev_error
        output = self.kp * error + self.ki * self.integral + self.kd * derivative
        self.prev_error = error
        return output

RPMdesejado = 750
KP = 0.2
KI = 0.01
KD = 0.001
INTEGRAL_MAX = 10
duty_cycle = 25

# Criação do objeto PIDController
pid = PIDController(RPMdesejado, KP, KI, KD, INTEGRAL_MAX)

# Configuração do Encoder
tachometerPin = Pin(28, Pin.IN, Pin.PULL_DOWN)
counter = 0
```

Fonte: Elaborado pelo autor

Após a realização das configurações iniciais, foram desenvolvidas duas funções principais para garantir o funcionamento adequado do sistema de controle. A primeira função, denominada tachometer, foi criada para a contagem de pulsos do encoder, permitindo a determinação precisa da velocidade do motor. Essa função é essencial para o feedback do sistema de controle PID, pois fornece os dados necessários para que os ajustes no sinal PWM sejam realizados de maneira eficiente.

A segunda função desenvolvida foi uma função de callback, responsável pelo recebimento de mensagens MQTT. Essa função permite a atualização remota dos parâmetros do controle, possibilitando ajustes dinâmicos no sistema sem a necessidade de reprogramação manual. Com essa abordagem, o usuário pode modificar valores como setpoint de velocidade e coeficientes do PID (Kp, Ki e Kd) diretamente por meio da interface MQTT, facilitando a calibração e otimização do experimento. A Figura 31 apresenta a implementação dessa função juntamente com o envio dos parâmetros iniciais.

Figura 31 – Funções desenvolvidas

```
# Função de interrupção do Encoder
def tachometer(pin):
    global counter
    counter += 1

tachometerPin.irq(trigger = Pin.IRQ_RISING,
                  handler = tachometer)

# Função de callback para receber mensagens MQTT
def on_message(topic, msg):
    global RPMdesejado, KP, KD, KI, pid
    if topic == b"motor/setpoint":
        print("setpoint atualizado", int(msg))
        RPMdesejado = int(msg)
        pid = PIDController(RPMdesejado, KP, KI, KD, INTEGRAL_MAX)
    elif topic == b"motor/kp":
        KP = float(msg)
        pid = PIDController(RPMdesejado, KP, KI, KD, INTEGRAL_MAX)
    elif topic == b"motor/ki":
        KI = float(msg)
        pid = PIDController(RPMdesejado, KP, KI, KD, INTEGRAL_MAX)
    elif topic == b"motor/kd":
        KD = float(msg)
        pid = PIDController(RPMdesejado, KP, KI, KD, INTEGRAL_MAX)
```

Fonte: Elaborado pelo auto

Com as funções essenciais implementadas, o próximo passo foi o desenvolvimento da comunicação MQTT, permitindo a troca de dados em tempo real entre a plataforma e a interface de monitoramento. Para isso, inicialmente definiu-se o broker MQTT responsável por gerenciar as mensagens publicadas e recebidas. Em seguida, a função de callback foi associada ao cliente MQTT, garantindo que as mensagens recebidas nos tópicos assinados fossem corretamente processadas. Essa estrutura possibilitou a atualização remota de parâmetros, como o setpoint da velocidade e os coeficientes do PID, além do envio contínuo dos dados coletados pelo sistema para análise externa.

Após essas configurações, os tópicos MQTT necessários foram assinados (*subscribe*), estabelecendo a comunicação com o broker. Por fim, a conexão MQTT

foi iniciada, garantindo que o sistema pudesse interagir de forma dinâmica com o ambiente externo. A Figura 32 exemplifica a implementação dessa comunicação.

Figura 32 - Código do MQTT

```
# ===== COMUNICAÇÃO MQTT =====
BROKER = '192.168.0.22'

def conectar_mqtt():
    cliente = MQTTClient('jaguar', BROKER, port=1883, keepalive=60)
    cliente.set_callback(on_message)
    res = cliente.connect()
    print("Resultado da conexão:", res)  # geralmente 0 = sucesso
    utime.sleep(0.5)
    return cliente

mqtt_client = conectar_mqtt()
utime.sleep(1)
mqtt_client.subscribe(b'motor/setpoint')
mqtt_client.publish("motor/setpoint", str(RPMdesejado))
mqtt_client.subscribe(b'motor/kp')
mqtt_client.publish("motor/kp", str(KP))
mqtt_client.subscribe(b'motor/kd')
mqtt_client.publish("motor/kd", str(KD))
mqtt_client.subscribe(b'motor/ki')
mqtt_client.publish("motor/ki", str(KI))
```

Fonte: Elaborado pelo autor

Com todas as configurações e funções preparadas, desenvolveu-se o loop principal do experimento, responsável pelo controle contínuo do sistema. O primeiro passo dentro do loop foi a checagem de mensagens MQTT, permitindo que qualquer atualização remota nos parâmetros fosse imediatamente aplicada ao sistema. Isso garantiu flexibilidade no ajuste do setpoint de velocidade e dos coeficientes do controlador PID, sem a necessidade de interrupção da execução.

Em seguida, realizou-se a leitura do encoder para determinar a velocidade do motor em RPM (rotações por minuto). Para minimizar erros de medição, foi adotado um método de média móvel, onde três leituras consecutivas foram armazenadas, convertendo inicialmente a contagem de pulsos para rotações por segundo (RPS) e posteriormente para RPM. A média desses valores foi utilizada para suavizar possíveis flutuações nos dados e melhorar a precisão da medição.

Após obter a velocidade real do motor, o próximo passo foi a execução do cálculo do PID. O controlador recebeu o valor medido e o comparou com o setpoint definido, ajustando dinamicamente o duty cycle do sinal PWM para corrigir qualquer desvio detectado. Por fim, os parâmetros atuais do sistema foram enviados via MQTT, permitindo o monitoramento remoto da velocidade do motor e das ações de controle aplicadas. A Figura 33 ilustra a implementação desse loop principal garantindo

visibilidade em tempo real sobre o comportamento do experimento, facilitando ajustes e análises.

Figura 33 - Código do controlador

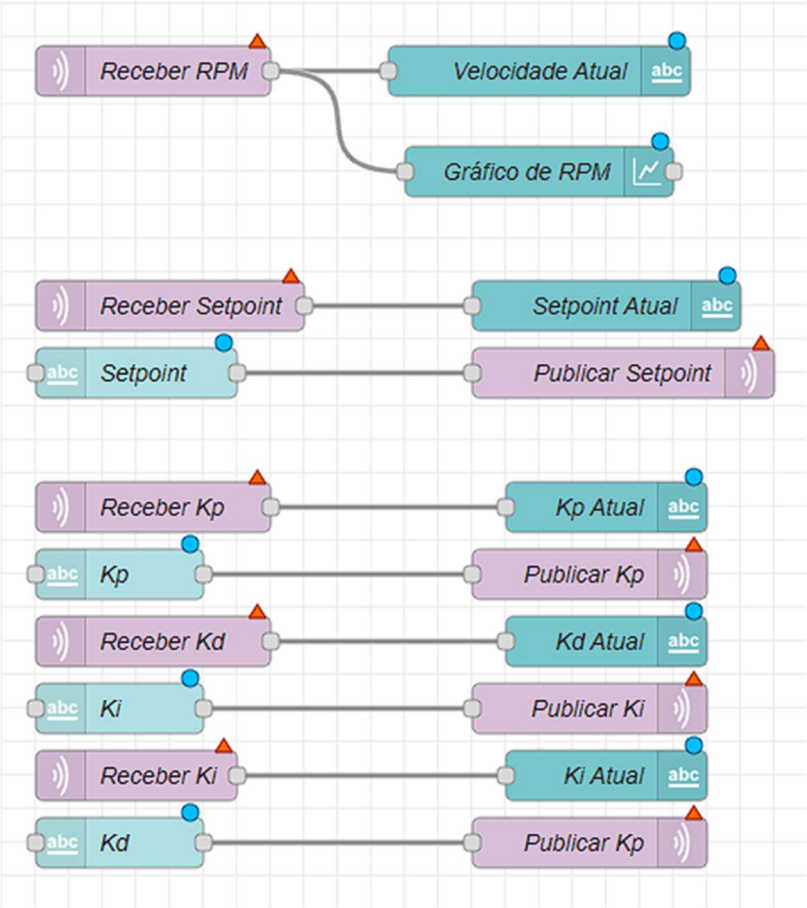
```
# Loop principal
while True:

    if utime.ticks_diff(utime.ticks_ms(), last_check) > 200:
        try:
            mqtt_client.check_msg()
        except Exception as e:
            print("MQTT erro:", e)
        last_check = utime.ticks_ms()

    # Leitura do encoder para cálculo de RPM
    SAMPLING_TIME = 1 # in seconds
    rpm = 0
    i = 0
    while i < 3:
        utime.sleep(SAMPLING_TIME)
        revolutions_per_sampling_time = counter/600
        revolutions_per_sec = revolutions_per_sampling_time/SAMPLING_TIME
        revolutions_per_minute = round(revolutions_per_sec * 60, 0)
        rpm = rpm + revolutions_per_minute
        # reset the counter to zero
        counter = 0
        i = i + 1

    RPM = rpm/3
    print("RPM: ", int(RPM))

# Cálculo do sinal de controle pelo PID
    pid_output = pid.compute(RPM)

# Constante para limitar o aumento gradual
    MAX_INCREMENT = 0.5
    MAX_DECREMENT = 0.5

# Cálculo da Nova Compensação
    newduty_cycle = pid_output + duty_cycle

# Limitação do Ciclo de Trabalho para evitar aumento repentino
    if newduty_cycle - duty_cycle > MAX_INCREMENT:
        newduty_cycle = min(duty_cycle + MAX_INCREMENT, 100)
    elif newduty_cycle - duty_cycle < -MAX_DECREMENT:
        newduty_cycle = max(duty_cycle - MAX_DECREMENT, 0)

# Aplicação do Ciclo de Trabalho ao Motor
    pwm_motor.set_duty_cycle(int(newduty_cycle))

# Atualização do ciclo de trabalho atual
    duty_cycle = newduty_cycle

# Publicação dos dados via MQTT
    mqtt_client.publish("motor/rpm", str(RPM))
```

Fonte: Elaborado pelo autor

Após a conclusão do desenvolvimento do código de controle, iniciou-se a criação do dashboard no Node-RED, que permite a visualização e ajuste do sistema em tempo real. O painel foi estruturado para oferecer uma interface intuitiva e funcional, possibilitando ao usuário monitorar a velocidade do motor em tempo real, ajustar o setpoint de velocidade e modificar os parâmetros do controlador PID (Kp, Ki, Kd).

A Figura 34 apresenta a estrutura do dashboard desenvolvido. O nó MQTT "Receber RPM" é responsável por escutar mensagens no tópico motor/velocidade, repassando o valor tanto para um display numérico (exibindo o RPM atual) quanto para um gráfico de linha, permitindo o monitoramento contínuo do desempenho do motor. Quando o usuário ajusta o setpoint, o novo valor é enviado automaticamente para o tópico motor/setpoint, possibilitando que o controle PID adapte-se às novas condições de operação. Da mesma forma, os inputs numéricos para Kp, Ki e Kd são utilizados para ajustar os parâmetros do controlador, sendo enviados para o sistema ao serem atualizados.

Figura 34 - Desenvolvimento Node-RED



Fonte: Elaborado pelo autor

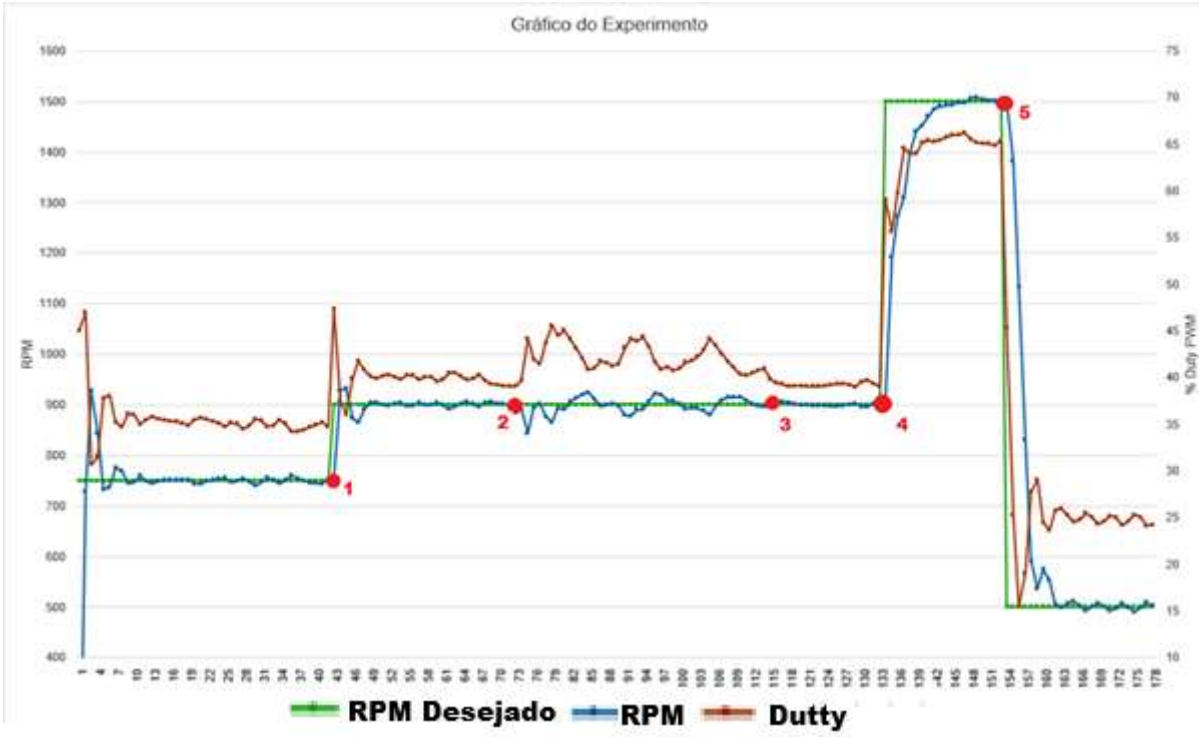
Com o desenvolvimento concluído, iniciaram-se os testes no ambiente de bancada para validar o desempenho da plataforma no controle de velocidade do motor. A Figura 35 apresenta o gráfico gerado a partir dos dados coletados durante a execução do experimento, permitindo uma análise visual detalhada da resposta

dinâmica do sistema de controle. Inicialmente, foi definido um valor de referência de 750 RPM, sendo possível observar que o sistema alcança essa velocidade de forma estável. No ponto 1, o valor de referência foi aumentado para 900 RPM, com o objetivo de avaliar a capacidade do sistema em responder a uma solicitação de aceleração. A curva evidencia que o motor atingiu a nova referência com estabilidade, demonstrando a eficiência do controle PID.

No ponto 2, uma carga mecânica foi aplicada ao eixo do motor por meio da resistência variável. A resposta do sistema mostra o reestabelecimento do equilíbrio, indicando a atuação correta do controlador em compensar a perturbação. No ponto 3, a carga foi removida e a velocidade rapidamente foi corrigida pelo controlador, restaurando o sistema à velocidade de referência.

Já no ponto 4, a referência foi elevada para 1500 RPM, com o objetivo de observar a resposta do sistema frente a uma demanda de aceleração mais intensa. O gráfico mostra uma subida contínua da velocidade até o novo ponto de equilíbrio, novamente sem oscilações significativas. Por fim, no ponto 5, o valor de referência foi reduzido para 500 RPM, simulando uma desaceleração brusca. O sistema respondeu prontamente à mudança, reduzindo a velocidade de forma controlada e estável, validando assim a capacidade da plataforma em lidar com transições rápidas e variações de carga, mantendo o controle preciso da velocidade ao longo de todo o experimento.

Figura 35 – Gráfico do experimento

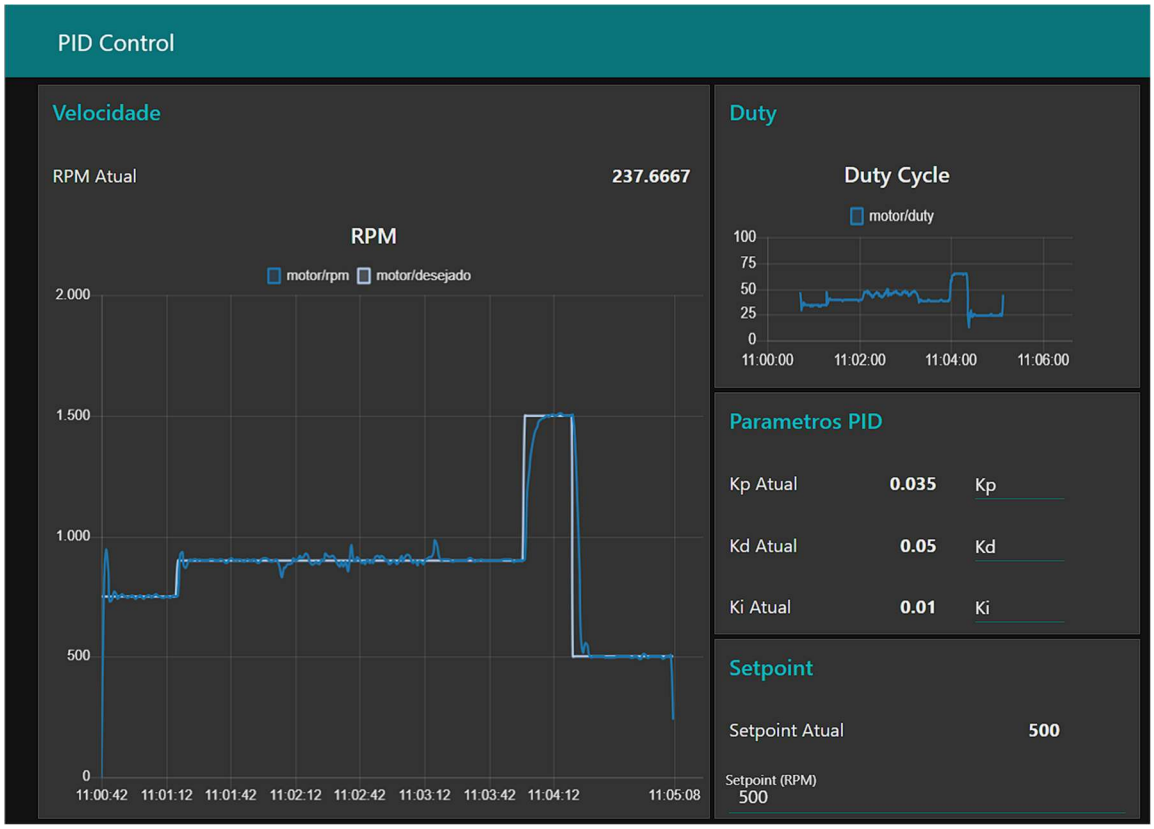


Fonte: Elaborado pelo autor

Durante o teste, a plataforma monitorava continuamente a velocidade real do motor, obtida pelo encoder, e realizava ajustes automáticos no sinal PWM de acordo com os cálculos do algoritmo PID. A transmissão dos dados via MQTT possibilitou o monitoramento em tempo real do sistema através do dashboard no Node-RED.

Com o experimento em funcionamento, analisou-se o envio e recepção das mensagens MQTT, bem como a exibição dos dados no dashboard, conforme ilustrado na Figura 36. Adicionalmente, o vídeo disponível no link https://youtu.be/rbEFqCIbv0c?si=TKTA0UZk-yRIU58f apresenta uma demonstração visual do desempenho do sistema, destacando a sequência de ajustes realizados durante o teste.

Figura 36 - Dashboard de dados



Fonte: Elaborado pelo autor

A validação da plataforma, utilizando o MicroPython em um cenário real de controle de processos, demonstrou que o sistema não apenas é funcional, mas também altamente adaptável a mudanças de carga e outras condições variáveis. O uso do MicroPython provou-se vantajoso, simplificando o processo de desenvolvimento e oferecendo uma alternativa eficiente em comparação às linguagens tradicionais, normatizadas pela IEC 61131-3, que, embora poderosas, tendem a demandar mais tempo e complexidade de configuração.

Esse teste evidenciou que a plataforma desenvolvida, além de ser uma alternativa acessível e open source, possui a confiabilidade e o desempenho necessários para uma aplicação prática de IIoT. A capacidade de adaptar o sistema a condições reais reforça sua adequação para cenários industriais de controle, mostrando-se uma ferramenta eficaz e de fácil uso para profissionais e desenvolvedores. Além disso, a facilidade de desenvolvimento e depuração no MicroPython contribui para um tempo de implementação reduzido, reforçando sua viabilidade como alternativa para sistemas industriais flexíveis.

5.3. Discussões

A plataforma de automação desenvolvida neste projeto apresenta vantagens frente aos CLPs convencionais, como os da Siemens (ex. S7-1200) ou da Allen-Bradley (ex. ControlLogix e MicroLogix), que são amplamente adotadas na indústria, reconhecidas por sua robustez e confiabilidade. No entanto, essas soluções apresentam limitações significativas em termos de custo, acessibilidade e flexibilidade, especialmente para pequenas e médias empresas e para desenvolvedores independentes.

Em termos de custo de hardware, esses CLPs são substancialmente mais caros devido à complexidade dos hardwares, aos recursos avançados e à marca. Um controlador básico, como o Siemens S7-1200, tem um preço médio de US$500, sem incluir os módulos de expansão e fontes de alimentação, que são essenciais para operar em sistemas completos. Em contraste, a plataforma proposta, baseada no microcontrolador RP2040, reduz drasticamente os custos de hardware, com uma faixa de preço entre US$50 a US$100. Essa característica amplia a acessibilidade para pequenas empresas e iniciativas que necessitam de orçamentos reduzidos sem sacrificar a capacidade de automação.

Além disso, o custo de licenciamento dos softwares de programação, configuração e monitoramento também diferencia as plataformas. Em CLPs tradicionais, ferramentas como o Siemens TIA Portal ou o Rockwell Studio 5000, que são essenciais para a operação, exigem licenças que custam entre US$1.000 a US$3.000, dependendo das funcionalidades. No presente projeto, o uso do OpenPLC e do MicroPython, ambos softwares open source, elimina essas despesas, permitindo o desenvolvimento e integração sem custo adicional. Essa abordagem torna o projeto acessível a um público mais amplo, facilitando seu uso e disseminação.

Outro ponto relevante é a flexibilidade do desenvolvimento. As arquiteturas de CLPs tradicionais são fechadas, tornando a customização uma tarefa restrita e geralmente onerosa. Com o OpenPLC, a implementação das linguagens da IEC 61131-3 facilita a integração sem as limitações de um sistema proprietário, oferecendo uma ampla gama de possibilidades para projetos industriais. A adição do MicroPython ainda reforça a flexibilidade, permitindo que a plataforma seja utilizada tanto para IoT quanto para automação industrial, atendendo a requisitos diversos de forma modular.

Existem também soluções comerciais de baixo custo, como controladores da série PLC Arduino e microcontroladores ESP32, que são frequentemente utilizados para prototipagem e projetos simples de hobby. No entanto, esses dispositivos não foram projetados para automação industrial e apresentam limitações importantes em comparação à plataforma proposta, como por exemplo, soluções como o ESP32 apresentam limitações na quantidade e tipos de entradas e saídas, enquanto a plataforma proposta abrange um conjunto completo para atender a diferentes requisitos industriais, como saídas PWM, entradas analógicas programáveis e portas dedicadas a encoder. Essas funcionalidades ampliam a flexibilidade e permitem a adaptação rápida a diferentes cenários industriais.

Em resumo, o desenvolvimento desta plataforma open source de automação e controle oferece uma alternativa viável e eficiente aos sistemas de automação tradicionais, reforçando a relevância de soluções de baixo custo e acessíveis para o setor industrial. A sua implementação não apenas facilita a prototipagem e a rápida adaptação, mas também oferece às empresas uma opção sustentável e econômica para inovação contínua. Dessa forma, o projeto traz ao mercado uma alternativa escalável e versátil, permitindo que as indústrias invistam em automação de maneira mais acessível e prática, atendendo tanto a aplicações de pequena escala quanto a demandas mais complexas.

6. CONCLUSÕES

A utilização de controladores industriais como parte de um sistema de Internet das Coisas Industrial (IIoT) se tornou fundamental para qualquer sistema de automação atualmente. Assim, o desenvolvimento de uma alternativa que atenda às necessidades da indústria, especialmente no que diz respeito à conectividade e comunicação, é vital para o avanço da implementação da Indústria 4.0 no Brasil. Neste contexto, este trabalho apresentou o desenvolvimento de uma Plataforma de Automação e Controle Open Source voltada para aplicações de IIoT, abrangendo elementos de hardware, software e comunicação.

A plataforma desenvolvida se destaca como uma alternativa alta flexibilidade para aplicações industriais, garantindo não apenas a confiabilidade e robustez necessárias, mas também a possibilidade de modificação e personalização conforme as demandas específicas de cada cliente. Essa capacidade de adaptação é essencial para a integração com outros serviços, alinhando-se aos requisitos da IIoT. O desenvolvimento do hardware foi realizado de forma a permitir que novos avanços possam emergir a partir da plataforma apresentada, o que foi garantido pela cuidadosa escolha dos componentes e da serialização, promovendo uma maior flexibilidade e facilidade na alteração de parâmetros.

A escolha do OpenPLC como software controlador se revelou adequada, demonstrando-se uma excelente alternativa aos softwares proprietários. Sua capacidade de integração com controladores existentes e a utilização dos padrões da norma IEC 61131-3, que atualmente dominam as linhas de produção industrial, reforçam a relevância da plataforma no contexto atual. Todos os desenvolvimentos foram testados em um cenário real de IIoT, validando a plataforma em operações robustas e assegurando um controle lógico confiável e uma comunicação eficaz.

Em um cenário onde os processos industriais estão cada vez mais interconectados, a capacidade de adaptar o sistema a diferentes aplicações sem incorrer em altos custos de reengenharia se torna uma vantagem estratégica. A flexibilidade proporcionada pelo MicroPython, juntamente com a disponibilidade de bibliotecas reutilizáveis, capacita engenheiros a desenvolver funcionalidades específicas para suas aplicações sem a complexidade associada às linguagens convencionais de controle industrial. Isso possibilita um desenvolvimento ágil, iterativo

e de fácil manutenção, estabelecendo um caminho promissor para futuras inovações no campo da automação industrial.

Em um setor tradicionalmente dominado por soluções proprietárias, este projeto representa uma contribuição significativa ao oferecer uma alternativa de alto desempenho, acessível e customizável. Com um projeto completamente open source, os usuários têm total liberdade para explorar, modificar e adaptar o sistema conforme suas necessidades, sem a dependência de licenças ou restrições impostas por fabricantes. Essa liberdade é crucial no contexto da Indústria 4.0, onde a personalização e conectividade são vitais para garantir a competitividade e eficiência das operações industriais.

Além disso, a plataforma atende à crescente demanda por sistemas de automação mais adaptáveis e integrados à Internet das Coisas Industrial (IIoT), um requisito essencial na Indústria 4.0. A utilização de ferramentas de software livre, aliada à documentação disponível, contribui para uma relação mais estreita entre a academia e a indústria, oferecendo uma base prática para testes, implementações experimentais e aplicações industriais.

Trabalhos futuros buscarão atualizar a plataforma para torná-la mais compatível com painéis industriais e implementar novas formas de conectividade. A disponibilização da documentação completa no GitHub não apenas reforça a colaboração e a continuidade do trabalho, mas também torna a plataforma facilmente replicável e expansível. Esse caráter open source incentiva uma adoção mais ampla e melhorias contínuas por parte da comunidade acadêmica e de novos desenvolvedores, promovendo um ambiente colaborativo e inovador.

REFERÊNCIAS

A. Karmakar, N. Dey, T. Baral, M. Chowdhury and M. Rehan, "Industrial Internet of Things: A Review," 2019 International Conference on Opto-Electronics and Applied Optics (Optronix), Kolkata, India, 2019, pp. 1-6, doi: 10.1109/OPTRONIX.2019.8862436.

A. W. Colombo et al., "A 70-Year Industrial Electronics Society Evolution Through Industrial Revolutions: The Rise and Flourishing of Information and Communication Technologies," in IEEE Industrial Electronics Magazine, vol. 15, no. 1, pp. 115-126, March 2021, doi: 10.1109/MIE.2020.3028058.

Automation Direct, “ProductivityOpen: Open-Source Agility Meets Industrial-Grade Toughness”, https://www.automationdirect.com/open-source/home, (acesso em Set. 5, 2023).

Autonomy. “1.1 OpenPLC Overview” .https://autonomylogic.com/docs/openplc-overview/ (acesso em Set. 5, 2023).

Autonomy. “3.3 ADDING NEW BLOCKS TO OPENPLC EDITOR’S LIBRARY”.https://autonomylogic.com/docs/3-3-adding-new-blocks-to-openplc-editors-library/ (acesso em Set. 5, 2023).

B. Bajic, A. Rikalovic, N. Suzic and V. Piuri, "Industry 4.0 Implementation Challenges and Opportunities: A Managerial Perspective," in IEEE Systems Journal, vol. 15, no. 1, pp. 546-559, March 2021, doi: 10.1109/JSYST.2020.3023041.

C. Andrei, G. Tudor, M. Arhip-Călin, G. Fierăscu and C. Urcan, "Raspberry Pi, an Alternative Low-Cost PLC," 2020 International Symposium on Fundamentals of Electrical Engineering (ISFEE), Bucharest, Romania, 2020, pp. 1-6, doi: 10.1109/ISFEE51261.2020.9756175.

Contieri, P.G.S., Hassui, A., Santa-Eulalia, L.A., Sigahi, T.F.A.C., Rampasso, I.S., Moraes, G.H.S.M.d. and Anholon, R. (2023), "Difficulties and challenges in the modernization of a production cell with the introduction of Industry 4.0 technologies", Benchmarking: An International Journal, Vol. ahead-of-print No. ahead-of-print. https://doi.org/10.1108/BIJ-02-2023-0071

D. G. S. Pivoto, L. F. F. de Almeida, R. da Rosa Righi, J. J. P. C. Rodrigues, A. B. Lugli, and A. M. Alberti, "Cyber-physical systems architectures for industrial internet

of things applications in Industry 4.0: A literature review," Journal of Manufacturing Systems, vol. 58, pp. 176-192, 2021. DOI: 10.1016/j.jmsy.2020.11.017.

FACTS Engineering, “Products”, https://facts-eng.com/products/, (acesso em Set. 5, 2023).

G. Vieira, J. Barbosa, P. Leitão and L. Sakurada, "Low-Cost Industrial Controller based on the Raspberry Pi Platform," 2020 IEEE International Conference on Industrial Technology (ICIT), Buenos Aires, Argentina, 2020, pp. 292-297, doi: 10.1109/ICIT45562.2020.9067148.

Gómez, R., et al. (2020). "Rapid Prototyping with MicroPython in Industrial IoT Systems." Automation Science and Engineering Review, 37(4), 235-249.

Huang, L., & Wang, J. (2022). "Open Hardware Solutions in Industrial Automation: Trends and Perspectives." Industrial Electronics Journal, 39(2), 87-98.

Jones, P., & García, L. (2020). "Comparative Study of MicroPython and C in Industrial Automation." International Journal of Automation and Control, 29(1), 80-95.

Jones, P., & Silva, R. (2020). "Rapid Prototyping in Industrial IoT Systems: The Role of MicroPython." International Journal of IoT and Automation, 12(2), 115-130.

Jones, P., & Smith, R. (2020). "Cost Implications of Traditional PLCs in the Age of Open Source and PC-Based Control." Automation Technology Review, 18(2), 75-90.

L. E. Fernandez-Rodriguez, J. Rodriguez-Resendiz and M. A. Martinez-Hernandez, "An open-source FPGA-based control and data acquisition hardware platform," 2021 XVII International Engineering Congress (CONIIN), Queretaro, México, 2021, pp. 1-11, doi: 10.1109/CONIIN54356.2021.9634743.

Lemos, A., Pereira, M., & Reis, C. (2021). "Open Source Automation Platforms and the Shift Away from Traditional PLCs: A Comparative Analysis." Journal of Industrial Engineering, 56(2), 109-127.

Lundh, M., Svensson, H., & Hellström, J. (2019). "Challenges and Opportunities in Modernizing PLC Systems: A Case Study on Flexibility and Control." Journal of Industrial Automation, 44(1), 23-37.

M. A. Sehr et al., "Programmable Logic Controllers in the Context of Industry 4.0," in IEEE Transactions on Industrial Informatics, vol. 17, no. 5, pp. 3523-3533, May 2021, doi: 10.1109/TII.2020.3007764.

M. Azarmipour, H. Elfaham, C. Gries and U. Epple, "PLC 4.0: A Control System for Industry 4.0," IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 2019, pp. 5513-5518, doi: 10.1109/IECON.2019.8927026.

M. Jbair, B. Ahmad, M. H. Ahmad and R. Harrison, "Industrial cyber physical systems: A survey for control-engineering tools," 2018 IEEE Industrial Cyber-Physical Systems (ICPS), St. Petersburg, Russia, 2018, pp. 270-276, doi: 10.1109/ICPHYS.2018.8387671.

Miller, A., et al. (2021). "Customer Experience and Technology Adoption in Industrial Automation." Journal of Automation and Control Engineering, 28(3), 134-150.

Müller, P. et al. (2020). "Customization in Industrial Automation Using Open Source Platforms." International Journal of Industrial Informatics, 25(2), 101-113.

Parmentier, G., Castellano, S., & Rinaldi, M. (2020). "Customization and Innovation in Industrial Automation: Enhancing Customer Satisfaction through Tailored Solutions." Industrial Engineering Journal, 33(4), 147-158.

Pereira, S. & Lima, T. (2022). "Tailoring Industrial Automation Systems with Open Source Hardware: Benefits and Challenges." Automation Technology Review, 49(3), 201-223.

R. Rout, S. K. Samal and G. Modak, "Design of Low-Cost PLC to Drive Three-Phase Induction Motor using Variable Frequency Drive," 2024 Control Instrumentation System Conference (CISCON), Manipal, India, 2024, pp. 1-5, doi: 10.1109/CISCON62171.2024.10696682.

Rajput, A., Singh, N., & Kumar, S. (2020). "Operational Efficiency through Flexible Automation Systems: Moving Beyond Traditional PLCs." Automation & Robotics Journal, 41(1), 58-70.

S. A. Namekar and R. Yadav, "Programmable Logic Controller (PLC) and its Applications," International Journal of Innovative Research in Technology, vol. 6, no. 11, pp. 372, April 2020, ISSN: 2349-6002.

Smith, A., & Jones, P. (2020). "MicroPython in Industrial Automation: A New Frontier." Journal of Embedded Systems, 23(2), 101-115.

W. Z. Khan, M. H. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, "Industrial internet of things: Recent advances, enabling technologies and open challenges," Computers & Electrical Engineering, vol. 81, p. 106522, 2020. DOI: 10.1016/j.compeleceng.2019.106522.

Wang, J., Liu, H., & Zhang, Y. (2021). "Flexibility and Interoperability in Open-Source Automation Systems." Journal of Industrial Automation, 29(2), 88-102.

Wang, Z., Liu, J., & Zhang, Y. (2021). "Industrial Connectivity and the Future of PLC Systems in Industry 4.0." IEEE Transactions on Industrial Informatics, 17(6), 4187-4196.