Universidade Federal de São Carlos - UFSCar
Centro de Ciências Exatas e de Tecnologia - CCET
Departamento da Ciência de Computação - DC
Programa de Pós-Graduação em Ciência da Computação - PPGCC

André Roberto da Silva

Avaliação do desempenho de microcontrolador com arquitetura RISC-V em relação a arquitetura ARM aplicado em processos contínuos utilizando o método de controle PID

São Carlos - SP
2024

ANDRÉ ROBERTO DA SILVA

Avaliação do desempenho de microcontrolador com arquitetura RISC-V em relação a arquitetura ARM aplicado em processos contínuos utilizando o método de controle PID

Versão Original

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Sistemas de Automação e Robótica - SAR

Orientador: Rafael Vidal Aroca

São Carlos - SP

2024

André Roberto da Silva
Avaliação do desempenho de microcontrolador com arquitetura RISC-V em relação a arquitetura ARM aplicado em processos contínuos utilizando o método de controle PID/ André Roberto da Silva. – São Carlos - SP, 2024- 91 p. : il.(alguma color.); 30 cm.
Prof. Dr.Rafael Vidal Aroca
Dissertação de Mestrado –
Universidade Federal de São Carlos - UFSCar
Centro de Ciências Exatas e de Tecnologia - CCET
Departamento da Ciência de Computação - DC
Programa de Pós-Graduação em Ciência da Computação - PPGCC, 2024.
Palavras chave principais: 1. RISC-V. 2. PID. 3. Processos Contínuos.
CDU 02:141:005.7

André Roberto da Silva

Avaliação do desempenho de microcontrolador com arquitetura RISC-V em relação a arquitetura ARM aplicado em processos contínuos utilizando o método de controle PID

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Sistemas de Automação e Robótica - SAR

Orientador: Rafael Vidal Aroca

Folha de aprovação. São Carlos - SP, 2024:

Documento assinado digitalmente
gov.br
RAFAEL VIDAL AROCA
Data: 19/05/2025 10:58:35-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Rafael Vidal Aroca
Presidente da Banca - Membro Interno

Documento assinado digitalmente
gov.br
SIDNEY BRUCE SHIKI
Data: 14/05/2025 15:46:47-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Sidney Bruce Shiki
Membro Titular Interno

Documento assinado digitalmente
gov.br
WILIAN MIRANDA DOS SANTOS
Data: 15/05/2025 10:23:43-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Wilian Miranda dos Santos
Membro Titular Externo

São Carlos - SP
2024

À minha família pelo amor e apoio incondicional

AGRADECIMENTOS

Agradeço à minha esposa Ana Paula Gavioli Silva, meus filhos Arthur Gavioli Silva e Ana Clara Gavioli Silva, aos meus pais, Maria Anselmo da Silva e José Roberto Camargo da Silva, pelo apoio e amor incondicional durante minha ausência em função da dedicação a este trabalho. Agradeço meu orientador e amigo Rafael Vidal Aroca pelo apoio e motivação que possibilitaram o desenvolvimento deste trabalho. Agradeço à Faculdade de Tecnologia SENAI Antonio Adolpho Lobbe por viabilizar os experimentos cedendo seus laboratórios e equipamentos.

"Quanto menos inteligente um homem é, menos misteriosa lhe parece a existência."
(Arthur Schopenhauer)

RESUMO

SILVA, A. R. **Avaliação do desempenho de microcontrolador com arquitetura RISC-V em relação a arquitetura ARM aplicado em processos contínuos utilizando o método de controle PID**. 2025. Dissertação (Mestrado Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Paulo, 2025

O presente trabalho avaliou o desempenho de um microcontrolador baseado na arquitetura open-source RISC-V em relação a outro que possui arquitetura ARM Cortex-M0+, no âmbito das aplicações em sistemas de controle de processos contínuos. O método de controle utilizado foi o PID (Proporcional, Integral e Derivativo), aplicado em uma planta de recalque, sendo o nível de água do reservatório a variável de processo controlada. Além das duas arquiteturas, outra variável investigada foi a linguagem de programação utilizada no desenvolvimento do firmware; optou-se pelo uso das linguagens C/C++ e MicroPython. O projeto incluiu o desenvolvimento de um hardware híbrido que suporta duas plataformas microcontroladas disponíveis no mercado nacional: uma delas é a LuatOS, sistema embarcado que utiliza o microcontrolador ESP32-C3 com arquitetura RISC-V, e a outra é a Raspberry Pi Pico, desenvolvida com o microcontrolador RP2040, que possui um núcleo ARM Cortex-M0+. Os resultados obtidos com este trabalho são úteis para balizar a escolha da arquitetura no projeto e desenvolvimento de dispositivos de controle. Devido ao bom desempenho apresentado pela linguagem MicroPython durante a fase de testes, foi desenvolvido um tutorial de programação em blocos com a plataforma BIPES para controle PID em sistemas embarcados com núcleo RISC-V, possibilitando assim maior acessibilidade ao uso dessa tecnologia emergente em projetos de automação e controle.

Palavras-chave: RISC-V, PID, Processos Contínuos.

ABSTRACT

SILVA, A. R. **Avaliação do desempenho de microcontrolador com arquitetura RISC-V em relação a arquitetura ARM aplicado em processos contínuos utilizando o método de controle PID**. 2025. Dissertação (Mestrado Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Paulo, 2025

The present work evaluated the performance of a microcontroller based on the open-source RISC-V architecture in comparison to another one with ARM Cortex-M0+ architecture, within the scope of applications in continuous process control systems. The control method used was PID (Proportional, Integral, and Derivative), applied in a booster pump system, with the reservoir water level as the controlled process variable. In addition to the two architectures, another variable investigated was the programming language used for firmware development; C/C++ and MicroPython were chosen. The project included the development of a hybrid hardware that supports two commercially available microcontroller platforms in the national market: one of them is LuatOS, an embedded system using the ESP32-C3 microcontroller with RISC-V architecture, and the other is the Raspberry Pi Pico, developed with the RP2040 microcontroller, which features an ARM Cortex-M0+ core. The results obtained from this work are useful for guiding the choice of architecture in the design and development of control devices. Due to the good performance shown by the MicroPython language during the testing phase, a block programming tutorial with the BIPES platform for PID control on embedded systems with a RISC-V core was developed, thus enabling greater accessibility to this emerging technology in automation and control projects.

Keywords: RISC-V, PID, Continuous Processes.

LISTA DE ILUSTRAÇÕES

Figura 1 – Planta de processos contínuos. 21
Figura 2 – Paralelo de diagrama de sinal de controle em malha fechada e sistema de controle utilizado. 24
Figura 3 – Diagrama de sinal de um sistema de controle em malha fechada. 28
Figura 4 – Camadas da hierarquia dos componentes de um comutador. 31
Figura 5 – Diagrama de blocos do processador ARM Cortex M0+. 32
Figura 6 – Empresas filiadas ao projeto RISC-V. 34
Figura 7 – Microcontrolador RP2040. 35
Figura 8 – Diagrama de blocos do microcontrolador RP2040. 37
Figura 9 – Microcontrolador ESP32C3 37
Figura 10 – Diagrama de blocos do microcontrolador ESP32C3 38
Figura 11 – Plataforma Arduino UNO 40
Figura 12 – Plataforma Raspberry Pi Modelo 3B+ 41
Figura 13 – Plataforma ESP8266 Node MCU 41
Figura 14 – Plataforma LuatOS 51
Figura 15 – Plataforma Raspberry Pi Pico 52
Figura 16 – Transdutor de Pressão com saída 4 a 20 mA 53
Figura 17 – Inversor de Frequência 54
Figura 18 – Conversor AD modelo ADS115 com 16 bits de resolução 54
Figura 19 – Conversor DA MCP4725 com 12 bits de resolução 55
Figura 20 – Conversor de Tensão Step Down 55
Figura 21 – Conexões elétricas das plataformas LuatOS e Raspberry Pi Pico 56
Figura 22 – Interface de entrada e saída analógica 56
Figura 23 – Placa de circuito impresso desenvolvida 57
Figura 24 – Placa eletrônica com as plataformas LuatOS e Raspberry Pi Pico . . . 58
Figura 25 – Conjunto *case* e tampa desenvolvidos 59
Figura 26 – Conjunto completo montado 59
Figura 27 – Fluxograma para desenvolvimento do *firmware* em C/C++ 60
Figura 28 – Fluxograma para desenvolvimento do *firmware* em MicroPython 61
Figura 29 – Visão geral de ferramentas disponíveis na plataforma ANACONDA . . 62
Figura 30 – Gráfico do valor do nível em função do tempo com ganho crítico 75 . . 63
Figura 31 – Trecho de código para cálculo dos ganhos K_p, K_i e K_d 64
Figura 32 – Gráfico de PV em função do tempo, núcleo ARM Cortex M0+ em linguagem de programação C/C++ 65
Figura 33 – Gráfico de PV em função do tempo, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20% 65

Figura 34 – Gráfico de PV em função do tempo resposta transitória, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20% 66
Figura 35 – Gráfico de PV em função do tempo regime permanente, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20% 67
Figura 36 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação C/C++ 68
Figura 37 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20% 69
Figura 38 – Gráfico de PV em função do tempo resposta transitória, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20% . . . 69
Figura 39 – Gráfico de PV em função do tempo regime permanente, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20% . . . 70
Figura 40 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação MicroPython 71
Figura 41 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20% 72
Figura 42 – Gráfico de PV em função do tempo resposta transitória, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20% 72
Figura 43 – Gráfico de PV em função do tempo regime permanente, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20% 73
Figura 44 – Gráfico de PV em função do tempo com os três arranjos experimentais e SP 20% 74
Figura 45 – Trecho de código para cálculo do SSE em relação a plataforma Raspberry Pi Pico 75
Figura 46 – Gráfico de PV em função do tempo utilizando dados gerados com recursos em nuvem, núcleo RISC-V em linguagem de programação MicroPython desenvolvido pela plataforma BIPES 76
Figura 47 – Esquema elétrico completo do circuito desenvolvido 79
Figura 48 – *Loop* principal do código em linguagem C/C++ 80
Figura 49 – Rotina de controle PID em linguagem C - Parte 1 81
Figura 50 – Rotina de controle PID em linguagem C - Parte 2 82
Figura 51 – Loop principal do código em linguagem MicroPython 83
Figura 52 – Rotina de controle PID em linguagem MicroPython - Parte 1 84
Figura 53 – Rotina de controle PID em linguagem MicroPython - Parte 2 85
Figura 54 – Rotina de controle PID em linguagem MicroPython - Parte 3 86
Figura 55 – Rotina de controle PID em linguagem MicroPython - Parte 4 87

LISTA DE TABELAS

Tabela 1 – Tabela de ganhos para o método da curva de reação. 29
Tabela 2 – Tabela de ganhos para o método do ganho crítico. 30
Tabela 3 – Questões de Pesquisa 42
Tabela 4 – Revisão Sistemática da Literatura 43
Tabela 5 – Comparação entre os microcontroladores RP2040 e ESP32-C3 52
Tabela 6 – Valores dos parâmetros de sintonia 64
Tabela 7 – Métricas estatísticas dos valores de PV no regime permanente, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20% 67
Tabela 8 – Métricas estatísticas dos valores de PV no regime permanente, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20% 70
Tabela 9 – Métricas estatísticas dos valores de PV no regime permanente, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20% 73

SUMÁRIO

1 INTRODUÇÃO 21
1.1 Contextualização 22
1.2 Abordagem utilizada 24
1.3 Importância da pesquisa 25
1.4 Objetivo 25
1.5 Estrutura do Texto 25

2 REVISÃO BIBLIOGRÁFICA 27
2.1 Fundamentação Teórica 27
2.1.1 Sistema de Controle 27
2.1.2 Controle PID 28
2.1.3 ARM 30
2.1.4 RISC-V 33
2.1.5 RP2040 35
2.1.6 ESP32-C3 37
2.1.7 Sistemas Embarcados 40
2.2 Trabalhos Relacionados 42

3 MATERIAIS E MÉTODOS 47
3.1 Materiais 47
3.2 Métodos 48

4 DESENVOLVIMENTO 51
4.1 Hardware 51
4.2 Firmware 60

5 RESULTADOS 63
5.1 Sintonia 63
5.2 ARM Cortex M0+ em Linguagem C 64
5.3 RISC-V em Linguagem C 67
5.4 RISC-V em Linguagem MicroPython 71
5.5 Análise Comparativa de Dados 73
5.6 Bipes e IoT 75

6 CONCLUSÃO 77
6.1 Estado atual 77
6.2 Estado futuro 77

7 APÊNDICE 79

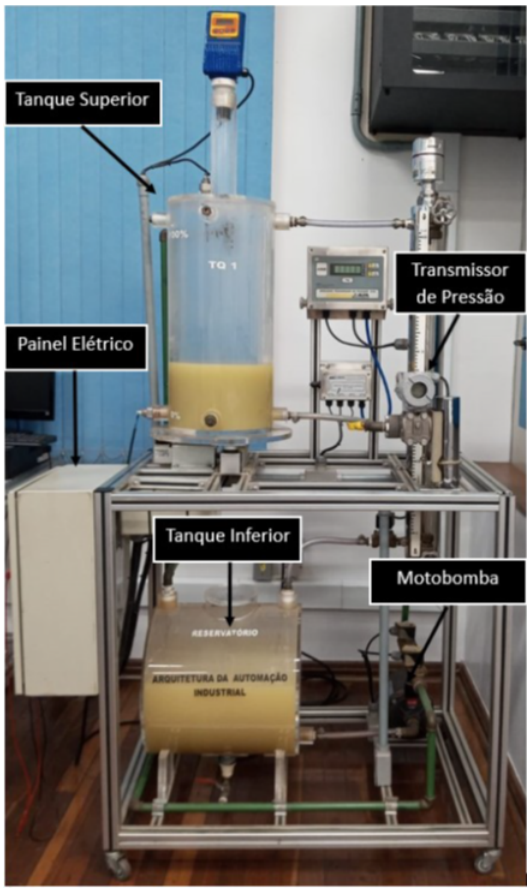
7.1 Apêndice A - Esquema elétrico 79
7.2 Apêndice B - Código em linguagem C/C++ 80
7.3 Apêndice C - Código de controle PID em linguagem C/C++ 81
7.4 Apêndice D - Código em linguagem MicroPython 83
7.5 Apêndice E - Código de controle PID em linguagem MicroPython . 84

REFERÊNCIAS 89

1 . INTRODUÇÃO

Este capítulo apresenta uma contextualização da área de conhecimento na qual este trabalho foi desenvolvido, também são explicitados a abordagem utilizada, o objetivo geral do trabalho e a forma de organização do documento. Os ensaios realizados durante o desenvolvimento deste trabalho foram feitos em uma planta de processos contínuos, apresentada na Figura 1, na qual estão destacados os componentes principais que foram utilizados durante as etapas de coleta de dados. O controle PID (Proporcional Integral Derivativo) é aplicado com o objetivo de regular o nível do Tanque Superior, o conjunto Motobomba é utilizado para aumentar o nível do tanque e tem sua vazão controlada pelo inversor de frequência instalado no Painel Elétrico, a aferição do nível é feita pelo Transmissor de Pressão e o escoamento do líquido para o Tanque Inferior é impulsionado pela gravidade. Todos dispositivos e equipamentos instalados são de uso industrial, o que aproxima este trabalho da aplicação direta na indústria. Foi acrescentado ao conjunto de tecnologias instaladas na planta o controlador desenvolvido ao longo dos trabalhos realizados neste projeto, controlador este, que será apresentado em detalhes no capítulo 5.

Figura 1 – Planta de processos contínuos.



Fonte: Autor.

1.1 Contextualização

A quarta Revolução Industrial, também conhecida como Indústria 4.0, modificou a forma de se pensar um processo produtivo. Os pilares da Indústria 4.0 proporcionam a tomada de decisão com base em análise de dados, resultando em uma decisão mais precisa, seja sobre o processo produtivo ou a otimização da utilização dos ativos. A evolução dos sistemas automatizados industriais segue na direção da conectividade, o advento da internet das coisas IoT (sigla em inglês para *internet of things*) impulsionou a demanda por sistemas automatizados industriais com esse recurso. Neste contexto, o uso dos sistemas embarcados em soluções de automação industrial se torna cada vez mais presente, considerando que tais plataformas oferecem uma alternativa de baixo custo com recursos de conectividade (MAIER; SHARP; VAGAPOV, 2017).

O Brasil ainda não conseguiu estabelecer uma estratégia eficaz para a modernização da indústria refletindo em uma desindustrialização e incapacidade de competir globalmente. Mesmo com as oportunidades oferecidas pela Indústria 4.0, sem investimentos contínuos em P&D, o Brasil pode ficar ainda mais atrás no cenário mundial (RODRIGUES; OURIQUES, 2022). O contexto da indústria brasileira pode demandar uma implementação adaptada frente aos conceitos desenvolvidos em países mais avançados, são necessários esforços para o desenvolvimento de trabalhos que promovam uma melhor percepção das indústrias sobre essas tecnologias e seus benefícios potenciais, contribuindo para o entendimento dos impactos reais da Indústria 4.0 (DALENOGARE et al., 2018).

No que tange as aplicações de dispositivos IoT existem vários aspectos relevantes a serem considerados, como a eficiência computacional, baixo consumo de energia, conectividade sem fio e protocolos de comunicação implementados em hardware. Existe uma vasta gama de microcontroladores disponíveis no mercado mundial que são capazes de coletar, processar e transmitir dados (PLAUSKA; LIUTKEVIčIUS; JANAVIčIūTė, 2022).

Outra característica importante em relação aos dispositivos IoT é o tamanho reduzido: em geral, esses dispositivos são utilizados para o desenvolvimento da camada digital em um processo industrial já existente, sendo assim, o menor impacto nas instalações é por vezes condição essencial para a viabilidade técnica do projeto. Por esse motivo, soluções IoT são em grande maioria implementadas por meio de microcontroladores, dispositivos estes que atendem a essas especificações (MAIER; SHARP; VAGAPOV, 2017).

Em relação a arquitetura do processador utilizado pelos microcontroladores se destaca em aplicações de sistemas embarcados a ARM (Advanced RISC Machine), altamente eficiente em termos de consumo de energia, projetada para aplicações de sistemas embarcados que necessitam de um processador otimizado. Suas características e benefícios incluem um conjunto de instruções que combina alta densidade de código com desempenho

de 32 bits, suporte para acesso I/O em ciclo único, otimização no controle de energia por meio do modo sleep, execução rápida de código, busca de código otimizada em memória flash e ROM, manipulação determinística e de alto desempenho de interrupções para aplicações críticas em real time, temporização determinística de ciclos de instrução dentre outas características (ARM, 2012)

Tendo como referência o requisito conectividade, uma escolha que ganhou notoriedade nos últimos anos foi o SoC (sigla em inglês para System On Chip) ESP32, fabricado pela empresa chinesa ESPRESSIF. Esse dispositivo é implementado por meio de um processador Xtensa da Cadence Tensilica. Tal processador permite o desenvolvimento de soluções para domínios com eficiência energética, é construído em torno de uma arquitetura RISC modular altamente flexível de 32 bits. Especificamente, o ESP32 tem como características a capacidade de trabalhar em ambientes industriais com condições de temperatura que variam de -40 °C até +125 °C, o baixo consumo de energia considerando diversos modos escalonáveis, o alto nível de integração com antenas incorporadas, amplificadores de potência e filtros. Porém, a característica que engaja este dispositivo no desenvolvimento de soluções IoT é a disponibilização de um chip híbrido WiFi e Bluetooth, o que possibilita a utilização dessas tecnologias através de interfaces populares como SPI/SDIO ou I2C/UART (ESPRESSIF, 2023a)

As duas arquiteturas supracitadas são exemplos de arquiteturas proprietárias, sendo assim, para que uma empresa desenvolva seu microcontrolador com esse tipo de arquitetura se faz necessário o pagamento de royalties aos desenvolvedores. Por outro lado, nos últimos anos, observa-se um crescente uso da arquitetura RISC-V, uma arquitetura com um conjunto de instruções abertas ISA (sigla em inglês para Instruction Set Architecture) desenvolvida pela UC UC Berkeley em 2010. Essa arquitetura foi projetada para simplificar o desenvolvimento de processadores em chip de silício, as instruções foram concebidas considerando uma estrutura previsível e uniforme, ou seja, altamente regular, dispõe também de um modelo de memória direta dispensando o uso de instruções complexas para acesso à memória de dados. A arquitetura RISC-V, está se tornando um padrão industrial em ascensão para a construção de processadores embarcados (SANTOS et al., 2021);

Considerado a rápida aceitação de uma arquitetura aberta no desenvolvimento de sistemas embarcados e a crescente utilização dessa arquitetura por diversas empresas, aumenta o interesse em validar o desempenho dessa tecnologia emergente em nichos específicos, dentre eles o controle de processos contínuos, que são operações industriais em que a produção ocorre de forma ininterrupta, com entrada constante de matérias-primas e saída contínua de produtos, principalmente na implementação de métodos de controles clássicos como o PID.

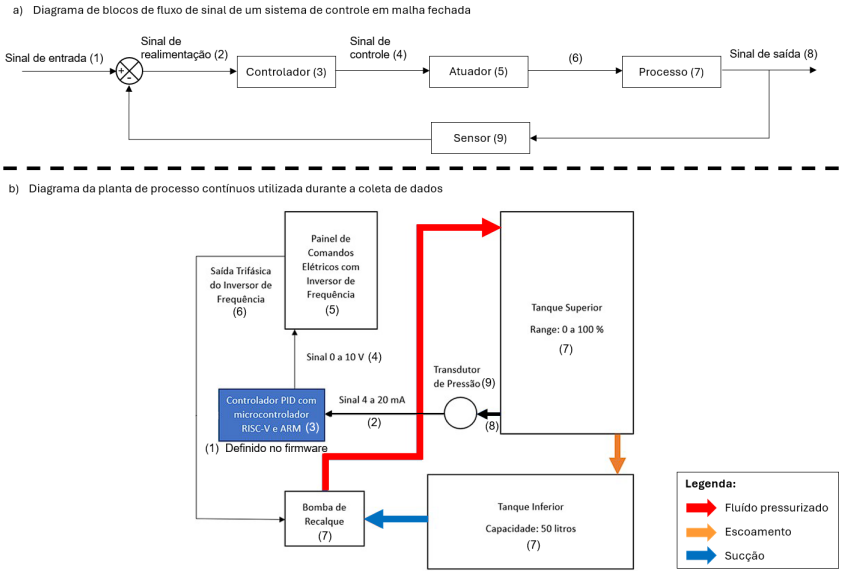
1.2 Abordagem utilizada

Diante do contexto apresentado, a proposta deste trabalho foi uma análise da viabilidade técnica para o uso de um microcontrolador com arquitetura RISC-V no controle da variável do processo PV (sigla em inglês para *process value*), em um sistema de controle de processo contínuo. A principal componente de análise é PV sendo esta comparada em relação ao valor ajustado ou SP (sigla em inglês para *setpoint*) definido, considera-se também o desempenho do sistema em função da linguagem de programação utilizada para o desenvolvimento do *firmware*.

O presente trabalho apresenta um sistema de controle de nível através da técnica de controle PID, para tal, foram utilizadas plataformas embarcadas com SoC's (sigla em inglês para System-on-a-Chip) desenvolvidos a partir de duas arquiteturas, RISC-V e ARM Cortex M0+, foram utilizados módulos de sensores e atuadores para leitura e escrita de sinal, todos recursos utilizados são de aplicação no âmbito educacional e industrial. Para objeto de estudo foi implementado um sistema de controle de nível onde, após definido o SP e os parâmetros de sintonia, o valor do PV foi registrado ao longo do tempo, buscando assim, verificar a eficiência do sistema de controle.

A Figura 2 ilustra um paralelo entre o diagrama de blocos de sinal de um sistema de controle em malha fechada e a arquitetura geral do sistema proposto neste trabalho, no qual os dois microcontroladores comparados são os dispositivos principais que determinam o funcionamento pleno do sistema.

Figura 2 – Paralelo de diagrama de sinal de controle em malha fechada e sistema de controle utilizado.



Fonte: Autor.

O valor da variável de processo é enviada ao microcontrolador por meio de um sinal de corrente proveniente de um transdutor de pressão, o sinal de controle é um sinal de tensão gerado pelo microcontrolador e demais componentes do hardware.

O elemento norteador para o desenvolvimento do presente trabalho é o objetivo geral descrito na seção 1.4.

1.3 Importância da pesquisa

Pretende-se contribuir para o meio científico apresentando a análise de dados provenientes de ensaios realizados com um sistema de controle em malha fechada, desenvolvido com um microcontrolador que utiliza a arquitetura aberta RISC-V. Todo o material produzido ao longo da pesquisa foi disponibilizado para acesso público na plataforma GitHub, permitindo assim a replicação dos experimentos e distribuição dos conjuntos de dados obtidos. Para promover uma forma mais acessível de implementação do sistema de controle, foi elaborado um tutorial de aplicação utilizando a plataforma BIPES.

1.4 Objetivo

Avaliar o desempenho de um sistema embarcado com arquitetura RISC-V aplicado em um sistema de controle de processos contínuos em relação a um sistema embarcado com arquitetura ARM CORTEX M0+, considerando como aplicação principal o método de controle PID.

1.5 Estrutura do Texto

Optou-se pela organização do texto deste trabalho da seguinte forma: capítulo 2 aborda a fundamentação teórica e trabalhos relacionados, o capítulo 3 apresenta a metodologia utilizada neste trabalho. No capítulo 4 é apresentado o desenvolvimento e no capítulo 5 são apresentados os resultados. O capítulo 6 apresenta a conclusão e no capítulo 7 são disponibilizados os apêndices.

2 . REVISÃO BIBLIOGRÁFICA

Este capítulo está dividido em duas partes, sendo a primeira a revisão bibliográfica das tecnologias utilizadas no desenvolvimento do projeto e a segunda uma pesquisa sobre trabalhos relacionados por meio de uma revisão sistemática da literatura.

2.1 Fundamentação Teórica

Nesta seção serão abordadas as tecnologias utilizadas para o desenvolvimento do presente trabalho.

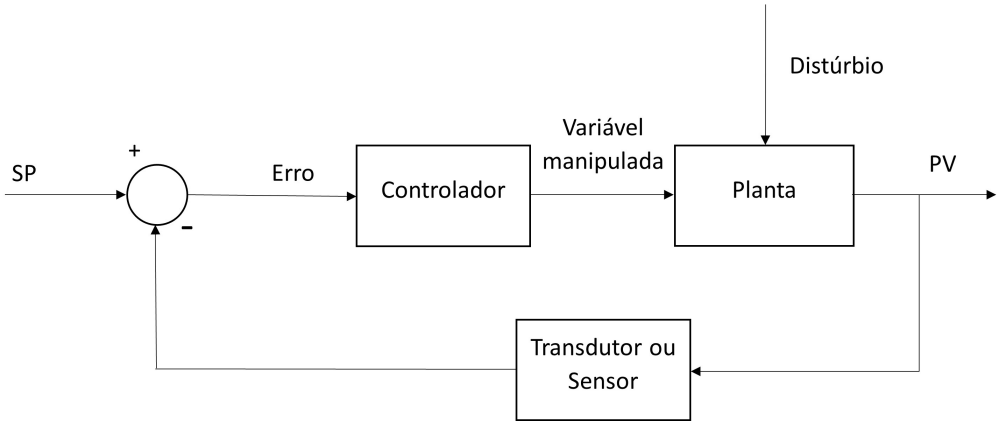
2.1.1 Sistema de Controle

No âmbito da automação industrial um processo contínuo é caracterizado por mudanças lineares que objetivam alcançar um valor pré-definido conhecido como SP (setpoint). Essas mudanças são ações causadas por componentes que atuam em conjunto formando assim um sistema que por vezes é denominado como planta. Em um sistema de controle existem duas variáveis envolvidas: a grandeza medida que é definida como variável controlada ou variável do processo PV (process value) e um sinal que afeta o valor da variável controlada conhecido como sinal de controle ou variável manipulada. Outro termo importante em sistemas de controle é o sinal definido como distúrbio, esse sinal pode ser interno caso afete o sinal de controle ou externos se gerado fora do sistema, nesse caso, são considerados sinais de entrada. Os sistemas de controle podem ser classificados como de malha aberta ou fechada. No caso de controle de malha aberta não há uma resposta acerca da ação do sistema de controle, por outro lado, em sistemas de malha fechada existe uma resposta sobre a ação do sistema de controle, possibilitando um ajuste mais efetivo dos componentes afim de se obter melhor estabilidade entorno do SP, a diferença entre o SP e PV é chamada de erro (OGATA, 2010).

Os sistemas de controle são encontrados na indústria em diversas aplicações, como por exemplo, atuando no controle de nível do líquido de um tanque de acordo com o valor de SP definido. Se o consumo do líquido aumentar, causando um distúrbio, o sistema de controle deverá ser capaz de corrigir o valor de PV de acordo com SP. A Figura 3 apresenta o diagrama de sinal de um sistema de controle em malha fechada. Outro exemplo de uso em nosso dia-a-dia são os elevadores, o transporte rápido ao nosso destino e a parada no andar correto é devido a atuação de um sistema de controle. No cenário do elevador o botão pressionado do andar de destino representa uma entrada definindo um novo SP, o andar atual é o PV, para esse sistema duas variáveis devem ser consideradas, a resposta transitória que nesse caso é a velocidade na qual o elevador deslocará o passageiro, o sistema deve funcionar de forma que não cause um desconforto sendo muito rápida ou demasiadamente lenta, a outra variável é o erro em regime permanente que é uma

especificação de desempenho importante, pois caso o elevador tenha um erro significativo a segurança e comodidade do passageiro é comprometida (NISE, 2023).

Figura 3 – Diagrama de sinal de um sistema de controle em malha fechada.



Fonte: OGATA.

2.1.2 Controle PID

Existem diversos métodos utilizados em sistemas de controle, sendo que em processos industriais um método popularmente difundido é o PID (Proporcional Integral Derivativo). Como mostrado na equação 2.1, o controlador PID é composto de três termos. A ação proporcional do controlador é representada pelo termo K_p que é relacionado ao valor do erro entre a saída e SP. Quanto maior o valor de K_p mais rápida será a resposta do sistema, porém um valor muito alto provocará oscilações e instabilidade. K_i é a ação integral do controlador que representa o erro ao longo do tempo. O termo integral propõe eliminar o erro em regime permanente. Quanto maior o valor de K_i menor o erro em regime permanente, porém isso poderá levar a uma resposta mais lenta e possibilidade de oscilações. O terceiro termo K_d é a ação derivativa do controlador e representa a taxa de variação do erro. A função da ação derivativa previne picos de *overshoot*, o que implica em uma resposta mais suave do sistema. Porém, se o valor de K_d for muito alto o sistema ficará sensível a ruídos.

$$u(t) = K_p e(t) + K_i \int e(t)d(t) + K_d \frac{de(t)}{dt} \tag{2.1}$$

Para representar o controlador PID por meio de uma função de transferência, aplica-se a Transformada de Laplace à equação diferencial que descreve a ação de controle. A função de transferência resultante é mostrada na equação (2.2) que representa o comportamento de um controlador PID no domínio da frequência, sendo uma alternativa a representação no domínio do tempo. É uma ferramenta utilizada para a análise de

sistemas lineares invariantes no tempo, especialmente em sistemas de controle em regime permanente.

$$G(s) = K_p + \frac{K_i}{s} + K_d s \tag{2.2}$$

Com a função de transferência de um controlador PID, é possível avaliar de forma objetiva como o controlador influenciará a resposta do sistema, permitindo a análise de sua estabilidade e desempenho. Os parâmetros Kp, Ki e Kd representam os ganhos proporcional, integral e derivativo, respectivamente. Eles devem ser ajustados adequadamente para otimizar o desempenho do sistema de controle, proporcionando uma resposta rápida e estável, além de reduzir o erro em regime permanente (DORF; H., 1997)

O processo de definir os parâmetros Kp, Ki e Kd é chamado de sintonia. Existem diversos métodos para calcular os valores dos parâmetros proporcional, integral e derivativo, na prática, dois métodos são bem difundidos, o **método da Curva de Reação** e **o método do Ganho Limite**, ambos desenvolvidos pelos engenheiros John G. Ziegler e Nathaniel B. Nichols, mas comumente chamados de métodos de Ziegler-Nichols.

O método da Curva de Reação foi desenvolvido a partir da resposta ao degrau em malha aberta, na qual Ziegler-Nichols observaram que a curva de reação de diversas plantas, inclusive de ordem elevada, apresentavam a resposta em forma de **S** para uma mudança abrupta do *setpoint*. Os valores de K_p, K_i e K_d são determinados a partir de **L** que é o tempo de atraso da resposta e **T** tempo de subida que o sistema leva para passar de 10% a 90% do valor final. A partir dos valores obtidos para T e S, pode-se obter os valores dos parâmetros K_p, K_i e K_d, conforme mostra a Tabela 1.

Tabela 1 – Tabela de ganhos para o método da curva de reação.

Tipo de Controlador	**Kp**	**Ti**	**Td**
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{T}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

O segundo método, o do Ganho Limite, consiste na técnica de encontrar um valor de ganho proporcional no qual a planta entra em oscilação. O método prevê a realização de um teste de ganho crítico, sendo que o primeiro passo é zerar os ganhos K_i e K_d e incrementar o valor de K_p até encontrar o valor no qual o sistema entra em oscilação. No ponto de oscilação, K_p é denominado ganho crítico K_{cr}, o período da oscilação é chamado P_{cr}, a partir desses valores, poderão ser calculados os demais ganhos, conforme pode ser vistos na Tabela 2.

Tabela 2 – Tabela de ganhos para o método do ganho crítico.

Tipo de Controlador	**Kp**	**Ti**	**Td**
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Mesmo sendo a regra de sintonia de Ziegler-Nichols uma excelente ferramenta, ela não define de imediato os parâmetros para os ganhos proporcionais, integrais e derivativos, ainda se faz necessário ajustes finos de parâmetros para se obter a melhor resposta do sistema de controle (OGATA, 2010).

2.1.3 ARM

A ISA (Instruction Set Architecture) é a interface pela qual o programador interage com o processador, sendo que essas instruções atuam no limiar entre hardware e software, é por meio delas que o programador entende as funcionalidades e limitações do processador. Podemos descrever a ISA com base em cinco características:

- Local onde os operandos são armazenados;
- A relação entre instrução e número de operandos;
- Se o operando da ALU (Arithmetic Logic Unit) está armazenado na CPU (Central Processing Unit) ou na memória;
- Quais operações a ISA realiza;
- Quais os tamanhos e tipos de operandos.

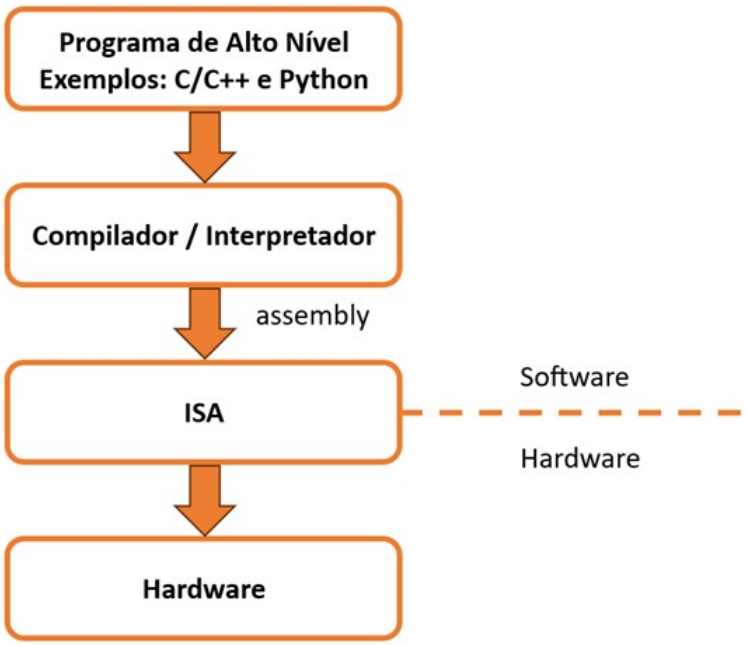
De forma geral, a ISA define qual será o padrão de operação das instruções em nível de bit's, especificando o hardware, uma vez que alterações de hardware acarretam alto custo. Na Figura 4, pode ser visto o nível de atuação da ISA. Dessa forma, o grande desafio no desenvolvimento de uma ISA é projetar um conjunto genérico de instruções que possa satisfazer as mais diversas aplicações.

Em geral um processador é desenvolvido com um entre dois modelos de conjunto de instruções:

- CISC (Complex Instruction Set Computers): criada para facilitar o desenvolvimento de compiladores afim de resolver problema de disparidade semântica (semantic gap);

- RISC (Reduced Instruction Set Computer): impulsionada pela análise do fato de que a maioria das instruções geradas por computadores são simples.

Figura 4 – Camadas da hierarquia dos componentes de um comutador.



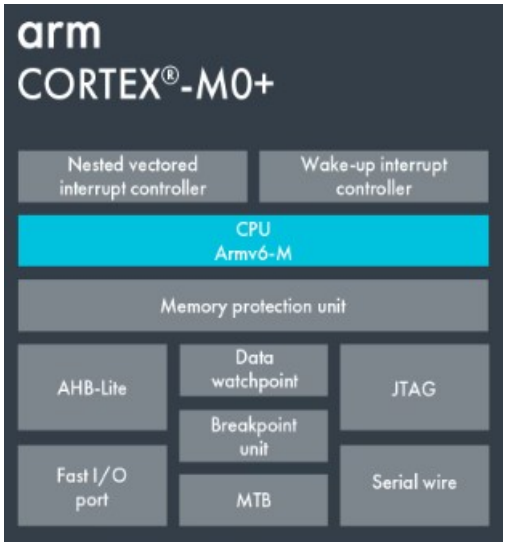
Fonte: Autor.

A ARM Holdings, empresa Britânica, desenvolveu uma família de microcontroladores e microprocessadores nomeados como ARM, com foco no projeto e desenvolvimento. A empresa não produz os processadores mas licencia o direito para que os fabricantes possam produzir seus chips com duas possibilidades: implementar diretamente um chip com base em modelos fornecidos pela ARM ou desenvolver seu processador compatível com a arquitetura ARM. Os processadores ARM utilizam a tecnologia RISC.

Esses chips apresentam como diferencial alta velocidade aliada ao pequeno tamanho e baixo consumo de energia. Essas características colocam os chips ARM na posição de preferidos para o desenvolvimento de sistemas embarcados, de forma que não é exagero dizer que a arquitetura ARM é a mais utilizada no mundo (STALLINGS, 2010)

Destaca-se na família ARM o processador Arm Cortex M0+, que foi desenvolvido a partir do processador Cortex-M0 que mantém compatibilidade de ferramentas e tem o conjunto completo de instruções com maior desempenho. Esse processador entrega um desempenho de 32 bits a um valor equivalente a de um processador de 8 bits, possibilita também uma grande seleção de opções promovendo um desenvolvimento mais flexível, característica desejável no cenário dos sistemas embarcados. A Figura 5 apresenta o diagrama de blocos de um processador Arm Cortex M0+.

Figura 5 – Diagrama de blocos do processador ARM Cortex M0+.



Fonte: developer.arm.com.

A CPU Armv6-M é implementada a partir da ISA Thumb (16 bits) e Thumb-2 (32 bits), os registradores do processador são de 32 bits e o conjunto de instruções Thumb suporta os seguintes tipos de dados:

- Ponteiros de 32 bits;
- Inteiros de 32 bits, com ou sem sinal;
- Inteiros de 8 ou 16 bits, com ou sem sinal;
- Inteiros positivos de 64 bits.

As instruções de armazenamento e transferência de dados permitem trabalhar com bytes (8 bits), words (16 bits) podendo até manipular dados de 64 bits utilizando recursos de manipulação de múltiplas words. Em relação aos registradores, esse núcleo disponibiliza treze registradores de 32 bits de uso geral, R0 a R12, e mais três registradores de uso específicos com as seguintes funcionalidades:

- SP (Stack Pointer): um ponteiro para pilha ativa, eventualmente referido como R13;
- LR (Link Register): registrador que armazena o endereço de retorno de uma subrotina, sendo também atualizado quando ocorre uma exceção, eventualmente chamada de R14;
- PC (Program Counter): carrega o endereço inicial a cada inicialização no processo de reset, chamado também de R15.

Além do funcionamento normal obtido por meio das instruções disponibilizadas na ISA, o processador permite a operação no modo debug (depuração) essencial durante o ciclo de desenvolvimento (ARM, 2018)

2.1.4 RISC-V

A RISC-V (RISC five) é uma ISA não proprietária, ou seja aberta, desenvolvida na Universidade da Califórnia, o projeto inicial foi financiado principalmente por uma combinação de recursos acadêmicos e públicos. O projeto teve início na Universidade da Califórnia, Berkeley, em 2010, apoiado por grupos de pesquisa e por financiamento governamental dos Estados Unidos. Os principais financiadores foram a DARPA (*Defense Advanced Research Projects Agency*), que é a agência de projetos de pesquisa do Departamento de Defesa dos EUA, interessada em promover novas arquiteturas de computação que fossem abertas e seguras, ao contrário das arquiteturas comerciais proprietárias.

A Universidade da Califórnia, Berkeley, ofereceu suporte significativo, disponibilizando infraestrutura e equipe para o desenvolvimento do RISC-V, dado o caráter acadêmico e experimental do projeto. A National Science Foundation (NSF) também concedeu financiamento, ajudando a sustentar as pesquisas em microarquiteturas e novos conceitos na área de arquitetura de computadores. Com o avanço do projeto, outras agências e parceiros industriais começaram a demonstrar interesse na arquitetura, reconhecendo seu potencial como uma alternativa aberta e livre de *royalties* às arquiteturas proprietárias, como ARM e x86. Esse apoio inicial permitiu que a equipe de Berkeley, liderada por pesquisadores como Andrew Waterman, David Patterson e Krste Asanović, desenvolvesse a arquitetura de conjunto de instruções (ISA) RISC-V, consolidando-se como uma inovação no campo da computação.

A ISA está dividida em três conjuntos básicos de instruções:

- Conjunto com 47 instruções denominado RV32I, completa para atender a demanda de sistemas operacionais modernos;
- O conjunto RV32E que tem apenas 15 registradores, porém é bem semelhante ao conjunto anterior;
- E o RV64I também semelhante ao conjunto RV32I, porém com barramento de registradores de dados e PC diferentes (Program Counter).

Projetada com foco na simplificação da implementação em processadores em silício, a ISA RISC-V é considerada uma codificação de instruções altamente regular, com um modelo de memória simples e sem o uso de instruções complexas para acesso a memória de dados (SANTOS et al., 2021).

O projeto RISC-V tem como escopo ser uma ISA padrão, seja para dispositivos de baixo desempenho como microcontroladores ou com alto desempenho como os supercomputadores. Com foco nesse objetivo foi elaborada considerando os seguintes requisitos:

- Atender desde os processadores embarcados até processadores de alto desempenho;
- Ter bom funcionamento independente do software ou linguagem de programação;
- Ser implementado por FPGAs (*Field-Programmable Gate Arrays*), ASICs (*Application-Specific Integrated Circuits*) visando até futuras tecnologias;
- Atuar como base para aceleradores customizados;
- Possuir estabilidade e não entrar em obsolescência.

Existem alguns diferenciais da RISC-V em relação a outras ISA's, destacam-se o fato de ser uma ISA recente criada tendo como bagagem os erros e acertos de outras ISA's e ser aberta não ficando atrelada a nenhuma empresa o que a torna uma ISA livre (PATTERSON; WATERMAN, 2017).

Observa-se um número significativo de empresas conhecidas que estão adotando a ISA RISC-V, algumas delas podem ser vistas na Figura 6, destacam-se entre estas empresa a Nvidia que usa RISC-V em projetos de carros autônomos, Alibaba com implementações de inteligência artificial.

Figura 6 – Empresas filiadas ao projeto RISC-V.

Fonte: riscv.org.

Grandes entidades governamentais como India, Estados Unidos, China e União Européia apoiam o desenvolvimento da RISC-V e promovem programas de treinamentos

em informática engenharia eletrônica em universidades (FROLOV; GALAKTIONOV; SANZHAROV, 2021).

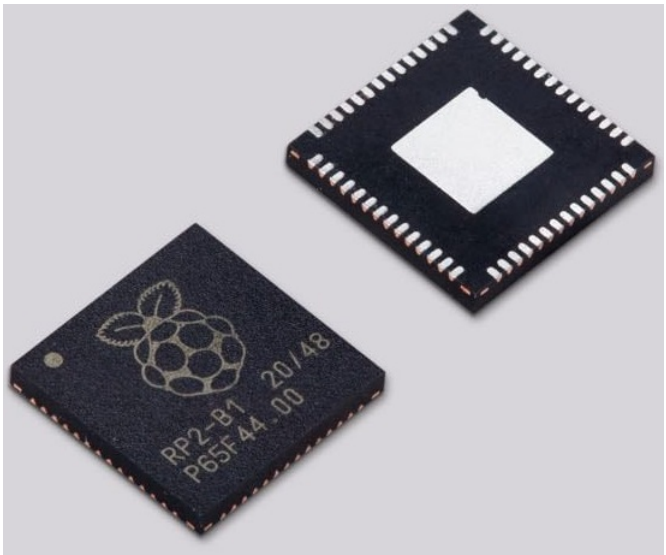
Em fevereiro de 2024, o Brasil ingressou como membro premium na RISC-V internacional. Como membro da diretoria da entidade, o Brasil passa a ter influência direta no desenvolvimento dessa tecnologia, beneficiando o ecossistema nacional de empresas e pesquisadores.

Outro diferencial da RISC-V em relação as outras ISA's existentes é ter um núcleo básico, o RV32I que pode executar uma pilha de software completa. Tem-se como definição que o RV32I nunca será alterado visando oferecer aos desenvolvedores de compiladores e programadores assembly uma estrutura estável e imutável. Existe possibilidade de implementações de extensões que podem ser incluídas no hardware possibilitando a inclusão de recursos de baixo consumo de energia, críticos no cenário dos microprocessadores embarcados, faz-se então a parametrização do compilador RISC-V afim de se gerar o melhor código para as extensões incluídas (PATTERSON; WATERMAN, 2017).

2.1.5 RP2040

A fundação Raspberry lançou em 2021 o microcontrolador RP2040, exibido na Figura 7, com a proposta de trazer ao mercado dos microcontroladores uma opção com alto desempenho, baixo custo e facilidade de uso, o chip foi desenvolvido com arquitetura ARM Cortex M0+.

Figura 7 – Microcontrolador RP2040.



Fonte: raspberrypi.com.

O chip é fabricado com tecnologia de 40 nm e oferece alta performance, modo de

baixo consumo de energia o que o torna uma excelente opção para o desenvolvimento de sistemas embarcados. Dentre as principais características estão:

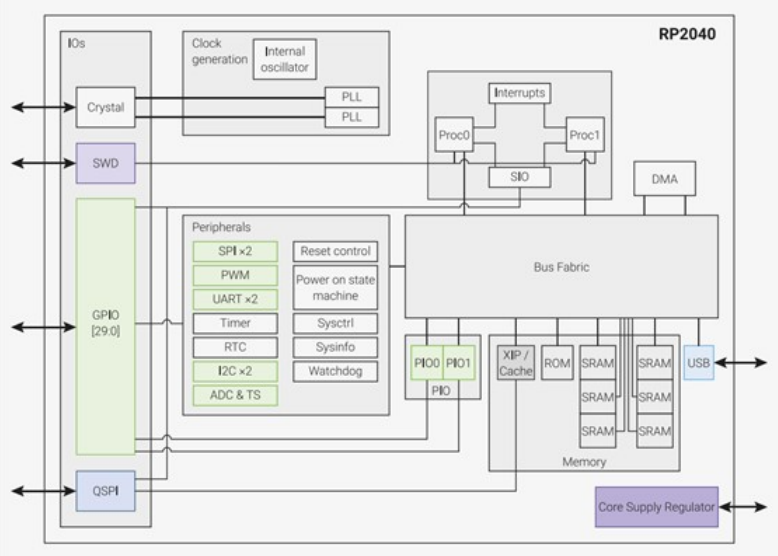
- Dual ARM Cortex-M0+ @ 133MHz;
- SRAM on-chip com 264 kB dividida em seis bancos independentes;
- Suporte para até 16 MB de flash externa com acesso via barramento QSPI;
- Controlador DMA (Direct Memory Access);
- Barramento AHB (Advanced High-performance Bus);
- Regulador de tensão LDO (low-dropout) interno ao chip;
- PLL (Phase Locked Loop) para gerar frequências de clock e USB;
- 30 GPIO (General Purpose Input/Output) podendo quatro ser usadas como canais de entrada analógica.

Em modo deep sleep com baixo consumo de energia, o chip consome aproximadamente 180 μA uma característica excelente para dispositivos alimentados por baterias. Visando a compatibilidade com sensores e atuadores disponíveis no mercado dos microcontroladores, o RP2040 disponibiliza os seguintes periféricos:

- 2 UARTs (Universal Asynchronous Receiver/Transmitter) para comunicação serial
- RS232;
- 2 controladores SPI (Serial Peripheral Interface);
- 2 controladores I2C (Inter-Integrated Circuit);
- 16 canais PWM (Pulse Width Modulation);
- Controlador USB 1.1;
- 8 PIO máquinas de estados.

Um diagrama de blocos com a visão geral do sistema do chip RP2040 pode ser visto na Figura 8. Dentre todas as características já apresentadas, o chip ainda possui um sensor de temperatura interno que se mostra como uma opção ideal para o monitoramento de funcionamento do processador (RASPBERRY PI, 2023).

Figura 8 – Diagrama de blocos do microcontrolador RP2040.



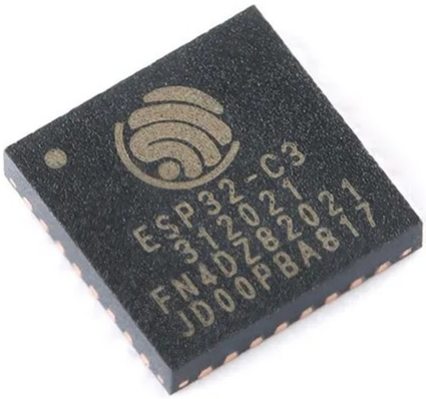
Fonte: raspberrypi.com.

2.1.6 ESP32-C3

Desenvolvido pela empresa ESPRESSIF, o ESP32-C3 é um microcontrolador SoC (System-on-a-Chip) com periféricos Wi-Fi e Bluetooth 5 (BLE) com núcleo baseado na arquitetura RISC-V.

Essa solução promete como entrega um equilíbrio entre potência, recursos de entrada e saída de dados e segurança. É uma excelente opção para projetos que necessitem de conectividade principalmente no cenário da IoT. O grande diferencial que posicionou o chip ESP32, que pode sr visto na Figura 9, na vanguarda do desenvolvimento de sistemas embarcados, foi a conectividade propiciada pelos periféricos Wi-Fi e BLE.

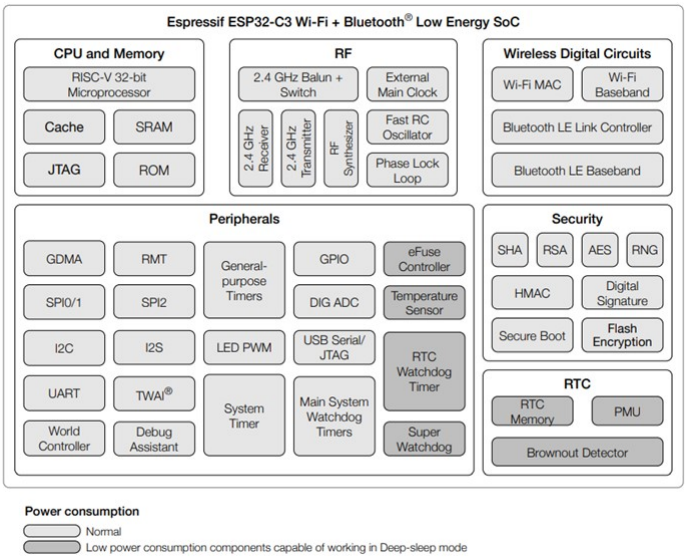
Figura 9 – Microcontrolador ESP32C3



Fonte: espressif.com.

O microcontrolador ESP32C3 é uma solução baseada em um microcontrolador com suporte a Wi-Fi de 2,4 GHz e Bluetooth com baixo consumo de energia BLE. O diagrama de blocos funcionais do SoC é mostrado na Figura 10, onde é possível verificar duas regiões destacadas, uma em cinza claro, que corresponde aos periféricos ativos em modo de funcionamento normal e os blocos destacados na cor cinza escuro, que correspondem aos módulos ativos em função deep sleep, ou seja, em baixo consumo de energia. Esses periféricos são necessários para o funcionamento do chip e possuem funcionalidade de gerar interrupções que provocam o wake up fazendo o chip retornar a função normal ativando todos os periféricos.

Figura 10 – Diagrama de blocos do microcontrolador ESP32C3



Fonte: espressif.com.

Considerando a alta densidade de dispositivos integrados no chip ESP32-C3 é interessante destacar suas características dividindo em funcionalidade mais gerais como módulo Wi-Fi, Bluetooth, CPU e Memória e Periféricos de Interface.

- Wi-Fi
  - IEEE 802.11 b/g/n-compliant
  - Suporta largura de bandas de 20 MHz e 40 MHz dentro da banda 2.4 GHz
  - Modo 1T1R com frequência acima de 150 Mbps
  - Wi-Fi Multimedia (WMM)
  - Monitoramento automático Beacon (hardware TSF)
  - 4 interfaces vituais Wi-Fi

- Bluetooth
  - Bluetooth LE: Bluetooth 5, Bluetooth mesh
  - Modo de alta potência (20 dBm)
  - Velocidade: 125 Kbps, 500 Kbps, 1 Mbps, 2 Mbps
  - mecanismo interno para coexistência de Wi-Fi e Bluetooth compartilhando mesma antena
- CPU e Memória
  - Processador 32-bit RISC-V single-core @ 160 MHz
  - 384 KB ROM
  - 400 KB SRAM (16 KB para cache)
  - 8 KB SRAM no RTC
- Periféricos de Interface
  - 22 GPIO
  - 3 SPI
  - 2 UART
  - 1 I2C
  - 1 I2S
  - 2 conversores ADC com 12 bits ultiplexado em 6 canais
  - Sensor de temperatura integrado
  - 2 Timers com 54 bits para uso geral

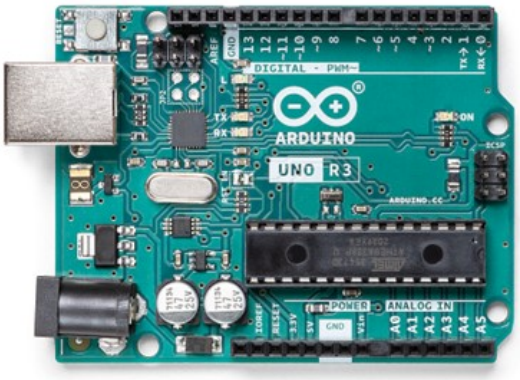
Em relação a segurança o chip disponibiliza o periférico nomeado como *Secure boot* que prevê o controle de acesso à memórias internas e externas. Também está disponível a criptografia/descriptografia de memória Flash possuindo a certificação AES-128/256 (FIPS PUB 197).

Com todas as características apresentadas esse microcontrolador é recomendado para aplicações diversas tais como: automação industrial, casas inteligentes, assistência médica, eletrônicos de consumo, agricultura inteligente, *hubs* de sensores IoT genéricos de baixo consumo de energia, registradores de dados IoT genéricos de baixo consumo de energia, dentre outros (ESPRESSIF, 2023b).

2.1.7 Sistemas Embarcados

Os sistemas embarcados estão presentes em diversos equipamentos do nosso dia-a-dia. São dispositivos eletrônicos com aplicações específicas compondo um sistema de controle para objetos, tais como: forno de micro-ondas, drones, robôs, carros autônomos, dentre outros. Estes dispositivos estão impulsionando o desenvolvimento tecnológico e criando diversas oportunidades de negócios. Nos últimos, anos houve uma grande disseminação de conhecimentos acerca de sistemas embarcados, graças a plataforma de desenvolvimento Arduino Figura 11. Esse projeto com origem na Itália, mas hoje apoiado por diversos colaboradores ao redor do mundo, está divulgado nas mais diversas mídias o que propiciou uma rápida curva de aprendizado dessa tecnologia.

Figura 11 – Plataforma Arduino UNO



Fonte: store.arduino.cc

O ecossistema do Arduino é composto pelas plataformas de desenvolvimento, várias bibliotecas e um Ambiente Integrado de Desenvolvimento (IDE - Sigla em inglês) que facilita as implementações de soluções. A linguagem de programação utilizada na IDE Arduino é C/C++, uma linguagem orientada a objeto com alto nível de abstração (BAPTISTA, 2023).

A popularidade do Arduino ocasionou a fabricação de diversos módulos chamados de *shields* que começaram a ser produzidos em larga escala. Isso facilitou a integração de diversos sistemas embarcados, possibilitando um maior acesso a dispositivos sensores e atuadores, nesse contexto surgiu a base para o desenvolvimento das aplicações de IoT sendo destacável principalmente os módulos WiFi, Ethernet e Bluetooth.

Em geral um sistema embarcado é composto por um microcontrolador e componentes auxiliares que são acondicionados em uma mesma placa, a qual é denominada de plataforma. Em caso que a eficiência do microcontrolador não é suficiente para atender as demandas do projeto, lança-se mão de soluções mais robustas que embarcam processadores

da família x86 e ARM por exemplo. Destaca-se nesse nicho a plataforma Raspberry Pi Figura 12, que é um projeto *open hardware* e na versão 3 utiliza um processador ARM de 1,2 GHz com barramento de dados de 64 bits. Além disso inclui recursos de conectividade como WiFi e Bluetooth, conta também com portas USB e saída HDMI.

Figura 12 – Plataforma Raspberry Pi Modelo 3B+

Fonte: raspberrypi.com

Outro diferencial da plataforma Raspberry Pi é o seu sistema operacional. A plataforma foi projetada para rodar o sistema GNU/Linux, um sistema com código fonte aberto que possibilitou a alteração rápida do Linux para ser utilizado na Raspberry Pi. O armazenamento do sistema operacional e dos arquivos na Raspberry Pi é feito em um cartão SD preparado para esse fim.

Nos últimos anos o microcontrolador ESP8266 e ESP32 da fabricante chinesa ESPRESSIF ganhou grande notoriedade no mercado dos sistema embarcados oferecendo soluções que trazem alto desempenho e baixo custo. Destaca-se dentre os módulos produzidos pela empresa o ESP8266, Figura 13, que possui versões com preços a partir de 1 dólar e entregam um microcontrolador de 32 bits com frequência padrão de 80 a 160 MHz e conectividade WiFi.

Figura 13 – Plataforma ESP8266 Node MCU

Fonte: eletrogate.com

O fato das plataformas embarcadas ESP8266 e ESP32 possuirem por padrão recursos de conectividade possibilitam o desenvolvimento de soluções IoT de forma rápida e barata. Milhares de desenvolvedores utilizam estes microcontroladores em suas aplicações para as mais diversas áreas (OLIVEIRA, 2017).

2.2 Trabalhos Relacionados

Este tópico faz uma descrição sobre o assunto abordado em 13 artigos recuperados por meio de um mapeamento sistemático, destacando como esses trabalhos estão relacionados com o objetivo desta pesquisa.

A revisão sistemática da literatura teve como norteador as questões de pesquisa que podem ser vista na Tabela 3.

Tabela 3 – Questões de Pesquisa

Ordem	**Questão**
Q1	Quais trabalhos buscam avaliar o desempenho do hardware em sistemas de automação industrial?
Q2	Quais trabalhos buscam avaliar o desempenho das linguagens de programação em sistemas de automação industrial?
Q3	Como a arquitetura aberta RISC-V está sendo implementada nas soluções de hardware e software embarcado?
Q4	Qual plataforma embarcada é preferida em implementações de soluções de IoT?
Q5	Qual esforço empregado na disseminação de tecnologias emergentes em ambientes educacionais e industriais?

Após elaboradas as perguntas efetuou-se pesquisa de trabalhos relacionados tendo como fonte principal o Portal de Periódicos da CAPES. Para refinar a busca priorizou-se trabalhos revisados por pares com menos de 10 anos de publicação no idioma Inglês. Com base nos resultados obtidos foi feita uma pré-seleção com a leitura do título e resumos o que resultou na obtenção de 13 artigos. A Tabela 4 apresenta a lista de estudos importados usando a estratégia de busca descrita anteriormente, na tabela é definido um índice e motivo da relevância.

Após leitura dos artigos destaca-se que seis deles tem relevância alta com o projeto. No artigo intitulado como *A Comparative Survey of Open-Source Application-Class RISC-V Processor Implementations* de Dörflinger et al. (2021) é possível evidenciar uma análise de desempenho de processadores com implementações desenvolvidas a partir da ISA RISC-V. Há uma discussão sobre a necessidade de dados que ajudem a definir a melhor implementação. Para tal, os autores fazem um *benchmark* entre quatro implementações Rocket, BOOM, CV A6 e SHAKTI analisando aspectos como performance do processador e consumo de energia.

Tabela 4 – Revisão Sistemática da Literatura

Título	**DOI**	**Relevância**	**Motivo da Relevância**
Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller	(1)	MÉDIA	Trabalho focado na análise de desempenho de linguagens de programação em sistema embarcados, a única diferença para este trabalho é que o autor avalia o desempenho da linguagem de programação e não do hardware. O trabalho se torna relevante, pois demonstra parâmetros de avaliação que podem ser considerados neste trabalho.
A Comparative Survey of Open-Source Application-Class RISC-V Processor Implementations	(2)	ALTA	Servirá de referência para este trabalho, está bem alinhado às expectativas de avaliação da ISA RISC-V em um cenário específico, realiza um *benchmark* com plataformas desenvolvidas na mesma concepção. São avaliados diversos pontos críticos considerando a implementação de sistemas embarcados.
A First Look at RISC-V Virtualization From an Embedded Systems Perspective	(3)	MÉDIA	Relevante por trabalhar com virtualização de um sistema embarcado baseado na ISA RISC-V, essa virtualização é bem vista flexibilizando o uso de um FPGA (*Field Gate Programmable Array*) como hardware base para um microcontrolador.
Computer Engineering Education Experiences with RISC-V Architectures—From Computer Architecture to Microcontrollers	(4)	MÉDIA	Apesar de não ter foco em aplicação industrial este trabalho foca no estudo da arquitetura RISC-V objetivando sua utilização em sistemas embarcados e destacas algumas perspectivas e desafios.
Investigation of RISC-V	(5)	MÉDIA	Um estudo sobre o quão importante é uma ISA na geração de um processador, são feitas discussões sobre o motivo pelo qual a RISC-V destaca-se de forma positiva com opção para futuros desenvolvedores.
Reliability analysis of a fault-tolerant RISC-V system-on-chip	(6)	ALTA	O trabalho apresenta ensaios sobre um processador com arquitetura RISC-V testando a sua tolerância a falhas em espaçonaves, ambiente esses mais hostil e crítico que a indústria.
RISC-V based virtual prototype: An extensible and configurable platform for the system-level	(7)	ALTA	Um trabalho que visa oferecer suporte ao desenvolvimento de soluções IoT acompanhando a alta popularidade da ISA RISC-V, está bem alinhado com os objetivos desse trabalho.
RISC-V Instruction Set Architecture Extensions: A Survey	(8)	ALTA	Outro trabalho que acompanha a popularidade a ISA RISC-V com foco no desenvolvimento de sistemas embarcados. Faz uma análise sobre aplicações específicas e oportunidades de desenvolvimento.
Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things	(9)	MÉDIA	Analisa uma das plataformas mais comuns no desenvolvimento de sistemas embarcados atualmente, são destacados pontos importantes que serão úteis no desenvolvimento do protótipo proposto.
Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol	(10)	ALTA	Apesar de não focar em análise de desempenho explora a utilização de uma plataforma mais acessível para implementações industrias indo de encontro ao objetivo do trabalho.
Leveraging Graphical User Interface Automation for Generic Robot Programming	(11)	MÉDIA	A correlação do artigo com a presente pesquisa está na busca por ferramentas que acelerem o processo de desenvolvimento de soluções de automação sem a necessidade de expertise em uma tecnologia específica. Considerando que isso é um diferencial para a boa aceitação de sistemas embarcados em soluções de automação industrial.
Low-Code as Enabler of Digital Transformation in Manufacturing Industry	(12)	MÉDIA	Este estudo contribui positivamente nas definições desta pesquisa considerando que tais ferramentas serão utilizadas no desenvolvimento de soluções embarcadas para a automação industrial. O uso dessas ferramentas almeja promover uma maior democratização do processo de digitalização industrial.
BIPES: Block Based Integrated Platform for Embedded Systems	(13)	ALTA	Não há uma referência direta ao âmbito industrial, mas o destaque em situações recorrente durante o ciclo de desenvolvimento de um software, seja em ambiente industrial ou não, destacado no estudo, apontam para soluções que sejam implementadas sem grandes complexidades de instalação, que possam ser acessadas via internet ou que possam rodar stand-alone, podendo isso ser adotado como estratégia durante esta pesquisa.

(1) https://doi.org/10.3390/electronics12010143
(2) https://doi.org/10.1145/3457388.3458657
(3) https://doi.org/10.48550/arXiv.2103.14951
(4) https://doi.org/10.3390/jlpea12030045
(5) https://doi.org/10.1134/S0361768821070045
(6) https://doi.org/10.1016/j.microrel.2021.114346
(7) https://doi.org/10.1016/j.sysarc.2020.101756
(8) https://doi.org/10.1109/ACCESS.2023.3246491
(9) https://doi.org/10.1109/ITECHA.2017.8101926
(10) https://doi.org/10.3934/ElectrEng.2020.1.57
(11) https://doi.org/10.3390/robotics10010003
(12) https://doi.org/10.3390/app10010012
(13) https://doi.org/10.1109/ACCESS.2020.3035083

Apesar da relevância alta do artigo *A Comparative Survey of Open-Source Application-Class RISC-V Processor Implementations* de Dörflinger et al. (2021) com este trabalho, há uma grande diferença nos objetivos. O artigo estudado fez a comparação entre plataformas baseadas no núcleo RISC-V, já este trabalho, utiliza dois núcleos distintos, ou seja, busca-se comparar o desempenho do núcleo RISC-V frente ao núcleo ARM Cortex M0+.

Outro artigo com alta relevância é o artigo *Reliability analysis of a fault-tolerant RISC-V system-on-chip* de Santos et al. (2021), nesse artigo os autores propõem o desenvolvimento do processador RISC-V munido de recursos que mitigam falhas, sendo que o objetivo é a implementação do processador para o desenvolvimento de um SoC (system-on-chip), o maior interesse se dá nos métodos utilizados para validação da solução proposta.

O artigo *RISC-V based virtual prototype: An extensible and configurable platform for the system-level* de Herdt et al. (2020) também foi considerado com relevância alta. Nesse artigo os autores relatam o desenvolvimento de um protótipo virtual baseado em RISC-V que atenda a demanda no desenvolvimento de soluções IoT, nesse contexto se torna uma ferramenta que possa contribuir com o desenvolvimento deste projeto.

No artigo *RISC-V Instruction Set Architecture Extensions: A Survey* de Cui, Li e Wei (2023) a relevância alta se dá por se tratar de um estudo profundo sobre processadores implementados a partir da ISA RISC-V, sendo um estudo recente, contribui positivamente com este trabalho no quesito de busca pelo estado da arte dessa tecnologia.

O artigo *Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol* de Aghenta e Iqbal (2020) relata o desenvolvimento de uma aplicação industrial de baixo custo utilizando a plataforma ESP32 em conjunto com tecnologias em nuvem e protocolo MQTT também *open soucers* está diretamente relacionado com esse projeto considerando que o microcontrolador

RISC-V utilizado será implementado por meio da plataforma ESP32 e que o baixo custo em aplicações industrias são almejados com a implementação proposta, porem diferencia-se pelo núcleo utilizado que neste trabalho será validado o RISC-V.

Finalizando o grupo de trabalhos classificados com alta relevância temos o artigo *BIPES: Block Based Integrated Platform for Embedded Systems* de Junior et al. (2020) que apresenta uma plataforma de desenvolvimento de aplicações para sistema embarcados baseada em programação por blocos, pretende-se ao final desse trabalho a implementação de bloco na referida plataforma para a comunicação via Modbus RTU.

Em relação aos trabalhos classificados com relevância média temos o artigo *Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller* de Plauska, Liutkevičius e Janavičiūtė (2022) o trabalho faz uma análise de performance entre quatro linguagens de programação com base na plataforma embarcada ESP32, apesar de não focar no hardware, o trabalho é importante neste contexto, pois apresenta como referência o tempo de execução de uma determinada tarefa, que pode ser o parâmetro chave na comparação de desempenho no que diz respeito a linguagem de programação nesse trabalho.

No artigo *A First Look at RISC-V Virtualization From an Embedded Systems Perspective* de Sá, Martins e Pinto (2021) é abordada a virtualização de um sistema embarcado baseado em RISC-V, apresenta o uso de FPGA como proposta de implementação. Após a leitura do trabalho foi possível uma maior apropriação de conhecimentos acerca do RISC-V.

O trabalho *Computer Engineering Education Experiences with RISC-V Architectures—From Computer Architecture to Microcontrollers* de Jamieson et al. (2022) descreve etapas no processo de ensino aprendizagem relativo ao conhecimento sobre arquitetura de computadores com foco em aplicação no desenvolvimento de microcontroladores, o trabalho contribui no aspecto de referencial teórico e apresenta ferramentas que reforçam esse conhecimento.

Pode-se observar no artigo *Investigation of RISC-V* de Frolov, Galaktionov e Sanzharov (2021) um estudo sobre a arquitetura RISC-V frente a outras arquiteturas, inclusive da ARM que é a referencia de comparação deste trabalho, sendo assim, é relevante no aspecto embasamento teórico.

O artigo *Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things* de Maier, Sharp e Vagapov (2017) traz um estudo acerca da plataforma ESP32 sob o aspecto do desenvolvimento de soluções IoT, é uma excelente fonte de conhecimento sobre esta plataforma, cuja uma de suas versões será utilizada neste trabalho.

No trabalho *Leveraging Graphical User Interface Automation for Generic Robot*

Programming de Ionescu (2020) é relatado o desenvolvimento de uma plataforma mais amigável para a programação de robôs de diversos fabricantes. Demonstra-se um esforço na direção da universalização do conhecimento tornando o conhecimento técnico mais acessível e aplicável em chão de fábrica, essa ideia vai de encontro com a proposta deste trabalho.

Por fim, o último artigo que faz parte do grupo classificado com relevância média é o *Low-Code as Enabler of Digital Transformation in Manufacturing Industry* de Sanchis et al. (2019) onde os autores pesquisam sobre a aceitação e perspectivas de ferramentas *Low-Code* em sistemas de automação industrial, considerando a necessidade emergente de transformação digital impulsionada pela Indútria 4.0. Esse estudo torna-se relevante, pois o presente trabalho poderá ao final contribuir com a plataforma BIPES que é uma plataforma *Low-Code*.

3 . MATERIAIS E MÉTODOS

Este capítulo versa sobre os materiais utilizados para a coleta de dados e descreve como os dados coletados são tratados e analisados, objetivando subsidiar evidências que contribuam para o preenchimento da lacuna sobre a análise de desempenho em aplicações específicas da tecnologia emergente abordada neste trabalho.

3.1 Materiais

Considerando que os resultados obtidos com esta pesquisa poderão fomentar a construção de soluções de automação reais de processos contínuos industriais, optou-se em fazer os ensaios utilizando equipamentos presentes no chão de fábrica. Para tal, a coleta de dados é feita em uma planta de processos contínuos, equipada com dois reservatórios de água no qual o líquido é transferido do reservatório inferior para o superior, afim de se manter o nível pré-definido ou SP (sigla em inglês para *setpoint*) para este reservatório.

A transferência do fluído ocorre por meio de um conjunto motobomba, controlado por um inversor de frequência, ambos equipamentos indicados para aplicações industriais. O inversor de frequência foi parametrizado de forma que a frequência de saída seja proporcional a um sinal de entrada de controle, que varia linearmente em um intervalo de 0 a 10 V.

A leitura do nível do reservatório é obtida por meio de um transdutor diferencial de pressão, equipamento este também indicado para aplicações industriais, o valor do nível coletado é convertido em um sinal de corrente com variação de saída em um intervalo de 4 a 20 mA sendo que 4 mA representa a situação em que reservatório encontra-se vazio e 20 mA indica que o reservatório está completamente cheio, a resolução de leitura de sinal é de 16 bits.

O sistema de controle é composto por uma placa eletrônica equipada com uma plataforma de sistema embarcado e circuitos de condicionamento de sinais. O sinal de corrente de entrada que representa o nível do tanque é denominado valor do processo ou PV (sigla em inglês para *process value*) sendo gerado pelo transdutor de pressão instalado na planta de processo contínuos, esse sinal é processado pelo microcontrolador presente na plataforma embarcada, com base em um algoritmo de controle PID.

O microcontrolador gera como saída, um sinal de tensão denominado sinal de controle ou CV (sigla em inglês para *control value*) que é responsável por controlar a frequência do inversor alterando a velocidade de giro do motor que está presente no conjunto motobomba.

A placa de controle é híbrida, ou seja, comporta duas plataformas embarcadas, uma delas desenvolvida com base em um microcontrolador RISC-V e a outra contém um

microcontrolador com núcleo ARM CORTEX M0+.

Para registro dos dados utiliza-se um cartão do tipo micro SD o qual registra o tempo relativo ao *reset* do microcontrolador em milissegundos e os valores do SP, PV, CV, K_p, K_i e K_d. Os valores são armazenados em um arquivo do tipo *comma-separated values* (CSV) esse arquivo é o conjunto de dados utilizado na etapa de análise de dados.

3.2 Métodos

A coleta de dados ocorre por meio de bateladas com durações que variam de três a cinco horas. Durante cada ciclo o SP inicia em 10 % do volume total e a cada cinco minutos é incrementado de forma automática em 10 % até atingir o valor de 70 %, a partir daí o valor é decrementado em 10 % até atingir o valor 10 %, essa variação persiste durante toda a batelada.

A placa de controle utiliza o algoritmo de PID para manter o valor de SP definido, a cada 500 ms a placa de controle salva os valores do tempo relativo, SP, PV, CV, K_p, K_i e K_d no arquivo de dados do cartão micro SD. O arquivo .csv contém dados relativos a um modelo de experimento podendo ser uma das seguintes configurações:

- Placa de controle equipada com sistema embarcado baseado no microcontrolador RISC-V com firmware desenvolvido em linguagem C/C++;
- Placa de controle equipada com sistema embarcado baseado no microcontrolador ARM Cortex M0+ com firmware desenvolvido em linguagem C/C++;
- Placa de controle equipada com sistema embarcado baseado no microcontrolador RISC-V com firmware desenvolvido em linguagem MicroPython;
- Coleta de dados relativos ao desempenho do sistema quando programado na plataforma BIPES que utiliza blocos para gerar códigos em linguagem MicroPython. Essa etapa do experimento é justificada tendo como premissa promover o uso de sistemas de controle de forma mais acessível.

Em consonância com o contexto explanado no capítulo 1, especificamente na seção 1.1, foram realizados testes com a inclusão da execução da pilha TCP/IP e o envio de dados para um serviço em nuvem, buscando avaliar o desempenho do sistema proposto no cenário de implementação de soluções IoT.

O processo de análise de dados para os três arranjos de experimentos seguiram a mesma sequência. Inicialmente os dados são classificados pelo SP em ordem crescente de tempo em que os eventos ocorreram. Para uma visão geral do resultado do experimento, são adotadas ferramentas gráficas com o objetivo de verificar se houve alguma descontinuidade ao longo do período de aquisição de dados.

A próxima etapa consiste na seleção de um intervalo específico para verificação da reposta do sistema de controle conforme o SP definido, essa amostra é então subdividida em duas partes, sendo a primeira para análise da reposta transitória e a segunda para o regime permanente.

A primeira análise tem como objetivo verificar o tempo de atuação do sistema e a segunda a variação de PV em torno de SP. São então calculados o tempo de ação do sistema de controle durante a resposta transitória e os pontos de máximo e mínimo no trecho de regime permanente.

Após a análise individual de cada arranjo experimental, as três amostras são comparadas, sendo a referência a resposta da plataforma embarcada com núcleo ARM Cortex M0+, são então calculados os erro quadrático para as outras duas amostras.

Para análise dos resultados obtidos, foi definido como padrão de desempenho o microcontrolador ARM Cortex M0+, por ser o preferido em aplicações de sistemas embarcados devido a sua performance e baixo consumo de energia.

4 . DESENVOLVIMENTO

Neste capítulo são descritas todas etapas que delinearam o desenvolvimento deste trabalho. O capítulo descreve as etapas durante o desenvolvimento de hardware e *firmware*.

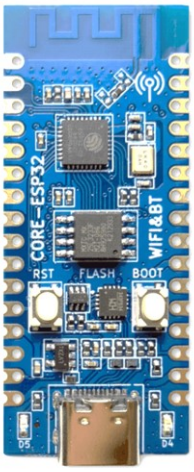
4.1 Hardware

O marco inicial do desenvolvimento do trabalho foi a definição da plataforma a ser avaliada, com base no contexto descrito em tópicos anteriores, as seguintes premissas foram consideradas:

- Microcontrolador com núcleo RISC-V;
- Frequência de clock padrão acima de 100 MHz;
- Interface de comunicação UART;
- Interface de comunicação I2C;
- Interface de comunicação Wi-Fi;
- Baixo consumo de energia;
- Plataforma acessível no mercado nacional.

Foi selecionada a plataforma de desenvolvimento LuatOS Figura 14, que possui um microcontrolador RISC-V ESP32-C3. A escolha se provou conveniente, pois segundo Aghenta e Iqbal (2020) a família de microcontroladores ESP32 são as mais utilizadas no desenvolvimento de dispositivos IoT.

Figura 14 – Plataforma LuatOS



Fonte: wiki.luatos.org

Após definida a plataforma utilizada na avaliação, se fez necessária a busca por outra plataforma com características semelhantes, mas baseada em um microcontrolador com núcleo ARM. Além dos requisitos utilizados para definição da plataforma de teste, foi considerado a possibilidade de buscar uma placa pino compatível com a placa de teste, foi então definida, a plataforma Raspberry Pi Pico Figura 15, que possui um microcontrolador RP2040 ARM Cortex M0+.

Figura 15 – Plataforma Raspberry Pi Pico

Fonte: raspberrypi.com

Como o foco deste trabalho está na validação da arquitetura RISC-V frente a ARM, foi feita uma análise no *datasheet* dos dois microcontroladores selecionados, objetivando uma comparação mais detalhada dos recursos disponibilizados pelos dois componentes. Uma síntese dessa análise é apresentada na Tabela 5 que sumariza as principais características.

Tabela 5 – Comparação entre os microcontroladores RP2040 e ESP32-C3

Característica	**RP 2040**	**ESP32-C3**
Arquitetura	ARM Cortex M0+	RISC-V
Clock máximo	133 MHz	160 MHz
SRAM	246 kB	400 kB
SPI	2	3
I2C	2	1
UART	2	2
Suporte para RAM externa	16 MB	16 MB
Barramento de dados	32 bits	32 bits
GPIO	30	22

Após análise foi considerado que as características mais relevantes dos microcon-

troladores selecionados não se diferenciam significativamente, considerando a aplicação proposta neste trabalho, isso corrobora com a premissa de que foram selecionados dois microcontroladores compatíveis em recursos.

Na sequência foi efetuado um comissionamento na planta de processos contínuos, afim de definir qual sinal seria coletado do sensor, o sinal de controle e a fonte de alimentação.

Para aferição do nível do tanque, são disponibilizados um transdutor ultrassônico, um transdutor do tipo célula de carga e um transdutor de pressão, este último pode ser visto na Figura 16. Todos os transdutores possuem saída do tipo corrente com variação de sinal entre 4 a 20 mA. Para este trabalho foi utilizado o sinal de saída do transdutor de pressão, pois o ponto no qual está instalado o sensor o torna estável mesmo quando o fluído trabalha em um regime turbulento.

Figura 16 – Transdutor de Pressão com saída 4 a 20 mA

Fonte: smar.com

Para controle do conjunto motobomba é utilizado um inversor de frequência, mostrado na Figura 17, parametrizado para controle de velocidade de giro do motor em função do sinal analógico de entrada. O sinal de controle é de 0 a 10 V, sendo 0 V saída 0 Hz e 10 V saída 200 Hz.

Além do sinal de controle, o *START* é efetuado por meio de um pulso em uma entrada digital. O inversor de frequência está instalado em um painel elétrico que comporta também componentes para proteção do circuito e além disso, é disponibilizado uma fonte de 24 V para alimentação de sensores e atuadores. Essa fonte de alimentação foi utilizada para alimentação do circuito de controle desenvolvido neste trabalho.

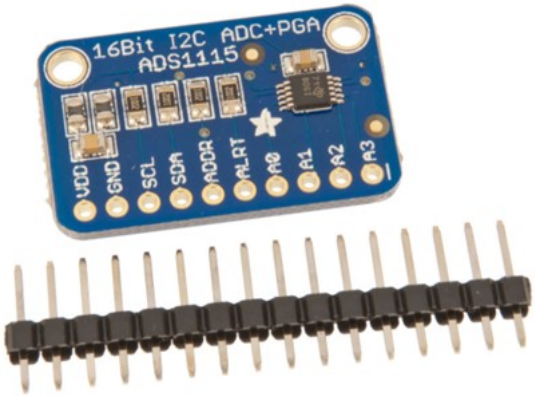
Após o levantamento sobre os sinais de leitura, controle e alimentação disponíveis na planta, foram pesquisados componentes para compor a interface de sinais entre os sensores e atuadores e as plataformas microcontroladas.

Figura 17 – Inversor de Frequência

Fonte: rockwellautomation.com

Considerando o tipo de sinal proveniente do transdutor de pressão, foi utilizado o módulo conversor de sinais analógico para digital, modelo ADS1115, Figura 18. Esse componente faz a leitura do sinal analógico de entrada e converte para um sinal digital com resolução de 16 bits. O valor convertido é então disponibilizado via protocolo de comunicação I2C.

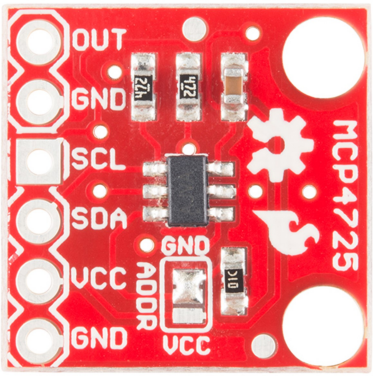
Figura 18 – Conversor AD modelo ADS115 com 16 bits de resolução



Fonte: adafruit.com

Em relação ao sinal de controle foi utilizado um conversor de sinais digitais para analógico modelo MCP4725 mostrado na Figura 19. Esse componente é controlado via protocolo I2C, e gera um sinal em seu canal de saída com resolução de 12 bits.

Figura 19 – Conversor DA MCP4725 com 12 bits de resolução



Fonte: sparkfun.com

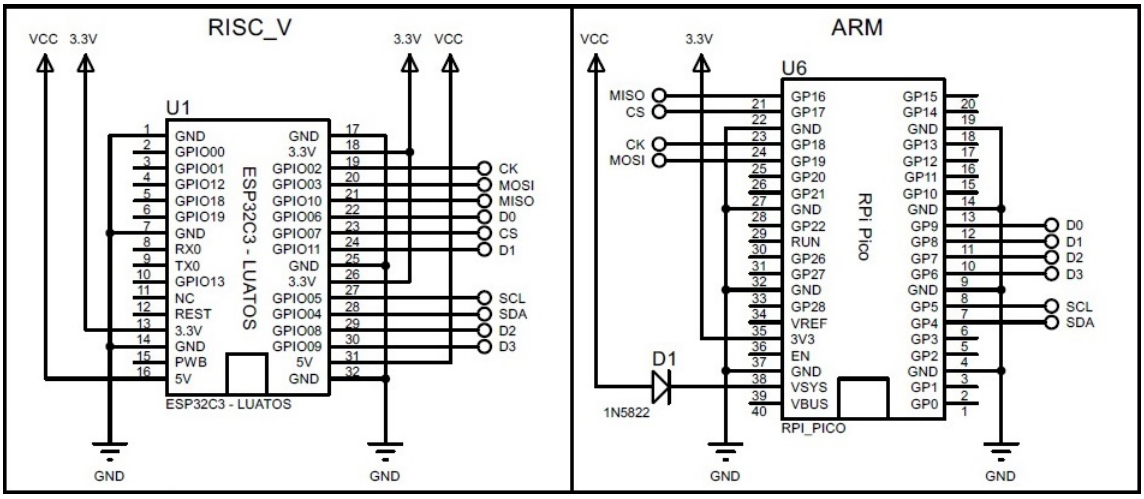
Outro componente chave necessário para o perfeito funcionamento da placa de controle é o módulo conversor de tensão do tipo *step down* LM2596 Figura 20. Esse componente faz a conversão do sinal da fonte de alimentação 24 V para uma saída regulada de 5 V.

Figura 20 – Conversor de Tensão Step Down

Fonte: eletrogate.com

Após definidos os componentes principais da placa de controle, a próxima etapa foi o desenvolvimento do esquema elétrico da placa. Foi elaborado um esquema elétrico híbrido que comporta, de forma não simultânea, as duas plataformas definidas. Na Figura 21 podem ser vistas as conexões das plataformas LuatOS e Raspberry Pi Pico.

Figura 21 – Conexões elétricas das plataformas LuatOS e Raspberry Pi Pico

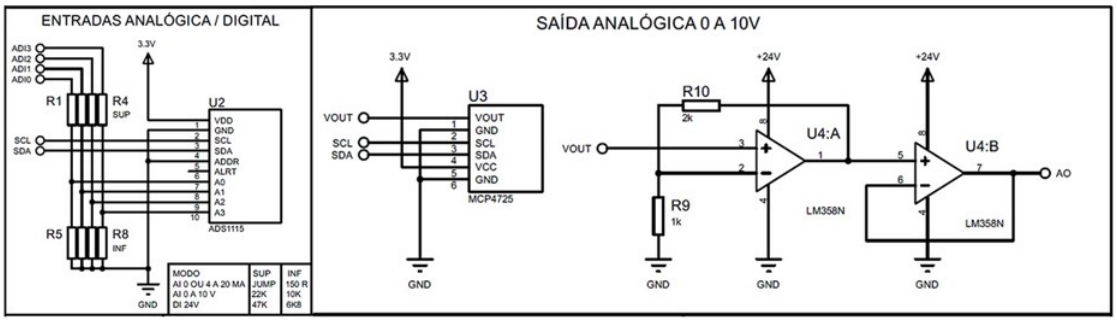


Fonte: Autor

O sinal de corrente enviado pelo transmissor de pressão é convertido para tensão e lido pelo módulo conversor AD e então, o microcontrolador efetua a leitura do sinal convertido via protocolo I2C e processa esse sinal no algoritmo PID.

Após o processamento de sinal o valor de CV é enviado via I2C ao conversor DA que gera o sinal em um *range* de 0 a 3,3 V em um circuito amplificador com ganho não-inversor, multiplicando o sinal de entrada por 3, dessa forma, é gerado um sinal de saída com uma variação aproximada de 0 a 10 V. A interface de sinais de entrada e saída pode ser vista na Figura 22.

Figura 22 – Interface de entrada e saída analógica



Fonte: Autor

A função do conversor *step down* é converter o sinal de alimentação de 24 V para 5 V, sendo possível a alimentação dos sistemas embarcados com esse nível de tensão. Os demais componentes auxiliares como resistores e capacitores são necessários para o correto

funcionamento do circuito e foram calculados com base em informações retiradas dos *datasheets* dos componentes principais.

É possível verificar no esquema elétrico completo, que está disponível no **Apêndice A**, a presença de um *buffer* ULN2803, essa inclusão se fez necessária em função do funcionamento do inversor de frequência.

Caso o sistema pare, deve-se aplicar um pulso na entrada digital do inversor de frequência, e isso é feito por meio do relé acionado por uma das saídas digitais do *buffer*.

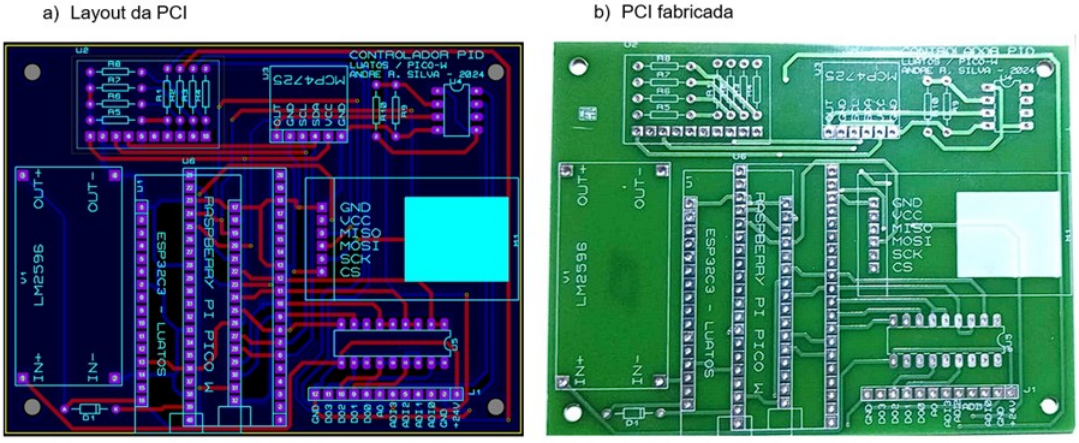
Outro componente de destaque incluído no circuito de controle é o módulo de cartão micro SD. Esse módulo funciona com base no protocolo de comunicação SPI e permite o registro dos dados no arquivo .csv presente no cartão de memória.

A partir do esquema elétrico desenvolvido, foi elaborado o desenho da placa de circuito impresso, na qual foi preconizado a melhor disposição dos componentes e menor tamanho possível da placa, pois, o circuito proposto deveria compor o sistema de processo contínuo sem grandes necessidades de adaptação.

Com a etapa de projeto de circuito eletrônico de controle finalizado, os arquivos de projetos foram enviados para uma empresa especializada para fabricação da placa de circuito impresso (PCI)

Apesar de existirem outras formas de prototipagem, como montagem em placa padrão, por exemplo, optou-se por uma fabricação profissional, visando minimizar possíveis erros provenientes de uma montagem inadequada da placa eletrônica. Na Figura 23 é possível ver o *layout* desenvolvido (item a) e a placa fabricada (item b).

Figura 23 – Placa de circuito impresso desenvolvida



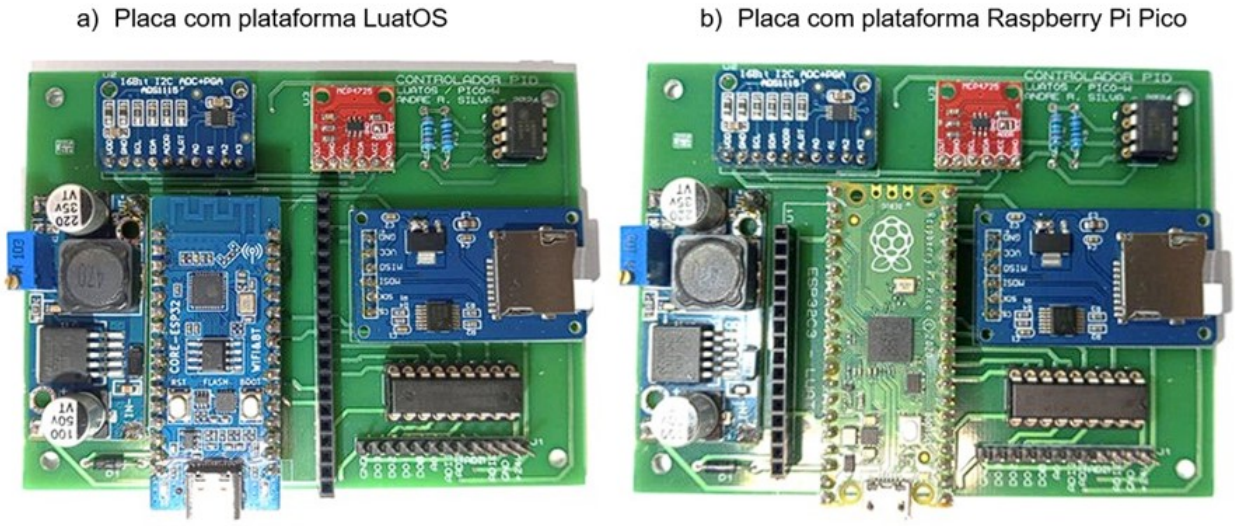
Fonte: Autor

Dando continuidade ao processo, os componentes necessários para montagem da

placa foram adquiridos e seguiu-se então com a montagem da placa de controle. Por se tratar de uma placa híbrida, é possível utilizar a placa montada com a plataforma Raspberry Pi Pico ou a plataforma LuatOS.

Para conexão dessas plataformas na placa de controle utilizou-se barra de pinos prevendo a necessidade de alternância de plataforma conforme a etapa de testes fosse desenvolvida. Na Figura 24 é possível verificar a placa com a plataforma RISC-V (item a) e ARM Cortex M0+ (item b).

Figura 24 – Placa eletrônica com as plataformas LuatOS e Raspberry Pi Pico



Fonte: Autor

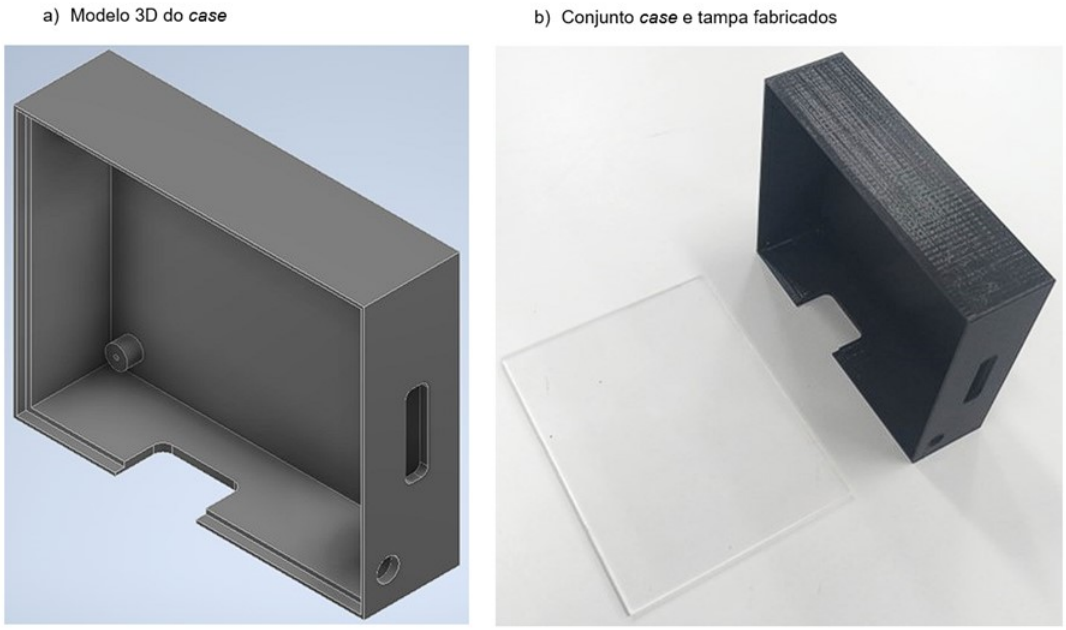
Uma vantagem a se destacar em utilizar a mesma placa durante as etapas de testes, é a eliminação de outras possíveis variáveis que possam influenciar no desempenho do circuito, como o mau funcionamento de um componente acessório, ou seja, a única mudança de circuito nos testes é a plataforma microcontrolada.

De posse da placa eletrônica de controle e considerando o ambiente no qual o circuito seria instalado, foi necessário a modelagem de um *case* dedicado para acomodação do circuito eletrônico. Para esta etapa do trabalho foi utilizado o software Autodesk Inventor Professional 2025, o modelo 3D desenvolvido pode ser visto na Figura 25 item a.

A fabricação do *case* foi feita pelo processo de manufatura aditiva ou também conhecida como impressão 3D. O material utilizado na fabricação do *case* foi o polímero ABS, a tampa foi fabricada em acrílico, utilizando para tal, uma máquina de corte à laser.

O conjunto fabricado pode ser visto na Figura 25 item b.

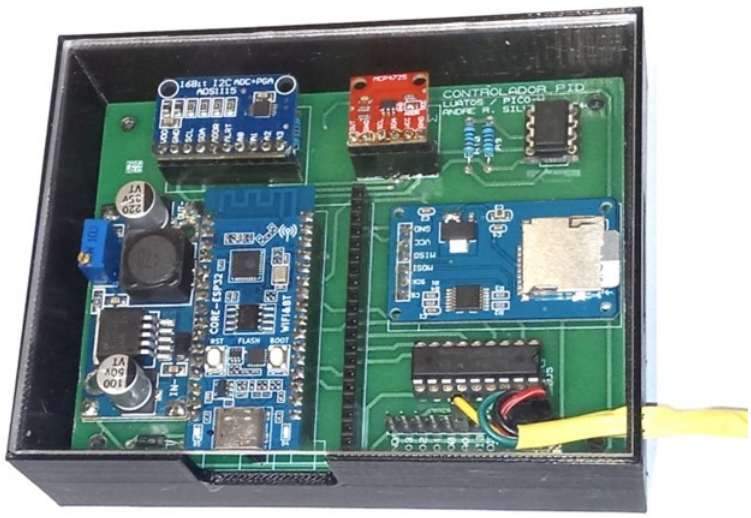
Figura 25 – Conjunto *case* e tampa desenvolvidos



Fonte: Autor

O conjunto completo montado que foi instalado na planta de processos contínuos, pode ser visto na Figura 26. A conexão de sinais é feita via terminais disponíveis na placa, canto inferior esquerdo da figura, e bornes de conexão no painel da planta. O método de instalação se mostrou eficiente de forma que durante a fase de testes a única alteração de hardware foi a alternância de plataforma. Na figura pode ser visto a placa com a plataforma RISC-V.

Figura 26 – Conjunto completo montado



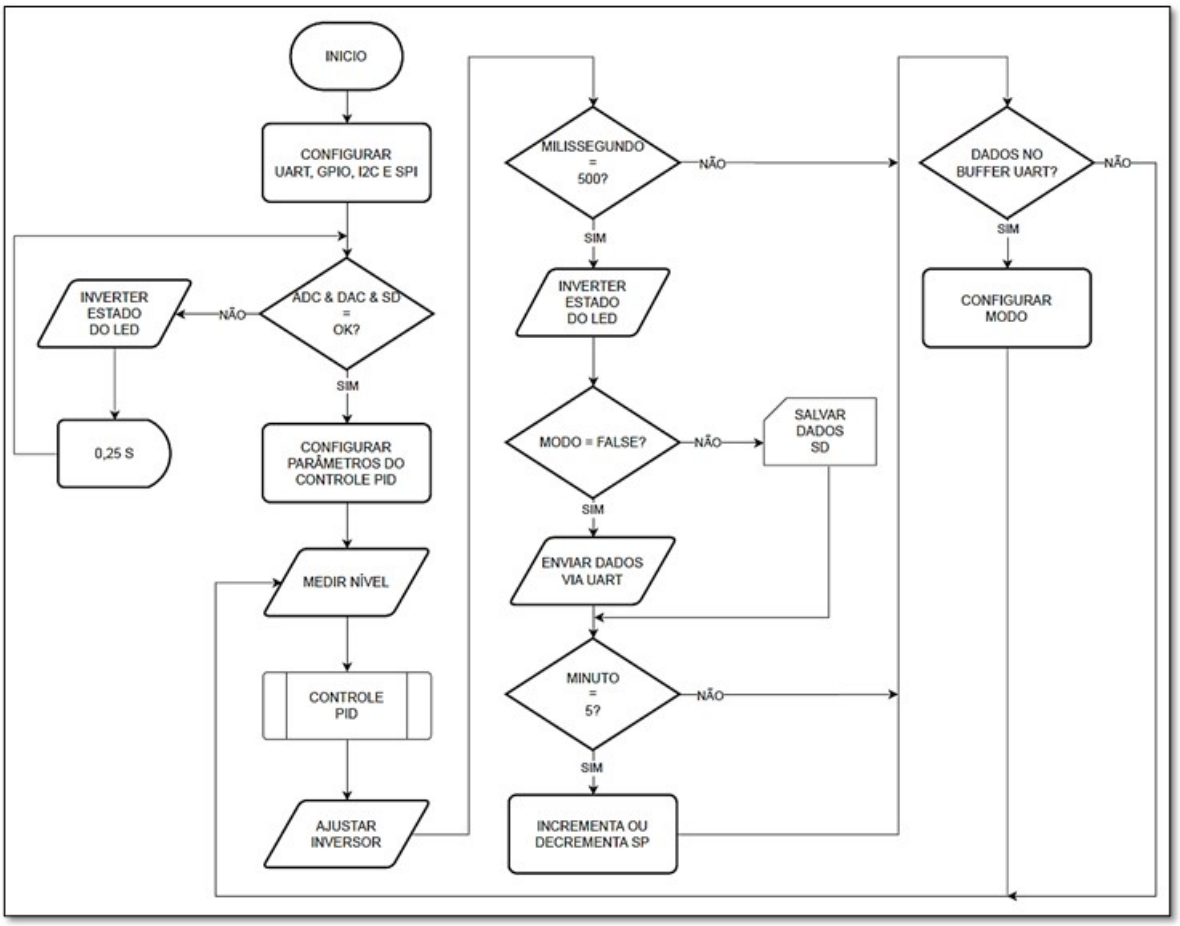
Fonte: Autor

4.2 Firmware

Finalizada a etapa de desenvolvimento de hardware foram elaborados os algoritmos para o funcionamento do sistema. Para simplificar a implementação e visando melhor documentação dessa etapa, foram desenvolvidos dois fluxogramas.

Durante a elaboração do primeiro fluxograma, Figura 27, foram consideradas as especificidades da linguagem de programação C/C++. Esse fluxograma foi base para o desenvolvimento dos *firmwares* utilizados nos experimentos realizados com a Linguagem C/C++ para a plataforma RISC-V e ARM Cortex M0+.

Figura 27 – Fluxograma para desenvolvimento do *firmware* em C/C++



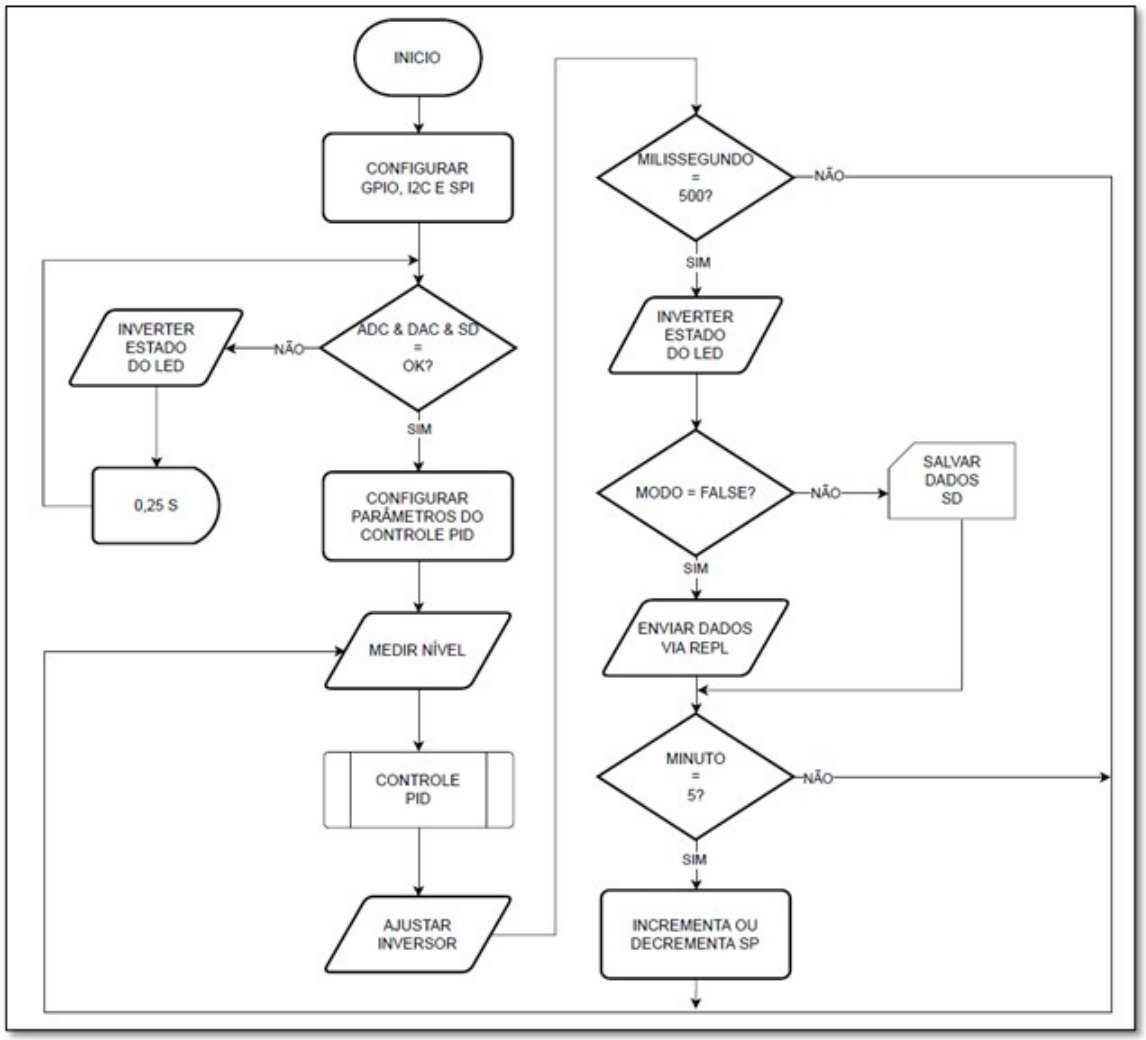
Fonte: Autor

No **Apêndice B** está disponibilizado o código referente ao *loop* principal desenvolvido em linguagem de programação C/C++. No fluxograma é possível ver a chamada da sub-rotina denominada CONTROLE PID, os códigos com a implementação completa dessa sub-rotina estão disponibilizados no **Apêndice C**.

A elaboração do segundo fluxograma, Figura 28, tomou como base o primeiro

fluxograma desenvolvido, com pequenas alterações considerando as especificidades da linguagem de programação MicroPython.

Figura 28 – Fluxograma para desenvolvimento do *firmware* em MicroPython



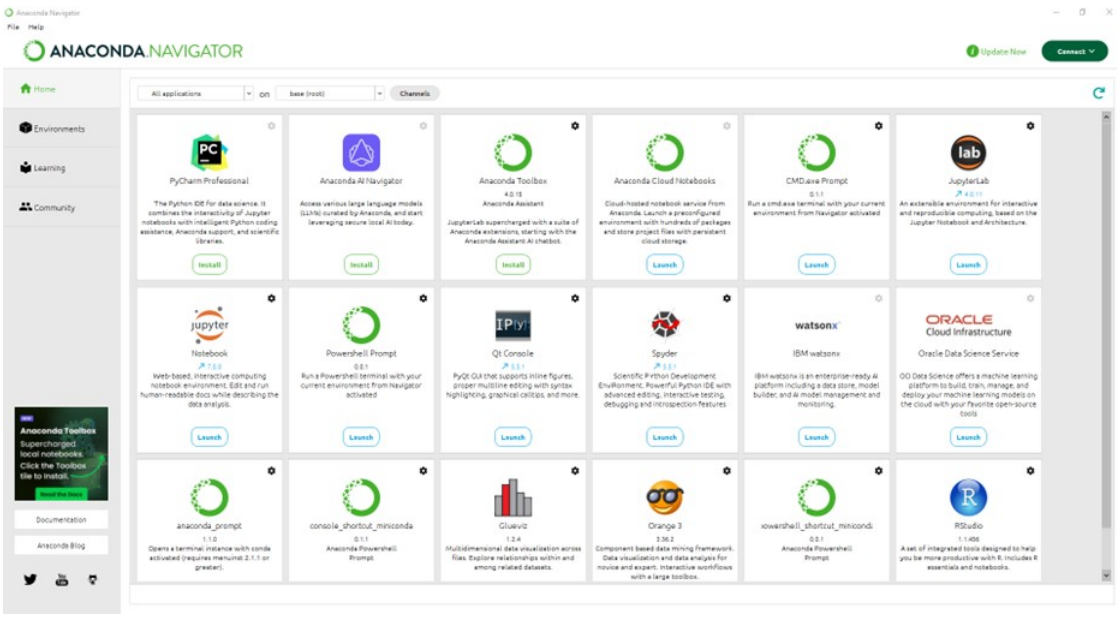
Fonte: Autor

Esse fluxograma foi base para o desenvolvimento do *firmware* em MicroPython para plataforma RISC-V. O código completo do *loop* principal desenvolvido, bem como, da sub-rotina CONTROLE PID estão disponibilizados no **Apêndice D** e **Apêndice E**.

Para viabilizar o registro dos dados, foi utilizado um cartão micro SD e o formato de arquivo CSV, após a coleta de dados, o conjunto de dado gerado ao longo da batelada foi analisado utilizando para tal um notebook com scripts em linguagem de programação Python.

Objetivando a utilização de uma ferramenta completa para análise de dados foi utilizado nesta etapa do projeto a plataforma ANACONDA Figura 29.

Figura 29 – Visão geral de ferramentas disponíveis na plataforma ANACONDA



Fonte: Autor

A escolha da plataforma foi motivada pela gama de recursos disponíveis em uma única ferramenta, dentre as diversas opções, foi utilizado principalmente o aplicativo Jupyter Notebook.

5 . RESULTADOS

Este capítulo apresenta os resultados obtidos durante a fase de coleta de dados na planta de processos contínuos. Os experimentos foram realizados por meio de bateladas na qual foram feitas configurações inicias na placa de controle, após isso, o sistema permaneceu em funcionamento de forma totalmente automatizada.

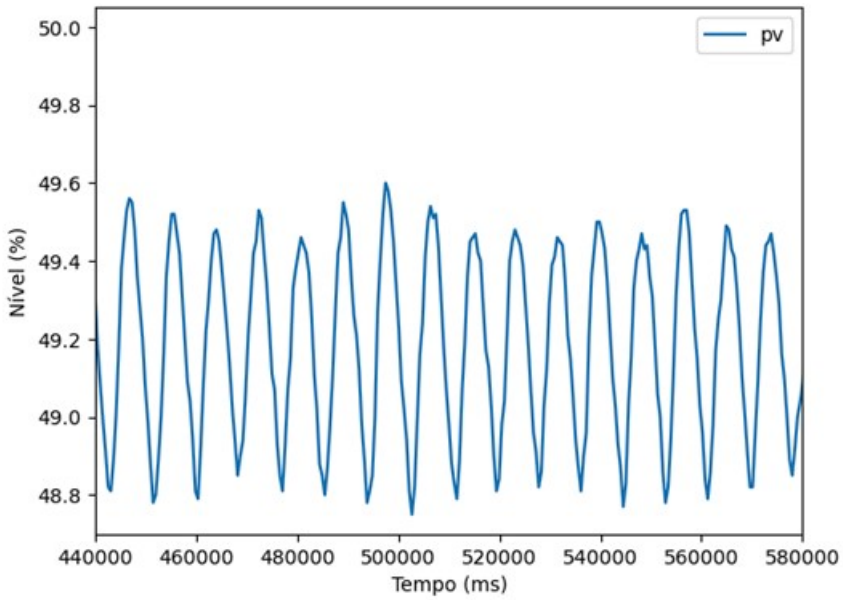
O ajuste do *setpoint* foi programado para ocorrer a cada cinco minutos começando em 10 % e incrementado em 10 unidades até atingir o valor máximo de 70 %, ao atingir esse valor o *setpoint* é decrementado em 10 unidades até chegar novamente em 10 %. O ciclo fica em funcionamento contínuo durante toda a batelada.

5.1 Sintonia

A primeira etapa para instalação de um controle PID é o processo de sintonia, para este trabalho foi escolhido o método Ziegler-Nichols denominado como **Ganho Limite**. Como explicado no capítulo 2, sub-seção 2.1.2, esse método prevê a busca por um ganho crítico K_c para que seja possível o cálculo dos ganhos K_p, K_i e K_d.

Considerando o método selecionado, os ganhos integrais e derivativos foram zerados e o ganho proporcional foi gradativamente incrementado até que o valor de PV entrasse em oscilação. Para a planta utilizada neste trabalho, o valor encontrado de K_c foi 75, o gráfico que representa o comportamento de PV para esse ganho, pode ser visto na Figura 30.

Figura 30 – Gráfico do valor do nível em função do tempo com ganho crítico 75



Fonte: Autor

Com base nos valores obtidos e na Tabela 2, sub-seção 2.1.2, foram calculados os ganhos K_p, K_i e K_d, a Figura 31 apresenta o trecho de código em linguagem de programação Python que calcula os ganhos do controlador PID, bem como, os ganhos calculados para a planta.

Figura 31 – Trecho de código para cálculo dos ganhos K_p, K_i e K_d

```
# função para calcular Kp, Ki e Kd pelo método Ziegler-Nichols
def ziegler_nichols_closed_loop(Kc, Tu):
    Kp = 0.6 * Kc
    Ki = 2 * Kp / Tu
    Kd = Kp * Tu / 8
    return Kp, Ki, Kd
```

Fonte: Autor

A Tabela 6 apresenta os valores obtidos e utilizados para parametrizar a planta de processo contínuos.

Tabela 6 – Valores dos parâmetros de sintonia

Parâmetro	**Valor**
K_p	45.0
K_i	9.91
K_d	51.08
Período de oscilação T_U	9.08 ms

5.2 ARM Cortex M0+ em Linguagem C

De acordo com a sequencia de experimentos definida no capítulo 3, a primeira etapa de coleta de dados foi utilizando a plataforma embarcada Raspberry Pi Pico que possui um microcontrolador RP2040 desenvolvido com um núcleo ARM Cortex M0+.

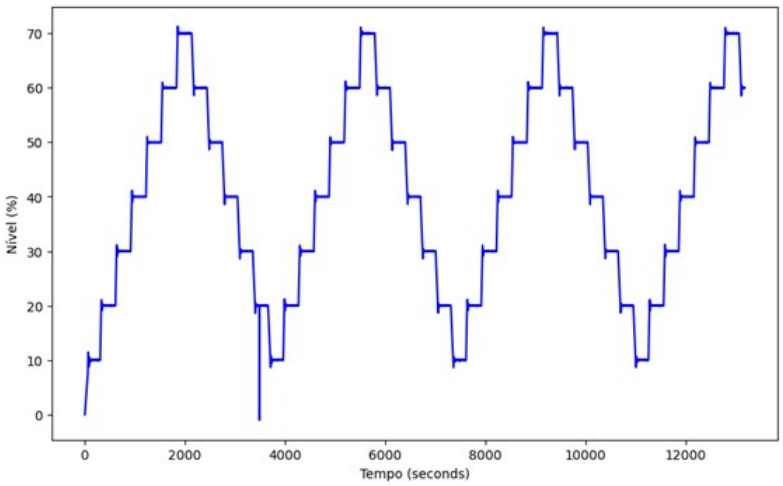
Nesta etapa, o microcontrolador foi programado utilizando a linguagem de programação C/C++ com base no fluxograma da Figura 27 da seção 4.2 deste documento.

O conjunto de dados analisado nesta etapa contém um total de 25.952 registros divididos em 44 amostras, sendo que cada amostra corresponde a um valor de *setpoint*, considerando a varição de 10 a 70 % do nível total do reservatório. A duração total da batelada foi de 3h39min32s.

Para se obter uma visão geral dos dados elaborou-se um gráfico da variação de PV em função do tempo conforme Figura 32. Analisando o gráfico, é possível verificar que ao longo da duração de toda a batelada o sistema funcionou de forma contínua, houve apenas um valor identificado como *outlier* gerado por um ruído na leitura analógica. O ruído foi

eliminado instalando-se um capacitor de 100 nF cerâmico em paralelo com a entrada de sinal da placa de controle, logo após foi novamente testado o sistema, e verificou-se que não houve alteração dos dados, permitindo assim a continuidade dos experimentos.

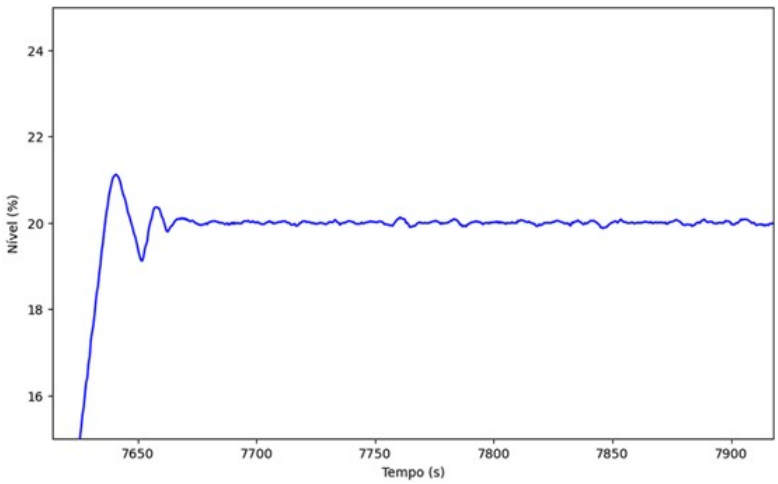
Figura 32 – Gráfico de PV em função do tempo, núcleo ARM Cortex M0+ em linguagem de programação C/C++



Fonte: Autor

Após a análise do gráfico, foi selecionado uma amostra dos dados para análise da atuação do sistema de controle no período de reposta transitória e funcionamento em regime permanente. O gráfico com o trecho selecionado pode ser visto na Figura 33.

Figura 33 – Gráfico de PV em função do tempo, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20%



Fonte: Autor

No gráfico é possível verificar com maior nitidez a região de atuação do sistema de

controle e o período de funcionamento em regime permanente.

Para melhor visualização dos dois momentos, a amostra foi dividida em duas partes.

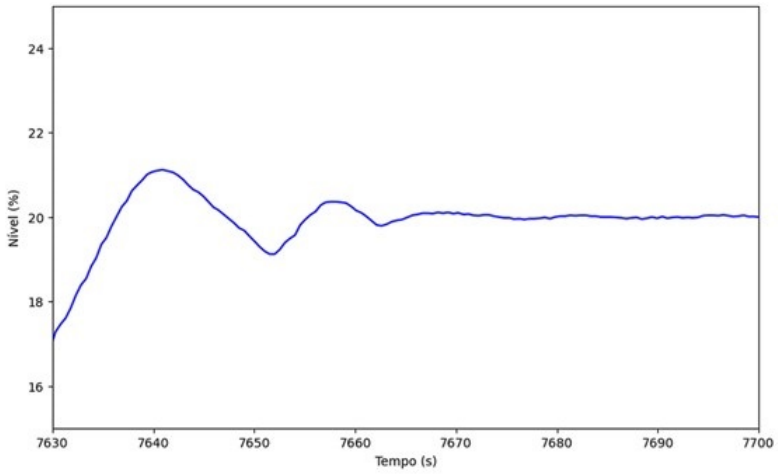
A primeira parte da amostra corresponde apenas a resposta transitória do sistema, conforme pode ser visto na Figura 34.

Na figura, é possível verificar que a região de resposta transitória ocorreu entre os valores 7640 e 7670, esse tempo pode ser justificado em função de algumas características do sistema, que são:

- Escoamento dependente da gravidade;
- Pontos de afunilamento na tubulação que diminuem a vazão do sistema como um todo.

Ao analisarmos o valor obtido para a taxa de decaimento, que foi de 0.0947, destaca-se uma excelente atuação do sistema em manter uma baixa oscilação de PV em torno de SP.

Figura 34 – Gráfico de PV em função do tempo resposta transitória, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20%

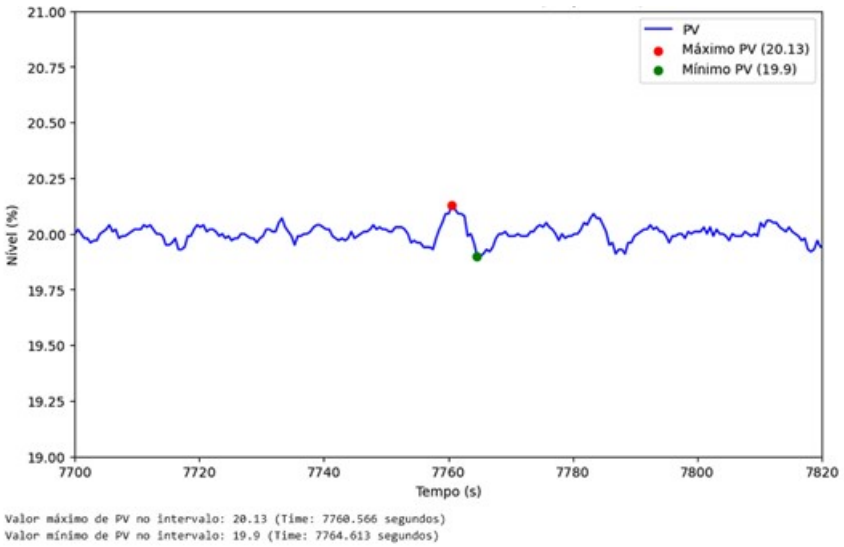


Fonte: Autor

A segunda parte da amostra de dados representa o sistema funcionando em regime permanente e o gráfico que representa esses dados pode ser visto na Figura 35.

Os pontos de máximos e mínimos foram 20,13 % e 19,9 % respectivamente.

Figura 35 – Gráfico de PV em função do tempo regime permanente, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20%



Fonte: Autor

Além dos valores máximos e mínimos, outras métricas estatísticas foram calculadas para a amostra, confirmando o bom desempenho do sistema. Os valores obtidos nos cálculos podem ser vistos na Tabela 7.

Os valores da Média e Mediana são exatamente iguais a SP, o desvio padrão é muito próximo de zero e os percentis 25 e 75 demonstram o bom desempenho do sistemas em manter PV próximo a SP.

Tabela 7 – Métricas estatísticas dos valores de PV no regime permanente, núcleo ARM Cortex M0+ em linguagem de programação C/C++ para amostra com SP 20%

Métrica	**Valor**
Média	20.00
Mediana	20.00
Desvio padrão	0.04
Amplitude	0.23
25th Percentil	19.98
75th Percentil	20.02

5.3 RISC-V em Linguagem C

Dando continuidade aos experimentos, a segunda etapa de coleta de dados foi utilizando a plataforma embarcada LuatOS que possui um microcontrolador ESP32C3 desenvolvido com um núcleo RISC-V. Nesta etapa, o microcontrolador também foi programado utilizando a linguagem de programação C/C++, foi utilizado o mesmo firmware

da primeira etapa, sendo necessário apenas a mudança do GPIO do LED e definição dos pinos I2C.

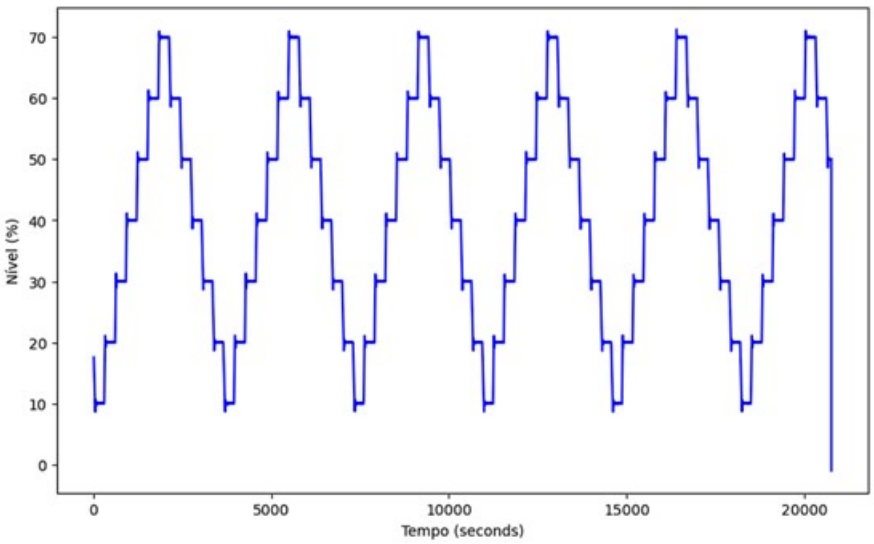
O conjunto de dados analisado nesta etapa contém um total de 41.125 registros divididos em 69 amostras, sendo que cada amostra corresponde a um valor de *setpoint*, considerando a varição de 10 a 70 % do nível total do reservatório.

A duração total da batelada foi de 5h45min32s.

Buscando-se obter uma visão geral dos dados elaborou-se um gráfico da variação de PV em função do tempo Figura 36.

Analisando o gráfico, é possível verificar que ao longo da duração da batelada não houve descontinuidade.

Figura 36 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação C/C++



Fonte: Autor

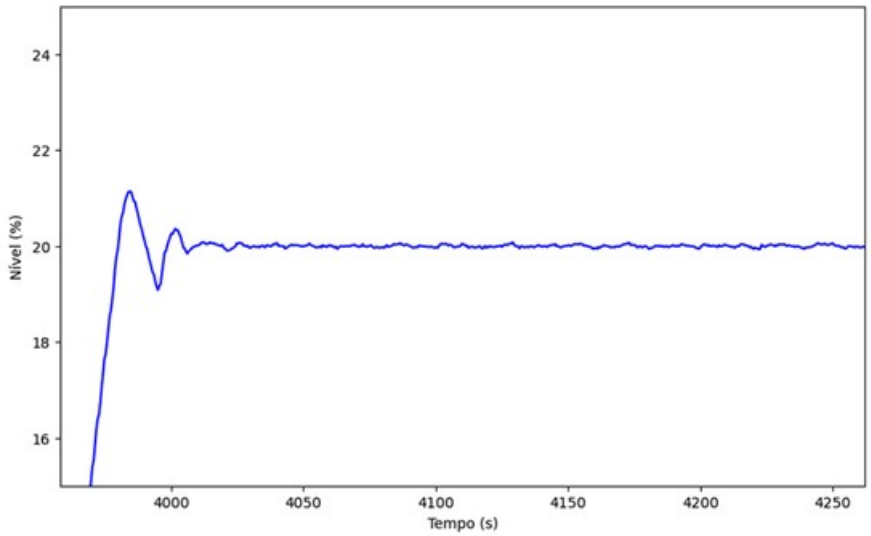
Assim como na primeira etapa, após a análise do gráfico, foi selecionado uma amostra dos dados para análise da atuação do sistema de controle no período de reposta transitória e funcionamento em regime permanente.

O gráfico com o trecho selecionado pode ser visto na Figura 37, é possível verificar que a amostra selecionada corresponde a um trecho onde o SP é 20, visando aproximar ao máximo da análise da primeira etapa do experimento.

No gráfico é possível verificar com maior nitidez a região de atuação do sistema de controle e o período de funcionamento em regime permanente.

Seguindo a mesma metodologia do primeiro experimento, a amostra foi dividida em duas partes.

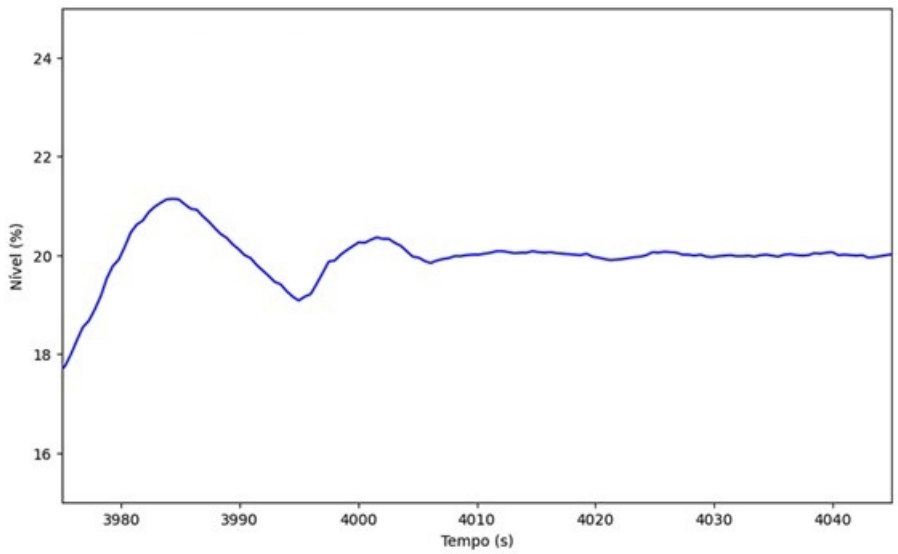
Figura 37 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20%



Fonte: Autor

A primeira parte da amostra corresponde a resposta transitória do sistema, conforme pode ser visto na Figura 38.

Figura 38 – Gráfico de PV em função do tempo resposta transitória, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20%



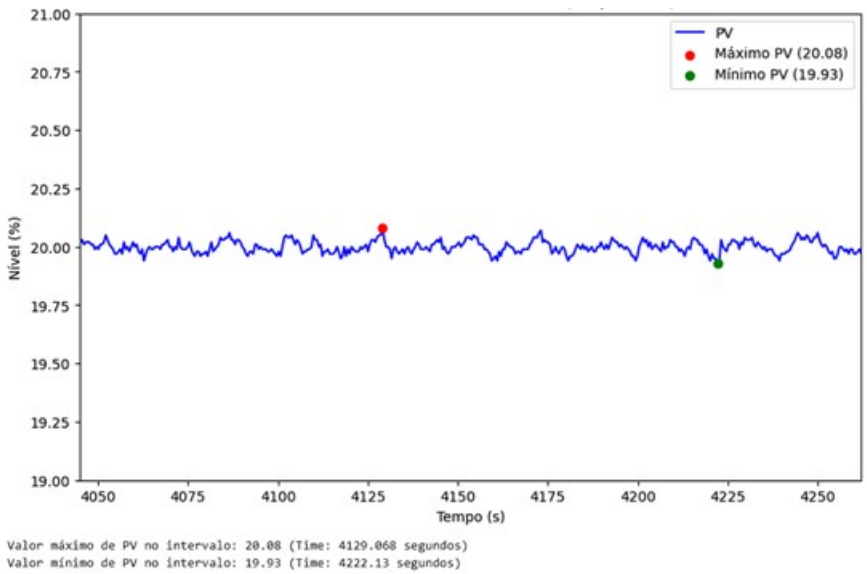
Fonte: Autor

Na figura, é possível verificar que a região de resposta transitória ocorreu entre os valores 3980 e 4010, se compararmos com o valor obtido na primeira fase de teste é

possível notar que o tempo foi o mesmo. Esse resultado comprova que o tempo depende de características intrínsecas do sistema.

O valor obtido para a taxa de decaimento foi de 0.0974, apesar de ligeiramente maior, esse valor comprova a excelente atuação do sistema em manter uma baixa oscilação de PV em torno de SP. A segunda parte da amostra de dados representa o sistema funcionando em regime permanente e o gráfico que representa esses dados pode ser visto na Figura 39. Os pontos de máximos e mínimos foram 20,08 % e 19,93 % respectivamente.

Figura 39 – Gráfico de PV em função do tempo regime permanente, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20%



Fonte: Autor

Algumas métricas estatísticas foram calculadas para a amostra, confirmando o bom desempenho do sistema. Os valores obtidos nos cálculos podem ser vistos na Tabela 8. Os valores da Média e Mediana repetiram o mesmo valores que o núcle ARM Cortex M0+ e foram exatamente igual a SP, o mesmo aconteceu com os valores percentis, o desvio padrão foi ligeiramente melhor atingindo 0,03, comprovando o ótimo desempenho do sistema.

Tabela 8 – Métricas estatísticas dos valores de PV no regime permanente, núcleo RISC-V em linguagem de programação C/C++ para amostra com SP 20%

Métrica	**Valor**
Média	20.00
Mediana	20.00
Desvio padrão	0.03
Amplitude	0.15
25th Percentil	19.98
75th Percentil	20.02

5.4 RISC-V em Linguagem MicroPython

Após os experimentos realizados com linguagem C/C++ para as duas plataformas, a próxima etapa do experimento foi a alteração da linguagem de programação. Nesta etapa, o microcontrolador foi programado utilizando a linguagem de programação MicroPython com base no fluxograma da Figura 28 da seção 4.2 deste documento.

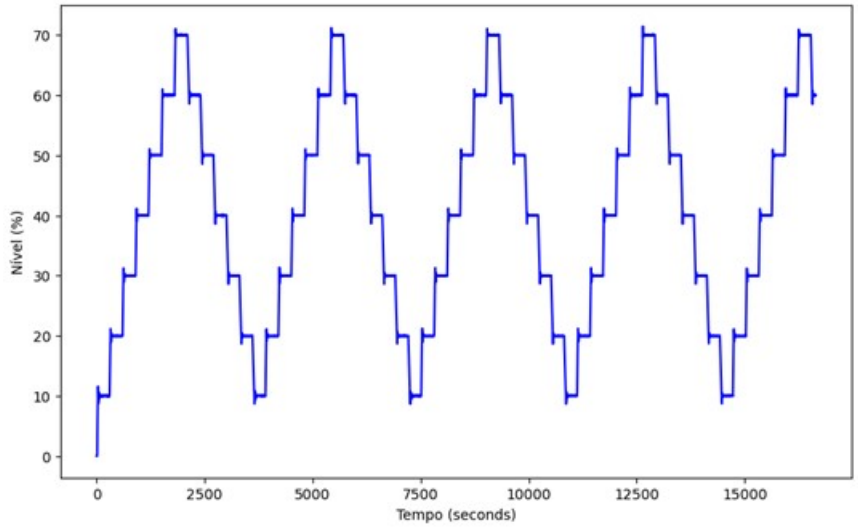
Foi realizada uma nova batelada para coleta de dados. O conjunto de dados analisado nesta etapa contém um total de 33.285 registros divididos em 56 amostras, sendo que cada amostra corresponde a um valor de *setpoint*, considerando a mesma varição de 10 a 70 % do nível total do reservatório, conforme os experimentos anteriores.

A duração total da batelada foi de 4h37min22s.

A visão geral dos dados é apresentada no gráfico da variação de PV em função do tempo Figura 40.

No gráfico, é possível verificar que ao longo da duração da batelada não houve descontinuidade, repetindo o mesmo desempenho dos demais arranjos experimentais.

Figura 40 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação MicroPython

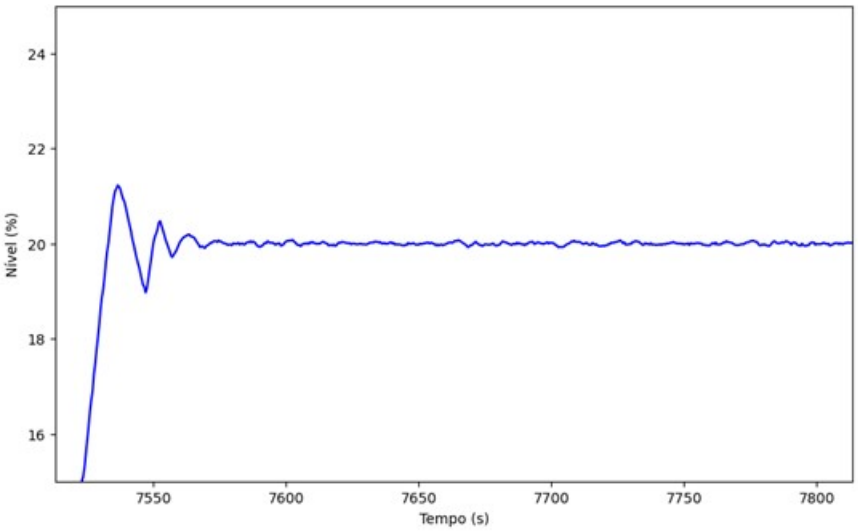


Fonte: Autor

Seguindo a mesma metodologia dos experimentos anteriores, após a análise do gráfico, foi selecionado uma amostra dos dados para análise da atuação do sistema de controle no período de reposta transitória e funcionamento em regime permanente.

O gráfico com o trecho selecionado pode ser visto na Figura 41, é possível verificar que a amostra selecionada corresponde a um trecho onde o SP é 20, visando aproximar ao máximo das análises da primeira e segunda etapa do experimento.

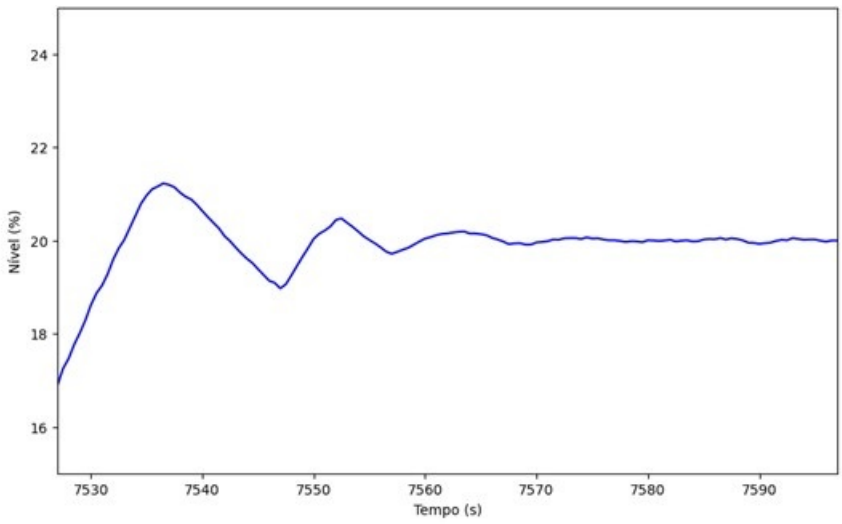
Figura 41 – Gráfico de PV em função do tempo, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20%



Fonte: Autor

Assim como nos demais experimentos, a amostra foi dividida em duas partes. A primeira parte da amostra corresponde a resposta transitória do sistema, conforme pode ser visto na Figura 42.

Figura 42 – Gráfico de PV em função do tempo resposta transitória, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20%

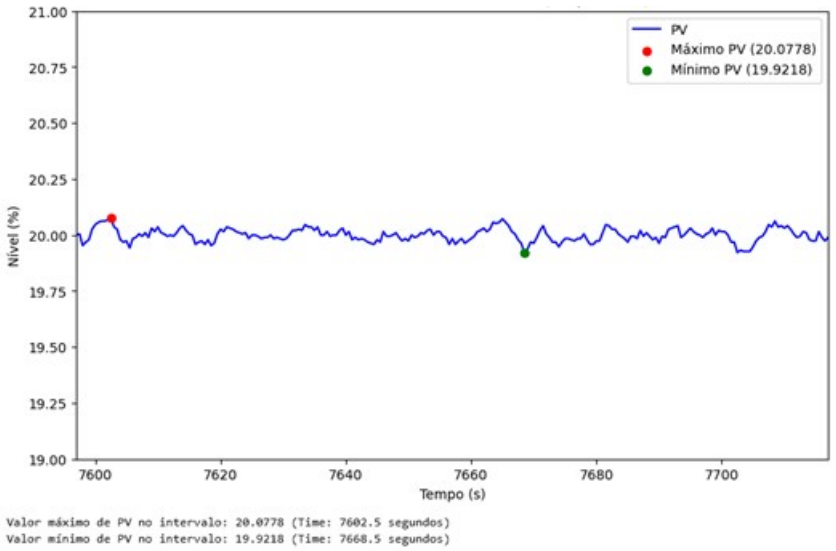


Fonte: Autor

O valor obtido para a taxa de decaimento foi de 0.0947, o mesmo valor alcançado na primeira etapa, esse valor comprova a excelente atuação do sistema em manter uma baixa oscilação de PV em torno de SP. A segunda parte da amostra de dados representa o sistema

funcionando em regime permanente e o gráfico que representa esses dados pode ser visto na Figura 43. Os pontos de máximos e mínimos foram 20,07 % e 19,92 % respectivamente.

Figura 43 – Gráfico de PV em função do tempo regime permanente, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20%



Fonte: Autor

Seguindo a mesma estratégia dos demais experimentos, métricas estatísticas foram calculadas para a amostra, confirmando o bom desempenho do sistema. Os valores obtidos nos cálculos podem ser vistos na Tabela 9.

Os valores da Média e Mediana repetiram o mesmos valores que os experimentos anteriores e foram exatamente igual a SP, o mesmo aconteceu com os valores percentis, o desvio padrão foi 0,03, comprovando o ótimo desempenho do sistema.

Tabela 9 – Métricas estatísticas dos valores de PV no regime permanente, núcleo RISC-V em linguagem de programação MicroPython para amostra com SP 20%

Métrica	**Valor**
Média	20.00
Mediana	20.00
Desvio padrão	0.03
Amplitude	0.16
25th Percentil	19.98
75th Percentil	20.02

5.5 Análise Comparativa de Dados

Considerando o escopo deste trabalho, após feita de forma individual a análise para cada arranjo experimental definido na metodologia, foi realizada uma análise comparativa

dos resultados obtidos.

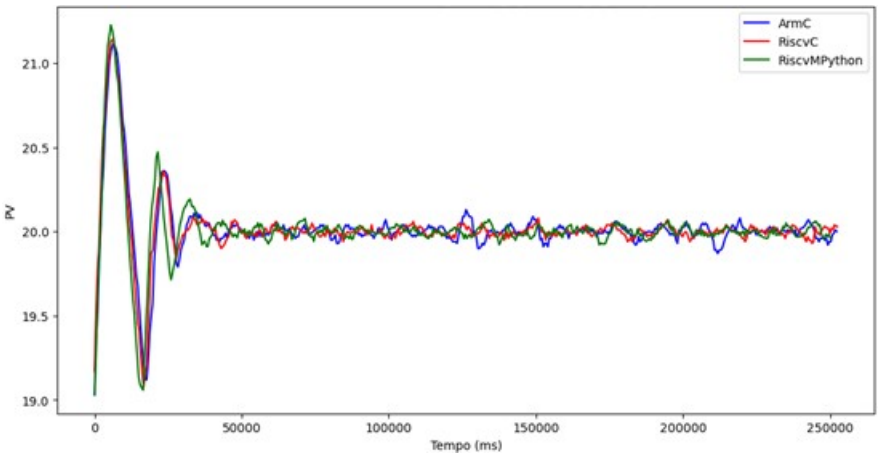
Esta etapa iniciou pela normalização dos dados, pois, apesar das amostras utilizadas serem referentes ao mesmo valor de SP, os valores de tempos relativos dos dados são diferentes.

O processo foi iniciado selecionando o mesmo número de registros a partir de um valor mínimo comum para as três amostras. A partir desse valor, foram selecionados os próximos 581 valores, isso corresponde a 4min51s, uma boa amostragem, considerando que o tempo total para um dado *setpoint* é de 5 minutos.

De posse dos valores, para cada amostra o valor do primeiro registro de tempo foi alterada para zero, promovendo assim a mesma origem para o eixo das variáveis independentes. O objetivo dessa estratégia e ter uma mesma referência para análise comparativa dos dados.

Com os dados normalizados, foi elaborado um gráfico de linha contendo os dados dos três conjunto de dados já normalizados. O gráfico gerado pode ser visto na Figura 44.

Figura 44 – Gráfico de PV em função do tempo com os três arranjos experimentais e SP 20%



Fonte: Autor

Ao analisarmos o gráfico obtido é possível perceber que o comportamento do sistema de controle é muito próximo para os três arranjos experimentais, resultado já predito pelas análises individuais dos experimentos. A pequena diferença observada, fica atribuída aos ajustes de tempo entre as amostras, pois os dados sobrepostos não foram obtidos no mesmo instante de tempo.

Considerando uma análise mais quantitativa dos dados, foram calculadas as Soma dos Erros Quadráticos (SSE) tendo como referência a plataforma microcontrolada com arquitetura ARM Cortex M0+ em relação aos outros dois arranjos. O trecho de código

que calcula esse valor pode ser visto na Figura 45.

Figura 45 – Trecho de código para cálculo do SSE em relação a plataforma Raspberry Pi Pico

```
# cálculo da Soma dos Erros Quadráticos (SSE)
y_ref = dfArmC_SP20['pv']
y_riscvC = dfRiscvC_SP20['pv']
y_riscvMPython = dfRiscvMPython_SP20['pv']

sse_riscvC = ((y_ref - y_riscvC) ** 2).sum()
sse_riscvMPython = ((y_ref - y_riscvMPython) ** 2).sum()

print("Soma dos Erros Quadráticos (SSE) para RiscvC_SP20:", sse_riscvC)
print("Soma dos Erros Quadráticos (SSE) para RiscvMPython_SP20:", sse_riscvMPython)
```

```
Soma dos Erros Quadráticos (SSE) para RiscvC_SP20: 2.0315000000000056
Soma dos Erros Quadráticos (SSE) para RiscvMPython_SP20: 7.074120479999994
```

Fonte: Autor

Os resultados obtidos de SSE confirmam o excelente desempenho do microcontrolador ESP32C3 em relação ao RP2040 em circuitos de controle PID.

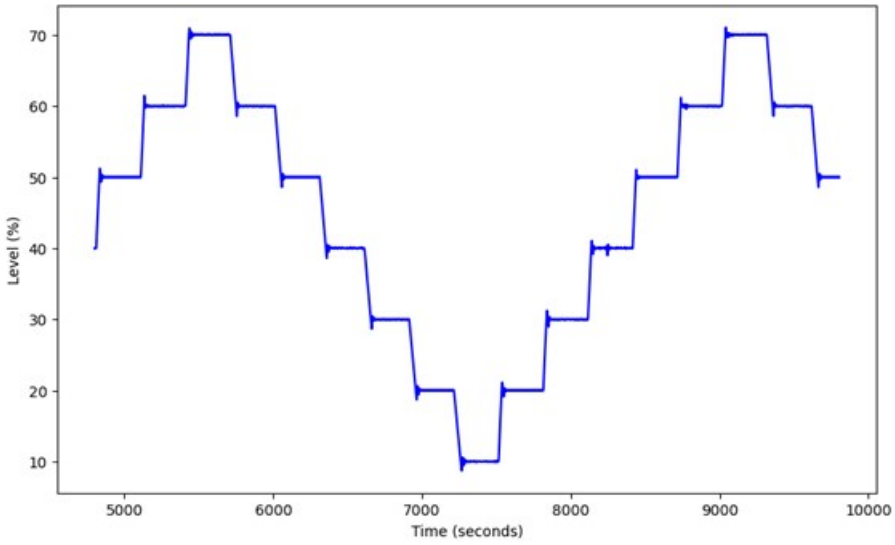
5.6 Bipes e IoT

O BIPES (*Block-based Integrated Platform for Embedded Systems*) é uma plataforma web que facilita o desenvolvimento de sistemas embarcados e aplicações de Internet das Coisas (IoT). Utiliza o conceito de programação por blocos, eliminando a necessidade de instalação de software e permitindo que qualquer dispositivo com navegador, seja utilizado para programar e testar dispositivos IoT.

Buscando validar a utilização da plataforma BIPES no desenvolvimento de sistemas de controle PID foi desenvolvido um firmware em linguagem de programação MicroPython através da plataforma Bipes. A refêrencia para o desenvolvimento foi o fluxograma da Figura 28 da seção 4.2 deste documento.

Além do algoritmo de controle da planta, foram incluídos blocos para conexão da placa de controle com a rede Wi-Fi e envio dos dados para uma plataforma em nuvem via protocolo de comunicação MQTT. A integração de todas essas tecnologias permitem o desenvolvimento de um dispositivo IoT para controle e monitoramento de plantas de processos contínuos. O gráfico dos dados gerados durante essa etapa pode ser visto na Figura 46.

Figura 46 – Gráfico de PV em função do tempo utilizando dados gerados com recursos em nuvem, núcleo RISC-V em linguagem de programação MicroPython desenvolvido pela plataforma BIPES



Fonte: Autor

Ao analisar os dados é possível verificar que não houve descontinuidade ao longo da batelada, isso comprova que mesmo com a utilização de mais recursos de hardware, como por exemplo, a utilização da pilha TCP/IP e o módulo Wi-Fi, o dispositivo manteve seu bom desempenho como controlador PID.

6 . CONCLUSÃO

Neste capítulo será discutido se o escopo do trabalho foi atingido com base nos resultados apresentados no capítulo 5.

6.1 Estado atual

Considerando o objetivo do trabalho que foi, avaliar o desempenho de um microcontrolador que utiliza a arquitetura RISC-V frente a um microcontrolador com arquitetura ARM, os resultado obtidos demonstraram que o desempenho do microcontrolador avaliado foi igual e em alguns casos, ligeiramente superior ao microcontrolador de referência.

Em relação ao funcionamento do dispositivo em resposta transitória, foi verificado que o microcontrolador com RISC-V obteve uma taxa de caimento no valor de SP de 0,0974, muito abaixo do definido como ideal que é 0,25.

Avaliando o remime permanente de trabalho, foi possível evidenciar que ao longo do período de realização dos experimentos, não houve descontinuidade do sinal, o que mostra a eficiência do sistema de controle PID. Observando especificamente os valores entre 25 e 75 percentis, foi constatada uma variação de apenas 0,1% de PV em relação a SP, comprovando o resultado excelente do sistema.

Entende-se que o objetivo principal deste trabalho foi alcançado, sendo possível verificar o excelente desempenho do microcontrolador com arquitetura RISC-V em aplicação de sistema de controle PID frente a uma arquitetura ARM Cortex M0+, utilizada como padrão de referência.

Sobre as linguagens de programação, foi verificado que, o desempenho do microcontrolador com núcleo RISC-V foi praticamente o mesmo para a linguagem C/C++ e MicroPython, restringindo-se ao tipo de aplicação a qual esse trabalho se propôs. Com base nessa evidência, foram disponibilizados, os dados obtidos neste trabalho, o esquema elétrico do hardware, os firmwares em linguagem C/C++ e MycroPython e um tutorial passo-a-passo de como implementar um sistema de controle PID utilizando a plataforma BIPES. Todo material pode ser acessado na url: <https://github.com/andreroberto83/PID-ARM-RISCV>.

6.2 Estado futuro

A partir dos resultados obtidos neste trabalho abre-se a oportunidade de pesquisas mais específicas em relação ao tema proposto, algumas possibilidades são:

- Desenvolvimento de um controlador PID com núcleo RISC-V e programação por blocos;

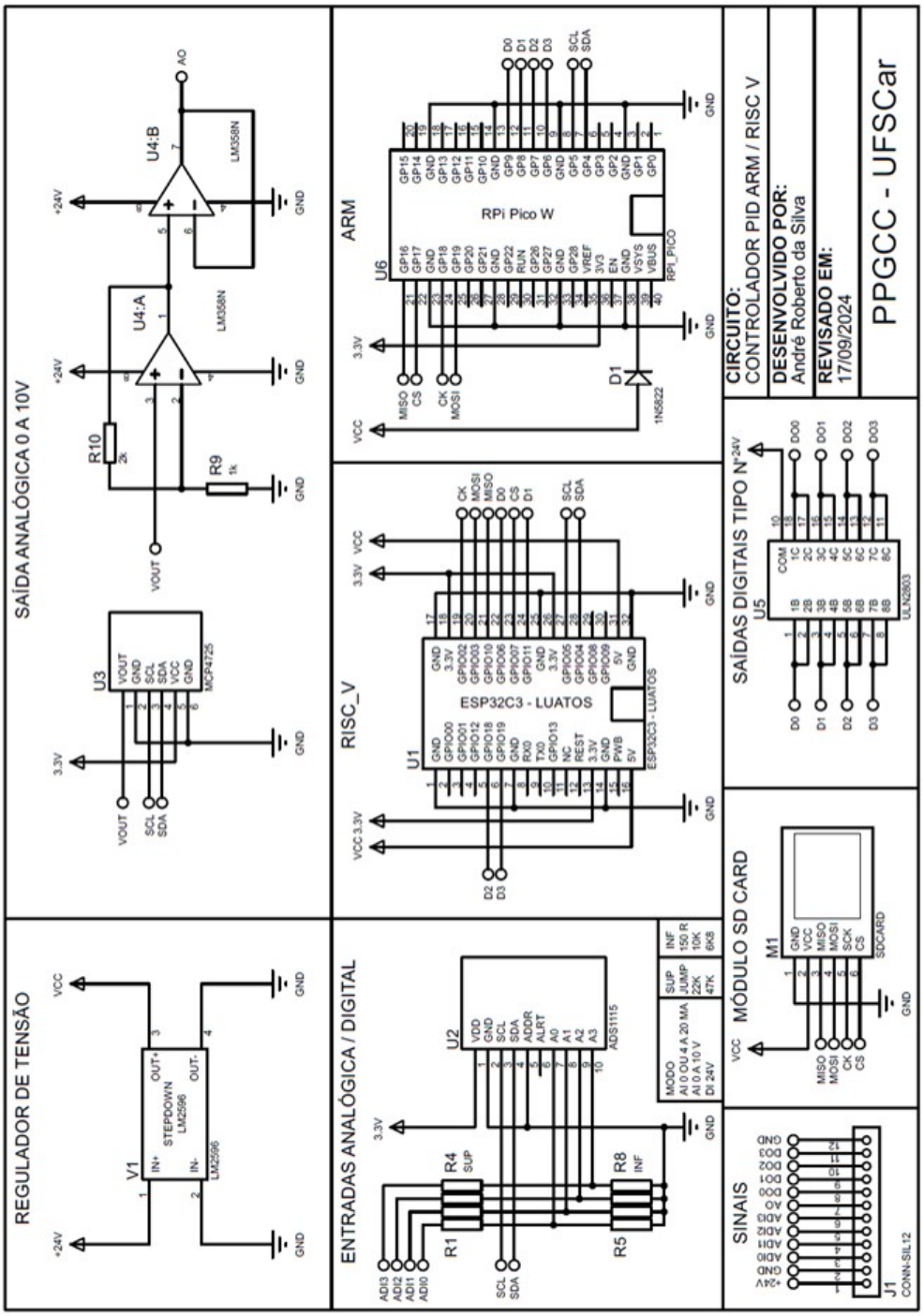
- Validação de sistema embarcado com núcleo RISC-V em longo períodos de tempo em chão-de-fábrica;
- Proposta de plataforma IoT para programação de hardware e coleta de dados com foco em sistema de controle PID.

7 . APÊNDICE

7.1 Apêndice A - Esquema elétrico

Na Figura 47 pode ser visto o esquema elétrico do circuito.

Figura 47 – Esquema elétrico completo do circuito desenvolvido



Fonte: Autor

7.2 Apêndice B - Código em linguagem C/C++

Na Figura 48 pode ser visto o trecho do código referente ao *loop* principal em linguagem C/C++.

Figura 48 – *Loop* principal do código em linguagem C/C++

```
void loop() {
  processValue = medirNivel();
  controlePID.SetTunings(kp, ki, kd);
  controlePID.Compute();
  if (!kp && !ki && !kd)
    ajustarInversor(0.0);
  else{
    ajustarInversor(controlVariable);
  }
  // led heart inverte o estado a cada 500 ms
  // registro de valores a cada 500 ms
  // altera o SP a cada 5 minutos
  tempoAtual = millis();
  if (tempoAtual - tempoAnterior >= intervalo)
  {
    tempoAnterior = tempoAtual;
    digitalWrite(led, !digitalRead(led));

    // se modo 0 envia valores via serial
    // se modo 1 salva valores no cartão SD
    // se modo 2 envia valores via serial e salva valores no cartão SD
    if (modo == 0 || modo == 2){
      enviarValores();
      if (modo == 2)
        salvarDadosSD();
    }
    else
      salvarDadosSD();
    // altera o SP entre o intervalo de 10 a 70 com incrementos/decrementos de 10
    if (contadorSetPoint < 600)
      contadorSetPoint++;
    else{
      contadorSetPoint = 0;
      // se chegou no sp mínimo de teste
      if (setPoint <= 10 && !spUp){
        spUp = true;
      }
      // se chegou no sp máximo de teste
      if (setPoint >= 70 && spUp){
        spUp = false;
      }
      // incrementa ou decramenta o sp
      if (setPoint < 70 && spUp)
        setPoint += 10;
      else if (setPoint > 10 && !spUp)
        setPoint -= 10;
    }
  }
  // ajusta valores via serial
  if (Serial.available() > 0){
    setarParametro();
  }
}
```

Fonte: Autor

7.3 Apêndice C - Código de controle PID em linguagem C/C++

Na Figura 49 pode ser visto o primeiro trecho do código referente a rotina controle PID em linguagem C/C++.

Figura 49 – Rotina de controle PID em linguagem C - Parte 1

```
#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#include <PID_v1.h>

PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int POn, int ControllerDirection)
    : myOutput(Output), myInput(Input), mySetpoint(Setpoint),
      inAuto(false), lastTime(millis() - SampleTime)
{
    SetOutputLimits(0, 255);
    SampleTime = 100;
    SetControllerDirection(ControllerDirection);
    SetTunings(Kp, Ki, Kd, POn);
}

PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int ControllerDirection)
    : PID(Input, Output, Setpoint, Kp, Ki, Kd, P_ON_E, ControllerDirection) {}

bool PID::Compute()
{
    if (!inAuto) return false;

    unsigned long now = millis();
    if (now - lastTime >= SampleTime)
    {
        double error = *mySetpoint - *myInput;
        double dInput = *myInput - lastInput;
        outputSum += ki * error;
        if (!pOnE) outputSum -= kp * dInput;
        outputSum = constrain(outputSum, outMin, outMax);
        *myOutput = constrain((pOnE ? (kp * error) : 0) + outputSum - kd * dInput, outMin, outMax);
        lastInput = *myInput;
        lastTime = now;
        return true;
    }
    return false;
}

void PID::SetTunings(double Kp, double Ki, double Kd, int POn)
{
    if (Kp < 0 || Ki < 0 || Kd < 0) return;
    pOn = POn;
    pOnE = (POn == P_ON_E);
    double SampleTimeInSec = SampleTime / 1000.0;
    kp = Kp; ki = Ki * SampleTimeInSec; kd = Kd / SampleTimeInSec;
    if (controllerDirection == REVERSE) kp = -kp, ki = -ki, kd = -kd;
}
```

Fonte: Autor

Na Figura 50 pode ser visto o segundo trecho do código referente a rotina controle PID em linguagem C/C++.

Figura 50 – Rotina de controle PID em linguagem C - Parte 2

```
void PID::SetSampleTime(int NewSampleTime)
{
    if (NewSampleTime > 0)
    {
        double ratio = static_cast<double>(NewSampleTime) / SampleTime;
        ki *= ratio; kd /= ratio;
        SampleTime = static_cast<unsigned long>(NewSampleTime);
    }
}

void PID::SetOutputLimits(double Min, double Max)
{
    if (Min >= Max) return;
    outMin = Min; outMax = Max;
    if (inAuto)
    {
        *myOutput = constrain(*myOutput, outMin, outMax);
        outputSum = constrain(outputSum, outMin, outMax);
    }
}

void PID::SetMode(int Mode)
{
    if (Mode == AUTOMATIC && !inAuto) Initialize();
    inAuto = (Mode == AUTOMATIC);
}

void PID::Initialize()
{
    outputSum = *myOutput;
    lastInput = *myInput;
    outputSum = constrain(outputSum, outMin, outMax);
}

void PID::SetControllerDirection(int Direction)
{
    if (inAuto && Direction != controllerDirection)
    {
        kp = -kp; ki = -ki; kd = -kd;
    }
    controllerDirection = Direction;
}

double PID::GetKp() { return dispKp; }
double PID::GetKi() { return dispKi; }
double PID::GetKd() { return dispKd; }
int PID::GetMode() { return inAuto ? AUTOMATIC : MANUAL; }
int PID::GetDirection() { return controllerDirection; }
```

Fonte: Autor

7.4 Apêndice D - Código em linguagem MicroPython

Na Figura 51 pode ser visto o trecho do código referente ao *loop* principal em linguagem MicroPython.

Figura 51 – Loop principal do código em linguagem MicroPython

```
while True:
    nivel = aux.lerNivel()
    saida = aux.calcularPID(nivel)
    aux.ajustarInversor(saida)

    if dados == True:
        dados = False
        tim0.deinit()

        led.value(not led.value())

        if modo == False:
            print(aux.formatarDados(tempo500ms, nivel, saida), end="")
        else:
            aux.salvarDados(aux.formatarDados(tempo500ms, nivel, saida))

        # se não passou 5 min.
        if contadorMinuto < 600:
            contadorMinuto += 1
        else:
            contadorMinuto = 0
            sp = aux.verificarSp()

            if incrementaSp == True:
                if sp < 70:
                    sp += 10
                else:
                    incrementaSp = False
                    sp -= 10
            else:
                if sp > 10:
                   sp -= 10
                else:
                   incrementaSp = True
                   sp += 10

            aux.alterarSp(sp)

        tim0.init(period=500, mode=Timer.PERIODIC, callback=millis500)
```

Fonte: Autor

7.5 Apêndice E - Código de controle PID em linguagem MicroPython

Na Figura 52 pode ser visto o primeiro trecho do código referente a rotina controle PID em linguagem MicroPython.

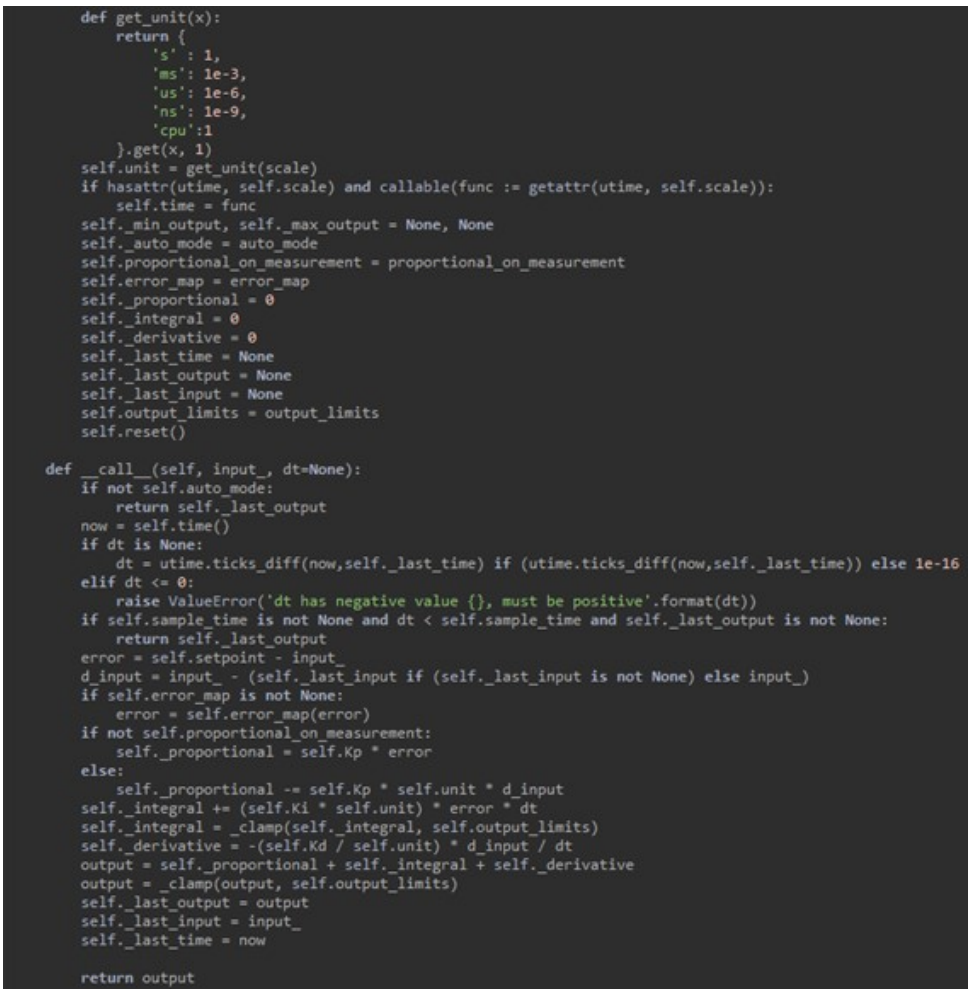
Figura 52 – Rotina de controle PID em linguagem MicroPython - Parte 1

```
import utime

def _clamp(value, limits):
    lower, upper = limits
    if value is None:
        return None
    elif (upper is not None) and (value > upper):
        return upper
    elif (lower is not None) and (value < lower):
        return lower
    return value

class PID(object):
    def __init__(
        self,
        Kp=1.0,
        Ki=0.0,
        Kd=0.0,
        setpoint=0,
        sample_time=None,
        scale='s',
        output_limits=[None, None],
        auto_mode=True,
        proportional_on_measurement=False,
        error_map=None
    ):
        self.Kp, self.Ki, self.Kd = Kp, Ki, Kd
        self.setpoint = setpoint
        self.sample_time = sample_time

        def get_scale(x):
            return {
                's' : 'time',
                'ms': 'ticks_ms',
                'us': 'ticks_us',
                'ns': 'time_ns',
                'cpu':'ticks_cpu'
            }.get(x, 'time')
        self.scale = get_scale(scale)
```

Fonte: Autor

Na Figura 53 pode ser visto o segundo trecho do código referente a rotina controle PID em linguagem MicroPython.

Figura 53 – Rotina de controle PID em linguagem MicroPython - Parte 2



```
        def get_unit(x):
            return {
                's' : 1,
                'ms': 1e-3,
                'us': 1e-6,
                'ns': 1e-9,
                'cpu':1
            }.get(x, 1)
        self.unit = get_unit(scale)
        if hasattr(utime, self.scale) and callable(func := getattr(utime, self.scale)):
            self.time = func
        self._min_output, self._max_output = None, None
        self._auto_mode = auto_mode
        self.proportional_on_measurement = proportional_on_measurement
        self.error_map = error_map
        self._proportional = 0
        self._integral = 0
        self._derivative = 0
        self._last_time = None
        self._last_output = None
        self._last_input = None
        self.output_limits = output_limits
        self.reset()

    def __call__(self, input_, dt=None):
        if not self.auto_mode:
            return self._last_output
        now = self.time()
        if dt is None:
            dt = utime.ticks_diff(now,self._last_time) if (utime.ticks_diff(now,self._last_time)) else 1e-16
        elif dt <= 0:
            raise ValueError('dt has negative value {}, must be positive'.format(dt))
        if self.sample_time is not None and dt < self.sample_time and self._last_output is not None:
            return self._last_output
        error = self.setpoint - input_
        d_input = input_ - (self._last_input if (self._last_input is not None) else input_)
        if self.error_map is not None:
            error = self.error_map(error)
        if not self.proportional_on_measurement:
            self._proportional = self.Kp * error
        else:
            self._proportional -= self.Kp * self.unit * d_input
        self._integral += (self.Ki * self.unit) * error * dt
        self._integral = _clamp(self._integral, self.output_limits)
        self._derivative = -(self.Kd / self.unit) * d_input / dt
        output = self._proportional + self._integral + self._derivative
        output = _clamp(output, self.output_limits)
        self._last_output = output
        self._last_input = input_
        self._last_time = now

        return output
```

Fonte: Autor

Na Figura 54 pode ser visto o terceiro trecho do código referente a rotina controle PID em linguagem MicroPython.

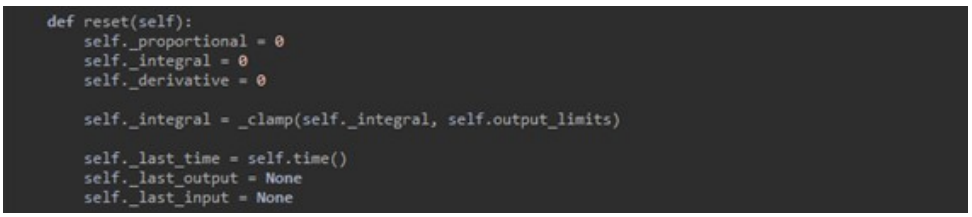
Figura 54 – Rotina de controle PID em linguagem MicroPython - Parte 3

```
    def __repr__(self):
        return (
            '{self.__class__.__name__}('
            'Kp={self.Kp!r}, Ki={self.Ki!r}, Kd={self.Kd!r}, '
            'setpoint={self.setpoint!r}, sample_time={self.sample_time!r}, '
            'output_limits={self.output_limits!r}, auto_mode={self.auto_mode!r}, '
            'proportional_on_measurement={self.proportional_on_measurement!r},'
            'error_map={self.error_map!r}'
            ')'
        ).format(self=self)

    @property
    def components(self):
        return self._proportional, self._integral, self._derivative

    @property
    def tunings(self):
        return self.Kp, self.Ki, self.Kd

    @tunings.setter
    def tunings(self, tunings):
        self.Kp, self.Ki, self.Kd = tunings

    @property
    def auto_mode(self):
        return self._auto_mode

    @auto_mode.setter
    def auto_mode(self, enabled):
        self.set_auto_mode(enabled)

    def set_auto_mode(self, enabled, last_output=None):
        if enabled and not self._auto_mode:
            self.reset()

            self._integral = last_output if (last_output is not None) else 0
            self._integral = _clamp(self._integral, self.output_limits)

        self._auto_mode = enabled

    @property
    def output_limits(self):
        return self._min_output, self._max_output

    @output_limits.setter
    def output_limits(self, limits):
        if limits is None:
            self._min_output, self._max_output = None, None
            return
        min_output, max_output = limits
        if (None not in limits) and (max_output < min_output):
            raise ValueError('lower limit must be less than upper limit')
        self._min_output = min_output
        self._max_output = max_output
        self._integral = _clamp(self._integral, self.output_limits)
        self._last_output = _clamp(self._last_output, self.output_limits)
```

Fonte: Autor

Na Figura 55 pode ser visto o quarto trecho do código referente a rotina controle PID em linguagem MicroPython.

Figura 55 – Rotina de controle PID em linguagem MicroPython - Parte 4



```
    def reset(self):
        self._proportional = 0
        self._integral = 0
        self._derivative = 0

        self._integral = _clamp(self._integral, self.output_limits)

        self._last_time = self.time()
        self._last_output = None
        self._last_input = None
```

Fonte: Autor

REFERÊNCIAS

AGHENTA, L. O.; IQBAL, M. T. Design and implementation of a low-cost, open source iot-based scada system using esp32 with oled, thingsboard and mqtt protocol. **AIMS Electronics and Electrical Engineering**, American Institute of Mathematical Sciences (AIMS), v. 4, n. 1, p. 57–86, 2020. ISSN 2578-1588. Disponível em: <http://dx.doi.org/10.3934/ElectrEng.2020.1.57>.

ARM. **Cortex-M0+ Technical Reference Manual**. 2012. <https://developer.arm.com/documentation/ddi0484/c>. [Accessed 22-07-2023].

ARM. **ARM®v6-M Architecture Reference Manual**. [S.l.], 2018. E.

BAPTISTA, E. C. **O Futuro Tecnológico Com Sistemas Embarcados - Arduino**. [S.l.]: Clube de Autores, 2023.

CUI, E.; LI, T.; WEI, Q. Risc-v instruction set architecture extensions: A survey. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 11, p. 24696–24711, 2023. ISSN 2169-3536. Disponível em: <http://dx.doi.org/10.1109/ACCESS.2023.3246491>.

DALENOGARE, L. S. et al. The expected contribution of industry 4.0 technologies for industrial performance. **International Journal of Production Economics**, Elsevier BV, v. 204, p. 383–394, out. 2018. ISSN 0925-5273. Disponível em: <http://dx.doi.org/10.1016/j.ijpe.2018.08.019>.

DORF, R. C.; H., R. **Modern Control Systems**. 8. ed. Upper Saddle River, NJ: Pearson, 1997. (World student series).

DöRFLINGER, A. et al. A comparative survey of open-source application-class risc-v processor implementations. In: **Proceedings of the 18th ACM International Conference on Computing Frontiers**. ACM, 2021. (CF '21). Disponível em: <http://dx.doi.org/10.1145/3457388.3458657>.

ESPRESSIF. **ESP32 A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications**. 2023. <https://www.espressif.com/en/products/socs/esp32>. [Accessed 12-09-2023].

ESPRESSIF. **ESP32-C3 Series Datasheet**. [S.l.], 2023. 1.45.

FROLOV, V. A.; GALAKTIONOV, V. A.; SANZHAROV, V. V. Investigation of risc-v. **Programming and Computer Software**, Pleiades Publishing Ltd, v. 47, n. 7, p. 493–504, dez. 2021. ISSN 1608-3261. Disponível em: <http://dx.doi.org/10.1134/S0361768821070045>.

HERDT, V. et al. Risc-v based virtual prototype: An extensible and configurable platform for the system-level. **Journal of Systems Architecture**, Elsevier BV, v. 109, p. 101756, out. 2020. ISSN 1383-7621. Disponível em: <http://dx.doi.org/10.1016/j.sysarc.2020.101756>.

IONESCU, T. B. Leveraging graphical user interface automation for generic robot programming. **Robotics**, MDPI AG, v. 10, n. 1, p. 3, dez. 2020. ISSN 2218-6581. Disponível em: <http://dx.doi.org/10.3390/robotics10010003>.

JAMIESON, P. et al. Computer engineering education experiences with risc-v architectures—from computer architecture to microcontrollers. **Journal of Low Power Electronics and Applications**, MDPI AG, v. 12, n. 3, p. 45, ago. 2022. ISSN 2079-9268. Disponível em: <http://dx.doi.org/10.3390/jlpea12030045>.

JUNIOR, A. G. D. S. et al. Bipes: Block based integrated platform for embedded systems. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 197955–197968, 2020. ISSN 2169-3536. Disponível em: <http://dx.doi.org/10.1109/ACCESS.2020.3035083>.

MAIER, A.; SHARP, A.; VAGAPOV, Y. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. In: **2017 Internet Technologies and Applications (ITA)**. IEEE, 2017. Disponível em: <http://dx.doi.org/10.1109/ITECHA.2017.8101926>.

NISE, N. **Engenharia de Sistemas de Controle**. LTC, 2023. ISBN 9788521638278. Disponível em: <https://books.google.com.br/books?id=eL0Q0AEACAAJ>.

OGATA, K. **Engenharia de Controle Moderno**. 5. ed. São Paulo: Pearson, 2010. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 05 nov. 2024.

OLIVEIRA, S. de. **Internet das Coisas com ESP8266, Arduino e Raspberry Pi**. [S.l.]: Novatec Editora, 2017.

PATTERSON, D. A.; WATERMAN, A. **The RISC-V reader: An open architecture atlas**. [S.l.]: Strawberry Canyon LLC, 2017.

PLAUSKA, I.; LIUTKEVIčIUS, A.; JANAVIčIūTė, A. Performance evaluation of c/c++, micropython, rust and tinygo programming languages on esp32 microcontroller. **Electronics**, MDPI AG, v. 12, n. 1, p. 143, dez. 2022. ISSN 2079-9292. Disponível em: <http://dx.doi.org/10.3390/electronics12010143>.

RASPBERRY PI. **RP2040 Datasheet**. [S.l.], 2023. A6fe703-clean.

RODRIGUES, D. S. N. d.; OURIQUES, H. R. O brasil frente à indústria 3.0 e 4.0: Consequências para a sua posição estrutural na economia-mundo. In: **Anais do Seminário de Graduação e Pós-graduação em Relações Internacionais**. São Paulo, Brasil: Even3, 2022. ISBN 978-65-5941-910-4. Disponível em: <https://www.even3.com.br/anais/spabri2022/500641-o-brasil-frente-a-industria-30-e-40--consequencias-para-a-sua-posicao-estrutural-na-economia-mund>.

SANCHIS, R. et al. Low-code as enabler of digital transformation in manufacturing industry. **Applied Sciences**, MDPI AG, v. 10, n. 1, p. 12, dez. 2019. ISSN 2076-3417. Disponível em: <http://dx.doi.org/10.3390/app10010012>.

SANTOS, D. A. et al. Reliability analysis of a fault-tolerant risc-v system-on-chip. **Microelectronics Reliability**, Elsevier BV, v. 125, p. 114346, out. 2021. ISSN 0026-2714. Disponível em: <http://dx.doi.org/10.1016/j.microrel.2021.114346>.

STALLINGS, W. **Arquitetura e organização de computadores**. [S.l.]: Pearson, 2010.

Sá, B.; MARTINS, J.; PINTO, S. **A First Look at RISC-V Virtualization from an Embedded Systems Perspective**. arXiv, 2021. Disponível em: <https://arxiv.org/abs/2103.14951>.