INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* SABARÁ
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Hebert Emmanuel Rocha Peluso

IMPLEMENTAÇÃO DO PROTOCOLO MQTT PARA DIGITALIZAÇÃO DA COLETA MANUAL EM POSTOS DE MEDIÇÃO INDUSTRIAL

Sabará - MG
2026

HEBERT EMMANUEL ROCHA PELUSO

IMPLEMENTAÇÃO DO PROTOCOLO MQTT PARA DIGITALIZAÇÃO DA COLETA MANUAL EM POSTOS DE MEDIÇÃO INDUSTRIAL

Trabalho de conclusão de curso apresentado ao Curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *Campus* Sabará para a obtenção do título de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Diego Oliveira Miranda

Sabará - MG
2026

Peluso, Hebert Emmanuel Rocha

P393i Implementação do Protocolo MQTT para digitalização da coleta manual em postos de medição industrial [manuscrito]. / Hebert Emmanuel Rocha Peluso. - 2026.

116 f. : il.

Orientadores: Prof. Dr. Diego Oliveira Miranda.

Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) – Instituto Federal de Minas Gerais, *Campus* Sabará.

1. Internet das coisas. – Monografia. 2. Microcontroladores. – Monografia. 3. Conectividade (Computadores). – Monografia. 4. Controle de processo - Métodos estatísticos. – Monografia. 5. Automação industrial. – Monografia. I. Miranda, Diego Oliveira. II. Instituto Federal de Minas Gerais, *Campus* Sabará. Bacharelado em Engenharia de Controle e Automação. III. Título.

CDU 681.5

César dos Santos Moreira / CRB6-2229
Biblioteca do IFMG *Campus* Sabará

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
Campus Sabará
Diretoria de Ensino, Pesquisa e Extensão
Conselho de Área - Controle e Processos Industriais
Rodovia MGC 262, Km 10 - Bairro Sobradinho - CEP 34590-390 - Sabará - MG - www.ifmg.edu.br

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Ao dia 13 do mês de julho do ano de 2026, às 15:00 horas, sob a presidência de Diego Oliveira Miranda, o discente **Hebert Emmanuel Rocha Peluso** do Curso de Engenharia de Controle e Automação, R.A nº 0050042 do IFMG *campus* Sabará, defendeu o Trabalho de Conclusão de Curso (TCC) intitulado "**IMPLEMENTAÇÃO DO PROTOCOLO MQTT PARA DIGITALIZAÇÃO DA COLETA MANUAL EM POSTOS DE MEDIÇÃO INDUSTRIAL**" e foi avaliado com a nota final de 75 pontos (de um total de 80), que está condicionada ao cumprimento dos procedimentos pós-defesa do TCC.

Compuseram a Banca Examinadora:
Membro 1: Dr. Diego Oliveira Miranda -IFMG *campus* Sabará (orientador),
Membro 2: Dr. Rodrigo Hiroshi Murofushi -IFMG *campus* Sabará
Membro 3: Dr. Daniel Neves Rocha - IFMG *campus* Sabará.

O discente deverá apresentar o trabalho com as devidas modificações em formato pdf, até 27/07/2026 à Coordenação de TCC e fazer o depósito no repositório institucional de TCC do IFMG. O não cumprimento dos procedimentos pós-defesa de TCC até a data estipulada implica no não cumprimento das horas referentes aos componentes curriculares de TCC I e II.

A sessão foi encerrada às 16:30 . Para constar, eu, Diego Oliveira Miranda redigi a presente ata que após lida publicamente, foi aprovada e assinada pelos membros da banca examinadora.

Sabará, 15 de julho de 2026.

Documento assinado eletronicamente por **Diego Oliveira Miranda**, **Professor EBTT**, em 15/07/2026, às 07:43, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Rodrigo Hiroshi Murofushi**, **Professor EBTT**, em 15/07/2026, às 11:32, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Daniel Neves Rocha**, **Professor EBTT**, em 15/07/2026, às 13:51, conforme Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador **2809151** e o código CRC **3E80CD58**.

23714.000993/2026-16	2809151v1

Dedico esta monografia aos meus amados pais, maiores incentivadores e fontes inesgotáveis de apoio, amor e compreensão.

AGRADECIMENTOS

Agradeço a toda à minha família, meus pais e meus amigos por acreditarem em mim e pelo incentivo constante na realização deste trabalho.

Agradeço ao meu orientador, coorientador e a todos que contribuíram de alguma forma para a realização deste trabalho.

“A melhor maneira de prever o futuro é inventá-lo."

Alan Kay

RESUMO

A Indústria 4.0 impulsiona a digitalização do controle de qualidade, mas, em muitos postos de medição industrial, a coleta de dados ainda é feita manualmente, com registro em papel, prática sujeita a erros de transcrição, baixa rastreabilidade e pouca visibilidade do processo. Este trabalho apresenta uma solução de Internet das Coisas (IoT), de baixo custo e orientada a eventos, para a digitalização dessa coleta. O sistema integra um microcontrolador ESP32, acoplado a um transdutor de deslocamento, que captura as medições mediante acionamento do operador, consolida-as estatisticamente e as transmite por meio do protocolo MQTT sobre TLS a um *broker* HiveMQ Cloud. Um assinante desenvolvido em Python valida e persiste os dados em um banco de dados PostgreSQL gerenciado na nuvem (Supabase), enquanto um painel construído em Streamlit apresenta gráficos de controle de Shewhart e os índices de capacidade do processo. A resiliência a falhas de conectividade foi implementada em duas camadas complementares (o armazenamento e encaminhamento em memória não volátil no próprio dispositivo e a sessão persistente do *broker*) e validada experimentalmente, com recuperação integral das medições após quedas de rede e de energia. A caracterização da via de comunicação MQTT indicou latência de ida e volta (*round-trip*) com mediana de aproximadamente 572 ms, com entrega confiável das mensagens e sem perda ou duplicação de dados. Os resultados demonstram a viabilidade técnica e econômica de substituir o registro manual em papel por uma cadeia automatizada de aquisição, transmissão e análise estatística de processo, a um custo de noventa e quatro reais em componentes.

Palavras-chave: Internet das Coisas; MQTT; Controle Estatístico de Processo; ESP32; Indústria 4.0.

ABSTRACT

Industry 4.0 drives the digitalization of quality control, yet in many industrial measurement stations data collection is still performed manually with paper records, a practice prone to transcription errors, low traceability, and poor process visibility. This work presents a low-cost, event-driven Internet of Things (IoT) solution for digitalizing this collection. The system integrates an ESP32 microcontroller, coupled to a displacement transducer, which captures measurements upon operator actuation, consolidates them statistically, and transmits them via the MQTT protocol over TLS to a HiveMQ Cloud broker. A Python subscriber validates and persists the data in a cloud-managed PostgreSQL database (Supabase), while a Streamlit dashboard displays Shewhart control charts and process capability indices. Resilience to connectivity failures was implemented in two complementary layers (store-and-forward in non-volatile memory on the device itself and the broker's persistent session) and experimentally validated, achieving full recovery of measurements after network and power outages. Characterization of the MQTT communication path indicated a median round-trip latency of approximately 572 ms, with reliable message delivery and no data loss or duplication. The results demonstrate the technical and economic feasibility of replacing manual paper recording with an automated chain of acquisition, transmission, and statistical process analysis, at a component cost of ninety-four reais.

Keywords: Internet of Things; MQTT; Statistical Process Control; ESP32; Industry 4.0.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade (*Atomicity, Consistency, Isolation, Durability*)
ADC	Conversor Analógico-Digital (*Analog-to-Digital Converter*)
AMQP	*Advanced Message Queuing Protocol*
API	Interface de Programação de Aplicações (*Application Programming Interface*)
AWS	*Amazon Web Services*
BaaS	*Backend* como Serviço (*Backend as a Service*)
CAD	Projeto Assistido por Computador (*Computer-Aided Design*)
CEP	Controle Estatístico de Processo
CoAP	*Constrained Application Protocol*
CPS	Sistemas Ciberfísicos (*Cyber-Physical Systems*)
CSV	Valores Separados por Vírgula (*Comma-Separated Values*)
DBaaS	Banco de Dados como Serviço (*Database as a Service*)
FIFO	Primeiro a Entrar, Primeiro a Sair (*First In, First Out*)
FSM	Máquina de Estados Finitos (*Finite State Machine*)
GND	Referência de tensão / Terra (*Ground*)
GPIO	Entrada/Saída de Uso Geral (*General Purpose Input/Output*)
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*)
GraphQL	*Graph Query Language*
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
I2C	*Inter-Integrated Circuit*
IoS	Internet dos Serviços (*Internet of Services*)
IoT	Internet das Coisas (*Internet of Things*)
IP	Protocolo de Internet (*Internet Protocol*)
IV-AQMS	*In-Vehicle Air Quality Monitoring System*

JSON	*JavaScript Object Notation*
KPI	Indicador-Chave de Desempenho (*Key Performance Indicator*)
LED	Diodo Emissor de Luz (*Light Emitting Diode*)
M2M	Máquina a Máquina (*Machine-to-Machine*)
MQTT	*Message Queuing Telemetry Transport*
NoSQL	*Not only SQL*
NVS	Armazenamento Não Volátil (*Non-Volatile Storage*)
OASIS	*Organization for the Advancement of Structured Information Standards*
PWM	Modulação por Largura de Pulso (*Pulse Width Modulation*)
QoS	Qualidade de Serviço (*Quality of Service*)
RAM	Memória de Acesso Aleatório (*Random Access Memory*)
RC	Resistor-Capacitor
RCP	Razão de Capacidade do Processo
REST	*Representational State Transfer*
RGB	Vermelho, Verde e Azul (*Red, Green, Blue*)
RLS	Segurança em Nível de Linha (*Row Level Security*)
RTT	Tempo de Ida e Volta (*Round-Trip Time*)
SCADA	*Supervisory Control and Data Acquisition*
SD	*Secure Digital*
SDK	Kit de Desenvolvimento de Software (*Software Development Kit*)
SoC	Sistema em um Chip (*System on a Chip*)
SPI	*Serial Peripheral Interface*
SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
SSL	*Secure Sockets Layer*
TCP	Protocolo de Controle de Transmissão (*Transmission Control Protocol*)
TLS	Segurança da Camada de Transporte (*Transport Layer Security*)

TOML	*Tom's Obvious, Minimal Language*
UART	Receptor/Transmissor Assíncrono Universal (*Universal Asynchronous Receiver/Transmitter*)
UFU	Universidade Federal de Uberlândia
USB	*Universal Serial Bus*
VCC	Tensão de Alimentação
WebGL	*Web Graphics Library*
Wi-Fi	*Wireless Fidelity*

LISTA DE SÍMBOLOS

C_p	Índice de capacidade potencial do processo
C_{pk}	Índice de capacidade efetiva (centrada) do processo
LSE	Limite superior de especificação
LIE	Limite inferior de especificação
LSC	Limite superior de controle
LIC	Limite inferior de controle
$\bar{x}$	Média amostral das medições
μ	Média do processo
σ	Desvio-padrão do processo
V_o	Tensão de saída do potenciômetro
V_s	Tensão de alimentação do potenciômetro
Ω	Ohm — unidade de resistência elétrica

SUMÁRIO

1 INTRODUÇÃO

A Indústria 4.0, ou Quarta Revolução Industrial, representa uma transformação profunda nos sistemas de produção, marcada pela integração de tecnologias digitais como a Internet das Coisas (IoT), a computação em nuvem, a análise de grandes volumes de dados e a inteligência artificial. Essas tecnologias vêm convertendo as fábricas tradicionais em ambientes automatizados e interconectados, em que máquinas e sistemas se comunicam entre si e tomam decisões em tempo real para otimizar a produção (SCHWAB, 2017). A conectividade e a automação trazidas pela Indústria 4.0 permitem uma produção mais flexível e capaz de se adaptar às demandas do mercado.

O controle de qualidade, peça fundamental em qualquer processo produtivo, também se beneficia dessas inovações. Métodos como o *Six Sigma*, tradicionalmente aplicados para reduzir a variabilidade e eliminar defeitos na fabricação, ganham novas ferramentas com a integração às tecnologias digitais. Ao focar na melhoria contínua dos processos por meio de técnicas estatísticas, o *Six Sigma* se torna ainda mais eficaz quando aliado aos sistemas de monitoramento e controle em tempo real proporcionados pela IoT (MONTGOMERY, 2012).

O uso de dispositivos de medição precisos é essencial para a aplicação eficaz de métodos como o *Six Sigma*. Esses dispositivos permitem medir com exatidão as dimensões físicas que fundamentam o controle de qualidade na manufatura. Com a crescente demanda por automação e conectividade na Indústria 4.0, surge a necessidade de integrar esses instrumentos a sistemas de controle de qualidade em tempo real. Nesse contexto, o protocolo MQTT se apresenta como uma solução de baixa sobrecarga para a comunicação entre os dispositivos de medição e os sistemas de controle centralizados (CIRANI *et al.*, 2018).

O MQTT é um protocolo de comunicação leve, projetado para consumir pouca largura de banda e poucos recursos, o que o torna adequado a ambientes industriais com dispositivos IoT. Por transmitir dados em tempo real, mantém as medições feitas por instrumentos digitais rapidamente disponíveis para análise e tomada de decisão (SILVA, 2019). Sua adoção em instrumentos de medição digitais aproxima a coleta da análise de qualidade e agiliza a resposta às variações de processo, em linha com os princípios de melhoria contínua do *Six Sigma*.

Diante desse cenário, este trabalho investiga a seguinte questão de pesquisa: é viável digitalizar a coleta manual de dados em postos de medição industrial, de forma resiliente e a baixo custo, por meio de uma arquitetura IoT orientada a eventos que assegure a integridade e a rastreabilidade das medições e as entregue, sem pré-processamento, a uma cadeia de análise estatística de processo? As seções seguintes desdobram essa questão nos objetivos geral e específicos que a delimitam.

1.1 Objetivos

1.1.1 *Objetivo geral*

O objetivo geral deste trabalho é desenvolver uma solução IoT orientada a eventos e de baixo custo para a digitalização da coleta de dados em postos de medição manual, integrando um microcontrolador ESP32 a um transdutor de deslocamento, com transmissão via MQTT e processamento estatístico em nuvem.

1.1.2 *Objetivos específicos*

- Desenvolver o *firmware* orientado a eventos para o ESP32, com máquina de estados para captura acionada pelo operador, confirmação por duplo-clique e sinalização visual por LEDs.
- Implementar a comunicação do microcontrolador com a nuvem, utilizando o protocolo MQTT sobre TLS para a transmissão segura dos eventos de medição.
- Implementar a primeira camada de resiliência, no próprio dispositivo, por meio do mecanismo de armazenamento e encaminhamento em memória não volátil, garantindo a preservação das medições durante períodos de indisponibilidade da rede.
- Implementar a segunda camada de resiliência, no lado da nuvem, por meio da sessão persistente do *broker* MQTT, de modo que as mensagens publicadas sejam retidas e entregues ao assinante após eventuais períodos de indisponibilidade do consumidor.
- Desenvolver *scripts* em Python para consumir, validar e persistir as medições no banco de dados em nuvem Supabase (PostgreSQL gerenciado).
- Demonstrar que as medições digitalizadas pelo sistema alimentam diretamente, sem etapa de pré-processamento adicional, uma cadeia de análise de capacidade e cartas de controle de Shewhart, com cálculo automatizado dos índices C_p/C_{pk} a partir dos dados persistidos em nuvem.
- Caracterizar o desempenho operacional do sistema por meio de métricas quantitativas (latência, estabilidade e taxa de recuperação após falhas).

1.2 Justificativa

A escolha do tema deste trabalho de conclusão de curso vem da experiência profissional anterior do autor em empresas nas quais o controle de dados era feito manualmente. Em várias delas, o processo de medição e verificação da qualidade dos produtos usava instrumentos de medição digitais cujos resultados eram anotados em folhas de papel. Embora seja uma prática ainda muito comum, ela apresenta limitações que justificam a busca por uma alternativa automatizada.

O método manual de controle de qualidade é propenso a uma série de problemas que comprometem a fidelidade e a integridade dos dados. As anotações manuais estão sujeitas a erros humanos, como falhas na transcrição dos resultados ou na leitura dos valores, o que pode gerar informações incorretas. Esse tipo de erro tem impacto direto na qualidade dos produtos, já que decisões importantes são tomadas com base em dados que podem não refletir a realidade.

Além disso, o registro de medições em papel oferece pouca visibilidade sobre o desempenho dos processos de produção. A análise dos dados se torna difícil e demorada, o que atrasa a identificação de problemas e a execução de ações corretivas. A falta de uma abordagem sistemática e automatizada de monitoramento também reduz a transparência e dificulta a manutenção de padrões de qualidade consistentes.

Outro problema relevante é a possibilidade de manipulação ou ocultação de falhas pelos operadores. Em um sistema manual, não há mecanismo eficaz para garantir que todas as medições sejam registradas com precisão e integridade, o que pode resultar na aceitação de produtos com defeito, prejudicando a reputação da empresa e a satisfação dos clientes.

Diante desses desafios, é necessário buscar uma solução que permita um controle mais preciso e transparente dos dados de medição. A implementação de um sistema automatizado é uma oportunidade de superar as limitações do método manual e alcançar um padrão mais alto de controle de qualidade.

Essa necessidade de melhoria é o que motiva a escolha deste tema. Ao tratar dessas questões, pretende-se contribuir para o avanço das práticas de controle de qualidade e para um ambiente de produção mais confiável e transparente.

1.3 Organização do Texto

Este trabalho está organizado da seguinte forma:

(a) Introdução

(b) Revisão Bibliográfica

(c) Fundamentação Teórica

(d) Metodologia

(e) Resultados

(f) Conclusão

(g) Referências

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta o estado da arte em arquiteturas IoT aplicadas a ambientes industriais, organizado em torno de três eixos centrais para o trabalho: (i) a evolução conceitual da Internet das Coisas e seu enquadramento na Indústria 4.0; (ii) os protocolos de comunicação adotados em sistemas IoT, com ênfase no MQTT; e (iii) a aplicação do Controle Estatístico de Processo em ambientes digitalizados. Ao final, aponta-se a lacuna de pesquisa e o posicionamento deste trabalho em relação à literatura.

2.1 Internet das Coisas e Indústria 4.0

A crescente necessidade de flexibilidade, eficiência no uso de recursos e redução do tempo de lançamento de produtos no mercado tem levado as empresas industriais a buscar soluções de informação e decisão mais adequadas (LASI, 2013). Para isso, a literatura propõe diversas abordagens voltadas a otimizar a aquisição e a transmissão de dados em ambientes industriais e urbanos, com atenção especial à implementação de protocolos de comunicação eficientes em instrumentos de medição.

Historicamente, o conceito da IoT remonta a 1982, quando uma máquina de Coca-Cola modificada foi conectada à internet e passou a informar quais bebidas continha e se estavam geladas (NIMODIYA; AJANKAR, 2022). Anos depois, em 1991, Mark Weiser apresentou uma visão próxima da concepção atual de IoT, sob a forma da computação ubíqua (ALI; ALI; BADAWY, 2015). O termo *Internet of Things*, porém, só foi proposto por Kevin Ashton em 1999, para descrever um sistema de dispositivos interconectados, e acabou se consolidando como o paradigma que sustentaria a Quarta Revolução Industrial.

No meio acadêmico brasileiro, (LIMA *et al.*, 2023) conduziram uma revisão bibliométrica sobre IoT e Indústria 4.0. O estudo cobriu publicações de 2014 a 2022 e apontou que, embora a IoT traga ganhos de produtividade, qualidade e segurança, ainda persistem desafios tecnológicos e econômicos para sua adoção em larga escala. No campo das tecnologias habilitadoras, (HERMANN; PENTEK; OTTO, 2016) mostraram que os sistemas ciberfísicos (*Cyber-Physical Systems* – CPS), a IoT, a internet de serviços (*Internet of Services* – IoS) e as fábricas inteligentes formam os pilares da Indústria 4.0.

2.2 Protocolos de Comunicação para IoT Industrial

A escolha do protocolo de comunicação é uma decisão crítica em arquiteturas IoT, pois afeta diretamente o consumo de banda, a latência e o gasto de energia dos dispositivos. (NAIK, 2017) comparou os principais protocolos da camada de aplicação, entre eles o MQTT, o CoAP, o AMQP e o HTTP, e observou que cada um tem características próprias que o tornam mais adequado a determinados cenários. Para o autor, o MQTT, baseado no modelo de publicação

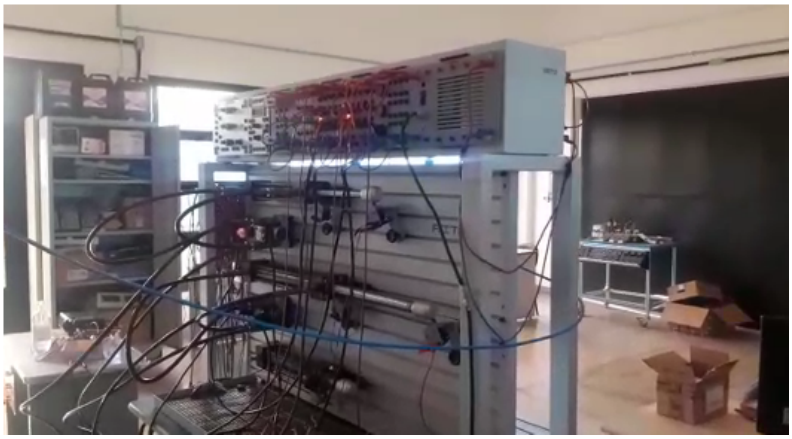
e assinatura (*publish/subscribe*) sobre TCP, leva vantagem em aplicações que exigem baixa sobrecarga de comunicação, alta confiabilidade e desacoplamento entre quem produz e quem consome os dados.

Em um estudo empírico voltado a um sistema de monitoramento da qualidade do ar veicular (IV-AQMS), (GOH *et al.*, 2019) avaliaram os protocolos HTTP e MQTT em um teste real com 40 ônibus na Malásia. O MQTT teve desempenho bem superior: a publicação de uma nova mensagem levou, em média, 3 segundos, contra 28 segundos do HTTP, um envio cerca de dez vezes mais rápido. Apesar de uma taxa de perda de pacotes um pouco maior em cenários de mobilidade, o MQTT manteve baixa variação no atraso de entrega e foi considerado mais eficiente para sistemas de monitoramento de baixo custo.

2.2.1 Aplicações Práticas do MQTT em Sistemas Industriais

Exemplos práticos confirmam a aplicabilidade do MQTT em ambientes industriais. (ALVES, 2022), em Trabalho de Conclusão de Curso na UFU intitulado "*Framework* para controle de sistemas industriais via MQTT", relatou testes bem-sucedidos em bancadas eletro-hidráulicas e pneumáticas, validando o processamento na borda e o envio de dados críticos para servidores na nuvem.

Figura 1 – Bancada eletro-hidráulica utilizada nos testes da solução via MQTT.



Fonte: ALVES, 2022.

No contexto agrícola, (GRGIć; ŠPEH; HEđI, 2016) desenvolveram uma solução IoT baseada em MQTT para acompanhar processos de secagem agrícola pela internet. O objetivo era acabar com o monitoramento manual de temperatura e umidade, que precisava ser feito 24 horas por dia em instalações rurais com conexão de internet limitada e instável. Os resultados confirmaram a robustez do protocolo em redes com conectividade intermitente.

Em uma aplicação muito próxima da deste trabalho, (AGHENTA; IQBAL, 2020) projetaram e implementaram um sistema SCADA (*Supervisory Control and Data Acquisition*) de baixo custo e código aberto, baseado no microcontrolador ESP32, na plataforma ThingsBoard e

no protocolo MQTT. O sistema foi validado no monitoramento de um painel solar fotovoltaico e comprovou a viabilidade técnica e econômica de combinar ESP32, MQTT e painel em nuvem como alternativa às soluções SCADA comerciais. O caso ilustra uma tendência recente da literatura: a construção de arquiteturas IoT industriais com componentes de custo muito baixo, sem abrir mão de funcionalidade.

2.2.2 Visualização e Painéis em Tempo Real

A camada de visualização liga os dados coletados à tomada de decisão. (HWANG; HWANG, 2018) apresentaram o desenvolvimento de um *software* de painel em tempo real para sistemas IoT baseados em MQTT, com o objetivo de ajudar as organizações a aproveitar melhor o volume crescente de dados gerados por dispositivos conectados. A ideia central é que, ao reunir ferramentas de análise de grandes volumes de dados e processamento em tempo real, as empresas conseguem decidir com mais agilidade e embasamento, convertendo dados brutos em informações úteis para o acompanhamento dos principais indicadores e a identificação de tendências.

Figura 2 – Painéis em tempo real para sistemas IoT baseados em MQTT.

그림 7. 테스트 IoT 시스템을 표현한 전체 시스템

Fonte: HWANG; HWANG, 2018.

2.3 Arquiteturas Orientadas a Eventos e Resilientes

Além dos trabalhos voltados à telemetria contínua, a literatura mais recente vem destacando a importância de arquiteturas IoT orientadas a eventos para aplicações industriais. Nesse modelo, a coleta de dados é disparada por ações específicas, como o comando de um operador ou a ocorrência de um evento de processo, e não por transmissões periódicas. Essa abordagem reduz

o tráfego de rede, simplifica a semântica das mensagens e se ajusta bem a fluxos de inspeção e controle de qualidade, em que cada medição corresponde a um evento isolado associado a uma peça ou lote (CIRANI *et al.*, 2018).

Outro ponto recorrente em arquiteturas IoT industriais é a necessidade de tolerância a falhas de conectividade, que em ambientes fabris reais são requisitos obrigatórios, e não exceções. O padrão de armazenamento e encaminhamento (*store-and-forward*) prevê que os dispositivos de borda guardem localmente as mensagens não entregues e as transmitam automaticamente assim que a rede se restabelece. Essa estratégia é útil sobretudo em postos de medição móveis ou em plantas com cobertura de rede irregular, pois preserva a integridade dos dados mesmo quando a infraestrutura de comunicação oscila.

2.4 Controle Estatístico de Processo na Indústria 4.0

A digitalização da coleta de dados no chão de fábrica vem transformando o Controle Estatístico de Processo (CEP) de uma prática reativa, apoiada em inspeção manual e registro em papel, para uma prática proativa, alimentada por dados coletados continuamente por sensores conectados. Como discutem (WOLNIAK; GREBSKI, 2024), a união entre CEP e Indústria 4.0 dá origem ao conceito de Qualidade 4.0, em que sensores IoT alimentam plataformas de análise em nuvem e permitem detectar variações de processo de imediato, calcular automaticamente índices de capacidade (C_p, C_{pk}) e aplicar regras estatísticas, como as de *Western Electric*, sem intervenção humana. Os autores ressaltam que essa integração encurta bastante o tempo entre a detecção de uma anomalia e a ação corretiva, o que aumenta a confiabilidade dos processos de qualidade.

Os trabalhos reunidos nesta revisão, ainda que demonstrem maturidade tecnológica em pontos específicos, como comunicação, aquisição e visualização, raramente apresentam o caminho completo, da medição manual digitalizada até o indicador estatístico de capacidade, com rastreabilidade fim-a-fim por evento. É essa a lacuna que este trabalho busca preencher.

2.5 Posicionamento do Presente Trabalho

A análise dos trabalhos consultados nesta revisão revela três tendências bem estabelecidas: (i) a adoção do MQTT como padrão de fato em arquiteturas IoT industriais, graças à sua eficiência em redes restritas; (ii) a viabilidade técnica e econômica de plataformas de baixo custo, em especial o microcontrolador ESP32, como elementos de borda em sistemas industriais; e (iii) o reconhecimento, ainda incipiente, da necessidade de arquiteturas resilientes e orientadas a eventos para ambientes fabris reais. A Tabela 1 reúne os principais trabalhos analisados.

Tabela 1 – Síntese comparativa dos trabalhos correlatos analisados.

Trabalho	**Aplicação**	**Hardware**	**Protocolo**	**Persistência**	**CEP**
ALVES, 2022	Bancada eletro-hidráulica	—	MQTT	Nuvem genérica	Não
GRGIć; ŠPEH; HEđI, 2016	Secagem agrícola	—	MQTT	Servidor próprio	Não
GOH *et al.*, 2019	Qualidade do ar veicular	—	HTTP/MQTT	—	Não
HWANG; HWANG, 2018	Painel IoT	—	MQTT	Grandes volumes	Não
AGHENTA; IQBAL, 2020	SCADA fotovoltaico	ESP32	MQTT	ThingsBoard	Não
Este trabalho	**Digitalização de coleta manual**	**ESP32**	**MQTT/TLS**	**Supabase**	**Sim**

A tabela apresentada indica que o presente trabalho ocupa um nicho ainda pouco explorado pela literatura: a digitalização orientada a eventos da coleta manual feita por operadores nos postos de medição, reunindo, em uma única arquitetura de referência, comunicação MQTT/TLS, persistência em banco SQL na nuvem, resiliência por armazenamento e encaminhamento local e análise estatística automatizada da capacidade de processo. A contribuição original, portanto, não está em nenhum dos elementos isolados, todos já consolidados na literatura, mas na forma como eles se articulam em uma solução fim-a-fim, de baixo custo, voltada a substituir o registro manual em papel em ambientes industriais.

3 FUNDAMENTAÇÃO TEÓRICA

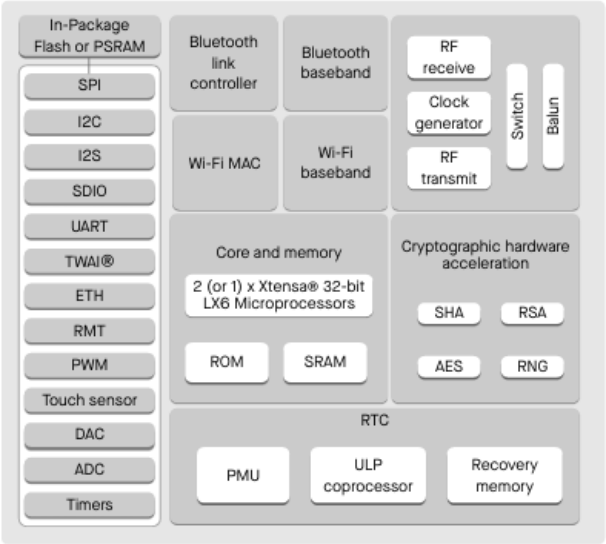
3.1 Microcontrolador ESP32

O ESP32 é uma plataforma de prototipagem eletrônica de baixo custo e alto desempenho, desenvolvida pela Espressif Systems. Seu principal componente é um microcontrolador do tipo SoC, que integra em um único chip um processador de dois núcleos, conectividade Wi-Fi e Bluetooth, o que o torna adequado ao desenvolvimento de projetos, sobretudo na área de Internet das Coisas e sistemas embarcados (Espressif Systems, 2025). Na prática, o ESP32 funciona como o elemento central do sistema ligado a ele, gerenciando desde sensores e atuadores até a comunicação sem fio. A plataforma é baseada em *hardware* e software de código aberto, o que ajudou a popularizá-la e a reunir uma grande comunidade de desenvolvedores em todo o mundo (KOLBAN, 2018).

O que torna o ESP32 atrativo para muitos projetos é a combinação de versatilidade e bom custo-benefício, que permite desenvolver aplicações complexas com um investimento acessível. A compatibilidade com o ambiente de desenvolvimento do Arduino, por exemplo, facilita o uso por quem não é da área técnica e torna o aprendizado rápido e prático.

O ESP32 conta com um conjunto amplo de periféricos, acessíveis por seus pinos de entrada e saída de uso geral (GPIO). As portas digitais podem operar como entradas ou saídas de dados binários (0 ou 1). Muitas delas também oferecem a função de modulação por largura de pulso (PWM), que permite controlar a intensidade de atuadores, como o brilho de LEDs ou a velocidade de motores. O ESP32 dispõe ainda de entradas analógicas, que leem sinais variáveis vindos de sensores e convertem a tensão em um valor digital para uma medição precisa. Há também portas de comunicação dedicadas, essenciais em projetos conectados: a interface serial (UART), para comunicação com computadores e outros dispositivos; o barramento I2C, que conecta vários sensores e displays usando poucos fios; e o SPI, protocolo de alta velocidade usado para ligar periféricos como leitores de cartão SD e módulos de comunicação mais avançados (Espressif Systems, 2025). A Figura 3 apresenta o diagrama de blocos funcional do ESP32, no qual se observam os blocos de rádio para Wi-Fi e Bluetooth, o núcleo de processamento com memória integrada, os periféricos de entrada e saída e o coprocessador de baixo consumo.

Figura 3 – Diagrama de blocos funcional do ESP32.



Fonte: Espressif Systems, 2025.

3.2 Protocolo MQTT

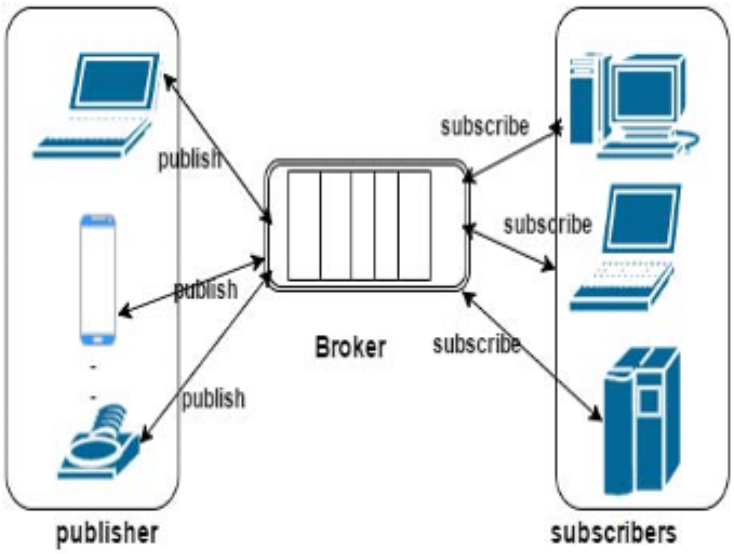
O protocolo MQTT é um dos mais populares da camada de aplicação para a Internet das Coisas, introduzido em 1999 por Andy Stanford-Clark e Arlen Nipper e padronizado pela OASIS em 2013 (YASSEIN *et al.*, 2017).

O MQTT é um protocolo voltado a mensagens, baseado em um modelo de publicação e assinatura. Ele funciona sobre a camada de transporte TCP e tem como objetivo conectar com eficiência dispositivos, redes e aplicações, o que o torna adequado ao universo da Internet das Coisas e à comunicação máquina a máquina (M2M). Sua concepção o torna apropriado para ambientes com dispositivos de baixa potência, pouca memória e recursos limitados, que operam em redes de baixa largura de banda ou instáveis (AL-FUQAHA *et al.*, 2015; GUPTA; M, 2021; KHALIL *et al.*, 2020).

A comunicação no MQTT se organiza em torno de três elementos: o publicador, o assinante e o broker. O broker é um servidor central que recebe as mensagens dos publicadores e as distribui aos assinantes interessados. A troca não ocorre diretamente entre os dispositivos: o assinante manifesta interesse ao se inscrever em um tópico específico no broker e, quando um publicador envia uma mensagem para esse mesmo tópico, o broker se encarrega de encaminhá-la a todos os assinantes inscritos nele (KHALIL *et al.*, 2020; AL-FUQAHA *et al.*, 2015; GUPTA; M, 2021).

Como o MQTT não oferece criptografia nativa, é prática comum operá-lo sobre uma camada de transporte segura, em geral o TLS (RESCORLA, 2018), que garante a confidencialidade e a integridade das mensagens em trânsito.

Figura 4 – Arquitetura do MQTT.



Fonte: YASSEIN *et al.*, 2017.

3.2.1 Estrutura das Mensagens MQTT

A eficiência do MQTT em redes de baixa largura de banda decorre, em grande parte, do formato enxuto de suas mensagens. Toda a comunicação do protocolo é realizada por meio de pacotes de controle, dos quais a especificação define quatorze tipos, entre eles `CONNECT` (estabelecimento da conexão), `PUBLISH` (publicação de mensagem), `PUBACK` (confirmação de recebimento) e `SUBSCRIBE` (inscrição em tópico).

Cada pacote é composto por até três partes. A primeira é o cabeçalho fixo, presente em todos os pacotes e com tamanho mínimo de dois *bytes*: o primeiro *byte* identifica o tipo do pacote de controle e quatro *flags* de controle (entre elas o nível de QoS e o atributo de retenção), enquanto o campo seguinte, denominado *remaining length*, indica o tamanho do restante do pacote por meio de uma codificação de comprimento variável de 1 a 4 *bytes*. A segunda parte é o cabeçalho variável, presente apenas em alguns tipos de pacote, que transporta informações como o identificador da mensagem. A terceira é a carga útil (*payload*) que, em um pacote `PUBLISH`, contém o nome do tópico e o conteúdo efetivo da mensagem.

Esse formato compacto, em que o *overhead* de protocolo de uma publicação pode se resumir a poucos *bytes* além da carga útil, é o que torna o MQTT especialmente adequado a dispositivos de borda com recursos limitados e a redes instáveis, em contraste com protocolos baseados em texto e cabeçalhos extensos, como o HTTP (YASSEIN *et al.*, 2017).

3.2.2 Níveis de Qualidade de Serviço (QoS)

Para garantir a entrega das mensagens em diferentes cenários de confiabilidade, o MQTT define três níveis de qualidade de serviço:

- **Nível 0**: a mensagem é enviada uma única vez, sem exigir confirmação de recebimento pelo destino (GUPTA; M, 2021).
- **Nível 1**: a entrega é garantida pelo menos uma vez. O sistema aguarda uma confirmação e, se ela não chega no tempo previsto, o broker reenvia a mensagem (GUPTA; M, 2021).
- **Nível 2**: oferece o maior nível de confiabilidade e assegura que a mensagem chegue exatamente uma vez, evitando o recebimento de duplicatas (GUPTA; M, 2021).

A escolha do nível de QoS em uma arquitetura IoT envolve um equilíbrio entre confiabilidade de entrega, sobrecarga de rede e complexidade de implementação. Os níveis mais altos (1 e 2) garantem a entrega ao broker por meio de confirmação e retransmissão, ao custo de mais pacotes trafegados e de maior consumo de operações nos serviços de broker em nuvem. Em sistemas IoT industriais, costuma-se considerar que a confiabilidade pode ser obtida em diferentes camadas da arquitetura: no nível do protocolo, pelo próprio QoS, ou no nível da aplicação, por meio de recursos como armazenamento local, identificadores únicos por evento e remoção de duplicatas no consumidor. A escolha entre essas abordagens depende dos requisitos do sistema, das características da biblioteca usada e do custo de operação aceitável (AL-FUQAHA *et al.*, 2015).

3.2.3 Hierarquia de Tópicos e Mensagens Retidas

A organização dos tópicos no MQTT é hierárquica e usa barras como separadores, no formato `nivel1/nivel2/nivel3`. Essa estrutura permite que os assinantes usem caracteres curinga para se inscrever em vários tópicos relacionados: o curinga `+` substitui exatamente um nível da hierarquia (por exemplo, `maquinas/+/medicao` recebe as medições de todas as máquinas), enquanto o `#` substitui todos os níveis abaixo da posição em que aparece (por exemplo, `maquinas/#` recebe qualquer tópico abaixo de `maquinas`). Esse recurso é muito útil em arquiteturas industriais escaláveis, em que um único assinante pode reunir dados de dezenas ou centenas de dispositivos sem precisar de assinaturas individuais.

O MQTT também oferece o conceito de mensagens retidas: ao publicar uma mensagem com o atributo de retenção ativado, o broker guarda a última mensagem de cada tópico e a entrega de imediato a qualquer novo assinante, sem que ele precise esperar a próxima publicação. Esse mecanismo é útil para divulgar o estado operacional dos dispositivos, como mensagens periódicas que indicam que um equipamento continua ativo, permitindo que os sistemas de monitoramento saibam o estado atual de cada dispositivo assim que se conectam ao broker.

3.3 Sistemas Orientados a Eventos e Máquinas de Estados Finitos

Em sistemas embarcados que interagem diretamente com operadores ou que respondem a eventos externos imprevisíveis, é comum adotar o paradigma orientado a eventos. Ao contrário

dos sistemas baseados em ciclos periódicos fixos, em que as ações são executadas em intervalos regulares independentemente do contexto, os sistemas orientados a eventos ficam em espera até que algo aconteça, seja uma ação do usuário, uma mudança no estado da rede ou o fim de um temporizador, e só então executam o processamento correspondente. Esse modelo reduz o uso de recursos computacionais, deixa o comportamento mais previsível e combina bem com os fluxos de trabalho industriais, em que ações discretas e identificáveis, como medições, inspeções e validações, são o foco da observação (LEE; SESHIA, 2017).

Na prática, esse paradigma costuma ser implementado em sistemas embarcados por meio de máquinas de estados finitos (FSM), em que o comportamento é modelado como um conjunto finito de estados e um conjunto de transições disparadas por eventos. A cada instante, o sistema está em exatamente um estado, e cada estado define um conjunto bem delimitado de ações permitidas e de eventos aos quais o sistema responde. Essa abordagem traz previsibilidade de comportamento, facilita a depuração, trata explicitamente as condições de erro e exceção e torna a verificação formal do sistema mais simples. Por isso, as máquinas de estados são empregadas em controladores industriais, instrumentação digital e protocolos de comunicação (LEE; SESHIA, 2017).

3.4 Tratamento de Sinais de Botões Mecânicos

O acionamento de botões mecânicos em sistemas digitais provoca um fenômeno conhecido como repique (*bouncing*): os contatos metálicos não se fecham instantaneamente e geram várias transições espúrias de tensão por alguns milissegundos, até o sinal estabilizar. Se não for tratado, esse comportamento faz com que um único toque intencional do operador seja interpretado pelo microcontrolador como vários cliques seguidos, o que prejudica a confiabilidade da interação humano-máquina (GANSSLE, 2008).

O tratamento desse fenômeno, chamado de filtragem de repique (*debouncing*), pode ser feito em *hardware*, normalmente com filtros resistor-capacitor (RC) que atenuam as transições rápidas, ou em software, verificando se o sinal permanece estável por um intervalo mínimo, em geral entre 20 e 100 milissegundos. A solução por software tem como vantagens a flexibilidade, já que o intervalo pode ser ajustado sem trocar componentes físicos, além da economia de *hardware* e da facilidade de manutenção, o que a torna a opção preferida em projetos com microcontroladores (GANSSLE, 2008).

3.5 Persistência Local e o Padrão de Armazenamento e Encaminhamento

Em aplicações IoT industriais, a indisponibilidade temporária da rede deve ser tratada como uma situação esperada de operação, e não como uma exceção rara. Quedas momentâneas do Wi-Fi, congestionamento de redes corporativas, manutenções não programadas e interrupções no serviço de broker em nuvem são eventos recorrentes em ambientes fabris reais. Em sistemas em

que cada mensagem representa um evento crítico, como uma medição associada a uma peça em controle de qualidade, a perda silenciosa de dados nesses períodos é inaceitável, pois compromete a rastreabilidade e a integridade do histórico operacional.

Para contornar esse problema, adota-se o padrão de armazenamento e encaminhamento (*store-and-forward*): o dispositivo de borda guarda localmente as mensagens que não puderam ser entregues no momento em que foram geradas e, ao perceber que a conexão com o servidor foi restabelecida, as transmite automaticamente, mantendo a ordem original no tempo. Essa abordagem transfere a garantia de entrega da camada de rede para a camada de aplicação, tornando o sistema confiável independentemente da qualidade momentânea da rede.

No ESP32, a persistência local pode ser feita com o sistema de armazenamento não volátil (NVS), uma partição da memória *flash* reservada ao armazenamento estruturado de pares chave-valor. Os dados gravados na NVS sobrevivem a desligamentos completos e a reinicializações do dispositivo, o que oferece durabilidade suficiente para implementar as filas locais usadas no armazenamento e encaminhamento. A capacidade depende do tamanho da partição NVS configurada na memória *flash*, e a vida útil é limitada por um número finito de ciclos de escrita por setor, restrições que precisam ser levadas em conta ao dimensionar a fila e definir a frequência de escrita (Espressif Systems, 2025).

3.6 Python

O Python é uma das principais linguagens para a análise de dados moderna, graças ao seu ecossistema de bibliotecas especializadas. A base desse ecossistema é o NumPy, que fornece a infraestrutura para computação numérica de alto desempenho por meio de seus *arrays* multidimensionais (HARRIS *et al.*, 2020). Sobre essa base, o pandas tornou-se uma das ferramentas mais usadas para manipular e estruturar dados, ao introduzir o *DataFrame*, que simplifica tarefas de limpeza, transformação e agregação (MCKINNEY, 2010). A etapa de visualização se apoia na Matplotlib, que dá controle programático para a criação de gráficos de alta qualidade (HUNTER, 2007). Para a visualização interativa em ambiente *web*, recorre-se ainda à biblioteca Plotly, que gera gráficos renderizados no navegador com suporte à interação direta (aproximação, deslocamento e inspeção individual de pontos) e à aceleração gráfica por *WebGL*, recurso adequado à exibição fluida de grandes volumes de medições.

Figura 5 – Exemplo de código em Python.

```python
class linspace(collections.abc.Sequence):
    """linspace(start, stop, num) -> linspace object

    Return a virtual sequence of num numbers from start to stop (inclusive).

    If you need a half-open range, use linspace(start, stop, num+1)[:-1].
    """

    def __init__(self, start, stop, num):
        if not isinstance(num, numbers.Integral) or num <= 1:
            raise ValueError('num must be an integer > 1')
        self.start, self.stop, self.num = start, stop, num
        self.step = (stop-start)/(num-1)
    def __len__(self):
        return self.num
    def __getitem__(self, i):
        if isinstance(i, slice):
            return [self[x] for x in range(*i.indices(len(self)))]
        if i < 0:
            i = self.num + i
        if i >= self.num:
            raise IndexError('linspace object index out of range')
        if i == self.num-1:
            return self.stop
        return self.start + i*self.step
    def __repr__(self):
        return '{}({}, {}, {})'.format(type(self).__name__,
                                       self.start, self.stop, self.num)
    def __eq__(self, other):
        if not isinstance(other, linspace):
            return False
        return ((self.start, self.stop, self.num) ==
                (other.start, other.stop, other.num))
    def __ne__(self, other):
        return not self==other
    def __hash__(self):
        return hash((type(self), self.start, self.stop, self.num))
```

Fonte: wiki.python.org/moin/BeginnersGuide/Examples, 2026.

3.7 Controle Estatístico de Processo

A análise de capacidade de um processo busca estimar o quanto ele atende às especificações ou requisitos definidos para o produto. Costuma ser útil expressar essa capacidade em termos simples e quantitativos. Para isso, usam-se as razões de capacidade do processo (RCPs), sendo as mais comuns C_p e C_{pk}.

A razão da capacidade do processo, C_p, é definida como:

$$C_p = \frac{LSE - LIE}{6\sigma} \tag{3.1}$$

em que LSE é o limite superior de especificação, LIE é o limite inferior de especificação e 6σ representa a dispersão natural do processo. O índice C_p mede a capacidade potencial do processo, ou seja, avalia se a dispersão é pequena o suficiente para caber dentro da faixa de especificação, mas não considera onde está a média do processo, μ. Um valor de $C_p < 1$ indica que o processo não é capaz de atender às especificações, enquanto valores de $C_p > 1$ indicam que a dispersão é menor ou igual à faixa de especificação (MONTGOMERY, 2012).

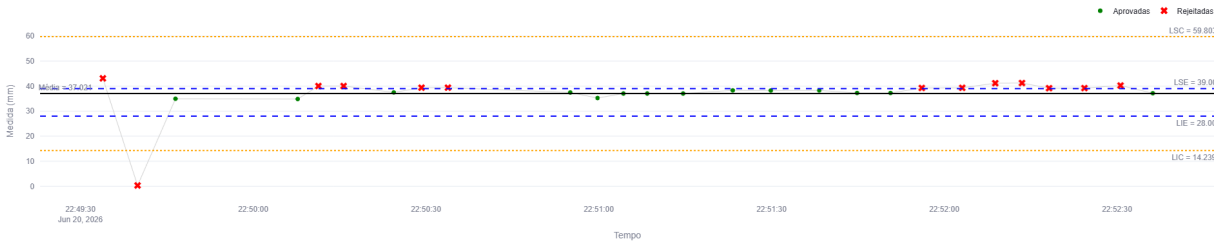
Como o C_p não considera a centralização do processo, foi desenvolvido o índice C_{pk} para medir a capacidade efetiva (MONTGOMERY, 2012). Ele é definido como:

$$C_{pk} = \min\left[\frac{LSE - \mu}{3\sigma}, \frac{\mu - LIE}{3\sigma}\right] \tag{3.2}$$

O índice C_{pk} leva em conta a localização da média do processo, μ, e seu valor é sempre menor ou igual ao de C_p. Os dois índices só serão iguais se o processo estiver perfeitamente

centrado no ponto médio das especificações. Um valor de $C_{pk} < 1$ indica que o processo está produzindo itens não conformes (MONTGOMERY, 2012).

Figura 6 – Gráfico de Shewhart.



Fonte: Adaptado de COSTA; EPPRECHT; CARPINETTI, 2010.

É importante destacar que a interpretação desses índices repousa sobre duas hipóteses fundamentais: que o processo se encontra sob controle estatístico, isto é, livre de causas especiais de variação, e que a característica de qualidade medida segue uma distribuição aproximadamente normal (MONTGOMERY, 2012). A primeira hipótese é condição para que a dispersão estimada represente apenas a variabilidade natural do processo; a segunda sustenta a associação direta entre os valores de C_p e C_{pk} e a fração esperada de itens não conformes, bem como os limiares qualitativos comumente adotados ($C_{pk} \geq 1{,}33$ para processos capazes, por exemplo). Quando essas premissas não se verificam, em processos fora de controle ou com distribuições acentuadamente assimétricas, os índices clássicos perdem o significado probabilístico e exigem métodos alternativos, como índices baseados em percentis ou a transformação prévia dos dados. Por essa razão, a verificação da aderência à normalidade, ainda que visual, precede a leitura quantitativa da capacidade.

3.8 Banco de Dados

Um sistema de gerenciamento de banco de dados pode ser definido como uma coleção de dados inter-relacionados acompanhada de um conjunto de programas para acessá-los, com o objetivo de armazenar e recuperar informações com eficiência e consistência (SILBERSCHATZ; KORTH; SUDARSHAN, 2010).

Historicamente dominados pelo modelo relacional, que organiza os dados em tabelas com linhas e colunas (CODD, 1970), os sistemas de banco de dados evoluíram para atender às demandas das aplicações modernas, sobretudo as ligadas à Internet das Coisas e à computação em nuvem. Nesse contexto, os bancos de dados NoSQL surgiram como uma alternativa que prioriza escalabilidade horizontal, flexibilidade de esquema e alta disponibilidade, características essenciais para sistemas que lidam com fluxos contínuos de dados heterogêneos (SADALAGE; FOWLER, 2012).

Entre os modelos NoSQL, os bancos orientados a documentos se destacam por armazenar os dados em estruturas flexíveis parecidas com o formato JSON, o que dispensa um esquema rígido e predefinido. Por outro lado, o modelo relacional continua sendo o paradigma mais estabelecido para cenários em que a integridade referencial, a tipagem estrita e a consulta declarativa via SQL são requisitos centrais. Essas características têm grande valor em aplicações IoT industriais, nas quais a estrutura dos eventos de medição é estável e bem definida e o consumo dos dados se dá principalmente por consultas analíticas com filtros temporais e agregações estatísticas (SADALAGE; FOWLER, 2012; SILBERSCHATZ; KORTH; SUDARSHAN, 2010).

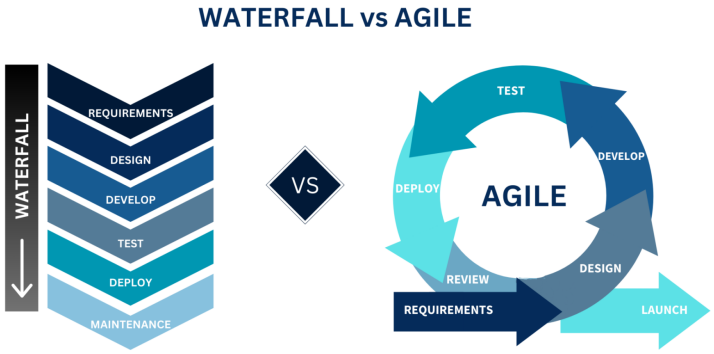
A nuvem possibilitou a oferta de banco de dados como serviço (DBaaS), em que o provedor abstrai a complexidade da infraestrutura de servidores, da replicação, dos *backups* e da disponibilidade, expondo o banco por meio de uma API de alto nível. Essa modalidade reduz bastante os custos de operação e dispensa a manutenção de instâncias dedicadas, o que está alinhado ao critério de baixo custo e acessibilidade que orienta este projeto.

O Supabase, plataforma de banco de dados em nuvem adotada neste trabalho, é um serviço gerenciado construído sobre o PostgreSQL, sistema de gerenciamento de banco de dados relacional distribuído sob licença de código aberto. A plataforma reúne as garantias clássicas do modelo relacional, como propriedades ACID, integridade referencial, tipagem forte e suporte completo à linguagem SQL, e acrescenta recursos modernos típicos de plataformas que oferecem o *backend* como serviço (BaaS), como autenticação integrada, geração automática de APIs REST e GraphQL a partir do esquema do banco e mecanismos de assinatura para notificações em tempo real (Supabase Inc., 2026). O Supabase disponibiliza um *SDK* oficial para Python (`supabase-py`), que se autentica por chave de API e oferece operações de leitura, escrita, atualização e remoção por meio de uma interface fluente. Para o caso de uso de séries temporais de medições industriais, alguns recursos nativos do PostgreSQL se mostram bem adequados, como índices em colunas de data e hora, consultas com cláusulas `WHERE` sobre intervalos de tempo e funções de agregação executadas no próprio servidor. Esses recursos reduzem o volume de dados transferidos ao mínimo necessário e passam o esforço de processamento da aplicação para o banco.

4 METODOLOGIA

A metodologia adotada neste trabalho segue uma abordagem híbrida, que combina o modelo Cascata para o desenvolvimento do *hardware* com uma abordagem ágil para o software. A escolha se justifica pela natureza distinta das tarefas: o desenvolvimento físico do protótipo exige uma sequência linear e bem definida, enquanto o software se beneficia de ciclos de desenvolvimento iterativos e flexíveis.

Figura 7 – Metodologias usadas.



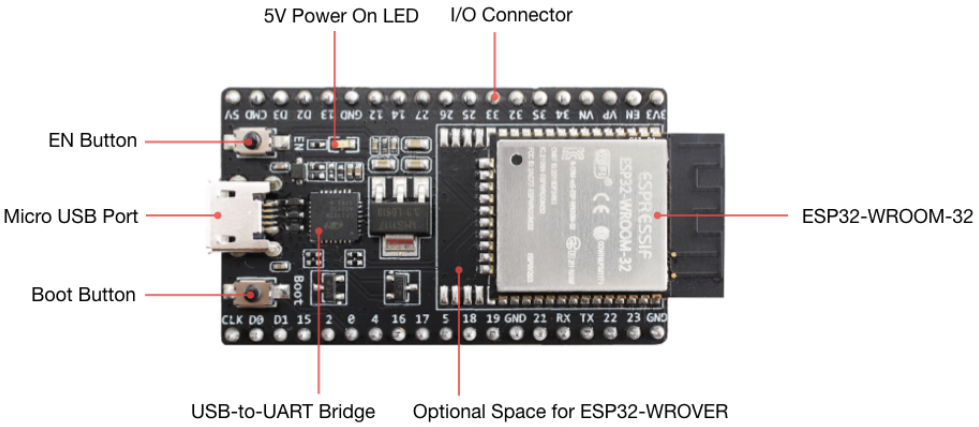
Fonte: www.grupoemprende.mx/distinga-entre-agile-y-waterfall/, 2026.

4.1 Modelo Cascata

A primeira fase do projeto, voltada ao *hardware*, seguiu o modelo Cascata, no qual cada etapa depende da conclusão da anterior. O desenvolvimento se concentra na montagem do conjunto eletrônico que compõe o posto de medição instrumentado, integrando o microcontrolador, o transdutor de deslocamento e os elementos de interação com o operador: o botão de acionamento e três LEDs para sinalização visual.

O microcontrolador adotado é o ESP32 DevKitC, da Espressif Systems. A escolha foi motivada pela combinação de processador de dois núcleos, conectividade Wi-Fi integrada, suporte nativo à comunicação segura via TLS, baixo custo (R$ 36,00) e ampla disponibilidade de bibliotecas para os protocolos necessários ao projeto.

Figura 8 – ESP32 DevKitC.



Fonte: Navegador, 2026.

O transdutor de deslocamento adotado é um potenciômetro linear deslizante de 10 kΩ, com curso útil de aproximadamente 75 mm e custo de R$ 17,00. O componente converte o deslocamento físico do cursor em uma variação de tensão proporcional, lida pelo conversor analógico-digital (ADC) de 12 bits do ESP32. A interface elétrica entre os componentes é feita pelos três terminais do potenciômetro: o pino de alimentação (VCC) vai ao pino de 3,3 V do microcontrolador, o pino de aterramento ao GND e o pino de sinal a uma das portas analógicas do processador (GPIO 34).

Figura 9 – Potenciômetro Linear Deslizante.

Fonte: Navegador, 2026.

Para a interação com o operador, foi previsto um botão de pressão momentâneo, conectado a um pino digital do ESP32 (GPIO 32) configurado com resistor de *pull-up* interno. O sinal do botão passa por uma rotina de filtragem de repique por software, detalhada na seção 4.3. Além disso, três LEDs, ligados aos pinos GPIO 25 (vermelho), GPIO 26 (verde) e GPIO 27 (branco)

por resistores limitadores de corrente de 220 Ω, fornecem ao operador uma sinalização visual contínua do estado atual do sistema.

O foco deste trabalho está na cadeia de aquisição, transmissão e análise dos dados, e não na construção de um instrumento de medição com qualificação metrológica. O potenciômetro deslizante atua como fonte de sinal mensurável, reproduzindo o comportamento de transdutores de deslocamento muito usados em postos de medição industrial.

4.2 Modelo Ágil

A segunda fase do projeto, dedicada ao desenvolvimento do *software*, adota uma abordagem ágil, organizada em ciclos de desenvolvimento iterativos. Essa flexibilidade permite testar e validar cada funcionalidade isoladamente, com revisões frequentes do escopo conforme novos requisitos aparecem.

O primeiro ciclo se concentra na aquisição estável do sinal do transdutor. A leitura do ADC é feita pela função `analogReadMilliVolts()` do *SDK* do ESP32, que aplica internamente a calibração de fábrica do conversor e corrige boa parte da não linearidade intrínseca do ADC do ESP32. Sobre esse sinal já calibrado em tensão, cada captura coleta uma janela de 50 amostras consecutivas, a partir da qual se calculam a média e o desvio-padrão amostral. Essa consolidação estatística por janela atenua o ruído de alta frequência e produz, a cada evento de medição, um único valor representativo do deslocamento e o desvio-padrão associado. Optou-se pela filtragem exclusivamente em *software*, em vez de um filtro analógico passa-baixa do tipo resistor-capacitor (RC), porque ela permite ajustar os parâmetros com mais liberdade e dispensa componentes eletrônicos adicionais.

O segundo ciclo é dedicado à comunicação e à persistência dos dados. O ESP32 atua como publicador MQTT: conecta-se a uma rede Wi-Fi e estabelece comunicação com o broker HiveMQ Cloud por meio de uma conexão TLS (RESCORLA, 2018) na porta 8883, o que garante a confidencialidade e a integridade do canal. As mensagens, estruturadas em JSON, não são publicadas periodicamente, e sim em resposta a eventos discretos disparados pelo operador, conforme detalhado na seção 4.3. Para receber e persistir os dados, a função de assinante é implementada diretamente em Python. O *script* se inscreve no tópico MQTT configurado no broker HiveMQ Cloud e, ao receber cada mensagem, desserializa o conteúdo JSON, valida a integridade e a faixa de validade dos campos esperados e, por fim, grava o registro no banco de dados em nuvem Supabase por meio do *SDK* oficial da plataforma (`supabase-py`). Essa abordagem reúne em um único módulo as responsabilidades de recepção, validação e persistência, o que reduz a complexidade de operação e elimina a necessidade de agentes intermediários.

O terceiro ciclo foca na análise e na visualização dos dados coletados. Com as medições gravadas no Supabase, *scripts* em Python (v3.10) recuperam os registros armazenados e os processam. Com as bibliotecas Pandas (v2.1.4) e NumPy (v1.26.3), são calculados indicadores

estatísticos como média, desvio-padrão e os índices de capacidade do processo C_p e C_{pk}. Por fim, a biblioteca Plotly gera os gráficos interativos do painel de visualização (gráficos de controle de Shewhart e séries temporais das medições), renderizados no navegador por meio do *framework* Streamlit, permitindo o monitoramento em tempo real e a análise histórica do processo.

A etapa final do projeto é dedicada à validação do sistema por meio de testes práticos. O objetivo é caracterizar critérios como a estabilidade da conexão Wi-Fi em operação prolongada, a latência da via de comunicação MQTT, e a robustez do mecanismo de armazenamento e encaminhamento sob diferentes cenários de falha de conectividade. Os resultados quantitativos de cada teste, junto com a documentação da montagem (fotos, esquemas e código-fonte), servem de base para a análise crítica do trabalho e para a discussão da viabilidade da solução proposta no contexto da Indústria 4.0.

4.3 Arquitetura Orientada a Eventos do Firmware

O *firmware* desenvolvido para o microcontrolador ESP32 foi estruturado segundo o paradigma orientado a eventos, implementado por meio de uma máquina de estados finitos com oito estados distintos: `AGUARDANDO`, `MEDINDO`, `AGUARDANDO_CONFIRMACAO`, `PUBLICANDO_DIRETO`, `STORING`, `DESCARTANDO`, `DRENANDO` e `ERRO_BUFFER_CHEIO`. Essa escolha de arquitetura reflete diretamente o fluxo de trabalho do operador no posto de medição: o sistema permanece ocioso até que uma ação intencional do operador, o clique no botão, inicie um ciclo de captura, processamento e publicação. Cada estado define um comportamento bem delimitado e um conjunto específico de transições, o que garante previsibilidade e facilita a verificação do comportamento do sistema em diferentes condições (LEE; SESHIA, 2017).

O sistema responde a três tipos de evento: os cliques do operador no botão, os temporizadores internos (fim da janela de aquisição de 500 ms, esgotamento dos 5 s de confirmação) e os resultados da comunicação (sucesso ou falha na publicação). Esses eventos se relacionam por meio dos estados: cada estado define quais eventos o sistema aceita e qual transição cada um dispara, de modo que um mesmo evento produz efeitos diferentes conforme o estado corrente, o clique inicia a captura em `AGUARDANDO`, confirma a medição em `AGUARDANDO_CONFIRMACAO` e é ignorado nos demais estados. Eventos não previstos para o estado corrente não têm efeito, o que garante que o sistema execute apenas sequências válidas do fluxo de medição.

O fluxo nominal de operação começa com o sistema no estado `AGUARDANDO`, sinalizado por LED verde fixo quando o dispositivo está conectado ao broker, ou por LED vermelho intermitente quando opera sem conexão. Ao detectar o primeiro clique do botão, já filtrado pela rotina de tratamento de repique por software com janela de 50 milissegundos, o sistema passa para `MEDINDO`, sinalizado por LED branco fixo. Nesse estado, é coletada uma janela de 50 amostras do conversor analógico-digital ao longo de 500 milissegundos, sobre a qual são calculados a média e o desvio-padrão amostral em milivolts. A consolidação estatística das amostras dentro

da janela atenua o ruído de alta frequência e produz uma medida representativa do deslocamento físico no instante da captura.

Após a coleta da janela, o sistema passa para `AGUARDANDO_CONFIRMACAO`, sinalizado por LED branco em piscada rápida. Nesse estado, o sistema espera por até cinco segundos um segundo clique do operador. O segundo clique funciona como confirmação explícita de que a medição foi proposital e protege o sistema contra acionamentos acidentais do botão, situação comum no ambiente fabril, em que o operador pode encostar sem querer nos controles enquanto manipula a peça. Se a confirmação ocorre dentro do tempo limite, o sistema passa para `PUBLICANDO_DIRETO`; caso contrário, passa para `DESCARTANDO`, sinalizando ao operador, com LED vermelho intermitente, que a medição foi descartada, antes de voltar ao estado `AGUARDANDO`.

4.4 Mecanismo de Armazenamento e Encaminhamento com Persistência em NVS

Para garantir a resiliência do sistema a falhas de conectividade, condição esperada em ambiente industrial, foi implementada uma fila local em memória não volátil (NVS) do ESP32, com capacidade configurada para até 100 medições. A fila é estruturada de forma circular, no padrão FIFO, preservando a ordem temporal das medições durante os períodos sem conexão. Cada medição é armazenada como uma estrutura binária que contém o identificador único do evento, o valor consolidado em milivolts, o desvio-padrão da janela de amostragem, o número de amostras coletadas e o instante da captura (tempo de atividade do dispositivo desde a inicialização).

A detecção da falha ocorre no momento da publicação: ao tentar publicar uma medição confirmada pelo operador no estado `PUBLICANDO_DIRETO`, o *firmware* verifica o estado da conexão com o broker MQTT. Se a publicação falha, o sistema passa para o estado `STORING`, no qual a medição é gravada na fila local da NVS antes de voltar ao estado `AGUARDANDO`. Do ponto de vista do operador, o ciclo de captura é concluído normalmente; a única diferença visível é o pulso branco no LED em vez do pulso verde, indicando que a medição foi preservada localmente mas ainda não transmitida.

Quando o *loop* principal detecta, ao mesmo tempo, que o sistema está no estado `AGUARDANDO`, que a conexão com o broker MQTT foi restabelecida e que existem medições pendentes na fila, ocorre uma transição automática para o estado `DRENANDO`. Nesse estado, sinalizado por LED branco em piscada rápida, o sistema reenvia automaticamente as medições pendentes, uma por ciclo do *loop* principal, removendo cada uma da fila somente após a publicação bem-sucedida. Essa estratégia de drenagem incremental, em vez de uma drenagem em lote que bloquearia o sistema, preserva a resposta às ações do operador, que pode iniciar novas medições mesmo durante a recuperação da fila. Se a conexão cair de novo durante a drenagem, o sistema volta ao estado `AGUARDANDO` e espera nova reconexão para retomar o processo.

Se a fila atinge sua capacidade máxima de 100 medições durante um período prolongado

sem conexão, o sistema passa para o estado `ERRO_BUFFER_CHEIO` e recusa novas medições, sinalizando ao operador, com LED vermelho fixo, que o limite foi atingido. Essa condição persiste até que pelo menos uma vaga seja liberada pela drenagem, momento em que o sistema retorna automaticamente ao estado `AGUARDANDO`. O dimensionamento da fila em 100 posições foi escolhido com base na expectativa de janelas curtas de indisponibilidade em ambiente fabril típico, podendo ser ajustado conforme as características de cada cenário de implantação.

4.5 Identificação, Rastreabilidade e Deduplicação

Cada evento de medição publicado pelo sistema carrega um identificador único de 32 bits (`event_id`), gerado localmente pelo ESP32 por meio da função `esp_random()` do *SDK* da Espressif. Além disso, cada evento carrega o identificador da máquina ou posto de trabalho no qual o dispositivo está instalado (`machine_id`), definido como constante no *firmware* e ajustado a cada implantação. Essa dupla identificação garante rastreabilidade completa de cada medição, permitindo que análises posteriores correlacionem os dados ao posto físico de origem e detectem variações de processo específicas por máquina, operador ou turno.

A hierarquia de tópicos MQTT adotada segue o formato `maquinas/<machine_id>/medicao` para os eventos de medição e `maquinas/<machine_id>/status` para as mensagens periódicas de estado operacional do dispositivo. Essa estrutura permite que o assinante filtre mensagens de uma máquina específica ou consuma ao mesmo tempo os dados de todas as máquinas pelo curinga `maquinas/+/medicao`, o que confere flexibilidade e escalabilidade a cenários com vários dispositivos.

O identificador único de evento (`event_id`) tem papel central na rastreabilidade individual de cada medição: ele permite que as análises posteriores referenciem eventos específicos sem ambiguidade e que os registros gravados no banco sejam correlacionados com os *logs* de operação do dispositivo.

4.6 Decisão do Nível de Qualidade de Serviço (QoS)

A escolha do nível de qualidade de serviço do MQTT define o equilíbrio entre a confiabilidade da entrega e a sobrecarga de comunicação. Para este sistema, adotou-se o nível 1 (QoS 1), tanto na publicação pelo ESP32 quanto na assinatura pelo consumidor. Nesse nível, o broker confirma o recebimento de cada mensagem por meio de um pacote de reconhecimento (PUBACK), e a mensagem é retransmitida caso a confirmação não chegue no tempo previsto, garantindo a entrega ao menos uma vez.

A adoção do QoS 1 na publicação exigiu uma decisão quanto à biblioteca MQTT do *firmware*. A biblioteca `PubSubClient`, muito difundida no ecossistema Arduino, não implementa o mecanismo de confirmação (PUBACK) necessário ao QoS 1 do lado do publicador, embora aceite o parâmetro sintaticamente. Optou-se, portanto, pela biblioteca `arduino-mqtt` (de Joël

Gähwiler), que implementa o QoS 1 completo na publicação e mantém um consumo de memória baixo, adequado ao ESP32.

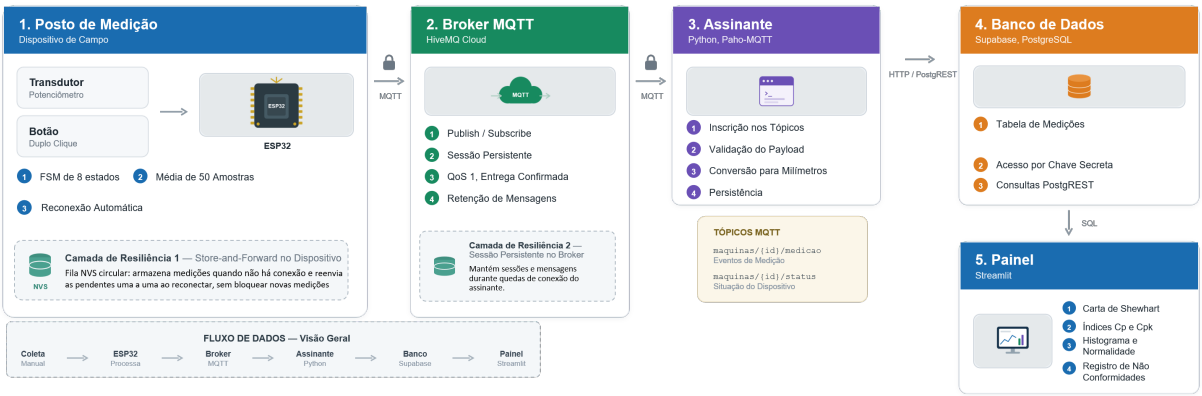
O uso do QoS 1 nas duas pontas é condição necessária para habilitar a sessão persistente do broker: ao configurar o assinante com identificador de cliente fixo e o atributo *clean session* desabilitado, o broker passa a reter as mensagens publicadas durante eventuais períodos de indisponibilidade do consumidor, entregando-as na reconexão. Esse mecanismo constitui uma das camadas de resiliência do sistema, validada experimentalmente no Capítulo 5.

O QoS 1 garante a entrega ao menos uma vez, o que admite, em cenários de retransmissão, a possibilidade teórica de entrega duplicada. No contexto deste trabalho, de medições discretas e de baixa frequência, com identificador único por evento (`event_id`) para fins de rastreabilidade, eventuais duplicatas não comprometem a análise estatística, e seu tratamento, caso necessário em implantações de maior escala, pode ser feito no nível da aplicação. A confiabilidade do sistema é assim construída em camadas: o QoS 1 e a sessão persistente atuam no nível do protocolo e do broker, enquanto o armazenamento local no dispositivo cobre as falhas de conectividade na origem (AL-FUQAHA *et al.*, 2015).

4.7 Representação Geral

A arquitetura do sistema proposto é sintetizada no diagrama de blocos da Figura 10, que ilustra o fluxo de dados de ponta a ponta, desde a aquisição no transdutor até a visualização final, junto aos mecanismos de resiliência e ao nível de qualidade de serviço adotados, detalhados nas seções anteriores.

Figura 10 – Arquitetura geral do sistema proposto.



Fonte: Elaboração própria, 2026.

O percurso dos dados pode ser descrito em cinco blocos funcionais encadeados. No primeiro, o transdutor de deslocamento (potenciômetro linear) converte a posição mecânica do cursor em um sinal elétrico analógico. No segundo, o ESP32 realiza a aquisição desse sinal por meio de seu conversor analógico-digital, executa a máquina de estados orientada a eventos,

consolida estatisticamente a janela de amostras e serializa o resultado em uma mensagem no formato JSON. No terceiro bloco, essa mensagem é transmitida pela rede: o ESP32 a publica, sobre uma conexão MQTT segura por TLS, em um tópico do *broker* HiveMQ Cloud, que a encaminha ao assinante inscrito. No quarto bloco, o assinante, implementado em Python, recebe a mensagem, valida e converte os campos e a persiste no banco de dados em nuvem Supabase. No quinto e último bloco, o painel desenvolvido em Streamlit consulta o banco e apresenta as medições por meio de gráficos de controle e dos índices de capacidade do processo.

O alcance do protocolo MQTT dentro dessa arquitetura precisa ser delimitado com precisão, pois ele não cobre todo o percurso dos dados. O MQTT atua exclusivamente no trecho compreendido entre o ESP32 e o assinante em Python, sempre intermediado pelo *broker*: o dispositivo é o publicador, o HiveMQ Cloud é o encaminhador central e o *script* em Python é o assinante. A partir do assinante, a comunicação muda de natureza: o mesmo *script* que recebe os dados via MQTT passa a gravá-los no Supabase por meio de chamadas HTTP à API PostgREST (biblioteca `supabase-py`), e o painel, por sua vez, consulta o banco por meio de SQL. O assinante funciona, portanto, como ponto de tradução entre o domínio MQTT (orientado a mensagens e tópicos) e o domínio do banco de dados relacional (orientado a requisições e tabelas), e não como um simples repassador. Essa delimitação evita a interpretação equivocada de que o MQTT seria o protocolo de toda a cadeia: seu papel é específico e se encerra na entrega da mensagem ao assinante.

4.8 Lista de Itens e Orçamento

A seleção dos componentes foi guiada por um critério de custo-benefício, dadas as restrições orçamentárias do projeto. A escolha buscou soluções de *hardware* acessíveis, sem comprometer os requisitos de funcionalidade do sistema. A Tabela 2 apresenta a lista de materiais.

Adicionalmente, são utilizados serviços em nuvem em seus respectivos planos gratuitos: o *broker* HiveMQ Cloud (limite de 100 conexões simultâneas e 10 GB de tráfego mensal) e o Supabase (volume ilimitado de requisições à API, 500 MB de armazenamento, 5 GB de tráfego mensal e 2 projetos ativos por organização). Nos dois casos, os limites do plano gratuito são suficientes para a operação de um único dispositivo em escala de protótipo; a expansão para vários dispositivos é discutida nas considerações finais. A decisão pelo Supabase no lugar do Firebase Cloud Firestore, inicialmente previsto na arquitetura, é detalhada na Seção 5.2 do Capítulo 5, onde se documentam as limitações práticas observadas com a plataforma inicial e a justificativa técnica da migração.

Tabela 2 – Lista de materiais e orçamento do projeto.

Item	**Descrição Detalhada**	**Custo (R$)**
Microcontrolador	Modelo ESP32 DevKitC (38 pinos). Responsável pela leitura do sensor, execução da máquina de estados e comunicação Wi-Fi via MQTT.	R$ 36,00
Cabo de dados/alimentação	Cabo Micro USB para conectar o ESP32 ao computador para programação e alimentação.	R$ 20,00
Transdutor de deslocamento	Potenciômetro linear deslizante de 10 kΩ. Converte o deslocamento mecânico do cursor em sinal elétrico analógico.	R$ 17,00
Botão de pressão	Botão táctil momentâneo, normalmente aberto, utilizado para o acionamento da medição pelo operador.	R$ 5,00
LEDs (branco, verde e vermelho)	Três LEDs comuns de 5 mm para a sinalização visual do estado da máquina ao operador.	R$ 0,50
Resistores	Três resistores de 220 Ω, um para cada LED.	R$ 1,00
Placa de ensaio (*protoboard*)	Matriz de contato de 830 pontos utilizada para a montagem do circuito sem solda.	R$ 9,50
Fios de conexão	*Jumpers* macho-fêmea e macho-macho para a interface elétrica entre os componentes e o microcontrolador.	R$ 5,00
Total		**R$ 94,00**

5 RESULTADOS

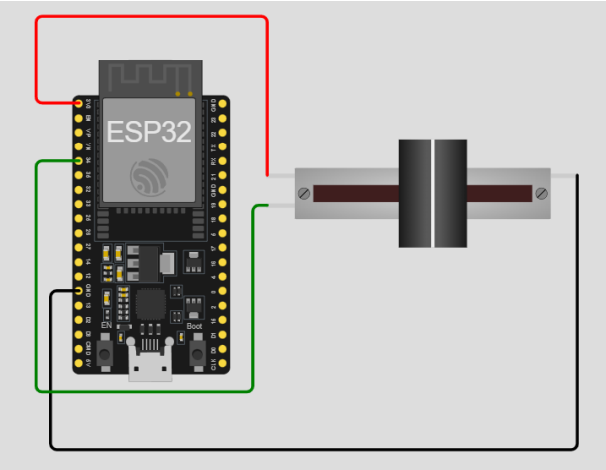
Este capítulo apresenta os resultados obtidos no desenvolvimento e na validação do sistema IoT proposto. A apresentação está organizada em três blocos: o primeiro descreve a fase preliminar de validação eletrônica do circuito de aquisição; o segundo apresenta a implementação final do *firmware* orientado a eventos e da cadeia de comunicação e persistência; e o terceiro reúne os testes de caracterização do sistema, incluindo a validação do mecanismo de armazenamento e encaminhamento, a caracterização da latência da via MQTT e a demonstração da aplicabilidade dos dados em análise estatística de processo.

5.1 Validação Eletrônica Preliminar

As figuras desta seção documentam a fase inicial de validação eletrônica do circuito de aquisição, realizada em ambiente de prototipagem antes da integração com a máquina de estados e com os mecanismos de comunicação descritos nas seções seguintes. Nessa etapa preliminar, o *firmware* operava em modo de leitura contínua periódica, publicando o valor bruto do ADC a cada segundo, com o objetivo exclusivo de validar a transdução do sinal pelo conversor analógico-digital, o mapeamento dos pinos e a integridade da comunicação MQTT com o broker HiveMQ Cloud.

A validação foi feita em duas etapas: simulação virtual no ambiente Wokwi e prototipagem física em *protoboard*. Na simulação, o circuito eletrônico responsável pela aquisição do sinal foi montado e testado, permitindo validar previamente a lógica de programação e o mapeamento dos pinos de comunicação. Conforme ilustra o esquemático da Figura 11, o pino de sinal (SIG) do potenciômetro linear foi conectado à porta do conversor analógico-digital GPIO 34 do microcontrolador ESP32. Os terminais de alimentação do sensor (VCC e GND) foram ligados, respectivamente, às saídas de 3,3 V e terra (GND) do microcontrolador, formando o circuito divisor de tensão necessário para a transdução do deslocamento mecânico em sinal elétrico.

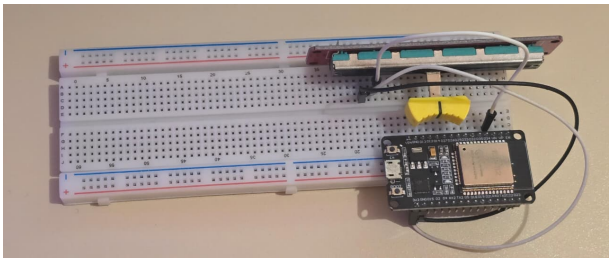
Figura 11 – Esquemático do potenciômetro linear deslizante conectado ao ESP32 (simulação Wokwi).



Fonte: Elaboração própria, 2026.

Após o sucesso na simulação, foi feita a montagem física em *protoboard* para validar o comportamento real dos componentes elétricos, conforme apresentado na Figura 12. A dupla validação, virtual e física, confirmou que o deslocamento do cursor é convertido com precisão pelo ADC de 12 bits do ESP32 em valores digitais entre 0 e 4095.

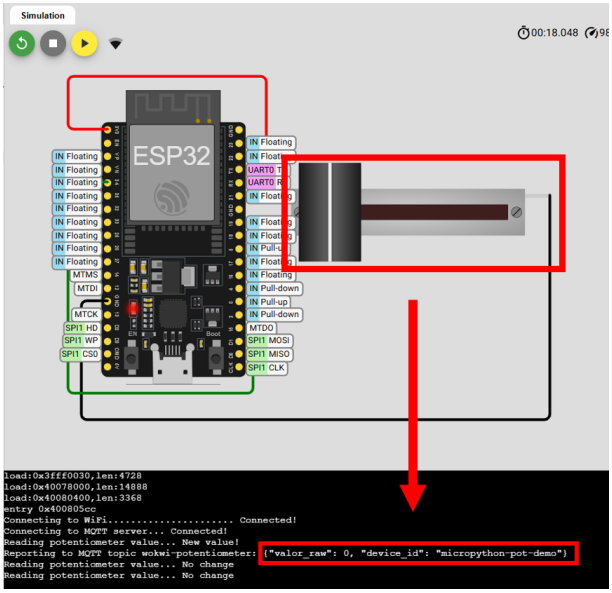
Figura 12 – Montagem física do potenciômetro linear deslizante e ESP32 em *protoboard*.



Fonte: Elaboração própria, 2026.

As Figuras 13, 14, 15 e 16 ilustram a resposta do sistema em diferentes posições do cursor, mostrando a conversão linear do deslocamento físico em valores digitais. Com o cursor na posição inicial, o ADC reporta valor próximo de zero; com o cursor na posição final do curso útil, o valor se aproxima de 4095, confirmando o uso integral da resolução de 12 bits do conversor.

Figura 13 – Potenciômetro na posição inicial (simulação).



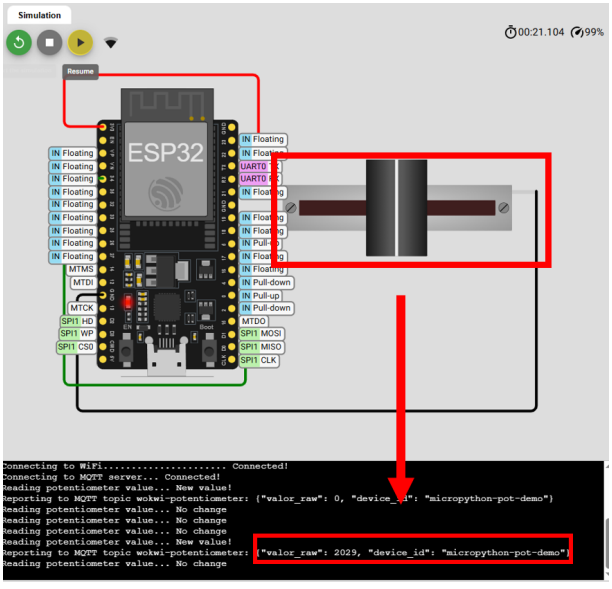
Fonte: Elaboração própria, 2026.

Figura 14 – Potenciômetro em posição intermediária aproximada (protoboard).

```
19:45:16.908 -> ............................
19:45:19.610 -> [WIFI] Conectado. IP: 192.168.2.21
19:45:19.610 -> [MQTT] Conectando ao HiveMQ Cloud (SSL)...CONECTADO!
19:45:21.928 -> [MQTT SSL] Publicando: {"raw": 1960, "voltage": 1.58, "sec": true}
19:45:22.892 -> [MQTT SSL] Publicando: {"raw": 1959, "voltage": 1.58, "sec": true}
19:45:23.935 -> [MQTT SSL] Publicando: {"raw": 1968, "voltage": 1.59, "sec": true}
19:45:24.911 -> [MQTT SSL] Publicando: {"raw": 1971, "voltage": 1.59, "sec": true}
```

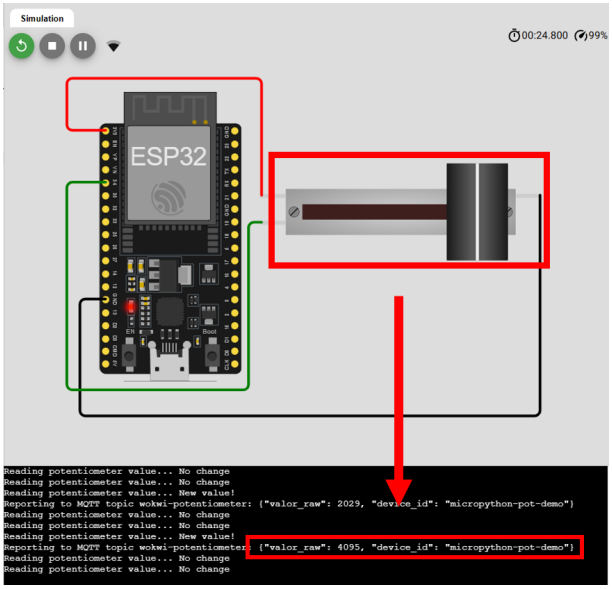
Fonte: Elaboração própria, 2026.

Figura 15 – Potenciômetro em posição intermediária (simulação).



Fonte: Elaboração própria, 2026.

Figura 16 – Potenciômetro na posição final (simulação).



Fonte: Elaboração própria, 2026.

Figura 17 – Potenciômetro na posição final, montagem em *protoboard*.

```
19:39:20.586 -> [BOOT] Iniciando Sistema IoT Seguro...
19:39:20.757 -> ..............)□.................................
19:39:29.057 -> [WIFI] Conectado. IP: 192.168.2.21
19:39:29.057 -> [MQTT] Conectando ao HiveMQ Cloud (SSL)...CONECTADO!
19:39:31.064 -> [MQTT SSL] Publicando: {"raw": 4095, "voltage": 3.30, "sec": true}
19:39:32.075 -> [MQTT SSL] Publicando: {"raw": 4095, "voltage": 3.30, "sec": true}
19:39:33.080 -> [MQTT SSL] Publicando: {"raw": 4095, "voltage": 3.30, "sec": true}
19:39:34.090 -> [MQTT SSL] Publicando: {"raw": 4095, "voltage": 3.30, "sec": true}
19:39:35.069 -> [MQTT SSL] Publicando: {"raw": 4095, "voltage": 3.30, "sec": true}
```

Fonte: Elaboração própria, 2026.

A versão final do *firmware*, descrita nas seções seguintes, substitui a lógica de leitura periódica desta fase preliminar por uma arquitetura completa orientada a eventos, com captura acionada pelo operador, janela de aquisição consolidada estatisticamente, confirmação por duplo-clique e mecanismo de armazenamento e encaminhamento para resiliência a falhas de rede.

5.2 Revisão da Plataforma de Persistência em Nuvem

A arquitetura inicialmente proposta no Capítulo 4 previa o uso do Firebase Cloud Firestore como banco de dados em nuvem para o armazenamento das medições. Durante a fase de validação integrada do sistema, no entanto, ficaram evidentes limitações práticas do plano gratuito da plataforma que motivaram a revisão dessa decisão. Esta seção documenta o problema observado, a análise de causa-raiz, a comparação das alternativas consideradas e a justificativa técnica da substituição do Firestore pelo Supabase, plataforma também baseada em serviço gerenciado de banco de dados em nuvem. A apresentação desta revisão antes da descrição da implementação final do assinante é proposital e segue a ordem cronológica do desenvolvimento: a decisão de

migração precedeu e moldou a versão definitiva do componente de persistência apresentada nas seções seguintes.

5.2.1 Identificação do Problema

Durante os primeiros testes integrados do sistema de ponta a ponta, observou-se que a cota diária de leituras do Firestore se esgotou em poucas horas de operação. O painel de monitoramento do Firebase reportou cerca de 51 mil operações de leitura em uma única sessão de teste, ultrapassando o limite de 50 mil leituras diárias do plano gratuito (*Spark*) e suspendendo temporariamente as consultas até a renovação automática da cota. As operações de escrita, em contraste, ficaram perto de 1,7 mil, valor bem abaixo do limite de 20 mil escritas por dia, o que indicava que o problema não estava na taxa de publicação do dispositivo, mas no padrão de leitura imposto pelo painel de visualização.

5.2.2 Análise de Causa-Raiz

O modelo de cobrança do Firebase Cloud Firestore contabiliza como uma leitura tarifável cada documento recuperado individualmente em uma consulta. O painel desenvolvido em Streamlit, ao ser atualizado, recarrega a coleção de medições para calcular os indicadores estatísticos e gerar os gráficos de controle, executando uma operação de leitura por documento existente na janela de interesse. Em um cenário com N documentos consultados e R atualizações do painel durante a sessão, o consumo total de leituras cresce na proporção $N \times R$, padrão incompatível com sessões de uso interativo mesmo para volumes modestos de dados.

A natureza do caso de uso, de monitoramento contínuo de processo industrial, com atualizações frequentes do painel e janelas de tempo móveis, revela uma incompatibilidade fundamental entre o modelo de cobrança por leitura do Firestore e o padrão de acesso típico de painéis em tempo real, em que a mesma coleção é consultada repetidamente em intervalos curtos. Ao contrário do que se previa no planejamento inicial, o gargalo da arquitetura não está na taxa de geração de dados pelo dispositivo, mas na taxa de consumo de dados pelo painel.

5.2.3 Análise de Alternativas

Diante do problema identificado, foram consideradas três alternativas, resumidas no Quadro 1.

Quadro 1 – Alternativas avaliadas para resolução do problema de cota de leituras.

Alternativa	**Descrição**	**Avaliação**
(i) Permanecer no Firestore com otimizações	Limitar artificialmente o número de leituras por atualização, aumentar o tempo de *cache* no Streamlit, implementar paginação.	Mitiga sintomas, não a causa-raiz. Recorrência esperada com o crescimento dos dados.
(ii) Migrar para plataforma com modelo de cobrança baseado em recursos e tráfego	Substituir o Firestore por um serviço gerenciado cujo plano gratuito não tarifa por número de operações de leitura.	Ataca a causa-raiz. Adotada.
(iii) Persistência local complementar em *cache*	Manter o Firestore como fonte primária dos dados e adicionar uma cópia local em SQLite para alimentar o painel.	Compromete a coerência da arquitetura ao fragmentar a fonte primária dos dados entre a nuvem e o dispositivo local.

A alternativa (i) foi descartada por ser, no fundo, uma mitigação dos sintomas e não da causa-raiz: qualquer crescimento futuro do volume de dados ou da frequência de uso do painel faria o problema voltar, mesmo com as otimizações em vigor. A alternativa (iii), embora tecnicamente viável e implementada de início como solução de contingência, compromete a coerência da arquitetura ao fragmentar a fonte da verdade entre a nuvem e a máquina local: um sistema de monitoramento remoto que depende da conexão local com o servidor que mantém o *cache* perde sua proposta original. A alternativa (ii) foi adotada por atacar a causa-raiz e preservar a arquitetura proposta, com o banco de dados em nuvem como fonte única da verdade.

5.2.4 Escolha do Supabase

A plataforma selecionada foi o Supabase, serviço de banco de dados em nuvem baseado em PostgreSQL gerenciado, conforme apresentado no Capítulo 3. Os critérios técnicos que motivaram a escolha são:

- **Modelo de cobrança compatível:** o plano gratuito do Supabase oferece volume ilimitado de requisições à API, dissociando o custo do número de consultas ao banco e tornando o uso interativo do painel sustentável independentemente da frequência de atualização.
- **Capacidade de filtragem no servidor:** o suporte nativo a SQL permite que o painel filtre as medições por janela de tempo por meio de cláusulas `WHERE` sobre a coluna de data e hora, reduzindo o volume de dados transferidos ao mínimo necessário e passando o esforço de processamento da aplicação para o banco. Essa vantagem não existe na consulta a coleções não relacionais do Firestore, em que a filtragem é menos expressiva e a recuperação de campos calculados exige a composição manual de consultas.
- **Compatibilidade com a arquitetura proposta:** o Supabase oferece *SDK* oficial para Python (`supabase-py`) com interface fluente e autenticação via chave de API, exigindo adaptações pontuais no assinante e no painel, sem impacto sobre o *firmware* do ESP32, o broker MQTT ou os mecanismos de validação implementados.

5.2.5 Implicações Arquiteturais e Adequação à Metodologia Ágil

A substituição do Firestore pelo Supabase exigiu apenas adaptações pontuais em dois componentes do *software*: o assinante em Python e o painel. Todo o resto permaneceu inalterado: o *firmware* do ESP32, o protocolo MQTT, o broker HiveMQ Cloud, a estrutura das mensagens JSON, o mecanismo de identificação única por `event_id` e o esquema lógico das medições. O assinante apenas substitui as chamadas do *SDK* `firebase-admin` por chamadas equivalentes do `supabase-py`, mantendo intacta a lógica de validação e persistência. O painel, por sua vez, se beneficia da migração ao adotar consultas SQL declarativas com filtragem no servidor, no lugar do padrão anterior de recuperação integral da coleção.

A revisão de arquitetura mostra, na prática, uma vantagem da abordagem ágil adotada na fase de software (Capítulo 4): a capacidade de revisar decisões de implementação à luz de evidências obtidas durante a validação, sem comprometer o cronograma global do projeto. A identificação precoce do problema, antes da implementação final do painel e dos testes de capacidade de processo, permitiu que a migração ocorresse sem retrabalho relevante e que os resultados apresentados nas próximas seções já refletissem a arquitetura definitiva do sistema.

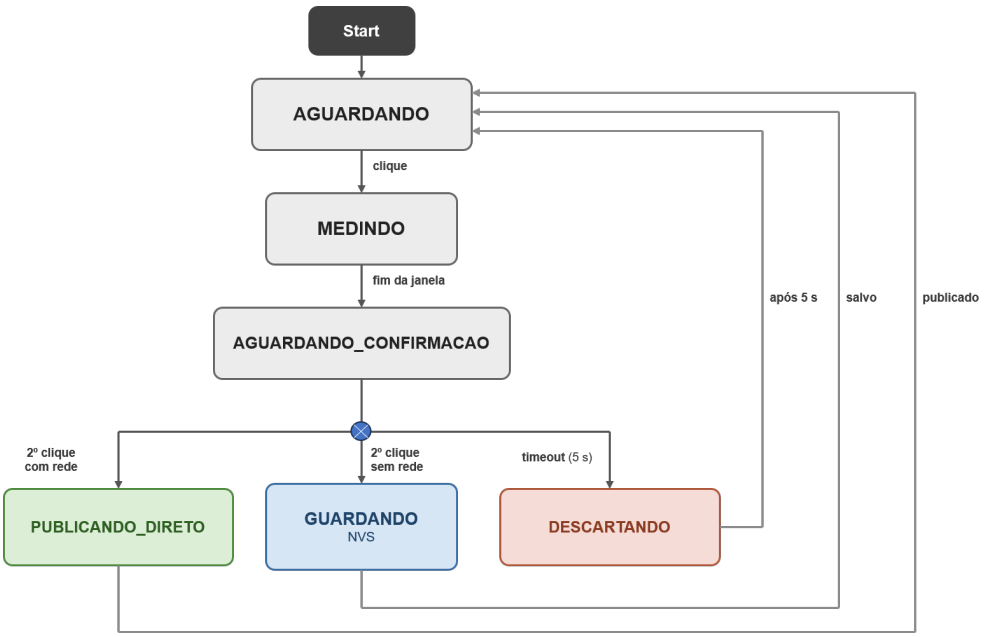
5.3 Implementação da Máquina de Estados

A versão final do *firmware* implementa uma máquina de estados finitos completa, com oito estados e transições determinísticas disparadas por eventos do operador, do sistema de comunicação e de temporizadores internos. O Quadro 2 resume os estados, a sinalização visual associada por meio dos LEDs e as principais condições de transição.

Quadro 2 – Estados da máquina de estados finitos do *firmware*.

Estado	**Sinalização (LEDs)**	**Comportamento e transições**
`AGUARDANDO`	Verde fixo (conectado) ou vermelho piscando (sem conexão)	Estado de repouso. Transita para `MEDINDO` ao detectar o primeiro clique do botão, ou para `DRENANDO` se houver conexão e medições pendentes na fila.
`MEDINDO`	Branco fixo	Coleta janela de 50 amostras em 500 ms; calcula média e desvio-padrão. Transita automaticamente para `AGUARDANDO_CONFIRMACAO`.
`AGUARDANDO_CONFIRMACAO`	Branco em piscada rápida	Aguarda o segundo clique do operador por até 5 s. Ao confirmar, avalia a conexão e transita para `PUBLICANDO_DIRETO` (se houver rede) ou para `STORING` (caso contrário); ao esgotar o tempo, transita para `DESCARTANDO`.
`PUBLICANDO_DIRETO`	Pulso verde por ≈2 s	Publica a medição diretamente via MQTT. Transita para `AGUARDANDO` em caso de sucesso ou para `STORING` em caso de falha.
`STORING`	Pulso branco por ≈2 s	Grava a medição na fila NVS. Transita para `AGUARDANDO` após a gravação.
`DESCARTANDO`	Vermelho piscando por ≈5 s	Sinaliza o descarte da medição não confirmada. Transita automaticamente para `AGUARDANDO`.
`DRENANDO`	Branco piscando, com pulso verde a cada medição reenviada	Reenvia as medições pendentes da fila, uma por ciclo do *loop*. Transita para `AGUARDANDO` ao esvaziar a fila.
`ERRO_BUFFER_CHEIO`	Vermelho fixo	Bloqueia novas medições enquanto a fila estiver cheia em modo sem conexão. Transita para `AGUARDANDO` ao liberar espaço por drenagem.

Figura 18 – Fluxograma da máquina de estados do firmware.



Fonte: Elaboração própria, 2026.

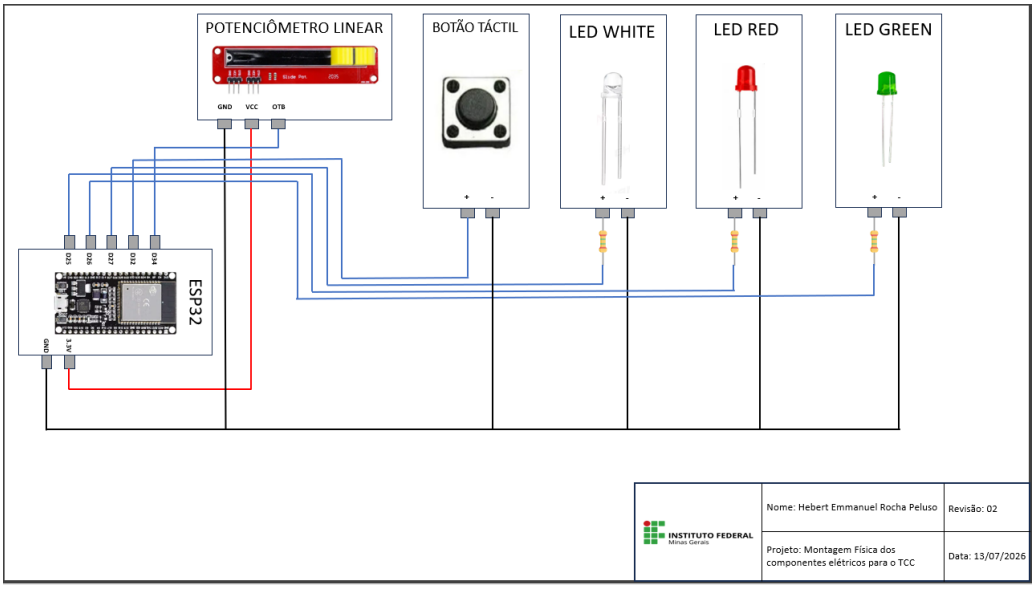
A sinalização foi implementada com três LEDs discretos (vermelho, verde e branco), acionados independentemente pelo *firmware*, em vez de um único módulo RGB. Cada estado é distinguido por uma combinação de cor e padrão temporal (aceso fixo, piscada lenta ou piscada

rápida), conforme detalhado no Quadro 2, o que torna a leitura do estado inequívoca para o operador sem recorrer a cores compostas.

A detecção do botão é feita por uma rotina de filtragem de repique por software, com janela de estabilização de 50 milissegundos, que elimina as transições espúrias características de contatos mecânicos e garante que cada pressionamento do operador seja interpretado como um único evento. A janela de aquisição executada no estado `MEDINDO` coleta 50 amostras consecutivas do ADC ao longo de 500 milissegundos, o que corresponde a uma taxa de amostragem efetiva de 100 Hz, e sobre elas são calculados a média e o desvio-padrão amostral em milivolts. A média representa o valor central da medição, enquanto o desvio-padrão funciona como indicador da estabilidade do sinal durante a captura, permitindo ao assinante e à camada de análise identificar medições potencialmente comprometidas por vibração ou má acomodação da peça. O código completo do *firmware* encontra-se no Apêndice A.

A Figura 19 apresenta o esquema elétrico completo do sistema na sua configuração definitiva, com o mapeamento de todos os pinos: o cursor do potenciômetro linear na entrada analógica GPIO 34, o botão de captura, com resistor de *pull-up* interno, no GPIO 32, e os três LEDs de sinalização nos pinos GPIO 25 (vermelho), GPIO 26 (verde) e GPIO 27 (branco), cada um com resistor limitador de corrente. Esse esquema corresponde diretamente à sinalização visual descrita no Quadro 2 e à interface de operação por botão.

Figura 19 – Esquema elétrico completo do sistema, com o potenciômetro, o botão de captura e os três LEDs de sinalização conectados ao ESP32.

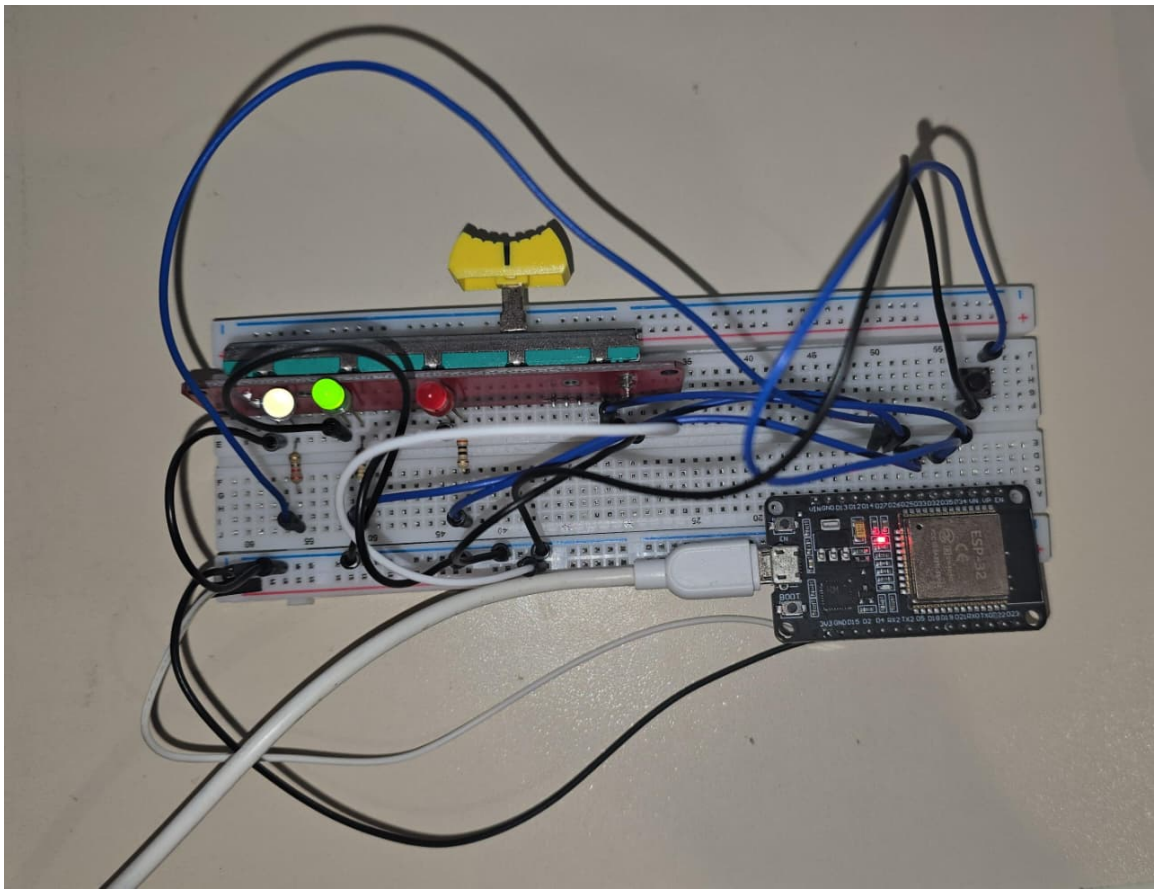


Fonte: Elaboração própria, 2026.

A montagem física correspondente a esse esquema é apresentada na Figura 20, que mostra o sistema completo integrado em *protoboard*, com o ESP32, o potenciômetro linear deslizante, o botão de captura e os três LEDs de sinalização, na configuração utilizada nos testes de validação

descritos nas seções seguintes.

Figura 20 – Montagem final do sistema em bancada, com ESP32, potenciômetro, botão de captura e LEDs de sinalização.



Fonte: Elaboração própria, 2026.

5.4 Publicação dos Eventos no *Broker*

O *firmware* desenvolvido para o ESP32 publica as medições confirmadas pelo operador no broker HiveMQ Cloud por meio de conexão segura SSL/TLS na porta 8883, utilizando o protocolo MQTT. Cada evento de medição é serializado em formato JSON e publicado no tópico `maquinas/<machine_id>/medicao`, conforme o exemplo a seguir:

```
{
  "event_id": 2847391024,
  "machine_id": "TORNO_01",
  "firmware_version": "1.0.0",
  "uptime_ms": 124583,
  "measurement": {
    "voltage_mean_mv": 1652.34,
    "voltage_stddev_mv": 1.82,
```

```
    "adc_raw_mean": 2051,
    "samples_count": 50,
    "window_ms": 502
  },
  "metadata": {
    "rssi_dbm": -62
  }
}
```

A estrutura da mensagem foi projetada para reunir, em um único envio, todas as informações necessárias à rastreabilidade do evento: o identificador único (`event_id`), o identificador da máquina de origem (`machine_id`), a versão do *firmware* (`firmware_version`), o instante da captura em tempo de atividade do dispositivo (`uptime_ms`), os dados consolidados da janela de medição e metadados operacionais como a intensidade do sinal Wi-Fi (`rssi_dbm`). Além disso, o *firmware* publica periodicamente, a cada 60 segundos, uma mensagem de estado no tópico `maquinas/<machine_id>/status`, contendo contadores operacionais como o número total de medições publicadas, o número de medições armazenadas localmente e o número de reconexões observadas. Essa mensagem é publicada com o atributo de retenção ativado, permitindo que sistemas de monitoramento conheçam de imediato o estado mais recente do dispositivo ao se conectarem ao broker.

5.5 Recepção, Validação e Persistência no Assinante

Para o recebimento das mensagens publicadas pelo ESP32, foi desenvolvido um *script* Python que atua como assinante MQTT, inscrevendo-se no mesmo tópico configurado no publicador. A conexão com o broker HiveMQ Cloud é estabelecida com autenticação por credenciais e nível 1 de qualidade de serviço (QoS 1) na assinatura. Tanto a publicação pelo ESP32 quanto a assinatura pelo *script* operam em QoS 1, o que habilita o mecanismo de sessão persistente do broker discutido na Seção 5.6.2: mensagens publicadas enquanto o assinante está temporariamente indisponível são enfileiradas pelo broker e entregues na reconexão.

Ao receber uma mensagem, o *script* desserializa o conteúdo JSON e executa uma etapa de validação, verificando a integridade e a faixa de validade dos campos esperados antes de prosseguir com o armazenamento. A persistência dos dados é feita por meio do *SDK* oficial do Supabase para Python (`supabase-py`), conforme decisão detalhada na Seção 5.2. O assinante se autentica na plataforma via chave de API e insere cada medição como uma nova linha da tabela `leituras` do banco PostgreSQL gerenciado. Cada linha armazena os campos da medição (valor bruto do ADC, tensão e medida convertida em milímetros), o tempo de atividade do dispositivo e a data e hora de recepção, gerada pelo servidor por meio do valor padrão `NOW()` da coluna correspondente.

5.6 Validação da Resiliência do Sistema

A integridade dos dados em sistemas IoT depende da capacidade do sistema de tolerar falhas de rede sem perda de medições. A solução proposta implementa resiliência em **duas camadas complementares**, que protegem contra falhas em pontos distintos do percurso dos dados, conforme ilustra o Quadro 3.

Quadro 3 – Camadas de resiliência do sistema.

Camada	**Protege contra**	**Mecanismo**
1 – Dispositivo (ESP32)	Perda de conectividade Wi-Fi do dispositivo e queda de energia	Fila de armazenamento e encaminhamento em memória não volátil (NVS)
2 – Broker (HiveMQ)	Indisponibilidade temporária do assinante (ponte de ingestão)	Sessão persistente MQTT (QoS 1 + *clean session* desabilitada)

Os testes de resiliência descritos nesta seção foram conduzidos com a versão de *firmware* voltada à validação da camada de comunicação, na qual a aquisição é contínua e automática (uma medição a cada cinco segundos), sem a interface física de captura por botão. O objetivo foi validar isoladamente os mecanismos de comunicação e resiliência antes da integração com a interface de operação final. As mensagens usadas nesses testes contêm os campos essenciais da medição (valor bruto do ADC, tensão e tempo de atividade); a estrutura completa descrita na Seção 5.4 será incorporada na versão final integrada.

5.6.1 Camada 1: Armazenamento e Encaminhamento no Dispositivo (NVS)

A primeira camada de resiliência fica no próprio ESP32. Quando a publicação de uma medição falha por falta de conectividade, a medição é gravada em uma fila circular implementada sobre a memória não volátil (NVS) do microcontrolador por meio da biblioteca `Preferences`. Nos testes desta seção, a fila foi configurada com capacidade para 20 registros, valor deliberadamente reduzido em relação às 100 posições do *firmware* de produção (Capítulo 4), de modo a tornar praticável a verificação do comportamento de esgotamento da fila sem prolongar excessivamente os ensaios. O caráter não volátil é essencial: ao contrário de uma fila mantida em memória RAM, os dados gravados na NVS sobrevivem a reinicializações e a quedas de energia do dispositivo. Ao restabelecer a conexão, o *firmware* drena a fila, republicando as medições pendentes em ordem cronológica antes de retomar a operação normal.

Foram conduzidos dois testes de validação desta camada.

Teste A — Resiliência à queda de Wi-Fi. Com o sistema em operação normal, a conectividade Wi-Fi foi propositalmente interrompida. Durante a indisponibilidade, o cursor do potenciômetro foi deslocado deliberadamente, a fim de verificar se as medições retidas refletiam

as variações reais do sinal, e não valores estáticos ou duplicados. O *log* do *Serial Monitor* durante o teste é reproduzido a seguir.

```
[MQTT] Publicado direto (raw=1991, QoS1)
[MQTT] Publicado direto (raw=1992, QoS1)
[WIFI] Sem conexao - operando OFFLINE (buffer NVS ativo)
[NVS] Medicao armazenada. Buffer: 1/20
[NVS] Medicao armazenada. Buffer: 2/20
[NVS] Medicao armazenada. Buffer: 3/20
[NVS] Medicao armazenada. Buffer: 4/20
[NVS] Medicao armazenada. Buffer: 5/20
[WIFI] Conectado! IP: 10.227.17.252
[MQTT] Tentando conectar ao HiveMQ... CONECTADO!
[NVS] Iniciando drenagem de 5 medicao(oes) pendente(s)...
[NVS]   (1/5) reenviada raw=1982
[NVS]   (2/5) reenviada raw=793
[NVS]   (3/5) reenviada raw=1294
[NVS]   (4/5) reenviada raw=2145
[NVS]   (5/5) reenviada raw=968
[NVS] Drenagem concluida: 5 medicao(oes) recuperada(s), buffer vazio.
[MQTT] Publicado direto (raw=977, QoS1)
```

O comportamento observado corresponde ao esperado: as cinco medições capturadas durante o período sem conexão foram preservadas e retransmitidas integralmente após a reconexão, com taxa de recuperação de 100%. Os valores brutos das medições drenadas (1982, 793, 1294, 2145 e 968) confirmam que o dispositivo capturou as variações reais do potenciômetro durante a indisponibilidade, comprovando a aquisição efetiva no modo desconectado.

Teste B — Resiliência à queda de energia. Para validar especificamente a propriedade não volátil da fila, o procedimento foi repetido, porém interrompendo a **alimentação** do ESP32 enquanto havia medições pendentes na fila e a conexão ainda indisponível. O *log* correspondente é reproduzido a seguir.

```
[WIFI] Sem conexao - operando OFFLINE (buffer NVS ativo)
[NVS] Medicao armazenada. Buffer: 1/20
[NVS] Medicao armazenada. Buffer: 2/20
[NVS] Medicao armazenada. Buffer: 3/20
[NVS] Estado carregado: head=9 tail=12 count=3
[NVS] 3 medicoes recuperadas do NVS apos boot!
[WIFI] Sem conexao - operando OFFLINE (buffer NVS ativo)
```

```
[NVS] Medicao armazenada. Buffer: 4/20
[WIFI] Conectado! IP: 10.227.17.252
[MQTT] Tentando conectar ao HiveMQ... CONECTADO!
[NVS] Iniciando drenagem de 4 medicao(oes) pendente(s)...
[NVS]   (1/4) reenviada raw=955
[NVS]   (2/4) reenviada raw=960
[NVS]   (3/4) reenviada raw=962
[NVS]   (4/4) reenviada raw=1069
[NVS] Drenagem concluida: 4 medicao(oes) recuperada(s), buffer vazio.
```

Neste teste, as três medições presentes na fila no instante do corte de energia foram recuperadas após a reinicialização do dispositivo, como evidencia a mensagem `3 medicoes recuperadas do NVS apos boot`. A preservação do estado interno da fila circular (`head=9`, `tail=12`, `count=3`) demonstra que não apenas os dados, mas também a estrutura de controle da fila, sobreviveram à queda de energia. Após a reinicialização, o dispositivo capturou ainda uma quarta medição em modo desconectado e, ao reconectar, drenou as quatro pendências integralmente. A Tabela 3 resume os dois testes.

Tabela 3 – Resultados dos testes de armazenamento e encaminhamento (camada NVS).

Teste	**Medições sem conexão**	**Recuperadas**	**Taxa**
A – Queda de Wi-Fi	5	5	100%
B – Queda de energia	3 (+1 pós-reinicialização)	4	100%

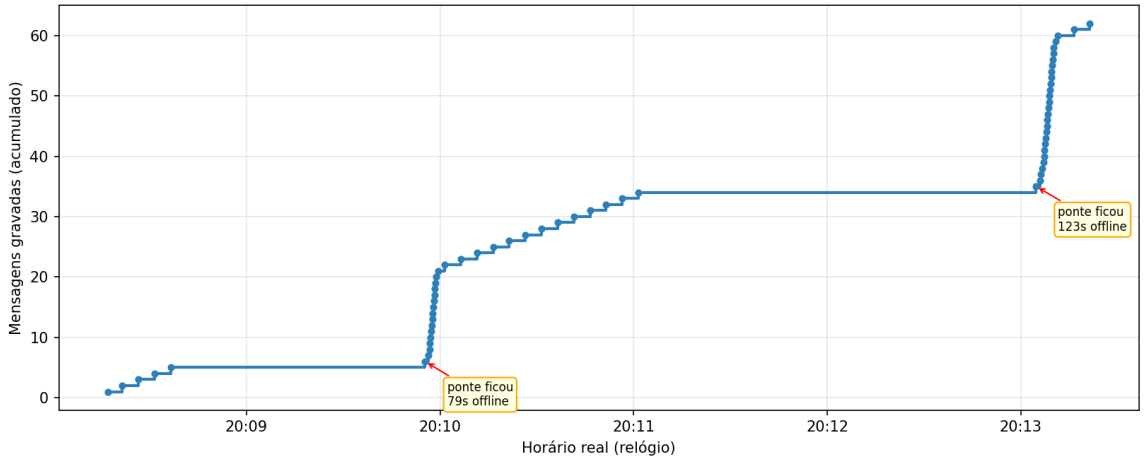
5.6.2 Camada 2: Sessão Persistente no *Broker*

A segunda camada de resiliência protege contra um modo de falha distinto: a indisponibilidade temporária do assinante (a ponte de ingestão), enquanto o ESP32 e o broker permanecem operacionais. Nesse cenário, a fila NVS do dispositivo não é acionada, pois o ESP32 continua publicando normalmente ao broker, e a retenção das mensagens fica a cargo do próprio broker.

O protocolo MQTT oferece esse mecanismo por meio da sessão persistente, que exige três condições simultâneas: publicação e assinatura em QoS ≥ 1, conexão do assinante com o atributo *clean session* desabilitado e uso de um identificador de cliente fixo. Satisfeitas essas condições, o broker reconhece o assinante ao reconectar e lhe entrega as mensagens enfileiradas durante a ausência.

Para validar esta camada, o assinante foi propositalmente interrompido enquanto o ESP32 mantinha a publicação contínua. A Figura 21 apresenta a análise temporal do experimento, comparando o instante de geração de cada medição (derivado do tempo de atividade do ESP32) com o instante de gravação no banco de dados.

Figura 21 – Recuperação de mensagens via sessão persistente do *broker* após reconexão do assinante.



Fonte: Elaboração própria, 2026.

A curva evidencia o mecanismo com clareza: nos intervalos em que o assinante esteve indisponível (79 segundos e 123 segundos, respectivamente, nos dois eventos registrados), o número de mensagens gravadas permaneceu constante, formando os "platôs" horizontais, enquanto o ESP32 seguia medindo sem interrupção. No instante da reconexão, o broker entregou de uma só vez todas as mensagens retidas, produzindo os "degraus" verticais de recuperação. Nos dois eventos, a totalidade das mensagens publicadas durante a ausência foi recuperada, sem perda de dados.

5.6.3 Síntese da Resiliência

Os testes confirmam que as duas camadas se complementam, formando uma estratégia de cobertura do caminho dos dados. A camada do dispositivo (NVS) garante a integridade dos dados mesmo na ausência total de conectividade ou em quedas de energia, enquanto a camada do broker (sessão persistente) cobre falhas na etapa de ingestão e persistência, sem exigir qualquer ação do dispositivo. Em conjunto, os mecanismos cobrem o percurso completo dos dados, do sensor ao banco de dados em nuvem, e conferem ao sistema tolerância a falhas em seus principais pontos de vulnerabilidade.

5.7 Caracterização da Latência

Para caracterizar o desempenho do sistema em condições normais de operação, foram realizados testes específicos de latência, estabilidade e integridade.

5.7.1 Caracterização da Latência da Via MQTT

Caracterizou-se isoladamente a contribuição da via de comunicação MQTT, o percurso entre o dispositivo publicador e o assinante, intermediado pelo broker. Isolar essa etapa permite distinguir a parcela de latência atribuível ao transporte das mensagens da parcela atribuível à validação e à gravação no banco de dados, além de quantificar o impacto da localização geográfica do broker sobre o desempenho do sistema.

5.7.1.1 Localização do *Broker* e Metodologia de Medição

O broker HiveMQ Cloud utilizado opera no plano *Serverless* gratuito, hospedado na infraestrutura da Amazon Web Services (AWS) na região `eu-central-1`, fisicamente localizada em Frankfurt, Alemanha. Como tanto o dispositivo ESP32 quanto o assinante em Python foram executados no Brasil, cada mensagem percorre aproximadamente 9.500 km até o broker e distância equivalente no retorno. Esse fator domina a latência observada e se reflete diretamente nos resultados apresentados a seguir.

A latência foi medida pelo método do tempo de ida e volta (*round-trip time*, RTT) sobre um único relógio, estratégia que elimina o erro de sincronização entre os relógios independentes do dispositivo e do assinante. O procedimento é o seguinte: o ESP32 publica uma mensagem contendo um número de sequência e registra o instante de envio pelo seu relógio monotônico interno (`millis()`); o assinante, ao receber a mensagem, devolve imediatamente uma confirmação (*acknowledgment*) com o mesmo número de sequência em um tópico dedicado; o ESP32, ao receber a confirmação, calcula o RTT como a diferença entre o instante de chegada e o de envio. Como ambos os instantes são lidos no mesmo relógio, a medição independe de qualquer sincronização externa. A partir do RTT, estima-se a latência de uma via como aproximadamente metade desse valor, sob a hipótese de caminho simétrico, de forma análoga à empregada pela ferramenta `ping`; ressalva-se, contudo, que o enlace empregado é assimétrico em banda (*download* e *upload* distintos), de modo que esse valor deve ser interpretado apenas como ordem de grandeza, e não como medida exata da latência em cada sentido.

Para esta caracterização, empregou-se uma variante do *firmware* configurada como publicador periódico simples, sem a máquina de estados e a captura por evento da versão final, de modo a maximizar o número de amostras coletadas em intervalo fixo. Foram conduzidas duas configurações de intervalo entre publicações: uma de 3 segundos, representativa do cenário operacional de coleta periódica, e outra de 1 segundo, na qual o rádio do enlace celular tende a permanecer ativo entre os envios. A conexão empregou um enlace 5G móvel (operadora Vivo; *download* de 380 Mbps, *upload* de 20 Mbps e latência de 38 ms até o servidor de referência nacional). Cada configuração coletou cerca de 300 amostras consecutivas.

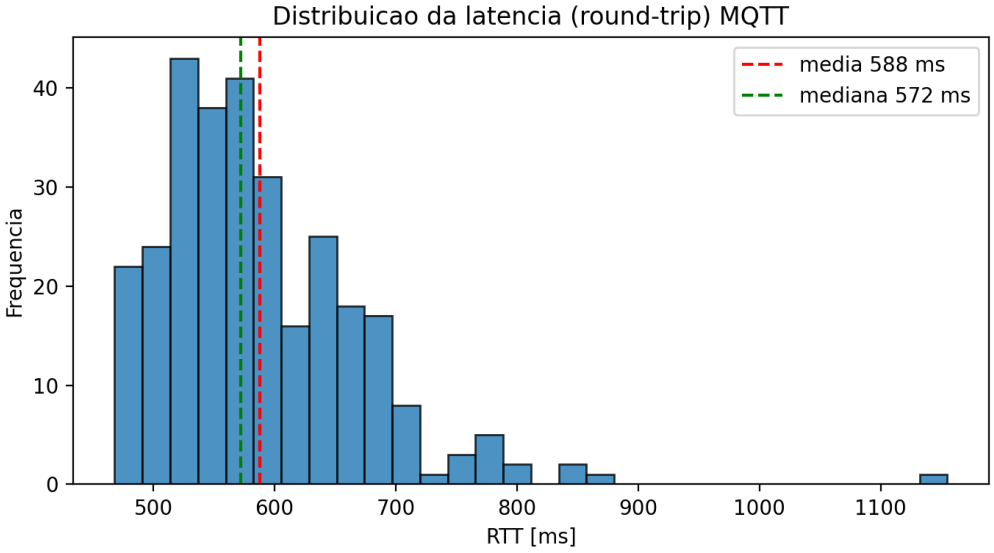
5.7.1.2 Resultados de Latência

A Tabela 4 resume as estatísticas obtidas nas duas configurações, e as Figuras 22 e 23 apresentam as distribuições correspondentes.

Tabela 4 – Estatísticas da latência da via MQTT (*round-trip*) nas duas configurações de intervalo.

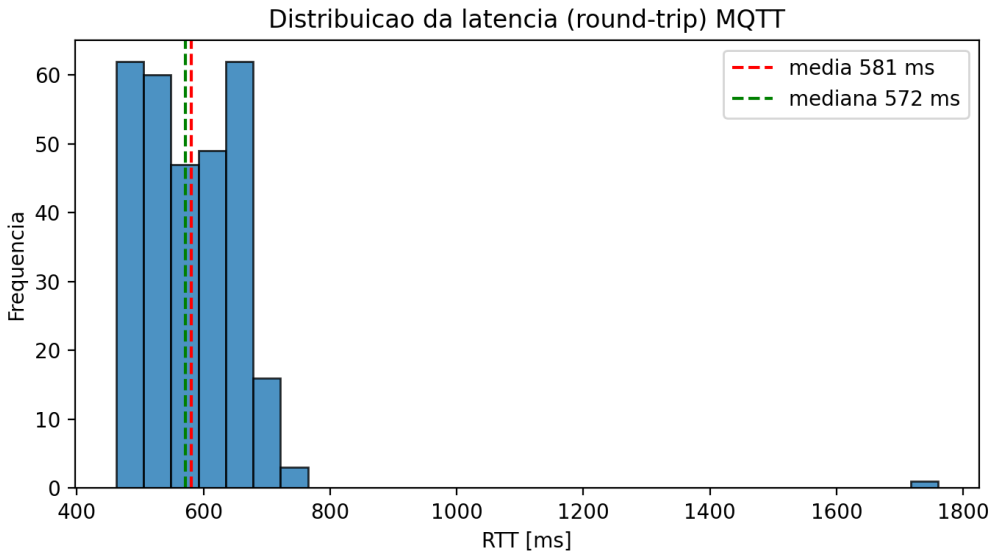
Métrica	**Intervalo 3 s**	**Intervalo 1 s**
Número de amostras	298	300
Taxa de entrega (*ACK*)	100%	100%
RTT mínimo (ms)	468	462
RTT mediano (ms)	572,0	571,5
RTT médio (ms)	587,8	580,8
RTT — percentil 95 (ms)	721,6	685,1
RTT — percentil 99 (ms)	838,1	723,0
RTT máximo (ms)	1155	1760
Desvio-padrão (ms)	82,7	97,2
Latência de uma via (estimada, ms)	≈286	≈286

Figura 22 – Distribuição da latência (*round-trip*) da via MQTT — intervalo de 3 segundos.



Fonte: Elaboração própria, 2026.

Figura 23 – Distribuição da latência (*round-trip*) da via MQTT — intervalo de 1 segundo.



Fonte: Elaboração própria, 2026.

Os resultados atestam, em primeiro lugar, a confiabilidade da via de comunicação: em ambas as configurações, a totalidade das mensagens foi confirmada, sem qualquer perda, evidência direta da garantia de entrega proporcionada pelo QoS 1. A latência mediana manteve-se praticamente constante entre as duas configurações (572,0 ms e 571,5 ms), resultado que indica que a latência típica é determinada pela distância geográfica até o broker, e não pelo intervalo de coleta nem pelo estado do rádio celular. As distribuições são assimétricas à direita, com média superior à mediana, comportamento característico de latência de rede: um piso bem definido acompanhado de uma cauda de amostras mais lentas.

O efeito do intervalo de coleta manifestou-se apenas nos percentis superiores: a configuração de 1 segundo, ao manter o rádio do enlace celular ativo entre os envios, reduziu a cauda da distribuição (percentil 99 de 838,1 ms para 723,0 ms). O valor máximo isolado de 1760 ms observado nessa configuração corresponde a um único evento atípico, característico da imprevisibilidade pontual de enlaces móveis (por exemplo, uma transferência de célula, ou *handover*), e não compromete a tendência central, robusta a observações isoladas. A diferença entre as médias das duas configurações (cerca de 7 ms) situa-se dentro da margem de incerteza amostral, de modo que, em tendência central, as duas configurações são estatisticamente equivalentes.

Em termos absolutos, uma latência mediana de aproximadamente 572 ms seria elevada para uma rede local, mas é adequada ao caso de uso proposto: a digitalização de uma coleta manual que, no procedimento original, consome segundos ou minutos por leitura. A via MQTT entrega cada mensagem ao assinante em menos de um segundo (ida e volta) mesmo com o broker hospedado em outro continente, folga compatível com o monitoramento operacional pretendido. O resultado aponta ainda uma otimização direta para trabalhos futuros: a adoção de um broker

hospedado em região geográfica mais próxima (por exemplo, a região `sa-east-1` da AWS, em São Paulo) reduziria a latência observada, ao eliminar as travessias transatlânticas.

5.7.1.3 Volume de Dados por Transmissão

Além da latência, caracterizou-se o volume de dados trafegado por publicação, parâmetro relevante para o dimensionamento do consumo de banda em enlaces móveis tarifados por volume. A Tabela 5 decompõe o tamanho de uma publicação na camada de aplicação e na camada de protocolo MQTT.

Tabela 5 – Composição do tamanho de uma publicação MQTT (QoS 1).

Componente	**Tamanho (bytes)**
Payload da aplicação (JSON)	99
Cabeçalho fixo MQTT	1
Campo de comprimento restante	1
Campo de comprimento do tópico	2
Nome do tópico (`maquinas/TORNO_01/medicao`)	25
Identificador do pacote (QoS 1)	2
Pacote PUBLISH completo	**130**

O *overhead* do protocolo MQTT sobre o *payload* da aplicação é de apenas 31 bytes, dos quais 25 correspondem ao nome do tópico. Ou seja, a maior parte do custo de protocolo decorre da extensão do identificador do tópico, e não do MQTT em si, o que evidencia a eficiência do protocolo para aplicações de telemetria. Considerando o intervalo operacional de 3 segundos, o volume médio trafegado na camada MQTT é da ordem de 43 bytes/s (cerca de 347 bits/s), valor desprezível frente à capacidade de qualquer enlace móvel contemporâneo, o que confirma que a largura de banda não constitui restrição para a aplicação. O *payload* medido nesta caracterização inclui um campo de número de sequência, necessário ao pareamento das confirmações no método de medição adotado; a versão final do *firmware* substitui esse campo pelo identificador único de evento (`event_id`) e demais campos descritos na Seção 5.4, mantendo o volume trafegado na mesma ordem de grandeza.

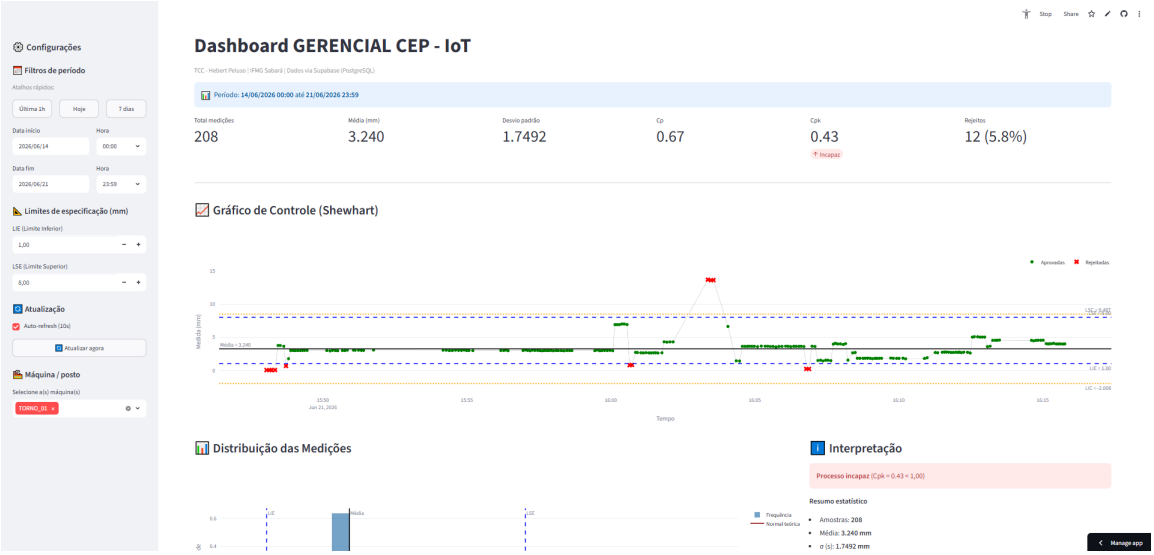
Não foram observadas medições corrompidas, perdidas ou gravadas em duplicidade no banco, o que confirma a integridade do sistema sob carga representativa do caso de uso industrial.

5.8 Painel de Visualização

Para consolidar a visualização dos dados coletados pelo sistema, foi desenvolvido um painel interativo em Python com a biblioteca Streamlit, voltada à construção de aplicações analíticas com interface acessível pelo navegador, sem exigir programação dedicada de interface. O painel consome diretamente a tabela `leituras` do banco de dados em nuvem Supabase por meio do *SDK* oficial (`supabase-py`) e apresenta, em tela única, os principais elementos de análise

estatística de processo: indicadores agregados, gráfico de controle de Shewhart, distribuição estatística das medições e tabela de não conformidades. A Figura 24 ilustra a interface.

Figura 24 – Painel de visualização das medições e indicadores estatísticos.



Fonte: Elaboração própria, 2026.

5.8.1 Demonstração das Funcionalidades

Esta seção percorre as funcionalidades do painel em operação e mostra que os dados digitalizados pelo sistema alimentam diretamente as ferramentas de controle estatístico de processo, sem qualquer pré-processamento além da consulta ao banco de dados em nuvem. É a demonstração prática do pressuposto central deste trabalho: a digitalização da coleta viabiliza o uso imediato das medições em análise estatística de qualidade.

5.8.1.1 Controles de Análise

A barra lateral concentra os controles que parametrizam toda a análise, conforme a Figura 25. Os filtros de período permitem selecionar a janela de interesse por data e hora, com granularidade de minuto, e contam com atalhos para as janelas mais usuais (*última hora*, *hoje* e *últimos sete dias*). Um seletor de máquina, baseado no identificador `machine_id`, permite restringir a análise a um posto de medição específico ou consolidar vários simultaneamente. Os limites de especificação inferior e superior (LIE e LSE) são ajustáveis em tempo real, e um controle de atualização habilita o recarregamento automático a cada dez segundos ou força a atualização imediata.

Figura 25 – Barra lateral com os controles de período, seleção de máquina e limites de especificação.

Fonte: Elaboração própria, 2026.

5.8.1.2 Indicadores Agregados

No topo da área principal, uma linha de seis indicadores sintetiza o estado do processo na janela selecionada (Figura 26): total de medições, média, desvio-padrão amostral, índice de capacidade C_p, índice de capacidade centrado C_{pk} e número de rejeitos com o respectivo percentual. O indicador de C_{pk} recebe automaticamente um rótulo qualitativo (*excelente*, *capaz*, *marginal* ou *incapaz*) com codificação por cor, oferecendo leitura imediata da capacidade do processo. Quando a janela contém menos de trinta amostras, o painel emite um aviso sobre a baixa confiabilidade estatística dos índices calculados.

Figura 26 – Linha de indicadores agregados (*KPIs*) calculados sobre a janela ativa.

Fonte: Elaboração própria, 2026.

5.8.1.3 Gráfico de Controle de Shewhart

O gráfico de controle (Figura 27) dispõe as medições ao longo do tempo e classifica cada ponto visualmente em aprovado (verde) ou rejeitado (vermelho), conforme esteja dentro ou fora

dos limites de especificação. Sobre a série são traçadas cinco linhas de referência: os limites de especificação (LSE e LIE), a média do processo e os limites de controle estatístico (LSC e LIC, definidos como $\bar{x} \pm 3\sigma$). O gráfico emprega renderização acelerada por GPU (*WebGL*, via `Scattergl`), o que preserva a fluidez mesmo com milhares de pontos, e é totalmente interativo, permitindo aproximação e inspeção individual de cada medição.

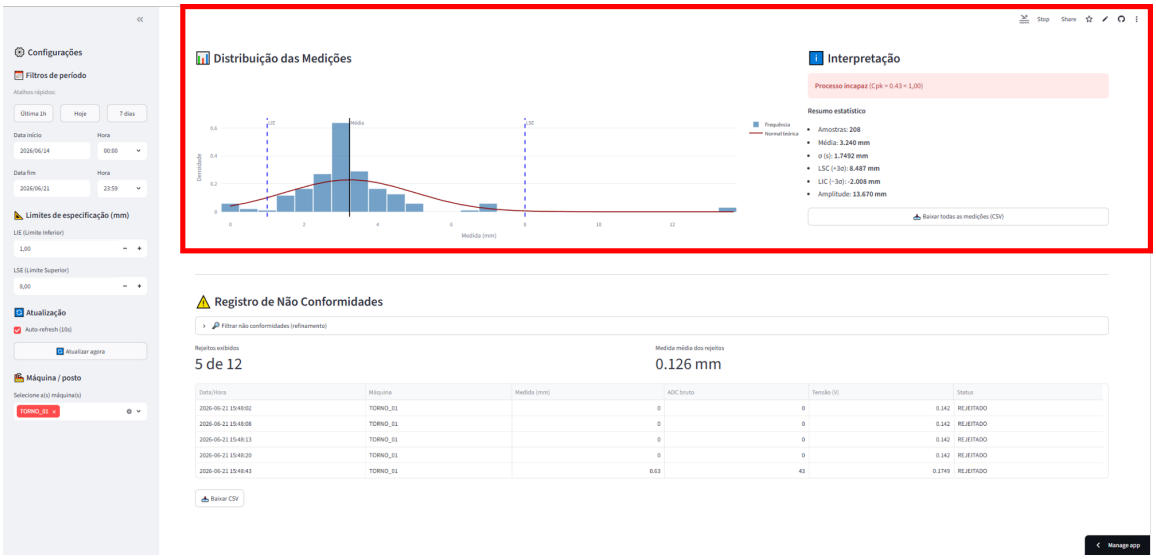
Figura 27 – Gráfico de controle de Shewhart com classificação por cor e limites de especificação e de controle.

Fonte: Elaboração própria, 2026.

5.8.1.4 Distribuição e Interpretação Automática

A distribuição das medições é apresentada em um histograma com a curva normal teórica sobreposta (Figura 28), permitindo a verificação visual da aderência à distribuição gaussiana, e marcada com os limites de especificação e a média. Ao lado, um painel de interpretação automática classifica o processo como *excelente* ($C_{pk} \geq 1{,}67$), *capaz* ($1{,}33 \leq C_{pk} < 1{,}67$), *marginal* ($1{,}00 \leq C_{pk} < 1{,}33$) ou *incapaz* ($C_{pk} < 1{,}00$), com codificação por cor consistente com a literatura de controle de qualidade (MONTGOMERY, 2012), acompanhado de um resumo estatístico (amostras, média, desvio-padrão, limites de controle e amplitude). Um botão dedicado exporta a totalidade das medições do período em formato CSV.

Figura 28 – Histograma com curva normal teórica e painel de interpretação automática da capacidade.

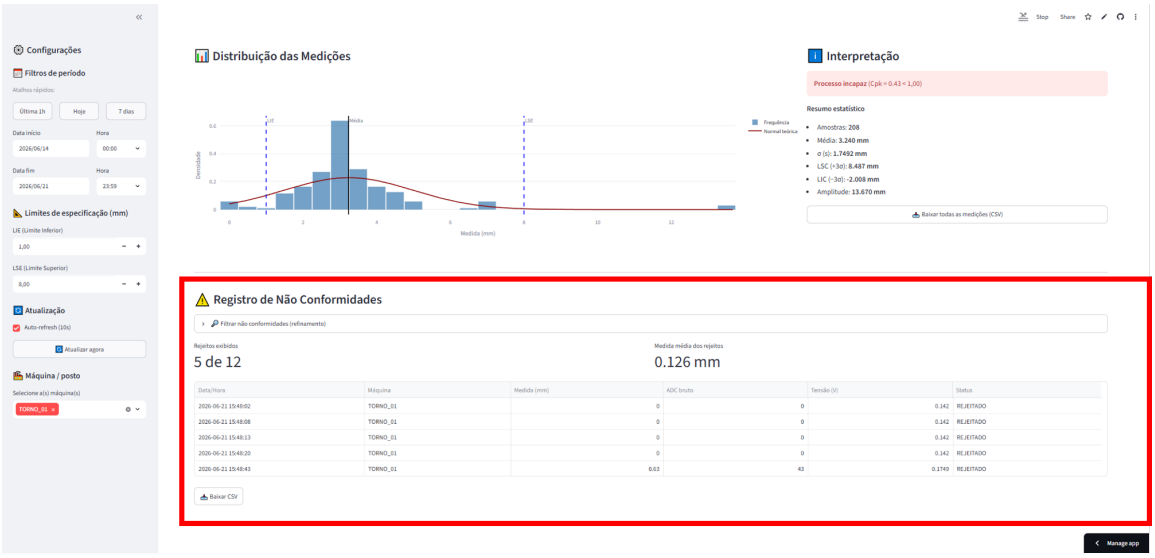


Fonte: Elaboração própria, 2026.

5.8.1.5 Registro de Não Conformidades

A última área do painel reúne as peças rejeitadas em uma tabela detalhada (Figura 29), com data e hora, máquina de origem, medida, valor bruto do ADC, tensão e situação. Um filtro de refinamento adicional permite restringir a lista por intervalo de data e hora e por faixa de medida, exibindo o número de rejeitos selecionados e sua medida média. Tanto a lista refinada de não conformidades quanto o conjunto completo de medições podem ser exportados em CSV, viabilizando a rastreabilidade e a análise externa. Quando não há rejeitos na janela, o painel sinaliza explicitamente a conformidade total do período.

Figura 29 – Registro de não conformidades com filtro de refinamento e exportação em CSV.



Fonte: Elaboração própria, 2026.

5.8.2 Integração com o Banco de Dados e Desempenho

A integração com o Supabase explora as capacidades de filtragem no servidor do PostgreSQL: a consulta é construída na interface fluente do *SDK* com cláusulas `gte` e `lte` sobre a coluna `timestamp`, traduzindo-se em uma cláusula `WHERE timestamp BETWEEN ...` executada no banco. Apenas as medições pertencentes à janela de tempo selecionada trafegam pela rede, o que reduz o consumo de banda e o tempo de resposta. Além disso, um *cache* local de dez segundos, configurado pelo decorador `@st.cache_data(ttl=10)` do Streamlit, suprime consultas redundantes em sessões interativas sucessivas, beneficiando tanto o consumo de cota quanto a fluidez da interface. Para acomodar sessões de coleta extensas, o limite de registros por consulta foi elevado de forma a contornar o teto padrão de mil linhas do PostgREST, com sinalização automática na interface caso esse teto seja atingido. Todos os instantes são convertidos para o fuso horário local (`America/Sao_Paulo`), assegurando a coerência temporal entre a coleta e a visualização.

5.8.3 Publicação em Nuvem e Modelo de Segurança

Para viabilizar o acesso público ao painel sem depender do equipamento local do desenvolvedor, a aplicação foi publicada na plataforma Streamlit Community Cloud, serviço gerenciado de hospedagem gratuita oferecido pela própria mantenedora da biblioteca. O processo de publicação consistiu na criação de um repositório público no GitHub com o código-fonte do painel, no arquivo `requirements.txt` com as dependências e na configuração das credenciais (*secrets*) no painel administrativo do Streamlit Cloud. A plataforma reconstrói automaticamente o ambiente de execução a partir das dependências declaradas e disponibiliza a aplicação em um endereço público.

A publicação em nuvem introduz, no entanto, um requisito de segurança que não existia na execução local: o código-fonte hospedado no repositório público não pode conter credenciais de acesso ao banco de dados. Para atender a esse requisito, foi implementada uma separação de responsabilidades entre o assinante e o painel, materializada pelo uso de duas chaves de API distintas, com permissões diferenciadas, conforme o Quadro 4.

Quadro 4 – Modelo de segurança aplicado ao banco de dados em nuvem.

Componente	**Local de execução**	**Chave utilizada**	**Permissões**
Assinante (ponte MQTT $\rightarrow$ banco)	Máquina local	*Service Role Key*	Leitura e escrita
Painel	Streamlit Community Cloud	*Publishable Key*	Apenas leitura

A restrição de permissões da chave do painel é aplicada na camada do banco por meio de políticas de *Row Level Security* (RLS) do PostgreSQL, recurso nativo do Supabase. Foram definidas duas políticas: a primeira (`leitura_publica`) autoriza a operação `SELECT` para qualquer cliente autenticado com a chave publicável; a segunda (`escrita_service_role`) autoriza

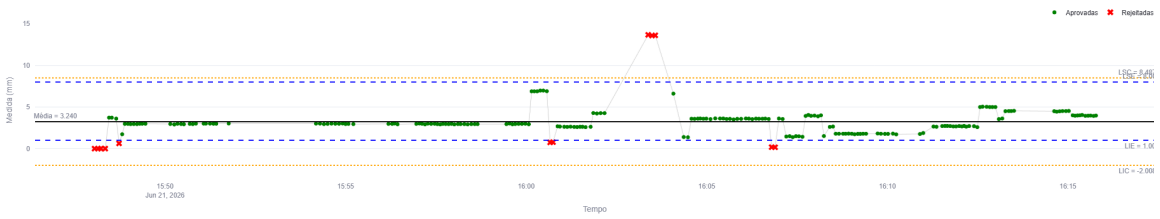
a operação `INSERT` exclusivamente para clientes autenticados com a chave de serviço. A consequência prática é que, ainda que a chave publicável seja exposta sem querer no repositório público ou no código-fonte servido ao navegador, um agente malicioso não consegue inserir, alterar ou remover registros da tabela: a integridade dos dados de medição permanece protegida pela camada de autorização do banco. A chave de serviço, com permissões plenas, é mantida exclusivamente na máquina local que executa o assinante e nunca é transmitida pela rede pública nem versionada em repositório.

A configuração das credenciais no Streamlit Cloud segue o padrão TOML e é gerenciada pela interface administrativa, separada do repositório público. Localmente, o mesmo padrão é replicado pelo arquivo `.streamlit/secrets.toml`, explicitamente listado no `.gitignore` do projeto para evitar exposição acidental.

5.8.4 Validação Operacional

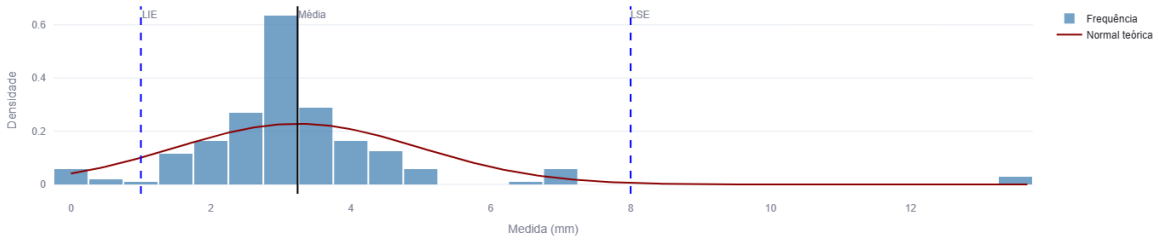
A operação de ponta a ponta foi validada submetendo o sistema integrado a uma sessão contínua de medições, por meio da manipulação do potenciômetro linear, com o painel acessado ao mesmo tempo em um navegador remoto. As Figuras 30, 31 e 32 apresentam o estado da interface durante essa validação, referente ao intervalo selecionado de 15:50 a 16:17 do dia 21/06/2026 (cerca de 27 minutos de medições contínuas). A Figura 30 mostra o gráfico de controle com a totalidade dos pontos coletados no intervalo, com transições visíveis entre estados de aprovação e rejeição conforme o cursor do potenciômetro percorria o curso; a Figura 31 apresenta o histograma das medições com a interpretação automática da capacidade; e a Figura 32 exibe a visão geral do painel, com os indicadores estatísticos calculados em tempo real sobre a janela ativa.

Figura 30 – Gráfico de controle de Shewhart durante a validação operacional.



Fonte: Elaboração própria, 2026.

Figura 31 – Histograma das medições e interpretação automática da capacidade durante a validação operacional.



Fonte: Elaboração própria, 2026.

Figura 32 – Painel em operação durante teste integrado.

Fonte: Elaboração própria, 2026.

A validação demonstra o funcionamento fluido do conjunto: as medições publicadas pelo ESP32 via MQTT se propagam pelo broker HiveMQ Cloud, são consumidas, validadas e gravadas pelo assinante no Supabase e ficam visíveis no painel público com latência compatível com o monitoramento operacional, sem intervenção manual em nenhuma etapa intermediária. O painel foi projetado para apoiar o operador e o supervisor de qualidade no acompanhamento contínuo do processo, na identificação de tendências fora de controle e na decisão corretiva baseada em dados quantitativos. A arquitetura também pode ser estendida com facilidade: o Supabase oferece o recurso *Realtime*, baseado em *WebSockets*, que permite a atualização automática do painel a cada inserção no banco, uma evolução natural para versões futuras do sistema.

Cabe delimitar o alcance desta validação. O sinal submetido ao sistema foi gerado pela manipulação manual do cursor do potenciômetro ao longo de seu curso, e não pela medição de um

processo produtivo sob controle estatístico. Em consequência, os índices de capacidade C_p/C_{pk} e a interpretação automática exibidos nas Figuras 30 a 32 têm caráter estritamente ilustrativo da funcionalidade da cadeia analítica, comprovam que as medições digitalizadas alimentam, sem qualquer pré-processamento, o cálculo dos indicadores e a geração das cartas de Shewhart, mas não caracterizam a capacidade de um processo industrial real, cuja avaliação pressupõe uma fonte de variação física genuína e a verificação das hipóteses estatísticas subjacentes. O objetivo desta etapa, coerente com o escopo definido na Seção 4.1, foi validar a integração de ponta a ponta do sistema, e não qualificar metrologicamente o instrumento. A caracterização do sistema em uma linha de produção real, com a consequente análise de capacidade de um processo efetivamente sob controle, constitui um desdobramento natural deste trabalho e fica indicada entre as propostas de continuidade do Capítulo 6.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho desenvolveu uma solução IoT orientada a eventos e de baixo custo para a digitalização da coleta de dados em postos de medição manual, com o objetivo de resolver problemas práticos da indústria, como os erros de transcrição e a falta de visibilidade dos dados de controle de qualidade. Esse objetivo foi atingido com a conclusão das principais etapas do projeto: a montagem e validação do circuito eletrônico de aquisição de sinal, o desenvolvimento do *firmware* orientado a eventos com mecanismo de armazenamento e encaminhamento para resiliência a falhas de rede, a implementação da cadeia de comunicação e persistência, e o desenvolvimento de um painel interativo publicado em nuvem, demonstrado em operação fim-a-fim sobre as medições digitalizadas, para a análise estatística de processo.

A arquitetura de ponta a ponta, que integra o microcontrolador ESP32, o broker HiveMQ Cloud, um assinante em Python e o banco de dados em nuvem Supabase (PostgreSQL gerenciado), mostrou-se tecnicamente viável e teve seus subsistemas validados experimentalmente — a resiliência e a latência da via de comunicação em variantes de *firmware* dedicadas a cada ensaio, e a operação integrada na configuração completa (Seção 5.8.4) —, atendendo aos requisitos de escalabilidade e acessibilidade que orientam este projeto. A escolha do Supabase, em substituição ao Firebase Cloud Firestore inicialmente previsto, foi motivada por evidências obtidas durante os testes integrados, conforme detalhado na Seção 5.2, e ilustra a capacidade da metodologia ágil adotada de absorver ajustes de arquitetura à luz dos resultados de validação, sem comprometer o cronograma. A publicação do painel no Streamlit Community Cloud, descrita na Seção 5.8, completou o ciclo de implementação ao tornar o sistema acessível remotamente, com um modelo de segurança baseado na separação de chaves de API e em políticas de *Row Level Security* no banco.

Conclui-se que a integração das tecnologias selecionadas constitui um caminho viável para modernizar a coleta de dados em ambientes industriais, a um custo acessível e apoiado em serviços de nuvem de planos gratuitos. A contribuição central deste trabalho reside na arquitetura de referência fim-a-fim, e não em valores particulares de capacidade de processo: ao assegurar a integridade e a rastreabilidade das medições e ao disponibilizá-las de imediato, sem pré-processamento, para a cadeia de análise estatística (C_p/C_{pk} e cartas de controle de Shewhart), o sistema substitui o registro manual em papel por um fluxo digital auditável. Cabe ressaltar que a validação operacional (Seção 5.8.4) foi conduzida com um sinal gerado pela manipulação manual do transdutor; os índices de capacidade ali exibidos têm, portanto, caráter ilustrativo da funcionalidade da cadeia analítica e não caracterizam um processo produtivo real. A caracterização metrológica do sistema em uma linha de produção, com a consequente análise de capacidade de um processo sob controle, constitui um desdobramento natural e fica indicada como trabalho futuro. Sobre essa base, abre-se espaço para a evolução rumo a sistemas mais amplos de gestão da qualidade.

Para a continuidade do projeto, sugerem-se melhorias em três frentes. Na frente de hard-

ware, recomenda-se substituir o transdutor potenciométrico por sensores sem desgaste mecânico, como fitas magnéticas absolutas ou *encoders* ópticos lineares, o que eliminaria o desgaste por atrito e ampliaria a vida útil do instrumento. Na frente de análise, propõe-se incorporar algoritmos de aprendizado de máquina para a análise preditiva da deriva do processo e a detecção precoce de causas especiais de variação, complementando os indicadores clássicos do CEP. Na frente de visualização e operação, sugere-se adotar o recurso *Realtime* do Supabase para a atualização automática do painel, configurar alertas por *e-mail* ou aplicativo móvel em caso de violação dos limites estatísticos e expandir a arquitetura para integrar vários instrumentos de medição em uma plataforma centralizada de gerenciamento de qualidade, eventualmente migrando a camada de visualização para ferramentas industriais já estabelecidas, como o Grafana. Ainda nessa frente, observa-se que a confirmação por duplo-clique, embora proteja o sistema contra acionamentos acidentais, permite que uma medição não confirmada seja descartada sem registro; o registro auditável também desses descartes constitui uma extensão natural para fechar por completo o laço de rastreabilidade contra a eventual omissão de leituras desfavoráveis, limitação que motivou parte da justificativa deste trabalho.

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**, v. 4, n. 1, p. 57–86, 2020. Citado 2 vezes nas páginas 22 e 25.

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols and applications. **IEEE Communications Surveys amp Tutorials**, v. 17, p. Fourthquarter 2015, 11 2015. Citado 3 vezes nas páginas 27, 29 e 41.

ALI, Z.; ALI, H.; BADAWY, M. Internet of things (iot): Definitions, challenges, and recent research directions. **International Journal of Computer Applications**, v. 128, p. 975–8887, 10 2015. Citado na página 21.

ALVES, T. d. S. **Framework para controle de sistemas industriais via MQTT**. Uberlândia, MG, 2022. Citado 2 vezes nas páginas 22 e 25.

CIRANI, S.; FERRARI, G.; PICONE, M.; VELTRI, L. **Internet of Things: Architectures, Protocols and Standards**. [S.l.]: John Wiley Sons, 2018. ISBN 9781119359715. Citado 2 vezes nas páginas 18 e 24.

CODD, E. F. A relational model of data for large shared data banks. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 13, n. 6, p. 377–387, jun. 1970. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/362384.362685>. Citado na página 33.

COSTA, A. F. B.; EPPRECHT, E. K.; CARPINETTI, L. **Controle Estatístico de Qualidade**. 2. ed. São Paulo, SP: Atlas, 2010. Citado na página 33.

Espressif Systems. **ESP32 Series Datasheet**. [S.l.], 2025. Version 4.9. Disponível em: <https://www.espressif.com/documentation/esp32_datasheet_en.pdf>. Citado 3 vezes nas páginas 26, 27 e 31.

GANSSLE, J. G. **A Guide to Debouncing**. 2008. The Ganssle Group. Disponível em: <http://www.ganssle.com/debouncing.pdf>. Acesso em: 15 jun. 2026. Citado na página 30.

GOH, C. C. *et al.* Iv-aqms: Http and mqtt protocol from a realistic testbed. In: **2019 IEEE International Conference on Sensors and Nanotechnology**. [S.l.: s.n.], 2019. p. 1–4. Citado 2 vezes nas páginas 22 e 25.

GRGIć, K.; ŠPEH, I.; HEđI, I. A web-based iot solution for monitoring data using mqtt protocol. In: **2016 International Conference on Smart Systems and Technologies (SST)**. [S.l.: s.n.], 2016. p. 249–253. Citado 2 vezes nas páginas 22 e 25.

GUPTA, P.; M, I. A survey of application layer protocols for internet of things. In: **2021 International Conference on Communication information and Computing Technology (ICCICT)**. [S.l.: s.n.], 2021. p. 1–6. Citado 2 vezes nas páginas 27 e 29.

HARRIS, C. R. *et al.* Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>. Citado na página 31.

HERMANN, M.; PENTEK, T.; OTTO, B. Design principles for industrie 4.0 scenarios. In: **2016 49th Hawaii International Conference on System Sciences (HICSS)**. [S.l.: s.n.], 2016. p. 3928–3937. Citado na página 21.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science Engineering**, v. 9, n. 3, p. 90–95, 2007. Citado na página 31.

HWANG, G.-T.; HWANG, K. Design and implementation of dashboard author and viewer for iot systems based on mqtt. In: . [s.n.], 2018. Disponível em: <https://api.semanticscholar.org/CorpusID:224359017>. Citado 2 vezes nas páginas 23 e 25.

KHALIL, K.; ELGAZZAR, K.; SELIEM, M.; BAYOUMI, M. Resource discovery techniques in the internet of things: A review. **Internet of Things**, v. 12, p. 100293, 2020. ISSN 2542-6605. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2542660520301256>. Citado na página 27.

KOLBAN, N. **Kolban's Book on ESP32**. [S.l.]: Self-Published, 2018. O autor descreve este trabalho como uma versão compilada e polida de suas anotações pessoais. Citado na página 26.

LASI, H. Industrial intelligence - a business intelligence-based approach to enhance manufacturing engineering in industrial companies. **Procedia CIRP**, v. 12, p. 384–389, 2013. ISSN 2212-8271. Eighth CIRP Conference on Intelligent Computation in Manufacturing Engineering. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2212827113007075>. Citado na página 21.

LEE, E. A.; SESHIA, S. A. **Introduction to Embedded Systems: A Cyber-Physical Systems Approach**. 2nd. ed. Cambridge, MA: MIT Press, 2017. ISBN 978-0262533812. Citado 2 vezes nas páginas 30 e 38.

LIMA, P. R. S.; ALBUQUERQUE, J. P. M.; RODRIGUES, A. B. M.; LIMA, D. L.; VASQUES, K. B. P. P. Internet of things e indÚstria 4.0: RevisÃo sistemÁtica bibliomÉtrica. **Revista Contemporânea**, v. 3, n. 5, p. 4424–4436, maio 2023. Disponível em: <https://ojs.revistacontemporanea.com/ojs/index.php/home/article/view/794>. Citado na página 21.

MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). **Proceedings of the 9th Python in Science Conference**. [S.l.: s.n.], 2010. p. 56 – 61. Citado na página 31.

MONTGOMERY, D. **Statistical Quality Control, 7th Edition**. Wiley, 2012. ISBN 9781118214688. Disponível em: <https://books.google.com.br/books?id=RgQcAAAAQBAJ>. Citado 4 vezes nas páginas 18, 32, 33 e 65.

NAIK, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: **2017 IEEE International Systems Engineering Symposium (ISSE)**. [S.l.]: IEEE, 2017. p. 1–7. Citado na página 21.

NIMODIYA, A.; AJANKAR, S. A review on internet of things. **International Journal of Advanced Research in Science, Communication and Technology**, p. 135–144, 01 2022. Citado na página 21.

RESCORLA, E. **The Transport Layer Security (TLS) Protocol Version 1.3**. [S.l.], 2018. Disponível em: <https://datatracker.ietf.org/doc/html/rfc8446>. Citado 2 vezes nas páginas 27 e 37.

SADALAGE, P. J.; FOWLER, M. **NoSQL Distilled**. [S.l.]: Addison-Wesley, 2012. Citado 2 vezes nas páginas 33 e 34.

SCHWAB, K. **The fourth industrial revolution**. [S.l.]: Portfolio Penguin, 2017. Citado na página 18.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database System Concepts**. 6th. ed. New York: McGraw-Hill, 2010. Citado 2 vezes nas páginas 33 e 34.

SILVA, J. G. da. Dispositivo para conexão a redes iot para indústria 4.0. **Universidade Tecnológica Federal do Paraná**, p. 1–62, 2019. Citado na página 18.

Supabase Inc. **Supabase Documentation**. 2026. Acesso em: 28 de julho de 2026. Disponível em: <https://supabase.com/docs>. Citado na página 34.

WOLNIAK, R.; GREBSKI, W. The usage of statistical process control (SPC) in industry 4.0 conditions. **Scientific Papers of Silesian University of Technology. Organization and Management Series**, n. 190, p. 649–668, 2024. Citado na página 24.

YASSEIN, M. B.; SHATNAWI, M. Q.; ALJWARNEH, S.; AL-HATMI, R. Internet of things: Survey and open issues of mqtt protocol. In: IEEE. **2017 international conference on engineering & MIS (ICEMIS)**. [S.l.], 2017. p. 1–6. Citado 2 vezes nas páginas 27 e 28.

7 FIRMWARE DO MEDIDOR (ESP32)

O firmware foi dividido em dois arquivos: `config.h`, que isola os parâmetros e as credenciais de cada implantação (Apêndice 7.1), e `medidor_iot.ino`, que contém a máquina de estados e a lógica de aquisição, publicação e armazenamento local (Apêndice 7.2). As credenciais reais foram substituídas por marcadores genéricos.

Listagem 7.1 – Arquivo de configuração `config.h`.

```
// config.h - credenciais e parametros do firmware
// TCC IFMG Sabara - Hebert Peluso

#ifndef CONFIG_H
#define CONFIG_H

// posto / identificacao -> mudar a cada implantacao
// machine_id e fixo por dispositivo, os topicos saem dele
#define MACHINE_ID        "TORNO_01"        // <<<< trocar pelo nome do posto
#define FIRMWARE_VERSION  "1.0.0"

// wifi
#define WIFI_SSID      "ADD_NOME_REDE_DO_WIFI"
#define WIFI_PASSWORD  "ADD_SENHA_REDE_DO_WIFI"

// mqtt - HiveMQ Cloud
#define MQTT_BROKER    "20c9404954c44cd4b27fdab38f999e70.s1.eu.hivemq.cloud"
#define MQTT_PORT      8883
#define MQTT_USER      "ADD_USUARIO_MQTT"
#define MQTT_PASSWORD  "ADD_SENHA_MQTT"

// client_id fixo pra sessao persistir no broker
#define MQTT_CLIENT_ID "ESP32_" MACHINE_ID

// topicos: maquinas/<id>/medicao  e  .../status
#define MQTT_TOPIC_MEDICAO  "maquinas/" MACHINE_ID "/medicao"
#define MQTT_TOPIC_STATUS   "maquinas/" MACHINE_ID "/status"

// pinos
#define PIN_POTENCIOMETRO  34   // ADC1_CH6
#define PIN_BOTAO          32   // INPUT_PULLUP, aciona em LOW
#define PIN_LED_R          25   // RED LEDD
#define PIN_LED_G          26   // GREEN LEDD
#define PIN_LED_B          27   // WHITE LED

// aquisicao / tempos
#define JANELA_AMOSTRAS      50     // amostras por medicao
```

```
#define JANELA_INTERVALO_MS   10      // 50 x 10ms = 500ms (~100Hz)
#define DEBOUNCE_MS           50      // repique do botao
#define TIMEOUT_CONFIRMA_MS   5000    // tempo pro 2o clique (confirmar)
#define STATUS_INTERVALO_MS   60000   // heartbeat de status a cada 60s

// buffer NVS (store-and-forward)
#define BUFFER_MAX   100    // buffer circular na flash

#endif // CONFIG_H
```

Listagem 7.2 – Firmware principal `medidor_iot.ino`.

```
// medidor_iot.ino
// firmware do medidor - captura por evento + buffer na NVS pra nao perder
    dado offline
// TCC IFMG Sabara - Hebert Peluso

#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <MQTT.h>
#include <Preferences.h>
#include <esp_system.h>
#include <math.h>
#include "config.h"

// --- maquina de estados ---
enum Estado {
  AGUARDANDO,
  MEDINDO,
  AGUARDANDO_CONFIRMACAO,
  PUBLICANDO_DIRETO,
  STORING,
  DESCARTANDO,
  DRENANDO,
  ERRO_BUFFER_CHEIO
};
Estado estado = AGUARDANDO;

// --- struct da medicao ---
struct Medicao {
  uint32_t event_id;
  uint32_t uptime_ms;
  int      adc_raw_mean;
  float    voltage_mean_mv;
  float    voltage_stddev_mv;
  int      samples_count;
  int      window_ms;
  int      rssi_dbm;
```

```
};
Medicao medicaoAtual;

// --- buffer na NVS ---
// deixo o Preferences aberto o tempo todo pra nao ficar dando begin/end (trava o FreeRTOS)
Preferences prefs;
int nvsHead = 0, nvsTail = 0, nvsCount = 0;

// salvo head/tail/count como 1 blob so -> grava num commit unico.
// antes eu fazia 3 putInt separados e se cortasse energia no meio
// o count/tail ficavam bugados e perdia medicao.
struct BufferEstado {
  int head;
  int tail;
  int count;
};

uint32_t cont_publicadas = 0;
uint32_t cont_reconexoes = 0;

WiFiClientSecure espClient;
MQTTClient       mqttClient(1024);

unsigned long t_confirma = 0;
unsigned long t_descarte = 0;
unsigned long t_status   = 0;

int  botaoEstavel         = HIGH;
int  botaoLeituraAnterior = HIGH;
unsigned long t_debounce  = 0;

// --- NVS (sem begin/end, ja abri no setup) ---
void nvsCarregarEstado() {
  BufferEstado e = { 0, 0, 0 };
  prefs.getBytes("estado", &e, sizeof(e));   // se nao tiver nada, fica zerado mesmo
  nvsHead  = e.head;
  nvsTail  = e.tail;
  nvsCount = e.count;
  Serial.printf("[NVS] Estado: head=%d tail=%d count=%d\n", nvsHead, nvsTail, nvsCount);
}

void nvsSalvarEstado() {
  BufferEstado e = { nvsHead, nvsTail, nvsCount };
  prefs.putBytes("estado", &e, sizeof(e));   // grava tudo de uma vez
```

```
80 }
81
82 // so retorna true se realmente escreveu os bytes na flash
83 bool nvsGravarMedicao(int idx, const Medicao &m) {
84   char chave[8]; snprintf(chave, sizeof(chave), "m%d", idx);
85   size_t escrito = prefs.putBytes(chave, &m, sizeof(Medicao));
86   return escrito == sizeof(Medicao);
87 }
88
89 bool nvsLerMedicao(int idx, Medicao &m) {
90   char chave[8]; snprintf(chave, sizeof(chave), "m%d", idx);
91   size_t len = prefs.getBytes(chave, &m, sizeof(Medicao));
92   return len == sizeof(Medicao);
93 }
94
95 bool bufferCheio() { return nvsCount == BUFFER_MAX; }
96 bool bufferVazio() { return nvsCount == 0; }
97
98 // grava o dado primeiro, e so mexe no tail/count se a gravacao deu certo.
99 // assim se o putBytes falhar (NVS cheia) nao fica item fantasma na
      contagem.
100 bool bufferEnfileirar(const Medicao &m) {
101   if (!nvsGravarMedicao(nvsTail, m)) {
102     Serial.println("[NVS][ERRO] Gravacao falhou (NVS cheia/flash?). Dado
        NAO enfileirado.");
103     return false;
104   }
105   nvsTail = (nvsTail + 1) % BUFFER_MAX;
106   nvsCount++;
107   nvsSalvarEstado();
108   Serial.printf("[NVS] Armazenada. Buffer: %d/%d\n", nvsCount, BUFFER_MAX);
109   return true;
110 }
111
112 // no boot confere se todo item contado no count ainda da pra ler
113 void nvsVerificarIntegridade() {
114   Serial.println("
        ------------------------------------------------------------");
115   if (nvsCount <= 0) {
116     Serial.println("[NVS] Boot: buffer VAZIO (count=0).");
117     Serial.println("
          ------------------------------------------------------------");
118     return;
119   }
120   int idx = nvsHead, legiveis = 0;
121   for (int i = 0; i < nvsCount; i++) {
122     Medicao tmp;
```

```
123     if (nvsLerMedicao(idx, tmp)) legiveis++;
124     idx = (idx + 1) % BUFFER_MAX;
125   }
126   Serial.printf("[NVS] Boot: count=%d | legiveis=%d/%d\n", nvsCount,
        legiveis, nvsCount);
127   if (legiveis != nvsCount)
128     Serial.println("[NVS][ALERTA] Itens contabilizados porem ILEGIVEIS no
          flash!");
129   else
130     Serial.println("[NVS] OK: todos os pendentes sobreviveram ao power-
          cycle.");
131   Serial.println("
        ------------------------------------------------------------");
132 }
133
134 // --- LEDs (3 leds separados, nao e modulo RGB) ---
135 void setLED(bool r, bool g, bool b) {
136   digitalWrite(PIN_LED_R, r);
137   digitalWrite(PIN_LED_G, g);
138   digitalWrite(PIN_LED_B, b);
139 }
140
141 void pulso(bool r, bool g, bool b, int ms) {
142   setLED(r, g, b);
143   delay(ms);
144   setLED(LOW, LOW, LOW);
145 }
146
147 // pisca a cor (r,g,b) por 'duracao_ms', trocando a cada 'intervalo_ms'.
148 // uso no verde de confirmacao de publicado.
149 void piscar(bool r, bool g, bool b, int duracao_ms, int intervalo_ms) {
150   unsigned long fim = millis() + duracao_ms;
151   while ((long)(fim - millis()) > 0) {
152     setLED(r, g, b);
153     delay(intervalo_ms);
154     setLED(LOW, LOW, LOW);
155     delay(intervalo_ms);
156   }
157   setLED(LOW, LOW, LOW);
158 }
159
160 void atualizarLED() {
161   bool online = mqttClient.connected();
162   unsigned long ms = millis();
163   switch (estado) {
164     case AGUARDANDO:
165       if (online) setLED(LOW, HIGH, LOW);                    // verde =
```

```
        online
166       else { bool on = (ms / 500) % 2; setLED(on, LOW, LOW); }   //
            vermelho piscando = offline
167       break;
168     case AGUARDANDO_CONFIRMACAO:
169       { bool on = (ms / 150) % 2; setLED(LOW, LOW, on); }   // branco
            piscando
170       break;
171     case DRENANDO:
172       { bool on = (ms / 150) % 2; setLED(LOW, LOW, on); }
173       break;
174     case DESCARTANDO:
175       { bool on = (ms / 150) % 2; setLED(on, LOW, LOW); }
176       break;
177     case ERRO_BUFFER_CHEIO:
178       setLED(HIGH, LOW, LOW);
179       break;
180     default:
181       break;
182   }
183 }
184
185 // --- aquisicao: janela de 50 amostras ---
186 void coletarJanela() {
187   setLED(LOW, LOW, HIGH);   // branco fixo = coletando
188   long  somaRaw = 0;
189   float somaMv  = 0.0f, somaMv2 = 0.0f;
190   unsigned long inicio = millis();
191
192   for (int i = 0; i < JANELA_AMOSTRAS; i++) {
193     int raw = analogRead(PIN_POTENCIOMETRO);
194     int mv  = analogReadMilliVolts(PIN_POTENCIOMETRO);
195     somaRaw += raw;
196     somaMv  += mv;
197     somaMv2 += (float)mv * (float)mv;
198     delay(JANELA_INTERVALO_MS);
199   }
200
201   float n        = (float)JANELA_AMOSTRAS;
202   float mediaMv  = somaMv / n;
203   float variancia = (somaMv2 - n * mediaMv * mediaMv) / (n - 1.0f);
204   if (variancia < 0) variancia = 0;
205
206   medicaoAtual.event_id          = esp_random();
207   medicaoAtual.uptime_ms         = millis();
208   medicaoAtual.adc_raw_mean      = (int)(somaRaw / JANELA_AMOSTRAS);
209   medicaoAtual.voltage_mean_mv   = mediaMv;
```

```
210   medicaoAtual.voltage_stddev_mv = sqrtf(variancia);
211   medicaoAtual.samples_count     = JANELA_AMOSTRAS;
212   medicaoAtual.window_ms         = (int)(millis() - inicio);
213   medicaoAtual.rssi_dbm          = WiFi.RSSI();
214
215   setLED(LOW, LOW, LOW);
216   Serial.printf("[MEDIR] raw_mean=%d  V_mean=%.2f mV  std=%.2f mV\n",
217                 medicaoAtual.adc_raw_mean, medicaoAtual.voltage_mean_mv,
218                 medicaoAtual.voltage_stddev_mv);
219 }
220
221 // --- publicacao MQTT ---
222 bool publicarMedicao(const Medicao &m) {
223   if (WiFi.status() != WL_CONNECTED || !mqttClient.connected()) return
        false;
224
225   char json[384];
226   snprintf(json, sizeof(json),
227     "{\"event_id\":%u,\"machine_id\":\"%s\",\"firmware_version\":\"%s\","
228     "\"uptime_ms\":%u,"
229     "\"measurement\":{\"voltage_mean_mv\":%.2f,\"voltage_stddev_mv\":%.2f,"
230     "\"adc_raw_mean\":%d,\"samples_count\":%d,\"window_ms\":%d},"
231     "\"metadata\":{\"rssi_dbm\":%d}}",
232     m.event_id, MACHINE_ID, FIRMWARE_VERSION,
233     m.uptime_ms,
234     m.voltage_mean_mv, m.voltage_stddev_mv,
235     m.adc_raw_mean, m.samples_count, m.window_ms,
236     m.rssi_dbm);
237
238   return mqttClient.publish(MQTT_TOPIC_MEDICAO, json, false, 1);
239 }
240
241 void publicarStatus() {
242   if (WiFi.status() != WL_CONNECTED || !mqttClient.connected()) return;
243   char json[256];
244   snprintf(json, sizeof(json),
245     "{\"machine_id\":\"%s\",\"total_publicadas\":%u,"
246     "\"armazenadas_local\":%d,\"reconexoes\":%u}",
247     MACHINE_ID, cont_publicadas, nvsCount, cont_reconexoes);
248   mqttClient.publish(MQTT_TOPIC_STATUS, json, true, 1);
249 }
250
251 // --- botao com debounce ---
252 bool lerBotaoClique() {
253   bool clique = false;
254   int leitura = digitalRead(PIN_BOTAO);
255   if (leitura != botaoLeituraAnterior) t_debounce = millis();
```

```
256
257   if (millis() - t_debounce > DEBOUNCE_MS) {
258     if (leitura != botaoEstavel) {
259       botaoEstavel = leitura;
260       if (botaoEstavel == LOW) clique = true;
261     }
262   }
263   botaoLeituraAnterior = leitura;
264   return clique;
265 }
266
267 // --- wifi / mqtt ---
268 unsigned long t_mqtt_retry         = 0;
269 unsigned long t_wifi_perda         = 0;
270 bool          wifi_conectado_antes = false;
271
272 void manterWiFi() {
273   bool conectado = (WiFi.status() == WL_CONNECTED);
274
275   if (conectado) {
276     if (!wifi_conectado_antes) {
277       Serial.println("[WIFI] Conectado! IP: " + WiFi.localIP().toString());
278       wifi_conectado_antes = true;
279       espClient.stop();      // mata socket TLS antigo
280       t_mqtt_retry = 0;      // libera reconexao MQTT imediata
281     }
282   } else {
283     if (wifi_conectado_antes) {
284       Serial.println("[WIFI] Conexao perdida - operando OFFLINE (NVS ativo)
              ");
285       wifi_conectado_antes = false;
286       espClient.stop();
287       t_wifi_perda = millis();
288     }
289     // as vezes o auto-reconnect do ESP32 trava. se passou muito tempo
290     // sem voltar, eu forco um begin() na mao.
291     if (t_wifi_perda != 0 && millis() - t_wifi_perda > 15000) {
292       Serial.println("[WIFI] Reconexao travada - forcando WiFi.begin()");
293       WiFi.disconnect();
294       WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
295       t_wifi_perda = millis();
296     }
297   }
298 }
299
300 void manterMQTT() {
301   if (WiFi.status() != WL_CONNECTED || mqttClient.connected()) return;
```

```
302   if (millis() - t_mqtt_retry < 8000) return;
303   t_mqtt_retry = millis();
304
305   // importante: quando a internet cai mas o wifi continua conectado, o
          socket
306   // TLS antigo fica "preso" e toda reconexao falha - ai nunca percebe que
          a net
307   // voltou. dar stop() antes de cada tentativa resolve.
308   espClient.stop();
309
310   Serial.print("[MQTT] Conectando ao HiveMQ...");
311   bool ok = mqttClient.connect(MQTT_CLIENT_ID, MQTT_USER, MQTT_PASSWORD);
312
313   if (ok) {
314     cont_reconexoes++;
315     Serial.println(" CONECTADO!");
316     publicarStatus();
317   } else {
318     Serial.printf(" falha (lastError=%d, returnCode=%d)\n",
319                   mqttClient.lastError(), mqttClient.returnCode());
320   }
321 }
322
323 // --- setup ---
324 void setup() {
325   Serial.begin(115200);
326   delay(500);
327
328   pinMode(PIN_BOTAO, INPUT_PULLUP);
329   pinMode(PIN_LED_R, OUTPUT);
330   pinMode(PIN_LED_G, OUTPUT);
331   pinMode(PIN_LED_B, OUTPUT);
332   setLED(LOW, LOW, LOW);
333
334   analogReadResolution(12);
335   analogSetAttenuation(ADC_11db);
336
337   espClient.setInsecure();
338   espClient.setHandshakeTimeout(3);
339   espClient.setTimeout(3000);
340
341   // abre a NVS uma vez so e deixa aberta
342   prefs.begin("paqbuf", false);
343   nvsCarregarEstado();
344   nvsVerificarIntegridade();                    // so log de diagnostico
345   if (nvsCount > 0)
346     Serial.printf("[NVS] %d medicao(oes) pendente(s) recuperada(s)!\n",
```

```
        nvsCount);
347
348   WiFi.mode(WIFI_STA);
349   WiFi.setAutoReconnect(true);
350   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
351
352   mqttClient.begin(MQTT_BROKER, MQTT_PORT, espClient);
353   mqttClient.setKeepAlive(30);          // detecta link morto mais rapido (
        era 60 no primeiro test)
354   mqttClient.setTimeout(5000);
355   mqttClient.setCleanSession(false);
356
357   t_status = millis();
358   Serial.printf("[BOOT] Pronto. machine_id=%s topico=%s\n", MACHINE_ID,
        MQTT_TOPIC_MEDICAO);
359 }
360
361 // --- loop principal (maquina de estados) ---
362 void loop() {
363   manterWiFi();
364   manterMQTT();
365   if (mqttClient.connected()) mqttClient.loop();
366
367   bool clique = lerBotaoClique();
368
369   if (millis() - t_status >= STATUS_INTERVALO_MS) {
370     t_status = millis();
371     publicarStatus();
372   }
373
374   switch (estado) {
375
376     case AGUARDANDO:
377       if (clique) {
378         estado = MEDINDO;
379       } else if (WiFi.status() == WL_CONNECTED && mqttClient.connected() &&
            !bufferVazio()) {
380         estado = DRENANDO;
381       }
382       break;
383
384     case MEDINDO:
385       coletarJanela();
386       t_confirma = millis();
387       estado = AGUARDANDO_CONFIRMACAO;
388       break;
389
```

```
390     case AGUARDANDO_CONFIRMACAO:
391       if (clique) {
392         // tem rede? manda direto. nao tem? guarda na NVS.
393         if (WiFi.status() == WL_CONNECTED && mqttClient.connected()) {
394           estado = PUBLICANDO_DIRETO;
395         } else {
396           Serial.println("[AVISO] Sem rede no momento do clique. Indo para
                NVS.");
397           estado = STORING;
398         }
399       } else if (millis() - t_confirma > TIMEOUT_CONFIRMA_MS) {
400         t_descarte = millis();
401         estado = DESCARTANDO;
402       }
403       break;
404
405     case PUBLICANDO_DIRETO:
406       // tenta mandar pra nuvem sem gravar na flash
407       if (publicarMedicao(medicaoAtual)) {
408         cont_publicadas++;
409         piscar(LOW, HIGH, LOW, 2000, 150); // verde 2s = publicado
410         Serial.printf("[PUBLICA] event_id=%u enviado diretamente! (Poupou
              NVS)\n", medicaoAtual.event_id);
411         estado = AGUARDANDO;
412       } else {
413         // se a net caiu sem avisar, cai aqui depois do timeout
414         Serial.println("[ERRO MQTT] Falha no envio direto! Acionando
              fallback para NVS...");
415         estado = STORING;
416       }
417       break;
418
419     case STORING:
420       if (bufferCheio()) {
421         estado = ERRO_BUFFER_CHEIO;
422       } else if (bufferEnfileirar(medicaoAtual)) {
423         piscar(LOW, LOW, HIGH, 2000, 150); // branco 2s = salvo no buffer
424         estado = AGUARDANDO;
425       } else {
426         // nao conseguiu gravar: nao marca como salvo, sinaliza erro
427         Serial.println("[STORING][ERRO] Nao foi possivel persistir a
              medicao.");
428         estado = ERRO_BUFFER_CHEIO;
429       }
430       break;
431
432     case DESCARTANDO:
```

```
433       if (millis() - t_descarte > 5000) {   // vermelho pisca 5s
434         Serial.println("[DESCARTA] medicao descartada");
435         estado = AGUARDANDO;
436       }
437       break;
438
439     case DRENANDO:
440       if (clique) {
441         estado = MEDINDO;
442       } else if (bufferVazio() || WiFi.status() != WL_CONNECTED || !
              mqttClient.connected()) {
443         estado = AGUARDANDO;
444       } else {
445         Medicao m;
446         // se a leitura falhar, nao trava o sistema
447         if (!nvsLerMedicao(nvsHead, m)) {
448           Serial.println("[ERRO NVS] Leitura falhou! Removendo item
                  corrompido.");
449           nvsHead = (nvsHead + 1) % BUFFER_MAX;
450           nvsCount--;
451           nvsSalvarEstado();
452           estado = AGUARDANDO;
453         }
454         else if (publicarMedicao(m)) {
455           cont_publicadas++;
456           nvsHead = (nvsHead + 1) % BUFFER_MAX;
457           nvsCount--;
458           nvsSalvarEstado();
459           Serial.printf("[PUBLICA] event_id=%u enviado ao broker (restam %d
                  )\n", m.event_id, nvsCount);
460           pulso(LOW, HIGH, LOW, 200);
461         }
462         else {
463           Serial.println("[ERRO MQTT] Falha ao publicar (Timeout/Rede).
                  Tentando novamente...");
464           estado = AGUARDANDO;
465         }
466       }
467       break;
468
469     case ERRO_BUFFER_CHEIO:
470       if (!bufferCheio()) estado = AGUARDANDO;
471       break;
472   }
473
474   atualizarLED();
475   delay(5);
```

```
476 }
```

8 PONTE MQTT–SUPABASE

Subscritor MQTT que recebe os eventos publicados pelo ESP32, valida o payload, converte o valor bruto do conversor analógico-digital para milímetros e persiste o registro no banco de dados.

Listagem 8.1 – Ponte `ponte_mqtt_supabase.py`.

```
"""
ponte_mqtt_supabase.py
Ponte MQTT -> Supabase
TCC IFMG Sabará - Hebert Peluso

Assina os tópicos do HiveMQ, lê o JSON que o ESP32 manda, converte o ADC
pra mm e joga no banco do Supabase.

Deixei a sessão persistente (clean_session=False + client_id fixo + QoS 1)
pra não perder mensagem se a ponte cair: o broker segura a fila e entrega
quando ela volta. Assino maquinas/+/medicao, então pega qualquer máquina.
"""

import paho.mqtt.client as mqtt
from supabase import create_client
import json
import ssl
from datetime import datetime, timezone

# --- Supabase ---
# o ideal é carregar de variável de ambiente, sem deixar a chave no código:
#   export SUPABASE_KEY="sua_service_role_key"
SUPABASE_URL  = "URL"
SUPABASE_KEY  = "URL KEY"    # mesma do arquivo atual

supabase = create_client(SUPABASE_URL, SUPABASE_KEY)
print(f">>> Supabase conectado: {SUPABASE_URL}")

# --- calibração do sensor (ADC -> mm) ---
ADC_MIN = 0
ADC_MAX = 4095
MM_MIN  = 0.0
MM_MAX  = 60.0

def adc_para_mm(valor_adc: int) -> float:
    valor_adc = max(ADC_MIN, min(ADC_MAX, valor_adc))
    return round((valor_adc - ADC_MIN) * (MM_MAX - MM_MIN) / (ADC_MAX -
        ADC_MIN) + MM_MIN, 3)

# --- MQTT (HiveMQ) ---
```

```
40 BROKER = "20c9404954c44cd4b27fdab38f999e70.s1.eu.hivemq.cloud"
41 PORT   = 8883
42 USER   = "ADD_USUARIO_MQTT"
43 PASS   = "ADD_SENHA_MQTT"
44
45 # '+' pega um nível só -> medição de qualquer machine_id
46 TOPIC  = "maquinas/+/medicao"
47
48 # id fixo pro broker saber que é a mesma ponte quando reconecta.
49 # se eu mudar isso, a sessão persistente perde a graça (cria uma nova
       zerada).
50 CLIENT_ID = "ponte_supabase_tcc_hebert"
51
52 # --- callbacks (Paho v2) ---
53 def on_connect(client, userdata, flags, rc, properties=None):
54     if rc == 0:
55         print(">>> Conectado ao HiveMQ!")
56         try:
57             sess_present = flags.session_present
58         except AttributeError:
59             sess_present = bool(flags)
60         if sess_present:
61             print(">>> SESS O RECUPERADA do broker (mensagens pendentes
                  serão entregues)")
62         else:
63             print(">>> Sessão nova (broker não tinha estado armazenado)")
64
65         # precisa ser QoS 1 pro broker segurar as mensagens offline
66         client.subscribe(TOPIC, qos=1)
67         print(f">>> Escutando: {TOPIC} (QoS 1)")
68     else:
69         print(f"!!! Falha na conexão. Código: {rc}")
70
71 def on_disconnect(client, userdata, disconnect_flags, rc, properties=None):
72     if rc != 0:
73         print(f"!!! Desconectado inesperadamente (rc={rc}). Auto-reconnect
              em andamento...")
74     else:
75         print(">>> Desconectado normalmente.")
76
77 def on_message(client, userdata, msg):
78     # lê o JSON
79     try:
80         payload = json.loads(msg.payload.decode())
81     except json.JSONDecodeError:
82         print(f"!!! JSON inválido: {msg.payload!r}")
83         return
```

```
84
85      # confere se os campos que preciso vieram
86      measurement = payload.get("measurement", {})
87      obrig_topo = {"event_id", "machine_id"}
88      if (obrig_topo - payload.keys()) or \
89         "adc_raw_mean" not in measurement or "voltage_mean_mv" not in
               measurement:
90          print(f"!!! Campos obrigatórios faltando. Payload: {payload}")
91          return
92
93      # tira os valores e já converte os tipos
94      try:
95          event_id    = int(payload["event_id"])
96          machine_id  = str(payload["machine_id"])
97          raw         = int(measurement["adc_raw_mean"])
98          tensao_v    = float(measurement["voltage_mean_mv"]) / 1000.0    # mV
               -> V
99          desvio_mv   = float(measurement.get("voltage_stddev_mv", 0.0))
100         uptime_s    = int(payload.get("uptime_ms", 0)) // 1000
101         rssi_dbm    = int(payload.get("metadata", {}).get("rssi_dbm", 0))
102     except (TypeError, ValueError):
103         print(f"!!! Tipos inválidos: {payload}")
104         return
105
106     if not (ADC_MIN <= raw <= ADC_MAX):
107         print(f"!!! raw fora da faixa: {raw}")
108         return
109
110     # ADC -> mm (mantém o sentido das colunas que já existiam)
111     medida_mm = adc_para_mm(raw)
112     agora = datetime.now(timezone.utc).isoformat()
113
114     print(f"[{machine_id}] event={event_id}  raw={raw}  V={tensao_v:.3f}
           =>  {medida_mm:.3f} mm")
115
116     # grava no Supabase. colunas antigas (valor_bruto, tensao, medida_mm,
117     # uptime_s) + as novas (machine_id, event_id, desvio_mv, rssi_dbm).
118     try:
119         supabase.table("leituras").insert({
120             "valor_bruto": raw,
121             "tensao":      round(tensao_v, 4),
122             "medida_mm":   medida_mm,
123             "uptime_s":    uptime_s,
124             "machine_id":  machine_id,
125             "event_id":    event_id,
126             "desvio_mv":   round(desvio_mv, 3),
127             "rssi_dbm":    rssi_dbm,
```

```
128             "timestamp":   agora,
129         }).execute()
130         print(">>> Supabase OK")
131     except Exception as e:
132         print(f"!!! Erro Supabase: {e}")
133
134 # --- cliente MQTT (sessão persistente) ---
135 client = mqtt.Client(
136     callback_api_version=mqtt.CallbackAPIVersion.VERSION2,
137     client_id=CLIENT_ID,
138     clean_session=False,   # pede pro broker guardar a sessão
139 )
140 client.username_pw_set(USER, PASS)
141
142 context = ssl.create_default_context()
143 client.tls_set_context(context)
144
145 client.on_connect    = on_connect
146 client.on_message    = on_message
147 client.on_disconnect = on_disconnect
148
149 # reconecta sozinho: começa em 1s e vai até 60s
150 client.reconnect_delay_set(min_delay=1, max_delay=60)
151
152 # --- loop ---
153 print(f"Iniciando ponte para {BROKER} (client_id={CLIENT_ID}, clean_session
    =False)...")
154 try:
155     client.connect(BROKER, PORT, keepalive=60)
156     client.loop_forever()
157 except KeyboardInterrupt:
158     print("\nEncerrado pelo usuário.")
159     client.disconnect()
160 except Exception as e:
161     print(f"Erro: {e}")
```

9 PAINEL DE CONTROLE ESTATÍSTICO DE PROCESSO

Aplicação web que consome os dados do banco e apresenta os gráficos de controle de Shewhart, o histograma e os índices de capacidade do processo.

Listagem 9.1 – Painel `supabase_dashboard.py`.

```
"""
supabase_dashboard.py
Dashboard de CEP (Controle Estatistico de Processo)
TCC IFMG Sabará - Hebert Peluso

Tem filtro de data + hora, filtro por máquina (machine_id), leitura ampliada
ate 50000 registros e exportacao em CSV.
"""

import streamlit as st
from supabase import create_client
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from scipy.stats import norm
from datetime import datetime, timedelta, time
from zoneinfo import ZoneInfo

TZ = ZoneInfo("America/Sao_Paulo")

# --- config da pagina ---
st.set_page_config(
    page_title="CEP - Paquímetro IoT",

    layout="wide",
)

st.title("Dashboard GERENCIAL CEP - IoT")
st.caption("TCC - Hebert Peluso | IFMG Sabará | Dados via Supabase (PostgreSQL)")

# --- conexao com o Supabase ---
# le a url e a chave dos secrets do Streamlit (nao deixar no codigo!)
try:
    SUPABASE_URL = st.secrets["SUPABASE_URL"]
    SUPABASE_KEY = st.secrets["SUPABASE_KEY"]
except KeyError as e:
    st.error(
        f"    Credencial faltando: {e}\n\n"
        "Configure em **Settings     Secrets** no Streamlit Cloud, ou crie um "
```

```
        "arquivo `.streamlit/secrets.toml` local com:\n\n"
        "```\nSUPABASE_URL = \"https://...supabase.co\"\n"
        "SUPABASE_KEY = \"sb_publishable_...\"\n```"
    )
    st.stop()

@st.cache_resource
def conectar_supabase():
    return create_client(SUPABASE_URL, SUPABASE_KEY)

sb = conectar_supabase()

# --- sidebar: filtros ---
st.sidebar.header("       Configurações")

st.sidebar.subheader("     Filtros de período")

agora = datetime.now(TZ).replace(tzinfo=None)
hoje = agora.date()
ontem = hoje - timedelta(days=1)

# so na primeira execucao, pra nao resetar os filtros toda hora
if "data_ini" not in st.session_state:
    st.session_state.data_ini = ontem
    st.session_state.hora_ini = time(0, 0)
    st.session_state.data_fim = hoje
    st.session_state.hora_fim = time(23, 59)

# botoes de atalho vem antes dos widgets, senao nao consigo mexer no
    session_state
st.sidebar.caption("Atalhos rápidos:")
col_a1, col_a2, col_a3 = st.sidebar.columns(3)

def aplicar_atalho(novo_ini: datetime, novo_fim: datetime):
    st.session_state.data_ini = novo_ini.date()
    st.session_state.hora_ini = novo_ini.time().replace(second=0,
        microsecond=0)
    st.session_state.data_fim = novo_fim.date()
    st.session_state.hora_fim = novo_fim.time().replace(second=0,
        microsecond=0)

if col_a1.button(" ltima  1h", use_container_width=True):
    aplicar_atalho(agora - timedelta(hours=1), agora)
    st.rerun()
if col_a2.button("Hoje", use_container_width=True):
    aplicar_atalho(
        datetime.combine(hoje, time(0, 0)),
```

```
84          datetime.combine(hoje, time(23, 59))
85      )
86      st.rerun()
87  if col_a3.button("7 dias", use_container_width=True):
88      aplicar_atalho(
89          datetime.combine(hoje - timedelta(days=7), time(0, 0)),
90          datetime.combine(hoje, time(23, 59))
91      )
92      st.rerun()
93
94  # widgets de data e hora
95  col_d_ini, col_h_ini = st.sidebar.columns([3, 2])
96  with col_d_ini:
97      data_inicio = st.date_input("Data início", key="data_ini")
98  with col_h_ini:
99      hora_inicio = st.time_input("Hora", key="hora_ini", step=60)
100
101 col_d_fim, col_h_fim = st.sidebar.columns([3, 2])
102 with col_d_fim:
103     data_fim = st.date_input("Data fim", key="data_fim")
104 with col_h_fim:
105     hora_fim = st.time_input("Hora", key="hora_fim", step=60)
106
107 dt_inicio = datetime.combine(data_inicio, hora_inicio)
108 dt_fim    = datetime.combine(data_fim, hora_fim)
109
110 if dt_inicio >= dt_fim:
111     st.sidebar.error("       Data/hora de início deve ser anterior    de
        fim")
112     st.stop()
113
114 st.sidebar.subheader("     Limites de especificação (mm)")
115 LIE = st.sidebar.number_input("LIE (Limite Inferior)", value=28.0, step
    =0.1, format="%.2f")
116 LSE = st.sidebar.number_input("LSE (Limite Superior)", value=32.0, step
    =0.1, format="%.2f")
117
118 if LIE >= LSE:
119     st.sidebar.error("       LIE deve ser menor que LSE")
120     st.stop()
121
122 st.sidebar.subheader("     Atualização")
123 auto_refresh = st.sidebar.checkbox("Auto-refresh (10s)", value=False)
124 if st.sidebar.button("     Atualizar agora", use_container_width=True):
125     st.cache_data.clear()
126     st.rerun()
127
```

```
128 # --- busca os dados ---
129 @st.cache_data(ttl=10)
130 def carregar_dados(dt_ini: datetime, dt_fim: datetime):
131     # pego ate o segundo 59 do ultimo minuto, senao perco medicao do fim da
        janela
132     inicio_iso = dt_ini.replace(tzinfo=TZ).isoformat()
133     fim_iso    = (dt_fim + timedelta(seconds=59)).replace(tzinfo=TZ).
        isoformat()
134
135     # select("*") pra nao quebrar se faltar/sobrar coluna
136     response = (
137         sb.table("leituras")
138         .select("*")
139         .gte("timestamp", inicio_iso)
140         .lte("timestamp", fim_iso)
141         .order("timestamp", desc=False)
142         .limit(50000)   # o PostgREST corta em 1000 por padrao, ai eu subo
            o teto
143         .execute()
144     )
145
146     if not response.data:
147         return pd.DataFrame()
148
149     df = pd.DataFrame(response.data)
150     df["timestamp"] = pd.to_datetime(df["timestamp"], utc=True)
151     df["timestamp"] = df["timestamp"].dt.tz_convert("America/Sao_Paulo")
152
153     # leituras antigas (firmware de teste) nao tinham machine_id
154     if "machine_id" not in df.columns:
155         df["machine_id"] = None
156     df["maquina"] = df["machine_id"].fillna("(sem identificação)")
157     return df
158
159 df = carregar_dados(dt_inicio, dt_fim)
160
161 # mostra o periodo selecionado
162 st.info(
163     f"     Período: **{dt_inicio.strftime('%d/%m/%Y %H:%M')}** até **{
        dt_fim.strftime('%d/%m/%Y %H:%M')}**"
164 )
165
166 if df.empty:
167     st.warning(
168         "Nenhuma medição encontrada no período selecionado. "
169         "Verifique se a ponte e o ESP32 estão rodando, ou amplie a janela
            temporal."
```

```
    )
    st.stop()

# se bateu no teto, provavelmente tem mais dado e precisaria paginar
if len(df) >= 50000:
    st.warning(
        f"       A consulta atingiu o teto de **{len(df)} registros**. "
        "Pode haver mais dados no período. Refine o filtro ou implemente
           paginação."
    )

# filtro por maquina: da pra ver um posto so ou todos juntos
st.sidebar.subheader("     Máquina / posto")
maquinas_disp = sorted(df["maquina"].unique().tolist())
maquinas_sel = st.sidebar.multiselect(
    "Selecione a(s) máquina(s)",
    options=maquinas_disp,
    default=maquinas_disp,
)
df = df[df["maquina"].isin(maquinas_sel)]

if df.empty:
    st.warning("Nenhuma medição para a(s) máquina(s) selecionada(s).")
    st.stop()

# --- classifica OK / rejeitado ---
df["status"] = np.where(
    (df["medida_mm"] >= LIE) & (df["medida_mm"] <= LSE),
    "OK", "REJEITADO"
)

# --- indicadores de CEP (Cp, Cpk, etc) ---
medidas = df["medida_mm"].values
n = len(medidas)
media = medidas.mean()
desvio = medidas.std(ddof=1) if n > 1 else 0.0

if desvio > 0:
    cp  = (LSE - LIE) / (6 * desvio)
    cpu = (LSE - media) / (3 * desvio)
    cpl = (media - LIE) / (3 * desvio)
    cpk = min(cpu, cpl)
else:
    cp = cpk = float("nan")

LSC = media + 3 * desvio
LIC = media - 3 * desvio
```

```
n_rejeitos = (df["status"] == "REJEITADO").sum()
pct_rejeito = 100 * n_rejeitos / n if n > 0 else 0.0

if n < 30:
    st.warning(
        f"       Apenas **{n}** amostra(s) no período.  ndices  Cp/Cpk
           calculados sobre "
        "menos de 30 amostras têm baixa confiabilidade estatística."
    )

# --- KPIs (cartoes do topo) ---
if np.isnan(cpk):
    cpk_delta, cpk_cor = None, "off"
elif cpk >= 1.33:
    cpk_delta, cpk_cor = "Capaz", "normal"
elif cpk >= 1.0:
    cpk_delta, cpk_cor = "Marginal", "inverse"
else:
    cpk_delta, cpk_cor = "Incapaz", "inverse"

col1, col2, col3, col4, col5, col6 = st.columns(6)
col1.metric("Total medições", f"{n}")
col2.metric("Média (mm)", f"{media:.3f}")
col3.metric("Desvio padrão", f"{desvio:.4f}")
col4.metric("Cp",  f"{cp:.2f}"  if not np.isnan(cp)  else "   ")
col5.metric("Cpk", f"{cpk:.2f}" if not np.isnan(cpk) else "   ",
            delta=cpk_delta, delta_color=cpk_cor)
col6.metric("Rejeitos", f"{n_rejeitos} ({pct_rejeito:.1f}%)",
            delta_color="inverse")

st.divider()

# --- grafico de controle (Shewhart) ---
st.subheader("     Gráfico de Controle (Shewhart)")

fig_ctrl = go.Figure()
df_ok  = df[df["status"] == "OK"]
df_rej = df[df["status"] == "REJEITADO"]

# uso Scattergl porque com milhares de pontos o normal trava
fig_ctrl.add_trace(go.Scattergl(
    x=df["timestamp"], y=df["medida_mm"],
    mode="lines", name="Tendência",
    line=dict(color="lightgray", width=1), showlegend=False,
))
fig_ctrl.add_trace(go.Scattergl(
    x=df_ok["timestamp"], y=df_ok["medida_mm"],
```

```
    mode="markers", name="Aprovadas",
    marker=dict(color="green", size=7),
))
fig_ctrl.add_trace(go.Scattergl(
    x=df_rej["timestamp"], y=df_rej["medida_mm"],
    mode="markers", name="Rejeitadas",
    marker=dict(color="red", size=10, symbol="x"),
))

fig_ctrl.add_hline(y=LSE, line=dict(color="blue", dash="dash"),
                   annotation_text=f"LSE = {LSE:.2f}", annotation_position=
                       "top right")
fig_ctrl.add_hline(y=LIE, line=dict(color="blue", dash="dash"),
                   annotation_text=f"LIE = {LIE:.2f}", annotation_position=
                       "bottom right")
fig_ctrl.add_hline(y=media, line=dict(color="black", dash="solid"),
                   annotation_text=f"Média = {media:.3f}",
                       annotation_position="top left")
fig_ctrl.add_hline(y=LSC, line=dict(color="orange", dash="dot"),
                   annotation_text=f"LSC = {LSC:.3f}", annotation_position=
                       "top right")
fig_ctrl.add_hline(y=LIC, line=dict(color="orange", dash="dot"),
                   annotation_text=f"LIC = {LIC:.3f}", annotation_position=
                       "bottom right")

fig_ctrl.update_layout(
    xaxis_title="Tempo", yaxis_title="Medida (mm)", height=450,
    hovermode="closest",
    legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right",
        x=1),
)
st.plotly_chart(fig_ctrl, use_container_width=True)

# --- histograma ---
col_hist, col_info = st.columns([2, 1])

with col_hist:
    st.subheader("     Distribuição das Medições")
    fig_hist = go.Figure()
    fig_hist.add_trace(go.Histogram(
        x=medidas, nbinsx=30, name="Frequência",
        marker_color="steelblue", opacity=0.75, histnorm="probability
            density",
    ))
    if desvio > 0 and n > 1:
        x_curve = np.linspace(medidas.min(), medidas.max(), 200)
        y_curve = norm.pdf(x_curve, loc=media, scale=desvio)
```

```
302         fig_hist.add_trace(go.Scatter(
303             x=x_curve, y=y_curve, mode="lines", name="Normal teórica",
304             line=dict(color="darkred", width=2),
305         ))
306     fig_hist.add_vline(x=LSE, line=dict(color="blue", dash="dash"),
            annotation_text="LSE")
307     fig_hist.add_vline(x=LIE, line=dict(color="blue", dash="dash"),
            annotation_text="LIE")
308     fig_hist.add_vline(x=media, line=dict(color="black"), annotation_text="
            Média")
309     fig_hist.update_layout(
310         xaxis_title="Medida (mm)", yaxis_title="Densidade", height=400,
                bargap=0.05,
311     )
312     st.plotly_chart(fig_hist, use_container_width=True)
313
314 with col_info:
315     st.subheader("       Interpretação")
316     if not np.isnan(cpk):
317         if cpk >= 1.67:
318             st.success(f"**Processo excelente** (Cpk = {cpk:.2f}     1,67)"
                    )
319         elif cpk >= 1.33:
320             st.success(f"**Processo capaz** (Cpk = {cpk:.2f}     1,33)")
321         elif cpk >= 1.0:
322             st.warning(f"**Capacidade marginal** (1,00     Cpk = {cpk:.2f}
                    < 1,33)")
323         else:
324             st.error(f"**Processo incapaz** (Cpk = {cpk:.2f} < 1,00)")
325
326     st.markdown(f"""
327     **Resumo estatístico**
328     - Amostras: **{n}**
329     - Média: **{media:.3f} mm**
330     -    (s): **{desvio:.4f} mm**
331     - LSC (+3  ): **{LSC:.3f} mm**
332     - LIC (  3  ): **{LIC:.3f} mm**
333     - Amplitude: **{medidas.max() - medidas.min():.3f} mm**
334     """)
335
336     # botao pra baixar tudo do periodo em CSV (com a maquina de origem)
337     csv_all = df[["timestamp", "maquina", "medida_mm", "valor_bruto", "
            tensao", "status"]].copy()
338     csv_all["timestamp"] = csv_all["timestamp"].dt.strftime("%Y-%m-%d %H:%M
            :%S")
339     st.download_button(
340         "     Baixar todas as medições (CSV)",
```

```
341         csv_all.to_csv(index=False).encode("utf-8"),
342         f"medicoes_{data_inicio}_{data_fim}.csv",
343         "text/csv",
344         use_container_width=True,
345     )
346
347 st.divider()
348
349 # --- tabela de nao conformidades (com filtro extra) ---
350 st.subheader("       Registro de Não Conformidades")
351
352 rejeitos_df = df[df["status"] == "REJEITADO"].copy()
353
354 if rejeitos_df.empty:
355     st.success("    Nenhuma peça rejeitada no período selecionado.")
356 else:
357     with st.expander("     Filtrar não conformidades (refinamento)",
        expanded=False):
358         st.caption(
359             f"Total de rejeitos no período principal: **{len(rejeitos_df)
                }**. "
360             "Use os filtros abaixo para refinar a lista."
361         )
362
363         rej_min = rejeitos_df["timestamp"].min().to_pydatetime()
364         rej_max = rejeitos_df["timestamp"].max().to_pydatetime()
365
366         col_rf1, col_rf2 = st.columns(2)
367         with col_rf1:
368             data_rej_ini = st.date_input(
369                 "Data início (refinar)",
370                 value=rej_min.date(),
371                 key="rej_data_ini",
372                 min_value=rej_min.date(),
373                 max_value=rej_max.date(),
374             )
375             hora_rej_ini = st.time_input(
376                 "Hora início (refinar)",
377                 value=rej_min.time().replace(second=0, microsecond=0),
378                 key="rej_hora_ini",
379                 step=60,
380             )
381         with col_rf2:
382             data_rej_fim = st.date_input(
383                 "Data fim (refinar)",
384                 value=rej_max.date(),
385                 key="rej_data_fim",
```

```
                min_value=rej_min.date(),
                max_value=rej_max.date(),
            )
            hora_rej_fim = st.time_input(
                "Hora fim (refinar)",
                value=rej_max.time().replace(second=0, microsecond=0),
                key="rej_hora_fim",
                step=60,
            )

        col_rf3, col_rf4 = st.columns(2)
        with col_rf3:
            medida_min_filter = st.number_input(
                "Medida mínima (mm)",
                value=float(rejeitos_df["medida_mm"].min()),
                step=0.1, format="%.3f",
                key="rej_medida_min"
            )
        with col_rf4:
            medida_max_filter = st.number_input(
                "Medida máxima (mm)",
                value=float(rejeitos_df["medida_mm"].max()),
                step=0.1, format="%.3f",
                key="rej_medida_max"
            )

    dt_rej_ini = pd.Timestamp(datetime.combine(data_rej_ini, hora_rej_ini),
        tz="America/Sao_Paulo")
    dt_rej_fim = pd.Timestamp(
        datetime.combine(data_rej_fim, hora_rej_fim) + timedelta(seconds
            =59),
        tz="America/Sao_Paulo"
    )

    rejeitos_filtrados = rejeitos_df[
        (rejeitos_df["timestamp"] >= dt_rej_ini) &
        (rejeitos_df["timestamp"] <= dt_rej_fim) &
        (rejeitos_df["medida_mm"] >= medida_min_filter) &
        (rejeitos_df["medida_mm"] <= medida_max_filter)
    ]

    col_c1, col_c2 = st.columns(2)
    col_c1.metric("Rejeitos exibidos", f"{len(rejeitos_filtrados)} de {len(
        rejeitos_df)}")
    if len(rejeitos_filtrados) > 0:
        col_c2.metric("Medida média dos rejeitos", f"{rejeitos_filtrados['
            medida_mm'].mean():.3f} mm")
```

```
429
430     if rejeitos_filtrados.empty:
431         st.warning("Nenhum rejeito corresponde aos filtros aplicados.")
432     else:
433         tabela = rejeitos_filtrados[
434             ["timestamp", "maquina", "medida_mm", "valor_bruto", "tensao",
                    "status"]
435         ].copy()
436         tabela["timestamp"] = tabela["timestamp"].dt.strftime("%Y-%m-%d %H
                :%M:%S")
437         tabela.columns = ["Data/Hora", "Máquina", "Medida (mm)", "ADC bruto
                ", "Tensão (V)", "Status"]
438         st.dataframe(tabela, use_container_width=True, hide_index=True)
439
440         csv = tabela.to_csv(index=False).encode("utf-8")
441         st.download_button(
442             "     Baixar CSV", csv,
443             f"nao_conformidades_{data_rej_ini}_{data_rej_fim}.csv",
444             "text/csv"
445         )
446
447 # --- auto-refresh ---
448 if auto_refresh:
449     import time as _time
450     _time.sleep(10)
451     st.rerun()
```

10 CÓDIGOS DOS TESTES DE LATÊNCIA NO FIRMWARE

Listagem 10.1 – Código do teste.

```
1
2  #include <WiFi.h>
3  #include <WiFiClientSecure.h>
4  #include <MQTTClient.h>   // arduino-mqtt (256dpi)
5  #include "config.h"
6
7  WiFiClientSecure net;
8  MQTTClient client(512);   // buffer interno em bytes (folgado p/ ~170 B)
9
10
11 char topicoMedicao[64];
12 char topicoAck[64];
13
14
15 uint32_t seq          = 0;      // n   de sequência do próximo envio
16 uint32_t seqEmVoo     = 0;      // seq aguardando ACK
17 uint32_t t0           = 0;      // millis() no instante do publish
18 bool     aguardandoAck = false;
19 uint32_t ultimoEnvio  = 0;
20 uint32_t coletadas    = 0;      // amostras com RTT válido já registradas
21 bool     testeConcluido = false;
22
23
24 char   payload[256];
25 size_t payloadLen = 0;
26
27
28 size_t tamanhoPacotePublish(const char* topico, size_t lenPayload, int qos)
       {
29   size_t remaining = 2 + strlen(topico) + (qos > 0 ? 2 : 0) + lenPayload;
30   size_t bytesRL = 1;
31   size_t r = remaining;
32   while (r >= 128) { r /= 128; bytesRL++; }   // codificação Variable Byte
       Integer
33   return 1 + bytesRL + remaining;
34 }
35
36
37 void montarPayload() {
38   const int N = 50;
39   long soma = 0;
40   int amostras[N];
41   for (int i = 0; i < N; i++) {
42     amostras[i] = analogRead(PIN_POT);
```

```
43      soma += amostras[i];
44      delayMicroseconds(200);
45    }
46    float media = (float)soma / N;
47    float somaQuad = 0;
48    for (int i = 0; i < N; i++) {
49      float d = amostras[i] - media;
50      somaQuad += d * d;
51    }
52    float desvio = sqrt(somaQuad / N);
53    long rssi = (long)WiFi.RSSI();
54
55    payloadLen = snprintf(payload, sizeof(payload),
56      "{\"machine_id\":\"%s\",\"seq\":%u,\"medicao\":{\"media\":%.2f,"
57      "\"desvio\":%.2f,\"amostras\":%d},\"rssi\":%ld}",
58      MACHINE_ID, seq, media, desvio, N, rssi);
59  }
60
61
62  // Callback de mensagens (o ACK do bridge chega aqui). Payload esperado: {"
        seq":N}
63
64  void onMessage(String &topic, String &mensagem) {
65    if (!aguardandoAck) return;
66
67    int p = mensagem.indexOf("\"seq\"");
68    if (p < 0) return;
69    int c = mensagem.indexOf(':', p);
70    if (c < 0) return;
71    uint32_t seqAck = (uint32_t) mensagem.substring(c + 1).toInt();
72
73    if (seqAck == seqEmVoo) {
74      uint32_t rtt = millis() - t0;
75      float umaVia = rtt / 2.0f;
76      size_t pacote = tamanhoPacotePublish(topicoMedicao, payloadLen, 1);
77
78      // CSV: seq,rtt_ms,lat_uma_via_ms,payload_bytes,pacote_mqtt_bytes,
            rssi_dbm,status
79      Serial.printf("%u,%u,%.1f,%u,%u,%ld,OK\n",
80                    seqEmVoo, rtt, umaVia,
81                    (unsigned)payloadLen, (unsigned)pacote, (long)WiFi.RSSI()
                          );
82
83      aguardandoAck = false;
84      coletadas++;
85    }
86  }
```

```
87
88  void conectarWiFi() {
89    Serial.printf("# Conectando ao Wi-Fi: %s\n", WIFI_SSID);
90    WiFi.mode(WIFI_STA);
91    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
92    while (WiFi.status() != WL_CONNECTED) {
93      delay(300);
94      Serial.print("# .");
95    }
96    Serial.printf("\n# Wi-Fi OK. IP: %s | RSSI: %ld dBm\n",
97                  WiFi.localIP().toString().c_str(), (long)WiFi.RSSI());
98  }
99
100 void conectarMQTT() {
101   Serial.print("# Conectando ao broker MQTT");
102   // clientID único p/ não colidir com o firmware definitivo no broker
103   String clientId = "esp32-teste-" + String((uint32_t)ESP.getEfuseMac(),
        HEX);
104   while (!client.connect(clientId.c_str(), MQTT_USER, MQTT_PASS)) {
105     Serial.print(".");
106     delay(500);
107   }
108   client.subscribe(topicoAck, 1);
109   Serial.printf("\n# MQTT OK. Inscrito em: %s\n", topicoAck);
110 }
111
112 void setup() {
113   Serial.begin(115200);
114   delay(500);
115   analogReadResolution(12);
116
117   snprintf(topicoMedicao, sizeof(topicoMedicao), "maquinas/%s/medicao",
        MACHINE_ID);
118   snprintf(topicoAck,     sizeof(topicoAck),     "maquinas/%s/ack",
        MACHINE_ID);
119
120   Serial.println("# ====== TESTE DE LATENCIA/PAYLOAD MQTT (Metodo A) ======
        ");
121   Serial.printf("# broker=%s:%d  machine=%s\n", MQTT_BROKER, MQTT_PORT,
        MACHINE_ID);
122   Serial.printf("# intervalo=%d ms  n_amostras=%d  timeout_ack=%d ms\n",
123                 INTERVALO_MS, N_AMOSTRAS, TIMEOUT_ACK_MS);
124
125   conectarWiFi();
126
127   net.setInsecure();   // TLS sem validar o cert do servidor: OK p/ o teste
        .
```

```
128                         // O custo do TLS é idêntico; a validação ocorre só
                               no
129                         // handshake (1x), não por mensagem -> não afeta lat
                               ência
130                         // nem o tamanho dos níveis 1/2.
131   client.begin(MQTT_BROKER, MQTT_PORT, net);
132   client.setKeepAlive(30);
133   client.onMessage(onMessage);
134   conectarMQTT();
135
136   // Cabeçalho CSV (uma única vez)
137   Serial.println("seq,rtt_ms,lat_uma_via_ms,payload_bytes,pacote_mqtt_bytes
          ,rssi_dbm,status");
138
139   ultimoEnvio = millis();   // 1   envio após um INTERVALO_MS
140 }
141
142 void loop() {
143   client.loop();
144
145   if (!client.connected()) {
146     Serial.println("# MQTT caiu, reconectando...");
147     conectarMQTT();
148   }
149
150   uint32_t agora = millis();
151
152   // Timeout de ACK -> registra como perda (TIMEOUT) e libera o próximo
          envio
153   if (aguardandoAck && (agora - t0 > (uint32_t)TIMEOUT_ACK_MS)) {
154     size_t pacote = tamanhoPacotePublish(topicoMedicao, payloadLen, 1);
155     // colunas rtt_ms e lat_uma_via_ms ficam vazias
156     Serial.printf("%u,,,%u,%u,%ld,TIMEOUT\n",
157                   seqEmVoo, (unsigned)payloadLen, (unsigned)pacote, (long)
                         WiFi.RSSI());
158     aguardandoAck = false;
159   }
160
161   // Fim do teste
162   if (!testeConcluido && coletadas >= (uint32_t)N_AMOSTRAS) {
163     testeConcluido = true;
164     Serial.printf("# ====== FIM: %u amostras validas coletadas ======\n",
            coletadas);
165     Serial.println("# Copie do cabecalho CSV ate aqui para um arquivo .csv"
            );
166     return;
167   }
```

```
168   if (testeConcluido) return;
169
170   // Próximo envio no tique do intervalo (só se não há ACK pendente)
171   if (!aguardandoAck && (agora - ultimoEnvio >= (uint32_t)INTERVALO_MS)) {
172     seq++;
173     montarPayload();
174     t0 = millis();                                    // marca ANTES do
          publish
175     bool ok = client.publish(topicoMedicao, payload, false, 1);  //
          retained=0, QoS=1
176     if (ok) {
177       seqEmVoo = seq;
178       aguardandoAck = true;
179       ultimoEnvio = agora;
180     } else {
181       Serial.printf("# Falha ao publicar seq=%u\n", seq);
182     }
183   }
184 }
```

11 CÓDIGOS DOS TESTES DE LATÊNCIA NO BRIDGE (SUBSCRIBER)

Listagem 11.1 – RTT.py

```
1
2  #include <WiFi.h>
3  #include <WiFiClientSecure.h>
4  #include <MQTTClient.h>   // arduino-mqtt (256dpi)
5  #include "config.h"
6
7  WiFiClientSecure net;
8  MQTTClient client(512);   // buffer interno em bytes (folgado p/ ~170 B)
9
10
11 char topicoMedicao[64];
12 char topicoAck[64];
13
14
15 uint32_t seq          = 0;      // n   de sequência do próximo envio
16 uint32_t seqEmVoo     = 0;      // seq aguardando ACK
17 uint32_t t0           = 0;      // millis() no instante do publish
18 bool     aguardandoAck = false;
19 uint32_t ultimoEnvio  = 0;
20 uint32_t coletadas    = 0;      // amostras com RTT válido já registradas
21 bool     testeConcluido = false;
22
23
24 char   payload[256];
25 size_t payloadLen = 0;
26
27
28 size_t tamanhoPacotePublish(const char* topico, size_t lenPayload, int qos)
       {
29   size_t remaining = 2 + strlen(topico) + (qos > 0 ? 2 : 0) + lenPayload;
30   size_t bytesRL = 1;
31   size_t r = remaining;
32   while (r >= 128) { r /= 128; bytesRL++; }   // codificação Variable Byte
         Integer
33   return 1 + bytesRL + remaining;
34 }
35
36
37 void montarPayload() {
38   const int N = 50;
39   long soma = 0;
40   int amostras[N];
41   for (int i = 0; i < N; i++) {
```

```
42      amostras[i] = analogRead(PIN_POT);
43      soma += amostras[i];
44      delayMicroseconds(200);
45    }
46    float media = (float)soma / N;
47    float somaQuad = 0;
48    for (int i = 0; i < N; i++) {
49      float d = amostras[i] - media;
50      somaQuad += d * d;
51    }
52    float desvio = sqrt(somaQuad / N);
53    long rssi = (long)WiFi.RSSI();
54
55    payloadLen = snprintf(payload, sizeof(payload),
56      "{\"machine_id\":\"%s\",\"seq\":%u,\"medicao\":{\"media\":%.2f,"
57      "\"desvio\":%.2f,\"amostras\":%d},\"rssi\":%ld}",
58      MACHINE_ID, seq, media, desvio, N, rssi);
59  }
60
61
62  // Callback de mensagens (o ACK do bridge chega aqui). Payload esperado: {"
        seq":N}
63
64  void onMessage(String &topic, String &mensagem) {
65    if (!aguardandoAck) return;
66
67    int p = mensagem.indexOf("\"seq\"");
68    if (p < 0) return;
69    int c = mensagem.indexOf(':', p);
70    if (c < 0) return;
71    uint32_t seqAck = (uint32_t) mensagem.substring(c + 1).toInt();
72
73    if (seqAck == seqEmVoo) {
74      uint32_t rtt = millis() - t0;
75      float umaVia = rtt / 2.0f;
76      size_t pacote = tamanhoPacotePublish(topicoMedicao, payloadLen, 1);
77
78      // CSV: seq,rtt_ms,lat_uma_via_ms,payload_bytes,pacote_mqtt_bytes,
            rssi_dbm,status
79      Serial.printf("%u,%u,%.1f,%u,%u,%ld,OK\n",
80                    seqEmVoo, rtt, umaVia,
81                    (unsigned)payloadLen, (unsigned)pacote, (long)WiFi.RSSI()
                          );
82
83      aguardandoAck = false;
84      coletadas++;
85    }
```

```
}

void conectarWiFi() {
  Serial.printf("# Conectando ao Wi-Fi: %s\n", WIFI_SSID);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print("# .");
  }
  Serial.printf("\n# Wi-Fi OK. IP: %s | RSSI: %ld dBm\n",
                WiFi.localIP().toString().c_str(), (long)WiFi.RSSI());
}

void conectarMQTT() {
  Serial.print("# Conectando ao broker MQTT");
  // clientID único p/ não colidir com o firmware definitivo no broker
  String clientId = "esp32-teste-" + String((uint32_t)ESP.getEfuseMac(),
      HEX);
  while (!client.connect(clientId.c_str(), MQTT_USER, MQTT_PASS)) {
    Serial.print(".");
    delay(500);
  }
  client.subscribe(topicoAck, 1);
  Serial.printf("\n# MQTT OK. Inscrito em: %s\n", topicoAck);
}

void setup() {
  Serial.begin(115200);
  delay(500);
  analogReadResolution(12);

  snprintf(topicoMedicao, sizeof(topicoMedicao), "maquinas/%s/medicao",
      MACHINE_ID);
  snprintf(topicoAck,     sizeof(topicoAck),     "maquinas/%s/ack",
      MACHINE_ID);

  Serial.println("# ====== TESTE DE LATENCIA/PAYLOAD MQTT (Metodo A) ======
      ");
  Serial.printf("# broker=%s:%d  machine=%s\n", MQTT_BROKER, MQTT_PORT,
      MACHINE_ID);
  Serial.printf("# intervalo=%d ms  n_amostras=%d  timeout_ack=%d ms\n",
                INTERVALO_MS, N_AMOSTRAS, TIMEOUT_ACK_MS);

  conectarWiFi();

```

```
127   net.setInsecure();    // TLS sem validar o cert do servidor: OK p/ o teste
          .
128                         // O custo do TLS é idêntico; a validação ocorre só
                               no
129                         // handshake (1x), não por mensagem -> não afeta lat
                               ência
130                         // nem o tamanho dos níveis 1/2.
131   client.begin(MQTT_BROKER, MQTT_PORT, net);
132   client.setKeepAlive(30);
133   client.onMessage(onMessage);
134   conectarMQTT();
135
136   // Cabeçalho CSV (uma única vez)
137   Serial.println("seq,rtt_ms,lat_uma_via_ms,payload_bytes,pacote_mqtt_bytes
          ,rssi_dbm,status");
138
139   ultimoEnvio = millis();    // 1   envio após um INTERVALO_MS
140 }
141
142 void loop() {
143   client.loop();
144
145   if (!client.connected()) {
146     Serial.println("# MQTT caiu, reconectando...");
147     conectarMQTT();
148   }
149
150   uint32_t agora = millis();
151
152   // Timeout de ACK -> registra como perda (TIMEOUT) e libera o próximo
          envio
153   if (aguardandoAck && (agora - t0 > (uint32_t)TIMEOUT_ACK_MS)) {
154     size_t pacote = tamanhoPacotePublish(topicoMedicao, payloadLen, 1);
155     // colunas rtt_ms e lat_uma_via_ms ficam vazias
156     Serial.printf("%u,,,%u,%u,%ld,TIMEOUT\n",
157                   seqEmVoo, (unsigned)payloadLen, (unsigned)pacote, (long)
                         WiFi.RSSI());
158     aguardandoAck = false;
159   }
160
161   // Fim do teste
162   if (!testeConcluido && coletadas >= (uint32_t)N_AMOSTRAS) {
163     testeConcluido = true;
164     Serial.printf("# ====== FIM: %u amostras validas coletadas ======\n",
            coletadas);
165     Serial.println("# Copie do cabecalho CSV ate aqui para um arquivo .csv"
            );
```

```
166       return;
167     }
168     if (testeConcluido) return;
169
170     // Próximo envio no tique do intervalo (só se não há ACK pendente)
171     if (!aguardandoAck && (agora - ultimoEnvio >= (uint32_t)INTERVALO_MS)) {
172       seq++;
173       montarPayload();
174       t0 = millis();                                    // marca ANTES do
            publish
175       bool ok = client.publish(topicoMedicao, payload, false, 1);  //
            retained=0, QoS=1
176       if (ok) {
177         seqEmVoo = seq;
178         aguardandoAck = true;
179         ultimoEnvio = agora;
180       } else {
181         Serial.printf("# Falha ao publicar seq=%u\n", seq);
182       }
183     }
184 }
```