Fundação Universidade Federal do ABC
Pró reitoria de pesquisa
Av. dos Estados, 5001, Santa Terezinha, Santo André/SP, CEP 09210-580
Bloco L, 3ºAndar, Fone (11) 3356-7617
iniciacao@ufabc.edu.br

Relatório Parcial de Iniciação Científica
referente ao Edital: 06/2025

Nome do aluno: Nicolas Benitez Lopes

Assinatura do aluno:

Nome do orientador: Hugo Puertas de Araujo

Assinatura do orientador:

Título do projeto: Sistema de Telemetria para Drones

Palavras-chave do projeto: Telemetria, Drone - Quadricóptero, Internet das Coisas, ESP32, Linguagem C/C++, Banco de Dados, MQTT, Node-RED

Área do conhecimento do projeto: Ciência da Computação; Sistemas Embarcados; Banco de Dados

Bolsista: Sim

Santo André

2026

Sumário

1 Resumo

O seguinte trabalho tem como objetivo desenvolver uma plataforma de telemetria para drones quadricópteros, destinada à coleta de dados em tempo real durante as operações do veículo e ao desenvolvimento de um banco de dados utilizável em análises e estudos das operações. As motivações para esse projeto incluem a relevância e importância para a indústria tecnológica atual das tecnologias a serem empregadas nele, como telemetria, Internet das Coisas (IoT) e bancos de dados. Além disso, a proposta terá a possibilidade de ser integrada a outros projetos que abrangem essa área, podendo resultar em uma realização maior. Os métodos empregados durante o processo incluirão conhecimentos da área da eletrônica e da computação, tendo em vista que será necessário: a criação e teste de um circuito impresso que integre os sensores que medirão as grandezas a um microcontrolador, e que seja compatível com a estrutura do drone modelo; a implementação de um sistema de comunicação que viabilize a troca de dados em tempo real; o desenvolvimento de um servidor remoto que armazenará os dados; bem como a aplicação de uma interface gráfica que permita que os dados sejam interpretados mais facilmente.

2 Introdução

As áreas do conhecimento abordadas nesta proposta são de grande relevância no atual cenário de inovação e tecnologia. A Internet das Coisas (IoT), por exemplo, destaca-se como uma das principais linhas de pesquisa envolvidas no projeto. Segundo o artigo *IoT: Communication Protocols and Security Threats:*

> …a previsão tecnológica para a Internet das Coisas indica um aumento de aproximadamente 300% no número de dispositivos conectados, passando de 8,7 bilhões em 2020 para mais de 25 bilhões em 2030

O que evidencia o seu crescimento exponencial e impacto global.

Além disso, os bancos de dados também desempenham papel central na indústria contemporânea. De acordo com Sistemas de Banco de Dados, de Ramez Elmasri e Shamkant B. Navathe:

> …os bancos de dados e os sistemas de bancos de dados se tornaram componentes essenciais no cotidiano da sociedade moderna. No decorrer do dia, a maioria de nós se depara com atividades que envolvem alguma interação com os bancos de dados

Destacando a sua importância para o funcionamento de sistemas modernos e o avanço tecnológico.

No contexto da Indústria 4.0, a telemetria -- foco deste projeto -- emerge como um componente essencial, justamente por integrar tecnologias como IoT e sistemas de banco de dados. Conforme destaca Sandvik (2020), a chamada “Quarta Revolução Industrial” tem transformado profundamente os processos produtivos, ao conectar o mundo físico ao digital por meio de sensores inteligentes, geração e análise de grandes volumes de dados, sistemas autônomos e inteligência artificial. Nesse cenário, a telemetria vem ganhando destaque por possibilitar o monitoramento remoto, em tempo real, de equipamentos e operações, sendo uma das áreas que mais crescem dentro dessa transformação digital.

Dessa forma, este projeto se insere diretamente nas demandas contemporâneas por digitalização e automação industrial, uma vez que as habilidades adquiridas pelo aluno em um dado contexto (monitoramento de um drone) podem ser prontamente extrapoladas para qualquer outro cenário, como monitoramento de equipamento industrial, agricultura inteligente, etc.

Além disso, o desenvolvimento deste trabalho é parte de uma iniciativa mais ampla, vinculada à criação de uma plataforma educacional baseada em um quadricóptero. Essa plataforma integrará controle remoto, firmware de controle de voo e diversos softwares de apoio — incluindo o sistema de telemetria aqui proposto, que oferecerá suporte completo à estrutura. Assim, a solução desenvolvida poderá servir como base para pesquisas futuras na UFABC, um exemplo disso, é o uso dos dados produzidos para a implementação de controladores convencionais e também os baseados em redes neurais, evidenciando o potencial de uso em contextos didáticos, experimentais ou de aprimoramento tecnológico.

2.1 Objetivos

Este projeto não visa propor novos protocolos de comunicação ou circuitos inéditos – adotaremos tecnologias consolidadas como MQTT, Wi-Fi e microcontroladores adequados, garantindo robustez e confiabilidade. O objetivo central é desenvolver uma **plataforma de telemetria como tecnologia habilitadora de alto impacto**, cujo valor será mensurado por sua capacidade de: (i) capturar, armazenar e disponibilizar dados de forma confiável e eficiente; (ii) integrar e dar suporte a múltiplos projetos de pesquisa e desenvolvimento; e (iii) servir como ferramenta didática para a formação prática de discentes.

A métrica primordial de sucesso será a efetividade na geração e no aproveitamento dos dados por esses projetos. Adicionalmente, o desempenho do sistema será quantificado por meio de métricas técnicas específicas, a serem validadas em testes de bancada e em voo:

- **Usabilidade do Sistema:** Será avaliada pela eficiência do fluxo de trabalho para um usuário final (pesquisador ou discente), medindo-se o tempo necessário desde a conexão do drone até a visualização dos primeiros dados no painel. O objetivo é que este tempo seja inferior a 5 minutos, com uma interface que não exija consulta constante a manuais.
- **Taxa de Amostragem:** Será medida empiricamente gravando-se a sequência de *timestamps* dos pacotes de dados recebidos. A meta é atingir uma taxa estável de no mínimo 100 Hz para dados inerciais e 10 Hz para dados de telemetria geral (tensão, corrente, posição).
- **Latência na Transmissão ("atraso"):** Será calculada como a diferença de tempo ($T_{recebimento} - T_{envio}$) medindo-se o *round-trip time* (RTT) de um pacote de eco. A latência fim-a-fim deve ser inferior a 50 ms para permitir o monitoramento em tempo real.

O sistema se insere como infraestrutura comum a um ecossistema de projetos aprovados e em andamento. Destacamos abaixo seu papel catalisador em três projetos centrais:

- **Projeto de um drone quadricóptero.** Plataforma central. O sistema de telemetria deve ser desenvolvido concomitantemente com um projeto de quadricóptero para uma integração mais efetiva entre ambos, como a redução de circuitos redundantes e favorecendo um projeto mais simples da placa de circuito impresso. Além disso, as etapas de caracterização e testes de bancada de subsistemas, como o acionamento dos motores por PWM, a caracterização de altitude por meio do barômetro e a captura de dados inerciais, contribuirão diretamente para o projeto do drone em si, uma vez que tais dados são necessários para a seleção de componentes, projeto de circuitaria e de firmware de controle. Uma vez finalizados, o sistema de telemetria permitirá o acompanhamento do voo, coletando dados elétricos (tensão e corrente) dos motores e como isso afeta a dinâmica do voo (dados inerciais como acelerações e velocidades angulares nos 3 eixos e variação da altitude).
- **Sistema de navegação indoor.** Deve funcionar como uma espécie de GPS para ambientes internos. Baseia-se na identificação de assinaturas de Wi-Fi para estimar a posição em um ambiente previamente mapeado. O sistema de telemetria recolherá essa assinatura de Wi-Fi ajudando na etapa de mapeamento e também fazendo uso do sistema em experiências de navegação indoor automática. A plataforma também permitiria pesquisas com enxames de drones coordenados.
- **Projeto de controle baseado em dados.** Atualmente, o desenvolvimento de pilotos automáticos para drones por meio de inteligência artificial é realizado em ambientes de simulação, com posterior transferência do controlador para as aeronaves físicas. O procedimento inclui a coleta de dados de voo reais, que são utilizados para treinar o modelo no ambiente simulado. O sistema de telemetria é fundamental, tanto para a aquisição inicial desses dados quanto para verificar a eficácia do controlador em operação.

Em suma, o sistema de telemetria proposto se consolida como a espinha dorsal tecnológica que integra e viabiliza estas frentes de pesquisa. Sua contribuição transcende o desenvolvimento em si, tornando-se um facilitador essencial para a geração de conhecimento, o suporte à docência em aulas práticas e o avanço de projetos inovadores, os quais seriam inviáveis ou consideravelmente mais limitados sem esta ferramenta fundamental.

3 Fundamentação teórica

A telemetria, consiste no processo pelo qual dados são coletados por instrumentos, convertidos em sinais de rádio e transmitidos a uma estação receptora, onde esses sinais são decodificados e registrados *Nasa Medical Telemetry 1978*, ainda o artigo *Telemetry: Connectivity and Productivity in Real Time – Project Implementation Guide* aponta a telemetria como

> …uma das aplicações que mais avançam em análise. [...] Qualquer sistema de telemetria tem pelo menos três partes: a entrada física para medição, um canal de comunicação e uma unidade de controle. Conforme definido pelo GMG, a telemetria é um processo tecnológico automatizado para comunicação de medições e outros dados entre locais remotos e equipamentos receptores

A telemetria se mostra presente em todos os setores da economia, tendo aplicações que abrangem: a agricultura de precisão, em que sensores em tratores e estações meteorológicas enviam dados via telemetria para otimizar irrigação, fertilização e colheita (Agrotelemetria); o setor automobilístico, com análises em tempo real de dados de desempenho, aperfeiçoando a performance dos veículos, (*Televeicle*); e até mesmo na prestação de serviços como a saúde, sendo a telemetria médica usada em UTI (Unidade de Terapia Intensiva) para monitorar sinais vitais à distância, permitindo alertas automáticos (*Medicaltele*).

Essa tecnologia será aplicada, como um caso de uso, em uma plataforma destinada ao monitoramento de um VANT (Veículo Aéreo Não Tripulado) -- normalmente denominado de *drone* -- que nada mais é do que uma aeronave que voa sem um piloto nos controles, mas sim com a assistência de um operador em solo, ou por meio de voo automatizado sem intervenção humana *Drone Disaster 2021*.

O modelo de veículo quadricóptero é um modelo com ampla gama de aplicações em diversas áreas, como: uso em monitoramento ambiental, resgate, inspeção de infraestruturas, vigilância e logística e, além disso, tem simplicidade de controle em comparação com aeronaves de asa fixa. Isso ocorre porque o quadricóptero pode ser direcionado apenas pela variação das velocidades de seus quatro motores, o que permite controlar altitude, direção e estabilidade, sem a necessidade de superfícies aerodinâmicas móveis. Em contraste, drones de asa fixa exigem a manipulação de ângulos de superfícies de controle, bem como maior conhecimento de aerodinâmica, resultando em sistemas de controle mais complexos.

Dados de telemetria são transmitidos em tempo real (ou ao menos armazenados localmente e depois transmitidos com a devida marcação temporal -- *timestamp* -- conforme as condições de comunicação vigentes no momento) por meio de tecnologia de comunicação sem-fio baseada em Wi-Fi, típica de aplicações de Internet das Coisas (IoT). Segundo o artigo *IoT: Communication Protocols and Security Threats*

> …pelo termo Internet das Coisas, abreviado para IoT, nos referimos aos inúmeros dispositivos tangíveis ao redor do mundo que podem ser conectados à internet. Todos esses dispositivos coletam e compartilham dados entre si ao mesmo tempo em que eliminam a necessidade de comunicação entre humanos ou mesmo entre humanos e computadores

Uma dos processos mais eficientes para o compartilhamento de dados via internet, como entre a plataforma embarcada no drone e um servidor remoto, é feito utilizando o protocolo MQTT — amplamente adotado em aplicações de IoT.

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de comunicação projetado para cenários do tipo *Machine-to-Machine* (M2M) e Internet das Coisas (IoT), caracterizando-se por sua leveza, simplicidade e eficiência no uso da largura de banda. Conforme apontado por Mishra e Kertész (2020), sua arquitetura baseia-se no modelo *publish/subscribe*, no qual os clientes (dispositivos que geram e/ou transmitem a informação) não se comunicam diretamente entre si, mas por meio de um *broker* (servidor responsável pelo gerenciamento das mensagens). Essa abordagem promove baixo acoplamento entre os dispositivos e garante escalabilidade em aplicações distribuídas. Além disso, o MQTT suporta três níveis de Qualidade de Serviço (*Quality of Service* — QoS): entrega no máximo uma vez (*at most once*), pelo menos uma vez (*at least once*) e exatamente uma vez (*exactly once*), que permitem ajustar a confiabilidade da transmissão de acordo com as restrições da aplicação. Devido a essas características, o protocolo tornou-se amplamente adotado em soluções que envolvem sensores remotos, telemetria e sistemas embarcados de baixo consumo energético, como a aplicação proposta.

Um servidor será responsável por receber as informações do VANT, em tempo real via protocolo MQTT, e armazenar os dados em um banco de dados. De forma simplificada, um banco de dados é uma coleção organizada de dados relacionados, com significado inerente e estruturada para representar aspectos do mundo real. Ele é projetado para atender a propósitos específicos, possibilitando o armazenamento lógico e coerente de informações com fontes definidas e aplicabilidades práticas (ELMASRI, 2005). No contexto deste projeto, o banco de dados terá como função central registrar e organizar os dados recebidos do quadricóptero, refletindo suas operações e permitindo análises precisas, inclusive com geração de modelos de observabilidade de estados e modelos de controle baseados em dados.

4 Metodologia

4.1 Materiais e Métodos

Para o desenvolvimento deste projeto, a pesquisa bibliográfica terá papel fundamental, contribuindo para a fundamentação teórica ao longo de todo o processo. Ela proporcionará ao aluno familiarização com conceitos, ferramentas e plataformas digitais utilizadas, além de desenvolver o embasamento necessário para a implementação prática do sistema.

O sistema eletrônico da plataforma será baseado no microcontrolador ESP32, um microcontrolador que combina Wi-Fi e Bluetooth de 2,4 GHz, projetado com a tecnologia de 40 nm de baixo consumo da TSMC para atingir o melhor desempenho de potência em RF, demonstrando robustez, versatilidade e confiabilidade em uma ampla variedade de aplicações e cenários de consumo de energia *ESP32datasheet*. O dispositivo será programado em linguagem C/C++, e será responsável pela aquisição de dados e comunicação entre drone e servidor.

Os circuitos eletrônicos usados na coleta de dados serão desenvolvidos por meio do software EasyEDA, responsável pelo projeto esquemático (desenho e simulação com SPICE) e de layout da PCB *Printed Circuit Board* -- Placa de Circuito Impresso). A placa projetada deverá ser integrada à estrutura do quadricóptero (aproximadamente 10 cm x 10 cm), que será construída em outro trabalho, realizado concomitantemente ao projeto aqui proposto.

Como forma de otimização e economia de recursos, serão utilizados os próprios conversores analógicos internos do ESP32 para medir a tensão nos motores, reduzindo a necessidade de sensores externos, tendo em vista que uma de das características principais demandadas do sistema é o baixo peso, a fim de minimizar o consumo energético durante o voo. No entanto, o sistema também contará com sensores essenciais como giroscópio e sensor barométrico de pressão.

Será desenvolvido um firmware dedicado para o ESP32, responsável pela aquisição e envio de dados dos sensores para o servidor, utilizando o protocolo MQTT.

A comunicação entre o sistema embarcado e o servidor será gerenciada pela plataforma Eclipse Mosquitto, um broker MQTT open-source amplamente utilizado. O gerenciamento lógico da rede, incluindo definição de tópicos, fluxos de entrada e saída de dados, será realizado com a plataforma Node-RED, que permitirá o controle dos processos. Essa é uma ferramenta de programação visual, que utiliza uma interface baseada em fluxos de nós interconectados, onde cada nó representa uma função ou operação (entrada, processamento ou saída de dados), reduzindo a necessidade de programação específica extensa *Nodered*.

Com a estrutura de comunicação estabelecida, será criado um banco de dados para armazenamento das informações coletadas, utilizando a ferramenta gratuita InfluxDB, projetada especificamente para armazenar e consultar grandes volumes de dados que variam ao longo do tempo(série temporal), ideal para a aplicação de telemetria proposta. Os dados armazenados serão analisados e visualizados por meio da plataforma Grafana, permitindo a construção de painéis dinâmicos, que facilitam a interpretação dos dados e a compreensão do comportamento do drone em diferentes condições.

Os aplicativos citados serão utilizados por meio da plataforma Docker. Este consiste em um sistema de virtualização leve baseado em contêineres, que permite empacotar aplicações juntamente com todas as suas dependências (bibliotecas, configurações e ambientes de execução) em unidades isoladas e portáveis. Dessa forma, cada serviço — como o broker MQTT, o Node-RED, o banco de dados e a ferramenta de visualização — pode ser executado de maneira independente, garantindo maior facilidade na configuração, reprodutibilidade do ambiente e escalabilidade do sistema. Além disso, o uso de contêineres simplifica a implantação em diferentes máquinas, reduz conflitos entre dependências e facilita a manutenção e atualização dos serviços ao longo do desenvolvimento do projeto

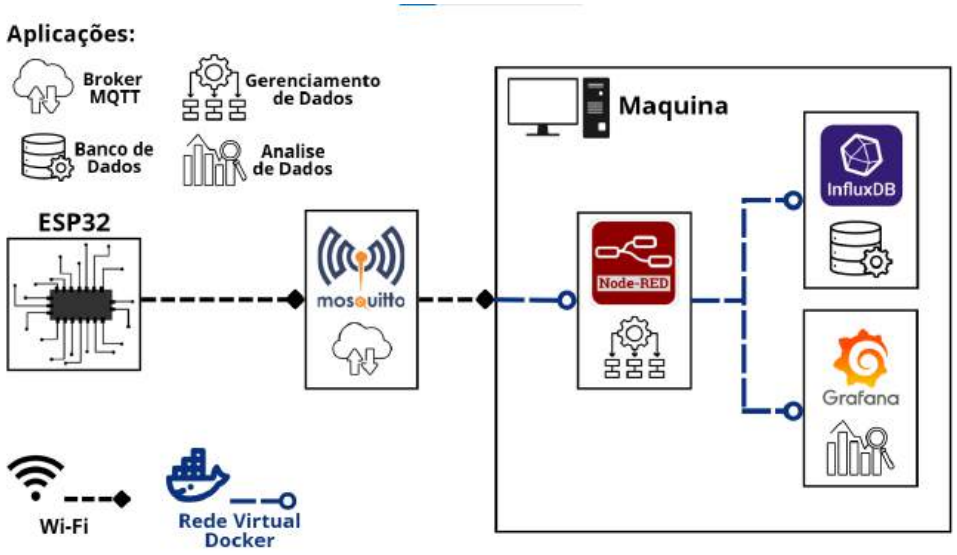

Imagem 1 - Diagrama Geral Sistema de Telemetria

Além disso, os dados mais relevantes também serão exportados para planilhas, permitindo análises estatísticas complementares e organização de informações para relatórios.

Ao longo do projeto, enquanto o drone está sendo desenvolvido, serão realizados testes de bancada envolvendo firmware, os circuitos de aquisição de dados e os respectivos sensores, visando-se a geração de informações de suporte ao próprio projeto do drone. Assim, serão realizados ensaios nos motores para a determinação de consumo máximo de corrente e medição do empuxo gerado; controle de velocidade dos motores por sinais PWM; calibração de altitude através do barômetro; e testes de estabilização da inclinação do drone por meio de sensores inerciais (acelerômetros e giroscópios).

Por fim, se possível, será realizada uma etapa de validação empírica, por meio de testes de campo no drone (então já construído), com o objetivo de avaliar o desempenho da plataforma de telemetria nas operações reais. Os resultados servirão de base para eventuais aprimoramentos do sistema, bem como para viabilizar a sua aplicabilidade em contextos educacionais e de pesquisa.

4.2 Etapas da pesquisa

A primeira etapa do desenvolvimento do projeto consistiu na criação do ambiente digital destinado à implementação do sistema de telemetria. As ferramentas selecionadas incluíram o broker Mosquitto, responsável pela comunicação via protocolo MQTT e gerenciamento das mensagens; o Node-RED, utilizado para processamento, organização e encaminhamento dos dados recebidos; o InfluxDB, empregado na criação de bancos de dados de séries temporais para armazenamento das informações de telemetria; e o Grafana, utilizado para visualização gráfica e análise dos dados.

Com o objetivo de facilitar a portabilidade do sistema para diferentes máquinas — como laptops utilizados em testes de campo — e garantir consistência no funcionamento, optou-se pelo uso de contêineres Docker. Essa abordagem permite que os serviços operem em ambientes isolados, evitando conflitos com outros softwares e assegurando a padronização da configuração do sistema.

Considerando a necessidade de executar múltiplos serviços simultaneamente, foi utilizado o Docker Compose, ferramenta que permite a orquestração de diversos contêineres, possibilitando sua execução conjunta, gerenciamento centralizado e comunicação por meio de redes virtuais isoladas.

Para tal, foi criado um arquivo de configuração denominado *docker-compose.yml(*código disponível no Apêndice A), no qual foram definidos os serviços do sistema, suas respectivas imagens, portas de acesso e volumes persistentes. Essas configurações garantem tanto a comunicação entre os contêineres quanto a persistência dos dados, mesmo após reinicializações do sistema, além de facilitar a replicação do sistema em diferentes dispositivos.

Testes iniciais foram conduzidos para validar o funcionamento do ambiente. Utilizando o Node-RED, foram gerados dados numéricos arbitrários, enviados via protocolo MQTT pelo Mosquitto e posteriormente armazenados em um *bucket* de teste no InfluxDB. Ressalta-se que, para a integração entre Node-RED e InfluxDB, foi necessária a instalação do pacote *node-red-contrib-influxdb*, responsável por disponibilizar os nós de comunicação entre as duas ferramentas.

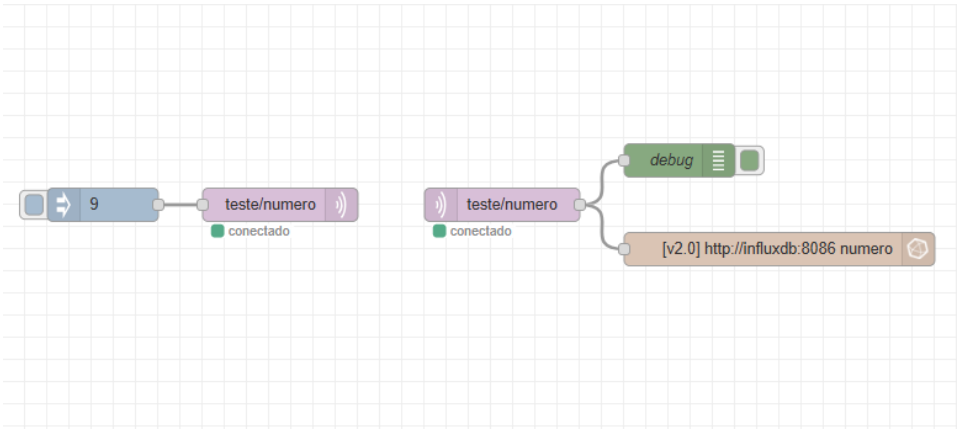

Imagem 2 - Fluxograma Node-RED Teste de Integração com Mosquitto e InfluxDB

Com o ambiente digital consolidado, iniciou-se a integração do microcontrolador ESP32 ao sistema, atuando como unidade de aquisição e transmissão de dados. Para validação inicial, foi desenvolvido um circuito simples com potenciômetro, permitindo a leitura de um sinal analógico de 12 bits, posteriormente convertido em tensão por meio de relação proporcional.

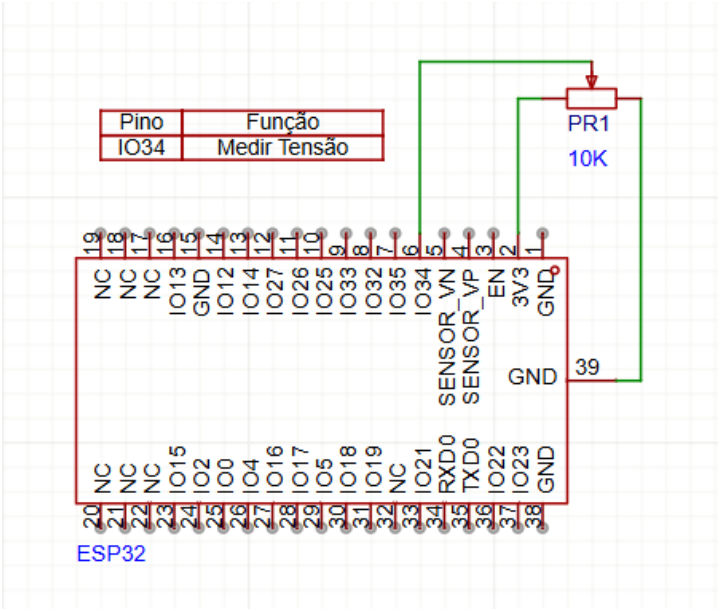

Imagem 3 - Circuito de Teste Comunicação MQTT com Potenciômetro

O principal objetivo dessa etapa foi verificar a comunicação do microcontrolador com o sistema via rede Wi-Fi, utilizando um ponto de acesso (*hotspot*). Para isso, foram empregadas as bibliotecas "*WiFi*" (nativa do ESP32) e "*PubSubClient*", de Nick O'Leary, responsável pela implementação do protocolo MQTT(Apêndice B). Os dados obtidos foram enviados a um *bucket* específico no InfluxDB, permitindo a análise de sua variação ao longo do tempo.

Após a validação da comunicação, iniciou-se a implementação de sensores. Foi utilizado o sensor inercial MPU6050, capaz de fornecer dados de aceleração e velocidade angular nos três eixos (*pitch, roll e yaw*), sendo essencial para aplicações de controle e estabilização em veículos aéreos não tripulados. A comunicação com o sensor foi realizada por meio do protocolo I2C, utilizando a biblioteca "*Wire*", que permite a troca de dados através das linhas SDA (dados) e SCL (clock), o codigo desse ensaio pode ser acessado no Apendice C.

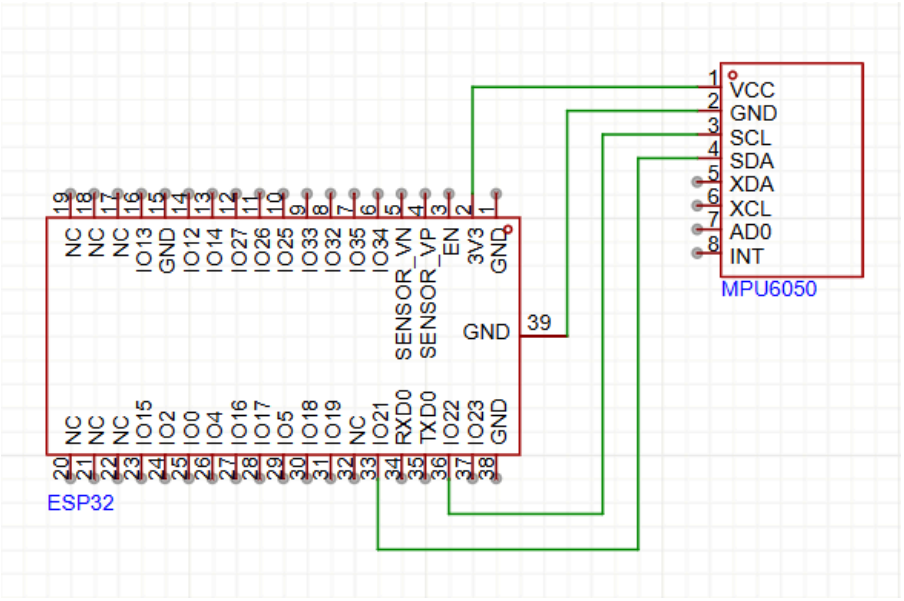

Imagem 4 - Circuito de Teste MQTT MPU6050

Essa etapa também possibilitou a validação do envio simultâneo de múltiplas variáveis, exigindo a organização adequada dos dados tanto no código quanto no ambiente Node-RED. Os dados referentes aos três eixos foram armazenados em um mesmo *bucket* no InfluxDB, permitindo sua análise conjunta.

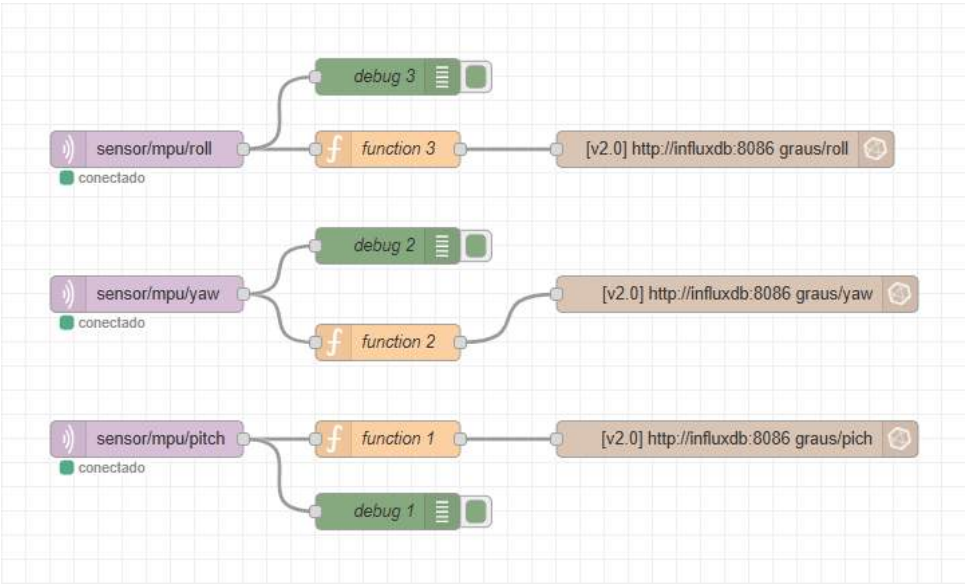

Imagem 5 - Fluxograma Node-RED de Direcionamento dos Dados do MPU6050

Na sequência, foi desenvolvido o sistema de medição das grandezas elétricas da bateria (tensão e corrente). Inicialmente, considerou-se a aplicação de um método de linearização baseado em mínimos quadrados para correção da não linearidade dos conversores analógico-digitais (ADC) do ESP32. Contudo, verificou-se que versões mais recentes do microcontrolador possuem o método *analogReadMilliVolts()*, que utiliza calibração interna de fábrica, fornecendo medições diretamente em milivolts com maior precisão.

Foi utilizada uma bateria LiPo de tensão nominal de 3,7 V e capacidade de 480 mAh. Para simulação de carga, foi empregado um resistor de 39 Ω, dimensionado para estabelecer uma corrente aproximada de 0,2C (~96 mA). A medição da tensão foi realizada por meio de um divisor resistivo, reduzindo o valor máximo de 4,2 V para aproximadamente 2,1 V, compatível com o limite de entrada do ADC do ESP32 (3,3 V).

Para a medição de corrente, foi utilizado um resistor shunt de baixa resistência (1 Ω) em série com a carga. Devido à baixa queda de tensão sobre esse componente, foi necessário o uso de um amplificador operacional para amplificação do sinal, permitindo sua leitura pelo ADC. A corrente foi então determinada por meio da aplicação da Lei de Ohm.

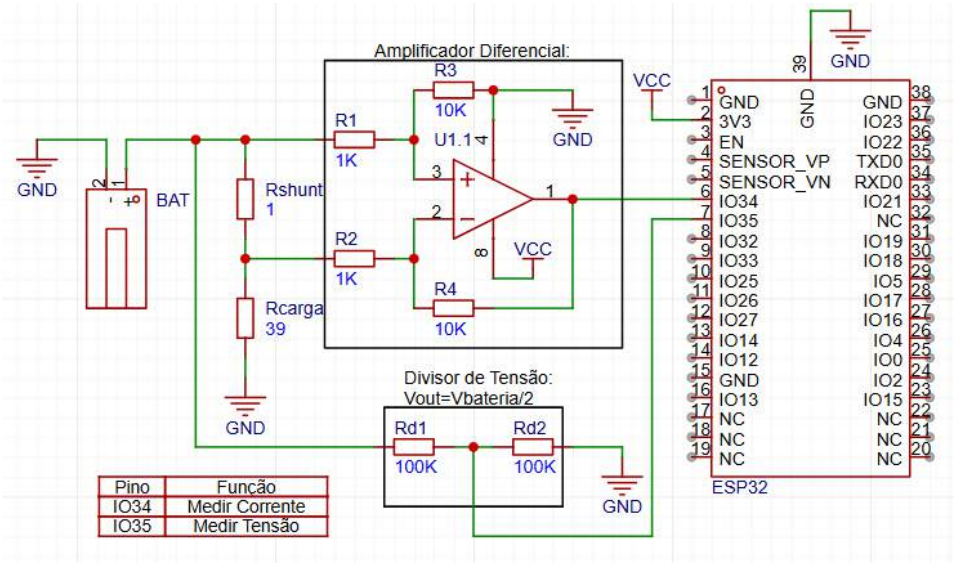

Imagem 6 - Circuito de Teste Medição das Grandezas Elétricas da Bateria

O circuito apresentado foi simulado com o objetivo de verificar os níveis de tensão nos terminais do microcontrolador. Para isso, utilizou-se uma fonte do tipo PWL (*Piecewise Linear*), configurada para gerar uma tensão variável ao longo do tempo, decaindo de 4,2 V até 3 V, de modo a representar a descarga de uma bateria. A simulação transitória permitiu a análise das tensões na saída da fonte (Vbateria), no divisor de tensão (Vbateria/2) e na saída do amplificador operacional (Vshunt × ganho).

Observa-se que a tensão no divisor de tensão (volprobe2) apresenta aproximadamente metade do valor medido na saída da fonte (volprobe1), conforme esperado. Adicionalmente, nota-se que a tensão na saída do amplificador (volprobe3) é menor, pois, apesar do ganho aproximado de 10 vezes, a queda de tensão sobre o resistor shunt é muito reduzida, resultando em um sinal de menor amplitude.

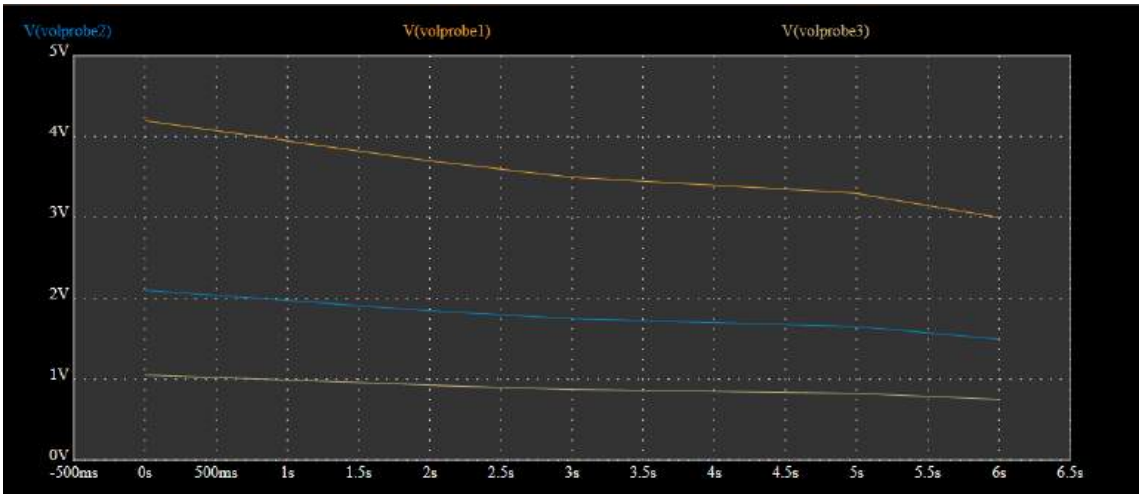

Imagem 7 - Gráfico de Simulação Transitória Circuito de Teste Medição das Grandezas Elétricas da Bateria

No software embarcado, além da configuração de comunicação Wi-Fi e MQTT, foram implementadas rotinas de leitura e processamento dos dados. Para aumentar a confiabilidade das medições, foram realizadas médias de 20 amostras em intervalos curtos. A tensão foi medida na GPIO34 e ajustada pelo fator do divisor resistivo, enquanto a corrente foi obtida na GPIO35, sendo corrigida pelo ganho do amplificador e pelo valor do resistor shunt.

Posteriormente, os códigos de leitura do sensor MPU6050 e das medições elétricas foram integrados em um único programa, reorganizado de forma modular para facilitar futuras expansões. As variáveis e funções foram estruturadas de maneira padronizada, permitindo a inclusão de novos sensores de forma simplificada. Os dados foram enviados ao ambiente de telemetria por meio do protocolo MQTT, sendo posteriormente organizados no Node-RED e armazenados em diferentes *buckets* no InfluxDB.

Considerando que o sistema embarcado também executará algoritmos de controle do veículo aéreo, foi necessária a adoção de um mecanismo de priorização de tarefas. Para isso, foi utilizado o sistema operacional de tempo real FreeRTOS, que permite a divisão do programa em múltiplas tarefas com diferentes níveis de prioridade. Dessa forma, as funções críticas de controle podem ser executadas com maior prioridade, enquanto as tarefas de telemetria são realizadas de forma assíncrona, sem comprometer o desempenho do sistema.

Além disso, explorou-se a arquitetura dual-core do ESP32, que possui dois núcleos de processamento independentes. Nesse contexto, foi adotada uma estratégia de separação funcional, na qual as tarefas de telemetria foram alocadas no núcleo 0, responsável também pelo gerenciamento da comunicação Wi-Fi, enquanto as rotinas de controle do veículo foram executadas no núcleo 1. Essa abordagem permite isolar o processamento crítico de controle das tarefas de comunicação e aquisição de dados, reduzindo interferências e garantindo maior estabilidade e previsibilidade temporal ao sistema. Dessa forma, assegura-se que eventuais atrasos na transmissão de dados não impactem negativamente o desempenho do controle do veículo, que demanda execução contínua e determinística.

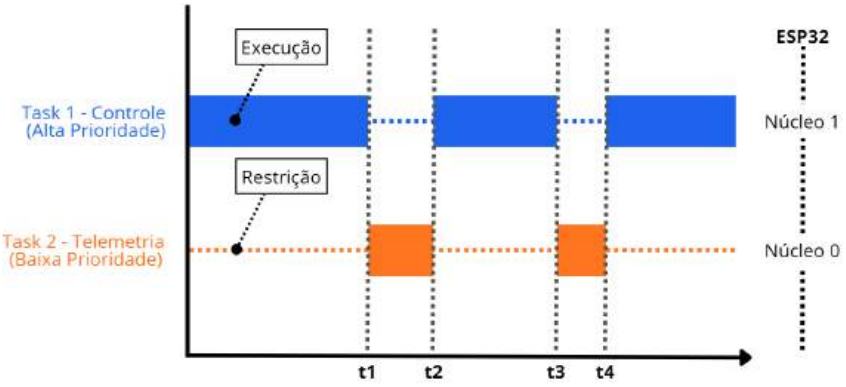

Imagem 8 - Diagrama de escalonamento de tarefas freeRTOS no ESP32

Adicionalmente, foi implementada a estratégia de envio consolidado dos dados de telemetria em uma única mensagem MQTT(código disponível no Apêndice D). Essa abordagem reduz a sobrecarga de comunicação e permite a posterior separação e organização dos dados no Node-RED, que também foi aprimorado com o uso da biblioteca *Node-RED Dashboard* para visualização gráfica em tempo real.

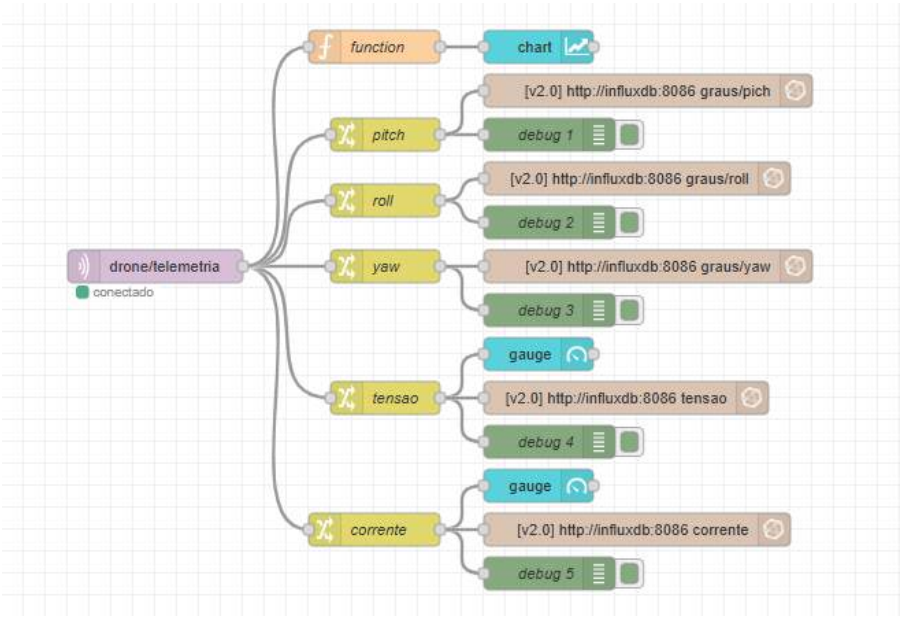

Imagem 9 - Fluxograma Node-RED Dados do MPU6050 e da Bateria com Dashboard

5 Resultados e discussão dos resultados

6 Conclusões e perspectivas de trabalhos futuros

Referências

DESAI, A.; NAYYAR, A.; MAHAPATRA, B. *IoT: Communication Protocols and Security Threats*. International Journal of Computer Applications, v. 113, n. 1, p. 1–13, 2023.

MISHRA, B.; KERTÉSZ, A. *The Use of MQTT in M2M and IoT Systems: A Survey*. IEEE Access, v. 8, p. 201071–201086, 2020.

NAWAWI, H. M. et al. *Applications of drone in disaster management: A scoping review*. Science & Justice, 2022.

RAO, A. et al. *Optimizing Electric Vehicle Efficiency with Real-Time Telemetry using Machine Learning*. arXiv preprint, 2023.

SICHONANY, O. R. A. O. et al. *Telemetria na transmissão de dados de desempenho de máquinas agrícolas utilizando tecnologias GSM/GPRS e ZigBee*. Ciência Rural, v. 53, n. 3, p. e20210675, 2012.

TOWNSEND, B.; ABAWAJY, J.; KIM, T.-H. *SMS-Based Medical Diagnostic Telemetry Data Transmission Protocol for Medical Sensors*. Sensors, v. 10, n. 6, p. 5364–5381, 2011.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 7. ed. Pearson, São Paulo, 2005.

NASA. *Medical Telemetry*. NASA Technical Reports Server, 1978.

MARÍN SANDVIK. *Telemetry: Connectivity and Productivity in Real Time – Project Implementation Guide*. 2020.

FREERTOS. Mastering the FreeRTOS Real-Time Kernel. Version 1.1.0, 2020.

NODE-RED. *Node-RED: Flow-based programming for the Internet of Things*. 2025. Disponível em: https://nodered.org.

INFLUXDATA. *InfluxDB: Time Series Database*. 2025. Disponível em: https://www.influxdata.com.

ECLIPSE FOUNDATION. *Eclipse Mosquitto – An open source MQTT broker*. 2025. Disponível em: https://mosquitto.org.

EASYEDA TEAM. *EasyEDA - Online PCB design tool*. 2025. Disponível em: https://easyeda.com.

GRAFANA LABS. *Grafana - The open observability platform*. 2025. Disponível em: https://grafana.com.

DOCKER INC. Docker Documentation. 2025. Disponível em: https://docs.docker.com.

ESPRESSIF SYSTEMS. *ESP32 Series Datasheet Version 5.0*. Espressif Systems, 2022. Disponível em: https://www.espressif.com.

APÊNDICE A – Código YML Docker Compose

```
version: "3.9"

services:
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    restart: always
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./mosquitto/config:/mosquitto/config
      - ./mosquitto/data:/mosquitto/data
      - ./mosquitto/log:/mosquitto/log
    networks:
      - iot-net

  nodered:
    image: nodered/node-red
    container_name: nodered
    restart: always
    ports:
      - "1880:1880"
    volumes:
      - ./nodered_data:/data
    depends_on:
      - mosquitto
    networks:
      - iot-net
```

```
 influxdb:
   image: influxdb:2.7
   container_name: influxdb
   restart: always
   ports:
    - "8086:8086"
   volumes:
    - ./influxdb_data:/var/lib/influxdb2

   networks:
    - iot-net

 grafana:
   image: grafana/grafana
   container_name: grafana
   restart: always
   ports:
    - "3000:3000"
   volumes:
    - ./grafana_data:/var/lib/grafana
   depends_on:
    - influxdb
   networks:
    - iot-net

networks:
 iot-net:
   driver: bridge
```

APÊNDICE B – Código C++ Teste Comunicação MQTT com Potenciômetro

```cpp
#include <WiFi.h>
#include <PubSubClient.h>

// ---- CONFIG Wi-Fi ----
const char* ssid = "nome_rede";
const char* password = "senha";

// ---- CONFIG MQTT ----
const char* mqtt_server = "IP_maquina";
const int mqtt_port = 1883;
const char* mqtt_topic = "sensor/potenciometro";

// ---- CONFIG PINOS ----
const int potPin = 34;

// ---- Objetos WiFi e MQTT ----
WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Conectando em ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi conectado!");
  Serial.print("Endereço IP: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Tentando conexão MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("conectado!");
    } else {
      Serial.print("falhou, rc=");
      Serial.print(client.state());
      Serial.println(" tentando novamente em 5s");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);

  analogReadResolution(12);  // 0-4095
```

```
}


void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  int raw = analogRead(potPin);         // 0 a 4095
  float vout = (raw / 4095.0) * 3.3;    // Converte para volts

  Serial.print("Leitura: ");
  Serial.print(raw);
  Serial.print("  |  Tensão: ");
  Serial.print(vout);
  Serial.println(" V");

  // Envia via MQTT
  char msg[50];
  snprintf(msg, sizeof(msg), "%.2f", vout);
  client.publish(mqtt_topic, msg);

  delay(2000);
}
```

APÊNDICE C – Código C++ Teste MQTT MPU6050

```
#include <MPU6050_tockn.h>
#include <Wire.h>
#include <WiFi.h>
#include <PubSubClient.h>

// ---- CONFIG Wi-Fi ----
const char* ssid = "nome_rede"; //hotspot
const char* password = "senha"; //senha

// ---- CONFIG MQTT ----
const char* mqtt_server = "IP_maquina"; //IP maquina
const int mqtt_port = 1883;
const char* mqtt_topic_pitch = "sensor/mpu/pitch";
const char* mqtt_topic_roll  = "sensor/mpu/roll";
const char* mqtt_topic_yaw   = "sensor/mpu/yaw";

// ---- OBJETOS ----
WiFiClient espClient;
PubSubClient client(espClient);
MPU6050 mpu(Wire);

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Conectando em ");
  Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi conectado!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Tentando conexão MQTT...");
    if (client.connect("ESP32_MPU6050")) {
      Serial.println("conectado!");
    } else {
      Serial.print("falhou, rc=");
      Serial.print(client.state());
      Serial.println(" tentando em 5s");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);

  // WiFi & MQTT
```

```
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);

  // MPU6050
  Wire.begin(21, 22);
  mpu.begin();
  mpu.calcGyroOffsets(true);

  Serial.println("MPU6050 inicializado!");
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  mpu.update();

  float pitch = mpu.getAngleX();
  float roll  = mpu.getAngleY();
  float yaw   = mpu.getAngleZ();

  // Debug serial
  Serial.print("Pitch: "); Serial.print(pitch);
  Serial.print(" | Roll: "); Serial.print(roll);
  Serial.print(" | Yaw: ");  Serial.println(yaw);

  // Publica no MQTT
  char msgPitch[10], msgRoll[10], msgYaw[10];
  dtostrf(pitch, 6, 2, msgPitch);
```

```
  dtostrf(roll,  6, 2, msgRoll);
  dtostrf(yaw,   6, 2, msgYaw);

  client.publish(mqtt_topic_pitch, msgPitch);
  client.publish(mqtt_topic_roll,  msgRoll);
  client.publish(mqtt_topic_yaw,   msgYaw);

  delay(30);
}
```

APÊNDICE D – Código C++ Geral do Sistema IoT com freeRTOS

```
#include <MPU6050_tockn.h>
#include <Wire.h>
#include <WiFi.h>
#include <PubSubClient.h>

//PARAMETROS CIRCUITO
const float ganho = 20;//ganho no amplificador diferencial
const float rshunt = 1;//resistencia do resistor shunt
const float divisao_tensao = 2;

// ---- CONFIG Wi-Fi ----
const char* ssid = "nome_rede"; //hotspot
const char* password = "senha"; //senha

// ---- CONFIG MQTT ----
const char* mqtt_server = "IP_maquina"; //IP maquina
const int mqtt_port = 1883;

//TOPICOS MQTT
//const char* mqtt_topic_pitch = "sensor/mpu/pitch";
//const char* mqtt_topic_roll  = "sensor/mpu/roll";
//const char* mqtt_topic_yaw   = "sensor/mpu/yaw";
//const char* mqtt_topic_corrente  = "bateria/corrente";
//const char* mqtt_topic_tensão   = "bateria/tensão";
const char* mqtt_telemetria = "drone/telemetria";

// ---- OBJETOS ----
```

```
WiFiClient espClient;

PubSubClient client(espClient);

MPU6050 mpu(Wire);

const int pinADCT = 34;

const int pinADCI = 35;


// ---VARIAVEIS MEDIDAS---

float Vbateria = 0;

float Ibateria = 0;

volatile float pitch = 0;

volatile float roll = 0;

volatile float yaw = 0;


void setup_wifi() {

  vTaskDelay(10 / portTICK_PERIOD_MS);;

  Serial.println();

  Serial.print("Conectando em ");

  Serial.println(ssid);


  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {

    vTaskDelay(500 / portTICK_PERIOD_MS);;

    Serial.print(".");

  }


  Serial.println("\nWiFi conectado!");

  Serial.print("IP: ");

  Serial.println(WiFi.localIP());

}
```

```
void reconnect() {
  if (client.connected()) return;

  Serial.print("Tentando conexão MQTT...");

  if (client.connect("ESP32_MPU6050")) {
    Serial.println("conectado!");
  } else {
    Serial.print("falhou, rc=");
    Serial.println(client.state());
  }
}

void setup() {
  Serial.begin(115200);

//xTaskCreatePinnedToCore(função,    nome,    stack,    parametros,
prioridade, alteração manual, core);

   xTaskCreatePinnedToCore(TaskControle, "Controle", 4096, NULL, 2,
NULL, 1);
  xTaskCreatePinnedToCore(TaskTelemetria, "Telemetria", 8192, NULL,
1, NULL, 0);

}

void medida_VBateria(){

long somaV = 0;
```

```
    for (int i = 0; i < 20; i++) {
      int leituraV = analogReadMilliVolts(pinADCT);
      somaV += leituraV;
      vTaskDelay(1 / portTICK_PERIOD_MS);;
    }
    float ADCbitsV = somaV / 20.0;
      Vbateria = ADCbitsV * divisao_tensao; //calcula tensão pelo polinomio de calibração na forma de Horner
    // Debug serial
    Serial.print("Tensão Bateria: "); Serial.print(Vbateria);
}

void medida_I(){

long somaI = 0;
    for (int i = 0; i < 20; i++) {
      int leituraI = analogReadMilliVolts(pinADCI);
      somaI += leituraI;
      vTaskDelay(1 / portTICK_PERIOD_MS);;
    }
    float ADCbitsI = somaI / 20.0;
     Ibateria = ((ADCbitsI/ganho)*divisao_tensao)/rshunt; //estimar corrente
    // Debug serial
    Serial.print(" | Corrente: "); Serial.println(Ibateria);
}

void medidas_MPU6050(){

mpu.update();
```

```
  pitch = mpu.getAngleX();
  roll  = mpu.getAngleY();
  yaw   = mpu.getAngleZ();

  // Debug serial
  Serial.print("Pitch: "); Serial.print(pitch);
  Serial.print(" | Roll: "); Serial.print(roll);
  Serial.print(" | Yaw: ");  Serial.println(yaw);

}

void TaskControle(void *pvParameters) {

// MPU6050
  Wire.begin(21, 22);
  mpu.begin();
  mpu.calcGyroOffsets(true);
  Serial.println("MPU6050 inicializado!");

  //portas motores etc..

  for(;;) {
    medidas_MPU6050(); //usado na calibração

    // CONTROLE DE VOO (SERÁ IMPLEMENTADO DEPOIS)
    // PID, motores, etc

    vTaskDelay(50 / portTICK_PERIOD_MS);
```

```
  }
}



void TaskTelemetria(void *pvParameters) {


// WiFi & MQTT
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);


//BATERIA


  pinMode(pinADCT, INPUT);
  pinMode(pinADCI, INPUT);
  analogReadResolution(11);


  for (;;){



    Serial.println("Telemetria rodando");
   if (!client.connected()) {
    reconnect();
  }
  client.loop();


  //AQUISIÇÃO DOS DADOS
  medida_VBateria();
  medida_I();
```

```
  // PUBLICA NO MQTT
char payload[200];

snprintf(payload, sizeof(payload),

"{\"pitch\":%.2f,\"roll\":%.2f,\"yaw\":%.2f,\"tensao\":%.2f,\"corre
nte\":%.2f}",
  pitch, roll, yaw, Vbateria, Ibateria
);

if (client.connected()) {
    client.publish(mqtt_telemetria, payload);
  }

  vTaskDelay(pdMS_TO_TICKS(1000)); // 1Hz pra teste

  }
}
 void loop() {
  vTaskDelay(portMAX_DELAY);
}
```