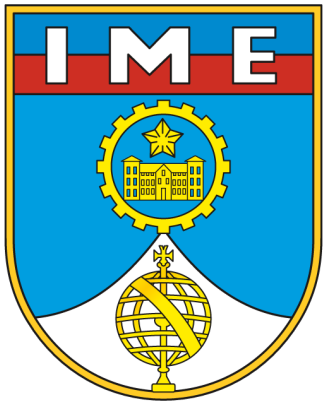
MINISTÉRIO DA DEFESA
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA

(REAL ACADEMIA DE ARTILHARIA, FORTIFICAÇÃO E DESENHO, 1792)



SEÇÃO DE ENGENHARIA ELÉTRICA (SE/3)

PROGRAMAÇÃO APLICADA À ENGENHARIA ELÉTRICA

RELATÓRIO FINAL

PROFESSOR: TEN NÍCOLAS OLIVEIRA

COMPONENTES DO GRUPO:

PEDRO **RUI ROCHA** DA FONSECA - 23063

PEDRO ARTHUR DORTA **MONTENEGRO** - 23061

LUCAS GOMES TAVEIRA SAMPAIO KUBRUSLY - 23041

Relatório Técnico

November 13, 2025

Contents

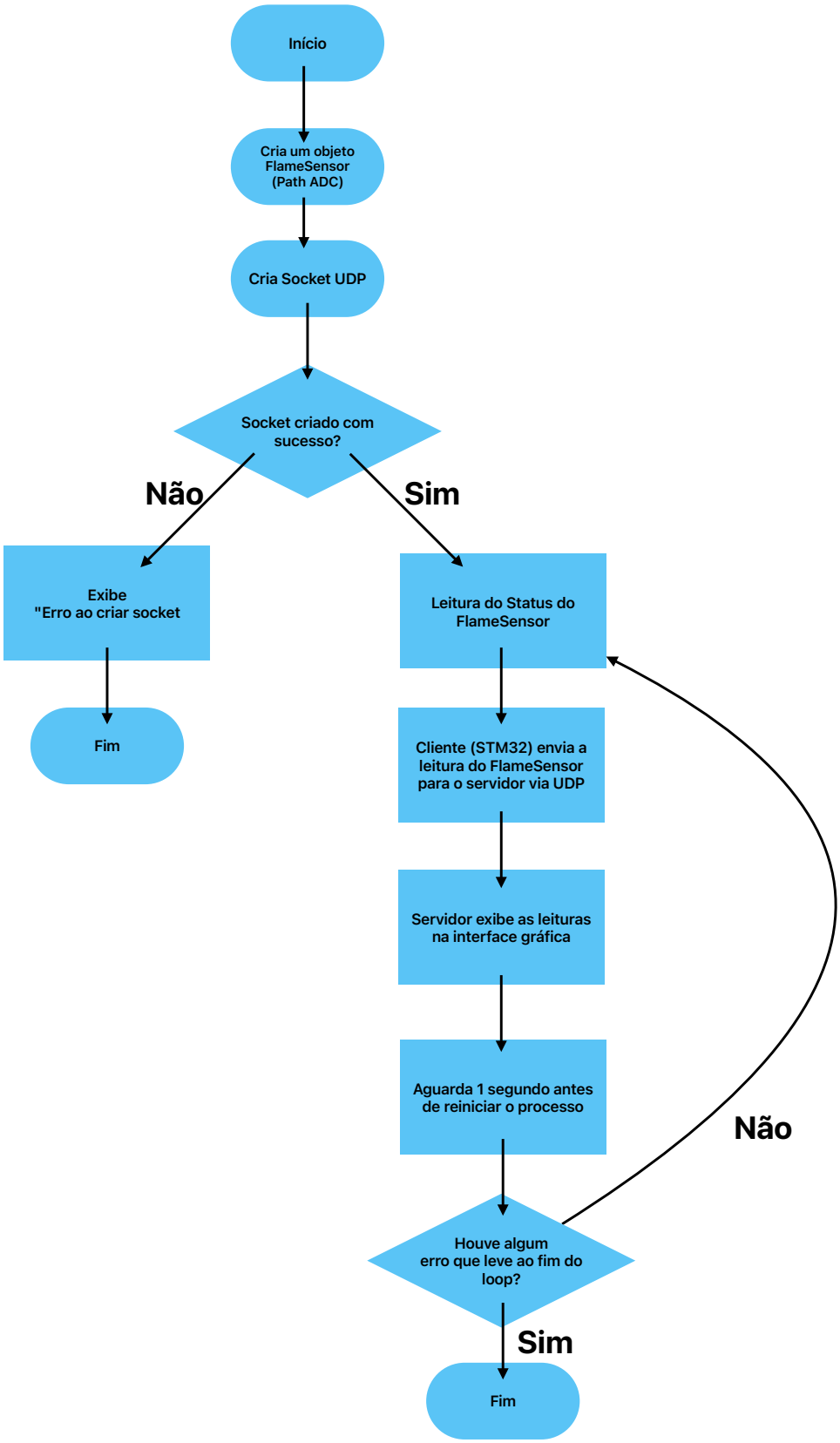
1 Introdução

O presente relatório tem como objetivo documentar o desenvolvimento de um sistema embarcado de monitoramento de cargas sensíveis, responsável por detecção de chamas que possam comprometer a sua integridade, aplicando conceitos de programação à engenharia elétrica. O projeto consiste na integração entre hardware e software, unindo um sensor de chama infravermelho conectado à placa STM32MP1 e uma aplicação em C++ capaz de enviar leituras para um servidor via protocolo UDP.

A proposta surgiu no contexto da disciplina Programação Aplicada à Engenharia Elétrica, com o intuito de proporcionar aos alunos uma experiência prática em programação de sistemas embarcados, comunicação em rede e visualização de dados em tempo real. A partir da leitura do sensor, o sistema identifica a presença de fogo e transmite o estado detectado para uma interface gráfica, que exibe o histórico das medições de forma dinâmica e intuitiva.

Além de explorar aspectos práticos de comunicação cliente-servidor e manipulação de sockets, o projeto também aborda a documentação estruturada do código utilizando Doxygen, reforçando a importância de boas práticas de engenharia de software em aplicações embarcadas

2 Fluxograma



3 Diagrama de classes

O diagrama de classes utilizado neste projeto encontra-se documentado no arquivo `Doc_FlameSensor.pdf` gerado pela aplicação **Doxygen**, que contém a estrutura completa das classes, seus relacionamentos e principais métodos. O documento se encontra no repositório Git e por esse motivo, o diagrama não será reproduzido neste relatório.

4 Instruções de compilação e uso

4.1 Clonar o Projeto

Clone o repositório e acesse o diretório do projeto com os comandos abaixo:

```
git clone https://github.com/Rui-rock/Trabalho_ProgAp/tree/main
cd Trabalho_ProgAp
```

4.2 Baixar a Toolchain

Baixe o arquivo `arm-buildroot-linux-gnueabihf_sdk-buildroot.tar.gz` no **Link**. Esta toolchain permitirá a compilação cruzada para o kit de desenvolvimento **DK1**.

4.3 Extraindo a Toolchain ARM

```
tar -xvf arm-buildroot-linux-gnueabihf_sdk-buildroot.tar.gz
```

Exemplo de Execução

O programa, quando executado corretamente, resultará em um loop exibindo em tempo real se há ou não uma chama detectada.

4.4 Compilação do Programa

```
arm-linux-gnueabihf-g++ -o flame_sensor_arm Cliente_flame.cpp -
   std=c++17
```

4.5 Execução na STM32MP1

Envie o binário para a placa via `scp`:

```
scp -O flame_sensor_arm root@<ip_da_placa>:/root
```

No terminal da placa, execute os comandos:

```
chmod +x flame_sensor_arm
./flame_sensor_arm
```

O programa exibirá continuamente o status da chama.

Exemplo de Saída

```
Valor XXXX: Não há chama próxima
Valor XXXX: Não há chama próxima
Valor XXXX: Há chama próxima
```

5 Documentação do projeto, do código e do protocolo

Esta seção descreve a organização do projeto, como o código foi estruturado e o protocolo de comunicação utilizado entre o Cliente (rodando na placa STM32) e o Servidor (rodando no PC).

5.1 Documentação do projeto

O objetivo do projeto é detectar a presença de chama utilizando um sensor infravermelho conectado à placa STM32MP1 e transmitir o estado lido para um servidor via comunicação UDP. A interface exibe a presença ou não de chama.

A filosofia do projeto foi: apenas dois códigos em C++ contendo o funcionamento do backend — um para o cliente e outro para o servidor, e um código em Python encarregado pelo frontend.

Arquitetura geral

O **cliente** é responsável por:

- Ler o valor de tensão produzido pelo sensor de chama;
- Comparar a leitura com um valor de threshold e classificar a presença ou não de chama;
- Enviar a tensão lida e seu estado correspondente por UDP para o servidor.

O **servidor** é responsável por:

- Receber datagramas UDP enviados pela placa;
- Armazenar as leituras no arquivo `log.txt`.

A **interface** é responsável por:

- Ler as informações em `log.txt`;
- Exibir as leituras recebidas em tempo real;
- Permitir exportação do histórico de leituras de um período de 40 segundos tanto em arquivo .log quando .csv;
- Alertar para leituras que extrapolam os limites gráficos pré-definidos

5.2 Documentação do código

Os códigos foram desenvolvidos e organizados em três arquivos principais:

- **Cliente_flame.cpp**: executado na placa STM32. Realiza a leitura do sensor de chama, cria o socket UDP e envia os dados para o servidor.
- **Servidor_flame.cpp**: executado no computador. Estabelece a comunicação via socket com o cliente e recebe as leituras do sensor.
- **interface.py**: executado no computador. Faz o tratamento visual das leituras captadas.

Esses códigos foram divididos em funções, facilitando a leitura e a manutenção. Além disso, o repositório contém as pastas `/latex/` e `/html/`, geradas automaticamente pelo **Doxygen**, permitindo exportar a documentação do código de leitura e transmissão de dados respectivamente em PDF ou HTML.

Todos os códigos citados estão no repositório Git do projeto.

5.3 Documentação do protocolo de comunicação

Após obter a leitura do sensor de chama, o sistema embarcado envia as informações para um servidor remoto via protocolo UDP (*User Datagram Protocol*).

Parâmetro	**Valor**
IP do servidor	192.168.42.10
IP da placa STM32MP1-DK1	192.168.42.2
Porta UDP	5000
Intervalo de envio	1 segundo
Formato da mensagem	Valor XXXX – "Há chama próxima" ou "Não há chama próxima"

Table 1: Parâmetros de comunicação via UDP

Funcionamento do Cliente UDP

1. O programa cria um *socket* UDP (`SOCK_DGRAM`) para envio dos pacotes.
2. A cada segundo, a classe `FlameSensor` lê o valor do ADC e decide se há ou não chama.
3. A mensagem correspondente é enviada ao IP do servidor via função:

```
sendto(sock, status.c_str(), status.size(), 0,
       (sockaddr*)&servAddr, sizeof(servAddr));
```

4. O servidor escuta na porta 5000 e exibe cada mensagem recebida.

Protocolo de Mensagem

O formato da mensagem enviado conforme a tabela 1 inclui a leitura da tensão lida pelo sensor (valor inteiro) e uma string "Há chama próxima" ou "Não há chama próxima".

A mensagem é enviada para a máquina executando o código de servidor, e armazena no arquivo `log.txt`, para que depois as informações sejam extraídas e expostas pela interface gráfica.

O UDP apresenta, neste caso, como vantagens a baixa latência, o overhead mínimo e a maior simplicidade de implementação em relação a protocolos como o TCP.

Esses aspectos são críticos para um projeto de sistemas embarcados, pois garantem que a comunicação entre os módulos ocorra de forma rápida, eficiente e com baixo consumo de recursos, atendendo às restrições de tempo real, de memória e de processamento típicas desse tipo de sistema.

6 Descrição do sensor e funcionamento

O sensor utilizado no projeto é um **sensor de chama infravermelho (IR)** baseado em um fotodiodo sensível ao espectro emitido por chamas reais, tipicamente entre **760 nm e 1100 nm**. Esse tipo de sensor detecta a radiação infravermelha pulsante característica de fogo e não apenas luz visível.

6.1 Componentes do módulo de chama

O módulo utilizado contém três elementos principais:

- **Fotodiodo IR** — converte a radiação infravermelha em sinal elétrico.
- **Comparador LM393** — converte o sinal analógico em digital (0/1).
- **Potenciômetro de ajuste** — regula a sensibilidade do comparador.

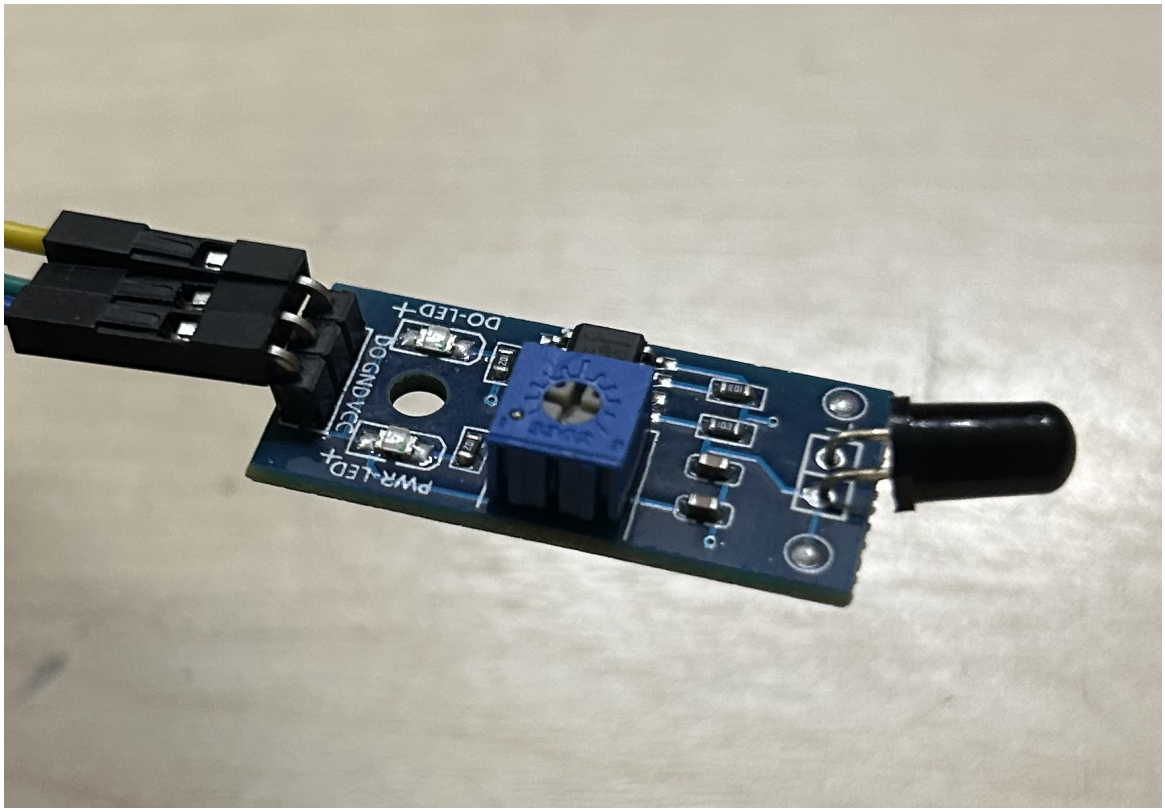


Figure 1: Módulo de chama utilizado.

6.2 Lógica de funcionamento

O módulo possui saida:

- **D0 (digital)** – definida pelo comparador LM393.

A saída **digital D0** é a utilizada neste projeto.

Importante: Ativo-baixo. A maioria dos módulos LM393 trabalha com saída digital **ativo-baixo**, ou seja:

$$D0 = 0 \Rightarrow \text{Chama detectada}$$

$$D0 = 1 \Rightarrow \text{Nenhuma chama}$$

Esse comportamento foi confirmado durante os testes.

6.3 Ligação com a placa STM32

A leitura foi feita conectando-se a saída digital do sensor ao GPIO configurado como entrada. Exemplo de ligação:

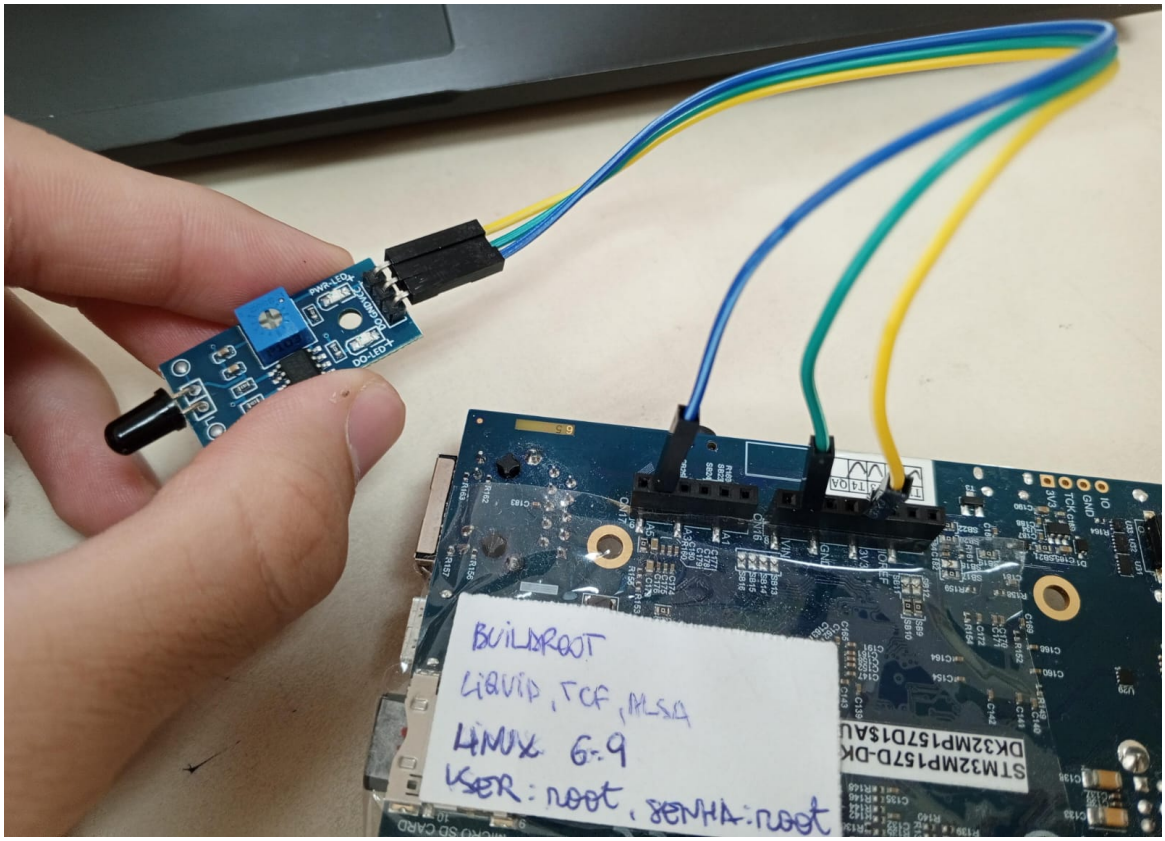


Figure 2: Configuração de conexão entre placa STM32 e sensor de chama .

- VCC → 3.3 V da placa
- GND → GND comum
- D0 → GPIO configurado no código (`Cliente_flame.cpp`)

O programa executa a leitura contínua do pino, armazena o valor e envia via UDP.

7 Capturas de tela da interface

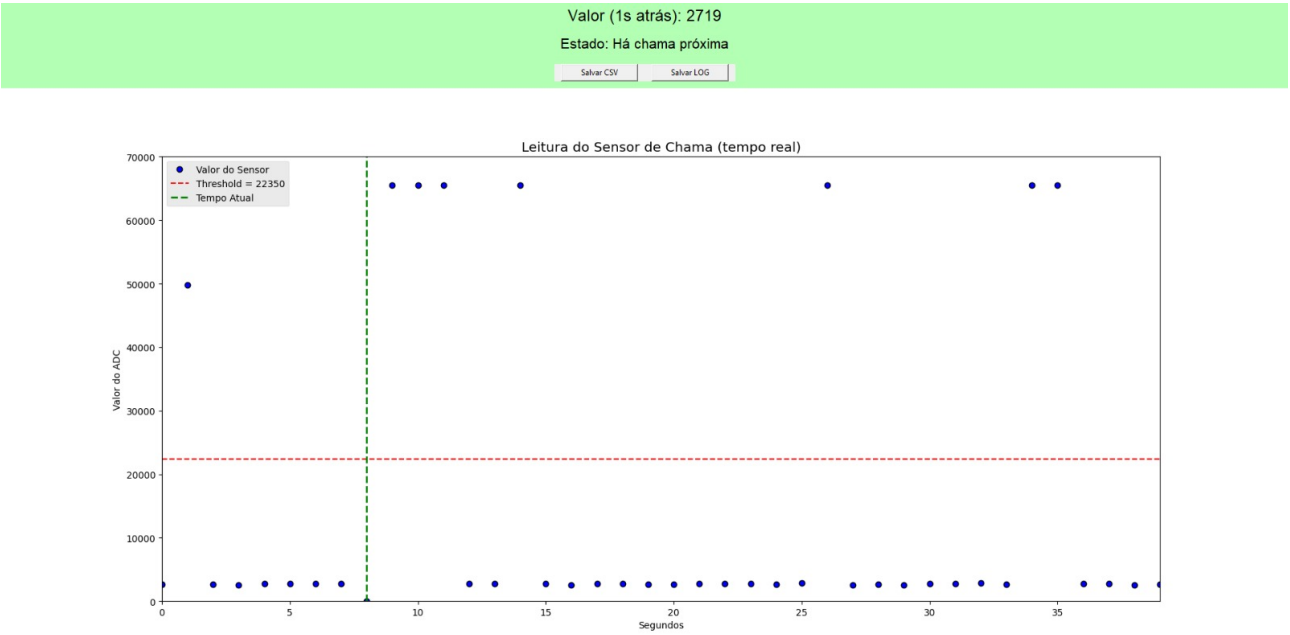


Figure 3: Imagem da interface em funcionamento

8 Análise dos valores medidos

A interface extrai as informações contidas em `log.txt` e as exibe conforme ilustrado na Figura 3. Uma linha vermelha é traçada no gráfico para indicar o valor de threshold configurado. Quando o sensor detecta a presença de chama, a interface posiciona um marcador azul abaixo dessa linha, no nível de tensão correspondente registrado em `log.txt`. De forma análoga, quando não há detecção de chama, o marcador é exibido acima do limiar.

O algoritmo é executado a cada segundo, atualizando o gráfico com um novo ponto e formando um histórico visual das leituras. Esse histórico permanece visível por 40 segundos. Após esse período, o gráfico é automaticamente reiniciado, e o processo de atualização recomeça.

A interface também oferece a opção de exportar as leituras para arquivos nos formatos .log e .csv, permitindo o armazenamento e a análise posterior dos dados.

9 Conclusão

O projeto desenvolvido demonstrou como a programação aplicada à engenharia elétrica é uma ferramenta relavante para desenvolver sistemas embarcados integrados e funcionais. Nesse sentido, a implementação do sensor de chama, a comunicação via UDP e a interface gráfica mostraram-se eficazes na detecção e exibição em tempo real do estado do sistema.

A estrutura modular e o uso de ferramentas como o Doxygen e a STM32MP1 evidenciaram a importância da integração entre hardware, software e documentação técnica, consolidando o aprendizado sobre desenvolvimento orientado a sistemas distribuídos.

Por fim, o trabalho ressaltou a importância de desenvolver soluções simples, eficientes e devidamente documentadas, capazes de atender aos requisitos do projeto — como o monitora-

mento contínuo do sistema e a preservação da integridade da carga contra chamas — garantindo assim confiabilidade no trabalho desenvolvido.