ITEC - Instituto de Tecnologia

ECP171 - Arquitetura e Programação de Microcontroladores

Projeto Final

Controlador de Temperatura com STM32H563zi

Professor: Júlio César dos Santos
E-mail: juliosantos@upf.br

14 de novembro de 2025

Sumário

1 Objetivo **1**

2 Diagrama de Blocos do Sistema **1**
2.1 IHM – Display Nextion NX8048P070 2
2.2 Telas da IHM 3
2.2.1 Tela 1: Estado Atual e Ajustes do Controlador 3
2.2.2 Tela 2: Modo Manual – Aquecedor e Ventilador 3

3 Sensor de Temperatura **4**

4 MCU: STM32H565ZI **5**

5 Driver de Potência **6**

6 Aquecedor **6**

7 Ventilador **7**

8 Heart Beat **8**

9 Lógica de Acionamento das Saídas **8**
9.1 Modo Automático 8
9.2 Zona Morta (Faixa de Inatividade) 9
9.3 Modo Manual 9
9.4 Driver Desabilitado 9

10 Resultados Esperados **10**

11 Recomendações para o Firmware **10**

12 Anexo A
Implementação de um Controlador Proporcional **11**

Anexo A – Implementação de um Controlador Proporcional **11**

1 Objetivo

Projetar e implementar um sistema completo de **controle de temperatura**, capaz de aquecer e resfriar um ambiente com base na temperatura medida e no valor programado pelo usuário (*set-point*). O sistema deverá ser desenvolvido utilizando o microcontrolador **STM32H565ZI**, fazendo uso obrigatório dos seguintes recursos de hardware:

- **ADC (Analog-to-Digital Converter)** para leitura do sensor de temperatura (LM35);
- **PWM (Pulse Width Modulation)** gerado a partir de temporizadores internos para controlar proporcionalmente o aquecedor e o ventilador;
- **USART / UART** para comunicação serial bidirecional com o display.
- **GPIOs digitais** para indicação visual do estado do sistema através do LED *Heart Beat*;
- **Estrutura modular de firmware** em linguagem C, utilizando STM32CubeIDE e biblioteca HAL, organizada em múltiplos arquivos (`.h` e `.c`);
- Implementação de uma **Máquina de Estados** para organizar os modos de operação (Automático, Manual e Segurança).

O controlador deverá ser do tipo **Proporcional (P)**, aplicando o ganho K_p ao erro entre o valor de processo (**PV**) e o *set-point* (**SP**), de modo a gerar níveis de PWM entre 0% e 100% para as cargas de aquecimento e resfriamento.

O aluno deverá desenvolver tanto a lógica de controle quanto a interface gráfica no Nextion, integrando hardware e firmware de forma segura, estável e coerente com as limitações físicas dos componentes utilizados.

2 Diagrama de Blocos do Sistema

O diagrama de blocos geral do sistema é apresentado na Figura 1

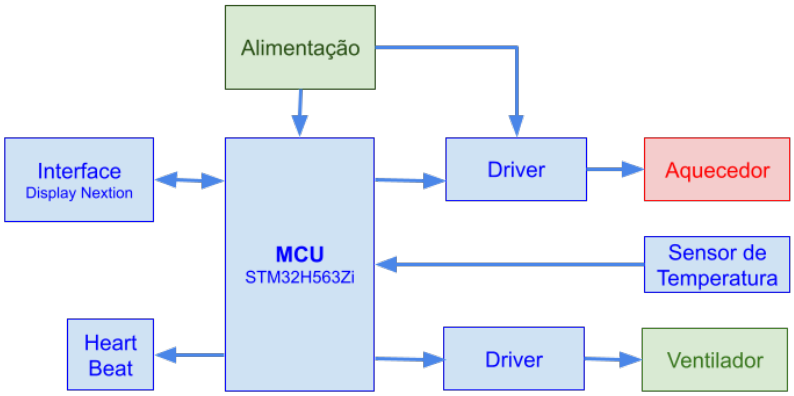

Figura 1: Diagrama de blocos geral do sistema de controle de temperatura.

- Sensor de Temperatura (LM35) → Conversor A/D (STM32H565ZI)
- STM32H565ZI: leitura da temperatura, cálculo do erro, aplicação do ganho proporcional e geração dos sinais PWM
- Driver de potência (L293D ou equivalente) para aquecedor e ventilador
- Saídas de potência: aquecedor (resistor de potência) e ventilador (motor DC)
- Interface homem-máquina (IHM): **Display Nextion NX8048P070** (7") para ajuste de parâmetros e visualização de estados

Interface com o Usuário

2.1 IHM – Display Nextion NX8048P070

O bloco de interface deverá utilizar obrigatoriamente um display **Nextion NX8048P070 (7")** como IHM principal do sistema. O projeto gráfico (páginas, botões, campos de texto, etc.) será desenvolvido pelos alunos no *Nextion Editor*.

O display deverá apresentar, no mínimo:

- Temperatura atual (**PV** – *Process Value*);
- Temperatura desejada (**SP** – *Set-Point*);
- Estado atual das saídas:
  - Aquecedor: `ON/OFF`;
  - Ventilador: `ON/OFF`;
  - Driver geral: Habilitado/Desabilitado.

Através de botões, sliders ou campos numéricos na IHM, o usuário deverá ser capaz de:

- Ajustar um novo **set-point** de temperatura;
- Ajustar o valor do **ganho proporcional** do controlador (Kp);
- Habilitar/Desabilitar o controlador (desligar o driver independente do ajuste de temperatura);
- Ligar/Desligar manualmente o aquecedor;
- Ajustar o valor da saída do aquecedor no modo manual (0–100%);
- Ligar/Desligar manualmente o ventilador;
- Ajustar o valor da saída do ventilador no modo manual (0–100%).

A comunicação entre o STM32H565ZI e o display Nextion será feita via UART (serial), utilizando protocolo de comandos do próprio Nextion (envio de textos, atualização de variáveis e leitura de eventos de toque).

2.2 Telas da IHM

O projeto deve apresentar, no mínimo, duas telas principais na IHM Nextion NX8048P070: uma tela de **monitoração e ajustes automáticos** e uma tela de **comandos manuais**.

2.2.1 Tela 1: Estado Atual e Ajustes do Controlador

Esta tela será a **tela principal** do sistema e deverá concentrar as informações de monitoração e os ajustes do modo automático. Deve conter, no mínimo:

- **SP (Set-Point)**: valor de temperatura desejada, ajustável pelo usuário;
- **PV (Process Value)**: valor de temperatura atual, medido pelo sensor (LM35);
- **Heat**: estado do aquecedor (Ligado = `ON`, Desligado = `OFF`);
- **Fan**: estado do ventilador (Ligado = `ON`, Desligado = `OFF`);
- **Driver**: estado geral do driver de saída (Habilitado = `ON`, Desabilitado = `OFF`);
- **Kp**: valor do ganho proporcional do controlador.

Nesta tela o usuário deverá ser capaz de:

- Ajustar o valor de temperatura desejada (**SP**), utilizando:
  - teclado numérico na tela; e/ou
  - botões de incremento/decremento; e/ou
  - slider (barra) de temperatura;
- Ajustar o valor do **ganho proporcional** K_p do controlador;
- Habilitar ou desabilitar o **driver de saída** do sistema.

Quando o driver estiver desabilitado:

- As saídas PWM para aquecedor e ventilador devem ser forçadas a zero;
- O LED *Heart Beat* deverá permanecer aceso continuamente.

2.2.2 Tela 2: Modo Manual – Aquecedor e Ventilador

Esta tela deverá permitir o controle **manual** das saídas, independentemente da lógica automática do controlador proporcional. O aluno poderá organizar os elementos da forma que julgar mais adequada, desde que sejam atendidos os seguintes requisitos:

- Comandos manuais para o **aquecedor**:
  - Botão para ligar/desligar o aquecedor (**Estado ON/OFF**);
  - Ajuste do valor percentual da saída no modo manual (0–100%);
  - Exibição clara do valor percentual quando em **Estado ON**;
  - Quando em **Estado OFF**, o valor percentual não deve ser exibido (ou deve ser mostrado como 0%).
- Comandos manuais para o **ventilador**:

- Botão para ligar/desligar o ventilador (**Estado ON/OFF**);
- Ajuste do valor percentual da saída no modo manual (0–100%);
- Exibição clara do valor percentual quando em **Estado ON**;
- Quando em **Estado OFF**, o valor percentual não deve ser exibido (ou deve ser mostrado como 0%).

O firmware deverá garantir que o modo manual seja claramente distinguível do modo automático, podendo ser indicado por textos, cores ou ícones específicos na própria tela. A lógica de prioridade entre o modo automático e o modo manual deve ser definida e documentada pelo aluno no código e no relatório do projeto.

3 Sensor de Temperatura

O sensor de temperatura a ser utilizado é o **LM35** Figura 2. Este sensor é do tipo analógico, com a saída variando aproximadamente **10 mV/ºC**.

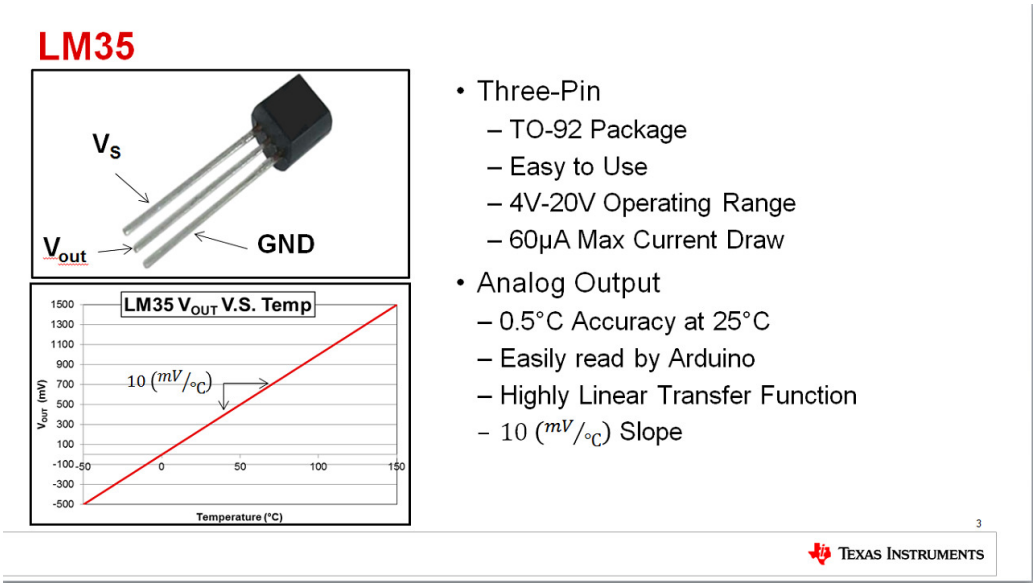

Figura 2: Diagrama de blocos geral do sistema de controle de temperatura.

Cabe ao aluno consultar o datasheet do LM35 para:

- Projetar o circuito de condicionamento adequado à faixa de temperatura desejada;
- Garantir que a faixa de tensão do LM35 esteja compatível com a entrada analógica (ADC) do STM32H565ZI;
- Definir o mapeamento entre leitura de ADC e valor de temperatura em graus Celsius.

Na Listagem 1 é apresentado a equação típica para converter o valor lido pelo ADC em Graus Celcius.

```
1 float tempC = (adc_raw * 3.3f / 4095.0f) * 100.0f;
```

Listagem 1: Conversão da leitura do ADC para temperatura em graus Celsius.

1

4 MCU: STM32H565ZI

O microcontrolador utilizado neste projeto será obrigatoriamente o **STM32H565ZI**, embarcado na placa de desenvolvimento **NUCLEO-STM32H565ZI**[1], apresentada na Figura 3.

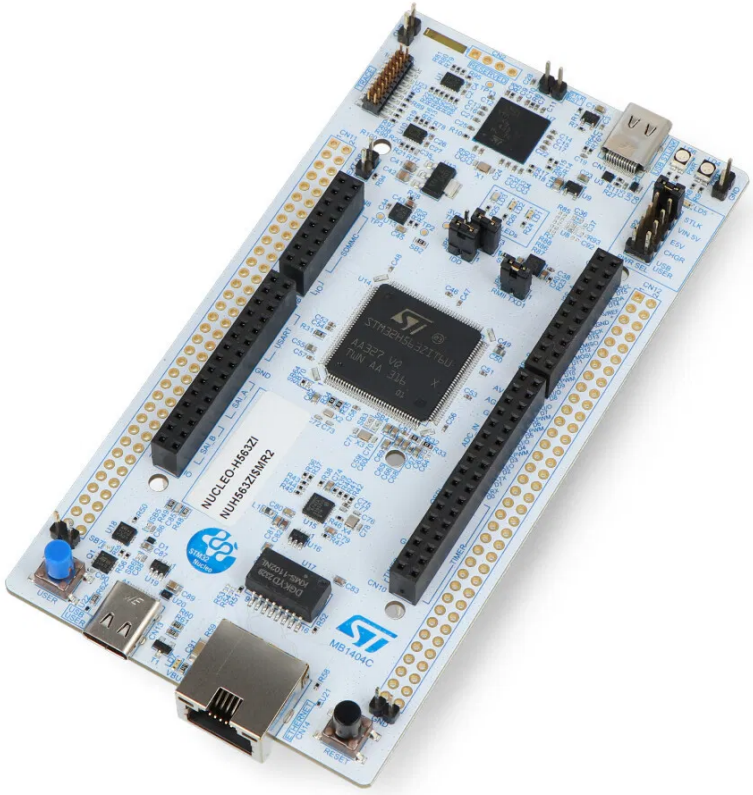

Figura 3: Placa de desenvolvimento NUCLEO-STM32H565ZI utilizada no projeto.

O STM32H565ZI será responsável por:

- Leitura do sinal analógico proveniente do LM35 através do ADC interno;
- Conversão da leitura de ADC para temperatura em ^{o}C;
- Cálculo do erro entre o set-point (SP) e o valor de processo (PV);
- Aplicação do ganho proporcional Kp;
- Geração de sinais PWM para o aquecedor e o ventilador;
- Comunicação via UART com o display Nextion NX8048P070;
- Controle do LED *Heart Beat*.

Para o desenvolvimento do firmware, deve-se utilizar a IDE **IDE STM32CubeIDE**, com biblioteca HAL (Hardware Abstraction Layer) e projeto organizado em arquivos `.h` e `.c`.

[1] Página oficial da placa NUCLEO-H563ZI no site da ST

5 Driver de Potência

O driver de potência sugerido é o circuito integrado **L293D – Quadruple Half-H Drivers**[2], ou outro driver equivalente capaz de comutar as cargas do aquecedor e do ventilador a partir dos sinais PWM gerados pelo STM32H565ZI. A Figura 4 apresenta o componente utilizado.

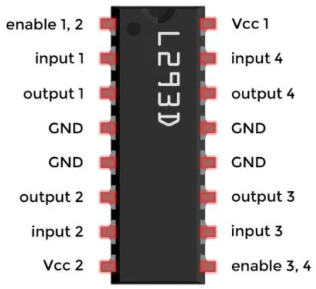

Figura 4: CI L293D utilizado como driver de potência para aquecedor e ventilador.

É responsabilidade do aluno:

- Verificar a pinagem e as limitações de corrente/tensão do L293D;
- Projetar o circuito de interface entre o driver e as cargas (aquecedor e motor do ventilador);
- Garantir proteção adequada (diodos de roda livre, se necessário, etc.).

6 Aquecedor

O aquecimento é realizado com um resistor de **22** Ω, potência de **1/2 W**, conforme ilustrado na Figura 5.

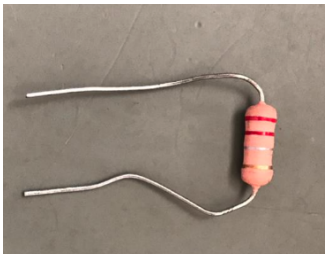

Figura 5: Resistor de 22 Ω utilizado para aquecimento.

O aluno deverá:

[2] https://www.ti.com/lit/ds/symlink/l293d.pdf

- Calcular a **tensão máxima** que pode ser aplicada a esse resistor sem ultrapassar a sua potência nominal;
- Determinar a faixa segura de operação e limitar esse valor no firmware;

Utilize a equação:

$$P = \frac{V^2}{R}$$

onde:

- P é a potência (W);
- V é a tensão aplicada (V);
- R é a resistência (Ω).

7 Ventilador

O ventilador é composto por um motor DC e uma hélice, conforme mostrado na Figura 6. O modelo de referência possui as seguintes características:

Figura 6: Ventilador utilizado no sistema.

- Tensão de alimentação: 12 VDC;
- Corrente típica: 130 mA.

O ventilador será acionado pelo driver de potência (L293D ou equivalente) com modulação PWM proveniente do STM32H565ZI.

8 Heart Beat

Um LED deverá ser utilizado para informar o estado de execução do sistema:

- Quando o sistema estiver em funcionamento normal e o driver habilitado, o LED deve **piscar** em aproximadamente 0,5 Hz;
- Quando o driver de saída estiver **desabilitado**, o LED deverá permanecer **aceso continuamente**.

9 Lógica de Acionamento das Saídas

Para garantir uma operação coerente e segura do sistema, o acionamento do **aquecedor** e do **ventilador** deverá seguir regras claras, tanto no **modo automático** quanto no **modo manual**. Esta seção descreve explicitamente o funcionamento esperado.

9.1 Modo Automático

No modo automático, o valor de saída é determinado pelo controlador proporcional (P) a partir da diferença entre o set-point (SP) e a temperatura medida (PV).

$$e(t) = SP - PV$$

Regra para o Aquecedor

O aquecedor deverá ser acionado sempre que:

$$SP > PV$$

Nessa condição:

- o erro é positivo ($e > 0$);
- o controlador gera um valor proporcional $u(t) > 0$;
- o PWM do aquecedor recebe este valor (após saturação entre 0–100%);
- o ventilador deve permanecer desligado.

Exemplo:

$$SP = 50°C, \quad PV = 42°C \Rightarrow e = 8°C$$

O sistema deve ligar o aquecedor e aplicar potência proporcional ao erro.

Regra para o Ventilador

O ventilador deverá ser acionado sempre que:

$$PV > SP$$

Nessa condição:

- o erro é negativo ($e < 0$);
- a ação de aquecimento não faz sentido (PWM do aquecedor = 0);

- a intensidade do ventilador deve ser proporcional ao módulo do erro:

$$u_{\text{fan}} = K_p \cdot |e|$$

- o aquecedor deve permanecer desligado.

Exemplo:

$$SP = 40°C, \quad PV = 47°C \Rightarrow e = -7°C$$

O sistema deve ligar o ventilador com potência proporcional ao erro.

9.2 Zona Morta (Faixa de Inatividade)

Para evitar oscilações rápidas entre aquecer e resfriar, recomenda-se implementar uma pequena **zona morta**, por exemplo:

$$|SP - PV| < 0{,}5°C$$

Nessa condição:

- aquecedor = 0%;
- ventilador = 0%;
- o sistema aguarda nova variação.

(O aluno pode ajustar ou omitir essa zona morta.)

9.3 Modo Manual

No modo manual:

- o usuário controla diretamente o PWM do aquecedor e do ventilador;
- o controlador proporcional deve ser ignorado;
- somente a saída selecionada como *ON* receberá o valor ajustado;
- se ambos forem ligados ao mesmo tempo (situação permitida a critério do aluno), o comportamento deve ser claramente indicado na documentação e no display.

9.4 Driver Desabilitado

Quando o driver de saída estiver desabilitado:

- PWM do aquecedor = 0%;
- PWM do ventilador = 0%;
- LED *Heart Beat* permanece aceso fixo.

Esse estado deve ter prioridade sobre qualquer modo de operação.

10 Resultados Esperados

Cada aluno deverá entregar:

- **Desenho esquemático** do circuito completo em formato PDF (0,1);
- **Firmware** com código modularizado, desenvolvido em C utilizando a **STM32CubeIDE** (0,4);
- **Montagem física** do sistema em bancada, com demonstração de funcionamento (0,3);
- **Repositório no GitHub** contendo o firmware, diagrama esquemático e fotos da montagem prática, com arquivo `README.md` descrevendo o projeto (0,2).

O arquivo `README.md` deve conter, no mínimo:

- Descrição do projeto e objetivos;
- Diagrama esquemático (imagem);
- Fotos da montagem prática;
- Instruções básicas de compilação e gravação do firmware.

11 Recomendações para o Firmware

- Divida o código em módulos (`.h` e `.c`) que possam ser testados separadamente;
- Fique atento às questões de hardware (pinos de PWM, entradas analógicas, UART para o Nextion, etc.);
- Utilize comentários em seções relevantes do código;
- Procure utilizar a abstração de **Máquina de Estados** para organizar o fluxo do programa;
- Utilize constantes e `#define` para criação das telas, IDs de componentes do Nextion e parâmetros do sistema;
- Documente as principais funções e arquivos no `README.md`;
- É permitido utilizar ferramentas de **Inteligência Artificial** (IA) para auxiliar na geração de código, comentários ou documentação, desde que o estudante revise e compreenda integralmente o conteúdo produzido.

Este projeto será a segunda (e última) nota da **ECP171 - Arquitetura e Programação de Microcontroladores**.

12 Anexo A
Implementação de um Controlador Proporcional

A.1 Conceito Básico

Um controlador proporcional (P) gera o sinal de controle $u(t)$ a partir do erro $e(t)$ entre o valor de referência $r(t)$ e a variável de processo $y(t)$:

$$e(t) = r(t) - y(t) \tag{1}$$

O controlador proporcional é definido por:

$$u(t) = K_p \cdot e(t) \tag{2}$$

onde:

- K_p é o ganho proporcional;
- $e(t)$ é o erro entre referência e saída;
- $u(t)$ é o sinal aplicado ao atuador (por exemplo, duty cycle de PWM).

Em implementação digital, normalmente trabalhamos em instantes discretos k:

$$u[k] = K_p \cdot e[k] = K_p \cdot (r[k] - y[k]) \tag{3}$$

A.2 Exemplo Numérico Simples

Considere um sistema de controle de temperatura com:

- Referência: $r = 50\,^{\circ}\text{C}$;
- Temperatura medida: $y = 42\,^{\circ}\text{C}$;
- Ganho proporcional: $K_p = 4\ \%/^{\circ}\text{C}$ (ou seja, para cada $1\,^{\circ}\text{C}$ de erro, o duty cycle do PWM aumenta em 4%).

Passo 1 – Cálculo do erro:

$$e = r - y = 50 - 42 = 8\,^{\circ}\text{C}$$

Passo 2 – Cálculo do sinal de controle (duty cycle em %):

$$u = K_p \cdot e = 4 \cdot 8 = 32\%$$

Logo, o PWM que aciona o aquecedor deve ser ajustado para aproximadamente **32% de duty cycle**.

Se o erro diminuir, por exemplo, $y = 48\,^{\circ}\text{C}$:

$$e = 50 - 48 = 2\,^{\circ}\text{C} \quad \Rightarrow \quad u = 4 \cdot 2 = 8\%$$

Ou seja, o controlador automaticamente reduz a potência aplicada ao aquecedor conforme a temperatura se aproxima da referência.

A.3 Conversão do Sinal de Controle para PWM (12 bits)

Suponha que o PWM do microcontrolador use um *timer* de 12 bits, com contagem de 0 a 4095. Desejamos converter o controle em % para um valor de `compare` do timer:

$$u_{\text{PWM}} = \frac{u_{\%}}{100} \cdot 4095$$

No exemplo anterior, para $u = 32\%$:

$$u_{\text{PWM}} = \frac{32}{100} \cdot 4095 \approx 1310$$

Na prática, o valor enviado ao registrador de comparação do timer seria algo em torno de `1310`.

A.4 Exemplo de Implementação em C (STM32)

A seguir, um exemplo didático de implementação de um controlador proporcional em C, considerando:

- Função `float get_temperature(void)` que retorna a temperatura medida;
- Função `void set_heater_pwm(uint16_t value)` que ajusta o PWM do aquecedor;
- Timer PWM configurado com `ARR = 4095`.

Um exemplo de código em C é apresentado na Listagem 2.
Essa função `control_loop_proporcional()` deve ser chamada periodicamente, por exemplo:

- Dentro do *superloop* principal, utilizando um atraso fixo ou contador de tempo (por exemplo, a cada 100 ms);
- Em uma interrupção de *timer* configurada para executar o laço de controle em uma taxa fixa.

A.5 Boas Práticas na Escolha de K_p

- Comece com um valor pequeno de K_p e aumente gradualmente até obter uma resposta rápida, mas sem oscilações excessivas;
- Verifique se há saturação frequente (PWM sempre em 0% ou 100%): isso indica que K_p pode estar muito alto ou que o atuador é subdimensionado;
- Registre os valores de $e[k]$, $u[k]$ e temperatura em um arquivo de log (ou envie via *serial*) para análise; alternativamente, utilize o ambiente de *debug* do **STM32CubeIDE** para inspecionar variáveis em tempo real e acompanhar o comportamento do controlador.
- Para sistemas com grande atraso térmico (como aquecedores), ajuste o ganho com cuidado, pois o efeito do controle leva alguns segundos para aparecer.

```c
/* Ganho proporcional (em % por grau Celsius) */
static const float Kp = 4.0f;

/* Referência de temperatura (setpoint) em graus Celsius */
static float temp_ref = 50.0f;

/* Valor máximo de PWM (ARR do timer) */
#define PWM_MAX 4095

void control_loop_proporcional(void)
{
    /* 1) Ler temperatura atual do sensor */
    float temp_meas = get_temperature();

    /* 2) Calcular erro: e = r - y */
    float error = temp_ref - temp_meas;

    /* 3) Calcular sinal de controle em % */
    float u_percent = Kp * error;  // pode ser negativo

    /* 4) Saturar o sinal de controle entre 0% e 100% */
    if (u_percent > 100.0f) {
        u_percent = 100.0f;
    } else if (u_percent < 0.0f) {
        u_percent = 0.0f;
    }

    /* 5) Converter para valor de PWM (0 a PWM_MAX) */
    uint16_t u_pwm = (uint16_t)((u_percent / 100.0f) * PWM_MAX);

    /* 6) Aplicar o valor ao atuador (aquecedor) */
    set_heater_pwm(u_pwm);
}
```

Listagem 2: Exemplo de implementação em C de um controlador proporcional para controle de temperatura.