Universidade do Estado do Rio de Janeiro

Faculdade de Engenharia

Departamento de Engenharia de Sistemas e Computação

Controle de Processos por Computador

Projeto Final
Parte 3

Professor: Luigi Maciel Ribeiro

Aluno: Mariana Florido Robaina

Rio de Janeiro

2025

1. Introdução

O projeto implementa o sistema de controle de temperatura para uma incubadora, adotando uma arquitetura distribuída que integra um microcontrolador ESP32 (executando o controle PID localmente) com um backend Django (hospedado no PythonAnywhere) para supervisão e monitoramento remoto.

O fluxo é dividido em dois ambientes:

- Django: Hospeda a API REST, armazena o histórico de telemetria e permite a configuração remota do setpoint.
- ESP32 - Wokwi: Executa o Controlador PID local em tempo real, monitora sensores e botões, e lida com a lógica de estados (Modo Automático, Manual e Alerta).

Tabela 1

Comunicação	Endpoint	Uso
GET	/api/v1/setpoint/<id_incubadora>/	ESP32 busca o setpoint configurado remotamente.
POST	/api/v1/historico/	ESP32 envia dados de telemetria e estado
GET	/api/v1/historico/<id_incubadora>/	Busca o histórico de uma unidade específica.

2. Simulação do sistema

A simulação do sistema foi implementada diretamente no microcontrolador ESP32 (Wokwi). MELHORAR

2.1. Configuração do sistema e modelagem

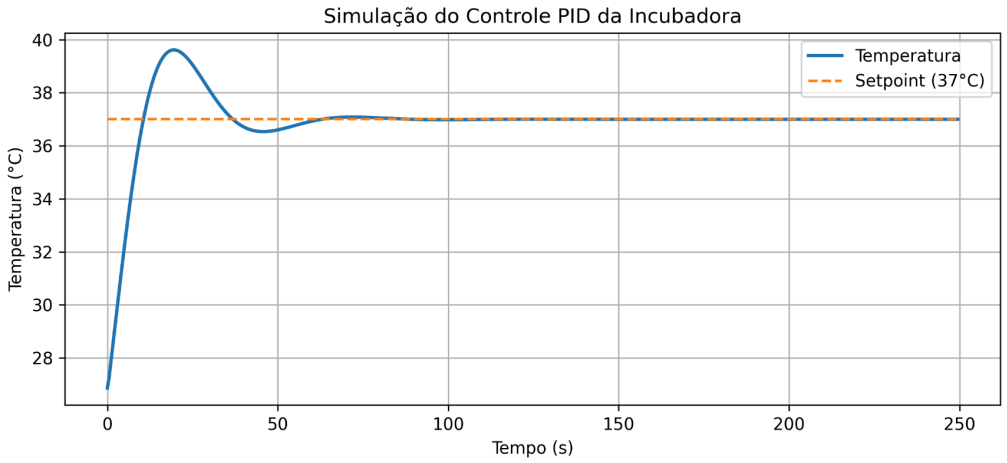
No código de simulação no Wokwi, é necessário configurar:

- O ID da incubadora, previamente registrado no dashboard.
- Uma temperatura padrão (setpoint default), utilizada em situações de perda de comunicação. Assim, mesmo sem o último setpoint salvo, o PID continua atuando corretamente.

O sensor DS18B20 realiza apenas uma leitura inicial real. Após isso, a temperatura passa a ser atualizada por um modelo térmico simulado, que representa a resposta da incubadora às ações do PID.

O PID foi inicialmente projetado e ajustado em Python para facilitar a análise, geração de gráficos e sintonia dos parâmetros. Após a validação, seus ganhos foram transferidos para o código em C++ do ESP32 no Wokwi.

A figura 1 apresenta o resultado da simulação em Python.



O controlador calcula a potência aplicada:

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

onde e(t) = setpoint - T_{atual}

2.2. Componentes do sistema

Os componentes da simulação estão apresentados na tabela 2:

Tabela 2

Componente	Função
ESP32	Microcontrolador principal
Sensor (DS18B20)	Leitura de temperatura inicial (depois, simulação)
Buzzer	Alerta de tampa aberta
LIGA/DESLIGA	Switch de parada imediata
Tampa	Switch de Segurança (Alerta)
Modo	Switch que alterna entre modo Manual/Automático

2.3. Modelagem térmica da incubadora

Para simular a dinâmica térmica de uma incubadora em um ambiente real, foi necessário modelar a relação entre a potência do atuador e a temperatura resultante. Essa

simulação permite que o controlador PIDatue sobre a potência de aquecimento (ou resfriamento), resultando em uma variação da temperatura interna, a qual é lida pelo "sensor" para fechar o ciclo de controle.

A dinâmica de temperatura da incubadora foi modelada como um sistema de primeira ordem com perdas por convecção, seguindo a Lei do Resfriamento de Newton. Este modelo representa o balanço de energia entre a potência térmica aplicada pelo atuador e a perda de calor para o ambiente externo.

O modelo matemático de temperatura do sistema é apresentado nas equações abaixo, e a descrição dos parâmetros físicos e suas respectivas unidades encontra-se na Tabela 3.

- A potência perdida é dada por:

$$P_{perda} = \frac{T_{atual} - T_{ambiente}}{R_{th}}$$

- A variação de temperatura é então:

$$\Delta T = \frac{P_{aplicada} - P_{perda}}{C_{th}} \cdot \Delta t$$

- A temperatura atualizada:

$$T_{nova} = T_{atual} - \Delta T$$

Tabela 3

Parâmetro	Símbolo	Valor	Unidade	Função
Temperatura Ambiente	$T_{ambiente}$	25.0	°C	Ponto de equilíbrio térmico da planta.
Resistência Térmica	R_{th}	5.0	°C/W	Representa o isolamento da incubadora (Perda de calor).
Capacidade Térmica	C_{th}	5.0	J/°C	Representa a inércia térmica do sistema.
Potência Máxima	P_{max}	±10.0	W	Limite de saturação do atuador (Aquecimento/Resfriamento).
Intervalo de Controle	Δt	0.5	s	Frequência do loop do PID e da simulação.

A modelagem do PID, como mencionado anteriormente, foi feita no Python, e então seu equivalente foi feito no Wokwi. O controlador PID controla a potência de atuação ($P_{aplicado}$)

para eliminar o erro ($e(t) = setpoint - T_{atual}$) a cada ciclo de 0.5 segundos. A tabela 4 mostra os parâmetros do PID.

Tabela 4

Ganho	Símbolo	Valor	Função no Sistema
Proporcional	K_p	0.5	Resposta imediata ao erro. Reduzido para amortecer a oscilação.
Integral	K_i	0.1	Eliminação do erro em regime estacionário.
Derivativo	K_d	0.01	Suaviza a resposta e antecipa mudanças bruscas.

As seguintes constantes globais, na tabela 5, regem a temporização e a precisão do controle no ESP32:

Tabela 5

Constante	Valor	Unidade	Função
POLLING_INTERVAL	15000	ms	Frequência de consulta ao Backend (API).
CONTROL_LOOP_TIME	500	ms	Frequência de execução do PID local.
PID_TOLERANCE	1	°C	Faixa de erro (±1C°) para o estado ESTÁVEL (LED Verde).
WIFI_RECONNECT_INTERVAL	3000	ms	Intervalo entre as tentativas de reconexão.

3. Execução da simulação e testes

As simulações foram executadas no ambiente Wokwi, utilizando a lógica de fluxo procedural que implementa a Máquina de Estados.

- **Cenário 1: Teste de Robustez (Perda de Conexão/Fallback)**
    - Condição: O sistema está funcionando no modo Automático e o sistema não consegue se comunicar com o backend e buscar o valor do setpoint.
    - Objetivo: Validar a transição imediata para o modo FALLBACK e a continuidade do controle local com o último setpoint registrado.

- **Cenário 2: Teste de Segurança (Alerta e Recuperação)**

- Condição: O sistema está no modo Automático. O SWITCH_TAMPA é movido para LOW (Tampa Aberta).
- Objetivo: Verificar a interrupção imediata da atuação, a ativação do Buzzer e a estabilidade do sistema ao retornar à operação normal.

- **Cenário 3: Teste de parada imediata**
  - Condição: O sistema está em qualquer um dos modos. O SWITCH_STOP é movido para LOW (Parada do sistema).
  - Objetivo: Verificar a interrupção imediata da atuação..

- **Cenário 4: Teste de troca de modo de operação**
  - Condição: O sistema está no modo Automático/Fallback. O SWITCH_MODE é movido para LOW.
  - Objetivo: Verificar se o sistema muda para modo MANUAL e se os botões de controle funcionam.

4. Análise dos Resultados

- Cenário 1: O sistema entrou em FALLBACK com sucesso.
- Cenário 2: O sistema para de funcionar com o acionamento do switch e o alarme começa a soar. Quando o switch é acionado novamente o alarme irá parar e o sistema volta a funcionar. Se o switch for acionado no momento de um POST do histórico, o sistema apresenta um bug, demonstrado na imagem abaixo (Figura 2).

```
T: 37.54°C | Set: 37.00°C | Pot: 10.11
T: 36.29°C | Set: 37.00°C | Pot: -10.27
ALERTA: Tampa Aberta! Buzzer ON.
ALERTA: Resolvido. Retornando ao Hub.
Setpoint remoto recebido: 15.30
ALERTA: Tampa Aberta! Buzzer ON.
```

- Cenário 3: O sistema para de funcionar com o acionamento do switch, quando o switch é acionado novamente o sistema é ligado novamente.
- Cenário 4: O sistema troca para o modo MANUAL sem problemas, os botões funcionam e o histórico é enviado se não houver nenhum erro de comunicação com o backend.

3. Identificação de Problemas e Sugestões de Melhorias

O principal problema observado foi o travamento do circuito quando o ESP32 iniciava o POST para enviar o histórico. Durante o envio e a espera pela confirmação do servidor (que pode levar vários segundos), o *loop* principal parava, as ações nos *switches* não eram percebidas e os LEDs congelavam no último estado. Uma melhoria que poderia solucionar esse problemas seria fazer o envio do POST assincronamente.

Além disso, apesar da eliminação do erro de regime pelo termo integral do PID, o

valor da temperatura ficava flutuando em algumas casas decimais próximo ao setpoint, para solucionar isso e o LED verde acender, foi introduzido um limiar de estabilidade (PID_TOLERANCE) de ±1.0°C.