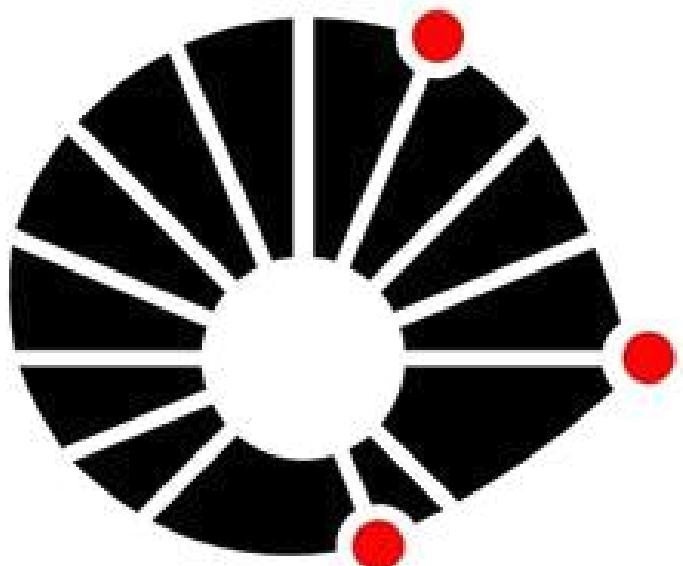


ES670 - Projeto de Sistemas Embarcados
Relatório Final



UNICAMP

<i>Nome</i>	<i>RA</i>
Eduardo Siqueira	196308
Henrique Akira	198741
Lucas Finzzeto Pavarini	182509

Sumário

1 Resumo	2
2 Documentação do sistema	3
2.1 Requisitos do projeto	3
2.2 Representação do hardware	4
2.3 Diagrama de camadas	5
2.4 Diagrama de classes	6
2.5 Fluxogramas	7
2.6 Máquina de estados	7
3 Procedimento de sintonização do controlador	9
3.1 Uso do cooler	19
4 Manual de utilização	20
4.1 Manutenção da Temperatura	20
4.2 Interface Local	20
4.3 Controle Remoto via UART	20
4.4 Alerta Sonoro	20
4.5 Controle de Temperatura com Cooler	20
4.6 Indicador Visual de Temperaturas	21
4.7 Overshoot Controlado	21
4.8 Faixa de Operação de Temperatura	21
4.9 Configuração da Interface Local	21
4.10 Configuração do Terminal Serial	21
4.11 Inserindo a Temperatura Desejada	21
4.12 Monitoramento do Sistema	21
4.13 Ajuste dos Parâmetros de Controle via UART	22
5 Manutenção do Sistema	22
5.1 Verificação Regular	22
5.2 Atualização do Software	22
6 Problemas identificados e não resolvidos	22
7 Autoria dos códigos fornecidos	24

1 Resumo

Este projeto visa desenvolver um sistema de controle de temperatura eficiente. O principal objetivo do sistema é controlar a temperatura de um resistor, ajustando-a conforme a temperatura alvo definida pelo usuário. Para atingir essa meta, o sistema manipula a corrente do resistor utilizando um sinal PWM (Pulse Width Modulation) dentro de uma malha de controle. Quando a temperatura desejada é maior do que a atual, o sistema utiliza o heather, quando é menor, utiliza o cooler.

Em termos de hardware, o sistema é constituído por um microcontrolador NUCLEO-G474RE, duas daughterboards, um display LCD e um teclado matricial (não utilizado). Uma das daughterboards é responsável por fornecer alimentação ao microcontrolador, enquanto a outra contém os elementos de interface com o usuário, além dos componentes de aquecimento e resfriamento. A interface do usuário inclui um buzzer, cinco LEDs, cinco botões, um teclado matricial com 16 teclas, um display LCD 2x16 e uma saída UART.

O sistema de aquecimento é composto por um resistor que funciona como aquecedor e uma ventoinha para resfriamento. Adicionalmente, há múltiplos sensores para garantir o funcionamento eficiente do sistema: dois sensores de temperatura, um digital e um analógico, para monitorar a temperatura do resistor; e um par de fotodiodo e LED utilizado como tacômetro para medir a velocidade de rotação da ventoinha.

Para a comunicação entre os componentes, utilizamos os protocolos I2C e UART. O protocolo I2C é empregado na comunicação entre o microcontrolador e o display LCD, enquanto o protocolo UART é usado para a comunicação entre o microcontrolador e o computador via USB.

O software do sistema foi desenvolvido em linguagem C utilizando o ambiente de desenvolvimento CubeIDE. Esse ambiente facilita a configuração de diversas funcionalidades do microcontrolador através de uma interface gráfica, permitindo a nomeação e configuração dos pinos, a configuração dos clocks e dos timers de interrupções. Essas configurações automáticas geradas pelo CubeIDE simplificam o desenvolvimento do código, que é organizado em módulos e bibliotecas para cada função, facilitando futuras modificações.

O controle de temperatura é realizado por um controlador PID que atua sobre a corrente do resistor através de um sinal PWM. A sintonia do controlador PID foi realizada utilizando o método Ziegler-Nichols, com um overshoot máximo definido de 2°C, assegurando precisão e eficiência ao sistema.

2 Documentação do sistema

2.1 Requisitos do projeto

Durante a fase inicial do projeto definiu-se os requisitos funcionais e não-funcionais.

Requisitos Funcionais

1. O sistema deve ser capaz de manter a temperatura do resistor no valor determinado pelo usuário.
2. No momento em que se solta o botão, o buzzer emite um som, independentemente de ser apenas um clique ou de estar segurando o botão.
3. O set point aumenta ou diminui em 1°C quando o botão "up" ou "down" é clicado.
4. O set point aumenta ou diminui em 10°C quando o botão "up" ou "down" é segurado, com incrementos a cada 500ms
5. O LCD possui três telas distintas. Para mudar de tela, é necessário segurar o botão "enter" por 3 segundos.
 - Primeira Tela: Exibe a temperatura atual, a temperatura desejada e a rotação do cooler.
 - Segunda Tela: Exibe a temperatura atual, a temperatura desejada, o duty cycle do aquecedor e o duty cycle do cooler.
 - Terceira Tela: Exibe os valores de K_p , K_i , K_d .
6. O sistema deverá apresentar um indicador visual das temperaturas utilizando LEDs de cores diferentes para indicar faixas de temperatura:
 - Verde para temperaturas entre 0°C e 40°C.
 - Amarelo para temperaturas entre 40°C e 60°C.
 - Vermelho para temperaturas entre 60°C e 90°C.
- Este indicador deve alterar as cores de acordo com a temperatura atual do resistor, proporcionando um feedback visual imediato sobre o estado de operação do sistema.
7. Implementada funcionalidade de ajuste dos valores de K_p , K_i , K_d via UART.

Requisitos Não Funcionais

1. O sistema deve ser capaz de aquecer o resistor o mais rápido possível para atingir a temperatura desejada em um curto período de tempo.
2. O overshoot, ou seja, a quantidade pela qual a temperatura ultrapassa o valor desejado antes de se estabilizar, deve ser de no máximo 2°C.
3. A velocidade do cooler deve ser monitorada.
4. O sistema opera entre a temperatura ambiente e 90°C. A temperatura máxima ocorre devido a limitações físicas do resistor.

2.2 Representação do hardware

A Figura 1 apresenta o hardware utilizado no projeto.

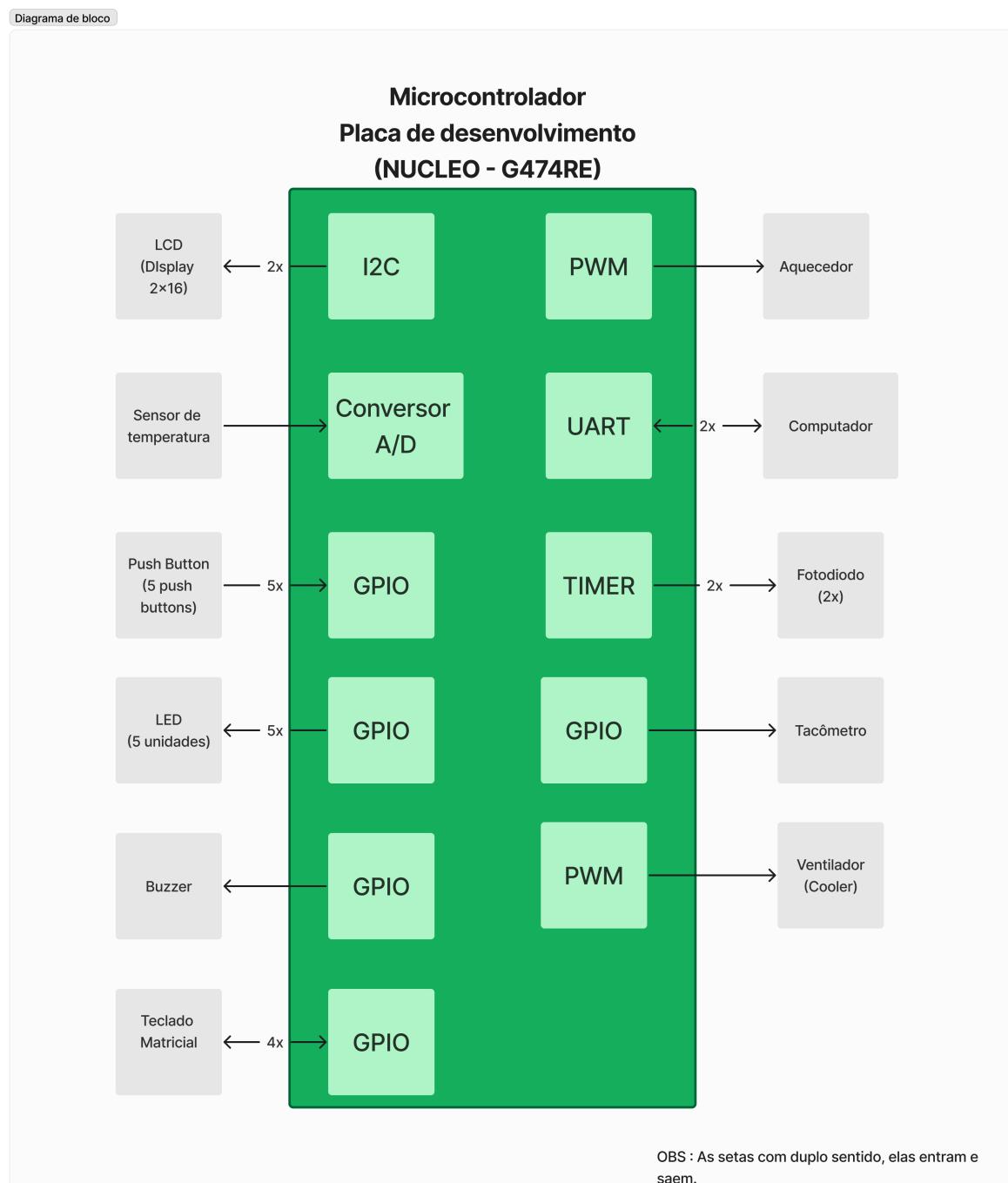


Figura 1: Diagrama de blocos

2.3 Diagrama de camadas

A Figura 2 apresenta o diagrama de camadas do projeto.

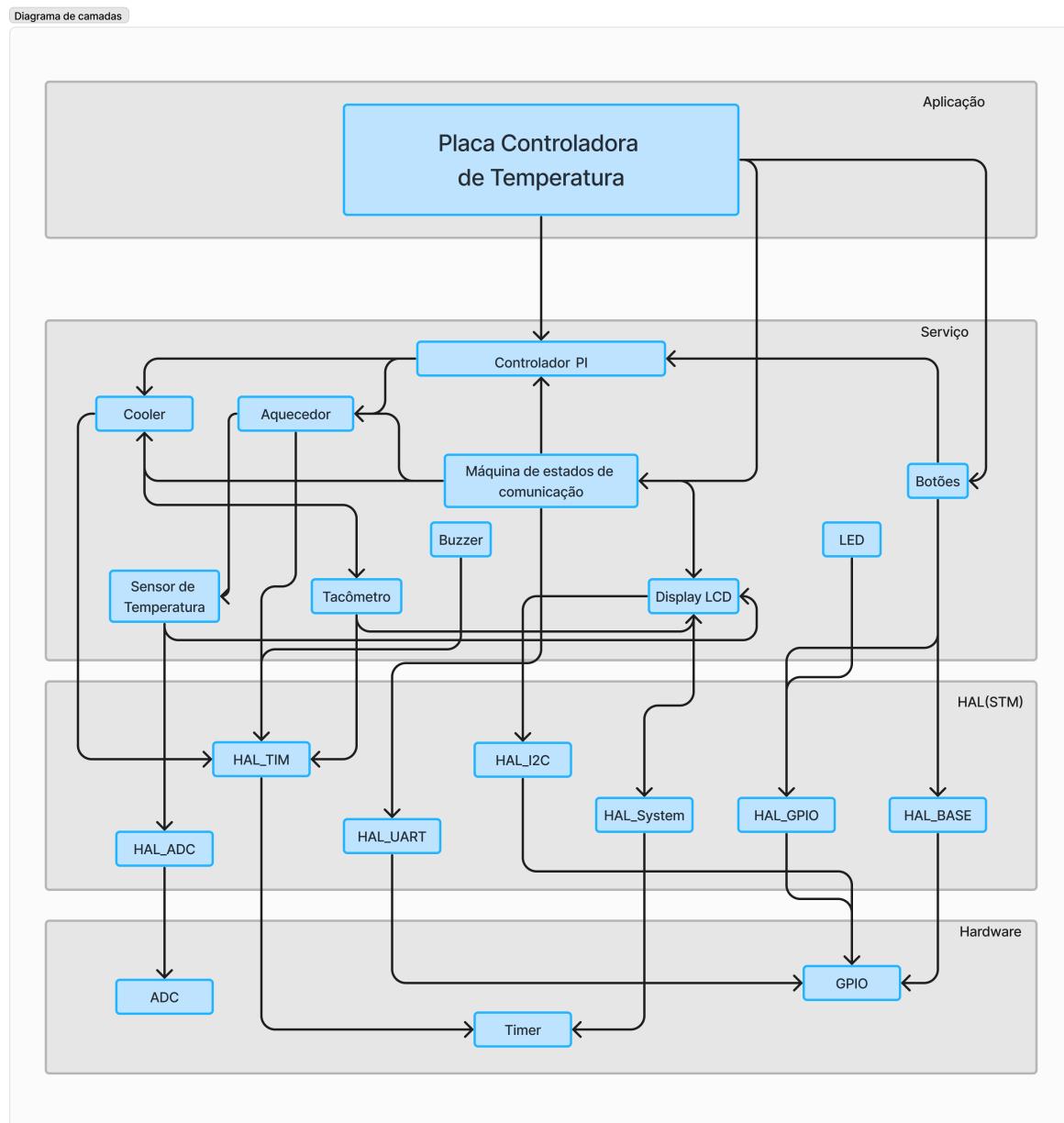


Figura 2: Diagrama de camadas

2.4 Diagrama de classes

A figura 3 apresenta o diagrama de classes do projeto.

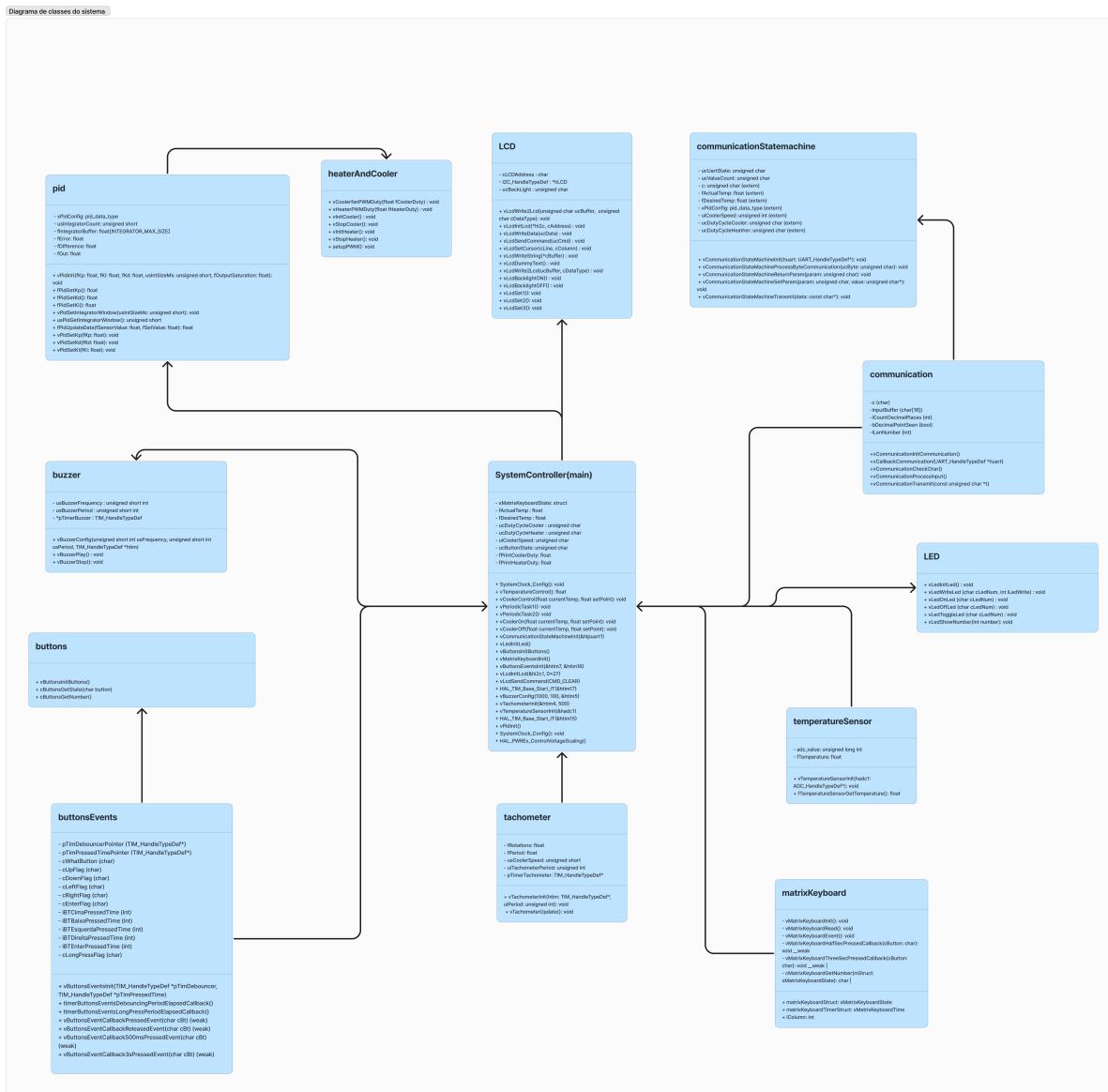


Figura 3: Diagrama de classes

2.5 Fluxogramas

A Figura 4 apresenta o fluxograma do controle de temperatura do projeto com as principais subrotinas do projeto, update LCD, update LED e update Tachometer (para atualizar a velocidade das pás do cooler). Para esse fluxograma não consideramos inputs do usuário, por isso nesse fluxograma não é possível visualizar a UART (controle do K_p , K_i , K_d); controle com o uso dos botões; buzzer (com sua funcionalidade atrelada aos botões).

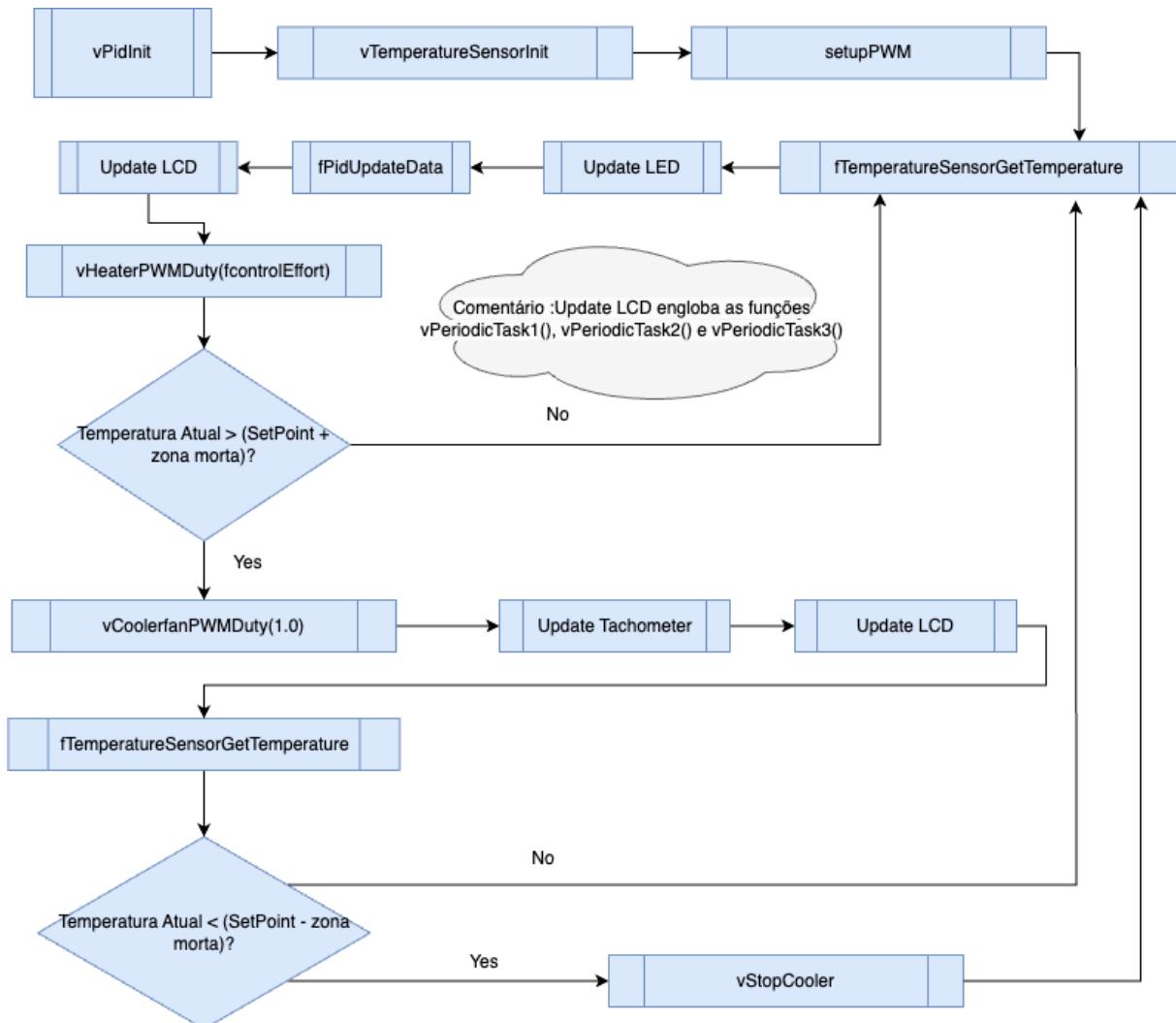


Figura 4: Fluxograma do controle de temperatura

2.6 Máquina de estados

O processamento das mensagens via comunicação UART foi implementado utilizando uma máquina de estados, conforme ilustrado abaixo. As mensagens seguem um padrão específico: devem começar com o caractere '#' e terminar com o caractere ';'. A Tabela 1 lista os comandos disponíveis.

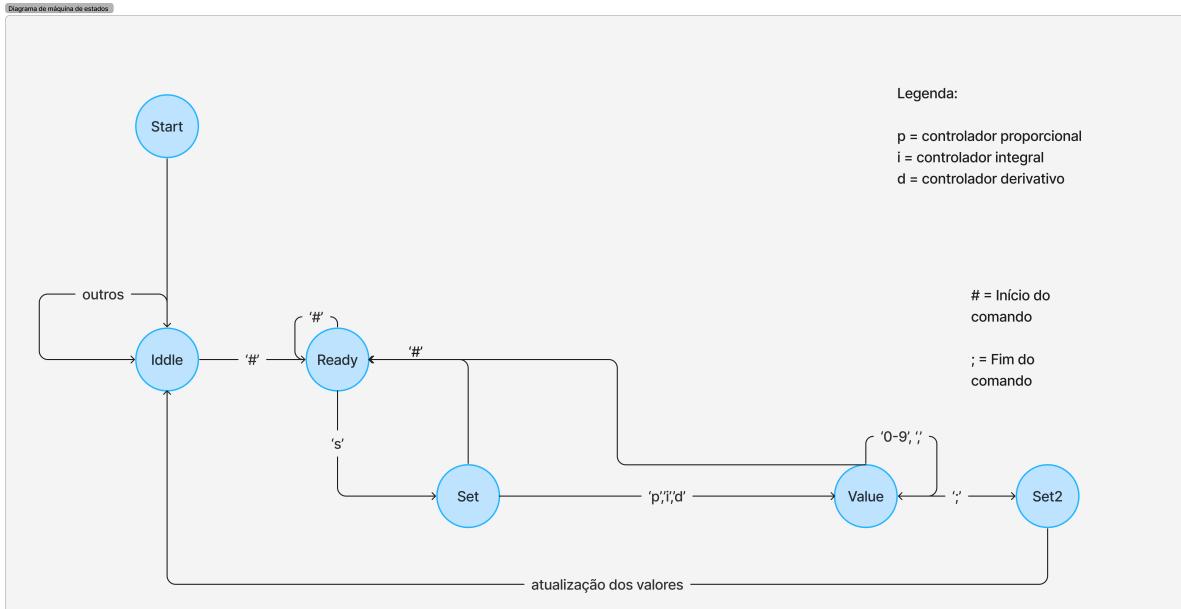


Figura 5: Máquina de estados da comunicação UART

Comando	Função	Variável
$\#sp<\text{valor}>;$	Set	Ganho proporcional
$\#si<\text{valor}>;$	Set	Ganho integral
$\#sd<\text{valor}>;$	Set	Ganho derivativo

Tabela 1: Comandos e suas respectivas funções e variáveis

3 Procedimento de sintonização do controlador

Vamos utilizar o método de *Ziegler-Nichols* em malha aberta, pois ele funciona bem para processos com constante de tempo de pelo menos duas vezes a de “tempo morto”, e isso costuma ocorrer para dados de temperatura. Além disso, não é necessário escolher aleatoriamente os ganhos K_p , K_i , K_d e a implementação é relativamente simples.

Os parâmetros iniciais obtidos com esta técnica fornecerão uma base sólida para ajustes subsequentes, garantindo que o sistema atenda aos requisitos específicos de overshoot máximo de 2°C e aquecimento rápido. A técnica de *Ziegler-Nichols* é amplamente validada na literatura e em aplicações industriais, proporcionando confiança adicional na sua aplicação para este projeto de controle de temperatura.

Uma das desvantagens do método *Ziegler-Nichols* observada na literatura é que ele pode resultar em sintonização menos precisa e overshoot elevado, vamos observar se isso irá ocorrer.

Vamos utilizar o código fornecido pelo Prof. Rodrigo M. Bacurau, para isso devemos transformar as variáveis no modo iterativo (Figura 6) encontradas no Ziegler-Nichols de malha aberta em variáveis do modo paralelo (Figura 7).

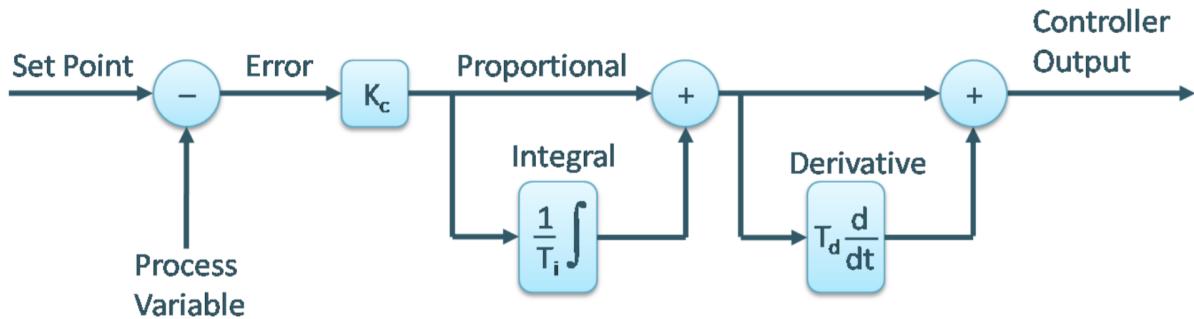


Figura 6: Modo iterativo

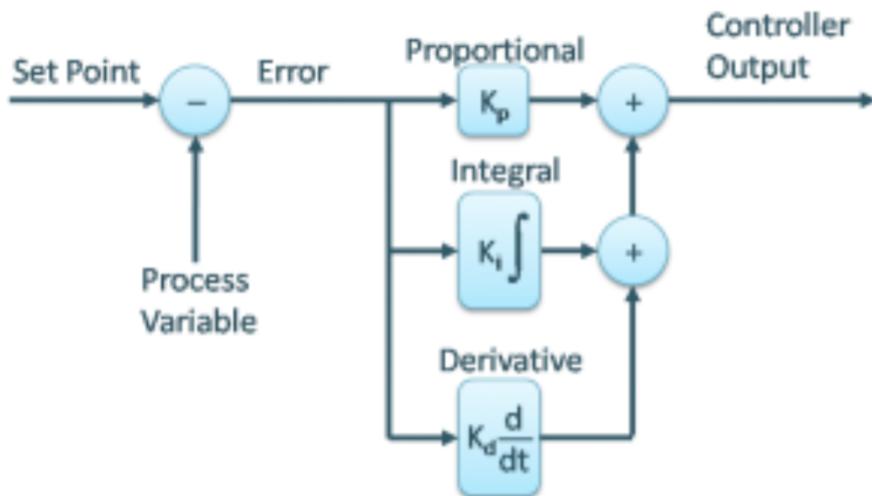


Figura 7: Modo paralelo

Para determinar t_d , τ , e g_p que são necessários para encontrar K_c , T_i e T_d , desenvolvemos um código em Python localizado na pasta do nosso repositório /jupyterNotebookForPlottingData/plottingData.ipynb, baseado no tutorial <https://blog.opticontrols.com/archives/477>.

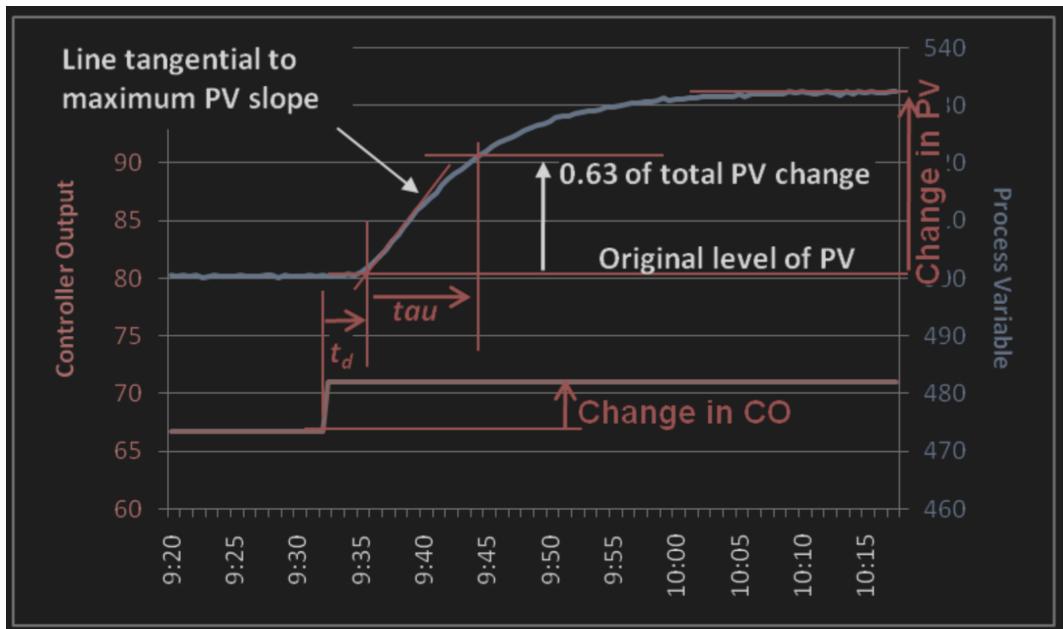


Figura 8: Modo iterativo do PID

A figura acima mostra as variáveis necessárias para encontrar os valores do modo iterativo do PID pelo método de Ziegler-Nichols de malha aberta.

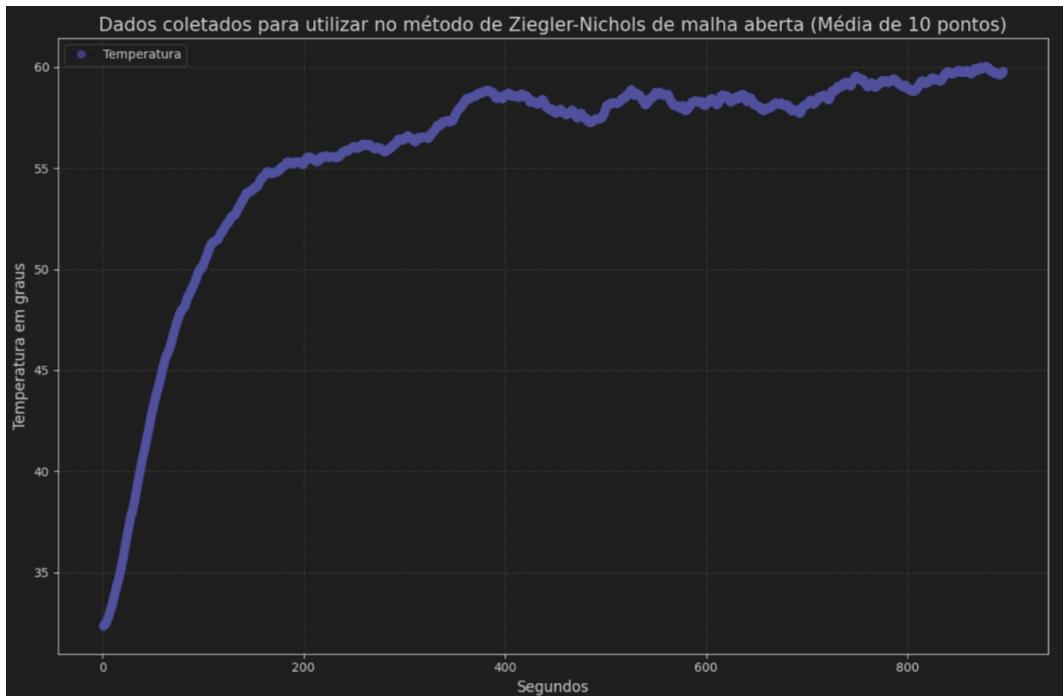


Figura 9: Dados coletados no método *Ziegles-Nichols*

Dados coletados para utilizar no método de *Ziegler-Nichols* de malha aberta com uma média móvel de 10 pontos (suavizar os dados e encontrar tendências gerais, removendo flutuações aleatórias de curto prazo)

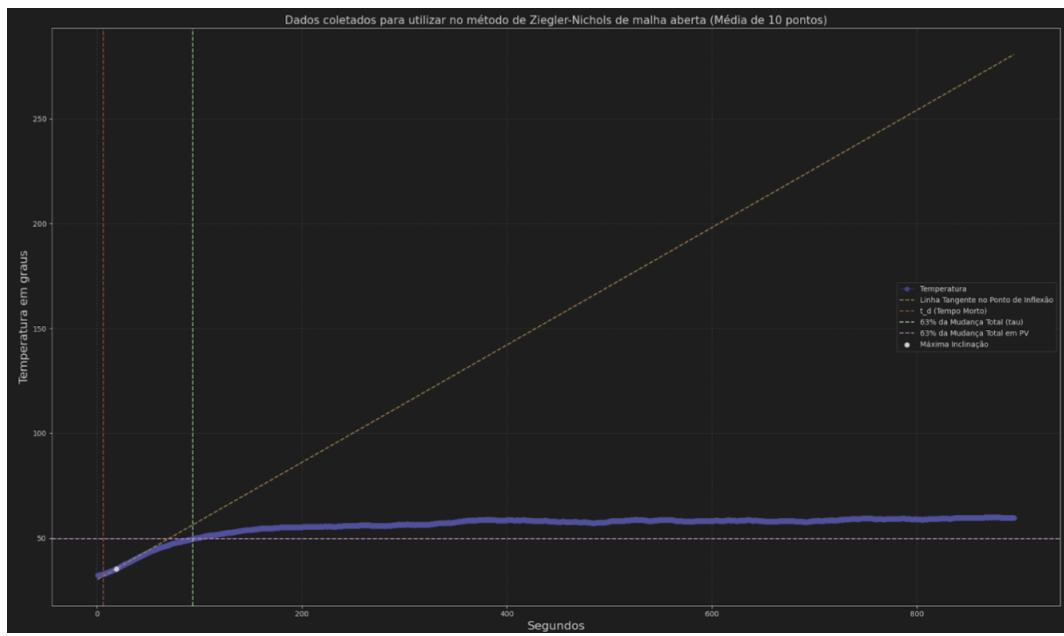


Figura 10: Determinação de t_d , τ , e g_p

```

1 #PV = Process Variable (Variável de Processo)
2 #CO = Control Output (Saída de Controle)
3
4 # Supondo que data_rolled já está carregado com os dados de 'Segundos' e 'Temperatura em graus'
5 # Identificação do Nível Inicial de PV (Temperatura Inicial)
6 initial_temp = data_rolled['Temperatura em graus'].iloc[0]
7
8 # Determinação do Valor Final de PV (Temperatura Final)
9 final_temp = data_rolled['Temperatura em graus'].iloc[-1]
10
11 # Cálculo da Mudança Total em PV
12 delta_pv = final_temp - initial_temp
13
14 # Determinação de 63% da Mudança Total em PV
15 PV_63_percent = initial_temp + 0.63 * delta_pv
16
17 # Cálculo da derivada primeira para encontrar a máxima inclinação
18 data_rolled['Derivada'] = np.gradient(data_rolled['Temperatura em graus'], data_rolled['Segundos'])
19
20 # Encontrar o tempo com a máxima inclinação (ponto de inflexão)
21 max_slope_index = data_rolled['Derivada'].idxmax()
22 max_slope_time = data_rolled['Segundos'].iloc[max_slope_index]
23 max_slope_temp = data_rolled['Temperatura em graus'].iloc[max_slope_index]
24 max_slope = data_rolled['Derivada'].iloc[max_slope_index]
25
26 # Desenhar a linha tangente
27 tangent_intercept = max_slope_temp - max_slope * max_slope_time
28 tangent_line = max_slope * data_rolled['Segundos'] + tangent_intercept
29
30 # Identifican o tempo de intersecção da linha tangente com o nível inicial de PV (Tempo morto)
31 intersection_indices = np.where(np.isclose(tangent_line, initial_temp, atol=0.5))[0]
32 if len(intersection_indices) > 0:
33     intersection_index = intersection_indices[0]
34     t_d = data_rolled['Segundos'].iloc[intersection_index]
35 else:
36     t_d = np.nan
37     print("Interseção não encontrada.")
38
39 # Identificação do Tempo de t (onde o valor de PV é 63% da mudança total)
40 for i in range(1, len(data_rolled)):
41     if data_rolled['Temperatura em graus'].iloc[i] >= PV_63_percent:
42         tau_time = data_rolled['Segundos'].iloc[i]
43         break
44
45 # Cálculo de t
46 tau = tau_time - t_d
47
48 # Mudança na entrada (CO)
49 initial_CO = 0.1 # Valor inicial da entrada (duty cycle do heater) - Esperei estabilizar com 10% de duty cycle do heater
50 final_CO = 0.7 # Valor final da entrada (duty cycle do heater)
51 delta_CO = final_CO - initial_CO
52
53 # Convertendo as mudanças em porcentagem em relação ao valor final
54 delta_pv_percent = (delta_pv / final_temp) * 100
55 delta_CO_percent = (delta_CO / final_CO) * 100
56
57 # Cálculo do Ganhos de Processo (Gp) em termos de % da faixa do dispositivo de medição
58 Gp = delta_pv_percent / delta_CO_percent
59

```

```

60 # Verificar se t_d é zero ou NaN e ajustar para um valor razoável se necessário
61 if np.isnan(t_d) or t_d == 0:
62     t_d = 5 # Ajuste para um valor apropriado baseado nos dados reais
63
64 # Cálculo dos valores para P, PI e PID controladores usando Ziegler-Nichols
65 Kc_P = tau / (Gp * t_d)
66
67 Kc_PI = 0.9 * tau / (Gp * t_d)
68 Ti_PI = 3.33 * t_d
69
70 Kc_PID = (1.2 * tau / (Gp * t_d))
71 Ti_PID = 2 * t_d
72 Td_PID = 0.5 * t_d
73
74 # Convertendo os parâmetros interativos para a forma paralela
75 Kc = Kc_PID
76 Ki = Kc_PID / Ti_PID
77 Kd = Kc_PID * Td_PID
78
79 # Exibição dos resultados
80 print(f"Ganho de Processo (Gp): {Gp}")
81 print(f"Tempo de Retardo (t_d): {t_d} segundos")
82 print(f"Constante de Tempo (tau): {tau} segundos")
83
84 print(f"Controlador P: Kc = {Kc_P}")
85 print(f"Controlador PI: Kc = {Kc_PI}, Ti = {Ti_PI}")
86 print(f"Controlador PID: Kc = {Kc_PID}, Ti = {Ti_PID}, Td = {Td_PID}")
87
88 print(f"Controlador PID Parallel Form: Kp = {Kc}")
89 print(f"Controlador PID Parallel Form: Ki = {Ki}")
90 print(f"Controlador PID Parallel Form: Kd = {Kd}")
91
92 # Plotar os dados com anotações
93 plt.figure(figsize=(20, 12))
94 plt.plot(data_rolled['Segundos'], data_rolled['Temperatura em graus'], marker='o', linestyle='--', linewidth=1, color='b', alpha=0.7,
95           label='Temperatura')
96 plt.plot(data_rolled[['Segundos']], tangent_line, linestyle='--', color='orange', label='Linha Tangente no Ponto de Inflexão')
97 plt.axvline(x=t_d, color='r', linestyle='--', label='t_d (Tempo Morto)')
98 plt.axvline(x=tau_time, color='g', linestyle='--', label='63% da Mudança Total (tau)')
99 plt.axhline(y=PV_63_percent, color='purple', linestyle='--', label='63% da Mudança Total em PV')
100 plt.scatter(max_slope_time, max_slope_temp, color='black', label='Máxima Inclinação', zorder=5)
101 plt.title('Dados coletados para utilizar no método de Ziegler-Nichols de malha aberta (Média de 10 pontos)', fontsize=15)
102 plt.xlabel('Segundos', fontsize=16)
103 plt.ylabel('Temperatura em graus', fontsize=16)
104 plt.grid(True, linestyle='--', alpha=0.5)
105 plt.legend()
106 plt.tight_layout()
107 plt.show()

Executed at 2024.06.08 21:16:19 in 429ms

```

▼ Ganho de Processo (Gp): 0.5351059085841694
 Tempo de Retardo (t_d): 6.25 segundos
 Constante de Tempo (tau): 87.4 segundos
 Controlador P: Kc = 26.13314444650918
 Controlador PI: Kc = 23.519829996458267, Ti = 20.8125
 Controlador PID: Kc = 31.35977332861102, Ti = 12.5, Td = 3.125
 Controlador PID Parallel Form: Kp = 31.35977332861102
 Controlador PID Parallel Form: Ki = 2.508781862888814
 Controlador PID Parallel Form: Kd = 97.99929165198943

Figura 11: Código utilizado para determinar os parâmetros do PID

Observa-se na *figura 9* que o sistema demorou em torno de 700s para atingir uma relativa estabilidade, e ainda apresentando algumas oscilações, o que não é o ideal. Ainda assim, com esse gráfico foi possível calcular t_d , τ , e g_p , o que nos permitiu encontrar K_c , T_i e T_d .

Como as especificações dadas pelo Professor referem-se ao modo paralelo do PID, foi necessário converter K_c , T_i e T_d (modo interativo) em K_p , K_i e K_d (modo paralelo), como é feito no código em Python (Linhas 75,76,77).

Os valores encontrados foram:

$$K_c = 31.35, \quad T_i = 12.5, \quad T_d = 3.125 \quad \text{para o modo iterativo}$$

E então convertemos, como mostrado no código, obtendo:

$$K_p = 31.35, \quad K_i = 2.50, \quad K_d = 98 \quad \text{para o modo paralelo}$$

A primeira tentativa de sintonia com o método de *Ziegler-Nichols* de malha aberta não funcionou, pois o *overshoot* foi de **14,88%** ou **5,95 °C** acima do valor desejado, um valor elevado. O erro estacionário também foi muito elevado, se estabilizando em 45 °C para um setpoint de 40 °C.

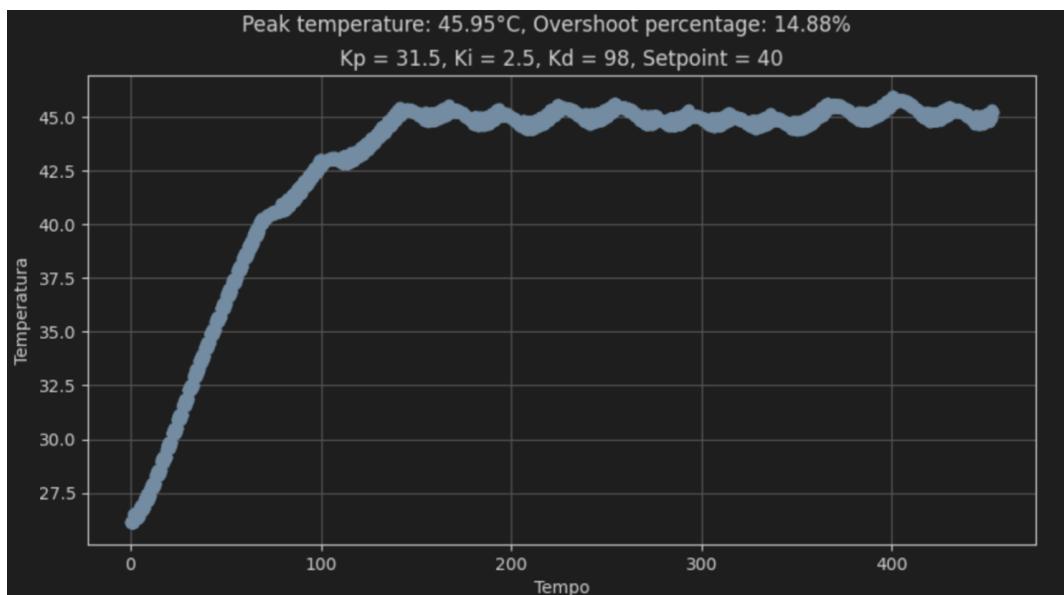


Figura 12: Pico da temperatura: 45,95 °C, Setpoint: 40°C, Overshoot: 14,88%

Os próximos valores serão obtidos por tentativa e erro, baseados nos valores obtidos pelo método de *Ziegler-Nichols* de malha aberta.

A segunda tentativa foi de aumentar o K_i para diminuir o erro estacionário, o que não surtiu muito efeito.

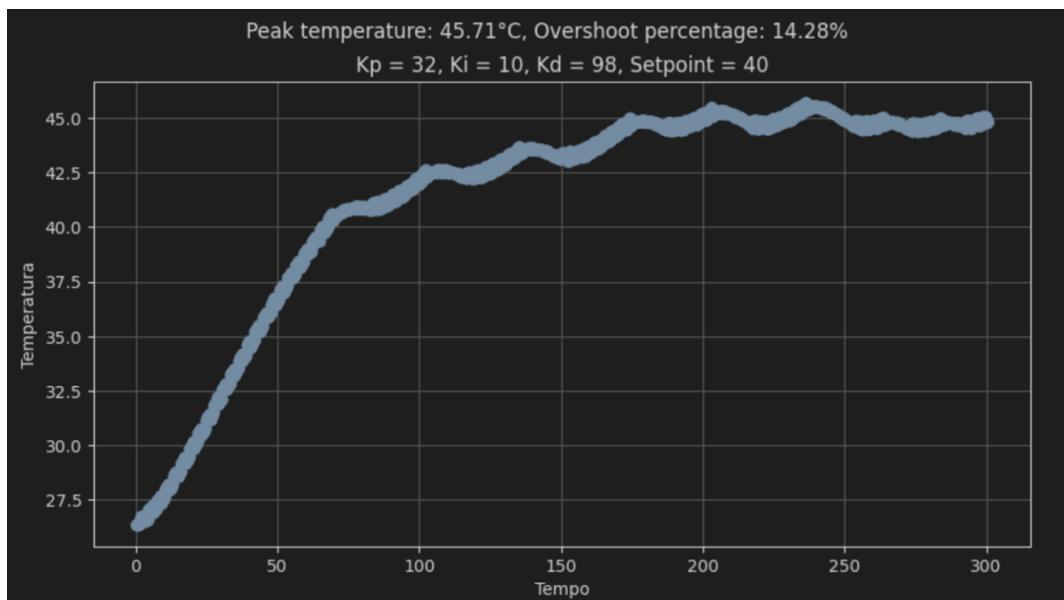


Figura 13: Pico da temperatura: 45,71 °C, Setpoint: 40°C, Overshoot: 14,28%

A terceira tentativa foi aumentar o K_d , na tentativa de diminuir o overshoot, o que fez o sistema passar muito o setpoint de 40 graus.

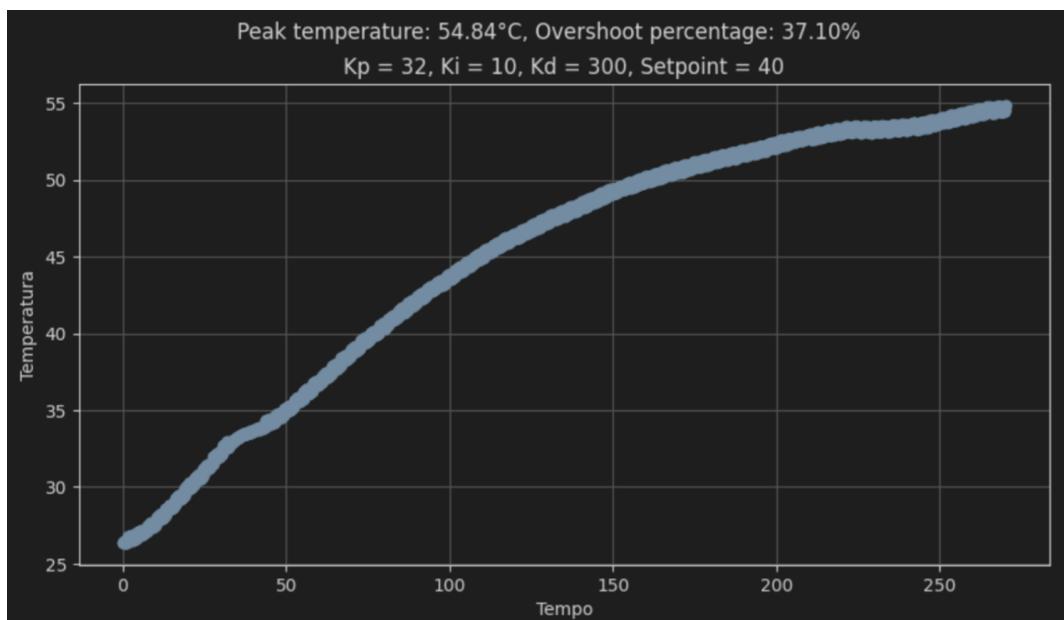


Figura 14: Pico da temperatura: 54,84 °C, Setpoint: 40°C, Overshoot: 37,10%

A quarta tentativa foi diminuir K_i e K_d em um fator de mais ou menos 10 para tentar chegar ao setpoint, o que funcionou razoavelmente bem.

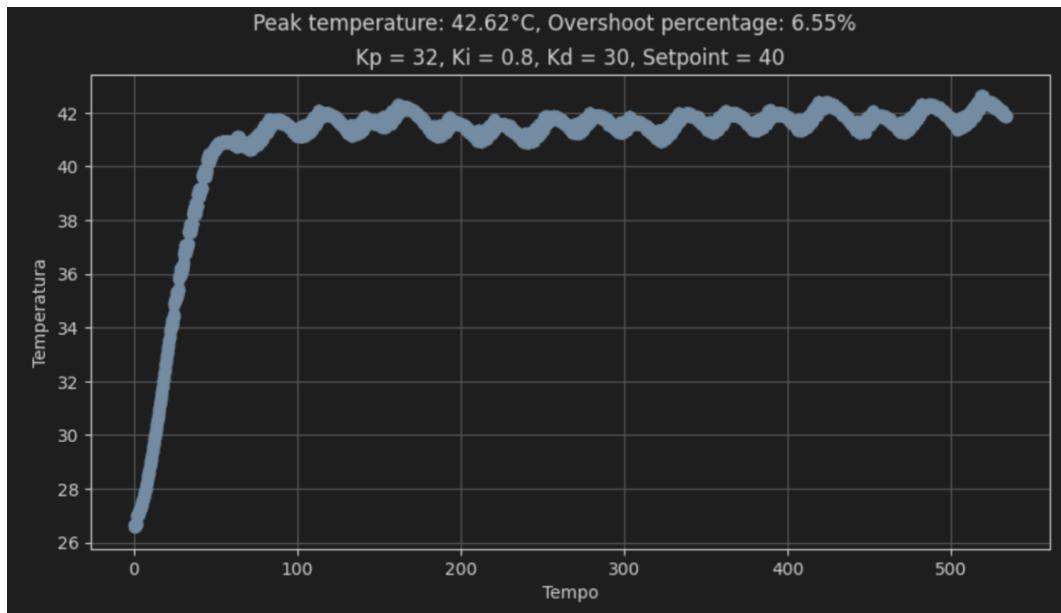


Figura 15: Pico da temperatura: 42,62 °C, Setpoint: 40°C, Overshoot: 6,55%

Como diminuir o K_i e K_d funcionou, diminuímos ainda mais o K_d para diminuir o overshoot.

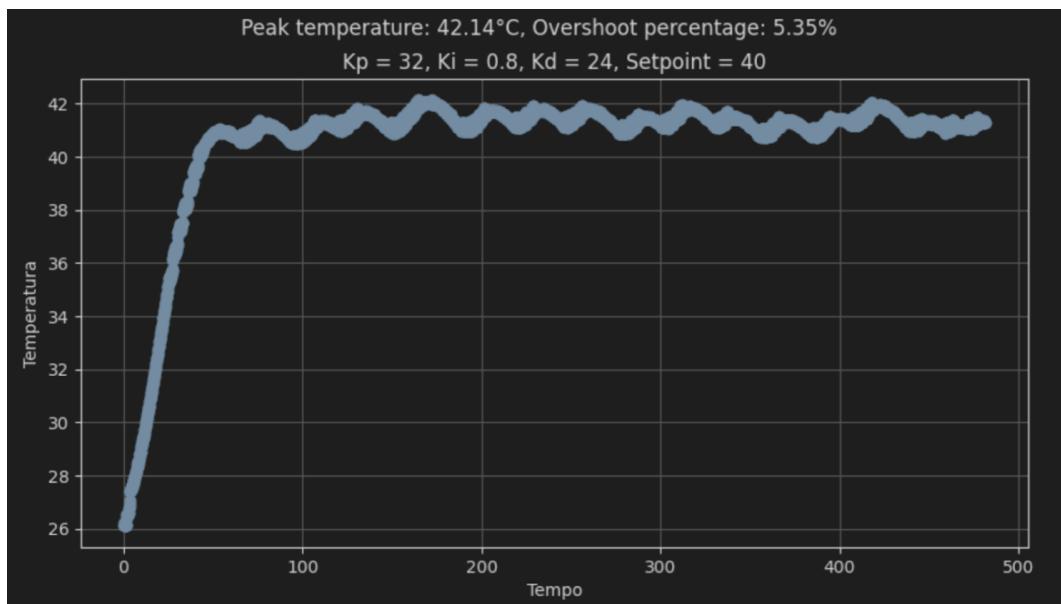


Figura 16: Pico da temperatura: 42,14 °C, Setpoint: 40°C, Overshoot: 5,35%

Novamente diminuindo o K_i e K_d porque surtiu efeito.

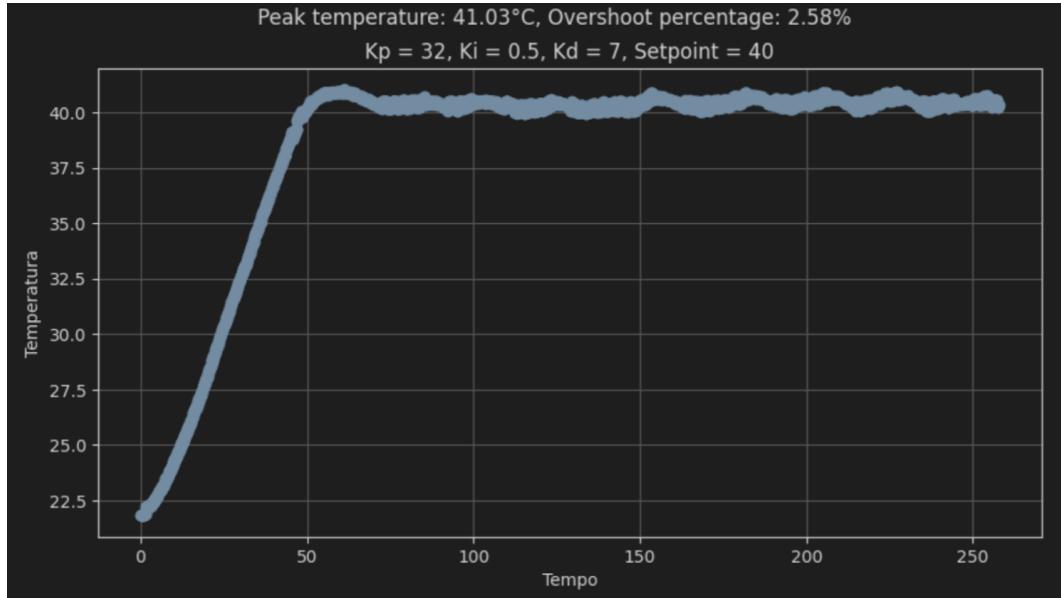


Figura 17: Pico da temperatura: 41,03 °C, Setpoint: 40°C, Overshoot: 2,58%

Como o erro estacionário já estava aceitável tentamos diminuir o K_d para diminuir o overshoot, o que não surtiu tanto efeito, apenas 0.20% de redução do overshoot.

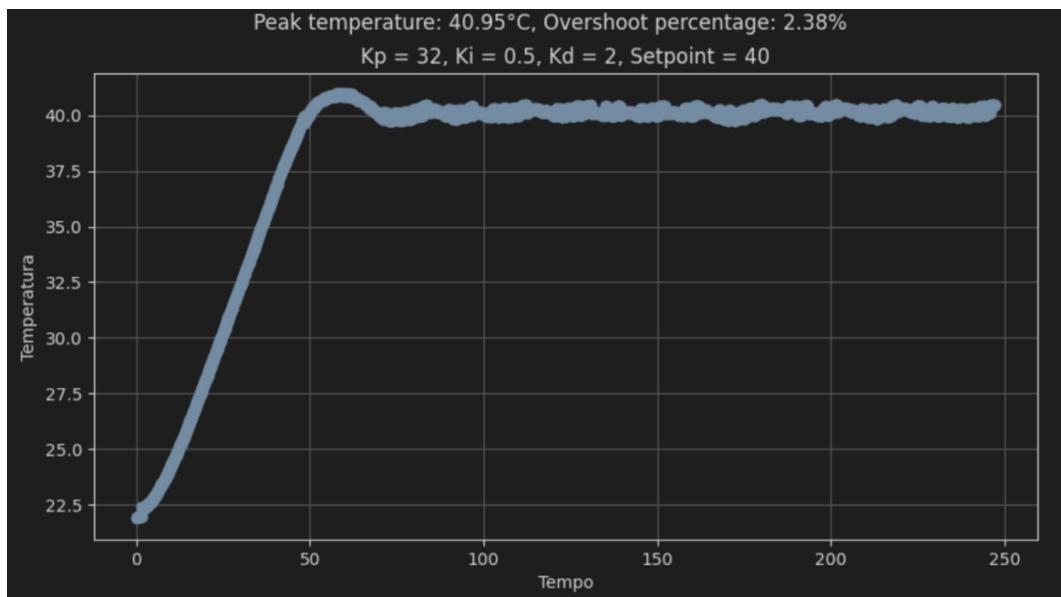


Figura 18: Pico da temperatura: 40,95 °C, Setpoint: 40°C, Overshoot: 2,38%

Como o overshoot ficou próximo do requisito do projeto, iremos testar os valores para outras faixas de temperaturas.

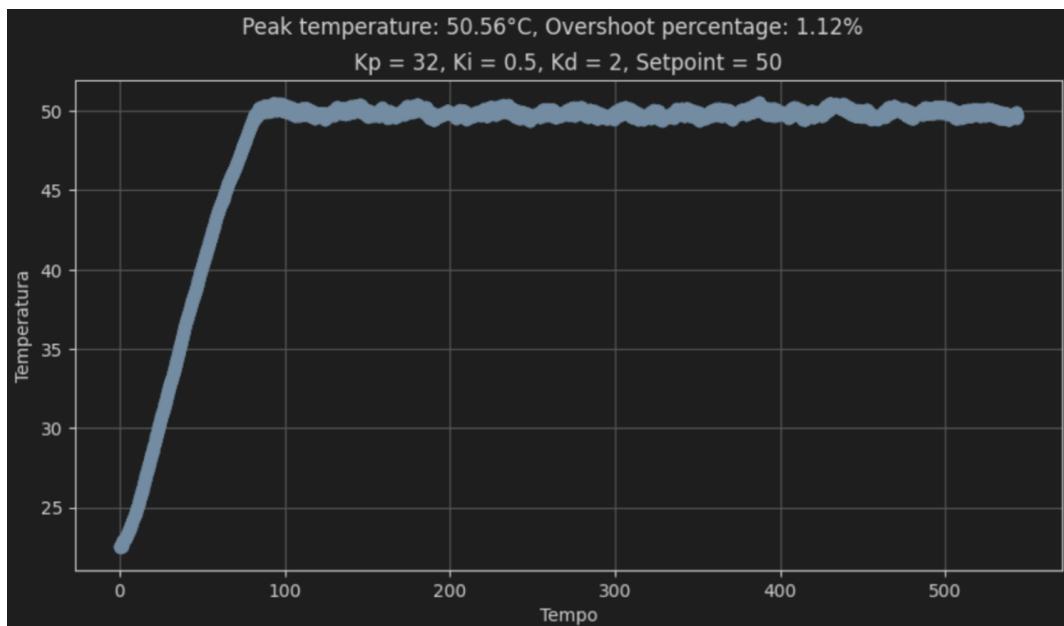


Figura 19: Pico da temperatura: 50,56 °C, Setpoint: 50°C, Overshoot: 1,12%

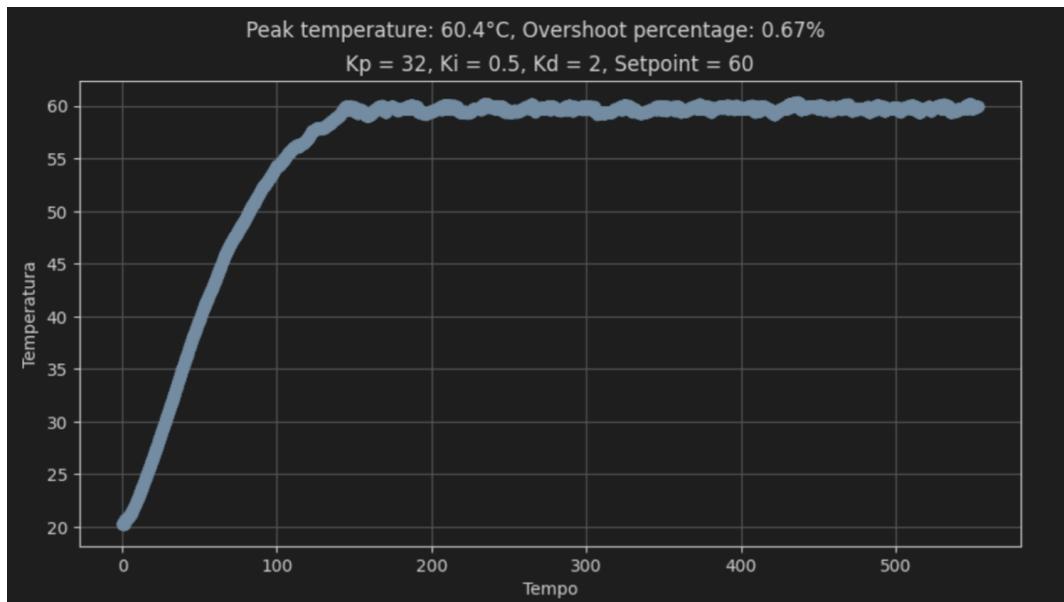


Figura 20: Pico da temperatura: 60,4 °C, Setpoint: 60°C, Overshoot: 0,67%

Testando em setpoints mais elevados, o *overshoot* foi ainda menor, chegando em 1,12% para o setpoint de 50°C e 0,67% para o setpoint de 60 °C.

Observa-se que quanto maior a temperatura do setpoint, menor o *overshoot*. Como na temperatura de 40 °C obtivemos um *overshoot* de 2,38% (0,95 °C) e conforme aumentamos o setpoint o *overshoot* diminuiu, consideramos aceitável os valores $K_p = 32$, $K_i = 0.5$, $K_d = 2$ para o nosso controlador.

1. O aquecimento de 22,5 °C para 40 °C ocorre em aproximadamente 50 segundos.
2. O aquecimento de 22,5 °C Celsius para 50 °C ocorre em aproximadamente 80 segundos.
3. O aquecimento de 22,5 °C Celsius para 60 °C ocorre em aproximadamente 130 segundos.

3.1 Uso do cooler

O cooler é utilizado no sistema de controle de temperatura para ajustar a temperatura quando há necessidade de um setpoint menor do que a temperatura atual. Nesse caso, apenas o uso do aquecedor (*heater*) não é suficiente; é necessário utilizar o cooler para reduzir a temperatura.

Para evitar que o cooler seja acionado e desligado repetidamente devido a pequenas variações na temperatura, o sistema utiliza de uma zona morta. A zona morta é uma faixa ao redor do setpoint dentro da qual o cooler não é acionado. No projeto, a zona morta é definida como 5% do setpoint, o que corresponde a $\pm 2.5^\circ\text{C}$ para um setpoint de 50°C.

Lógica de Controle

1. **Acionamento do Cooler:** O cooler é acionado quando a temperatura atual excede o setpoint mais da zona morta. Por exemplo, com um setpoint de 50°C e uma zona morta de $\pm 2.5^\circ\text{C}$, o cooler é acionado se a temperatura desejada ultrapassar 52.5°C.

2. **Desligamento do Cooler:** O cooler é desligado quando a temperatura cai abaixo do setpoint menos a zona morta. No mesmo exemplo, o cooler é desligado se a temperatura cair abaixo de 47.5°C.

3. **Controle do Aquecedor:** Se a temperatura estiver dentro da zona morta, o sistema utiliza um controle de aquecedor para ajustar a temperatura de forma mais precisa.

O uso da zona morta no controle do cooler proporciona uma operação mais estável do sistema de controle de temperatura. Essa abordagem evita o acionamento frequente do cooler devido a pequenas variações de temperatura.

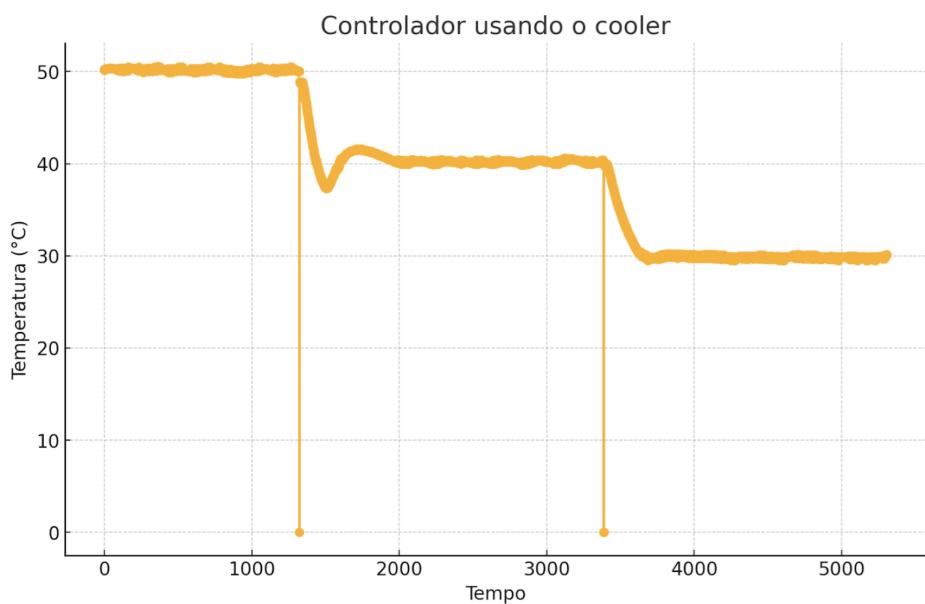


Figura 21: Controlador usando o cooler

4 Manual de utilização

Este manual fornece instruções detalhadas sobre como utilizar o sistema de controle de temperatura, que inclui uma interface local e comunicação remota via UART. O sistema permite ao usuário definir localmente a temperatura desejada e monitorá-la, monitorar a temperatura atual, a velocidade do cooler, o duty cycle do heater, o duty cycle do cooler e a velocidade do cooler. O sistema também permite ao usuário ajustar os parâmetros de controle da planta via UART.

4.1 Manutenção da Temperatura

O sistema deve ser capaz de manter a temperatura do resistor no valor determinado pelo usuário.

4.2 Interface Local

O sistema fornece uma interface local composta por push buttons e um display LCD. Através dos push buttons, o usuário pode inserir a temperatura desejada e mudar as telas. No display LCD, o sistema exibe três telas diferentes, que podem ser alternadas segurando o botão Enter por 3 segundos:

- Primeira tela:
 - Temperatura atual do sistema (°C)
 - Temperatura desejada (°C)
 - Rotação do cooler (RPM)
- Segunda tela:
 - Temperatura atual do sistema (°C)
 - Temperatura desejada (°C)
 - Duty cycle do heater
 - Duty cycle do cooler
- Terceira tela:
 - Kp
 - Ki
 - Kd

4.3 Controle Remoto via UART

O sistema permite o controle através de comandos seriais recebidos por uma interface UART. Isso possibilita a comunicação e configuração remota do sistema via um computador conectado. O usuário pode ajustar os ganhos do controlador (Kp, Ki, Kd) para responder a mudanças na planta de controle.

4.4 Alerta Sonoro

O buzzer apita quando o botão é solto, independentemente de ser apenas um clique ou de estar segurando o botão.

4.5 Controle de Temperatura com Cooler

O sistema utiliza automaticamente um cooler para controlar a temperatura do resistor. O cooler é ativado quando o setpoint é menor que a temperatura atual do resistor, como explicado em 3.1.

4.6 Indicador Visual de Temperaturas

O sistema possui um indicador visual utilizando LEDs de cores diferentes para indicar faixas de temperatura:

- Verde: 0°C a 40°C
- Amarelo: 40°C a 60°C
- Vermelho: 60°C a 90°C

As cores dos LEDs mudam de acordo com a temperatura atual do resistor, proporcionando um feedback visual imediato sobre o estado de operação do sistema.

4.7 Overshoot Controlado

O overshoot, ou seja, a quantidade pela qual a temperatura ultrapassa o valor desejado antes de se estabilizar é de 1°C para o setpoint de 40°C, e diminui conforme aumentamos a temperatura.

4.8 Faixa de Operação de Temperatura

O sistema opera entre a temperatura ambiente e 90°C. A temperatura máxima é definida pela própria estrutura do resistor.

4.9 Configuração da Interface Local

1. Ligue o sistema.
2. Verifique as informações exibidas no display LCD.
3. Pressione o push button "enter" por 3 segundos para mudar de tela.

4.10 Configuração do Terminal Serial

1. Conecte o sistema ao computador usando a porta UART.
2. Abra o terminal serial (por exemplo, PuTTY).
3. Configure o terminal com a porta COM correta e a taxa de transmissão (baud rate) de 115200.
4. Utilize os comandos 'Set' conforme necessário.

4.11 Inserindo a Temperatura Desejada

1. Utilize os push buttons para inserir a temperatura desejada. O setpoint aumenta/diminui em 1°C quando clica-se no push button up/down e aumenta/diminui em 10 quando segura-se o push button por 500ms.
2. O buzzer apita para indicar que o botão foi solto.
3. Confirme a temperatura inserida observando o display LCD.

4.12 Monitoramento do Sistema

1. Acompanhe os parâmetros no display LCD.
2. Observe as cores dos LEDs para verificar a faixa de temperatura.

4.13 Ajuste dos Parâmetros de Controle via UART

1. Conecte o sistema ao computador e abra o terminal serial.
2. Use os comandos 'Set' para ajustar os parâmetros Kp, Ki e Kd (o usuário pode trocar de tela no monitor LCD para monitorar os parâmetros de controle).

5 Manutenção do Sistema

5.1 Verificação Regular

1. Verifique regularmente os componentes do sistema para garantir que estão funcionando corretamente.
2. Limpe o cooler e o aquecedor para evitar acúmulo de poeira e garantir eficiência.

5.2 Atualização do Software

1. Mantenha o software do sistema atualizado para garantir a melhor performance e segurança.
2. Siga as instruções de atualização fornecidas pelo fabricante.

6 Problemas identificados e não resolvidos

```
if (!bCoolerActivated && fTemperature > (fSetPoint + fHysteresis * fSetPoint)) {  
    vCoolerOn(fTemperature, fSetPoint);  
}  
} else if (bCoolerActivated && fTemperature < (fSetPoint - fHysteresis * fSetPoint)) {  
    vCoolerOff(fTemperature, fSetPoint);  
    vTemperatureControl();  
}  
} else {  
    vTemperatureControl();  
}
```

Figura 22: Problema identificado: Cooler permanece ligado

1. O cooler continua funcionando mesmo quando o heater começa a funcionar, principalmente para a mudança de temperatura em poucos graus celsius, o que causa perda desnecessária de energia. No entanto, essa configuração apresentou um bom resultado de controle de temperatura. (Figura 22)
2. Alguns valores mostrados no LCD ficam piscando a cada ciclo e alguns não piscam.
3. Optamos por fazer a visualização da temperatura atual e desejada, a velocidade de rotação do cooler, o duty cycle do cooler e do heater, todos no LCD, logo, não há necessidade de exibir na UART. Esquecemos de apagar o código. (Figura 23 e 24)
4. Na UART não é possível setar os decimais do Kp, Ki, Kd com vírgula, é necessário usar ponto.

```
44 ⊕ void vCommunicationStateMachineInit(UART_HandleTypeDef *huart) {
45     HAL_UART_Receive_IT(&hlpuart1, (uint8_t*) &c, 1);
46     fActualTemp = 20.0; //t
47     fDesiredTemp = 25.0; //d
48     uiCoolerSpeed = 10; // v
49     ucDutyCycleCooler = 20; // c
50     ucDutyCycleHeather = 50; // h
51 }
```

Figura 23: Problema identificado: Não apagamos variáveis não utilizadas na máquina de estados

```
case GET:
    if ('ut' == ucByte || 'c' == ucByte || 'h' == ucByte
        || 'v' == ucByte) {
        ucParam = ucByte;
        ucUartState = PARAM;
    } else
        ucUartState = IDLE;
break;
```

Figura 24: Problema identificado: Não apagamos o fragmento de código relacionado ao GET da máquina de estados

7 Autoria dos códigos fornecidos

Não foram utilizados códigos de terceiros. Todos os outros códigos foram de autoria de um dos membros do grupo. Com exceção dos códigos gerados pelo software CubeIDE. Nota-se também que diversas funções foram fornecidas pelos slides das aulas do Prof. Dr. Rodrigo Bacurau.

Referências

- [1] STMicroelectronics, "STM32 Reference Manual,"[Online]. Available: <https://www.st.com>. [Acessado: 13 de junho de 2024].
- [2] D. G. Alciatore and M. B. Histand, *Introduction to Mechatronics and Measurement Systems*. McGraw-Hill Education, 2011.
- [3] Omega Engineering, "Temperature Measurement Handbook,"[Online]. Available: <https://www.omega.com>. [Acessado: 13 de junho de 2024].
- [4] Texas Instruments, "Precision Temperature Sensors,"[Online]. Available: <https://www.ti.com>. [Acessado: 13 de junho de 2024].
- [5] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Pearson, 2015.
- [6] R. Isermann, *Digital Control Systems*. Springer, 2006.
- [7] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Transactions of the ASME*, vol. 64, no. 11, pp. 759-768, 1942.
- [8] OptiControls, "PID Tuning Guide,"[Online]. Available: <https://blog.opticontrols.com/archives/477>. [Acessado: 13 de junho de 2024].