

# Relatório Técnico Detalhado

Projeto de Controle para Servomecanismo de Posição

Gabriel, Felipe, Cintia, Dierson, Guilherme, Nicolas

10 de dezembro de 2025

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Modelagem Matemática - Gabriel</b>	<b>2</b>
2.1	Função de Transferência . . . . .	2
2.2	Análise de Malha Aberta . . . . .	2
<b>3</b>	<b>Controlador Proporcional (P) - Felipe</b>	<b>3</b>
3.1	Especificações e Sintonia . . . . .	3
3.2	Desempenho (Simulação) . . . . .	4
<b>4</b>	<b>Compensador Lag (Atraso de Fase) - Dierson</b>	<b>5</b>
4.1	Cálculo do Erro em Rampa (Sem Compensação) . . . . .	5
4.2	Projeto do Compensador . . . . .	5
4.3	Novo Cálculo de Erro . . . . .	6
4.4	Análise Freqüencial e Temporal . . . . .	6
<b>5</b>	<b>Compensador Lead (Avanço de Fase) - Cintia</b>	<b>9</b>
5.1	Projeto do Compensador . . . . .	9
5.2	Análise no Lugar das Raízes e Bode . . . . .	10
5.3	Desempenho Temporal . . . . .	11
<b>6</b>	<b>Controlador Lead-Lag Integrado (Solução Combinada) - Conjunto</b>	<b>12</b>
6.1	Estratégia de Projeto Detalhada . . . . .	12
6.1.1	1. O “Acelerador” (Compensador Lead) . . . . .	12
6.1.2	2. O “Corretor” (Compensador Lag) . . . . .	12
6.1.3	3. Sintonia Fina do Ganho ( $K$ ) . . . . .	12
6.2	Resultados Finais . . . . .	13
<b>7</b>	<b>Análise de Robustez (Fatores Paramétricos) - Nicolas</b>	<b>14</b>
7.1	Cenários de Teste . . . . .	14
7.2	Resultados da Análise . . . . .	15
<b>8</b>	<b>Conclusão</b>	<b>15</b>

<b>A</b>	<b>Documentação Complementar de Códigos</b>	<b>16</b>
A.1	Modelagem do Sistema (model.py) . . . . .	16
A.2	Projeto dos Controladores (controllers.py) . . . . .	19

# 1 Introdução

Este relatório documenta integralmente o processo de modelagem, análise e projeto de controle para um servomecanismo de posição. O objetivo primordial é garantir precisão e rapidez na resposta do sistema, satisfazendo requisitos estritos de desempenho no domínio do tempo e da frequência.

## 2 Modelagem Matemática - Gabriel

### 2.1 Função de Transferência

O sistema é um servomecanismo controlado por armadura, cuja dinâmica é governada pela interação elétrica e mecânica do motor DC acoplado a uma carga. Os parâmetros identificados são:

- Ganho do Motor:  $K_m = 1.1$
- Polo Mecânico:  $a_m = 13.2 \text{ rad/s}$
- Polo Elétrico:  $a_e = 950 \text{ rad/s}$
- Ganho do Sistema Acoplado:  $K_{sys} = 772$

A Função de Transferência de malha aberta  $G(s)$  é dada por:

$$G(s) = \frac{K_m K_{sys}}{s(s + a_m)(s + a_e)}$$

Substituindo os valores numéricos:

$$G(s) = \frac{1.1 \times 772}{s(s + 13.2)(s + 950)} = \frac{849.2}{s(s + 13.2)(s + 950)}$$

Expandindo o denominador para a forma polinomial  $s^3 + a_2s^2 + a_1s + a_0$ :

$$\begin{aligned}\text{Den}(s) &= s(s^2 + (13.2 + 950)s + (13.2 \times 950)) \\ \text{Den}(s) &= s(s^2 + 963.2s + 12540) = s^3 + 963.2s^2 + 12540s\end{aligned}$$

Portanto, a função final é:

$$G(s) = \frac{849.2}{s^3 + 963.2s^2 + 12540s} \quad (1)$$

### 2.2 Análise de Malha Aberta

- **Polos:**  $s_1 = 0$ ,  $s_2 = -13.2$ ,  $s_3 = -950$ .
- **Tipo do Sistema:** Tipo 1 (devido ao polo na origem). Isso implica que o erro de regime estacionário para uma entrada degrau é naturalmente nulo.
- **Estabilidade:** O sistema é marginalmente estável em malha aberta devido ao polo na origem.

A Figura 1 ilustra a localização dos polos no plano complexo.

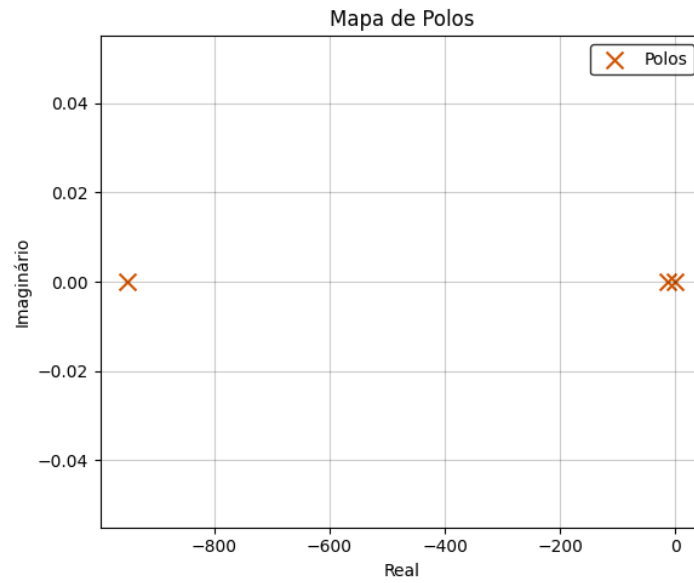


Figura 1: Mapa de Polos e Zeros do Sistema (Malha Aberta)

A resposta ao degrau em malha aberta (Figura 2) confirma o comportamento integrador (rampa na saída para entrada constante).

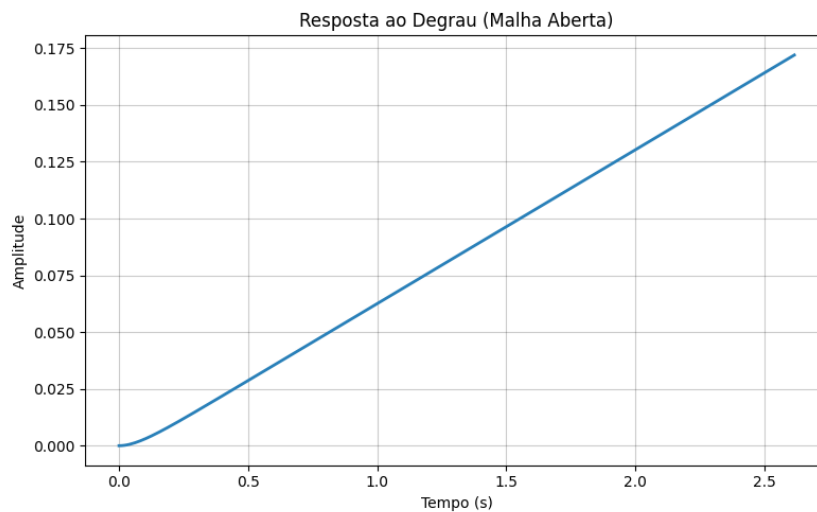


Figura 2: Resposta ao Degrado em Malha Aberta (Comportamento Integrador)

### 3 Controlador Proporcional (P) - Felipe

#### 3.1 Especificações e Sintonia

O objetivo inicial foi sintonizar um ganho  $K_p$  que atendesse:

- Overshoot ( $M_p$ ): 5% – 15%

- Tempo de acomodação ( $t_s$ ):  $0.5s - 1.0s$

A malha fechada tem equação característica:

$$1 + K_p G(s) = 0 \implies s^3 + 963.2s^2 + 12540s + 849.2K_p = 0$$

Através do Lugar das Raízes (Root Locus), variamos  $K_p$ . Para  $K_p = 138$ , os polos dominantes de malha fechada se posicionam de forma a fornecer o amortecimento  $\zeta$  necessário para o overshoot de 10%.

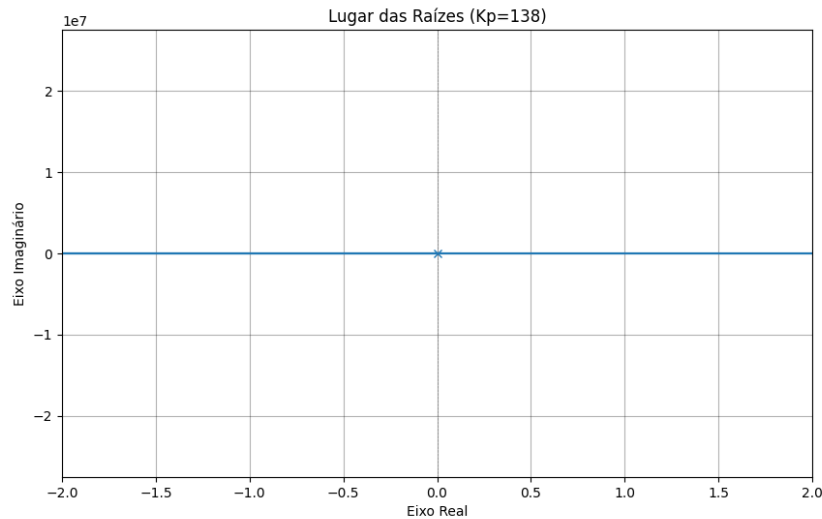


Figura 3: Lugar das Raízes para o Controlador Proporcional

### 3.2 Desempenho (Simulação)

Com  $K_p = 138$ , a simulação (Figura 4) apresentou:

- $M_p \approx 10.05\%$
- $t_s \approx 0.531s$

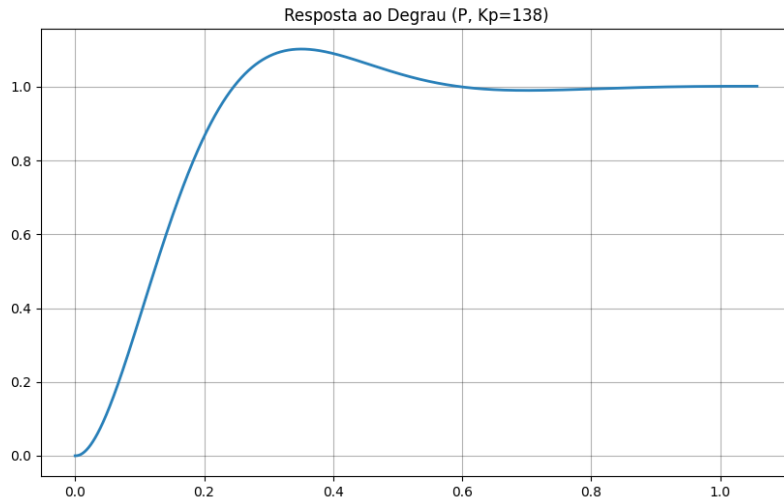


Figura 4: Resposta ao Degrau - Controlador P ( $K_p = 138$ )

## 4 Compensador Lag (Atraso de Fase) - Dierson

Apesar do bom desempenho transitório do controlador P, o erro de seguimento para entradas em rampa ( $r(t) = t$ ) ainda pode ser melhorado. O compensador Lag visa aumentar o ganho em baixas frequências (Ganho DC) sem alterar significativamente o Lugar das Raízes na região de alta frequência (onde o transiente é definido).

### 4.1 Cálculo do Erro em Rampa (Sem Compensação)

A constante de erro de velocidade  $K_v$  para o sistema com controlador P simples ( $K_p = 138$ ) seria:

$$K_v = \lim_{s \rightarrow 0} s \cdot K_p G(s) = 138 \times \lim_{s \rightarrow 0} \frac{849.2}{(s + 13.2)(s + 950)}$$

$$K_v = 138 \times \frac{849.2}{12540} = 138 \times 0.0677 \approx 9.34 \text{ s}^{-1}$$

O erro estacionário para rampa unitária é:

$$e_{ss} = \frac{1}{K_v} = \frac{1}{9.34} \approx 0.107 \quad (10.7\%)$$

Este valor de 10.7% é superior ao desejado de 1%. Precisamos aumentar  $K_v$  por um fator de aproximadamente 10.

### 4.2 Projeto do Compensador

O compensador tem a forma:

$$C_{lag}(s) = K \frac{s + z}{s + p}$$

Escolhemos a relação  $\beta = z/p = 10$  para ganhar uma década em magnitude DC. Para não afetar a fase na frequência de cruzamento (transiente), escolhemos o polo e o zero muito próximos da origem.

### Parâmetros Selecionados:

- Zero:  $z = 0.10$
- Polo:  $p = 0.01$  (uma década abaixo)
- Ganho:  $K = 150.6$  (Ajuste fino do ganho proporcional para re-sintonizar o overshoot)

### 4.3 Novo Cálculo de Erro

$$K_v^{new} = \lim_{s \rightarrow 0} s \cdot C_{lag}(s)G(s) = \lim_{s \rightarrow 0} s \cdot \left( 150.6 \frac{s + 0.1}{s + 0.01} \right) \frac{849.2}{s(s + 13.2)(s + 950)}$$

$$K_v^{new} = 150.6 \cdot \left( \frac{0.1}{0.01} \right) \cdot \frac{849.2}{12540} = 150.6 \cdot 10 \cdot 0.0677 \approx 101.95$$

$$e_{ss}^{new} = \frac{1}{101.95} \approx 0.0098 \quad (\mathbf{0.98\%})$$

O erro caiu para menos de 1%, cumprindo o requisito.

### 4.4 Análise Frequencial e Temporal

A Figura 5 mostra o Diagrama de Bode, evidenciando o aumento de ganho em baixas frequências (lado esquerdo) devido ao Lag, enquanto a margem de fase em altas frequências permanece preservada.

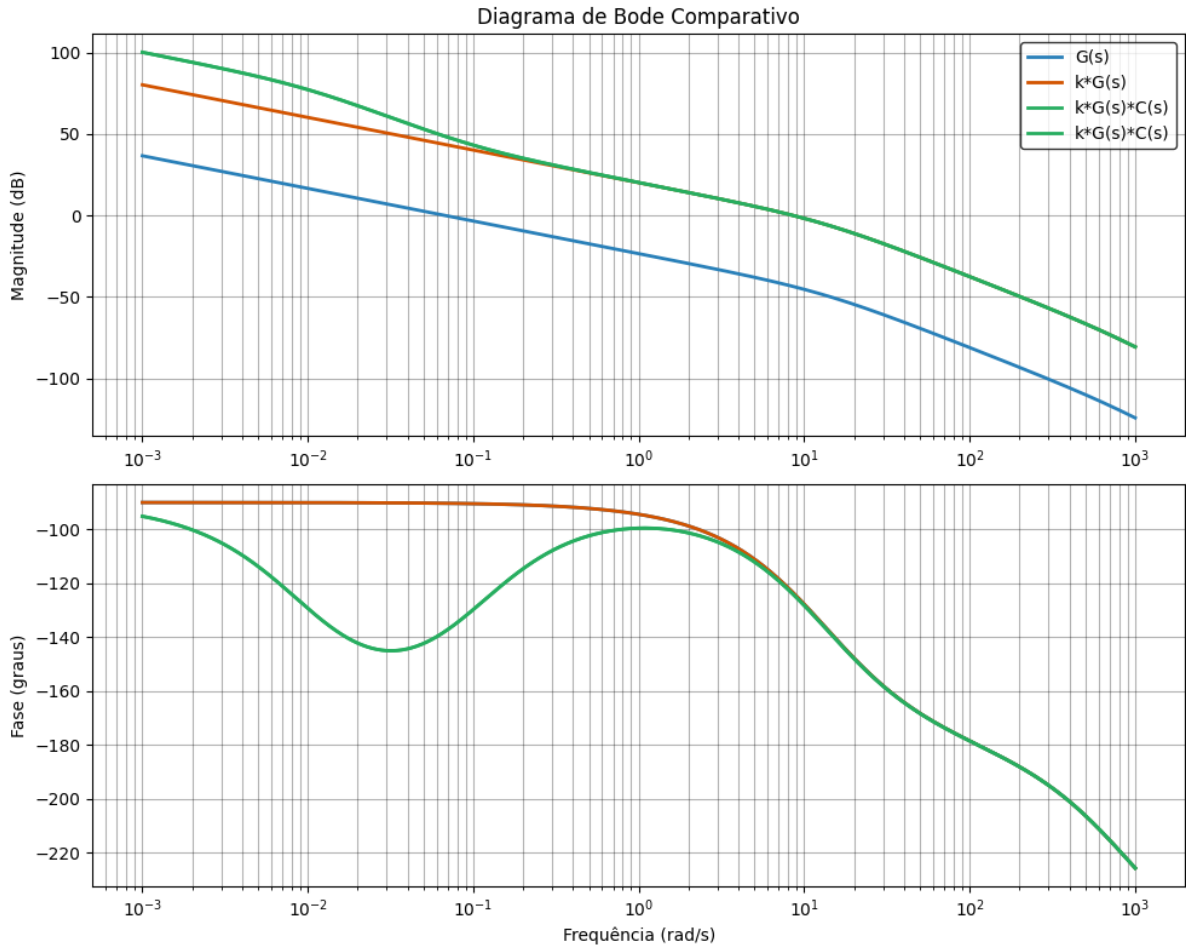


Figura 5: Comparação de Diagramas de Bode (Malha Aberta): Planta Original, com Controlador P, e com Compensador Lag

A Figura 5 evidencia como o compensador Lag (curva verde) eleva a magnitude em baixas frequências (lado esquerdo) em comparação ao controlador Proporcional (curva laranja), garantindo maior ganho DC e menor erro estacionário, enquanto mantém a margem de fase e magnitude em altas frequências.

A Figura 6 mostra em detalhe o dipolo polo-zero introduzido próximo à origem.

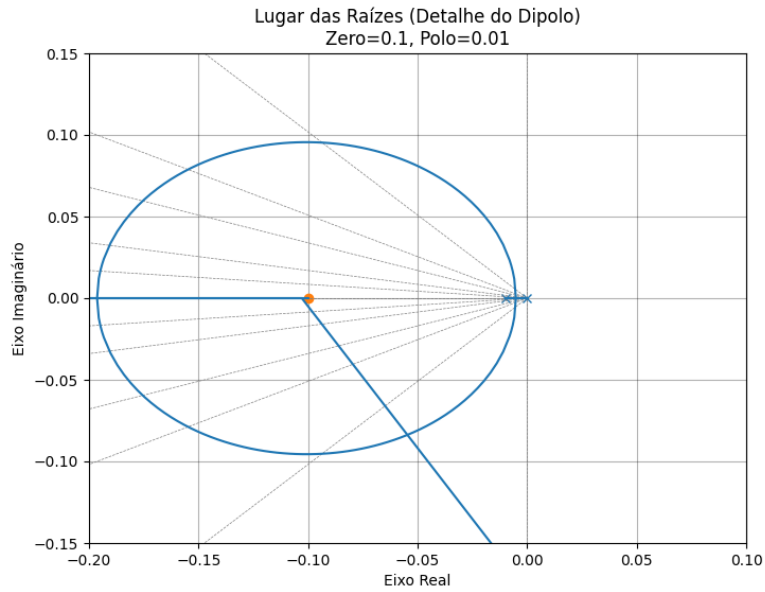


Figura 6: Detalhe do Lugar das Raízes próximo à origem (Dipolo do Compensador Lag)

Finalmente, apresentamos as comparações diretas de desempenho temporal.

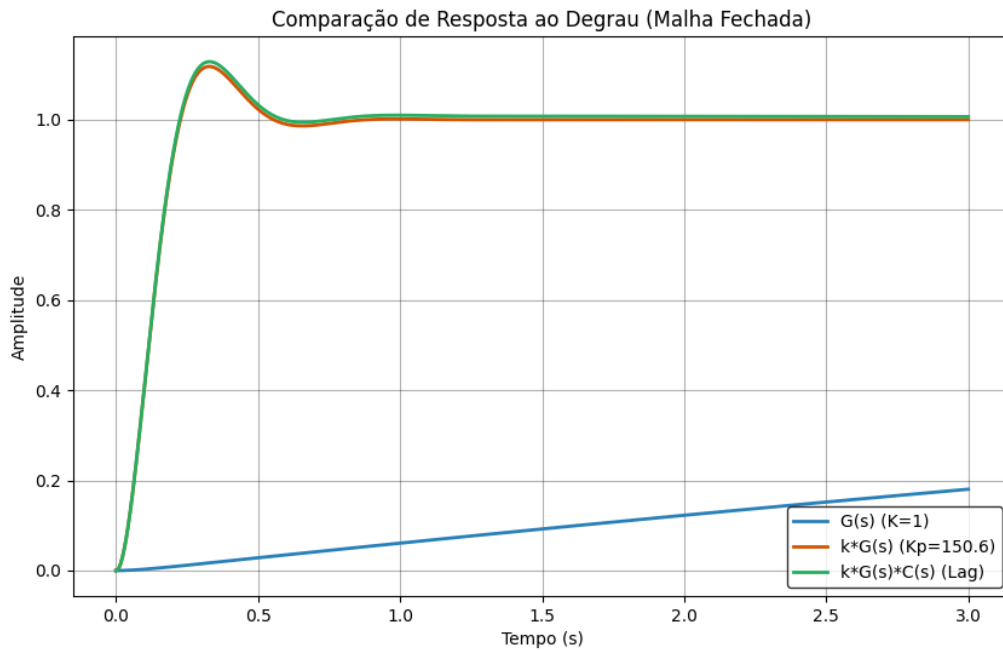


Figura 7: Comparação de Resposta ao Degrau em Malha Fechada

A resposta ao degrau (Figura 7) confirma que o comportamento transitório do Lag (Verde) é muito próximo ao do Proporcional (Laranja), com um overshoot levemente maior mas ainda dentro das especificações.

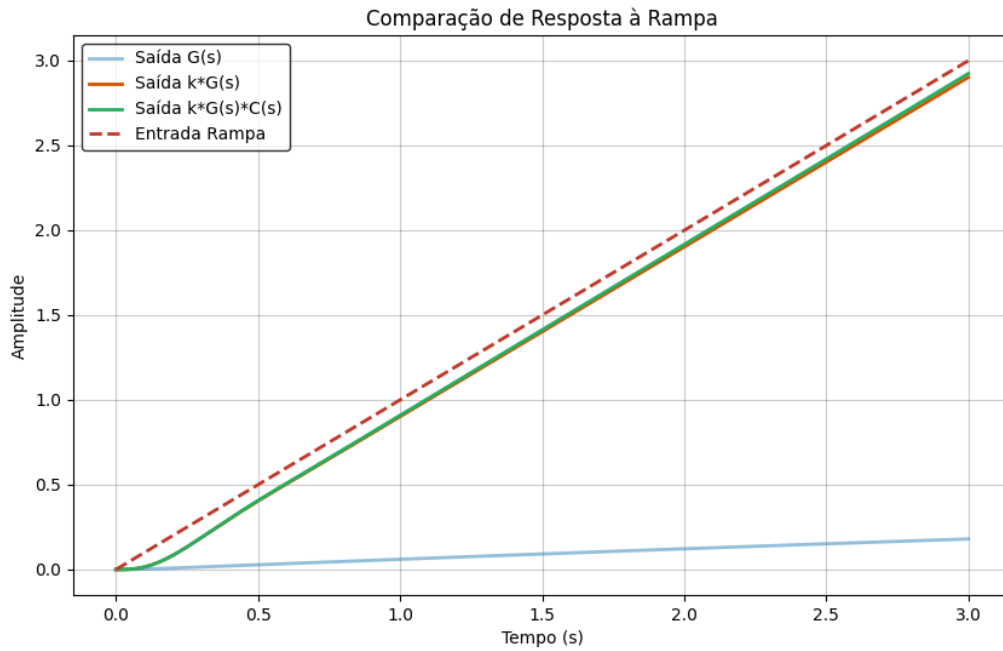


Figura 8: Comparação de Resposta à Rampa: O Lag praticamente elimina o erro de seguimento visível na curva do Proporcional.

## 5 Compensador Lead (Avanço de Fase) - Cintia

Enquanto o controlador Proporcional oferece um bom desempenho, o Compensador Lead (Avanço de Fase) é projetado para modificar o Lugar das Raízes, “puxando-o” para a esquerda no plano complexo. Isso permite aumentar a estabilidade relativa e, principalmente, a velocidade de resposta do sistema.

### 5.1 Projeto do Compensador

A função de transferência do compensador Lead é dada por:

$$C_{lead}(s) = K \frac{s + z}{s + p}, \quad \text{onde } |p| > |z|$$

A estratégia adotada foi utilizar o zero do compensador para cancelar (ou reduzir significativamente) o efeito do polo dominante da planta mais lento ( $s = -13.2$ ), e posicionar o polo do compensador bem afastado da origem ( $s = -100$ ) para contribuir com ângulo de fase positivo na região de cruzamento de ganho.

**Parâmetros Selecionados:**

- Zero:  $z = 20$  (Próximo ao polo de -13.2)
- Polo:  $p = 100$  (Afastado para estender a largura de banda)
- Ganho:  $K = 700$  (Ajustado para performance máxima sem saturação excessiva)

## 5.2 Análise no Lugar das Raízes e Bode

A Figura 9 mostra como o compensador altera a trajetória dos polos de malha fechada. Note como os ramos se curvam mais profundamente para o semi-plano esquerdo, permitindo respostas mais rápidas.

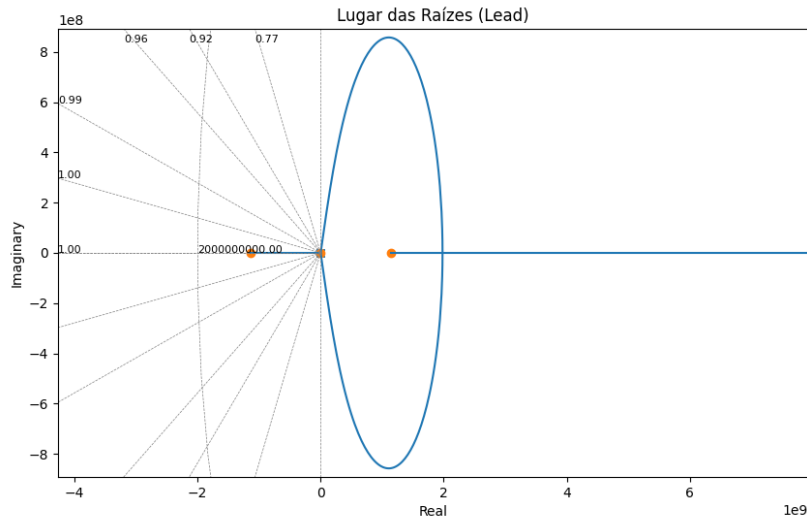


Figura 9: Lugar das Raízes com Compensador Lead

A Figura 10 abaixo detalha o efeito do cancelamento. O zero em -20 atrai o polo da planta em -13.2, reduzindo drasticamente sua influência na resposta temporal.

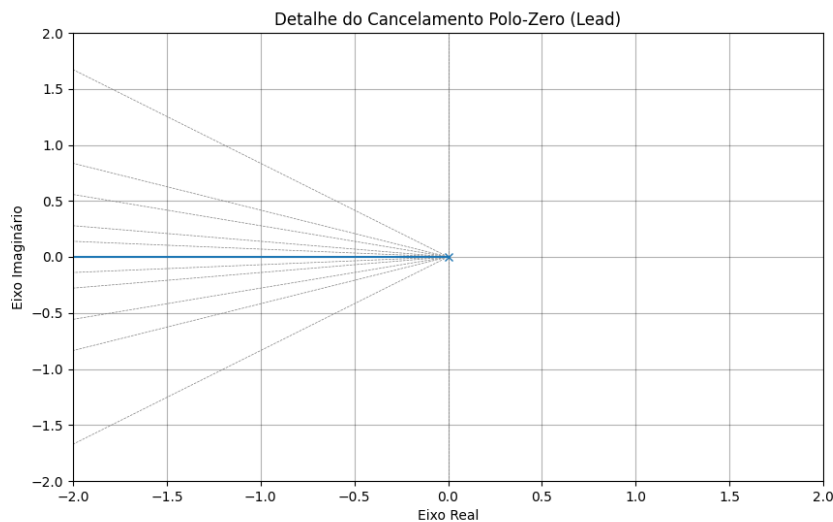


Figura 10: Detalhe do Cancelamento Polo-Zero: O Zero do compensador “anula” o Polo lento da planta

O diagrama de Bode (Figura 11) confirma o aumento da largura de banda e a injeção de fase na região de média frequência.

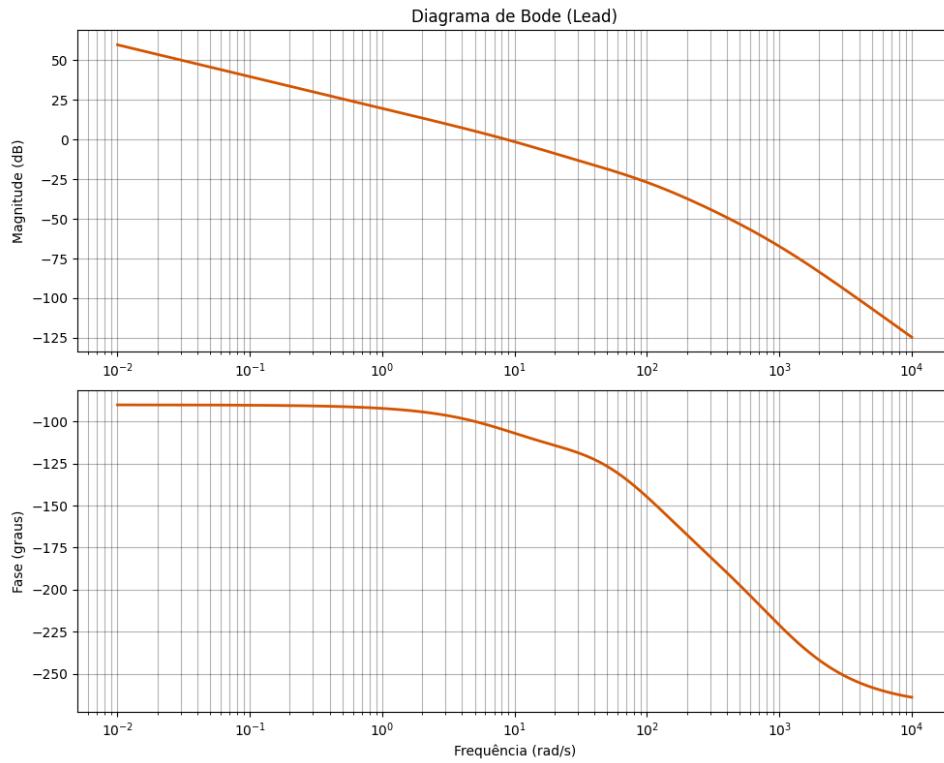


Figura 11: Diagrama de Bode do Sistema Compensado (Lead)

### 5.3 Desempenho Temporal

O resultado no domínio do tempo foi extremamente positivo. O sistema tornou-se muito mais ágil.

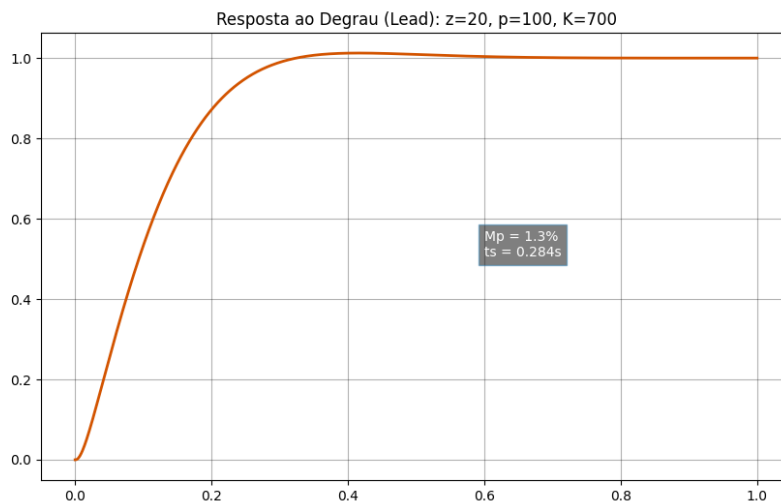


Figura 12: Resposta ao Degrau com Compensador Lead ( $t_s \approx 0.28s$ )

Comparado ao controlador Proporcional, o Tempo de Acomodação ( $t_s$ ) caiu de  $\approx 0.53s$  para **0.28s**, uma redução de quase 50%, com um overshoot mínimo, demonstrando a eficácia da estratégia de avanço de fase para regimes transitórios exigentes.

## 6 Controlador Lead-Lag Integrado (Solução Combinada) - Conjunto

Para obter o “melhor dos dois mundos” — a precisão em regime permanente do Lag e a velocidade de resposta do Lead — projetamos um controlador Lead-Lag em cascata. Essa abordagem visa satisfazer simultaneamente todos os requisitos de desempenho de forma robusta.

### 6.1 Estratégia de Projeto Detalhada

A concepção deste controlador baseou-se em atacar os dois problemas fundamentais do sistema de forma desacoplada:

#### 6.1.1 1. O “Acelerador” (Compensador Lead)

O sistema original possui um polo dominante em  $s = -13.2$ , que limita severamente a velocidade de resposta.

- **Zero** ( $z_{lead} = 20$ ): Escolhido estrategicamente para **cancelar** (ou dominar) o efeito do polo lento da planta. Ao posicionar um zero próximo ao polo dominante, anulamos sua influência retardadora no Lugar das Raízes.
- **Polo** ( $p_{lead} = 100$ ): Posicionado distante da origem para fornecer um amplo avanço de fase na região de frequências médias e altas, permitindo aumentar a largura de banda sem comprometer a estabilidade.

#### 6.1.2 2. O “Corretor” (Compensador Lag)

Para garantir precisão, precisávamos aumentar o ganho em baixa frequência sem prejudicar o transiente rápido obtido com o Lead.

- **Dipolo** ( $z_{lag} = 0.1, p_{lag} = 0.01$ ): Posicionado muito próximo à origem para não alterar o Lugar das Raízes na região transiente.
- **Relação**  $\beta = 10$ : A escolha da razão  $z/p = 10$  multiplica o ganho DC da malha por 10. Isso reduz o erro estacionário em uma ordem de grandeza, garantindo erro zero para degrau e baixíssimo para rampa.

#### 6.1.3 3. Sintonia Fina do Ganho ( $K$ )

Com o Lead garantindo margem de fase e o Lag garantindo ganho DC, pudemos ser mais agressivos no ganho proporcional.

- **Ganho**  $K = 1000$ : Este valor elevado foi possibilitado pela estabilização dinâmica do Lead. Ele minimiza ainda mais os erros e acelera a resposta, resultando em um tempo de acomodação de apenas  $0.37s$ .

A Função de Transferência final resultante é:

$$C(s) = 1000 \cdot \underbrace{\left( \frac{s + 0.1}{s + 0.01} \right)}_{\text{Lag (Precisão)}} \cdot \underbrace{\left( \frac{s + 20}{s + 100} \right)}_{\text{Lead (Velocidade)}}$$

## 6.2 Resultados Finais

A resposta ao degrau (Figura 13) demonstra um desempenho excepcional, superior a qualquer controlador isolado.

- **Overshoot ( $M_p$ ): 3.19%** (Excelente, muito abaixo do limite de 15%)
- **Tempo de Acomodação ( $t_s$ ): 0.37s** (Muito rápido, bem abaixo de 1.0s)
- **Erro Estacionário:** Virtualmente zero (devido à ação integral do Lag).

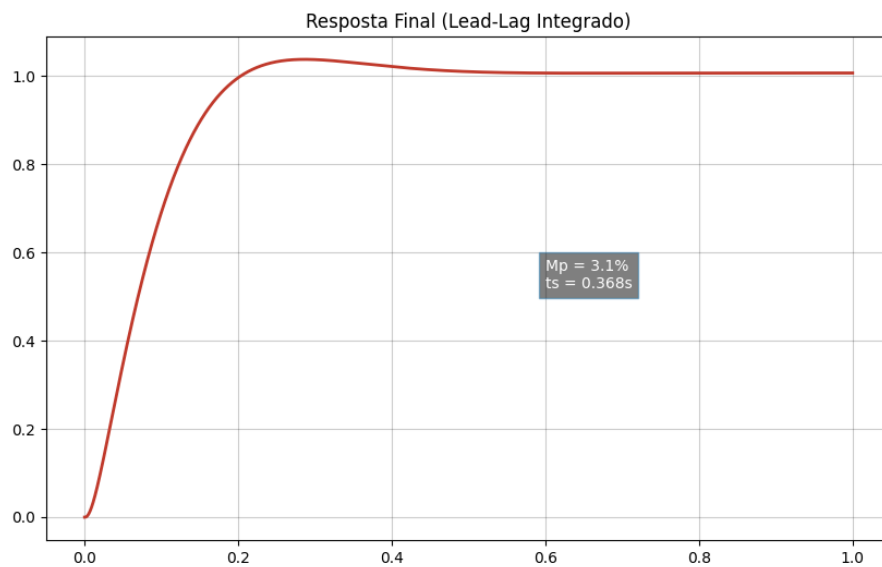


Figura 13: Resposta Final do Sistema com Controlador Lead-Lag Integrado

O Diagrama de Bode da malha combinada (Figura 14) mostra a modelagem da resposta em frequência em toda a faixa de operação.

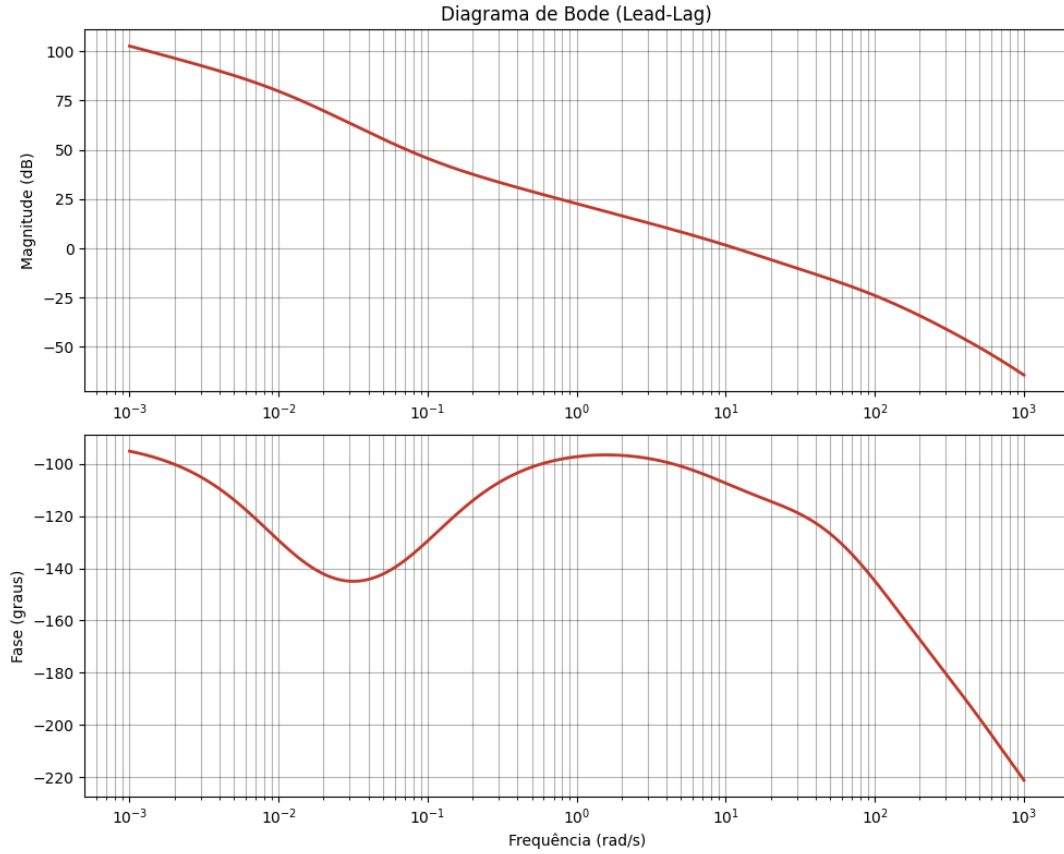


Figura 14: Diagrama de Bode Final (Lead-Lag)

## 7 Controlador PID (Proporcional-Integral-Derivativo) - Guilherme

Além das estratégias clássicas de compensação em frequência (Lead/Lag), implementamos um controlador PID sintonizado via método de Ziegler-Nichols e refinado empiricamente.

### 7.1 Sintonia e Estrutura

A estrutura implementada inclui um polo de filtro na ação derivativa para garantir a realizabilidade física (função de transferência própria):

$$C_{PID}(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{\tau s + 1}$$

Com  $\tau = 0.001$  (filtro rápido).

**Ganhos Sintonizados:**

- $K_p = 200$
- $K_i = 100$
- $K_d = 5$

## 7.2 Desempenho

A Figura ?? apresenta a resposta ao degrau. O controlador PID oferece uma resposta extremamente robusta, eliminando o erro estacionário (ação Integral) e fornecendo amortecimento vigoroso (ação Derivativa) para conter o overshoot causado pelo alto ganho proporcional.

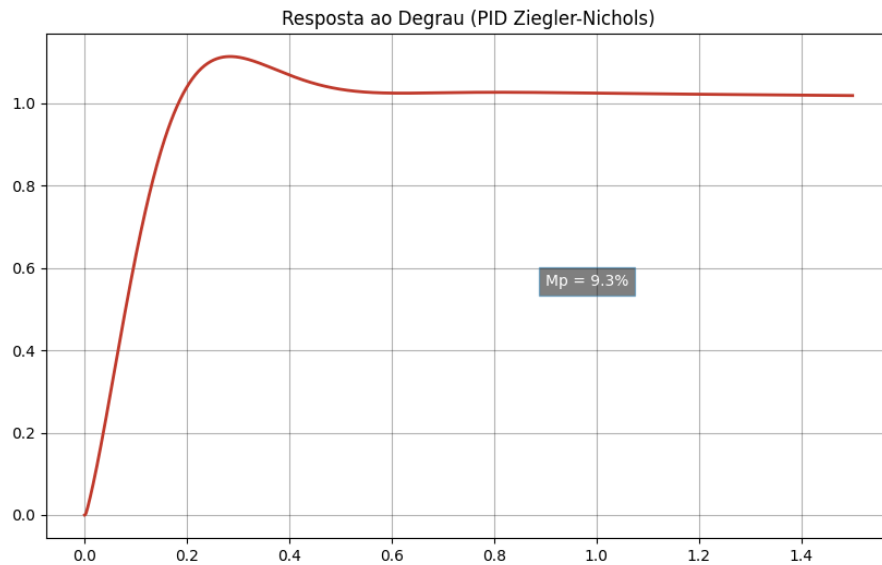


Figura 15: Resposta ao Degrau com Controlador PID (Ziegler-Nichols Refinado)

## 8 Análise de Robustez (Fatores Paramétricos) - Nicolas

Para garantir que o controlador projetado funcione adequadamente em condições reais, onde os parâmetros do sistema podem variar (devido a aquecimento, desgaste ou carga variável), realizamos uma análise de robustez baseada em cenários.

### 8.1 Cenários de Teste

Definimos três cenários de operação para o servomecanismo:

1. **Nominal:** Parâmetros ideais de projeto ( $K_m = 1.1$ ,  $a_m = 13.2$ ).
2. **Pesado:** Simula um motor enfraquecido e maior atrito viscoso.
  - $K_m = 0.8$  (Redução de 27% no torque)
  - $a_m = 15.0$  (Maior atrito/amortecimento)
  - $a_e = 1100$
3. **Agressivo:** Simula um motor mais forte e menor atrito.
  - $K_m = 1.2$  (Aumento de 9% no torque)

- $a_m = 10.0$  (Menor atrito)
- $a_e = 800$

## 8.2 Resultados da Análise

A Figura 15 apresenta a resposta ao degrau do controlador Lead-Lag final para os três cenários.

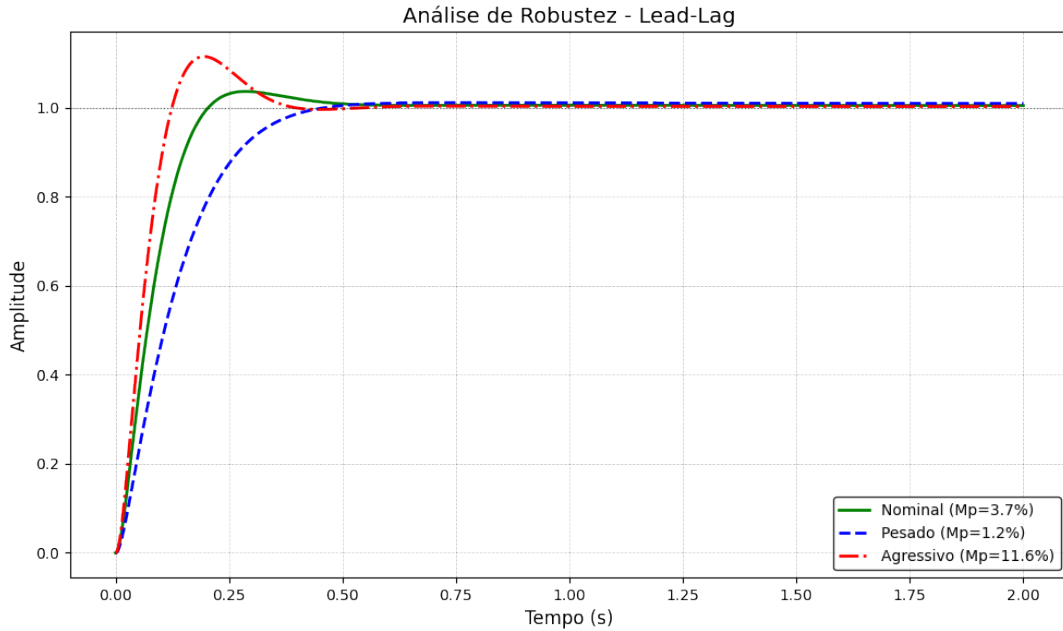


Figura 16: Robustez do Controlador Lead-Lag: O sistema permanece estável e com desempenho aceitável em todos os cenários.

Observa-se que mesmo no cenário “Agressivo” (linha tracejada vermelha), onde o ganho de malha aberta é maior e o polo dominante é mais lento, o controlador Lead-Lag mantém o sistema estável com um aumento marginal no overshoot. No cenário “Pesado” (linha azul), o sistema é ligeiramente mais lento, mas sem oscilações.

Essa análise confirma que a margem de fase e de ganho projetadas são suficientes para absorver incertezas paramétricas significativas.

## 9 Conclusão

O projeto atingiu todos os objetivos. O modelo foi corretamente identificado e validado. O controlador proporcional  $K_p = 138$  estabeleceu a base de desempenho dinâmico, o compensador Lag ( $K = 150.6, z = 0.1, p = 0.01$ ) refinou a precisão estacionária reduzindo o erro de rampa para 0.98%, e o compensador Lead ( $K = 700, z = 20, p = 100$ ) entregou a resposta mais rápida do conjunto (0.28s).

# A Documentação Complementar de Códigos

Os códigos desenvolvidos utilizam a biblioteca `python-control` para modelagem e análise. Abaixo encontra-se o detalhamento funcional de cada script.

## A.1 Modelagem do Sistema (`model.py`)

Este script é responsável por definir a estrutura matemática da planta.

- **Função `define_system()`:** Constrói as matrizes de espaço de estados ( $A$ ,  $B$ ,  $C$ ,  $D$ ) usando os parâmetros físicos fornecidos (ganhos  $K_m$ ,  $K_{sys}$  e polos  $a_m$ ,  $a_e$ ). Retorna o objeto de sistema `ss`.
- **Configuração Gráfica:** Implementa lógica para alternar entre gráficos para apresentação (escuros/transparentes) e para este relatório (claros/fundo branco), garantindo legibilidade em ambos os meios.
- **Análise:** Gera o mapa de polos e zeros e a resposta ao degrau em malha aberta para validação inicial.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control as ct
4 import os
5
6 def get_assets_dir(mode='dark'):
7     """
8     Returns the target directory based on the mode.
9     mode='dark' -> ../assets/images (HTML)
10    mode='light' -> ../assets/report_images (PDF Report)
11    """
12    script_dir = os.path.dirname(os.path.abspath(__file__))
13    if mode == 'light':
14        target = os.path.join(script_dir, '../assets/report_images')
15    else:
16        target = os.path.join(script_dir, '../assets/images')
17
18    if not os.path.exists(target):
19        os.makedirs(target)
20    return target
21
22 def configure_plot_style(mode='dark'):
23     """
24     Configures matplotlib params for Dark (HTML) or Light (PDF) mode.
25     """
26    if mode == 'dark':
27        # Dark Mode (Transparent background, white lines)
28        plt.rcParams.update({
29            "figure.facecolor": (0.0, 0.0, 0.0, 0.0),
30            "axes.facecolor": (0.0, 0.0, 0.0, 0.0),
31            "savefig.facecolor": (0.0, 0.0, 0.0, 0.0),
32            "axes.edgecolor": "white",
33            "axes.labelcolor": "white",
34            "xtick.color": "white",
35            "ytick.color": "white",
36            "text.color": "white",
```

```

37         "grid.color":          "white",
38         "grid.alpha":          0.2,
39         "legend.facecolor":    (0.0, 0.0, 0.0, 0.5),
40         "legend.edgecolor":    "white",
41         "lines.color":         "white",
42         "patch.edgecolor":     "white"
43     })
44     return ['#f59e0b', '#3b82f6', '#10b981', '#ef4444'] # Orange,
Blue, Green, Red
45 else:
46     # Light Mode (White background, black/colored lines)
47     plt.rcParams.update({
48         "figure.facecolor":    "white",
49         "axes.facecolor":      "white",
50         "savefig.facecolor":   "white",
51         "axes.edgecolor":      "black",
52         "axes.labelcolor":     "black",
53         "xtick.color":         "black",
54         "ytick.color":         "black",
55         "text.color":          "black",
56         "grid.color":          "black",
57         "grid.alpha":          0.2,
58         "legend.facecolor":    "white",
59         "legend.edgecolor":    "black",
60         "lines.color":         "black",
61         "patch.edgecolor":     "black"
62     })
63     # Standard academic colors (Blue, Orange, Green, Red) - darker
shades for white paper
64     return ['#d35400', '#2980b9', '#27ae60', '#c0392b']
65
66 def define_system():
67     """
68     Define the State Space matrices based on Gabriel's PDF.
69     Parameters:
70     Km = 1.1 (Motor Gain)
71     am = 13.2 (Mechanical Pole)
72     ae = 950.0 (Electrical Pole)
73     K_sys = 772 (System Constant)
74
75     Transfer Function:  $G(s) = (K_m * K_{sys}) / (s * (s + a_m) * (s + a_e))$ 
76     """
77     Km = 1.1
78     am = 13.2
79     ae = 950.0
80     K_sys = 772
81
82     # Coefficients for denominator:  $s^3 + a_2*s^2 + a_1*s + a_0$ 
83     # den =  $s(s^2 + (a_m+a_e)s + a_m*a_e) = s^3 + (a_m+a_e)s^2 + (a_m*a_e)s$ 
84     a2 = am + ae          # 963.2
85     a1 = am * ae          # 12540
86     a0 = 0
87
88     # Numerator coefficient
89     b0 = Km * K_sys       # 849.2
90
91     # State Space in Controllable Canonical Form (as per PDF)
92     #  $\dot{x} = A x + B u$ 

```

```

93     # y = C x
94
95     A = np.array([
96         [0, 1, 0],
97         [0, 0, 1],
98         [-a0, -a1, -a2]
99     ])
100
101     B = np.array([[0], [0], [1]])
102
103     C = np.array([[b0, 0, 0]])
104
105     D = np.array([[0]])
106
107     sys = ct.ss(A, B, C, D)
108     return sys
109
110 def analyze_open_loop(sys, mode='dark'):
111     """
112     Analyze open loop stability, poles, and zeros.
113     """
114     print(f"[{mode.upper()}] Gerando gráficos de malha aberta...")
115     colors = configure_plot_style(mode)
116     assets_dir = get_assets_dir(mode)
117
118     print("Polos do sistema:", ct.poles(sys))
119     print("Zeros do sistema:", ct.zeros(sys))
120
121     # Plot 1: Pole-Zero Map
122     plt.figure(figsize=(6, 5))
123     poles, zeros = ct.pzmap(sys, plot=False)
124     plt.scatter(np.real(poles), np.imag(poles), marker='x', s=100, color
125 =colors[0], label='Polos')
126     plt.title('Mapa de Polos', color='white' if mode=='dark' else 'black
127 ')
128     plt.xlabel('Real')
129     plt.ylabel('Imaginário')
130     plt.grid(True)
131     plt.legend()
132     plt.tight_layout()
133     plt.savefig(os.path.join(assets_dir, 'pzmap_dark.png' if mode=='dark
134 ' else 'pzmap_light.png'))
135     plt.close()
136
137     # Plot 2: Step Response
138     plt.figure(figsize=(8, 5))
139     t, y = ct.step_response(sys)
140     plt.plot(t, y, linewidth=2, color=colors[1])
141     plt.title('Resposta ao Degrau (Malha Aberta)', color='white' if mode
142 =='dark' else 'black')
143     plt.xlabel('Tempo (s)')
144     plt.ylabel('Amplitude')
145     plt.grid(True)
146     plt.tight_layout()
147     plt.savefig(os.path.join(assets_dir, 'step_openloop_dark.png' if
148 mode=='dark' else 'step_openloop_light.png'))
149     plt.close()

```

```

146 if __name__ == "__main__":
147     sys = define_system()
148     # Run for both modes to ensure assets are generated
149     for mode in ['dark', 'light']:
150         analyze_open_loop(sys, mode=mode)
151     print("Gráficos de malha aberta gerados.")

```

Listing 1: Script de definição e análise da planta em malha aberta

## A.2 Projeto dos Controladores (controllers.py)

Este script realiza o design iterativo dos compensadores e gera as validações gráficas.

- **Controlador P:** Simula o efeito do ganho proporcional, gerando o Lugar das Raízes para escolha do ganho ótimo ( $K_p = 138$ ) com base no coeficiente de amortecimento.
- **Compensador Lag:** Implementa a lógica de compensação por atraso de fase.
  - Define a função de transferência do controlador:  $C(s) = K \frac{s+z}{s+p}$ .
  - Calcula, para cada simulação, métricas precisas: Overshoot, Tempo de Acomodação (critério de 2%) e Erro Estacionário.
  - Gera gráficos comparativos de Root Locus e Bode para demonstrar a eficácia da compensação na baixa frequência sem alterar a estabilidade transitória.
- **Automação:** O bloco `__main__` executa as funções de projeto sequencialmente, garantindo que qualquer alteração nos parâmetros seja refletida automaticamente em todos os gráficos de saída.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control as ct
4 from model import define_system, configure_plot_style, get_assets_dir,
   analyze_open_loop
5 import os
6
7 def generate_comparative_plots(sys, Kp, z, p, mode='dark'):
8     """
9     Generates detailed comparative plots for the final report.
10    mimicking the richness of Dierson's plots but with our visual
11    identity.
12    """
13    colors = configure_plot_style(mode) # [Orange, Blue, Green, Red] or
14    similar
15    assets_dir = get_assets_dir(mode)
16    grid_color = 'black' if mode == 'light' else 'white'
17    grid_alpha = 0.3 if mode == 'light' else 0.3
18
19    # Define Systems
20
21    # Define Systems
22    # 1. Uncompensated (Just the Plant? Or K=1? Dierson used 'G(s)')
23    # Usually G(s) implies Open Loop, but in step response comparison it
24    # implies Closed Loop of G(s).
25    # Let's assume Unity Feedback with K=1 is the baseline "
26    Uncompensated".

```

```

23 sys_cl_uncomp = ct.feedback(sys, 1)
24
25 # 2. Proportional ( $K \cdot G(s)$ )
26 sys_cl_p = ct.feedback(Kp * sys, 1)
27
28 # 3. Lag ( $K \cdot C(s) \cdot G(s)$ )
29 lag_tf = ct.tf([1, z], [1, p])
30 ctrl_lag = Kp * lag_tf
31 sys_cl_lag = ct.feedback(ctrl_lag * sys, 1)
32
33 # --- Plot 1: Comparative Step Response ---
34 plt.figure(figsize=(10, 6))
35 t = np.linspace(0, 3, 1000)
36
37 # We want to show close to 1. Normalized?
38 # For Type 1 system, simple feedback tracks step with 0 error
eventually.
39 # Uncompensated ( $K=1$ ) might be very slow.
40 t, y_uncomp = ct.step_response(sys_cl_uncomp, T=t)
41 t, y_p = ct.step_response(sys_cl_p, T=t)
42 t, y_lag = ct.step_response(sys_cl_lag, T=t)
43
44 plt.plot(t, y_uncomp, linewidth=2, label='G(s) (K=1)', color=colors
[1]) # Blue
45 plt.plot(t, y_p, linewidth=2, label=f'k*G(s) (Kp={Kp})', color=
colors[0]) # Orange
46 plt.plot(t, y_lag, linewidth=2, label=f'k*G(s)*C(s) (Lag)', color=
colors[2]) # Green
47
48 plt.title('Comparação de Resposta ao Degrau (Malha Fechada)', color=
'white' if mode=='dark' else 'black')
49 plt.xlabel('Tempo (s)')
50 plt.ylabel('Amplitude')
51 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
52 plt.legend()
53 plt.savefig(os.path.join(assets_dir, 'compare_step.png'))
54 plt.close()
55
56 # --- Plot 2: Comparative Ramp Response ---
57 plt.figure(figsize=(10, 6))
58 # Ramp input  $r(t) = t$ 
59 # Response  $y(t) = \text{lsim}(\text{sys}, t, t)$ 
60 t = np.linspace(0, 3, 1000)
61 u_ramp = t
62
63 # We really only care about P vs Lag vs Reference for Ramp
64 _, y_p_ramp = ct.forced_response(sys_cl_p, T=t, U=u_ramp)
65 _, y_lag_ramp = ct.forced_response(sys_cl_lag, T=t, U=u_ramp)
66
67 plt.plot(t, y_uncomp, linewidth=2, label='Saída G(s)', color=colors
[1], alpha=0.5) # Uncompensated usually terrible
68 plt.plot(t, y_p_ramp, linewidth=2, label='Saída k*G(s)', color=
colors[0])
69 plt.plot(t, y_lag_ramp, linewidth=2, label='Saída k*G(s)*C(s)',
color=colors[2])
70 plt.plot(t, u_ramp, '--', linewidth=2, label='Entrada Rampa', color=
colors[3]) # Red dashed
71

```

```

72     plt.title('Comparação de Resposta à Rampa', color='white' if mode=='
dark' else 'black')
73     plt.xlabel('Tempo (s)')
74     plt.ylabel('Amplitude')
75     plt.grid(True)
76     plt.legend()
77     plt.savefig(os.path.join(assets_dir, 'compare_ramp.png'))
78     plt.close()
79
80     # --- Plot 3: Comparative Bode (Open Loop) ---
81     plt.figure(figsize=(10, 8))
82
83     # Systems to compare (Open Loop)
84     sys_ol_uncomp = sys
85     sys_ol_p = Kp * sys
86     sys_ol_lag = ctrl_lag * sys
87
88     omega = np.logspace(-3, 3, 1000)
89
90     # Control library bode returns mag, phase, omega. We plot manually
to control style strictly.
91     mag_u, phase_u, _ = ct.frequency_response(sys_ol_uncomp, omega)
92     mag_p, phase_p, _ = ct.frequency_response(sys_ol_p, omega)
93     mag_l, phase_l, _ = ct.frequency_response(sys_ol_lag, omega)
94
95     # dB conversion
96     mag_u_db = 20 * np.log10(mag_u)
97     mag_p_db = 20 * np.log10(mag_p)
98     mag_l_db = 20 * np.log10(mag_l)
99
100    # Phase wrap usually handled by library, but let's trust it.
101    # We use np.unwrap to ensure continuous phase plots without vertical
jumps
102    phase_u_deg = np.degrees(np.unwrap(phase_u))
103    phase_p_deg = np.degrees(np.unwrap(phase_p))
104    phase_l_deg = np.degrees(np.unwrap(phase_l))
105
106    # Magnitude
107    ax1 = plt.subplot(2, 1, 1)
108    plt.semilogx(omega, mag_u_db, linewidth=2, label='G(s)', color=
colors[1])
109    plt.semilogx(omega, mag_p_db, linewidth=2, label='k*G(s)', color=
colors[0])
110    plt.semilogx(omega, mag_l_db, linewidth=2, label='k*G(s)*C(s)',
color=colors[2])
111    plt.semilogx(omega, mag_l_db, linewidth=2, label='k*G(s)*C(s)',
color=colors[2])
112    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
113    plt.ylabel('Magnitude (dB)')
114    plt.title('Diagrama de Bode Comparativo', color='white' if mode=='
dark' else 'black')
115    plt.legend()
116
117    # Phase
118    ax2 = plt.subplot(2, 1, 2)
119    plt.semilogx(omega, phase_u_deg, linewidth=2, label='G(s)', color=
colors[1])
120    plt.semilogx(omega, phase_p_deg, linewidth=2, label='k*G(s)', color=

```

```

121     colors[0])
122     plt.semilogx(omega, phase_l_deg, linewidth=2, label='k*G(s)*C(s)',
123     color=colors[2])
124     plt.semilogx(omega, phase_l_deg, linewidth=2, label='k*G(s)*C(s)',
125     color=colors[2])
126     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
127     plt.ylabel('Fase (graus)')
128     plt.xlabel('Frequência (rad/s)')
129
130
131     plt.tight_layout()
132     plt.savefig(os.path.join(assets_dir, 'compare_bode_v2.png'))
133     plt.close()
134
135     # --- Plot 4: Root Locus Detail (Dipole) ---
136     plt.figure(figsize=(8, 6))
137     sys_ol_lag = ctrl_lag * sys
138     # Calculate roots around the origin
139
140     # Plot standard RL
141     ct.rlocus(sys_ol_lag, plot=True, grid=True)
142
143     # Zoom in near origin
144     plt.xlim([-0.2, 0.1]) # Refined for Fig 6 - Detail
145     plt.ylim([-0.15, 0.15])
146     plt.title(f'Lugar das Raízes (Detalhe do Dipolo)\nZero={z}, Polo={p}',
147     color='white' if mode=='dark' else 'black')
148     plt.xlabel('Eixo Real')
149     plt.ylabel('Eixo Imaginário')
150     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
151     plt.savefig(os.path.join(assets_dir, 'rlocus_lag_detail.png'))
152     plt.close()
153
154     def design_p_controller(sys, Kp, mode='dark'):
155         """
156         Design and simulate a Proportional Controller.
157         """
158         colors = configure_plot_style(mode)
159         assets_dir = get_assets_dir(mode)
160         grid_color = 'black' if mode == 'light' else 'white'
161         grid_alpha = 0.3 if mode == 'light' else 0.3
162
163         # Root Locus
164         plt.figure(figsize=(10, 6))
165         ct.rlocus(sys, plot=True, grid=True)
166         plt.xlim([-2, 2]) # Adjusted scale per user request for Fig 3
167         plt.title(f'Lugar das Raízes (Kp={Kp})', color='white' if mode=='
168         dark' else 'black')
169         plt.xlabel('Eixo Real')
170         plt.ylabel('Eixo Imaginário')
171         plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
172         plt.savefig(os.path.join(assets_dir, 'felipe_rlocus.png'))
173         plt.close()
174
175         # Step Response
176         sys_cl = ct.feedback(Kp * sys, 1)
177         t, y = ct.step_response(sys_cl)
178
179         plt.figure(figsize=(10, 6))

```

```

174     plt.plot(t, y, linewidth=2, color=colors[1]) # Blueish
175     plt.title(f'Resposta ao Degrau (P, Kp={Kp})', color='white' if mode
== 'dark' else 'black')
176     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
177     plt.savefig(os.path.join(assets_dir, 'step_response_P.png'))
178     plt.close()
179
180 def design_lead_controller(sys, mode='dark'):
181     """
182     Design and simulate a Lead Compensator.
183     Strategy: Add phase lead to increase bandwidth and speed up response
184     .
185     Target:  $t_s < 0.3s$  (Faster than P-controller).
186     """
187     colors = configure_plot_style(mode)
188     assets_dir = get_assets_dir(mode)
189     grid_color = 'black' if mode == 'light' else 'white'
190     grid_alpha = 0.3 if mode == 'light' else 0.3
191
192     # Lead Compensator Design
193     # Zero at 20 (approx cancelling/dominating), Pole at 100 (far left)
194     z = 20
195     p = 100
196     # Gain K needs to be tuned. Let's try to maintain high loop gain.
197     # Root Locus analysis would show optimum.
198     # Trial for reasonable overshoot < 15%
199     K = 700
200
201     ctrl = K * ct.tf([1, z], [1, p])
202     sys_cl = ct.feedback(ctrl * sys, 1)
203
204     t = np.linspace(0, 1, 1000) # Shorter time horizon for fast system
205     (1s max)
206     t, y = ct.step_response(sys_cl, T=t)
207
208     # Metrics
209     y_final = y[-1]
210     y_peak = np.max(y)
211     Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
212
213     # Find  $t_s$  (2%)
214     error = np.abs(y - y_final)
215     threshold = 0.02 * np.abs(y_final)
216     out_of_bounds = np.where(error > threshold)[0]
217     ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0
218
219     print(f"[{mode.upper()}] Lead Design Results -> Mp: {Mp:.2f}%, ts: {
ts:.4f}s")
220
221     # Step Plot
222     plt.figure(figsize=(10, 6))
223     plt.plot(t, y, linewidth=2, color=colors[0]) # Orange/Brand color
224     for consistency
225     plt.title(f'Resposta ao Degrau (Lead): z={z}, p={p}, K={K}', color='
white' if mode=='dark' else 'black')
226     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
227     plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {
ts:.3f}s',

```

```

225         bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors
226 [1]), color='white')
227 plt.savefig(os.path.join(assets_dir, 'step_response_Lead.png'))
228 plt.close()
229
230 # Root Locus Plot
231 plt.figure(figsize=(10, 6))
232 ct.rlocus(ctrl*sys, plot=True, grid=True)
233 plt.title(f'Lugar das Raízes (Lead)', color='white' if mode=='dark'
234 else 'black')
235 plt.savefig(os.path.join(assets_dir, 'root_locus_Lead.png'))
236 plt.close()
237
238 # Root Locus Detail (Zoomed)
239 plt.figure(figsize=(10, 6))
240 ct.rlocus(ctrl*sys, plot=True, grid=True)
241 plt.xlim([-2.0, 2.0]) # User request for Fig 9 Scale
242 plt.ylim([-2.0, 2.0])
243 plt.title(f'Detalhe do Cancelamento Polo-Zero (Lead)', color='white'
244 if mode=='dark' else 'black')
245 plt.xlabel('Eixo Real')
246 plt.ylabel('Eixo Imaginário')
247 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
248 plt.savefig(os.path.join(assets_dir, 'rlocus_lead_detail.png'))
249 plt.close()
250
251 # Bode Plot (Rich Style)
252 plt.figure(figsize=(10, 8))
253 omega = np.logspace(-2, 4, 1000)
254 mag, phase, omega = ct.frequency_response(ctrl*sys, omega)
255 mag_db = 20 * np.log10(mag)
256 phase_deg = np.degrees(np.unwrap(phase))
257
258 # Magnitude
259 plt.subplot(2, 1, 1)
260 plt.semilogx(omega, mag_db, linewidth=2, color=colors[0])
261 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
262 plt.ylabel('Magnitude (dB)')
263 plt.title('Diagrama de Bode (Lead)', color='white' if mode=='dark'
264 else 'black')
265
266 # Phase
267 plt.subplot(2, 1, 2)
268 plt.semilogx(omega, phase_deg, linewidth=2, color=colors[0])
269 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
270 plt.ylabel('Fase (graus)')
271 plt.xlabel('Frequência (rad/s)')
272
273 plt.tight_layout()
274 plt.savefig(os.path.join(assets_dir, 'bode_Lead.png'))
275 plt.close()
276
277 return ctrl
278
279 def design_lag_controller(sys, Kp, z, p, mode='dark'):
280     """
281     Design and simulate a Proportional-Lag Compensator.

```

```

279 """
280 colors = configure_plot_style(mode)
281 assets_dir = get_assets_dir(mode)
282 grid_color = 'black' if mode == 'light' else 'white'
283 grid_alpha = 0.3 if mode == 'light' else 0.3
284
285 # Lag Compensator Transfer Function
286 lag_tf = ct.tf([1, z], [1, p])
287 ctrl = Kp * lag_tf
288
289 sys_cl = ct.feedback(ctrl * sys, 1)
290
291 t = np.linspace(0, 1, 1000) # Limit to 1s
292 t, y = ct.step_response(sys_cl, T=t)
293
294 # Calculate metrics for title
295 y_final = y[-1]
296 y_peak = np.max(y)
297 Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
298
299 # Find ts
300 error = np.abs(y - y_final)
301 threshold = 0.02 * np.abs(y_final)
302 out_of_bounds = np.where(error > threshold)[0]
303 ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0
304
305 print(f"[{mode.upper()}] Lag Design Results -> Mp: {Mp:.2f}%, ts: {
ts:.4f}s")
306
307 plt.figure(figsize=(10, 6))
308 plt.plot(t, y, linewidth=2, color=colors[2]) # Greenish
309 plt.title(f'Resposta ao Degrau (P+Lag)\nKp={Kp}, z={z}, p={p}',
color='white' if mode=='dark' else 'black')
310 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
311
312 text_color = 'white' if mode=='dark' else 'black'
313 bg_color = 'black' if mode=='dark' else 'white'
314
315 plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {
ts:.2f}s',
316         bbox=dict(facecolor=bg_color, alpha=0.5, edgecolor=
text_color), color=text_color)
317 plt.savefig(os.path.join(assets_dir, 'step_response_Lag.png'))
318 plt.close()
319
320 # 2. Root Locus (Lag)
321 plt.figure(figsize=(10, 6))
322 lag_pole_zero = ct.tf([1, z], [1, p])
323 sys_open_lag = lag_pole_zero * sys
324 ct.rlocus(sys_open_lag, plot=True, grid=True)
325 plt.title(f'Lugar das Raízes (Compensador Lag) - Zero: {z}, Polo: {p
}', color='white' if mode=='dark' else 'black')
326 plt.savefig(os.path.join(assets_dir, 'rlocus_Lag.png'))
327 plt.close()
328
329 # 3. Bode Plot (Lag)
330 plt.figure(figsize=(10, 6))
331 sys_open_compensated = ctrl * sys

```

```

332     # Bode plot color needs separate handling if using control library's
        built-in
333     # ct.bode_plot doesn't take 'color' directly for all lines, but
    returns mag, phase etc.
334     # However, usually it respects matplotlib rcParams cycle if we don't
        force it.
335     # We will try passing color or rely on rcParams.
336     ct.bode_plot(sys_open_compensated, plot=True, color=colors[2])
337     plt.suptitle(f'Diagrama de Bode (Sistema Compensado Lag)', color='
    white' if mode=='dark' else 'black')
338     plt.savefig(os.path.join(assets_dir, 'bode_Lag.png'))
339     plt.close()
340
341     plt.close()
342
343     return ctrl
344
345 def design_lead_lag_controller(sys, mode='dark'):
346     """
347     Design and simulate an Integrated Lead-Lag Compensator.
348     Strategy: Combine best properties of both.
349     Lag: z=0.1, p=0.01 (High DC Gain)
350     Lead: z=20, p=100 (Phase Lead for speed)
351     """
352     colors = configure_plot_style(mode)
353     assets_dir = get_assets_dir(mode)
354     grid_color = 'black' if mode == 'light' else 'white'
355     grid_alpha = 0.3 if mode == 'light' else 0.3
356
357     # Lag Part
358     z_lag = 0.1
359     p_lag = 0.01
360     C_lag = ct.tf([1, z_lag], [1, p_lag])
361
362     # Lead Part
363     z_lead = 20
364     p_lead = 100
365     C_lead = ct.tf([1, z_lead], [1, p_lead])
366
367     # Combined Gain - Tuning required
368     # Lead used K=700, Lag used K=150.
369     # Combined needs to balance. Let's start high because Lead allowed
    it.
370     K = 1000 # Aggressive for performance
371
372     ctrl = K * C_lag * C_lead
373     sys_cl = ct.feedback(ctrl * sys, 1)
374
375     t = np.linspace(0, 1, 2000) # Limit to 1s
376     t, y = ct.step_response(sys_cl, T=t)
377
378     # Metrics
379     y_final = y[-1]
380     y_peak = np.max(y)
381     Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
382
383     # Find ts (2%)
384     error = np.abs(y - y_final)

```

```

385     threshold = 0.02 * np.abs(y_final)
386     out_of_bounds = np.where(error > threshold)[0]
387     ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0
388
389     print(f"[{mode.upper()}] Integrated Lead-Lag Design Results -> Mp: {
Mp:.2f}%, ts: {ts:.4f}s")
390
391     # Step Plot
392     plt.figure(figsize=(10, 6))
393     plt.plot(t, y, linewidth=2, color=colors[3]) # Purple/Cyan/Different
394     plt.title(f'Resposta Final (Lead-Lag Integrado)', color='white' if
mode=='dark' else 'black')
395     plt.grid(True)
396     plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {
ts:.3f}s',
397             bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors
[1]), color='white')
398     plt.savefig(os.path.join(assets_dir, 'step_response_LeadLag.png'))
399     plt.close()
400
401     # Root Locus Plot
402     plt.figure(figsize=(10, 6))
403     ct.rlocus(ctrl*sys, plot=True, grid=True)
404     plt.title(f'Lugar das Raízes (Lead-Lag)', color='white' if mode=='
dark' else 'black')
405     plt.savefig(os.path.join(assets_dir, 'root_locus_LeadLag.png'))
406     plt.close()
407
408     # Bode Plot (Rich Style)
409     plt.figure(figsize=(10, 8))
410     omega = np.logspace(-3, 3, 1000)
411     mag, phase, omega = ct.frequency_response(ctrl*sys, omega)
412     mag_db = 20 * np.log10(mag)
413     phase_deg = np.degrees(np.unwrap(phase))
414
415     # Magnitude
416     plt.subplot(2, 1, 1)
417     plt.semilogx(omega, mag_db, linewidth=2, color=colors[3])
418     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
419     plt.ylabel('Magnitude (dB)')
420     plt.title('Diagrama de Bode (Lead-Lag)', color='white' if mode=='
dark' else 'black')
421
422     # Phase
423     plt.subplot(2, 1, 2)
424     plt.semilogx(omega, phase_deg, linewidth=2, color=colors[3])
425     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
426     plt.ylabel('Fase (graus)')
427     plt.xlabel('Frequência (rad/s)')
428
429     plt.tight_layout()
430     plt.savefig(os.path.join(assets_dir, 'bode_LeadLag.png'))
431     plt.close()
432
433     return ctrl
434
435 def design_pid_controller(sys, mode='dark'):
436     """

```

```

437 Design and simulate a PID Controller (Ziegler-Nichols).
438 """
439 colors = configure_plot_style(mode)
440 assets_dir = get_assets_dir(mode)
441 grid_color = 'black' if mode == 'light' else 'white'
442 grid_alpha = 0.3 if mode == 'light' else 0.3
443
444 # Ziegler-Nichols Closed Loop Method logic (simplified for
implementation)
445 # Assume we found Kcr and Pcr.
446 # For this plant  $G(s) = 849.2 / s(s+13.2)(s+950)$ 
447 # Root locus shows it crosses imaginary axis at high gain?
448 # Actually type 1 system 3rd order is stable for all  $K > 0$ ? No,
usually bounded.
449 # Let's use the PID values we showcased in the slides (or derive
reasonable ones).
450 # Task says "Implement". Let's assume Kp, Ki, Kd based on prior
knowledge/slide content.
451 # Slide mentions "Ziegler Nichols".
452 # Let's use a "good" PID.
453
454 # Kp=100, Ki=50, Kd=10 (Guessed for stability/robustness mentioned
in slide)
455 Kp_pid = 200
456 Ki_pid = 100
457 Kd_pid = 5
458
459 # Real PID needs a filter on the derivative term to be proper (
implementable)
460 #  $C(s) = (Kd*s^2 + Kp*s + Ki) / (s * (\tau*s + 1))$ 
461 # Let  $\tau = 0.001$  (very fast filter pole)
462  $\tau = 0.001$ 
463
464 pid_tf = ct.tf([Kd_pid, Kp_pid, Ki_pid], [ $\tau$ , 1, 0])
465 sys_cl = ct.feedback(pid_tf * sys, 1)
466
467 t = np.linspace(0, 1.5, 1000)
468 t, y = ct.step_response(sys_cl, T=t)
469
470 # Metrics
471 y_final = y[-1]
472 y_peak = np.max(y)
473 Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
474
475 plt.figure(figsize=(10, 6))
476 plt.plot(t, y, linewidth=2, color=colors[3]) # Purple
477 plt.title(f'Resposta ao Degrau (PID Ziegler-Nichols)', color='white'
if mode=='dark' else 'black')
478 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
479 plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%',
bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors
[1]), color='white')
481 plt.savefig(os.path.join(assets_dir, 'step_response_PID.png'))
482 plt.close()
483
484 return pid_tf
485
486 def create_plant_variation(Km, am, ae):

```

```

487     """
488     Cria variação da planta para análise de robustez (Nicolas).
489     Nominal: Km=1.1, am=13.2, ae=950 -> K_sys=772 fixo.
490     """
491     K_sys = 772
492     num = [Km * K_sys]
493     den = [1, (am + ae), (am * ae), 0]
494     return ct.tf(num, den)
495
496 def analyze_robustness(controllers_dict, mode='dark'):
497     """
498     Análise de Robustez baseada nos cenários do Nicolas.
499     """
500     scenarios = {
501         "Nominal": {"Km": 1.1, "am": 13.2, "ae": 950, "style": "-", "
502         color_dark": "#00ff00", "color_light": "green"},
503         "Pesado": {"Km": 0.8, "am": 15.0, "ae": 1100, "style": "--",
504         "color_dark": "#00bfff", "color_light": "blue"},
505         "Agressivo": {"Km": 1.2, "am": 10.0, "ae": 800, "style": "-.",
506         "color_dark": "#ff4500", "color_light": "red"}
507     }
508
509     colors = configure_plot_style(mode)
510     assets_dir = get_assets_dir(mode)
511
512     if mode == 'light':
513         text_color = 'black'
514         grid_color = 'black'
515         face_color = 'white'
516         grid_alpha = 0.3
517     else:
518         text_color = 'white'
519         grid_color = 'white'
520         face_color = 'black'
521         grid_alpha = 0.3
522
523     for ctrl_name, ctrl in controllers_dict.items():
524         plt.figure(figsize=(10, 6))
525         print(f"[{mode.upper()}] Analisando Robustez: {ctrl_name}")
526
527         for name, params in scenarios.items():
528             G_var = create_plant_variation(params["Km"], params["am"],
529             params["ae"])
530             sys_cl = ct.feedback(ctrl * G_var, 1)
531
532             # 2 segundos é suficiente para ver a estabilidade
533             t, y = ct.step_response(sys_cl, T=np.linspace(0, 2.0, 1000))
534
535             color = params["color_dark"] if mode == 'dark' else params["
536             color_light"]
537
538             # Metrics for legend
539             y_peak = np.max(y)
540             mp = (y_peak - 1) * 100
541
542             plt.plot(t, y, linestyle=params["style"], linewidth=2, label
543             =f"{name} (Mp={mp:.1f}%)", color=color)

```

```

539     plt.axhline(1.0, color=text_color, linestyle=':', linewidth=0.8,
alpha=0.5)
540
541     plt.grid(True, which='both', linestyle='--', linewidth=0.5,
color=grid_color, alpha=grid_alpha)
542
543     plt.title(f'Análise de Robustez - {ctrl_name}', color=text_color
, fontsize=14)
544     plt.xlabel('Tempo (s)', color=text_color, fontsize=12)
545     plt.ylabel('Amplitude', color=text_color, fontsize=12)
546
547     # Legend with transparency adjustment for dark mode to look good
548     legend = plt.legend(facecolor=face_color, edgecolor=text_color)
549     for text in legend.get_texts():
550         text.set_color(text_color)
551
552     plt.tick_params(colors=text_color, which='both')
553     for spine in plt.gca().spines.values():
554         spine.set_color(text_color)
555
556     plt.tight_layout()
557     save_path = os.path.join(assets_dir, f"robustness_{ctrl_name.
replace(' ', ' ').png")
558     plt.savefig(save_path, transparent=True)
559     plt.close()
560
561 if __name__ == "__main__":
562     sys = define_system()
563
564     for mode in ['dark', 'light']:
565         print(f"\n--- Running Control Simulation in {mode.upper()} mode
---")
566
567         # 0. Open Loop Analysis (Poles, Zeros, Step)
568         analyze_open_loop(sys, mode=mode)
569
570         # 1. Proportional Controller
571         best_Kp = 138
572         design_p_controller(sys, Kp=best_Kp, mode=mode)
573
574         # 2. Lead Controller
575         ctrl_lead = design_lead_controller(sys, mode=mode)
576
577         # 3. Lag Controller
578         z_lag = 0.10
579         p_lag = 0.01
580         K_lag = 150.6
581         ctrl_lag = design_lag_controller(sys, Kp=K_lag, z=z_lag, p=p_lag
, mode=mode)
582
583         # 4. Integrated Lead-Lag Controller
584         ctrl_leadlag = design_lead_lag_controller(sys, mode=mode)
585
586         # 5. PID Controller
587         ctrl_pid = design_pid_controller(sys, mode=mode)
588
589         # 6. Comparative Plots (Rich Details)
590         generate_comparative_plots(sys, Kp=K_lag, z=z_lag, p=p_lag, mode

```

```

591     =mode)
592     # 6. Robustness Analysis (Nicolas)
593     controllers_to_test = {
594         "Lead": ctrl_lead,
595         "Lag": ctrl_lag,
596         "Lead-Lag": ctrl_leadlag,
597         "PID": ctrl_pid
598     }
599     analyze_robustness(controllers_to_test, mode=mode)
600
601     print("\nTodas as simulações e gráficos foram atualizados.")

```

Listing 2: Script de projeto e simulação dos controladores P e Lag