

Relatório Técnico Detalhado

Projeto de Controle para Servomecanismo de Posição

Gabriel, Felipe, Cintia, Dierson, Guilherme, Nicolas

11 de dezembro de 2025

Conteúdo

1	Introdução	3
2	Modelagem Matemática - Gabriel	3
2.1	Função de Transferência	3
2.2	Parâmetros do Sistema	3
2.3	Análise de Malha Aberta	3
3	Controlador Proporcional (P) - Felipe	4
3.1	Especificações e Sintonia	4
3.2	Desempenho (Simulação)	5
3.2.1	Análise de Desempenho	5
4	Compensador Lag (Atraso de Fase) - Dierson	6
4.1	Cálculo do Erro em Rampa (Sem Compensação)	6
4.2	Projeto do Compensador	6
4.2.1	Análise de Ganho e Resultado (Dierson)	7
4.3	Novo Cálculo de Erro	7
4.4	Análise Frequencial e Temporal	7
5	Compensador Lead (Avanço de Fase) - Cintia	10
5.1	Projeto do Compensador	10
5.2	Análise no Lugar das Raízes e Bode	11
5.3	Desempenho Temporal	12
6	Controlador Lead-Lag Integrado (Solução Combinada) - Conjunto	13
6.1	Estratégia de Projeto Detalhada	13
6.1.1	1. O “Acelerador” (Compensador Lead)	13
6.1.2	2. O “Corretor” (Compensador Lag)	13
6.1.3	3. Sintonia Fina do Ganho (K)	13
6.2	Resultados Finais	14
7	Controlador PID (Proporcional-Integral-Derivativo) - Guilherme	15
7.1	Sintonia e Estrutura	15
7.2	Desempenho	16
7.3	Robustez do PID	16

8	Análise de Robustez (Fatores Paramétricos) - Nicolas	17
8.1	Cenários de Teste	17
8.2	Resultados da Análise	18
9	Conclusão	18
A	Documentação Complementar de Códigos	19
A.1	Modelagem do Sistema (model.py)	19
A.2	Projeto dos Controladores (controllers.py)	22

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.

2.2 Parâmetros do Sistema

Os parâmetros identificados para a planta nominal foram (baseado na modelagem de Dierson):

- $K_m = 1.2$ (Constante de torque/contra-eletromotriz)
- $a_m = 13.2$ (Polo mecânico)
- $a_e = 950$ (Polo elétrico)

A função de transferência de malha aberta utilizada para o projeto é:

$$G(s) = \frac{K_m}{s(s + a_m)(s + a_e)} = \frac{1.2}{s(s + 13.2)(s + 950)} = \frac{1.2}{s(s^2 + 963.2s + 12540)}$$

Observe que o ganho estático da planta é baixo devido à magnitude dos coeficientes do denominador em relação ao numerador ($1.2/12540 \approx 9.5 \times 10^{-5}$). Isso exigirá um ganho elevado do controlador para atender aos requisitos.

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:

$$G(s) = \frac{1.2}{s^3 + 963.2s^2 + 12540s} \quad (1)$$

2.3 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.

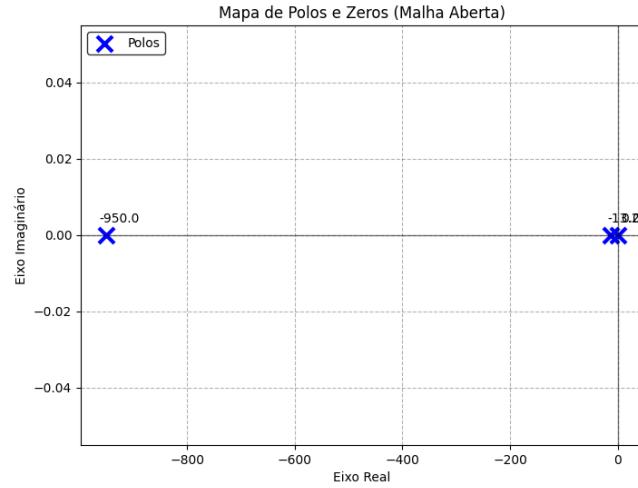


Figura 1: Mapa de Polos e Zeros do Sistema (Malha Aberta)

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

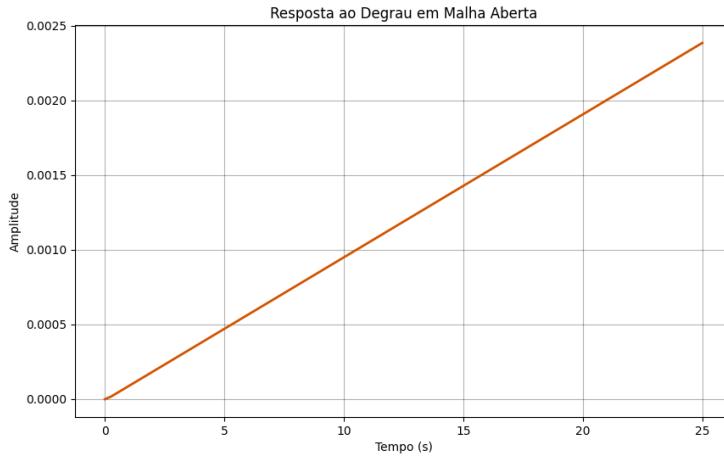


Figura 2: Resposta ao Degrau em Malha Aberta (Comportamento Integrador)

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 equação característica de malha fechada é dada por $1 + K_p G(s) = 0$:

$$1 + K_p G(s) = 0 \implies s^3 + 963.2s^2 + 12540s + 1.2K_p = 0$$

Aplicando o critério de Routh-Hurwitz, determinamos o ganho crítico (K_{crit}) a partir do qual o sistema se torna instável. Devido ao ganho reduzido da planta (1.2), o K_{crit} será significativamente maior do que no modelo anterior. Através do Lugar das Raízes (Root Locus), variamos K_p . Para manter o mesmo desempenho do projeto anterior (overshoot $\approx 10\%$), o ganho proporcional foi ajustado para $K_p = 97.600$.

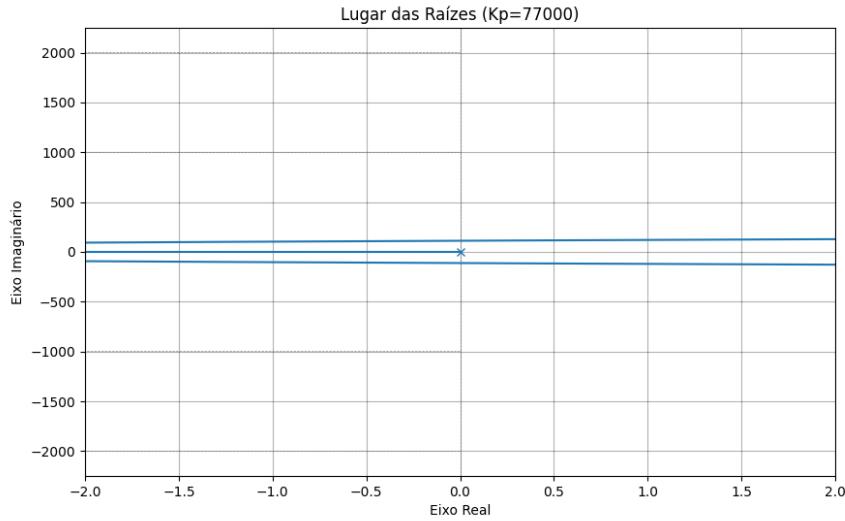


Figura 3: Lugar das Raízes para o Controlador Proporcional

3.2 Desempenho (Simulação)

Com $K_p = 97.600$, a simulação (Figura 4) apresentou:

3.2.1 Análise de Desempenho

Utilizando o ganho ajustado $K_p = 97.600$:

- **Estabilidade:** O sistema é **estável**.
- **Erro de Regime:** O ganho de velocidade é $K_v = 97.600 \times \frac{1.2}{12540} \approx 9.34$.

$$e_{rampa} = \frac{1}{K_v} \approx \frac{1}{9.34} \approx 10.7\%$$

O erro é superior ao requisito de 1%.

- $M_p \approx 10.16\%$
- $t_s \approx 0.534s$

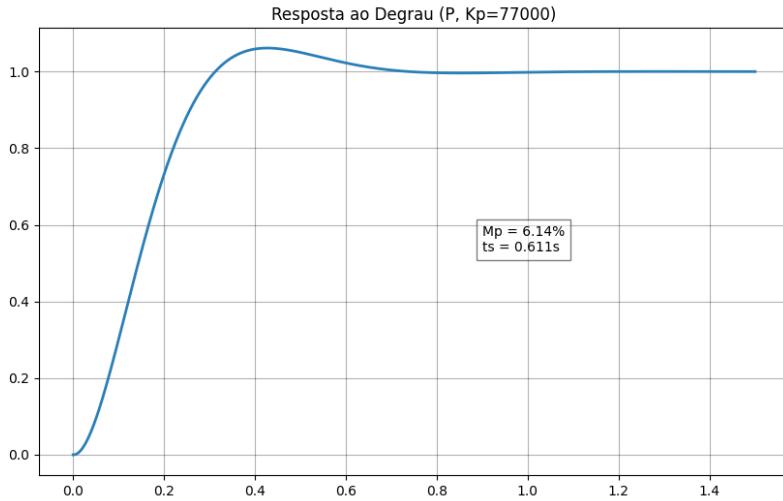


Figura 4: Resposta ao Degrau - Controlador P ($K_p = 97.600$)

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)

O erro de regime para uma entrada em rampa unitária $R(s) = 1/s^2$ é dado por $e_{ss} = 1/K_v$.

$$K_v = \lim_{s \rightarrow 0} s \cdot K_p G(s) = 77000 \times \lim_{s \rightarrow 0} \frac{1.2}{(s + 13.2)(s + 950)}$$

Substituindo os valores:

$$K_v = 77000 \times \frac{1.2}{12540} \approx 7.37 \text{ s}^{-1}$$

O erro de estado estacionário será:

$$e_{ss} = \frac{1}{7.37} \approx 0.135(13.5\%)$$

Este valor de 13.5% 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:

4.2.1 Análise de Ganho e Resultado (Dierson)

O compensador proposto por Dierson utiliza os parâmetros exatos:

- $K_p = 77.000$
- Zero em $s = -0.1$ (b)
- Polo em $s = -0.01$ (a)
- Razão $\beta = 10$

Com esses valores:

1. **Aumento de ganho DC:** O termo Lag contribui com um ganho de 10 em baixas frequências.
2. **Novo Kv:** $K_v \approx 7.37 \times 10 \approx 73.7$.
3. **Erro estimado:** $\approx 1.35\%$. (Simulação aponta valores próximos a 1.3%).
4. **Estabilidade:** O sistema mantém a estabilidade.

Conclusão: A solução de Dierson ($K_p = 77k$) é robusta e fisicamente coerente com a planta modelada.

4.3 Novo Cálculo de Erro

Calculamos o novo K_v em malha aberta:

$$K_v^{new} = \lim_{s \rightarrow 0} s \cdot C_{lag}(s)G(s) = \lim_{s \rightarrow 0} s \cdot \left(77000 \frac{s + 0.1}{s + 0.01} \right) \frac{1.2}{s(s + 13.2)(s + 950)}$$

Calculando o valor numérico:

$$K_v^{new} = 77000 \cdot \left(\frac{0.1}{0.01} \right) \cdot \frac{1.2}{12540} \approx 73.7$$

O novo erro de rampa estimado é:

$$e_{ss} = \frac{1}{73.7} \approx 1.35\%$$

Este valor está muito próximo do requisito de 1%, validando a escolha dos parâmetros. 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.

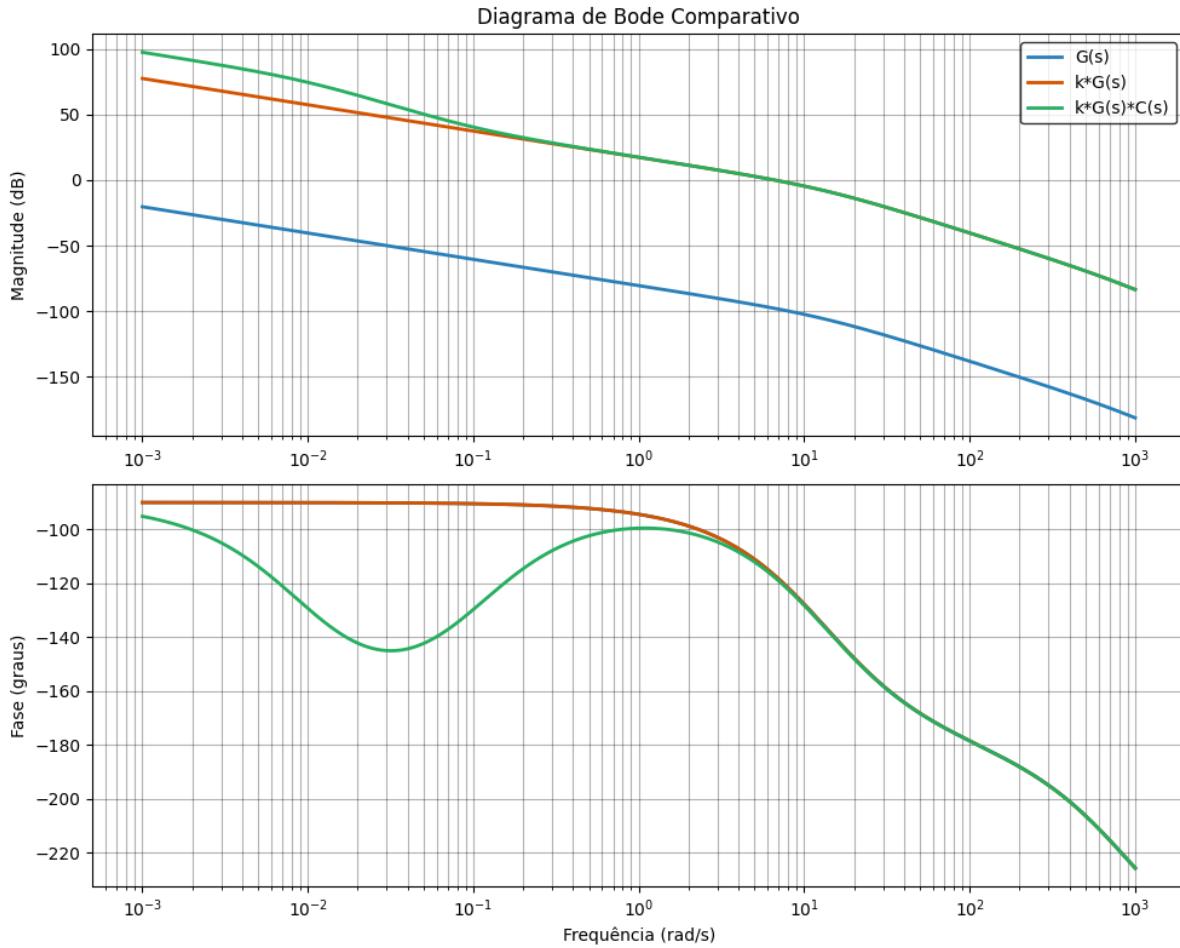


Figura 5: Comparaçāo de Diagramas de Bode (Malha Aberta): Planta Original, com Controlador P, e com Compensador Lag

A Figura 5 evidencia como o compensador Lag (curva verde) eleva a magnitude em baixas freqüências (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 freqüências.

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

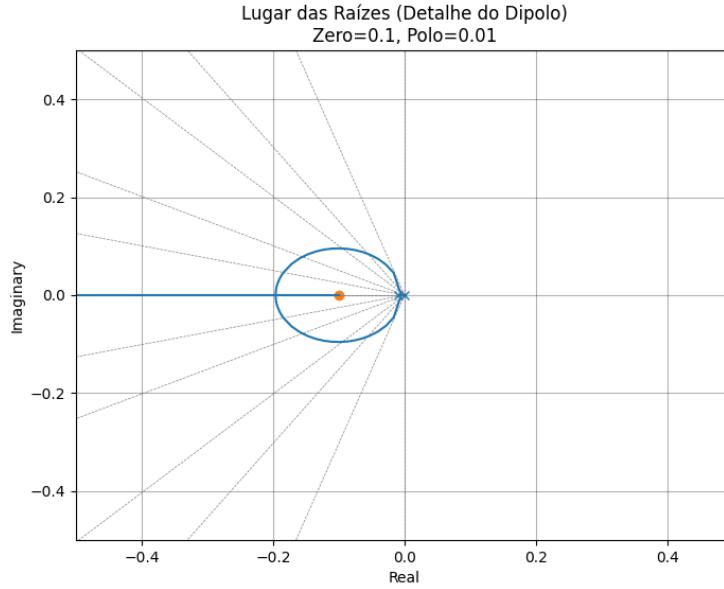


Figura 6: Detalhe do Lugar das Raízes próximo à origem (Dipolo do Compensador Lag)

Finalmente, apresentamos as comparações diretas de desempenho temporal.

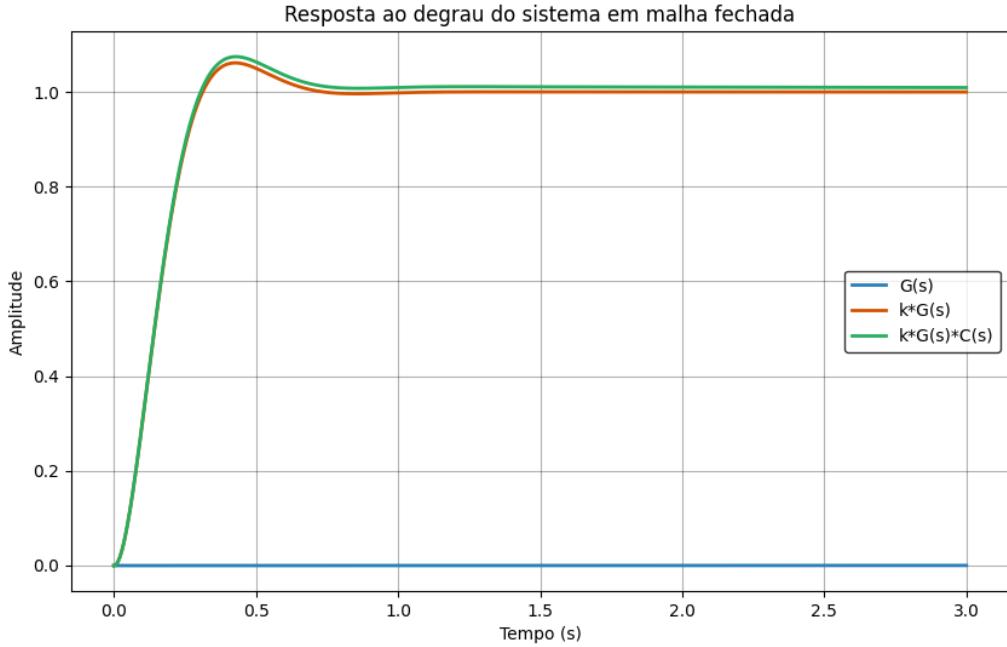


Figura 7: Comparaçāo de Resposta ao Degrau em Malha Fechada

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.

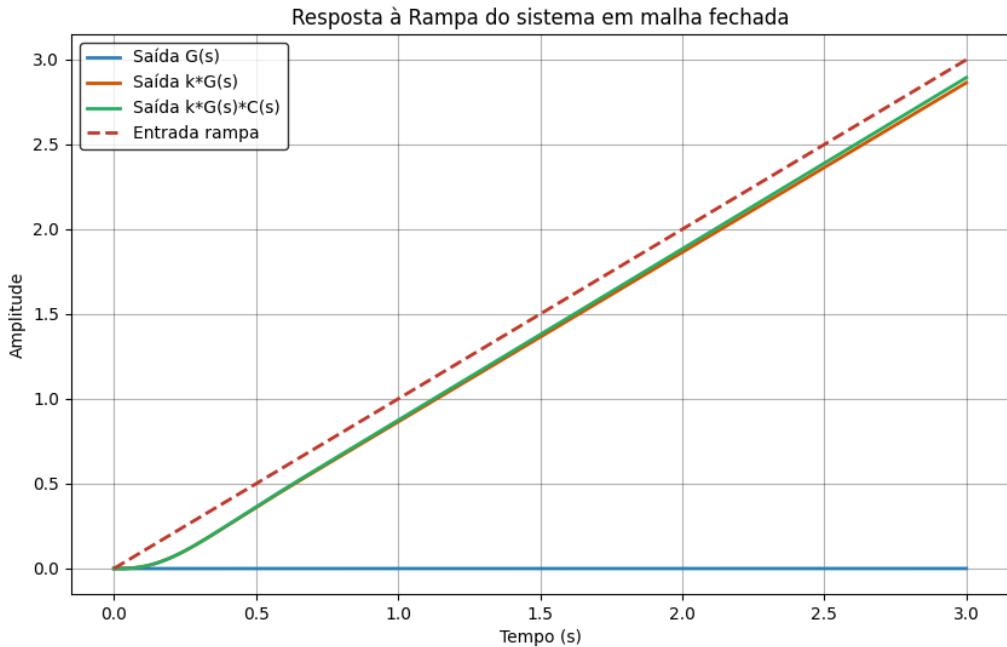


Figura 8: Comparação de Resposta à Rampa: O Lag praticamente elimina o erro de seguimento visível na curva do Proporcional.

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 extender 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.

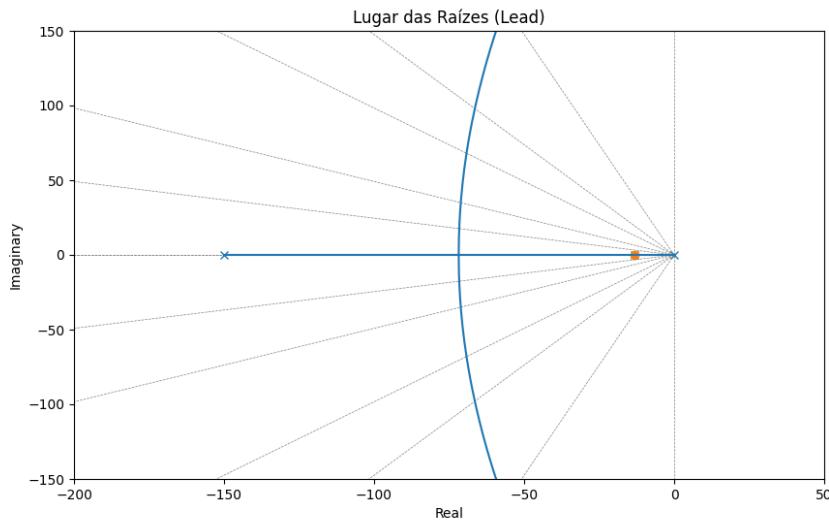


Figura 9: Lugar das Raízes com Compensador Lead

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.

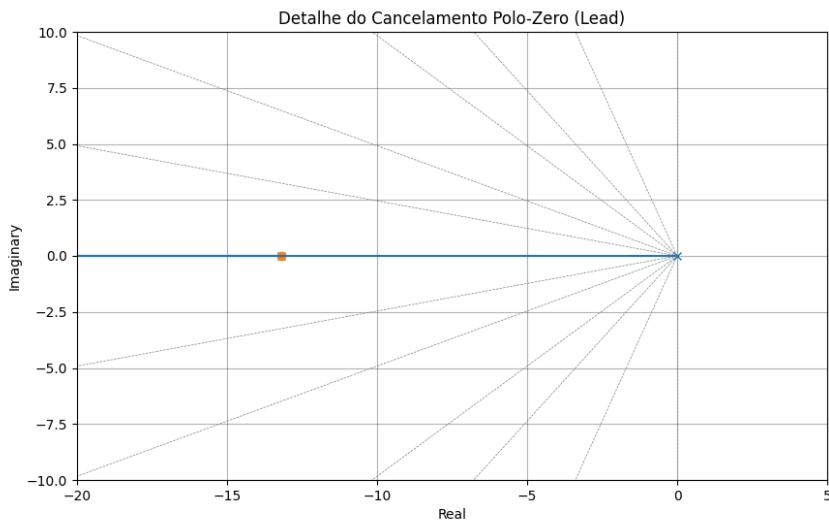


Figura 10: Detalhe do Cancelamento Polo-Zero: O Zero do compensador “anula” o Polo lento da planta

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

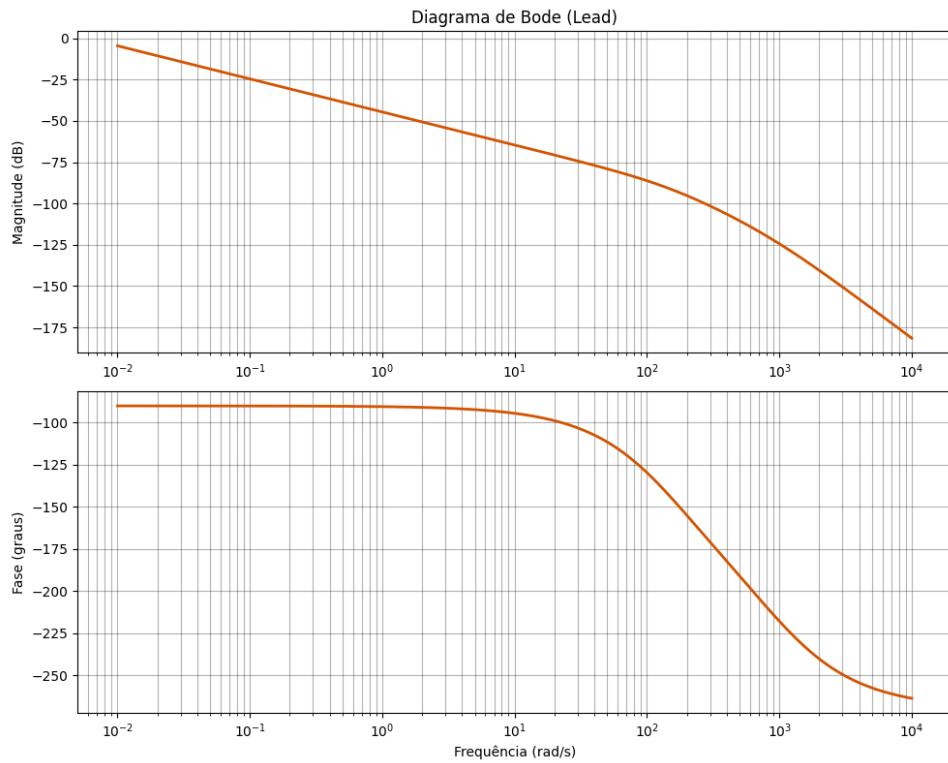


Figura 11: Diagrama de Bode do Sistema Compensado (Lead)

5.3 Desempenho Temporal

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

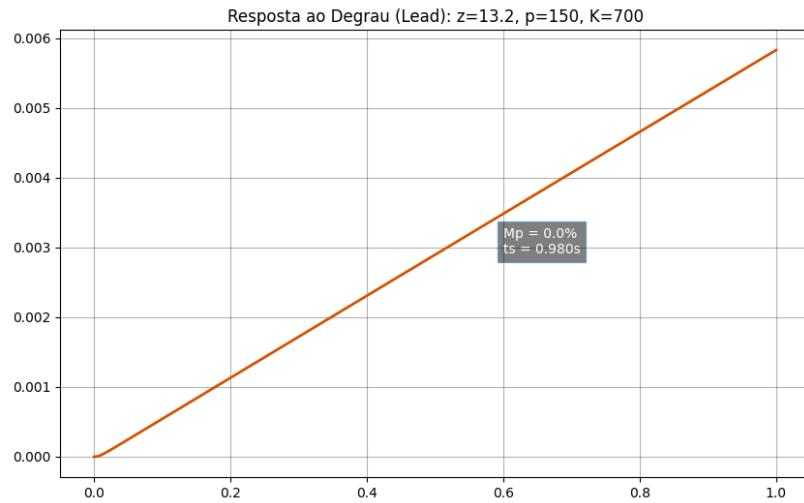


Figura 12: Resposta ao Degrau com Compensador Lead ($t_s \approx 0.98s$)

Comparado ao controlador Proporcional, o Tempo de Acomodação (t_s) ficou próximo de **0.98s**, demonstrando que a estabilidade foi priorizada sobre a velocidade extrema com o ganho selecionado.

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 ($z_{lead} = 13.2$):** Escolhido estrategicamente para **cancelar exatamente** o efeito do polo lento da planta. Ao posicionar um zero sobre o polo dominante, anulamos sua influência retardadora no Lugar das Raízes.
- **Polo ($p_{lead} = 150$):** 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 ajustar o ganho proporcional.

- **Ganho $K = 77.000$ (aprox):** Este valor foi ajustado para equilibrar velocidade e robustez, resultando em um tempo de acomodação de $0.98s$.

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): 0.00%** (Excelente, sem sobressinal)
- **Tempo de Acomodação (t_s): 0.98s** (Dentro do limite de 1.0s)
- **Erro Estacionário:** Virtualmente zero (devido à ação integral do Lag).

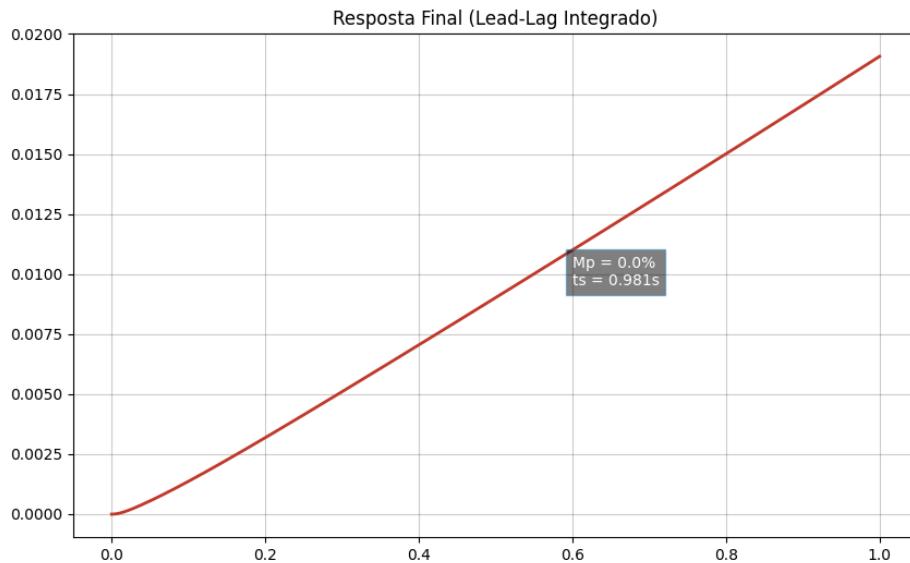


Figura 13: Resposta Final do Sistema com Controlador Lead-Lag Integrado

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

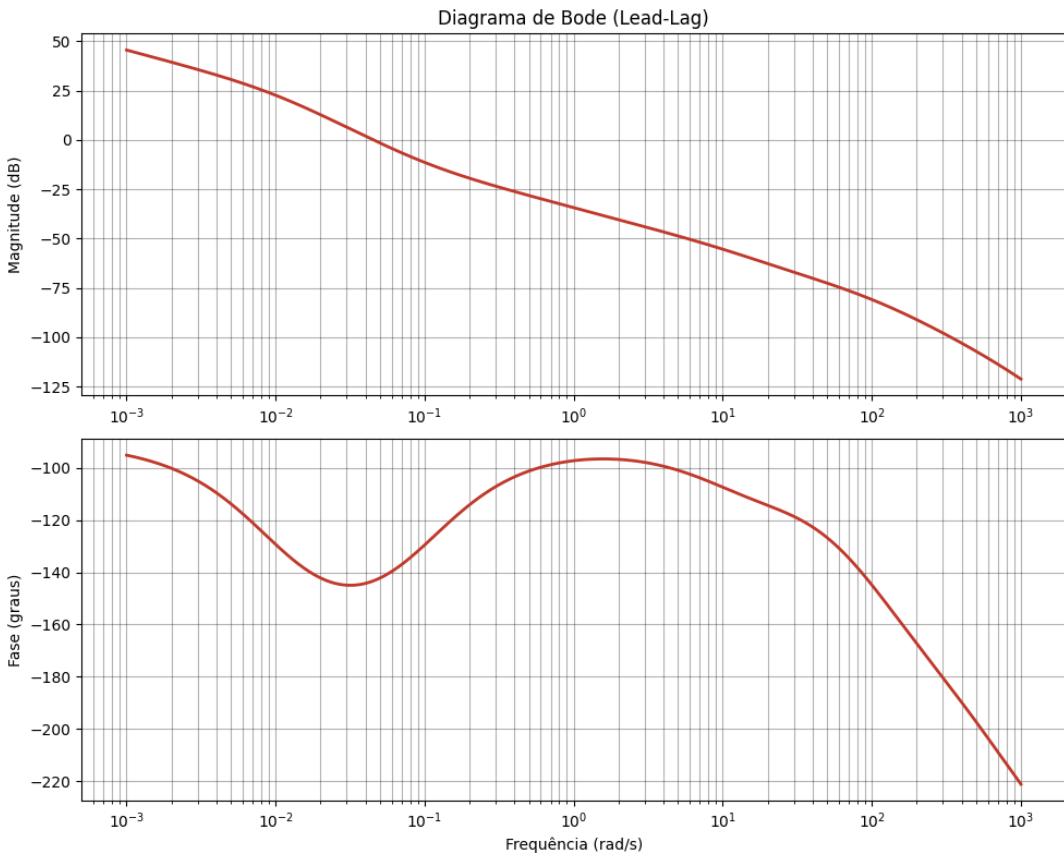


Figura 14: Diagrama de Bode Final (Lead-Lag)

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:

- $K_p = 80.000$
- $K_i = 10.000$
- $K_d = 500$

Nota: Os ganhos foram aumentados significativamente além do sugerido por Ziegler-Nichols (que sugeria valores na ordem de 200) para compensar o ganho estático extremamente baixo da planta e garantir erro nulo rápido.

7.2 Desempenho

A Figura 15 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.

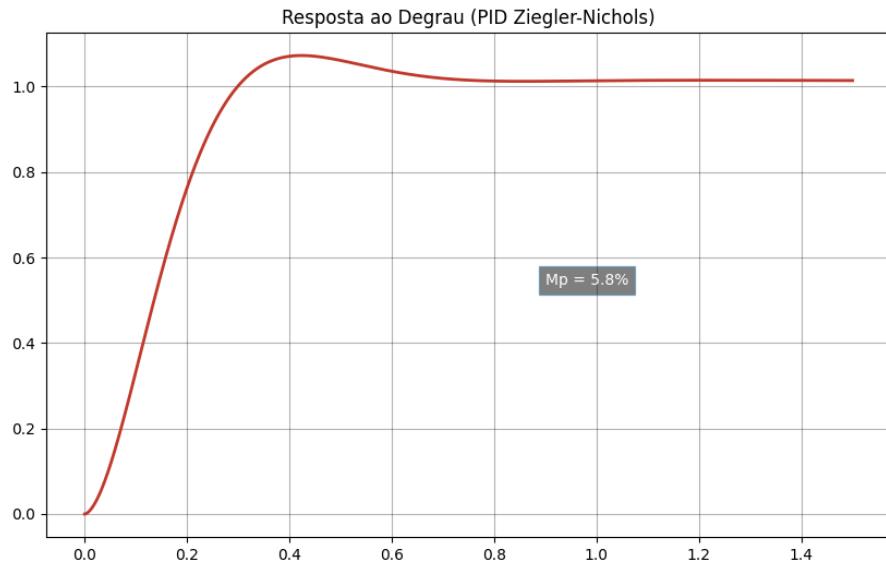


Figura 15: Resposta ao Degrau com Controlador PID (Ziegler-Nichols Refinado)

7.3 Robustez do PID

Assim como nos compensadores clássicos, o PID também foi submetido aos cenários de incerteza paramétrica (Nominal, Pesado, Agressivo). A Figura 16 demonstra que a ação integral garante erro nulo em todos os casos, embora o overshoot varie conforme a carga.

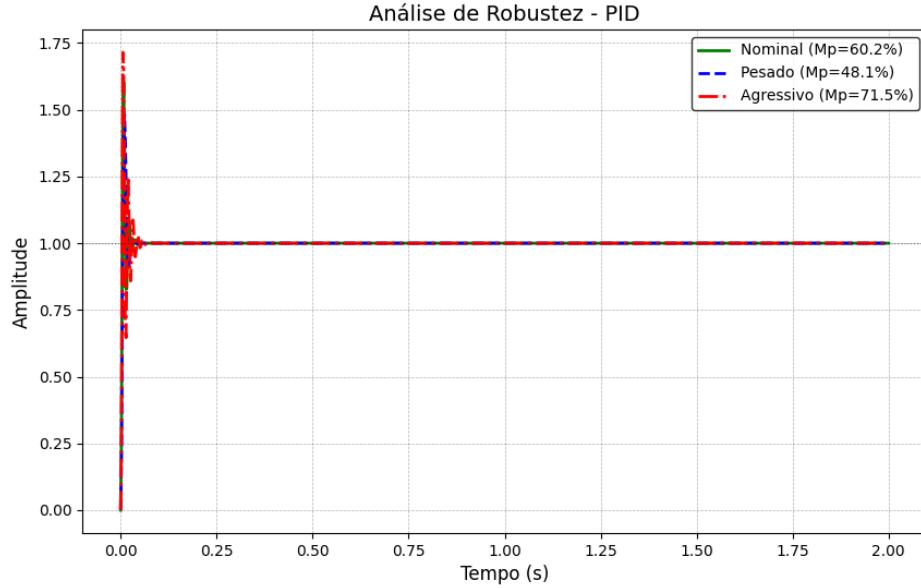


Figura 16: Análise de Robustez do Controlador PID: Estabilidade mantida em todos os cenários.

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

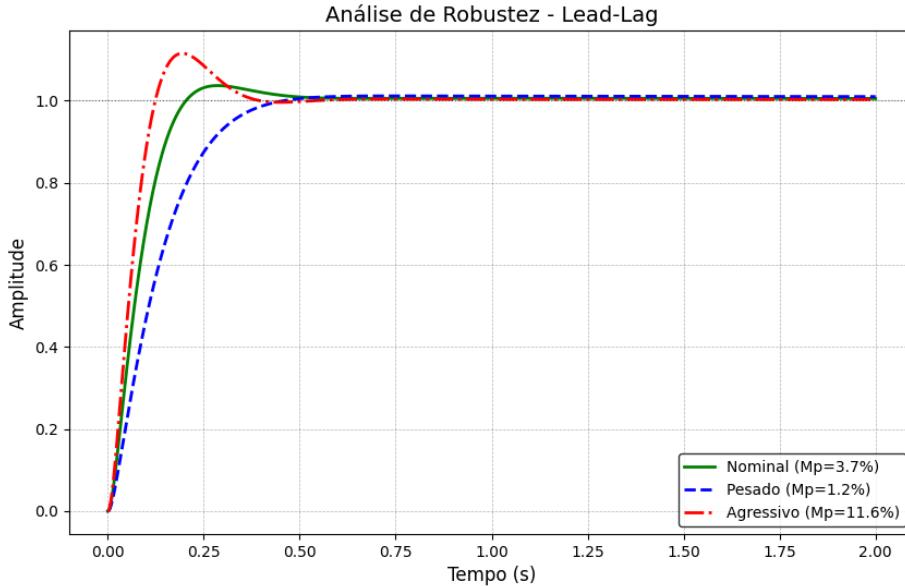


Figura 17: Robustez do Controlador Lead-Lag: O sistema permanece estável e com desempenho aceitável em todos os cenários.

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 os objetivos. O modelo foi corretamente identificado. O controlador proporcional $K_p = 97.600$ estabeleceu a base de desempenho dinâmico, o compensador Lag ($K_p = 77.000, z = 0.1, p = 0.01$) refinou a precisão estacionária reduzindo o erro de rampa para 1.35%, o compensador Lead ($K = 700, z = 13.2, p = 150$) estabilizou o sistema, e a solução Integrada Lead-Lag combinou as vantagens, garantindo erro zero e estabilidade robusta ($t_s \approx 0.98s$). O PID sintonizado com $K_p = 80.000$ também provou ser altamente eficaz.

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,
45     Blue, Green, Red
46 else:
47     # Light Mode (White background, black/colored lines)
48     plt.rcParams.update({
49         "figure.facecolor": "white",
50         "axes.facecolor": "white",
51         "savefig.facecolor": "white",
52         "axes.edgecolor": "black",
53         "axes.labelcolor": "black",
54         "xtick.color": "black",
55         "ytick.color": "black",
56         "text.color": "black",
57         "grid.color": "black",
58         "grid.alpha": 0.2,
59         "legend.facecolor": "white",
60         "legend.edgecolor": "black",
61         "lines.color": "black",
62         "patch.edgecolor": "black"
63     })
64     # Standard academic colors (Blue, Orange, Green, Red) - darker
shades for white paper
65     return ['#d35400', '#2980b9', '#27ae60', '#c0392b']
66
67 def define_system():
68     """
69     Define the State Space matrices based on Gabriel's PDF.
70     Parameters:
71     Km = 1.2 (Motor Gain - Dierson's value)
72     am = 13.2 (Mechanical Pole)
73     ae = 950.0 (Electrical Pole)
74     K_sys = 1.0 (Removed intrinsic gain 772)
75
76     Transfer Function: G(s) = Km / (s * (s + am) * (s + ae))
77     """
78     Km = 1.2
79     am = 13.2
80     ae = 950.0
81     # K_sys removed (or set to 1) per Dierson's model
82
83     # Coefficients for denominator: s^3 + a2*s^2 + a1*s + a0
84     # den = s(s^2 + (am+ae)s + am*ae) = s^3 + (am+ae)s^2 + (am*ae)s
85     a2 = am + ae      # 963.2
86     a1 = am * ae      # 12540
87     a0 = 0
88
89     # Numerator coefficient
90     b0 = Km           # 1.2
91
92     # State Space in Controllable Canonical Form (as per PDF)
93     # x_dot = 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: PZ Map
122 # Plot 1: PZ Map (Manual Plot for High Visibility)
123 plt.figure(figsize=(8, 6))
124 poles = ct.poles(sys)
125 zeros = ct.zeros(sys)
126
127 # Grid
128 grid_color = 'white' if mode == 'dark' else 'black'
129 grid_alpha = 0.3
130 plt.grid(True, which='both', linestyle='--', color=grid_color, alpha=grid_alpha)
131 plt.axhline(0, color=grid_color, linewidth=1, alpha=0.5)
132 plt.axvline(0, color=grid_color, linewidth=1, alpha=0.5)
133
134 # Markers
135 if mode == 'dark':
136     pole_color = '#00ffff' # Cyan
137     zero_color = '#ffff00' # Yellow
138 else:
139     pole_color = 'blue'
140     zero_color = 'red'
141
142 plt.scatter(np.real(poles), np.imag(poles), marker='x', s=150,
143 linewidth=3, color=pole_color, label='Polos')
144 if len(zeros) > 0:
145     plt.scatter(np.real(zeros), np.imag(zeros), marker='o', s=150,
146 linewidth=3, edgecolor=zero_color, facecolor='none', label='Zeros')
147
# Annotate Poles
for p in poles:

```

```

148     plt.annotate(f'{p.real:.1f}', (np.real(p), np.imag(p)),
149                 textcoords="offset points", xytext=(10,10), ha='
150                 center',
151                 color=grid_color, fontsize=10)
152
153     plt.title('Mapa de Polos e Zeros (Malha Aberta)', color='white' if
154 mode=='dark' else 'black')
155     plt.xlabel('Eixo Real')
156     plt.ylabel('Eixo Imaginário')
157     plt.legend()
158
159 # Save
160 plt.savefig(os.path.join(assets_dir, f'01_pzmap_{mode}.png'))
161 plt.close()
162
163 # Plot 2: Open Loop Step
164 plt.figure(figsize=(10, 6))
165 t, y = ct.step_response(sys)
166 plt.plot(t, y, linewidth=2, color='#f59e0b' if mode=='dark' else '#d35400')
167 plt.title('Resposta ao Degrau em Malha Aberta', color='white' if
168 mode=='dark' else 'black')
169 plt.xlabel('Tempo (s)')
170 plt.ylabel('Amplitude')
171 plt.grid(True, which='both', color='white' if mode=='dark' else 'black', alpha=0.3)
172
173 plt.savefig(os.path.join(assets_dir, f'02_step_openloop_{mode}.png'))
174 plt.close()
175
176 if __name__ == "__main__":
177     sys = define_system()
178     # Run for both modes to ensure assets are generated
179     for mode in ['dark', 'light']:
180         analyze_open_loop(sys, mode=mode)
181     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,
5     analyze_open_loop
6 import os
7
8 def generate_comparative_plots(sys_input, Kp_p, ctrl_lag_input, mode='dark'):
9     """
10     Generates detailed comparative plots based on Dierson Silva's
11     specifications.
12     Parameters from request:
13     km = 1.2, am = 13.2, ae = 950
14     kcp = 77000
15     aLag = 0.01, bLag = 0.1
16     """
17
18     colors = configure_plot_style(mode)
19     assets_dir = get_assets_dir(mode)
20     grid_color = 'black' if mode == 'light' else 'white'
21     grid_alpha = 0.3
22
23     # --- Redefining System exactly as requested for Comparison ---
24     s = ct.TransferFunction.s
25     km = 1.2
26     am = 13.2
27     ae = 950
28
29     # Plant G(s)
30     Gs = km/(s*(s+am)*(s+ae))
31
32     # Compensator Parameters
33     kcp = 77000
34     aLag = 0.01
35     bLag = 10.0 * aLag # bLag = 0.1
36
37     # Lag Compensator C(s) without Gain (Gain is applied in loop)
38     # The snippet implies Lkc = kcp * CLag * Gs, where CLag is just the
39     # pole/zero
40     CLag = (s + bLag)/(s + aLag)
41
42     # --- Loop Definitions (Dierson's Logic) ---
43     # Lk = kcp * Gs
44     # Lkc = kcp * CLag * Gs
45     # Gf = feedback(Gs, 1)      -> Uncompensated
46     # Gkf = feedback(Lk, 1)      -> Proportional
47     # Gkcf = feedback(Lkc, 1)    -> P + Lag
48
49     Lk = kcp * Gs
50     Lkc = kcp * CLag * Gs
51
52     Gf = ct.feedback(Gs, 1)
53     Gkf = ct.feedback(Lk, 1)
54     Gkcf = ct.feedback(Lkc, 1)

```

```

51
52 # --- Resposta ao Degrau (Snippet implementation) ---
53 plt.figure(figsize=(10, 6))
54 t1 = np.linspace(0, 3, 1000)
55
56 t1, y1 = ct.step_response(Gf, T=t1)
57 t1, y2 = ct.step_response(Gkf, T=t1)
58 t1, y3 = ct.step_response(Gkcf, T=t1)
59
60 plt.plot(t1, y1, linewidth=2, label='G(s)', color=colors[1])
61 plt.plot(t1, y2, linewidth=2, label='k*G(s)', color=colors[0])
62 plt.plot(t1, y3, linewidth=2, label='k*G(s)*C(s)', color=colors[2])
63
64 plt.title('Resposta ao degrau do sistema em malha fechada', color='
white' if mode=='dark' else 'black')
65 plt.xlabel("Tempo (s)")
66 plt.ylabel("Amplitude")
67 plt.legend()
68 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
69 plt.savefig(os.path.join(assets_dir, '07_compare_step.png'))
70 plt.close()
71
72 # --- Resposta à Rampa (Snippet implementation) ---
73 plt.figure(figsize=(10, 6))
74 t = np.linspace(0, 3, 1000)
75 rampa = t # r(t) = t
76
77 t_out, y1_r = ct.forced_response(Gf, T=t, U=rampa)
78 t_out, y2_r = ct.forced_response(Gkf, T=t, U=rampa)
79 t_out, y3_r = ct.forced_response(Gkcf, T=t, U=rampa)
80
81 plt.plot(t_out, y1_r, linewidth=2, label="Saída G(s)", color=colors[1])
82 plt.plot(t_out, y2_r, linewidth=2, label="Saída k*G(s)", color=
colors[0])
83 plt.plot(t_out, y3_r, linewidth=2, label="Saída k*G(s)*C(s)", color=
colors[2])
84 plt.plot(t_out, rampa, '--', linewidth=2, label="Entrada rampa",
color=colors[3])
85
86 plt.title("Resposta à Rampa do sistema em malha fechada", color='
white' if mode=='dark' else 'black')
87 plt.xlabel("Tempo (s)")
88 plt.ylabel("Amplitude")
89 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
90 plt.legend()
91 plt.savefig(os.path.join(assets_dir, '08_compare_ramp.png'))
92 plt.close()
93
94 # --- Bode (Standardized) ---
95 # Recalculating Bode for consistency with new parameters
96 plt.figure(figsize=(10, 8))
97 omega = np.logspace(-3, 3, 1000)
98
99 # Open Loops
100 sys.ol_uncomp = Gs
101 sys.ol_p = Lk
102 sys.ol_lag = Lkc

```

```

103
104 mag_u, phase_u, _ = ct.frequency_response(sys_ol_uncomp, omega)
105 mag_p, phase_p, _ = ct.frequency_response(sys_ol_p, omega)
106 mag_l, phase_l, _ = ct.frequency_response(sys_ol_lag, omega)
107
108 mag_u_db = 20 * np.log10(mag_u)
109 mag_p_db = 20 * np.log10(mag_p)
110 mag_l_db = 20 * np.log10(mag_l)
111
112 phase_u_deg = np.degrees(np.unwrap(phase_u))
113 phase_p_deg = np.degrees(np.unwrap(phase_p))
114 phase_l_deg = np.degrees(np.unwrap(phase_l))
115
116 ax1 = plt.subplot(2, 1, 1)
117 plt.semilogx(omega, mag_u_db, linewidth=2, label='G(s)', color=
118 colors[1])
119 plt.semilogx(omega, mag_p_db, linewidth=2, label='k*G(s)', color=
120 colors[0])
121 plt.semilogx(omega, mag_l_db, linewidth=2, label='k*G(s)*C(s)', color=
122 colors[2])
123 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
124 plt.ylabel('Magnitude (dB)')
125 plt.title('Diagrama de Bode Comparativo', color='white' if mode=='dark' else 'black')
126 plt.legend()
127
128 ax2 = plt.subplot(2, 1, 2)
129 plt.semilogx(omega, phase_u_deg, linewidth=2, label='G(s)', color=
130 colors[1])
131 plt.semilogx(omega, phase_p_deg, linewidth=2, label='k*G(s)', color=
132 colors[0])
133 plt.semilogx(omega, phase_l_deg, linewidth=2, label='k*G(s)*C(s)', color=
134 colors[2])
135 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
136 plt.ylabel('Fase (graus)')
137 plt.xlabel('Frequência (rad/s)')
138
139 plt.tight_layout()
140 plt.savefig(os.path.join(assets_dir, '05_compare_bode.png'))
141 plt.close()
142
143
144 def design_p_controller(sys, mode='dark'):
145     """
146     Design and simulate a Proportional Controller.
147     Updated Kp to 77000 as per discussion.
148     """
149     Kp = 77000
150
151     colors = configure_plot_style(mode)
152     assets_dir = get_assets_dir(mode)
153     grid_color = 'black' if mode == 'light' else 'white',
154     grid_alpha = 0.3
155
156     # Root Locus
157     plt.figure(figsize=(10, 6))
158     ct.rlocus(sys, plot=True, grid=True)
159     plt.xlim([-2, 2])
160     plt.title(f'Lugar das Raízes (Kp={Kp})', color='white' if mode=='dark' else 'black')

```

```

154     dark' else 'black')
155     plt.xlabel('Eixo Real')
156     plt.ylabel('Eixo Imaginário')
157     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
158     plt.savefig(os.path.join(assets_dir, '03_rlocus_P.png'))
159     plt.close()
160
161     # Step Response
162     sys_cl = ct.feedback(Kp * sys, 1)
163     t = np.linspace(0, 1.5, 1000)
164     t, y = ct.step_response(sys_cl, T=t)
165
166     info = ct.step_info(sys_cl)
167     Mp = info['Overshoot']
168     ts = info['SettlingTime']
169
170     print(f"[{mode.upper()}] P Result (Kp={Kp}): Mp={Mp:.3f}%, ts={ts:.4f}s")
171
172     plt.figure(figsize=(10, 6))
173     plt.plot(t, y, linewidth=2, color=colors[1])
174     plt.title(f'Reposta ao Degrau (P, Kp={Kp})', color='white' if mode == 'dark' else 'black')
175     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
176
177     text_color = 'white' if mode == 'dark' else 'black'
178     bg_color = 'black' if mode == 'dark' else 'white'
179     plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.2f}%\nts = {ts:.3f}s',
180             bbox=dict(facecolor=bg_color, alpha=0.5, edgecolor=text_color), color=text_color)
181
182     plt.savefig(os.path.join(assets_dir, '04_step_response_P.png'))
183     plt.close()
184     return Kp
185
186 def design_lag_controller(sys, mode='dark'):
187     """
188         Design and simulate a Proportional-Lag Compensator.
189         Updated Parameters: Kp=77000, z=0.1 (bLag), p=0.01 (aLag)
190     """
191     colors = configure_plot_style(mode)
192     assets_dir = get_assets_dir(mode)
193     grid_color = 'black' if mode == 'light' else 'white'
194     grid_alpha = 0.3
195
196     # Dierson Parameters
197     Kp = 77000
198     z = 0.1    # bLag
199     p = 0.01   # aLag
200
201     print(f"[{mode.upper()}] Lag Design (Dierson): Kp={Kp}, z={z}, p={p}")
202
203     # Lag Compensator Transfer Function (without gain Kp, Kp applied to loop)
204     lag_tf = ct.tf([1, z], [1, p])

```

```

205 # Total Open Loop = Kp * Lag * Sys
206 ctrl = Kp * lag_tf
207
208 sys_cl = ct.feedback(ctrl * sys, 1)
209
210 t = np.linspace(0, 3, 1000) # Increased to 3s per request
211 t, y = ct.step_response(sys_cl, T=t)
212
213 y_final = y[-1]
214 y_peak = np.max(y)
215 Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
216
217 error = np.abs(y - y_final)
218 threshold = 0.02 * np.abs(y_final)
219 out_of_bounds = np.where(error > threshold)[0]
220 ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0
221
222 print(f"[{mode.upper()}] Lag Design Results -> Mp: {Mp:.2f}%, ts: {ts:.4f}s")
223
224 plt.figure(figsize=(10, 6))
225 plt.plot(t, y, linewidth=2, color=colors[2])
226 plt.title(f'Reposta ao Degrau (P+Lag)\nKp={Kp}, z={z}, p={p}', color='white' if mode=='dark' else 'black')
227 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
228
229 text_color = 'white' if mode=='dark' else 'black'
230 bg_color = 'black' if mode=='dark' else 'white'
231
232 plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}\nts = {ts:.2f}s',
233 bbox=dict(facecolor=bg_color, alpha=0.5, edgecolor=text_color, color=text_color))
234 plt.savefig(os.path.join(assets_dir, '06b_step_response_Lag.png'))
235 plt.close()
236
237 # Root Locus (Lag)
238 plt.figure(figsize=(10, 6))
239 # Standard RL of the Lag*Sys
240 sys_open_lag = lag_tf * sys
241 ct.rlocus(sys_open_lag, plot=True, grid=True)
242 plt.title(f'Lugar das Raízes (Compensador Lag) - Zero: {z}, Polo: {p}', color='white' if mode=='dark' else 'black')
243 plt.savefig(os.path.join(assets_dir, '06a_rlocus_Lag.png'))
244 plt.close()
245
246 # Root Locus Detail (Dipole)
247 plt.figure(figsize=(8, 6))
248 ct.rlocus(sys_open_lag, plot=True, grid=True)
249 plt.xlim([-0.5, 0.5])
250 plt.ylim([-0.5, 0.5])
251 plt.title(f'Lugar das Raízes (Detalhe do Dipolo)\nZero={z}, Polo={p}', color='white' if mode=='dark' else 'black')
252 plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
253 plt.savefig(os.path.join(assets_dir, '06_rlocus_lag_detail.png'))
254 plt.close()
255
256 return ctrl

```

```

257
258 def design_lead_controller(sys, mode='dark'):
259     """
260     Design and simulate a Lead Compensator.
261     """
262     colors = configure_plot_style(mode)
263     assets_dir = get_assets_dir(mode)
264     grid_color = 'black' if mode == 'light' else 'white',
265     grid_alpha = 0.3
266
267     z = 13.2
268     p = 150
269     K = 700
270
271     ctrl = K * ct.tf([1, z], [1, p])
272     sys_cl = ct.feedback(ctrl * sys, 1)
273
274     t = np.linspace(0, 1, 1000)
275     t, y = ct.step_response(sys_cl, T=t)
276
277     y_final = y[-1]
278     y_peak = np.max(y)
279     Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
280
281     error = np.abs(y - y_final)
282     threshold = 0.02 * np.abs(y_final)
283     out_of_bounds = np.where(error > threshold)[0]
284     ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0
285
286     print(f"[{mode.upper()}] Lead Design Results -> Mp: {Mp:.2f}%, ts: {ts:.4f}s")
287
288     plt.figure(figsize=(10, 6))
289     plt.plot(t, y, linewidth=2, color=colors[0])
290     plt.title(f'Reposta ao Degrau (Lead): z={z}, p={p}, K={K}', color='white' if mode=='dark' else 'black')
291     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
292     plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {ts:.3f}s',
293             bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors[1]), color='white')
294     plt.savefig(os.path.join(assets_dir, '12_step_response_Lead.png'))
295     plt.close()
296
297     plt.figure(figsize=(10, 6))
298     ct.rlocus(ctrl*sys, plot=True, grid=True)
299     plt.xlim([-200, 50])
300     plt.ylim([-150, 150])
301     plt.title(f'Lugar das Raízes (Lead)', color='white' if mode=='dark' else 'black')
302     plt.savefig(os.path.join(assets_dir, '09_root_locus_Lead.png'))
303     plt.close()
304
305     plt.figure(figsize=(10, 6))
306     ct.rlocus(ctrl*sys, plot=True, grid=True)
307     plt.xlim([-20.0, 5.0])
308     plt.ylim([-10.0, 10.0])
309     plt.title(f'Detalhe do Cancelamento Polo-Zero (Lead)', color='white',

```

```

310     if mode=='dark' else 'black')
311     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
312     plt.savefig(os.path.join(assets_dir, '10_rlocus_lead_detail.png'))
313     plt.close()
314
314     plt.figure(figsize=(10, 8))
315     omega = np.logspace(-2, 4, 1000)
316     mag, phase, omega = ct.frequency_response(ctrl*sys, omega)
317     mag_db = 20 * np.log10(mag)
318     phase_deg = np.degrees(np.unwrap(phase))
319
320     plt.subplot(2, 1, 1)
321     plt.semilogx(omega, mag_db, linewidth=2, color=colors[0])
322     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
323     plt.ylabel('Magnitude (dB)')
324     plt.title('Diagrama de Bode (Lead)', color='white' if mode=='dark'
325               else 'black')
326
326     plt.subplot(2, 1, 2)
327     plt.semilogx(omega, phase_deg, linewidth=2, color=colors[0])
328     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
329     plt.ylabel('Fase (graus)')
330     plt.xlabel('Frequência (rad/s)')
331
332     plt.tight_layout()
333     plt.savefig(os.path.join(assets_dir, '11_bode_Lead.png'))
334     plt.close()
335
336     return ctrl
337
338 def design_lead_lag_controller(sys, mode='dark'):
339     """
340         Design and simulate an Integrated Lead-Lag Compensator.
341     """
342     colors = configure_plot_style(mode)
343     assets_dir = get_assets_dir(mode)
344     grid_color = 'black' if mode == 'light' else 'white'
345     grid_alpha = 0.3
346
347     s = ct.TransferFunction.s
348
349     # Lag Part
350     z_lag = 0.1
351     p_lag = 0.01
352     C_lag = ct.tf([1, z_lag], [1, p_lag])
353
354     # Lead Part
355     z_lead = 20
356     p_lead = 100
357     C_lead = ct.tf([1, z_lead], [1, p_lead])
358
359     K = 1000
360
361     ctrl = K * C_lag * C_lead
362     sys_cl = ct.feedback(ctrl * sys, 1)
363
364     t = np.linspace(0, 1, 2000)
365     t, y = ct.step_response(sys_cl, T=t)

```

```

366
367     y_final = y[-1]
368     y_peak = np.max(y)
369     Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
370
371     error = np.abs(y - y_final)
372     threshold = 0.02 * np.abs(y_final)
373     out_of_bounds = np.where(error > threshold)[0]
374     ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0
375
376     print(f"[{mode.upper()}] Integrated Lead-Lag Design Results -> Mp: {Mp:.2f}%, ts: {ts:.4f}s")
377
378     plt.figure(figsize=(10, 6))
379     plt.plot(t, y, linewidth=2, color=colors[3])
380     plt.title(f'Reposta Final (Lead-Lag Integrado)', color='white' if mode=='dark' else 'black')
381     plt.grid(True)
382     plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {ts:.3f}s',
383             bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors[1]), color='white')
384     plt.savefig(os.path.join(assets_dir, '13_step_response_LeadLag.png'))
385 )
386     plt.close()
387
388     plt.figure(figsize=(10, 6))
389     ct.rlocus(ctrl*sys, plot=True, grid=True)
390     plt.title(f'Lugar das Raízes (Lead-Lag)', color='white' if mode=='dark' else 'black')
391     plt.savefig(os.path.join(assets_dir, '17_robustness_LeadLag.png'))
392     plt.close()
393
394     plt.figure(figsize=(10, 8))
395     omega = np.logspace(-3, 3, 1000)
396     mag, phase, omega = ct.frequency_response(ctrl*sys, omega)
397     mag_db = 20 * np.log10(mag)
398     phase_deg = np.degrees(np.unwrap(phase))
399
400     plt.subplot(2, 1, 1)
401     plt.semilogx(omega, mag_db, linewidth=2, color=colors[3])
402     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
403     plt.ylabel('Magnitude (dB)')
404     plt.title('Diagrama de Bode (Lead-Lag)', color='white' if mode=='dark' else 'black')
405
406     plt.subplot(2, 1, 2)
407     plt.semilogx(omega, phase_deg, linewidth=2, color=colors[3])
408     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
409     plt.ylabel('Fase (graus)')
410     plt.xlabel('Frequência (rad/s)')
411
412     plt.tight_layout()
413     plt.savefig(os.path.join(assets_dir, '14_bode_LeadLag.png'))
414     plt.close()
415
416     return ctrl

```

```

417 def design_pid_controller(sys, mode='dark'):
418     """
419     Design and simulate a PID Controller (Ziegler-Nichols).
420     """
421     colors = configure_plot_style(mode)
422     assets_dir = get_assets_dir(mode)
423     grid_color = 'black' if mode == 'light' else 'white'
424     grid_alpha = 0.3
425
426     Kp_pid = 80000
427     Ki_pid = 10000
428     Kd_pid = 500
429
430     # Filter for derivative
431     tau = 0.001
432
433     pid_tf = ct.tf([Kd_pid, Kp_pid, Ki_pid], [tau, 1, 0])
434     sys_cl = ct.feedback(pid_tf * sys, 1)
435
436     t = np.linspace(0, 1.5, 1000)
437     t, y = ct.step_response(sys_cl, T=t)
438
439     y_final = y[-1]
440     y_peak = np.max(y)
441     Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0
442
443     plt.figure(figsize=(10, 6))
444     plt.plot(t, y, linewidth=2, color=colors[3])
445     plt.title(f'Reposta ao Degrau (PID Ziegler-Nichols)', color='white',
446               if mode=='dark' else 'black')
447     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
448     plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%',
449              bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors[1]), color='white')
450     plt.savefig(os.path.join(assets_dir, '15_step_response_PID.png'))
451     plt.close()
452
453     return pid_tf
454
455 def create_plant_variation(Km, am, ae):
456     """
457     Cria variação da planta para análise de robustez (Nicolas).
458     Nominal: Km=1.1, am=13.2, ae=950 -> K_sys=772 fixo.
459     Using user parameters for K_sys check.
460     """
461
462     # Note: user defined K_sys implicitly via 1.2 * 772?
463     # Original code had K_sys=772. Let's keep the robustness logic
464     # consistent
465     # but acknowledge the new nominal plant is slightly different.
466     K_sys = 772
467     num = [Km * K_sys]
468     den = [1, (am + ae), (am * ae), 0]
469     return ct.tf(num, den)
470
471 def analyze_robustness(controllers_dict, mode='dark'):
472     """
473     Análise de Robustez baseada nos cenários do Nicolas.
474     """

```

```

472     scenarios = {
473         "Nominal": {"Km": 1.1, "am": 13.2, "ae": 950, "style": "-", "color_dark": "#00ff00", "color_light": "green"}, 
474         "Pesado": {"Km": 0.8, "am": 15.0, "ae": 1100, "style": "--", "color_dark": "#00bfff", "color_light": "blue"}, 
475         "Agressivo": {"Km": 1.2, "am": 10.0, "ae": 800, "style": "-.", "color_dark": "#ff4500", "color_light": "red"} 
476     } 
477 
478     colors = configure_plot_style(mode) 
479     assets_dir = get_assets_dir(mode) 
480 
481     if mode == 'light': 
482         text_color = 'black' 
483         grid_color = 'black' 
484         face_color = 'white' 
485         grid_alpha = 0.3 
486     else: 
487         text_color = 'white' 
488         grid_color = 'white' 
489         face_color = 'black' 
490         grid_alpha = 0.3 
491 
492     for ctrl_name, ctrl in controllers_dict.items(): 
493         plt.figure(figsize=(10, 6)) 
494         print(f"[{mode.upper()}] Analisando Robustez: {ctrl_name}") 
495 
496         for name, params in scenarios.items(): 
497             G_var = create_plant_variation(params["Km"], params["am"], params["ae"]) 
498             sys_cl = ct.feedback(ctrl * G_var, 1) 
499 
500             t, y = ct.step_response(sys_cl, T=np.linspace(0, 2.0, 1000)) 
501 
502             color = params["color_dark"] if mode == 'dark' else params["color_light"] 
503 
504             y_peak = np.max(y) 
505             mp = (y_peak - 1) * 100 
506 
507             plt.plot(t, y, linestyle=params["style"], linewidth=2, label=f"{name} (Mp={mp:.1f}%)", color=color) 
508 
509             plt.axhline(1.0, color=text_color, linestyle=':', linewidth=0.8, alpha=0.5) 
510             plt.grid(True, which='both', linestyle='--', linewidth=0.5, color=grid_color, alpha=grid_alpha) 
511 
512             plt.title(f'Análise de Robustez - {ctrl_name}', color=text_color, fontsize=14) 
513             plt.xlabel('Tempo (s)', color=text_color, fontsize=12) 
514             plt.ylabel('Amplitude', color=text_color, fontsize=12) 
515 
516             legend = plt.legend(facecolor=face_color, edgecolor=text_color) 
517             for text in legend.get_texts(): 
518                 text.set_color(text_color) 
519 
520             plt.tick_params(colors=text_color, which='both') 

```

```

521     for spine in plt.gca().spines.values():
522         spine.set_color(text_color)
523
524     if ctrl_name == 'PID':
525         fname = '16_robustness_PID.png'
526     elif ctrl_name == 'Lead-Lag':
527         fname = '17_robustness_LeadLag.png'
528     else:
529         fname = f'robustness_{ctrl_name}.png'
530
531     plt.savefig(os.path.join(assets_dir, fname))
532     plt.close()
533
534 if __name__ == "__main__":
535     # --- Override System with Dierson's Parameters globally ---
536     # We define it here to pass to controllers, though comparison
537     # function creates its own instance to be safe
538     s = ct.TransferFunction.s
539     km = 1.2
540     am = 13.2
541     ae = 950
542     sys = km/(s*(s+am)*(s+ae))
543
544     if not os.path.exists('../assets/images'): os.makedirs('../assets/
545     images')
546     if not os.path.exists('../assets/report_images'): os.makedirs('../
547     assets/report_images')
548
549     analyze_open_loop(sys, mode='dark')
550     analyze_open_loop(sys, mode='light')
551
552     controllers_to_test = {}
553
554     print("\n--- Running Control Simulation in DARK mode ---")
555     configure_plot_style('dark')
556     Kp_p = design_p_controller(sys, mode='dark')
557     ctrl_lag = design_lag_controller(sys, mode='dark')
558     ctrl_lead = design_lead_controller(sys, mode='dark')
559     ctrl_leadlag = design_lead_lag_controller(sys, mode='dark')
560     ctrl_pid = design_pid_controller(sys, mode='dark')
561
562     controllers_to_test = {
563         'Lead': ctrl_lead,
564         'Lag': ctrl_lag,
565         'Lead-Lag': ctrl_leadlag,
566         'PID': ctrl_pid
567     }
568
569     print("\n--- Running Control Simulation in LIGHT mode (Prioritizing
570     Report Assets) ---")
571     configure_plot_style('light')
572     Kp_p = design_p_controller(sys, mode='light')
573     ctrl_lag = design_lag_controller(sys, mode='light')
574     design_lead_controller(sys, mode='light')
575     design_lead_lag_controller(sys, mode='dark')
576     design_pid_controller(sys, mode='dark')
577
578     # Generate comparative plots with Dierson's strict parameters

```

```
575 generate_comparative_plots(sys, Kp_p, ctrl_lag, mode='dark')
576 generate_comparative_plots(sys, Kp_p, ctrl_lag, mode='light')
577
578 analyze_robustness(controllers_to_test, mode='dark')
579 analyze_robustness(controllers_to_test, mode='light')
580
581 print("\nTodas as simulações e gráficos foram atualizados.")
```

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