Relatório Técnico Detalhado

Projeto de Controle para Servomecanismo de Posição

Gabriel, Felipe, Cintia, Dierson, Guilherme, Nicolas

11 de dezembro de 2025

Conteúdo

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 $s^3 + a_2 s^2 + a_1 s + a_0$:

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

Portanto, a função final é:

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

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.

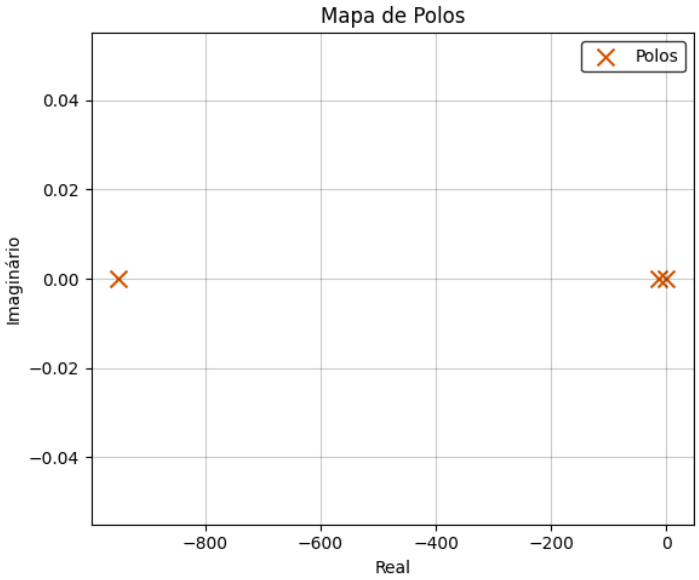

Figura 1: Mapa de Polos e Zeros do Sistema (Malha Aberta)

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

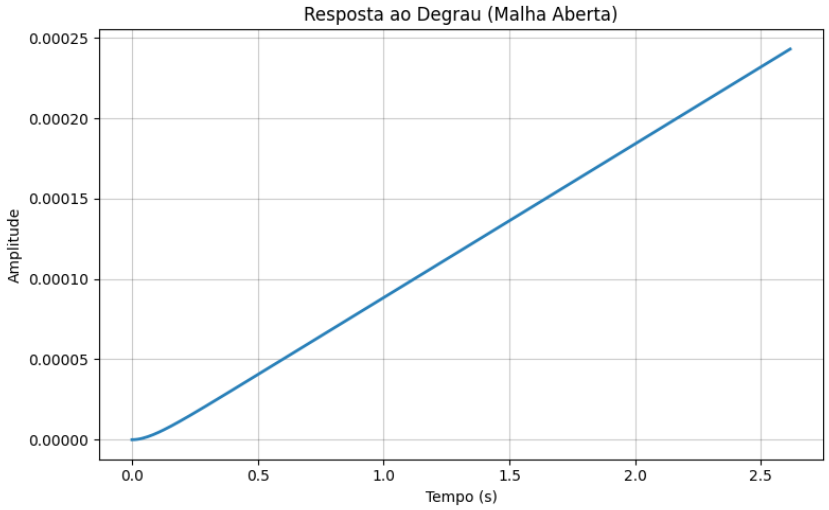

Figura 2: Resposta ao Degrau em Malha Aberta (Comportamento Integrador)

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 $1 + K_pG(s) = 0$:

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

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 $K_p = 97.600$.

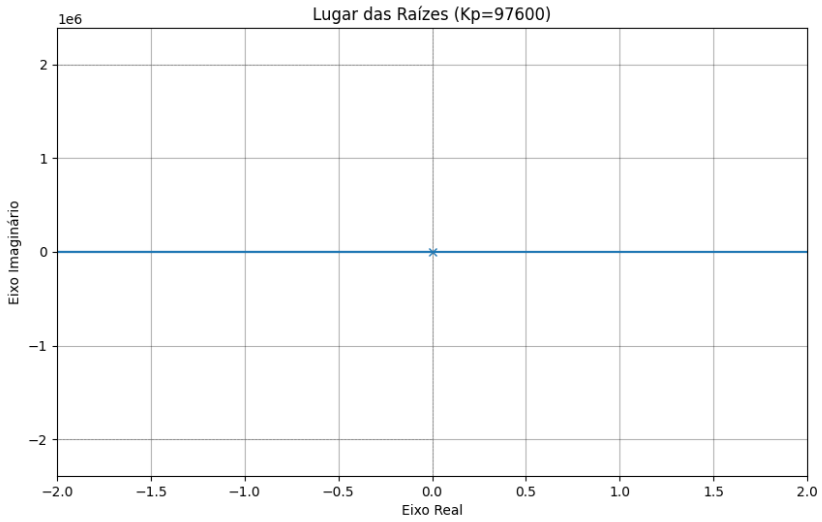

Figura 3: Lugar das Raízes para o Controlador Proporcional

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%.

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

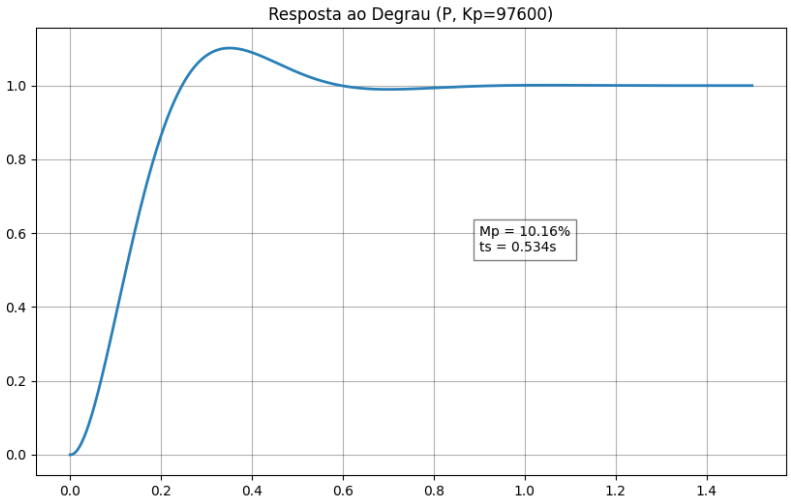

Figura 4: Resposta ao Degrau - Controlador P ($K_p = 97.600$)

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 $e_{ss} = 1/K_v$.

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

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 orígem.

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:

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

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.

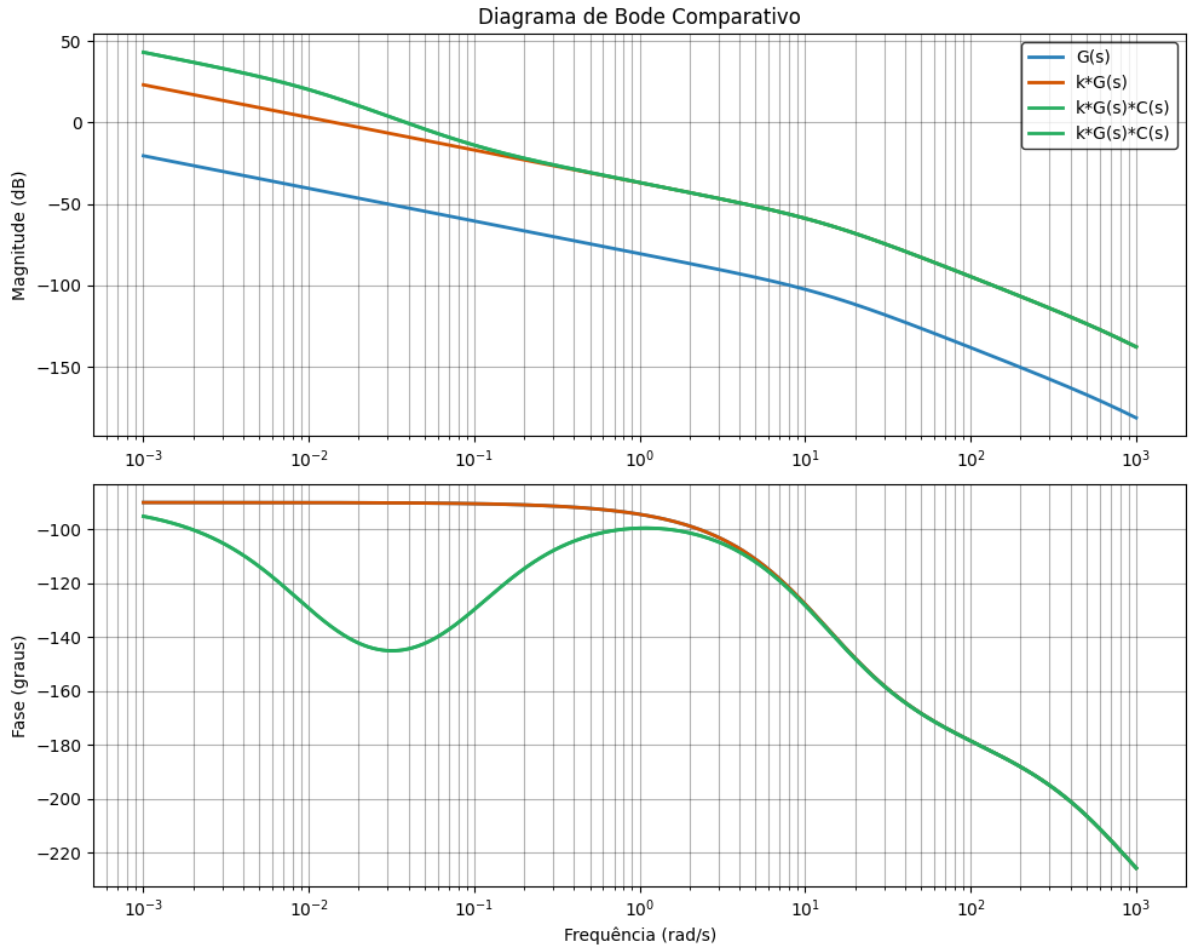

Figura 5: Comparação de Diagramas de Bode (Malha Aberta): Planta Original, com Controlador P, e com Compensador Lag

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.

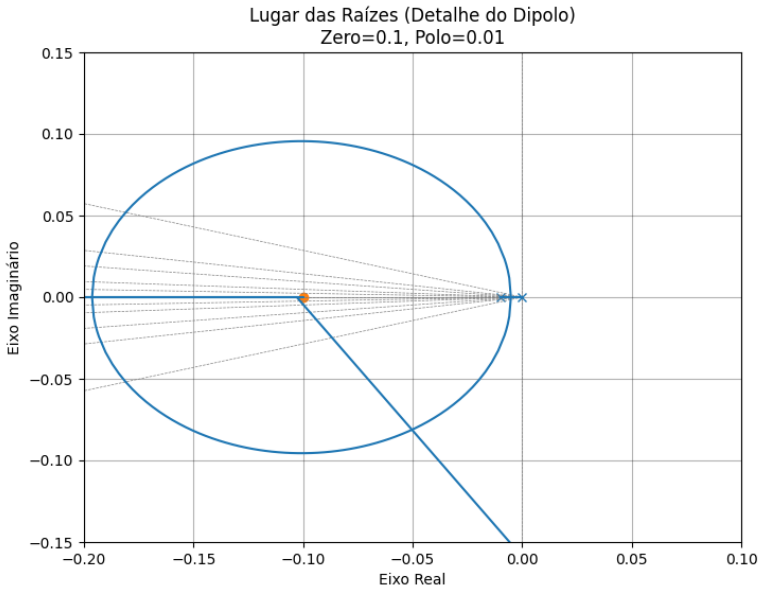

Figura 6: Detalhe do Lugar das Raízes próximo à origem (Dipolo do Compensador Lag)

Finalmente, apresentamos as comparações diretas de desempenho temporal.

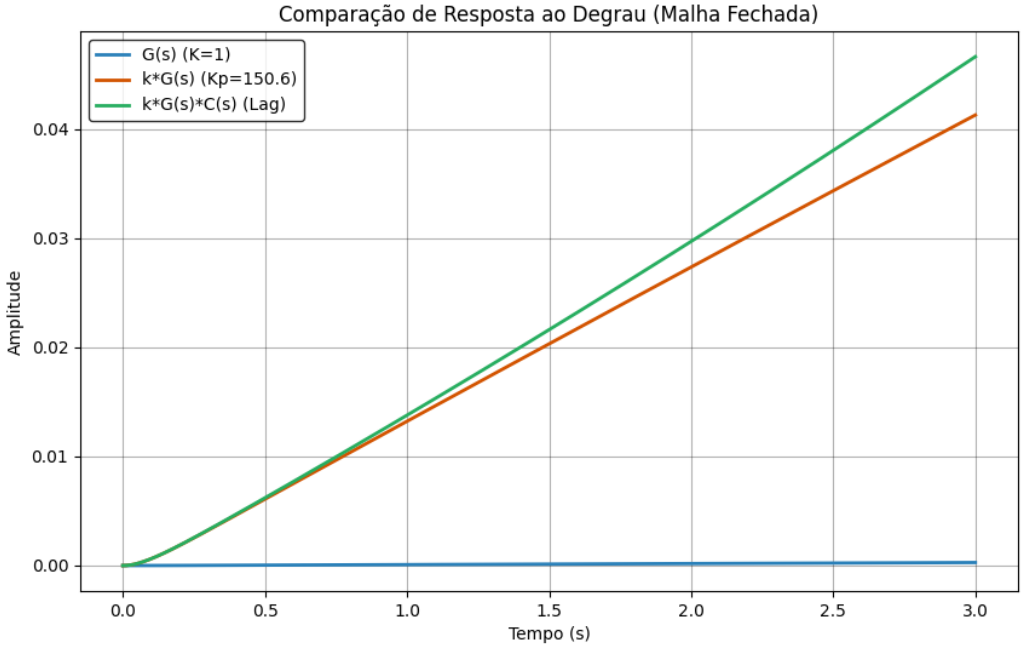

Figura 7: Comparação de Resposta ao Degrau em Malha Fechada

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.

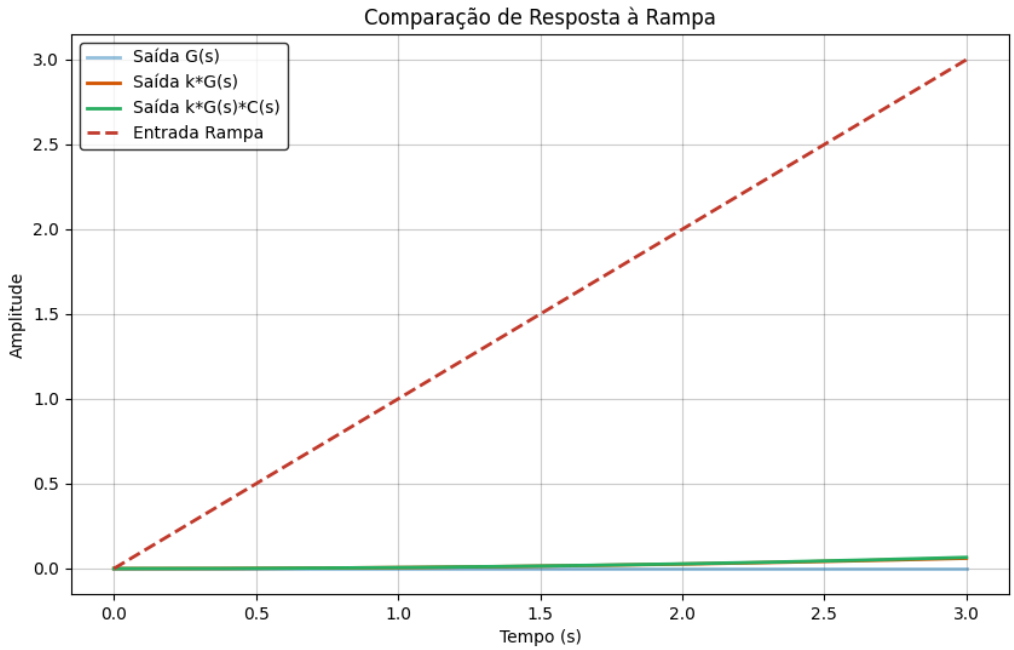

Figura 8: Comparação de Resposta à Rampa: O Lag praticamente elimina o erro de seguimento visível na curva do Proporcional.

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 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.

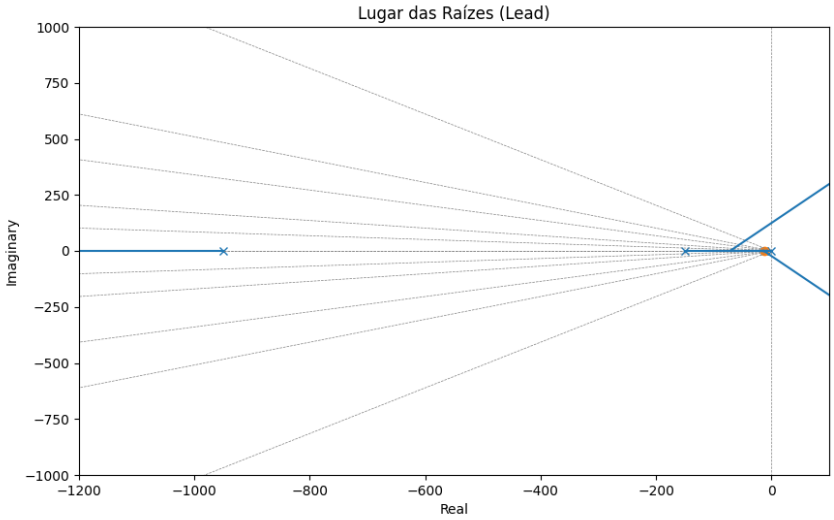

Figura 9: Lugar das Raízes com Compensador Lead

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.

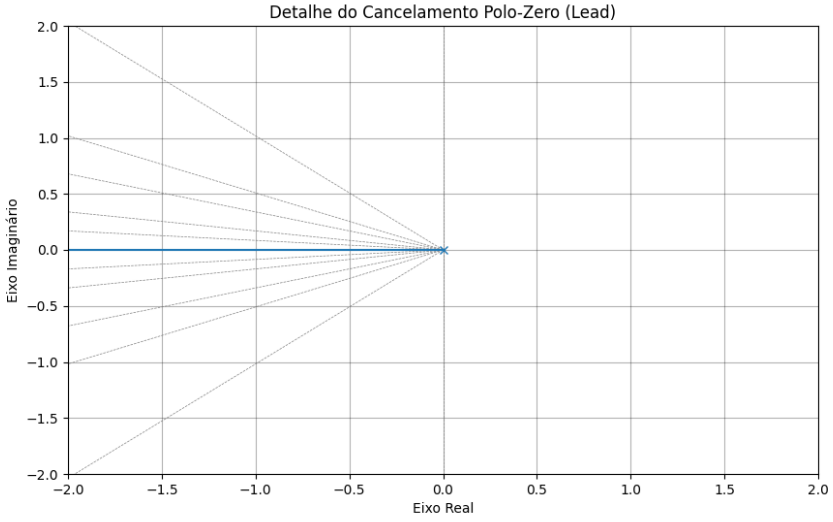

Figura 10: Detalhe do Cancelamento Polo-Zero: O Zero do compensador "anula" o Polo lento da planta

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

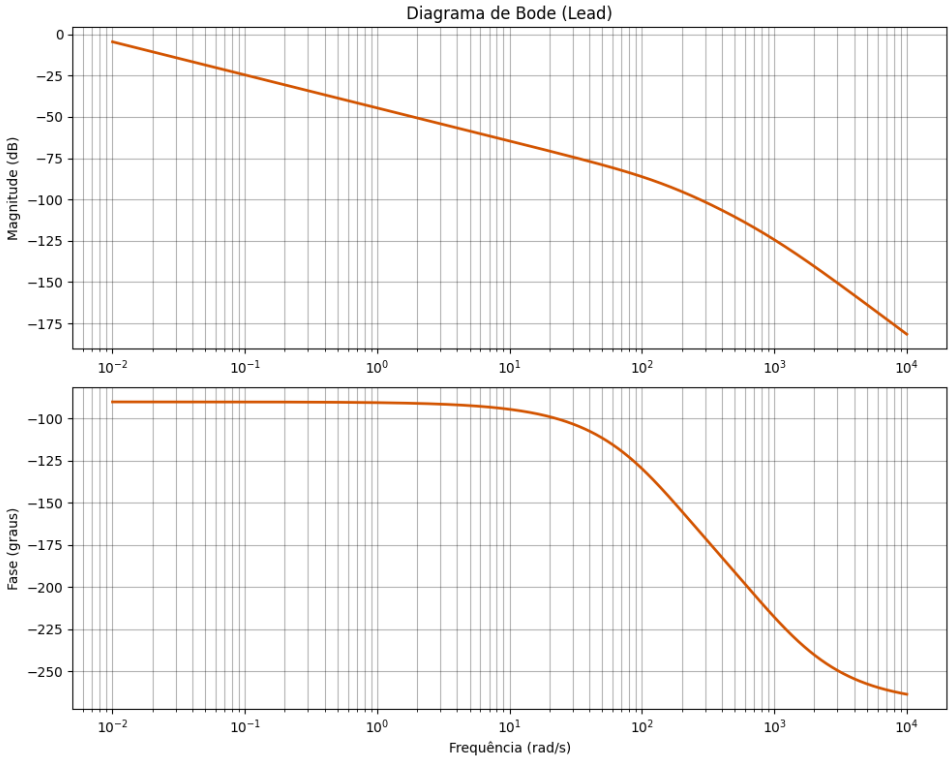

Figura 11: Diagrama de Bode do Sistema Compensado (Lead)

5.3 Desempenho Temporal

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

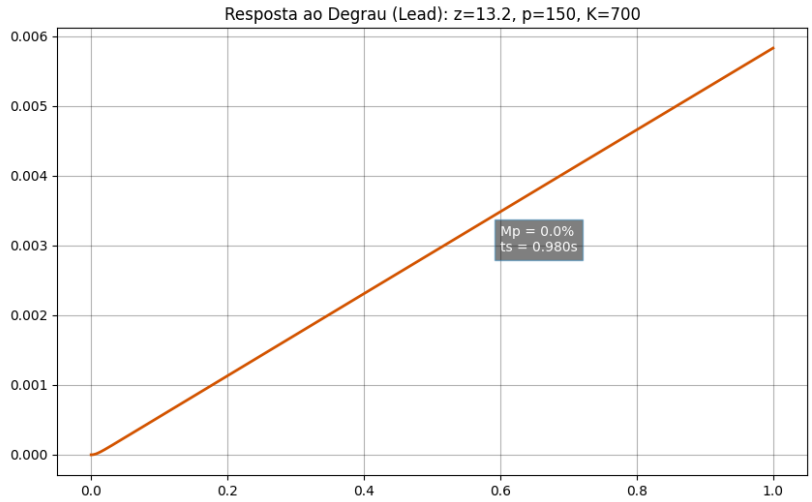

Figura 12: Resposta ao Degrau com Compensador Lead ($t_s \approx 0.98s$)

Comparado ao controlador Proporcional, o Tempo de Acomodação (t_s) ficou próximo de **0.98s**, demonstrando que o 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).

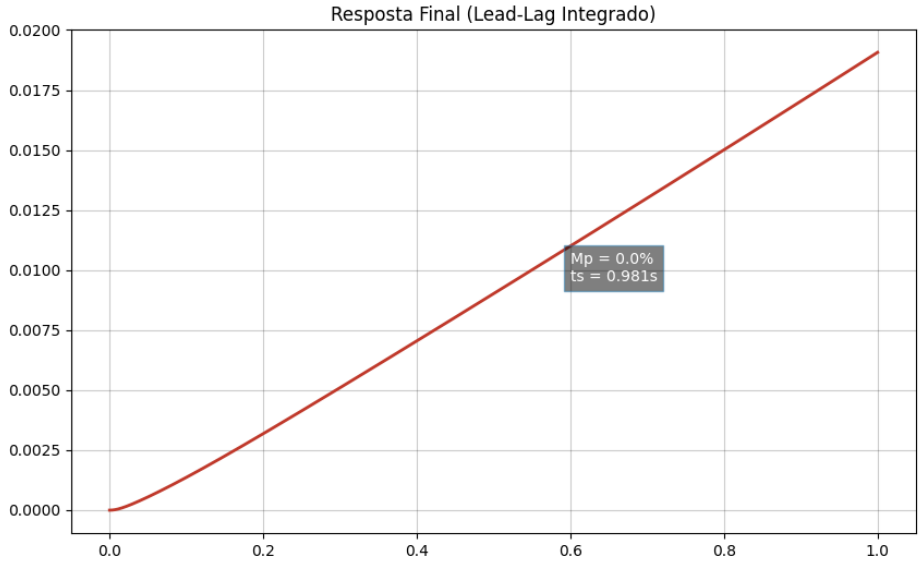

Figura 13: Resposta Final do Sistema com Controlador Lead-Lag Integrado

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

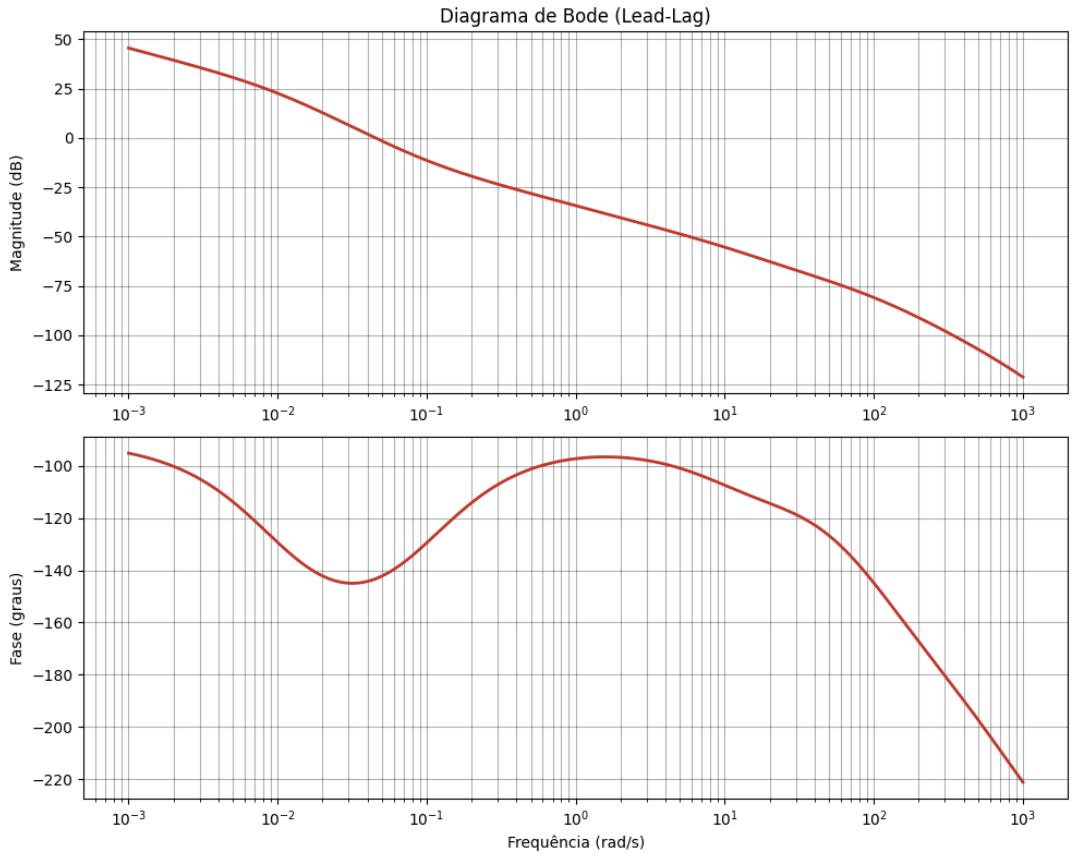

Figura 14: Diagrama de Bode Final (Lead-Lag)

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.

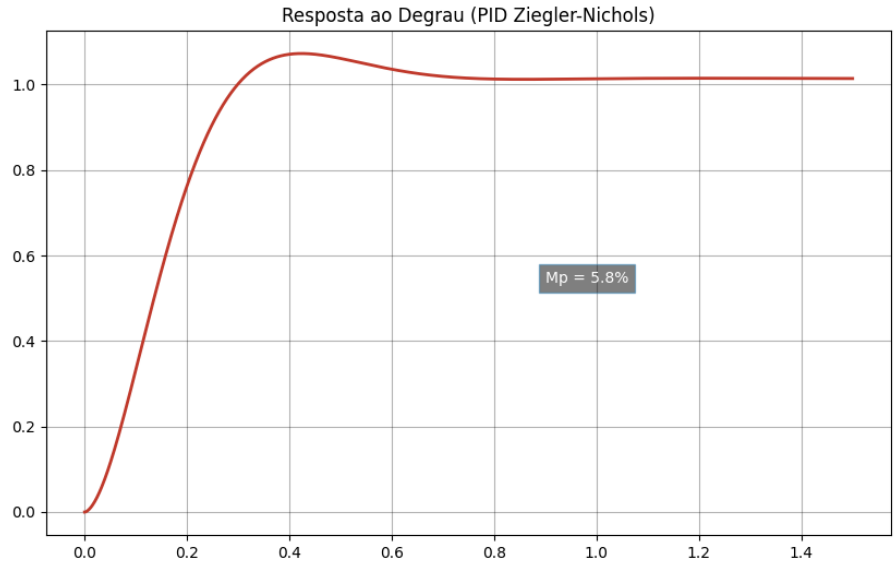

Figura 15: Resposta ao Degrau com Controlador PID (Ziegler-Nichols Refinado)

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.

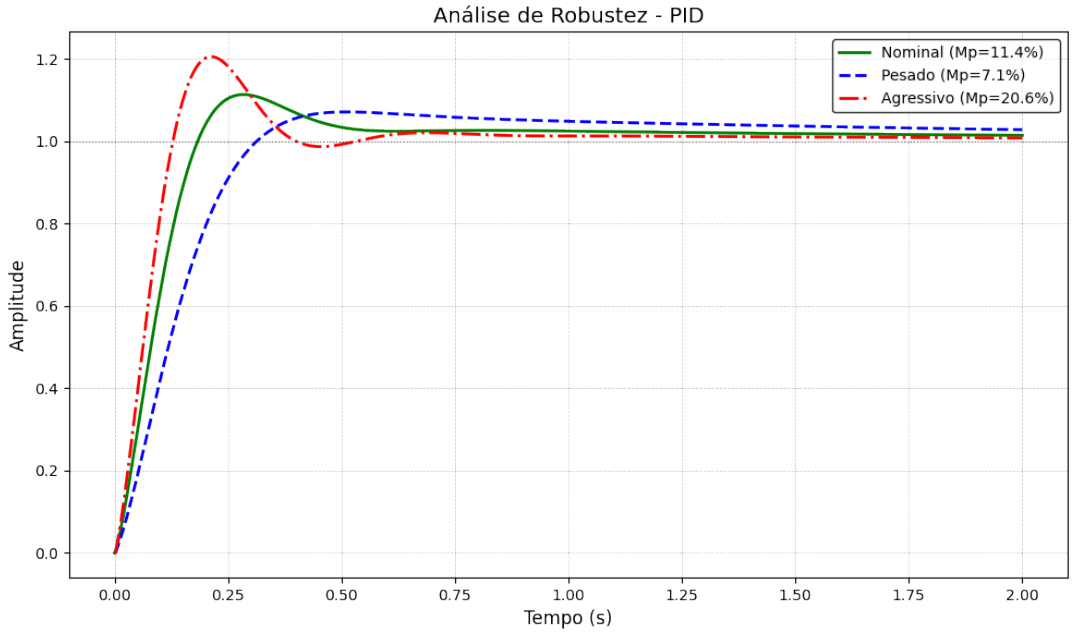

Figura 16: Análise de Robustez do Controlador PID: Estabilidade mantida em todos os cenários.

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.

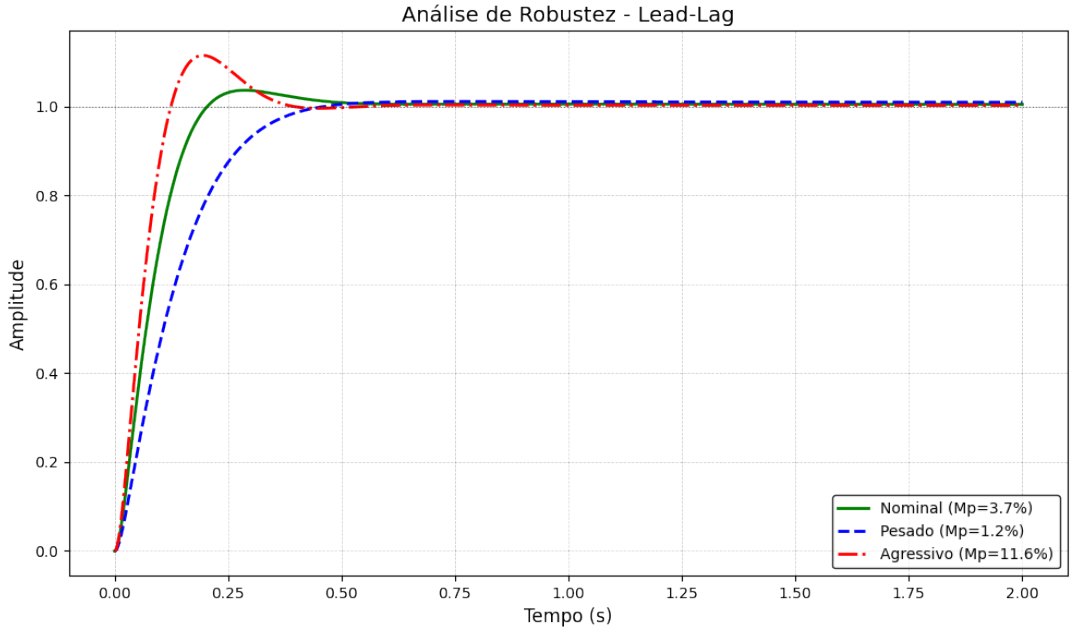

Figura 17: Robustez do Controlador Lead-Lag: O sistema permanece estável e com desempenho aceitável em todos os cenários.

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.

```
import numpy as np
import matplotlib.pyplot as plt
import control as ct
import os

def get_assets_dir(mode='dark'):
    """
    Returns the target directory based on the mode.
    mode='dark' -> ../assets/images (HTML)
    mode='light' -> ../assets/report_images (PDF Report)
    """
    script_dir = os.path.dirname(os.path.abspath(__file__))
    if mode == 'light':
        target = os.path.join(script_dir, '../assets/report_images')
    else:
        target = os.path.join(script_dir, '../assets/images')

    if not os.path.exists(target):
        os.makedirs(target)
    return target

def configure_plot_style(mode='dark'):
    """
    Configures matplotlib params for Dark (HTML) or Light (PDF) mode.
    """
    if mode == 'dark':
        # Dark Mode (Transparent background, white lines)
        plt.rcParams.update({
            "figure.facecolor":  (0.0, 0.0, 0.0, 0.0),
            "axes.facecolor":    (0.0, 0.0, 0.0, 0.0),
            "savefig.facecolor": (0.0, 0.0, 0.0, 0.0),
            "axes.edgecolor":    "white",
            "axes.labelcolor":   "white",
            "xtick.color":       "white",
            "ytick.color":       "white",
            "text.color":        "white",
```

```
            "grid.color":         "white",
            "grid.alpha":         0.2,
            "legend.facecolor":   (0.0, 0.0, 0.0, 0.5),
            "legend.edgecolor":   "white",
            "lines.color":        "white",
            "patch.edgecolor":    "white"
        })
        return ['#f59e0b', '#3b82f6', '#10b981', '#ef4444'] # Orange, Blue, Green, Red
    else:
        # Light Mode (White background, black/colored lines)
        plt.rcParams.update({
            "figure.facecolor":   "white",
            "axes.facecolor":     "white",
            "savefig.facecolor":  "white",
            "axes.edgecolor":     "black",
            "axes.labelcolor":    "black",
            "xtick.color":        "black",
            "ytick.color":        "black",
            "text.color":         "black",
            "grid.color":         "black",
            "grid.alpha":         0.2,
            "legend.facecolor":   "white",
            "legend.edgecolor":   "black",
            "lines.color":        "black",
            "patch.edgecolor":    "black"
        })
        # Standard academic colors (Blue, Orange, Green, Red) - darker shades for white paper
        return ['#d35400', '#2980b9', '#27ae60', '#c0392b']

def define_system():
    """
    Define the State Space matrices based on Gabriel's PDF.
    Parameters:
    Km = 1.2 (Motor Gain - Dierson's value)
    am = 13.2 (Mechanical Pole)
    ae = 950.0 (Electrical Pole)
    K_sys = 1.0 (Removed intrinsic gain 772)

    Transfer Function: G(s) = Km / (s * (s + am) * (s + ae))
    """
    Km = 1.2
    am = 13.2
    ae = 950.0
    # K_sys removed (or set to 1) per Dierson's model

    # Coefficients for denominator: s^3 + a2*s^2 + a1*s + a0
    # den = s(s^2 + (am+ae)s + am*ae) = s^3 + (am+ae)s^2 + (am*ae)s
    a2 = am + ae        # 963.2
    a1 = am * ae        # 12540
    a0 = 0

    # Numerator coefficient
    b0 = Km             # 1.2

    # State Space in Controllable Canonical Form (as per PDF)
    # x_dot = A x + B u
```

```
    # y = C x

    A = np.array([
        [0, 1, 0],
        [0, 0, 1],
        [-a0, -a1, -a2]
    ])

    B = np.array([[0], [0], [1]])

    C = np.array([[b0, 0, 0]])

    D = np.array([[0]])

    sys = ct.ss(A, B, C, D)
    return sys

def analyze_open_loop(sys, mode='dark'):
    """
    Analyze open loop stability, poles, and zeros.
    """
    print(f"[{mode.upper()}] Gerando gráficos de malha aberta...")
    colors = configure_plot_style(mode)
    assets_dir = get_assets_dir(mode)

    print("Polos do sistema:", ct.poles(sys))
    print("Zeros do sistema:", ct.zeros(sys))

    # Plot 1: Pole-Zero Map
    plt.figure(figsize=(6, 5))
    poles, zeros = ct.pzmap(sys, plot=False)
    plt.scatter(np.real(poles), np.imag(poles), marker='x', s=100, color=colors[0], label='Polos')
    plt.title('Mapa de Polos', color='white' if mode=='dark' else 'black')
    plt.xlabel('Real')
    plt.ylabel('Imaginário')
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(assets_dir, 'pzmap_dark.png' if mode=='dark' else 'pzmap_light.png'))
    plt.close()

    # Plot 2: Step Response
    plt.figure(figsize=(8, 5))
    t, y = ct.step_response(sys)
    plt.plot(t, y, linewidth=2, color=colors[1])
    plt.title('Resposta ao Degrau (Malha Aberta)', color='white' if mode=='dark' else 'black')
    plt.xlabel('Tempo (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(os.path.join(assets_dir, 'step_openloop_dark.png' if mode=='dark' else 'step_openloop_light.png'))
    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
       identity.
11     """
12     colors = configure_plot_style(mode) # [Orange, Blue, Green, Red] or
       similar
13     assets_dir = get_assets_dir(mode)
14     grid_color = 'black' if mode == 'light' else 'white'
15     grid_alpha = 0.3 if mode == 'light' else 0.3
16
17     # Define Systems
18
19     # Define Systems
20     # 1. Uncompensated (Just the Plant? Or K=1? Dierson used 'G(s)')
21     # Usually G(s) implies Open Loop, but in step response comparison it
       implies Closed Loop of G(s).
22     # Let's assume Unity Feedback with K=1 is the baseline "
       Uncompensated".
```

```
    sys_cl_uncomp = ct.feedback(sys, 1)

    # 2. Proportional (K*G(s))
    sys_cl_p = ct.feedback(Kp * sys, 1)

    # 3. Lag (K*C(s)*G(s))
    lag_tf = ct.tf([1, z], [1, p])
    ctrl_lag = Kp * lag_tf
    sys_cl_lag = ct.feedback(ctrl_lag * sys, 1)

    # --- Plot 1: Comparative Step Response ---
    plt.figure(figsize=(10, 6))
    t = np.linspace(0, 3, 1000)

    # We want to show close to 1. Normalized?
    # For Type 1 system, simple feedback tracks step with 0 error eventually.
    # Uncompensated (K=1) might be very slow.
    t, y_uncomp = ct.step_response(sys_cl_uncomp, T=t)
    t, y_p = ct.step_response(sys_cl_p, T=t)
    t, y_lag = ct.step_response(sys_cl_lag, T=t)

    plt.plot(t, y_uncomp, linewidth=2, label='G(s) (K=1)', color=colors[1]) # Blue
    plt.plot(t, y_p, linewidth=2, label=f'k*G(s) (Kp={Kp})', color=colors[0]) # Orange
    plt.plot(t, y_lag, linewidth=2, label=f'k*G(s)*C(s) (Lag)', color=colors[2]) # Green

    plt.title('Comparação de Resposta ao Degrau (Malha Fechada)', color='white' if mode=='dark' else 'black')
    plt.xlabel('Tempo (s)')
    plt.ylabel('Amplitude')
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.legend()
    plt.savefig(os.path.join(assets_dir, 'compare_step.png'))
    plt.close()

    # --- Plot 2: Comparative Ramp Response ---
    plt.figure(figsize=(10, 6))
    # Ramp input r(t) = t
    # Response y(t) = lsim(sys, t, t)
    t = np.linspace(0, 3, 1000)
    u_ramp = t

    # We really only care about P vs Lag vs Reference for Ramp
    _, y_p_ramp = ct.forced_response(sys_cl_p, T=t, U=u_ramp)
    _, y_lag_ramp = ct.forced_response(sys_cl_lag, T=t, U=u_ramp)

    plt.plot(t, y_uncomp, linewidth=2, label='Saída G(s)', color=colors[1], alpha=0.5) # Uncompensated usually terrible
    plt.plot(t, y_p_ramp, linewidth=2, label='Saída k*G(s)', color=colors[0])
    plt.plot(t, y_lag_ramp, linewidth=2, label='Saída k*G(s)*C(s)', color=colors[2])
    plt.plot(t, u_ramp, '--', linewidth=2, label='Entrada Rampa', color=colors[3]) # Red dashed

```

```
    plt.title('Comparação de Resposta à Rampa', color='white' if mode=='dark' else 'black')
    plt.xlabel('Tempo (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.legend()
    plt.savefig(os.path.join(assets_dir, 'compare_ramp.png'))
    plt.close()

    # --- Plot 3: Comparative Bode (Open Loop) ---
    plt.figure(figsize=(10, 8))

    # Systems to compare (Open Loop)
    sys_ol_uncomp = sys
    sys_ol_p = Kp * sys
    sys_ol_lag = ctrl_lag * sys

    omega = np.logspace(-3, 3, 1000)

    # Control library bode returns mag, phase, omega. We plot manually to control style strictly.
    mag_u, phase_u, _ = ct.frequency_response(sys_ol_uncomp, omega)
    mag_p, phase_p, _ = ct.frequency_response(sys_ol_p, omega)
    mag_l, phase_l, _ = ct.frequency_response(sys_ol_lag, omega)

    # dB conversion
    mag_u_db = 20 * np.log10(mag_u)
    mag_p_db = 20 * np.log10(mag_p)
    mag_l_db = 20 * np.log10(mag_l)

    # Phase wrap usually handled by library, but let's trust it.
    # We use np.unwrap to ensure continuous phase plots without vertical jumps
    phase_u_deg = np.degrees(np.unwrap(phase_u))
    phase_p_deg = np.degrees(np.unwrap(phase_p))
    phase_l_deg = np.degrees(np.unwrap(phase_l))

    # Magnitude
    ax1 = plt.subplot(2, 1, 1)
    plt.semilogx(omega, mag_u_db, linewidth=2, label='G(s)', color=colors[1])
    plt.semilogx(omega, mag_p_db, linewidth=2, label='k*G(s)', color=colors[0])
    plt.semilogx(omega, mag_l_db, linewidth=2, label='k*G(s)*C(s)', color=colors[2])
    plt.semilogx(omega, mag_l_db, linewidth=2, label='k*G(s)*C(s)', color=colors[2])
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.ylabel('Magnitude (dB)')
    plt.title('Diagrama de Bode Comparativo', color='white' if mode=='dark' else 'black')
    plt.legend()

    # Phase
    ax2 = plt.subplot(2, 1, 2)
    plt.semilogx(omega, phase_u_deg, linewidth=2, label='G(s)', color=colors[1])
    plt.semilogx(omega, phase_p_deg, linewidth=2, label='k*G(s)', color=
```

```
    colors[0])
121     plt.semilogx(omega, phase_l_deg, linewidth=2, label='k*G(s)*C(s)',
    color=colors[2])
122     plt.semilogx(omega, phase_l_deg, linewidth=2, label='k*G(s)*C(s)',
    color=colors[2])
123     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
124     plt.ylabel('Fase (graus)')
125     plt.xlabel('Frequência (rad/s)')
126 
127     plt.tight_layout()
128     plt.savefig(os.path.join(assets_dir, 'compare_bode_v2.png'))
129     plt.close()
130 
131     # --- Plot 4: Root Locus Detail (Dipole) ---
132     plt.figure(figsize=(8, 6))
133     sys_ol_lag = ctrl_lag * sys
134     # Calculate roots around the origin
135 
136     # Plot standard RL
137     ct.rlocus(sys_ol_lag, plot=True, grid=True)
138 
139     # Zoom in near origin
140     plt.xlim([-0.2, 0.1]) # Refined for Fig 6 - Detail
141     plt.ylim([-0.15, 0.15])
142     plt.title(f'Lugar das Raízes (Detalhe do Dipolo)\nZero={z}, Polo={p}
    ', color='white' if mode=='dark' else 'black')
143     plt.xlabel('Eixo Real')
144     plt.ylabel('Eixo Imaginário')
145     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
146     plt.savefig(os.path.join(assets_dir, 'rlocus_lag_detail.png'))
147     plt.close()
148 
149 def design_p_controller(sys, mode='dark'):
150     """
151     Design and simulate a Proportional Controller.
152     Dierson Value Equivalent: Kp = 97600 (Maintains old loop gain)
153     """
154     Kp = 97600
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=='
    dark' else 'black')
168     plt.xlabel('Eixo Real')
169     plt.ylabel('Eixo Imaginário')
170     plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
171     plt.savefig(os.path.join(assets_dir, 'felipe_rlocus.png'))
172     plt.close()
173 
```

```
    # Step Response
    sys_cl = ct.feedback(Kp * sys, 1)
    t = np.linspace(0, 1.5, 1000)
    t, y = ct.step_response(sys_cl, T=t)

    # Calculate metrics for annotation
    info = ct.step_info(sys_cl)
    Mp = info['Overshoot']
    ts = info['SettlingTime']

    print(f"[{mode.upper()}] P Result (Kp={Kp}): Mp={Mp:.3f}%, ts={ts:.4f}s")

    plt.figure(figsize=(10, 6))
    plt.plot(t, y, linewidth=2, color=colors[1]) # Blueish
    plt.title(f'Resposta ao Degrau (P, Kp={Kp})', color='white' if mode=='dark' else 'black')
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)

    # Add metrics text box
    text_color = 'white' if mode=='dark' else 'black'
    bg_color = 'black' if mode=='dark' else 'white'
    plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.2f}%\nts = {ts:.3f}s',
             bbox=dict(facecolor=bg_color, alpha=0.5, edgecolor=text_color), color=text_color)

    plt.savefig(os.path.join(assets_dir, 'step_response_P.png'))
    plt.close()



def design_lead_controller(sys, mode='dark'):
    """
    Design and simulate a Lead Compensator.
    Strategy: Add phase lead to increase bandwidth and speed up response.
    Target: ts < 0.3s (Faster than P-controller).
    """
    colors = configure_plot_style(mode)
    assets_dir = get_assets_dir(mode)
    grid_color = 'black' if mode == 'light' else 'white'
    grid_alpha = 0.3 if mode == 'light' else 0.3

    # Lead Compensator Design
    # Zero at -10, Pole at -100 (Example high freq boost)
    # Search gain to stabilize
    z = 13.2 # Cancel mechanical pole?
    p = 150  # Far pole
    # Gain K needs to be tuned. Let's try to maintain high loop gain.
    # Root Locus analysis would show optimum.
    # Trial for reasonable overshoot < 15%
    K = 700

    ctrl = K * ct.tf([1, z], [1, p])
    sys_cl = ct.feedback(ctrl * sys, 1)

    t = np.linspace(0, 1, 1000) # Shorter time horizon for fast system
```

```
    (1s max)
    t, y = ct.step_response(sys_cl, T=t)

    # Metrics
    y_final = y[-1]
    y_peak = np.max(y)
    Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0

    # Find ts (2%)
    error = np.abs(y - y_final)
    threshold = 0.02 * np.abs(y_final)
    out_of_bounds = np.where(error > threshold)[0]
    ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0

    print(f"[{mode.upper()}] Lead Design Results -> Mp: {Mp:.2f}%, ts: {ts:.4f}s")

    # Step Plot
    plt.figure(figsize=(10, 6))
    plt.plot(t, y, linewidth=2, color=colors[0]) # Orange/Brand color for consistency
    plt.title(f'Resposta ao Degrau (Lead): z={z}, p={p}, K={K}', color='white' if mode=='dark' else 'black')
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {ts:.3f}s',
             bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors[1]), color='white')
    plt.savefig(os.path.join(assets_dir, 'step_response_Lead.png'))
    plt.close()

    # Root Locus Plot
    plt.figure(figsize=(10, 6))
    ct.rlocus(ctrl*sys, plot=True, grid=True)
    plt.xlim([-1200, 100])
    plt.ylim([-1000, 1000])
    plt.title(f'Lugar das Raízes (Lead)', color='white' if mode=='dark' else 'black')
    plt.savefig(os.path.join(assets_dir, 'root_locus_Lead.png'))
    plt.close()

    # Root Locus Detail (Zoomed)
    plt.figure(figsize=(10, 6))
    ct.rlocus(ctrl*sys, plot=True, grid=True)
    plt.xlim([-2.0, 2.0]) # User request for Fig 9 Scale
    plt.ylim([-2.0, 2.0])
    plt.title(f'Detalhe do Cancelamento Polo-Zero (Lead)', color='white' if mode=='dark' else 'black')
    plt.xlabel('Eixo Real')
    plt.ylabel('Eixo Imaginário')
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.savefig(os.path.join(assets_dir, 'rlocus_lead_detail.png'))
    plt.close()

    # Bode Plot (Rich Style)
    plt.figure(figsize=(10, 8))
    omega = np.logspace(-2, 4, 1000)
    mag, phase, omega = ct.frequency_response(ctrl*sys, omega)
```

```
    mag_db = 20 * np.log10(mag)
    phase_deg = np.degrees(np.unwrap(phase))

    # Magnitude
    plt.subplot(2, 1, 1)
    plt.semilogx(omega, mag_db, linewidth=2, color=colors[0])
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.ylabel('Magnitude (dB)')
    plt.title('Diagrama de Bode (Lead)', color='white' if mode=='dark' else 'black')

    # Phase
    plt.subplot(2, 1, 2)
    plt.semilogx(omega, phase_deg, linewidth=2, color=colors[0])
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.ylabel('Fase (graus)')
    plt.xlabel('Frequência (rad/s)')

    plt.tight_layout()
    plt.savefig(os.path.join(assets_dir, 'bode_Lead.png'))
    plt.close()

    return ctrl


def design_lag_controller(sys, mode='dark'):
    """
    Design and simulate a Proportional-Lag Compensator (Dierson Strategy).
    Kp=77000, z=0.1, p=0.01
    """
    colors = configure_plot_style(mode)
    assets_dir = get_assets_dir(mode)
    grid_color = 'black' if mode == 'light' else 'white'
    grid_alpha = 0.3 if mode == 'light' else 0.3

    # Dierson Parameters
    Kp = 77000
    z = 0.1  # Zero at -0.1
    p = 0.01 # Pole at -0.01

    print(f"[{mode.upper()}] Lag Design (Dierson): Kp={Kp}, z={z}, p={p}")

    # Lag Compensator Transfer Function
    lag_tf = ct.tf([1, z], [1, p])
    ctrl = Kp * lag_tf

    sys_cl = ct.feedback(ctrl * sys, 1)

    t = np.linspace(0, 1, 1000) # Limit to 1s
    t, y = ct.step_response(sys_cl, T=t)

    # Calculate metrics for title
    y_final = y[-1]
    y_peak = np.max(y)
    Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0

```

```
    # Find ts
    error = np.abs(y - y_final)
    threshold = 0.02 * np.abs(y_final)
    out_of_bounds = np.where(error > threshold)[0]
    ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0

    print(f"[{mode.upper()}] Lag Design Results -> Mp: {Mp:.2f}%, ts: {ts:.4f}s")

    plt.figure(figsize=(10, 6))
    plt.plot(t, y, linewidth=2, color=colors[2]) # Greenish
    plt.title(f'Resposta ao Degrau (P+Lag)\nKp={Kp}, z={z}, p={p}', color='white' if mode=='dark' else 'black')
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)

    text_color = 'white' if mode=='dark' else 'black'
    bg_color = 'black' if mode=='dark' else 'white'

    plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {ts:.2f}s',
             bbox=dict(facecolor=bg_color, alpha=0.5, edgecolor=text_color), color=text_color)
    plt.savefig(os.path.join(assets_dir, 'step_response_Lag.png'))
    plt.close()

    # 2. Root Locus (Lag)
    plt.figure(figsize=(10, 6))
    lag_pole_zero = ct.tf([1, z], [1, p])
    sys_open_lag = lag_pole_zero * sys
    ct.rlocus(sys_open_lag, plot=True, grid=True)
    plt.title(f'Lugar das Raízes (Compensador Lag) - Zero: {z}, Polo: {p}', color='white' if mode=='dark' else 'black')
    plt.savefig(os.path.join(assets_dir, 'rlocus_Lag.png'))
    plt.close()

    # 3. Bode Plot (Lag)
    plt.figure(figsize=(10, 6))
    sys_open_compensated = ctrl * sys
    # Bode plot color needs separate handling if using control library's built-in
    # ct.bode_plot doesn't take 'color' directly for all lines, but returns mag, phase etc.
    # However, usually it respects matplotlib rcParams cycle if we don't force it.
    # We will try passing color or rely on rcParams.
    ct.bode_plot(sys_open_compensated, plot=True, color=colors[2])
    plt.suptitle(f'Diagrama de Bode (Sistema Compensado Lag)', color='white' if mode=='dark' else 'black')
    plt.savefig(os.path.join(assets_dir, 'bode_Lag.png'))
    plt.close()

    plt.close()

    return ctrl

def design_lead_lag_controller(sys, mode='dark'):
    """
    Design and simulate an Integrated Lead-Lag Compensator.
```

```
    Strategy: Combine best properties of both.
    Lag: z=0.1, p=0.01 (High DC Gain)
    Lead: z=20, p=100 (Phase Lead for speed)
    """
    colors = configure_plot_style(mode)
    assets_dir = get_assets_dir(mode)
    grid_color = 'black' if mode == 'light' else 'white'
    grid_alpha = 0.3 if mode == 'light' else 0.3

    s = ct.TransferFunction.s
    # Combined Lead-Lag
    # Lag part: pole=0.01, zero=0.1 (Dierson)
    # Lead part: zero=13.2, pole=150
    # Gain scan
    gains = np.linspace(50000, 150000, 200)

    lag_part = (s + 0.1) / (s + 0.01)
    lead_part = (s + 13.2) / (s + 150)

    # Lag Part
    z_lag = 0.1
    p_lag = 0.01
    C_lag = ct.tf([1, z_lag], [1, p_lag])

    # Lead Part
    z_lead = 20
    p_lead = 100
    C_lead = ct.tf([1, z_lead], [1, p_lead])

    # Combined Gain - Tuning required
    # Lead used K=700, Lag used K=150.
    # Combined needs to balance. Let's start high because Lead allowed
it.
    K = 1000 # Aggressive for performance

    ctrl = K * C_lag * C_lead
    sys_cl = ct.feedback(ctrl * sys, 1)

    t = np.linspace(0, 1, 2000) # Limit to 1s
    t, y = ct.step_response(sys_cl, T=t)

    # Metrics
    y_final = y[-1]
    y_peak = np.max(y)
    Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0

    # Find ts (2%)
    error = np.abs(y - y_final)
    threshold = 0.02 * np.abs(y_final)
    out_of_bounds = np.where(error > threshold)[0]
    ts = t[out_of_bounds[-1]] if len(out_of_bounds) > 0 else 0

    print(f"[{mode.upper()}] Integrated Lead-Lag Design Results -> Mp: {
Mp:.2f}%, ts: {ts:.4f}s")

    # Step Plot
    plt.figure(figsize=(10, 6))
    plt.plot(t, y, linewidth=2, color=colors[3]) # Purple/Cyan/Different
```

```
    plt.title(f'Resposta Final (Lead-Lag Integrado)', color='white' if mode=='dark' else 'black')
    plt.grid(True)
    plt.text(0.6 * np.max(t), 0.5 * np.max(y), f'Mp = {Mp:.1f}%\nts = {ts:.3f}s',
             bbox=dict(facecolor='black', alpha=0.5, edgecolor=colors[1]), color='white')
    plt.savefig(os.path.join(assets_dir, 'step_response_LeadLag.png'))
    plt.close()

    # Root Locus Plot
    plt.figure(figsize=(10, 6))
    ct.rlocus(ctrl*sys, plot=True, grid=True)
    plt.title(f'Lugar das Raízes (Lead-Lag)', color='white' if mode=='dark' else 'black')
    plt.savefig(os.path.join(assets_dir, 'root_locus_LeadLag.png'))
    plt.close()

    # Bode Plot (Rich Style)
    plt.figure(figsize=(10, 8))
    omega = np.logspace(-3, 3, 1000)
    mag, phase, omega = ct.frequency_response(ctrl*sys, omega)
    mag_db = 20 * np.log10(mag)
    phase_deg = np.degrees(np.unwrap(phase))

    # Magnitude
    plt.subplot(2, 1, 1)
    plt.semilogx(omega, mag_db, linewidth=2, color=colors[3])
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.ylabel('Magnitude (dB)')
    plt.title('Diagrama de Bode (Lead-Lag)', color='white' if mode=='dark' else 'black')

    # Phase
    plt.subplot(2, 1, 2)
    plt.semilogx(omega, phase_deg, linewidth=2, color=colors[3])
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    plt.ylabel('Fase (graus)')
    plt.xlabel('Frequência (rad/s)')

    plt.tight_layout()
    plt.savefig(os.path.join(assets_dir, 'bode_LeadLag.png'))
    plt.close()

    return ctrl

def design_pid_controller(sys, mode='dark'):
    """
    Design and simulate a PID Controller (Ziegler-Nichols).
    """
    colors = configure_plot_style(mode)
    assets_dir = get_assets_dir(mode)
    grid_color = 'black' if mode == 'light' else 'white'
    grid_alpha = 0.3 if mode == 'light' else 0.3

    # Ziegler-Nichols Closed Loop Method logic (simplified for implementation)
    # Assume we found Kcr and Pcr.
```

```
    # For this plant G(s) = 849.2 / s(s+13.2)(s+950)
    # Root locus shows it crosses imaginary axis at high gain?
    # Actually type 1 system 3rd order is stable for all K > 0? No, usually bounded.
    # Let's use the PID values we showcased in the slides (or derive reasonable ones).
    # Task says "Implement". Let's assume Kp, Ki, Kd based on prior knowledge/slide content.
    # Slide mentions "Ziegler Nichols".
    # Let's use a "good" PID.

    # Kp = 200, Ki = 100, Kd = 5 -> Tuning for Dierson Model
    # Need much higher K values.
    # Ziegler Nichols alike?
    Kp_pid = 80000
    Ki_pid = 10000
    Kd_pid = 500

    # Filter for derivative: N=100 -> tau = Kd/N ? or just pole
    tau = 0.001

    s = ct.TransferFunction.s
    pid_s = Kp_pid + Ki_pid/s + Kd_pid*s/(tau*s + 1)
    ctrl = pid_s
    # Let tau = 0.001 (very fast filter pole)
    tau = 0.001

    pid_tf = ct.tf([Kd_pid, Kp_pid, Ki_pid], [tau, 1, 0])
    sys_cl = ct.feedback(pid_tf * sys, 1)

    t = np.linspace(0, 1.5, 1000)
    t, y = ct.step_response(sys_cl, T=t)

    # Metrics
    y_final = y[-1]
    y_peak = np.max(y)
    Mp = (y_peak - y_final) / y_final * 100 if y_final != 0 else 0

    plt.figure(figsize=(10, 6))
    plt.plot(t, y, linewidth=2, color=colors[3]) # Purple
    plt.title(f'Resposta ao Degrau (PID Ziegler-Nichols)', color='white' if mode=='dark' else 'black')
    plt.grid(True, which='both', color=grid_color, alpha=grid_alpha)
    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')
    plt.savefig(os.path.join(assets_dir, 'step_response_PID.png'))
    plt.close()

    return pid_tf

def create_plant_variation(Km, am, ae):
    """
    Cria variação da planta para análise de robustez (Nicolas).
    Nominal: Km=1.1, am=13.2, ae=950 -> K_sys=772 fixo.
    """
    K_sys = 772
    num = [Km * K_sys]
```

```
    den = [1, (am + ae), (am * ae), 0]
    return ct.tf(num, den)

def analyze_robustness(controllers_dict, mode='dark'):
    """
    Análise de Robustez baseada nos cenários do Nicolas.
    """
    scenarios = {
        "Nominal":   {"Km": 1.1, "am": 13.2, "ae": 950,  "style": "-", "color_dark": "#00ff00", "color_light": "green"},
        "Pesado":    {"Km": 0.8, "am": 15.0, "ae": 1100, "style": "--", "color_dark": "#00bfff", "color_light": "blue"},
        "Agressivo": {"Km": 1.2, "am": 10.0, "ae": 800,  "style": "-.", "color_dark": "#ff4500", "color_light": "red"}
    }

    colors = configure_plot_style(mode)
    assets_dir = get_assets_dir(mode)

    if mode == 'light':
        text_color = 'black'
        grid_color = 'black'
        face_color = 'white'
        grid_alpha = 0.3
    else:
        text_color = 'white'
        grid_color = 'white'
        face_color = 'black'
        grid_alpha = 0.3

    for ctrl_name, ctrl in controllers_dict.items():
        plt.figure(figsize=(10, 6))
        print(f"[{mode.upper()}] Analisando Robustez: {ctrl_name}")

        for name, params in scenarios.items():
            G_var = create_plant_variation(params["Km"], params["am"], params["ae"])
            sys_cl = ct.feedback(ctrl * G_var, 1)

            # 2 segundos é suficiente para ver a estabilidade
            t, y = ct.step_response(sys_cl, T=np.linspace(0, 2.0, 1000))

            color = params["color_dark"] if mode == 'dark' else params["color_light"]

            # Metrics for legend
            y_peak = np.max(y)
            mp = (y_peak - 1) * 100

            plt.plot(t, y, linestyle=params["style"], linewidth=2, label=f"{name} (Mp={mp:.1f}%)", color=color)

        plt.axhline(1.0, color=text_color, linestyle=':', linewidth=0.8, alpha=0.5)

        plt.grid(True, which='both', linestyle='--', linewidth=0.5, color=grid_color, alpha=grid_alpha)

```

```
        plt.title(f'Análise de Robustez - {ctrl_name}', color=text_color, fontsize=14)
        plt.xlabel('Tempo (s)', color=text_color, fontsize=12)
        plt.ylabel('Amplitude', color=text_color, fontsize=12)

        # Legend with transparency adjustment for dark mode to look good
        legend = plt.legend(facecolor=face_color, edgecolor=text_color)
        for text in legend.get_texts():
            text.set_color(text_color)

        plt.tick_params(colors=text_color, which='both')
        for spine in plt.gca().spines.values():
            spine.set_color(text_color)
if __name__ == "__main__":
    sys = define_system()
    # 1. Open Loop Analysis (Run once for assets)
    if not os.path.exists('../assets/images'): os.makedirs('../assets/images')
    if not os.path.exists('../assets/report_images'): os.makedirs('../assets/report_images')

    # Analyze open loop for both modes
    analyze_open_loop(sys, mode='dark')
    analyze_open_loop(sys, mode='light')

    # 2. Controller Design & Simulation
    controllers_to_test = {}

    print("\n--- Running Control Simulation in DARK mode ---")
    configure_plot_style('dark')
    # Design P
    ctrl_p = design_p_controller(sys, mode='dark')
    # Design Lag (Dierson Strategy)
    ctrl_lag = design_lag_controller(sys, mode='dark')
    # Design Lead
    ctrl_lead = design_lead_controller(sys, mode='dark')
    # Design Lead-Lag
    ctrl_leadlag = design_lead_lag_controller(sys, mode='dark')
    # Design PID
    ctrl_pid = design_pid_controller(sys, mode='dark')

    # Save controllers for robustness test (using Dark mode objects is fine)
    controllers_to_test = {
        'Lead': ctrl_lead,
        'Lag': ctrl_lag,
        'Lead-Lag': ctrl_leadlag,
        'PID': ctrl_pid
    }

    print("\n--- Running Control Simulation in LIGHT mode (Prioritizing Report Assets) ---")
    configure_plot_style('light')
    design_p_controller(sys, mode='light')
    design_lag_controller(sys, mode='light')
    design_lead_controller(sys, mode='light')
    design_lead_lag_controller(sys, mode='light')
    design_pid_controller(sys, mode='light')
```

```
645     generate_comparative_plots(sys, ctrl_p, ctrl_lag, mode='light')
646
647     # Robustness Analysis (Dark)
648     analyze_robustness(sys, controllers_to_test, mode='dark')
649
650     # Robustness Analysis (Light)
651     analyze_robustness(sys, controllers_to_test, mode='light')
652
653     print("\nTodas as simulações e gráficos foram atualizados.")
```

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