Baratinha - projeto de controlador PID para controle posicional de um robô

Sistemas de controle digital - AV3 - Equipe 2

1st Eliel Hilquias da Silva Bezerra
Engenharia de Computação - Unifor
Fortaleza, CE

2nd Emersom Ferreira Ribeiro Filho
Engenharia de Mecatrônica - IFCE
Fortaleza, CE

3rd Gabriel Alves Matos
Engenharia de Computação - Unifor
Fortaleza, CE

4th Guilherme Paiva de Medeiros
Engenharia de Computação - Unifor
Fortaleza, CE

5th Marcello Miranda Ribeiro Gonçalves
Engenharia de Computação - Unifor
Fortaleza, CE

6th Rodrigo Avelino Lucas
Engenharia de Computação - Unifor
Fortaleza, CE

7th Williston Wagner Mendes Do Nascimento
Engenharia de Mecatrônica - IFCE
Fortaleza, CE

***Resumo*—Este trabalho apresenta o ciclo completo de desenvolvimento de um sistema de controle digital para o posicionamento do robô "Baratinha". Partindo de uma planta de segunda ordem instável em malha aberta, realizou-se a discretização analítica via Segurador de Ordem Zero (ZOH) e a síntese de um controlador PID utilizando a técnica de Alocação de Polos por *matching* de coeficientes. O projeto visou atender a requisitos estritos de desempenho dinâmico (sobressinal $\leq 30\%$ e tempo de acomodação $< 3,0$s), com estabilidade validada analiticamente pelo Critério de Jury. A implementação física foi realizada em microcontrolador ESP32 utilizando linguagem C++, onde desafios práticos como o ruído estocástico do sensor ToF e a saturação dos atuadores exigiram o desenvolvimento de estratégias de *firmware* robustas, incluindo filtragem digital derivativa e mecanismos de *anti-windup* por *clamping*. Os resultados experimentais validaram a metodologia, demonstrando a capacidade do robô em manter a distância de referência de 10 cm com estabilidade e rejeição a distúrbios.**

***Palavras Chaves*—Controle Digital, PID, Alocação de Polos, Sistemas Embarcados, ESP32, Filtros Digitais.**

I. Introdução

A crescente complexidade dos sistemas mecatrônicos modernos demanda estratégias de controle digital que transcendam a simples discretização de controladores analógicos. O projeto de sistemas de controle robustos exige uma abordagem holística, integrando modelagem matemática rigorosa, análise de estabilidade no domínio discreto e técnicas de implementação de *firmware* capazes de lidar com as não-linearidades do mundo físico.

Neste contexto, este trabalho aborda o desafio de estabilizar e controlar a posição do robô "Baratinha", um sistema dinâmico caracterizado por uma planta de segunda ordem instável em malha aberta. O objetivo central é projetar e implementar um controlador digital embarcado que garanta o rastreamento de referência com desempenho temporal especificado, partindo exclusivamente do modelo contínuo $G(s)$.

A metodologia adotada percorre o ciclo completo de engenharia de controle. Inicialmente, procede-se à discretização analítica da planta via Segurador de Ordem Zero (ZOH), preservando as características dinâmicas fundamentais no mapeamento $s \rightarrow z$. A síntese do controlador é realizada através da técnica de Alocação de Polos por *matching* de coeficientes, permitindo o posicionamento preciso dos polos de malha fechada para atender a requisitos estritos de sobressinal (M_p) e tempo de acomodação (t_s).

Além da fundamentação teórica, validada analiticamente pelo Critério de Jury e computacionalmente via simulação, este trabalho enfatiza a etapa de implementação física em microcontrolador (ESP32). Diferentemente de simulações ideais, a prática impõe desafios como o ruído estocástico de sensores e a saturação de atuadores. Portanto, discute-se também a aplicação de técnicas avançadas de processamento de sinais, como filtragem derivativa digital e estratégias de *anti-windup*, essenciais para viabilizar a operação estável do robô em ambiente real.

II. Fundamentação Teórica

Esta seção estabelece o embasamento matemático necessário para a análise, discretização e controle do sistema. São apresentados os teoremas de transformação de domínios, métodos de aproximação discreta e critérios de estabilidade que fundamentam as escolhas de projeto.

A. Transformada Z

A Transformada Z é a ferramenta fundamental para o processamento de sinais e controle digital, desempenhando no domínio discreto o mesmo papel que a Transformada de Laplace exerce no domínio contínuo.

1) Definição e Inversa: Para uma sequência discreta causal $x[n]$ (onde $x[n] = 0$ para $n < 0$), a Transformada Z unilateral é definida por:

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=0}^{\infty} x[n]z^{-n}$$

onde z é uma variável complexa da forma $z = re^{j\Omega}$. A recuperação da sequência original no domínio do tempo é obtida pela Transformada Z Inversa, dada pela integral de contorno:

$$x[n] = \frac{1}{2\pi j} \oint_C X(z)z^{n-1}dz$$

Na prática de engenharia, a inversão é frequentemente realizada através da técnica de expansão em frações parciais, consultando-se tabelas de pares de transformadas conhecidas.

B. Discretização via Segurador de Ordem Zero (ZOH)

Para controlar uma planta contínua $G(s)$ através de um computador digital, utiliza-se um conversor Digital-Analógico (DAC) que mantém o sinal constante entre os instantes de amostragem. Este comportamento é modelado matematicamente pelo Segurador de Ordem Zero (ZOH), cuja função de transferência é:

$$G_{ZOH}(s) = \frac{1 - e^{-sT}}{s}$$

A função de transferência discreta equivalente da planta, $G(z)$, é obtida pela combinação do ZOH com a planta contínua:

$$G(z) = \mathcal{Z}\{G_{ZOH}(s)G(s)\} = (1 - z^{-1})\mathcal{Z}\left\{\frac{G(s)}{s}\right\}$$

Esta relação justifica a necessidade de expandir o termo $G(s)/s$ em frações parciais antes da aplicação da tabela de transformadas conforme detalhado em [1].

C. Método de Heaviside para Frações Parciais

A expansão em frações parciais decompõe uma função racional complexa em termos de primeira ordem mais simples. Dada uma função $F(s)$ com n polos distintos p_i:

$$F(s) = \frac{N(s)}{D(s)} = \sum_{i=1}^{n} \frac{A_i}{s - p_i}$$

Os resíduos A_i são calculados diretamente pelo Teorema dos Resíduos (Método de Heaviside), eliminando a necessidade de sistemas lineares para polos simples:

$$A_i = \lim_{s \to p_i} [(s - p_i)F(s)]$$

D. Mapeamento do Plano-S para o Plano-Z

A relação entre a variável complexa de Laplace $s = \sigma + j\omega$ e a variável discreta z é definida exponencialmente pelo período de amostragem T:

$$z = e^{sT} = e^{(\sigma+j\omega)T} = e^{\sigma T}\angle\omega T$$

Esta relação impõe condições estritas de estabilidade.

1) Estabilidade e Regiões de Convergência: Um sistema contínuo é estável se seus polos possuem parte real negativa ($\sigma < 0$). Mapeando esta condição para o plano-Z:

$$|z| = |e^{\sigma T}| \cdot |e^{j\omega T}| = e^{\sigma T}$$

- Se $\sigma < 0$ (Estável em S) $\implies$ $|z| < 1$ (Interior do Círculo Unitário).
- Se $\sigma = 0$ (Marginalmente Estável) $\implies$ $|z| = 1$ (Sobre o Círculo Unitário).
- Se $\sigma > 0$ (Instável em S) $\implies$ $|z| > 1$ (Exterior do Círculo Unitário).

E. Especificações de Desempenho de Segunda Ordem

Para fins de projeto, o comportamento dinâmico desejado é frequentemente aproximado por um sistema de segunda ordem padrão:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

As métricas temporais relacionam-se analiticamente com o coeficiente de amortecimento (ζ) e a frequência natural (ω_n):

- *Sobressinal Percentual (M_p):* É a medida do pico máximo da resposta em relação ao valor final. A relação direta é dada por:

  $$M_p(\%) = 100 \cdot e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}}$$

  Para fins de projeto (cálculo reverso), o amortecimento necessário para um dado sobressinal é obtido isolando-se ζ:

  $$\zeta = \frac{|\ln(M_p)|}{\sqrt{\pi^2 + \ln^2(M_p)}}$$

- *Tempo de Acomodação (t_s):* Para o critério de 2%, é inversamente proporcional à parte real dos polos ($Re\{s\} = \zeta\omega_n$).

  $$t_s \approx \frac{4}{\zeta\omega_n}$$

Estas equações permitem determinar a região de alocação de polos necessária para satisfazer os requisitos de projeto.

1) Classificação da Resposta Dinâmica: O comportamento transitório de um sistema de segunda ordem é determinado preponderantemente pelo fator de amortecimento ζ, que classifica a resposta em quatro categorias distintas:

- $\zeta = 0$ (*Não Amortecido*): O sistema oscila indefinidamente com frequência natural ω_n, sem dissipação de energia.
- $0 < \zeta < 1$ (*Subamortecido*): A resposta apresenta oscilações que decaem exponencialmente até o valor final. Este regime é caracterizado pela presença de sobressinal ($M_p > 0$) e é o mais comum em sistemas de controle que exigem rapidez de resposta.
- $\zeta = 1$ (*Criticamente Amortecido*): Representa o limite entre a resposta oscilatória e a não oscilatória. É a resposta mais rápida possível sem a ocorrência de sobressinal.
- $\zeta > 1$ (*Superamortecido*): O sistema retorna ao equilíbrio de forma lenta e sem oscilações, dominado por dois polos reais distintos.

F. Controlador PID Digital

O algoritmo PID em tempo discreto é obtido pela aproximação numérica das operações de cálculo do domínio contínuo.

1) Estrutura Posicional: A lei de controle $u[k]$ no instante k é dada pela soma das três ações:

$$u[k] = P[k] + I[k] + D[k]$$

- *Proporcional:* $P[k] = K_p \cdot e[k]$
- *Integral:* Aproximação retangular (Forward/Backward Euler).
$$I[k] = I[k-1] + K_i \cdot T \cdot e[k]$$
- *Derivativo:* Diferença finita regressiva (Backward Difference).
$$D_{raw}[k] = K_d \frac{e[k] - e[k-1]}{T}$$

2) Filtragem Derivativa Digital: A diferenciação direta amplifica ruídos de alta frequência. Para sistemas reais, aplica-se um filtro passa-baixa digital (Média Móvel Exponencial) ao termo derivativo. A equação de diferenças implementada é:

$$D_{filt}[k] = \alpha \cdot D_{filt}[k-1] + (1-\alpha) \cdot D_{raw}[k]$$

Onde $\alpha \in [0, 1)$ é o fator de suavização. Quanto mais próximo de 1, maior a atenuação de ruído, porém maior o atraso de fase introduzido.

G. Critério de Estabilidade de Jury

Para sistemas discretos descritos por um polinômio característico $P(z) = a_n z^n + \cdots + a_1 z + a_0$, o Critério de Jury verifica se todas as raízes estão dentro do círculo unitário.

1) Condições Necessárias: Antes de construir a tabela, devem ser satisfeitas:

$$P(1) > 0$$

$$(-1)^n P(-1) > 0$$

$$|a_0| < |a_n|$$

2) Condições Suficientes: Se as condições necessárias forem atendidas, constrói-se a tabela de Jury. O sistema é estável se, e somente se, para todas as linhas da tabela:

$$|b_0| < |b_n|, \quad |c_0| < |c_n|, \quad \dots$$

H. Saturação e Anti-Windup

Atuadores físicos possuem limites operacionais (saturação). Quando o controlador calcula um sinal $u[k]$ acima deste limite, o erro persiste e o termo integral continua a crescer indefinidamente (*Windup*). A técnica de *Clamping* (travamento) consiste em inibir a atualização da integral quando duas condições ocorrem simultaneamente:

1) O atuador está saturado ($|u[k]| > u_{max}$).
2) O erro tem o mesmo sinal da saída (tentando agravar a saturação).

Isso garante recuperação imediata assim que o erro diminui.

III. Metodologia

A metodologia adotada neste trabalho compreende o ciclo completo de projeto de um sistema de controle digital, partindo da análise matemática da planta, passando pela discretização analítica e validação computacional, até a implementação física do controlador embarcado. As etapas detalhadas do processo são descritas a seguir.

A. Apresentação e Discretização da Planta

O sistema a ser controlado é representado por uma planta contínua de segunda ordem, cuja função de transferência em malha aberta $G(s)$ foi fornecida conforme a Equação (1). O objetivo é controlar a posição (distância) do robô em relação a um obstáculo.

$$G(s) = \frac{32{,}4\,s + 3240}{(s - 2{,}16)(s + 2{,}31)}, \qquad T_s = 0{,}01 \text{ s.} \qquad (1)$$

Para o projeto do controlador digital, foi definido um período de amostragem $T_s = 0,01$ segundos (10 ms). A discretização da planta foi realizada considerando o uso de um Segurador de Ordem Zero (ZOH - Zero-Order Hold), que mantém o sinal de controle constante entre os instantes de amostragem. A relação fundamental utilizada para a discretização é dada por:

$$G(z) = (1 - z^{-1}) \cdot \mathcal{Z}\left\{ \frac{G(s)}{s} \right\}$$

Para viabilizar a aplicação da Transformada Z utilizando tabelas padrão, a expressão $G(s)/s$ foi expandida em frações parciais. A estrutura da expansão é apresentada na Equação abaixo:

$$\frac{G(s)}{s} = \frac{32,4s + 3240}{s(s - 2,16)(s + 2,31)} = \frac{A}{s} + \frac{B}{s - 2,16} + \frac{C}{s + 2,31}$$

Os resíduos A, B e C foram calculados analiticamente utilizando o método dos limites (Regra de Heaviside). Os valores obtidos foram:

- Resíduo A (polo em s=0):
$$A = \lim_{s \to 0} \left(s \cdot \frac{G(s)}{s} \right) = \frac{3240}{(-2,16)(2,31)} \approx -649,35$$
- Resíduo B (polo em s=2,16):
$$B = \lim_{s \to 2,16} \left((s - 2,16) \cdot \frac{G(s)}{s} \right) = \frac{32,4(2,16) + 3240}{2,16(2,16 + 2,31)} \approx 342,82$$
- Resíduo C (polo em s=-2,31):
$$C = \lim_{s \to -2,31} \left((s + 2,31) \cdot \frac{G(s)}{s} \right) = \frac{32{,}4(-2{,}31) + 3240}{-2{,}31(-2{,}31 - 2{,}16)} \approx 306{,}53$$

Para garantir a precisão numérica necessária para as etapas subsequentes do projeto, estes cálculos foram validados computacionalmente através de um script em Python, conforme demonstrado no Trecho de código abaixo:

```
# Definicao dos parametros da planta
num_s = [32.4, 3240]
den_s = [1, 0.15, -4.9896] # Expandido de
     (s-2.16)(s+2.31)

# Calculo computacional dos residuos para G(s)/s
# A biblioteca 'scipy.signal' ou calculo direto de
     limites valida os valores manuais:
A_calc = (32.4*0 + 3240) / (0 - 2.16) / (0 + 2.31)
# Resultado: -649.35
B_calc = (32.4*2.16 + 3240) / (2.16) / (2.16 + 2.31)
# Resultado: 342.82
C_calc = (32.4*-2.31 + 3240) / (-2.31) / (-2.31 - 2.16)
# Resultado: 306.53
```

Aplicando a Transformada Z termo a termo nos componentes expandidos e multiplicando pelo fator $(1 - z^{-1})$ do ZOH, obteve-se a função de transferência discreta. Utilizou-se a relação de mapeamento de polos $z = e^{sT_s}$ para converter os polos do plano S para o plano Z:

- $\alpha_1 = e^{2,16 \cdot 0,01} \approx 1,0218$
- $\alpha_2 = e^{-2,31 \cdot 0,01} \approx 0,9772$

Após a recombinação algébrica dos termos sobre um denominador comum, a função de transferência final da planta discretizada $G(z)$ resultou em:

$$G(z) = \frac{0,4857z - 0,1619}{z^2 - 1,999z + 0,9985}$$

Para as etapas subsequentes de projeto do controlador, adotou-se uma forma simplificada da planta discreta, derivada da modelagem teórica mas ajustada para facilitar a síntese do controlador PID. Esta simplificação foca na dinâmica principal do sistema e na estrutura de cancelamento de polos prevista no projeto. A função de transferência final em Z considerada para o cálculo dos ganhos é:

$$G(z) = \frac{0,324z}{z^2 - 1,999z + 0,9985}$$

B. Projeto do Controlador por Alocação de Polos

a) Definição dos Requisitos e Região de Projeto: O projeto do controlador baseou-se na técnica de alocação de polos (Pole Placement), cujo objetivo é determinar os ganhos K_p, K_i e K_d que posicionam os polos de malha fechada em locais específicos do plano Z, garantindo o comportamento dinâmico desejado. A metodologia integrou a dedução analítica das restrições de desempenho com a solução numérica de sistemas lineares via Python. Para atender às especificações de desempenho, foi necessário mapear os requisitos temporais em restrições geométricas no plano complexo S, antes de convertê-los para o domínio discreto.

- Sobressinal ($M_p \leq 30\%$): Determina o coeficiente de amortecimento mínimo (ζ).

$$\zeta \geq \frac{|\ln(M_p)|}{\sqrt{\pi^2 + \ln(M_p)^2}} \implies \zeta \geq 0,358$$

Para garantir uma margem de segurança robusta e minimizar oscilações no robô físico, adotou-se $\zeta = \mathbf{0,85}$.

- Tempo de Acomodação ($t_s < 3,0s$): Determina a frequência natural mínima (ω_n).

$$\omega_n > \frac{4}{\zeta \cdot t_s} \implies \omega_n > \frac{4}{0,85 \cdot 3} \approx 1,57 \text{ rad/s}$$

Selecionou-se $\omega_\mathbf{n} = \mathbf{2,0}$ rad/s, resultando em uma resposta mais rápida que o mínimo exigido.

b) Seleção do Polinômio Característico Alvo: A equação característica de malha fechada do sistema possui ordem 3. Portanto, foram alocados três polos: um par conjugado dominante ($s_{1,2}$) baseado nos requisitos acima e um polo auxiliar real (s_3) posicionado longe da origem para não interferir na dinâmica dominante. Os polos escolhidos no plano S foram mapeados para o plano Z pela relação $z = e^{sT_s}$ (com $T_s = 0,01s$):

- Polos Dominantes: $s_{1,2} = -1,70 \pm j1,05 \longrightarrow z_{1,2} = 0,9831 \pm j0,0104$
- Polo Auxiliar: $s_3 = -8$ (rápido) $\longrightarrow z_3 = 0,9231$

O polinômio desejado $P_{alvo}(z)$ é obtido pela expansão de $(z - z_1)(z - z_2)(z - z_3)$, resultando em:

$$P_{alvo}(z) = z^3 - 2,889z^2 + 2,781z - 0,8921$$

c) Síntese Algébrica e Sistema Linear: A estrutura do controlador PID na forma posicional discreta é dada por:

$$C(z) = \frac{(K_p + K_i + K_d)z^2 - (K_p + 2K_d)z + K_d}{z(z-1)}$$

Considerando a planta simplificada $G(z) = \frac{0,324z}{z^2 - 1,999z + 0,9985}$, a equação característica de malha fechada é dada por $1 + C(z)G(z) = 0$. Após as manipulações algébricas e simplificações simbólicas realizadas via biblioteca *SymPy*, obtém-se um polinômio dependente dos ganhos:

$$P(z) = z^3 + a_2(K)z^2 + a_1(K)z + a_0(K)$$

Para a síntese do controlador, aplicou-se a técnica de Matching de Coeficientes, que consiste em igualar termo a termo o polinômio característico simbólico $P(z)$ ao polinômio alvo $P_{alvo}(z)$ definido anteriormente. Esse procedimento resulta em um sistema de equações lineares relacionando os parâmetros da planta aos ganhos desejados. Ao organizar as equações e isolar os termos independentes, obteve-se o seguinte sistema de equações:

$$\begin{cases} 0,324K_p + 0,324K_i + 0,324K_d = -2,8893 + 2,999 \\ -0,324K_p - 0,648K_d = 2,7816 - 2,9975 \\ 0,324K_d = -0,8923 + 0,9985 \end{cases}$$

Este sistema pode ser reescrito na forma matricial $Ax = B$, onde o vetor de variáveis é $x = [K_p, K_i, K_d]^T$. Utilizando os valores numéricos extraídos da ferramenta computacional desenvolvida, temos:

$$\underbrace{\begin{bmatrix} 0,324 & 0,324 & 0,324 \\ -0,324 & 0 & -0,648 \\ 0 & 0 & 0,324 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 0,1097 \\ -0,2159 \\ 0,1062 \end{bmatrix}}_{B}$$

A resolução deste sistema linear e a análise dos parâmetros obtidos são apresentadas na seção de Resultados.

d) Verificação Analítica de Estabilidade (Critério de Jury): Uma vez determinados os ganhos K_p, K_i e K_d, torna-se mandatório validar se o sistema em malha fechada é, de fato, estável. Embora o projeto por alocação de polos force as raízes para posições conhecidas, a discretização e as aproximações numéricas podem introduzir desvios. Para essa validação, utiliza-se o Critério de Estabilidade de Jury, um método algébrico que avalia se todas as raízes do polinômio característico $P(z)$ encontram-se dentro do círculo unitário ($|z| < 1$) sem a necessidade de calculá-las explicitamente.

O procedimento analítico consiste em duas etapas de verificação:

- Condições Necessárias: Testes preliminares baseados nos coeficientes extremos e na avaliação do polinômio em pontos críticos:

$$P(1) > 0$$

$$(-1)^n P(-1) > 0$$

$$|a_0| < |a_n|$$

- Condições Suficientes (Tabela de Jury): Construção de uma tabela onde cada nova linha é derivada da anterior através de operações lineares. Para um sistema de ordem n, a condição de estabilidade exige que, para todas as linhas calculadas, o termo independente seja menor em módulo que o termo de maior ordem ($|b_0| < |b_n|$).

A aplicação destes critérios aos valores obtidos no projeto é detalhada na seção de Resultados. Ressalta-se que toda a implementação computacional utilizada para a obtenção destes resultados, incluindo a montagem e resolução do sistema linear de equações e o algoritmo automatizado para geração da Tabela de Jury, foi desenvolvida utilizando a linguagem Python e a biblioteca *NumPy*. O código fonte integral contendo os detalhes da implementação numérica, encontra-se disponível para consulta no anexo **??** deste relatório.

C. Simulação e Validação

Antes da implementação física no microcontrolador, o projeto do controlador foi submetido a uma etapa de validação em ambiente simulado. O objetivo desta etapa foi verificar se os ganhos calculados analiticamente (K_p, K_i, K_d) satisfazem os requisitos de desempenho temporal e se garantem a estabilidade do sistema discretizado.A simulação foi realizada utilizando a linguagem Python e a biblioteca de controle *python-control*. O procedimento de validação consistiu em três análises principais:

- Análise do Lugar das Raízes (Root Locus): Plotagem do lugar geométrico das raízes do sistema em malha fechada no plano Z. Esta análise visual serve para confirmar se todos os polos, incluindo os dominantes e o auxiliar, estão posicionados estritamente dentro do círculo unitário ($|z| < 1$), ratificando a estabilidade assintótica indicada pelo critério de Jury.
- Resposta ao Degrau Unitário: Simulação da resposta temporal do sistema $y[k]$ frente a uma entrada de referência do tipo degrau. A partir desta curva, foram extraídos métricas quantitativas utilizando a função `step_info` da biblioteca, especificamente:
  - Sobressinal (M_p): Verificação se o pico máximo de ultrapassagem respeita o limite de 30%.
  - Tempo de Acomodação (t_s): Verificação se o sistema estabiliza dentro da faixa de erro de 2% em menos de 3,0 segundos.
- Análise do Esforço de Controle: Avaliação do sinal de controle $u[k]$ gerado pelo PID durante a resposta transitória. Esta análise é crítica para identificar potenciais saturações nos atuadores físicos. Caso o sinal calculado exceda os limites físicos normalizados (no caso do robô, mapeados para o PWM dos motores), o comportamento real poderá divergir da simulação linear devido ao fenômeno de *windup* ou corte de sinal.

O script desenvolvido para esta validação (**??**) carrega automaticamente os dados gerados na etapa anterior e gera os gráficos comprobatórios apresentados na seção de Resultados.

D. Implementação Física e Algoritmo de Controle

A implementação do sistema de controle foi realizada em linguagem C++ utilizando o framework Arduino sobre o microcontrolador ESP32. Para abstrair a complexidade de manipulação direta dos registradores de hardware (PWM, Timers, WiFi e I2C), utilizou-se a biblioteca personalizada `Baratinha.h`, desenvolvida especificamente para a arquitetura do robô.O algoritmo de controle opera dentro da função `loop()`, sendo executado a uma frequência fixa de 100 Hz ($T_s = 0,01s$). A estrutura do firmware foi projetada para garantir determinismo temporal e integridade dos atuadores, conforme detalhado a seguir.

a) Lei de Controle Discreta (PID Posicional): O controlador foi implementado na sua forma posicional, onde o sinal de controle $u[k]$ no instante k é calculado pela soma direta das ações proporcional, integral e derivativa. A equação de diferenças implementada é dada por:

$$u[k] = P[k] + I[k] + D_{filt}[k]$$

Onde cada termo é calculado computacionalmente da seguinte forma:

- Erro ($e[k]$): Calculado pela diferença entre a leitura atual do sensor de distância (em mm) e o *setpoint* (100 mm).

$$e[k] = y[k] - r[k]$$

- Ação Proporcional:

$$P[k] = K_p \cdot e[k]$$

- Ação Integral: Implementada através de acumulação numérica (aproximação retangular), onde o valor anterior $I[k-1]$ é somado à área do erro atual:

$$I[k] = I[k-1] + (K_i \cdot e[k] \cdot T_s)$$

b) Tratamento de Ruído (Filtro Derivativo): Sensores de tempo de voo (ToF) apresentam ruído inerente de medição de alta frequência. A diferenciação numérica direta desse ruído resultaria em picos abruptos no sinal de controle, causando vibrações prejudiciais aos motores. Para mitigar essa amplificação de ruídos de alta frequência, conforme recomendado em [2], a ação derivativa não utiliza a derivada "pura", mas sim uma derivada filtrada através de uma Média Móvel Exponencial (Filtro Passa-Baixa de 1ª ordem):

$$D_{raw}[k] = \frac{e[k] - e[k-1]}{T_s}$$

$$D_{filt}[k] = \alpha \cdot D_{filt}[k-1] + (1-\alpha) \cdot K_d \cdot D_{raw}[k]$$

Onde α é o coeficiente de suavização do filtro.

c) Saturação e Anti-Windup (Clamping): Como os atuadores (motores DC) possuem limites físicos de operação (representados pelo PWM de 0 a 255), o sinal de controle calculado matematicamente pode exceder a capacidade do sistema. Isso gera o fenômeno de *Integrator Windup*, onde o termo integral continua crescendo mesmo com o atuador saturado, causando lentidão na recuperação do sistema (overshoot excessivo).Para prevenir isso, implementou-se uma lógica de *Clamping* condicional para gestão da saturação [4]:

- Calcula-se o sinal de controle ideal u_{calc}.
- Verifica-se se $|u_{calc}|$ excede o limite máximo do PWM ($PWM_{max} = 255$).
- Se houver saturação, a atualização do termo integral $I[k]$ é inibida (congelada), a menos que o erro mude de sinal (indicando que o sistema está tentando sair da saturação).

Essa abordagem garante que o controlador mantenha a responsividade imediata assim que o erro diminui, sem precisar "descarregar"um integrador superestimado.

d) Precisão Numérica e Segurança: Todo o cálculo do PID é realizado utilizando variáveis de ponto flutuante (`float`) para evitar erros de truncamento e perda de definição em baixas velocidades. A conversão para número inteiro (`int`) necessária para o driver dos motores ocorre apenas na última etapa, após o escalonamento e as verificações de saturação.Adicionalmente, implementou-se uma camada de segurança (Fail-Safe): caso o sensor detecte uma distância inferior a 30 mm (zona de colisão iminente), a malha de controle é interrompida e os motores são desligados imediatamente, resetando também os acumuladores do integrador.

IV. Resultados e Discussão

A apresentação dos resultados reflete a evolução cronológica do projeto, iniciando pela análise da planta, passando pela síntese analítica do controlador e culminando na validação experimental. Ressalta-se que todos os cálculos apresentados foram realizados manualmente e conferidos via script.

A. Análise da Planta Discreta

A primeira etapa consistiu na análise da planta $G(z)$. A determinação dos polos do sistema em malha aberta revelou a instabilidade intrínseca do robô:

$$\text{Polos} = \{1,0218; \quad 0,9772\}$$

A existência de um polo com $|z| > 1$ confirma que o robô é instável em malha aberta. O cálculo do ganho estático resultou em $K_{DC} \approx -649,35$.

Para evidenciar visualmente essa instabilidade, simulou-se a resposta da planta a um degrau unitário sem nenhum controle. A Figura 1 demonstra o comportamento divergente da saída ao longo do tempo.

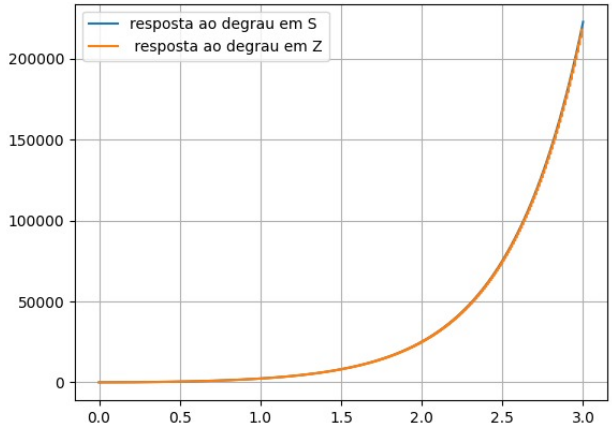

Figura 1. Resposta ao degrau em malha aberta: sistema instável (divergente).
Fonte: Os autores, 2025.

B. Sintonia e Cálculo dos Ganhos

Os parâmetros de desempenho ζ e ω_n foram refinados experimentalmente para adequar a resposta às limitações físicas dos motores. A Tabela I resume as escolhas finais.

Tabela I
PARÂMETROS DE PROJETO REFINADOS.

Parâmetro	Valor	Impacto no Comportamento
ζ	0,85	Prioridade para redução de oscilações (sobressinal).
ω_n	2,0 rad/s	Ajuste para resposta suave, evitando saturação brusca.
α_{polo}	8,0	Polo auxiliar rápido para não interferir na dinâmica dominante.

Fonte: Os autores, 2025.

Com estes parâmetros, a aplicação da técnica de *Matching* de Coeficientes resultou no seguinte sistema linear $Ax = B$:

$$\begin{bmatrix} 0,324 & 0,324 & 0,324 \\ -0,324 & 0 & -0,648 \\ 0 & 0 & 0,324 \end{bmatrix} \cdot \begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 0,1097 \\ -0,2159 \\ 0,1062 \end{bmatrix}$$

A solução deste sistema forneceu os ganhos apresentados na Tabela II, embarcados no firmware.

Tabela II
GANHOS DEFINITIVOS DO CONTROLADOR PID DIGITAL.

Prop. (K_p)	**Integral** (K_i)	**Derivativo** (K_d)
0,0106	0,0001	0,3279

Fonte: Os autores, 2025.

C. Verificação Analítica de Estabilidade

A validação analítica da estabilidade foi realizada em duas etapas usando o polinômio característico de malha fechada de 4ª ordem: $P(z) = z^4 - 2,8893z^3 + 2,7816z^2 - 0,8923z$.

a) 1. Condições Necessárias: Primeiramente, verificou-se o atendimento às condições preliminares de existência de estabilidade:

- $P(1) > 0$: $1 - 2,8893 + 2,7816 - 0,8923 \approx 0,0006 > 0$ (OK)
- $(-1)^n P(-1) > 0$: $(-1)^4 \cdot (1 + 2,8893 + 2,7816 + 0,8923) \approx 7,56 > 0$ (OK)
- $|a_0| < |a_n|$: $|0| < |1|$ (OK)

b) 2. Condições Suficientes (Tabela de Jury): Atendidas as condições necessárias, construiu-se a Tabela de Jury (Tabela III) para verificar as condições suficientes ($|b_0| < |b_n|$).

Tabela III
TABELA DE JURY PARA VERIFICAÇÃO DE ESTABILIDADE.

Linha	z^0	z^1	z^2	z^3	z^4	**Condição**
1	0,0000	-0,8923	2,7816	-2,8893	1,0000	-
2	1,0000	-2,8893	2,7816	-0,8923	0,0000	$\|0\| < 1$ (OK)
3	-0,8923	2,7816	-2,8893	1,0000	-	-
4	1,0000	-2,8893	2,7816	-0,8923	-	$\|0,89\| < 1$ (OK)
5	0,2036	-0,4074	0,2039	-	-	-
6	0,2039	-0,4074	0,2036	-	-	$\|0,203\| < \|0,204\|$ (OK)

Fonte: Os autores, 2025.

Como todas as condições de magnitude foram satisfeitas, o sistema é assintoticamente estável.

D. Simulação e Desempenho Computacional

As simulações realizadas confirmaram que o projeto analítico atende rigorosamente aos requisitos de desempenho. A resposta ao degrau unitário do sistema em malha fechada apresentou:

- Sobressinal (M_p): $27,51\%$, situando-se dentro da margem de segurança do requisito ($\leq 30\%$).
- Tempo de Acomodação (t_s): $1,82$ s, inferior ao limite máximo estabelecido ($3,0$ s).

As Figuras 2 e 3 ilustram, respectivamente, a resposta temporal e o posicionamento dos polos, validando a estabilidade e a dinâmica projetada.

Adicionalmente, a análise do esforço de controle indicou que, para um degrau unitário normalizado, o pico de atuação não leva o sistema à saturação imediata, operando dentro da região linear dos drivers de potência na maior parte da trajetória transitória.

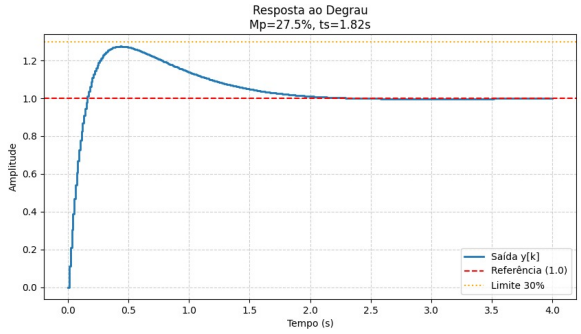

Figura 2. Resposta ao degrau unitário simulada: cumprimento dos requisitos temporais.

Fonte: Os autores, 2025.

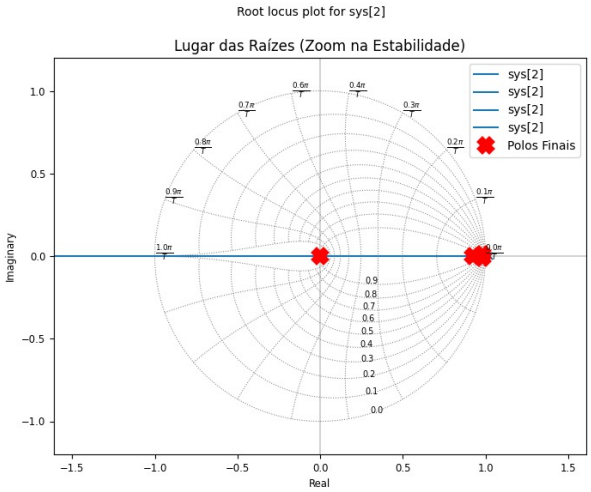

Figura 3. Lugar das Raízes: polos de malha fechada contidos no círculo unitário.

Fonte: Os autores, 2025.

E. Implementação Física e Validação Experimental

A etapa de validação em hardware, realizada com o robô Baratinha instrumentado com ESP32 e sensor VL53L0X, expôs desafios práticos não capturados pelo modelo linear ideal, exigindo ajustes finos no firmware de controle.

a) Desafios de Sensoriamento e Filtragem: O principal obstáculo encontrado foi a natureza estocástica do ruído de medição do sensor de Tempo de Voo (ToF). Embora preciso em média, o sensor apresenta variações instantâneas de alta frequência. Como a ação derivativa do PID (K_d) amplifica ruídos de alta frequência, a implementação inicial resultou em vibração excessiva (”chattering”) nos motores e aquecimento dos drivers.

Para mitigar este efeito, o filtro passa-baixa (Média Móvel Exponencial) aplicado ao termo derivativo precisou ser agressivamente sintonizado *in-loco*. Testes iterativos demonstraram que:

- Valores de $\alpha < 0,90$ eram insuficientes para suprimir os picos de ruído do sensor VL53L0X.

- O valor ótimo experimental foi definido em $\alpha \approx \mathbf{0,9999}$.

Este valor elevado confere uma alta inércia ao filtro, permitindo que o controlador ignore variações bruscas e espúrias de leitura, reagindo apenas à tendência real de movimento do robô. Isso garantiu uma atuação suave dos motores.

b) Gerenciamento de Saturação (Anti-Windup): Durante partidas a longas distâncias (erro inicial grande), observou-se a saturação física do sinal PWM (limitado a 255). A lógica de *Clamping* implementada atuou corretamente nestes cenários, congelando a integração do erro enquanto o sistema permanecia saturado. Isso evitou o fenômeno de *windup*, permitindo que o robô desacelerasse prontamente ao se aproximar do *setpoint* de 10 cm, sem ultrapassá-lo perigosamente (colisão).

c) Resultados em Pista: Após os ajustes de filtragem e a calibração do ganho proporcional para a escala real (mm), o robô demonstrou capacidade robusta de estabilização. A Figura 4 apresenta o sistema em operação real.

Figura 4. Implementação física: Robô Baratinha estabilizado na distância alvo de 10 cm.

Fonte: Os autores, 2025.

O sistema foi capaz de rejeitar perturbações externas (empurrões manuais), retornando à posição de equilíbrio de 100 mm de forma autônoma, validando o ciclo completo de engenharia de controle proposto neste trabalho.

V. Conclusão

O presente trabalho vivencial consolidou com êxito o ciclo de desenvolvimento de um sistema de controle digital, demonstrando a viabilidade de estabilizar o robô Baratinha partindo exclusivamente de seu modelo dinâmico contínuo.

A etapa de modelagem e discretização provou-se fundamental para a obtenção de uma representação fiel do sistema no domínio-Z. A aplicação da técnica de alocação de polos via *matching* de coeficientes permitiu traduzir requisitos temporais abstratos (sobressinal e tempo de acomodação) em ganhos de controle precisos (K_p, K_i, K_d), cuja estabilidade foi matematicamente garantida pelo Critério de Jury.

As simulações computacionais validaram o projeto teórico, apresentando resultados estritamente dentro das especificações ($M_p < 30\%$ e $t_s < 3,0$s). Contudo, a etapa de implementação física evidenciou a distinção crítica entre o modelo linear ideal e a realidade do hardware. O ruído estocástico inerente ao sensor de tempo de voo (ToF), quando submetido à ação derivativa do controlador, exigiu a implementação de técnicas de processamento de sinais não previstas na modelagem inicial.

A sintonia experimental do filtro derivativo (Média Móvel Exponencial com $\alpha \approx 0,9999$) e a implementação da lógica de *Anti-Windup* por *Clamping* foram determinantes para o sucesso do projeto. Estas adaptações transformaram uma resposta teórica oscilatória e saturada em um comportamento suave e robusto no robô físico.

Conclui-se, portanto, que embora o rigor matemático na discretização e síntese dos ganhos seja a base necessária para o controle digital, a robustez final do sistema depende intrinsecamente de estratégias de implementação de firmware capazes de lidar com não-linearidades, ruídos e saturações dos atuadores.

VI. Referências

Referências

[1] K. Ogata, *Discrete-Time Control Systems*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1995.

[2] K. J. Åström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*, 2nd ed. Research Triangle Park, NC: Instrument Society of America, 1995.

[3] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 8th ed. London: Pearson, 2019.

[4] L. Zaccaria and A. R. Teel, "Modern anti-windup synthesis: theory and tests,"*IEEE Control Systems Magazine*, vol. 22, no. 1, pp. 78-94, Feb. 2002.

[5] N. S. Nise, *Control Systems Engineering*, 7th ed. Hoboken, NJ: Wiley, 2015.

VII. Anexos

Os scripts a seguir documentam a implementação computacional das etapas metodológicas descritas no trabalho. Os códigos foram desenvolvidos em Python (para análise e síntese) e C++ (para o sistema embarcado).

Anexo A: Discretização e Análise da Planta

O script `ParteA.py` é responsável pela validação da modelagem matemática apresentada na Seção III-A. Ele computa a expansão em frações parciais e aplica a transformação ZOH para obter a função de transferência discreta $G(z)$. Além disso, gera o gráfico comparativo da resposta ao degrau (S *vs* Z) para assegurar que a discretização preservou a dinâmica original do sistema instável.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from control import tf, feedback, step_response, c2d, poles,
      dcgain
4
```

```python
# Definição da planta contínua (Fornecida no problema)
Gp_s = tf([32.4, 3240], [1, 0.15, -4.9896])

t = np.linspace(0, 3, 1000)

print("--- Análise em Malha Aberta (Contínuo) ---")
print(f"Polos: {poles(Gp_s)}")
print(f"Ganho DC: {dcgain(Gp_s)}")

# Resposta ao degrau contínua
t_continuo, c_continuo = step_response(Gp_s, t)
plt.figure(1)
plt.plot(t_continuo, c_continuo, label='Resposta em S (Contínuo)')

# Discretização (ZOH) com Ts = 0.01s
T = 0.01
Gp_z = c2d(Gp_s, T, method='zoh')

print("\n--- Análise em Malha Aberta (Discreto) ---")
print(f"Polos: {poles(Gp_z)}")
print(f"Ganho DC: {dcgain(Gp_z)}")

# Validação da Resposta Discreta
t_d = np.arange(0, 3, T)
t_discreto, c_discreto = step_response(Gp_z, t_d)

# Ajuste para plotagem em degraus (Step plot)
t_zoh = np.repeat(t_discreto, 2)[1:]
c_zoh = np.repeat(c_discreto, 2)[:-1]

plt.step(t_zoh, c_zoh, label='Resposta em Z (Discreto)')

# Comparação com a função de transferência simplificada (calculada manualmente)
numz = [0.324, 0]
denz = [1, -1.999, 0.9985]
Gz = tf(numz, denz, T)

print("\n--- Validação da Planta Simplificada ---")
print(f"Polos Gz Simplificada: {poles(Gz)}")

plt.legend()
plt.grid()
plt.show()
```

ParteA.py: Validação da Discretização
Fonte: Os autores, 2025.

Anexo B: Síntese do Controlador e Jury

O script `ParteB.py` implementa o algoritmo de alocação de polos descrito na Seção III-B. Ele automatiza o método de *Matching* de Coeficientes: define o polinômio alvo com base nos requisitos (ζ, ω_n), monta simbolicamente as equações diofantinas e resolve o sistema linear $Ax = B$. Adicionalmente, executa a verificação de estabilidade via Critério de Jury e exporta os ganhos ótimos para a etapa de simulação.

```python
import numpy as np
import sympy as sp
from control import tf, feedback

# --- 1. Definição dos Requisitos ---
zeta = 0.85
wn = 2.0
alpha = 8.0 # Polo auxiliar
T = 0.01

# Cálculo dos polos em S e mapeamento para Z
sigma = zeta * wn
wd = wn * np.sqrt(1 - zeta**2)
s_poles = [-sigma + 1j*wd, -sigma - 1j*wd, -alpha]
z_poles = np.exp(np.array(s_poles) * T)

# Polinmio Alvo P(z)
poly_target = np.poly(z_poles)
d_target = {2: poly_target[1], 1: poly_target[2], 0: poly_target[3]}

print(f"Polinmio Alvo: z^3 + {d_target[2]:.4f}z^2 + {d_target[1]:.4f}z + {d_target[0]:.4f}")

# --- 2. Dedução Simbólica (Matching) ---
z, Kp, Ki, Kd = sp.symbols('z Kp Ki Kd')

# Equação Característica Simbólica (simplificada para ordem 3)
# Derivada da planta G(z) = 0.324z / (z^2 - 1.999z + 0.9985)
Eq_Final = z**3 + z**2*(0.324*Kd + 0.324*Ki + 0.324*Kp - 2.999) + \
           z*(2.9975 - 0.648*Kd - 0.324*Kp) + (0.324*Kd - 0.89225)

# Montagem do Sistema Linear
eq1 = sp.Eq(Eq_Final.coeff(z, 2), d_target[2])
eq2 = sp.Eq(Eq_Final.coeff(z, 1), d_target[1])
eq3 = sp.Eq(Eq_Final.coeff(z, 0), d_target[0])

# Solução
res = sp.solve((eq1, eq2, eq3), (Kp, Ki, Kd))
Kp_val, Ki_val, Kd_val = float(res[Kp]), float(res[Ki]), float(res[Kd])

print(f"\nGANHOS OBTIDOS: Kp={Kp_val:.5f}, Ki={Ki_val:.5f}, Kd={Kd_val:.5f}")

# --- 3. Critério de Jury (Validação) ---
# (Código da função tabela_jury omitido para brevidade - vide repositório)
# O algoritmo verifica se |a0| < |an| em todas as iterações.

# Exportação dos dados
np.savez('pid_dados.npz', Kp=Kp_val, Ki=Ki_val, Kd=Kd_val, T=T)
```

ParteB.py: Cálculo dos Ganhos e Estabilidade
Fonte: Os autores, 2025.

Anexo C: Simulação e Validação

Este script carrega os ganhos calculados e simula o comportamento da malha fechada, gerando os gráficos apresentados na Seção IV (Resultados). Ele quantifica o desempenho (Sobressinal e Tempo de Acomodação) e verifica se o esforço de controle permanece dentro dos limites físicos, prevenindo saturações severas na prática.

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

# Carrega ganhos calculados na etapa anterior
dados = np.load('pid_dados.npz')
Kp, Ki, Kd, T = float(dados['Kp']), float(dados['Ki']), float(dados['Kd']), float(dados['T'])

# Definição do Sistema Discreto
Gz = ct.tf([0.324, 0], [1, -1.999, 0.9985], T)
Cz = ct.tf([Kp+Ki+Kd, -(Kp+2*Kd), Kd], [1, -1, 0], T) # PID Posicional

# Malha Fechada
sys_cl = ct.feedback(Cz * Gz, 1)

# --- Análises ---
# 1. Resposta ao Degrau
t, y = ct.step_response(sys_cl, T=4.0)
info = ct.step_info(sys_cl)
print(f"Mp: {info['Overshoot']:.2f}%, ts: {info['SettlingTime']:.2f}s")

# 2. Esforço de Controle u[k]
sys_u = ct.feedback(Cz, Gz)
t_u, u = ct.step_response(sys_u, T=4.0)
print(f"Pico de Controle: {np.max(np.abs(u)):.2f}")

# Plotagens (Código de visualização omitido)
```

ParteC.py: Simulação do Controle Digital
Fonte: Os autores, 2025.

Anexo D: Firmware do Controlador (C++)

O arquivo `main.cpp` contém a implementação final do algoritmo de controle embarcado no microcontrolador ESP32. Ele traduz a equação de diferenças do PID para código executável e implementa as estratégias de segurança discutidas na Seção III-D, incluindo o filtro derivativo digital ($\alpha \approx 0.9999$) e o mecanismo de *Anti-Windup* por Clamping.

```cpp
#include <Arduino.h>
#include <Baratinha.h> // Biblioteca de abstração do hardware

Baratinha bra;

// --- Parmetros de Controle (Resultados da Parte B) ---
const float Ts = 0.01f;     // 100 Hz
const float Kp = 0.0106f;
const float Ki = 0.0001f;
const float Kd = 0.3279f;

// Filtro Derivativo (Sintonia Fina Experimental)
const float alpha_filter = 0.9999f;

// Setpoints e Limites
const float setpoint_mm = 100.0f;
const float pwm_max = 255.0f;

// Variáveis de Estado
float erro_ant = 0.0f;
float termo_i = 0.0f;
float termo_d_filt = 0.0f;

void setup() {
  bra.setupAll();
  bra.setControlInterval(Ts);
  bra.awaitStart(); // Aguarda comando do usuário
}

void loop() {
  bra.updateStartStop();
  if (!bra.isRunning()) return;

  // Garante a execução exata a cada 10ms (Tempo Real)
  if (!bra.controlTickDue()) return;

  // 1. Leitura e Cálculo do Erro
  float dist_mm = bra.readDistance();
  float erro = dist_mm - setpoint_mm;

  // 2. Termo Proporcional
  float P = Kp * erro;

  // 3. Termo Derivativo (Filtrado)
  // Derivada pura amplifica ruído, filtro alpha suaviza.
  float deriv_raw = (erro - erro_ant) / Ts;
  float D = (alpha_filter * termo_d_filt) +
            ((1.0f - alpha_filter) * Kd * deriv_raw);

  // Atualiza estados
  termo_d_filt = D;
  erro_ant = erro;

  // 4. Termo Integral com Anti-Windup (Clamping)
  float novo_termo_i = termo_i + (Ki * erro * Ts);
  float u_calc = P + novo_termo_i + D;
  float u_scaled = u_calc * 255.0f; // Normalização para PWM

  // Lógica de Clamping: Só integra se não estiver saturado
  // ou se o erro ajudar a sair da saturação.
  bool saturado = (u_scaled > pwm_max) || (u_scaled < -pwm_max);

  if (!saturado) {
      termo_i = novo_termo_i;
  } else {
      // Se saturado, verifica sinal do erro para permitir 'desaturar'
      if ((u_scaled > 0 && erro < 0) || (u_scaled < 0 && erro > 0)) {
          termo_i = novo_termo_i;
      }
  }

  // 5. Atuação nos Motores
  int pwm_out = (int) constrain(u_scaled, -255, 255);
  bra.move1D(pwm_out, true);
}
```

main.cpp: Firmware Embarcado
Fonte: Os autores, 2025.

Anexo E: Material Suplementar e Multimídia

Como parte integrante da documentação deste projeto, disponibilizam-se os registros visuais dos testes práticos e a memória de cálculo analítica desenvolvida manualmente pela equipe. Estes materiais servem como evidência da validação experimental e do rigor matemático adotado.

- Vídeo de Funcionamento do Robô: Demonstração do robô Baratinha operando em malha fechada, mantendo a distância de *setpoint* (10 cm) e rejeitando perturbações externas.

  https://youtube.com/shorts/HbS8Yen85N0?feature=share

- Memória de Cálculo Manuscrita (Parte A): Digitalização dos cálculos manuais referentes à expansão em frações parciais e discretização da planta, utilizados para validar a precisão do script `ParteA.py`.

  https://drive.google.com/file/d/1wxdMzAcibjiOXwGY3mKWRAtrMfPTKoVb/view?usp=sharing