Universidade Federal de Santa Catarina

Relatório Final

Processo Dinâmico Tamagotchi

Alunos
Iago de Oliveira Silvestre
Ígor Assis Rocha Yamamoto
João Paulo Zomer Machado
Luís Artur Kretzer Tavares Sobral

Professor
Antonio A. R. Coelho

30 de Novembro de 2016

Sumário

1 Etapa 1: Identificação dos Modelos de Ordem Reduzida

Dada o modelo do processo MDT apresentado na equação 1, foi requisitado uma aproximação de menor ordem. Além disso, será feita a análise das características do processo em malha aberta, trabalhando tanto na forma contínua quanto na discreta.

$$P(s) = \frac{1}{(1+2s)^5} \tag{1}$$

Para adequação ao Matlab, o modelo da equação 1 foi expandido, ficando com a forma da equação 2.

$$P_{ex}(s) = \frac{1}{(32s^5 + 80s^4 + 80s^3 + 40s^2 + 10s + 1)} \tag{2}$$

O modelo então, apresenta as seguintes caracteristicas:

- $t_{5\%} = 18.3$ segundos
- Overshoot = 0%

Neste trabalho, ambas as aproximações foram feitas utilizando o aplicativo *toolbox System Identification* do Matlab. A ferramenta utiliza um estimador polinomial por método iterativo, que faz aproximação por mínimos quadrados não linear e escolhe automaticamente o melhor método de busca.

1.1 Modelo de Primeira Ordem com Atraso

Utilizando o Matlab, o sistema buscado pode ser representado genericamente pela equação 3.

$$P1_{Gen}(s) = \frac{Kp}{(1+Tp.s)} exp(-Td.s) \tag{3}$$

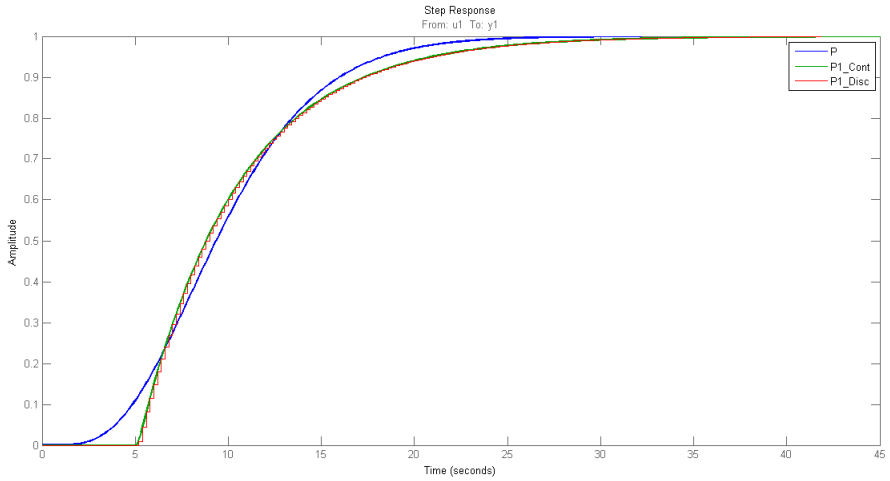
Para o tempo contínuo, o modelo foi obtido escolhendo o valor do Kp para 1, baseado na equação 1, que possui ganho unitário. Assim, o *System Identification*, baseado em método recursivo, aproximou pela equação 4.

$$P1_{Cont}(s) = \frac{1}{(1+5.2637s)} exp(-5.1585s) \tag{4}$$

Levando em contra o período de amostragem fixo do controlador usado em aula, equivalente à 0.2 segundos e utilizando o método $Zero-OrderHold$, chegamos à planta discretizada da equação 5.

$$P1_{Disc}(z) = \frac{0.007855z + 0.02943}{(z - 0.9627)} z^{(-26)} \tag{5}$$

Podemos comparar o *step test* das plantas na figura abaixo:



Assim, a planta apresentou as seguintes características:

- $t_{5\%} = 20.9$ segundos
- Overshoot = 0%
- Adequação ao modelo original = 91.48%

1.2 Modelo de Segunda Ordem com Atraso

Utilizando o Matlab, o sistema buscado pode ser representado genericamente pela equação 6. Nesse caso, foi considerado que o sistema possuiria coeficiente de amortecimento unitário, promovendo resposta mais rápida sem *overshoot* e com polos reais.

$$P2_{Gen}(s) = \frac{Kp}{(1 + Tp_1 s)(1 + Tp_2 s)} exp(-Td.s) \tag{6}$$

Para o tempo contínuo, o modelo foi obtido escolhendo o valor do Kp para 1, baseado na equação 1, que possui ganho unitário. Assim, o *System Identification*, baseado em método recursivo, aproximou pela equação 4.

$$P2_{Cont}(s) = \frac{1}{(11.39s^2 + 6.751s + 1)} exp(-3.4468s) \tag{7}$$

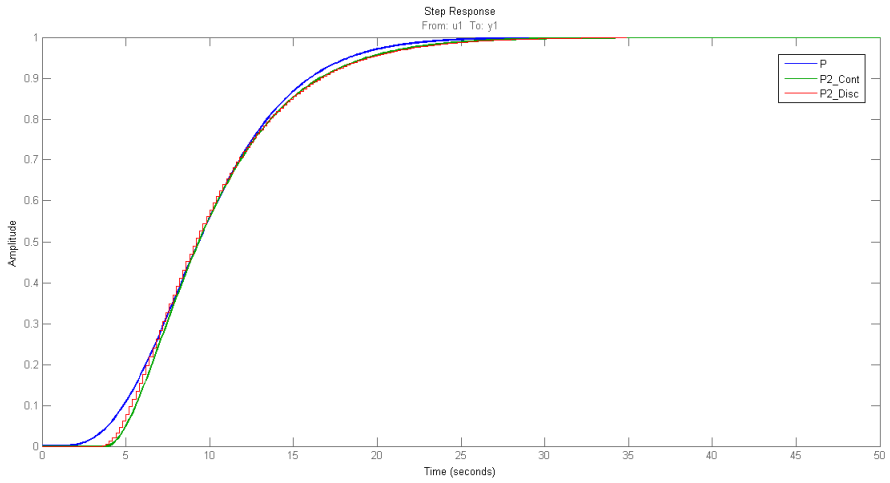
Nesse caso, os polos encontrados foram iguais, então podemos simplificar para a forma da equação 8.

$$P2_{Cont}(s) = \frac{1}{(1+3.3753s)^2} exp(-3.4468s) \tag{8}$$

Novamente, levando em conta as especificações já citadas na seção anterior, chegamos à planta discretizada da equação 9.

$$P2_{Disc}(z) = \frac{0.0009997z^2 + 0.002224z + 8.6110^{-5}}{(z^2 - 1.885z + 0.8882)} exp(-18z) \tag{9}$$

Podemos comparar o *step test* das plantas na figura abaixo:



Assim, a planta apresentou as seguintes características:

- $t_{5\%}$ = 19.5 segundos
- Overshoot = 0%
- Adequação ao modelo original = 96.17%

2 Etapa 2: Simulação em Malha Aberta

Para o teste de modelo discreto o tempo de amostragem foi alterado (conforme indicação do professor). Assim, mudamos de 0,2 segundos para 0,5 segundos. Dessa maneira, as funções de transferência foram alteradas e obtemos as seguintes funções de transferência no domínio discreto.

$$P1_{Disc}(z) = \frac{0.06282z + 0.0278}{(z - 0.9094)} z^{-11} \tag{10}$$

$$P2_{Disc}(z) = \frac{0.0001229z^2 + 0.01171z + 0.007119}{(z^2 - 1.725z + 0.7436)} z^{-7} \tag{11}$$

2.1 Simulações

2.1.1 Modelo Contínuo

Através do Simulink, realizamos o modelo da planta no tempo contínuo (figura 1) e aplicamos variações em degrau nos instantes 1, 50 e 80 com magnitudes de 1, -0.5 e 2, respectivamente. Obtivemos o gráfico da figura 2.

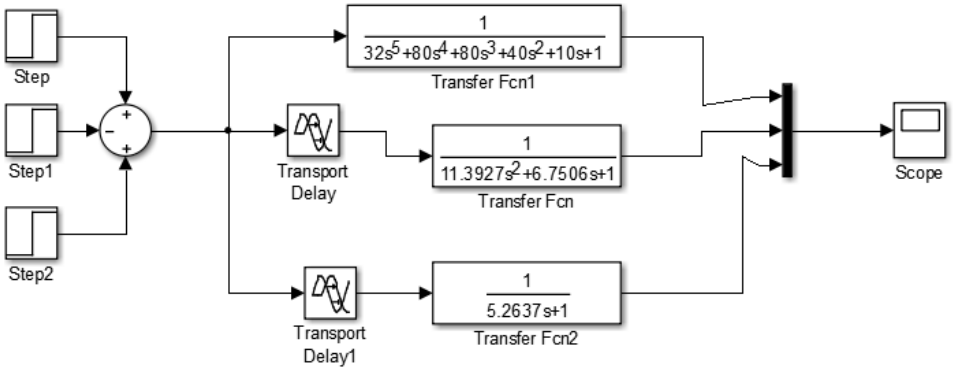


Figura 1: Modelo da Planta em Tempo Contínuo

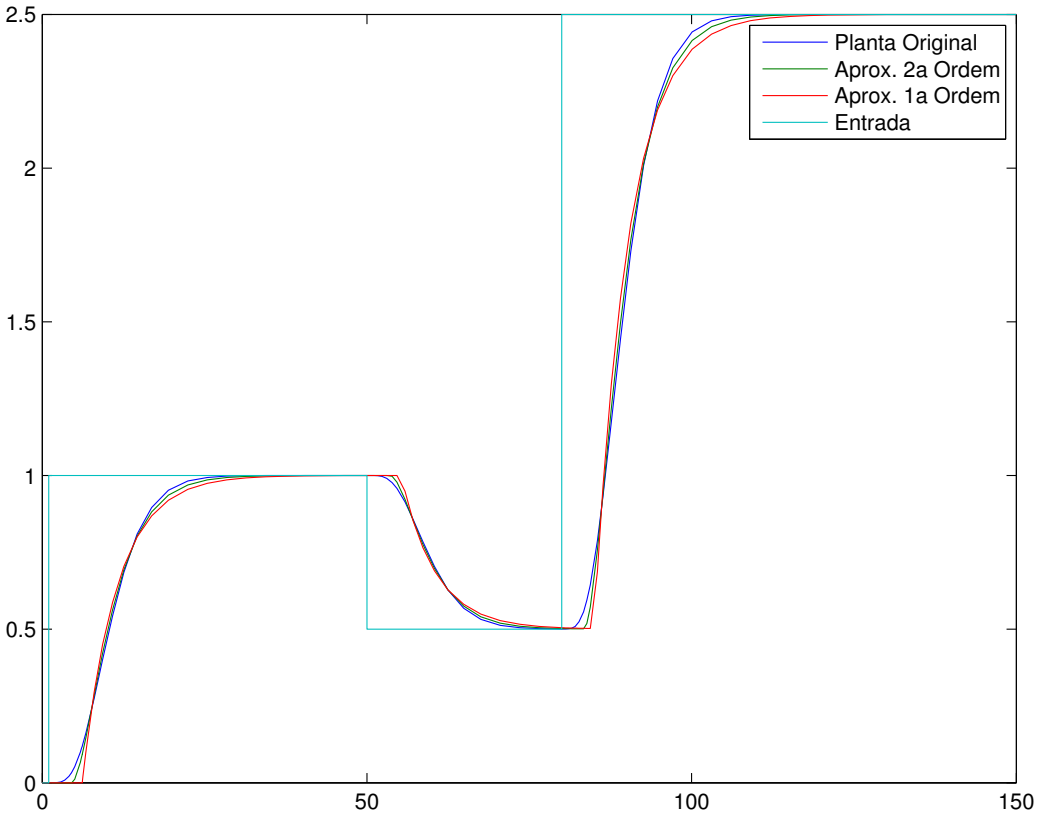


Figura 2: Simulação - Resposta em Malha Aberta da Planta

2.1.2 Modelo Discreto

Através do código em Matlab, mostrado abaixo, realizamos as simulações para os modelos discretizados da planta (equações 10 e 11), para uma entrada em degrau no instante de 5 s. Os resultados são apresentados nas figuras 3, 4 e 5.

```
% Modelos Aproximados da Planta - Continuo
fo = tf([1],[5.2637 1],'InputDelay',5.1585);
so = tf([1],[11.3927 6.7506 1],'InputDelay',3.4468);

% Simulacao - Planta Continua
tsim = 150;
sim('modelo_continuo')
plot(Output.time,Output.signals.values)
legend('Planta Original','Aprox. 2a Ordem','Aprox. 1a Ordem','Entrada')
print('sim_cont','-depsc')
```

```
% Modelos Aproximados da Planta - Discretizacao
Ts = 0.5; % tempo de amostragem
d_fo = c2d(fo,Ts)
d_so = c2d(so,Ts)

% Simulacao - Modelo Discreto de Primeira Ordem
a = d_fo.num{1}(1);
b = d_fo.num{1}(2);
c = d_fo.den{1}(2);
% Entrada Degrau Unitario em t=10
t=10;
U = [zeros(1,t) ones(1,200)];
Y1 = zeros(1,t+12);
for i=(t+13):size(U,2)
    % equacao a diferencas
    Y1(i) = -c*Y1(i-1) + U(i-11)*a +U(i-12)*b;
end
% Figura 1
[yc1,tc1]=step(fo);
tc1 = tc1+t*Ts;
figure(1)
T = Ts*(1:200+t);
stairs(T,Y1)
hold on
plot(tc1,yc1,'g',T,U,'r')
legend('Discreto','Continuo','Entrada')
print('fo','-depsc')

% Simulacao - Modelo Discreto de Segunda Ordem
a = d_so.num{1}(1);
b = d_so.num{1}(2);
c = d_so.num{1}(3);
d = d_so.den{1}(2);
e = d_so.den{1}(3);
% Entrada Degrau Unitario em t=10
t=10;
U = [zeros(1,t) ones(1,200)];
Y2 = zeros(1,t+9);
for i=(t+10):size(U,2)
    % equacao a diferencas
```

```
    Y2(i) = -d*Y2(i-1) - e*Y2(i-2) + U(i-7)*a + U(i-8)*b +U(i-9)*c;
end
% Figura 2
[Yc2,Tc2]=step(so);
Tc2 = Tc2+t*Ts;
figure(2)
stairs(T,Y2)
hold on
plot(Tc2,Yc2,'g',T,U,'r')
xlabel('Tempo')
ylabel('Saida do Processo')
legend('Discreto','Continuo','Entrada')
print('so','-depsc')

% Comparacao - Simulacoes de Primeira e Segunda Ordem
figure(3)
hold on
stairs(T,Y1)
stairs(T,Y2,'g')
plot(T,U,'r')
legend('Discreto 1a ordem','Discreto 2a ordem','Entrada')
print('fo_so','-depsc')
```

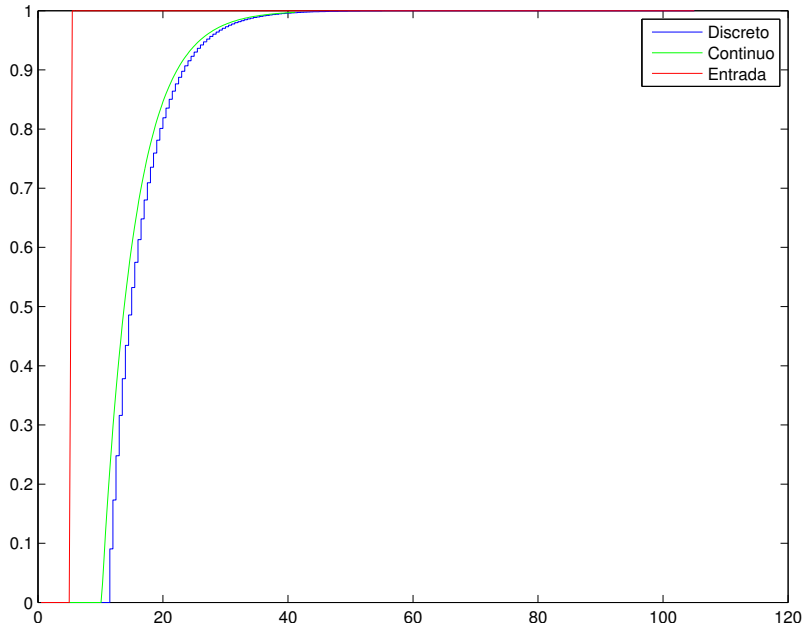


Figura 3: Comparação entre os modelos aproximados de primeira ordem - contínuo e discreto

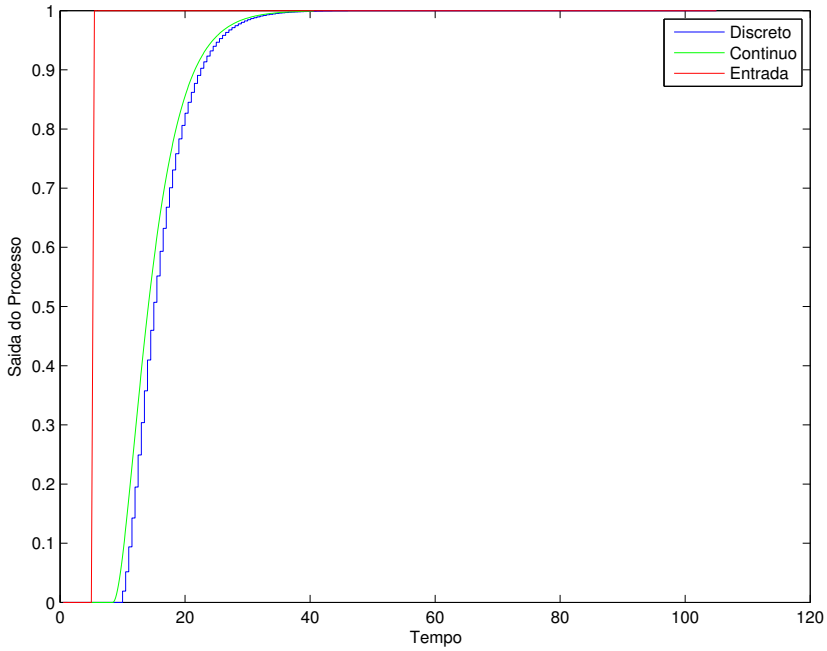


Figura 4: Comparação entre os modelos aproximados de segunda ordem - contínuo e discreto

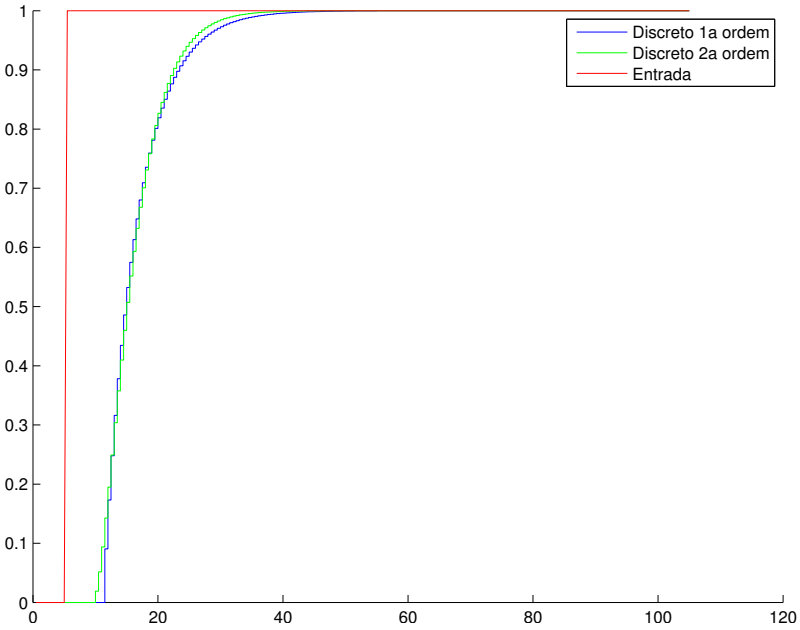


Figura 5: Comparação entre os modelos discretos de primeira e segunda ordem

3 Etapa 3: Sintonia PI,PID via Métodos da Literatura

3.1 Sintonias utilizadas

Tabela 1: Sintonias utilizadas no projeto

Sintonia	K_p	T_i	T_d
Ziegler-Nichols PI	$\frac{0.9\tau}{K_c\theta}$	$\frac{\theta}{0.3}$	-
Chien PI	$\frac{0.35\tau}{K_c\theta}$	1.17τ	-
IMC PI	$\frac{\tau}{K_c(\theta+\tau)}$	τ	-
IMC PID	$\frac{\tau_1}{2\theta K_c}$	$min(\tau_1, 8\theta)$	τ_2
Honeywell PID	$\frac{\tau_1+\tau_2}{K_c(\frac{\tau_1+\tau_2}{3})+\theta}$	$\tau_1+\tau_2$	$\frac{\tau_1\tau_2}{\tau_1+\tau_2}$
Closed-loop specified PID	$\frac{\tau_1}{2K_c\theta}$	τ_1	τ_2

Os parâmetros τ, θ e K_c para os PIs são obtidos através da equação 4. Os parâmetros τ_1, τ_2, θ e K_c para os PIDs são obtidos através da equação 7.

3.2 Código Matlab

```
% Condições iniciais
u(1:6) = 0; yp(1:6) = 0; y(1:6) = 0; du(1:6) = 0; erro(1:6) = 0;
ts = 0.5; nit = 400;
% Parâmetros do processo mdt
kp = 1; tau = 5.2637; teta = 5.1585;
tau1 = 3.3124; tau2 = 3.4388; teta2 = 3.4468;
% Seleciona sintonia do controlador
ordem = menu('CONTROLADOR','PI','PID');
switch ordem
    case 1 % Sintonia PI
        sintonia = menu('SINTONIA','ZIEGLER-NICHOLS','CHIEN','IMC');
        switch sintonia
            case 1 % Sintonia Ziegler-Nichols
                kc = (0.9*teta)/(kp*teta)
                ti = teta/0.3
            case 2 % Sintonia Chien
                kc = 0.35*tau/(kp*teta)
                ti = 1.17*tau
            case 3 % Sintonia IMC (Internal Model Control)
               kc = tau /((kp)*(teta + tau))
               ti = tau
        end
    case 2 % Sintonia PID
        sintonia = menu('SINTONIA','IMC','HONEYWELL','CLOSED-LOOP');
        switch sintonia
            case 1 % Sintonia IMC
                kc = (tau1)/(kp*(2*teta2))
                ti = min([tau1 4*(2*teta2)])
                td = tau2
            case 2 % Sintonia HONEYWELL
                kc = (tau1+tau2)/(kp*(tau1+tau2)/3 +teta2)
                ti = tau1 + tau2
                td = (tau1*tau2)/(tau1+tau2)
            case 3 %Sintonia CLOSED-LOOP specified
                kc = tau1/(2*kp*teta2)
                ti = tau1
                td = tau2
        end
end
```

```
% Referência e perturbação
yr(1:120) = 1; yr(121:280) = 4; yr(281:400) = 2;
do(1:nit/2) = 0; do((nit/2 +1):nit) = 0.2;
% Discretização da planta
nps = 1;
dps = [32 80 80 40 10 1]; % mdt = 1/(2*s+1)^5
[npz,dpz] = c2dm(nps,dps,ts,'zoh');
switch ordem
    case 1
        % Sintonia PI Paralelo
        % Aproximação
        % integral => trapezoidal
        s = tf('s');
        cs = kc*(1+s*ti)/(s*ti)
        cz = c2d(cs,ts,'tustin')
        q0 = kc*(1 + ts/(2*ti));
        q1 = -kc*(1 - ts/(2*ti));
        % Simulação
        for k = 6:nit
        % Saída da planta
              yp(k) = -dpz(6)*yp(k-5)-dpz(5)*yp(k-4)-dpz(4)*yp(k-3)-dpz(3)*yp(k-
                  npz(6)*u(k-5)+npz(5)*u(k-4)+npz(4)*u(k-3)+npz(3)*u(k-2)+npz(2)
              y(k) = yp(k) + 0*do(k);
        % Lei de controle
              erro(k) = yr(k) - y(k);
              u(k) = u(k-1) + q0*erro(k) + q1*erro(k-1);
        end
        % Resultados
        figure(1)
        t = 0:ts:ts*(nit-1);
        subplot(2,1,1),plot(t,y,'r',t,yr,'--k','linewidth',2),
                       ylabel('saída e referência'),xlabel('tempo (s)'),grid;
        subplot(2,1,2),plot(t,u,'b','linewidth',2),
                       ylabel('controle'),xlabel('tempo (s)'),grid;
    case 2
        % Sintonia PID Paralelo
        % Aproximações
        % integral => trapezoidal
        % derivada => diferença de primeira ordem
        s = tf('s');
```

```
        cs = kc*(1+1/(ti*s)+td*s)
        cz = c2d(cs,ts,'tustin')
        q0 = kc*(1 + (ts/(2*ti)) + (td/ts));
        q1 = -kc*(1 - (ts/(2*ti)) + ((2*td)/ts));
        q2 = kc*(td/ts);
        % Simulação
        for k = 6:nit
        % Saída da planta
              yp(k) = -dpz(6)*yp(k-5)-dpz(5)*yp(k-4)-dpz(4)*yp(k-3)-dpz(3)*yp(k-
                  npz(6)*u(k-5)+npz(5)*u(k-4)+npz(4)*u(k-3)+npz(3)*u(k-2)+npz(2)
              y(k) = yp(k) + 0*do(k);
        % Lei de controle
              erro(k) = yr(k) - y(k);
              u(k) = u(k-1) + q0*erro(k) + q1*erro(k-1) + q2*erro(k-2);
        end
        % Resultados
        figure(1)
        t = 0:ts:ts*(nit-1);
        subplot(2,1,1),plot(t,y,'r',t,yr,'--k','linewidth',2),
                       ylabel('saída e referência'),xlabel('tempo (s)'),grid;
        subplot(2,1,2),plot(t,u,'b','linewidth',2),
                       ylabel('controle'),xlabel('tempo (s)'),grid;
end
```

3.3 Modelo dos Controladores PI

3.3.1 Sintonia Ziegler-Nichols

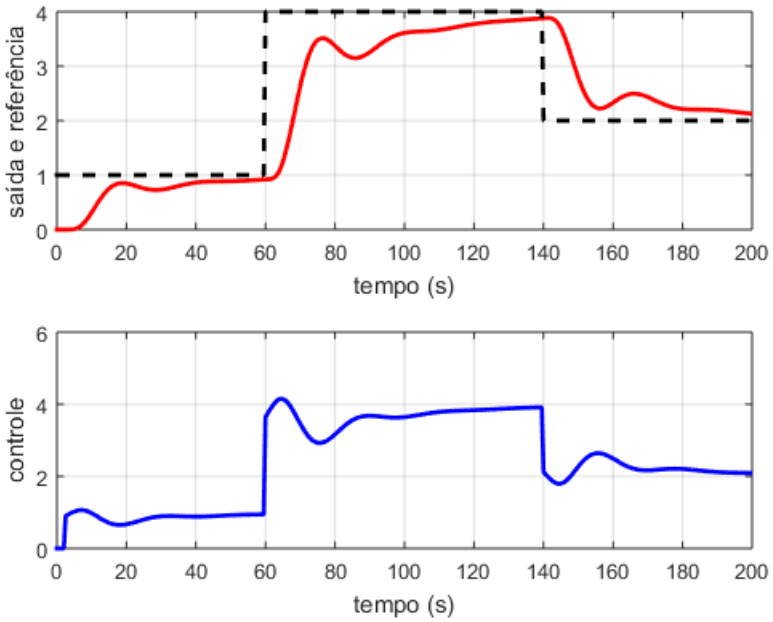


Figura 6: Resposta ao degrau do controlador

As equações abaixo mostram o modelo do controlador:

$$C_1(s) = \frac{0.9(17.195s + 1)}{17.195s} \tag{12}$$

$$C_1(z) = \frac{0.9z - 0.8738}{z - 1} \tag{13}$$

3.3.2 Sintonia Chien

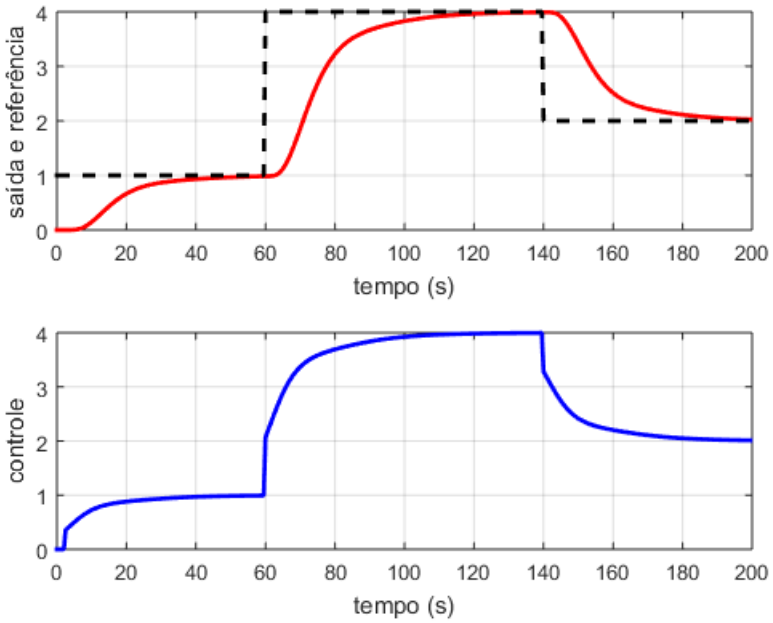


Figura 7: Resposta ao degrau do controlador

As equações abaixo mostram o modelo do controlador:

$$C_2(s) = \frac{0.3571(6.1585s + 1)}{6.1585s} \tag{14}$$

$$C_2(z) = \frac{0.3571z - 0.3281}{z - 1} \tag{15}$$

3.3.3 Sintonia IMC

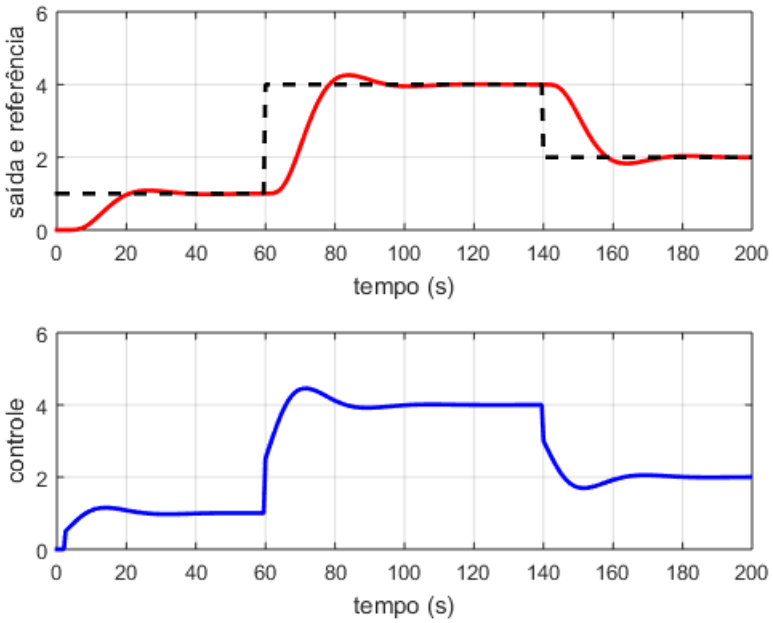


Figura 8: Resposta ao degrau do controlador

As equações abaixo mostram o modelo do controlador:

$$C_3(s) = \frac{0.5050(5.2637s + 1)}{5.2637s} \tag{16}$$

$$C_3(z) = \frac{0.5050z - 0.4571}{z - 1} \tag{17}$$

3.4 Modelo dos Controladores PID

3.4.1 Sintonia IMC

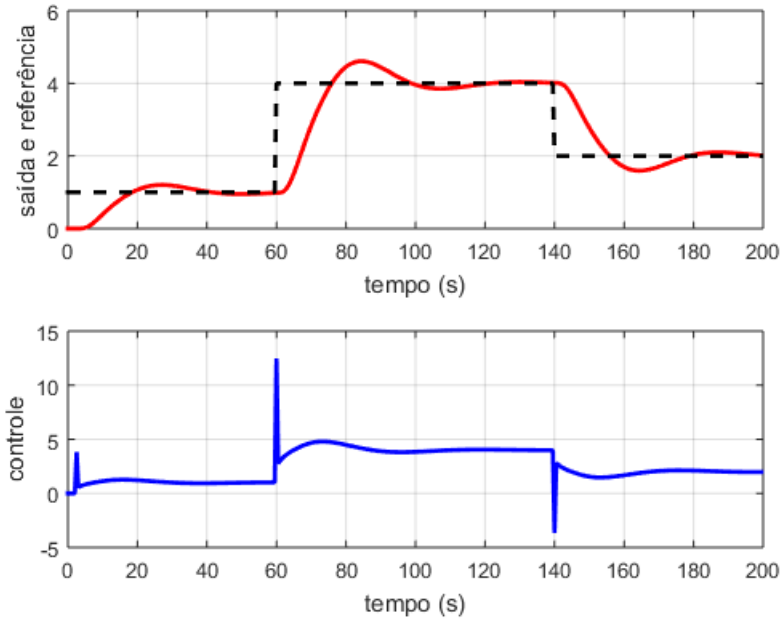


Figura 9: Resposta ao degrau do controlador

As equações abaixo mostram o modelo do controlador:

$$C_4(s) = 0.4805(1 + 3.4388s + \frac{1}{3.3124s}) \tag{18}$$

$$C_4(z) = \frac{7.126z^2 - 13.15z + 6.165}{z^2 - 1} \tag{19}$$

3.4.2 Sintonia Honeywell

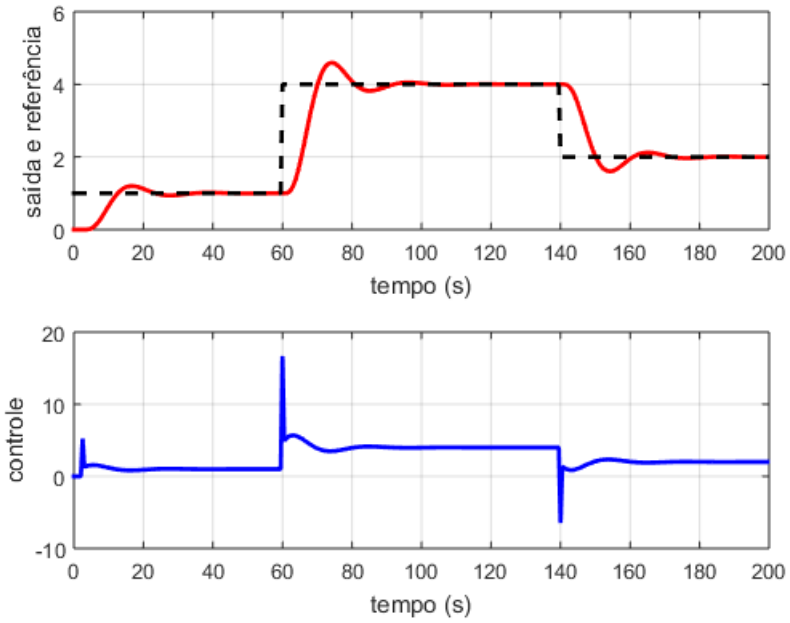


Figura 10: Resposta ao degrau do controlador

As equações abaixo mostram o modelo do controlador:

$$C_5(s) = 1.1850(1 + 1.6872s + \frac{1}{6.7512s}) \tag{20}$$

$$C_5(z) = \frac{9.226z^2 - 15.91z + 6.856}{z^2 - 1} \tag{21}$$

3.4.3 Sintonia Closed-loop

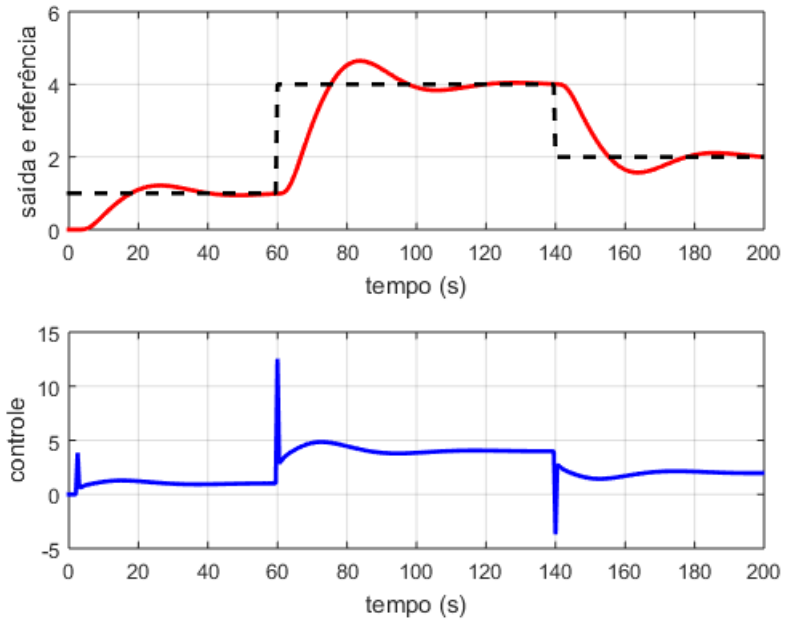


Figura 11: Resposta ao degrau do controlador

As equações abaixo mostram o modelo do controlador:

$$C_6(s) = 0.4988(1 + 3.3124s + \frac{1}{3.3124s}) \tag{22}$$

$$C_6(z) = \frac{7.146z^2 - 13.14z + 6.148}{z^2 - 1} \tag{23}$$

3.5 Diagrama de Blocos Simulink

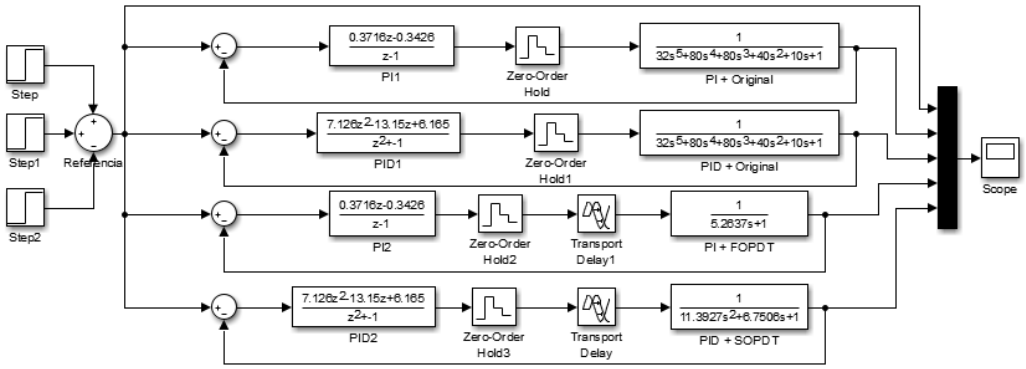


Figura 12: Diagrama de Blocos Simulink

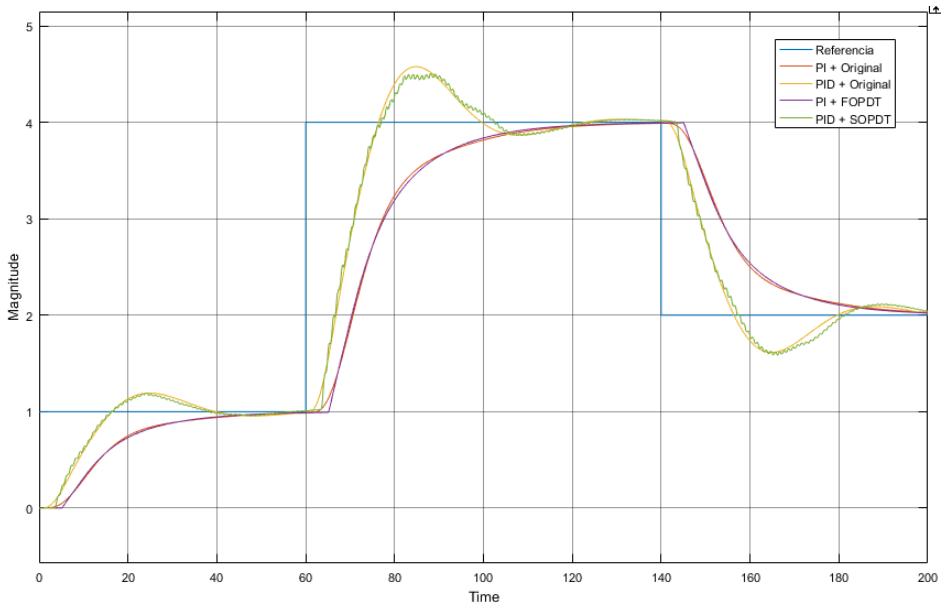


Figura 13: Simulação Simulink

Para o teste no ambiente *Simulink* foram utilizados os controladores obtidos pela sintonia Chien para o PI e pela sintonia IMC para o PID.

4 Etapa 4: Sintonia PI,PID via Lugar das Raízes

O tempo de amostragem dos relatórios anterioes era $T_s = 0.5$ segundos, porém pelo fato da próxima etapa do projeto ser efetuar um HIL no PID da Novus, se escolhe um tempo de amostragem compatível com a máquina, portanto:

$$T_s = 0.2\,s$$

Discretizando as plantas por Tustin se obtém:

$$P1_{Disc}(z) = \frac{0.007853z + 0.02943}{(z - 0.9627)} z^{-26} \tag{24}$$

$$P2_{Disc}(z) = \frac{0.0009994z^2 + 0.002224z + 8.618e - 05}{(z^2 - 1.885z + 0.8882)} z^{-18} \tag{25}$$

4.1 Projeto do Controlador PI

Para realizar o projeto do controlador PI, foi utilizado o modelo fopdt com a aproximação de Padé de primeira ordem para o atraso, para que assim pudéssemos analisar o lugar das raízes e ajustar os polos e zeros do controlador.
Sendo L o tempo do atraso:

$$e^{-Ls} = \frac{1 - \frac{L}{2}s}{1 + \frac{L}{2}s}$$

$$P1_{\ Cont}(s) = \frac{1}{(1 + 5.2637s)} . \frac{1 - \frac{5.1585}{2}s}{1 + \frac{5.1585}{2}s} \tag{26}$$

O ajuste dos polos e zeros foram feitos com auxílio do programa Matlab pela ferramenta SISOTOOL, que faz o desenho do lugar das raízes da equação característica 1+CP.

Além disso, para o ajuste do controlador PI foi utilizado o modelo de primeira ordem com tempo morto (FOPDT) apresentado na equação 26.

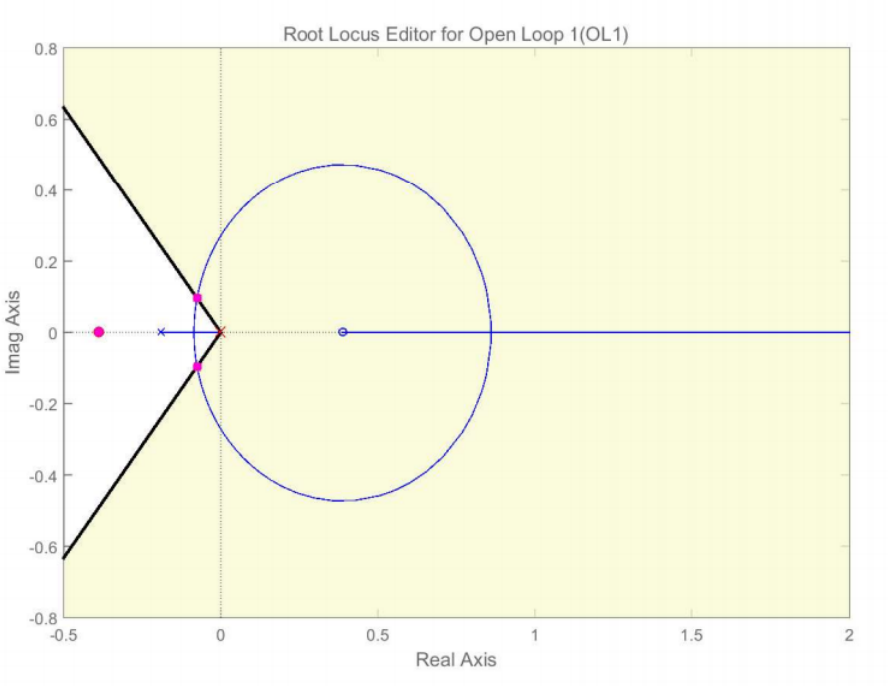


Figura 14: Lugar das Raizes PI-FOPDT

Pelo projeto feito no SISOTOOL, obtemos um controlador PI com um zero em $s = -0.3877$ e polo em $s = 0$:

$$C_{PI}(s) = \frac{0.2021\,(s + 0.3877)}{s}$$

Discretizando por *tustin*, com Ts = 0.2 s obtemos:

$$Cz_{PI}(z) = \frac{0.20993(z - 0.9254)}{(z - 1)}$$

4.2 Projeto do Controlador PID

O ajuste dos polos e zeros foram feitos com auxílio do programa Matlab pela ferramenta SISOTOOL, que faz o desenho do lugar das raízes da equação característica 1+CP.

Além disso, para o ajuste do controlador PID foi utilizado o modelo de segunda ordem com tempo morto (SOPDT), com aproximação de padé de primeira ordem, apresentado por:

$$P2\ _{Cont}(s) = \frac{1}{(11.39s^2 + 6.751s + 1)} . \frac{1 - \frac{3.4468}{2}s}{1 + \frac{3.4468}{2}s} \tag{27}$$

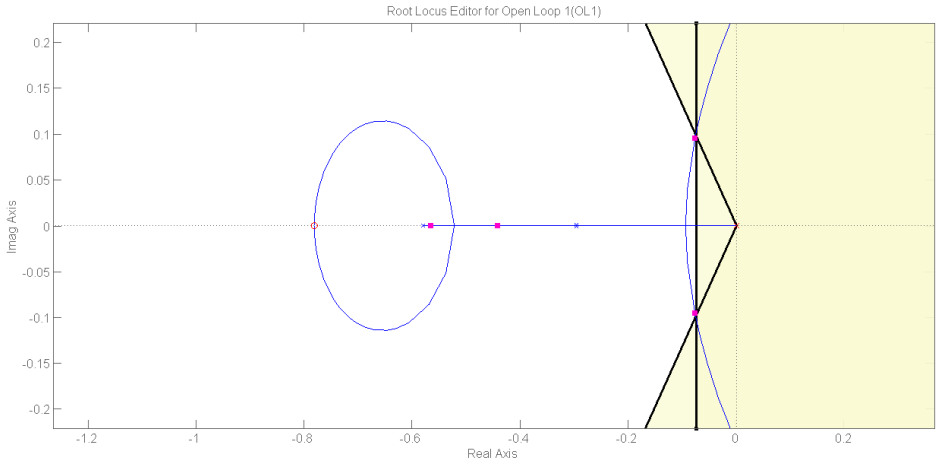


Figura 15: Lugar das Raizes PID SOPDT

Pelo projeto feito no SISOTOOL, obtemos um controlador PI com dois zero em $s = -0.7811$ e polos em $s = 0 e s = -10$:

$$C_{PID}(s) = \frac{1.2068\,(s + 0.7811)^2}{s\,(s + 10)}$$

Discretizando por *tustin*, com Ts = 0.2 s obtemos:

$$Cz_{PID}(z) = \frac{0.70137\,(z - 0.8551)^2}{z\,(z - 1)}$$

Como agora temos a ação derivativa no controlador, podemos considerar que Taw=$\sqrt{Ti.Td}$

4.3 Projeto Controlador com Anti-windup

Para evitar o efeito de windup, que acontece quando o comportamento dinâmico do sistema em malha fechada se difere do comportamento linear que o sistema de controle foi modelado para, por causa de limitações dos sinais de controle no sistema, podemos utilizar a técnica anti-windup, que vai calcular o erro entre o sinal de controle calculado e o sinal de controle enviado realmente para a planta e usar esse erro para diminuir a efeito integrativo do controlador.

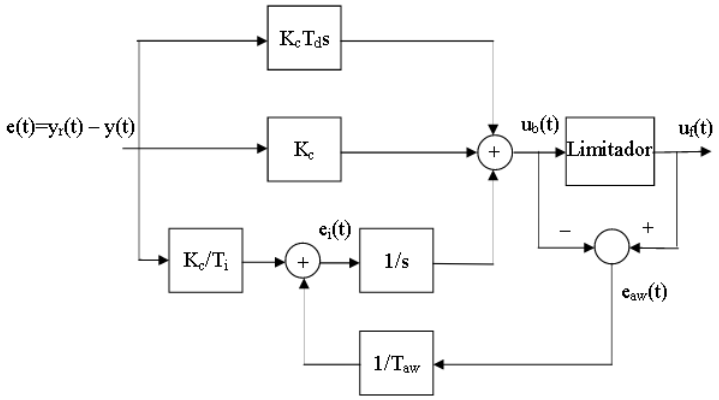


Figura 16: Topologia base técnica anti-windup PID

O Taw deve ser analisado e escolhido pelo projetista, uma das regras empiricas utilizadas é:

$$Taw = \sqrt{Ti.Td} \quad ou \quad Taw = Ti$$

Como o primeiro controlador calculado era um PI sem ação derivativa, foi escolhido Taw=Ti.

4.3.1 Controlador PI

Para o controlador PI encontrado no item 2, podemos escrever ele como:

$$C_{PI}(s) = Kc + \frac{Kc}{Ti}.\frac{1}{s}$$

E os valores de Kc e Ti encontrados foram:

$$Kc = 0.2020$$
$$Ti = 2.5765$$

Além disso, o Taw da ação anti-windup no caso do controlador PI é igual a Ti.

4.3.2 Controlador PID

Para o controlador PID encontrado no item 3, podemos escrever ele como:

$$C_{PI}(s) = Kc + \frac{Kc}{Ti}.\frac{1}{s} + Kc.Td.s$$

E os valores de Kc,Ti e Td encontrados foram:

$$Kc = 0.1810$$
$$Ti = 2.4592$$
$$Td = 0.5691$$

Além disso, o Taw da ação anti-windup no caso do controlador PID é igual a $\sqrt{Ti.Td}$.

4.4 Diagrama de Blocos Simulink

4.4.1 Diagrama

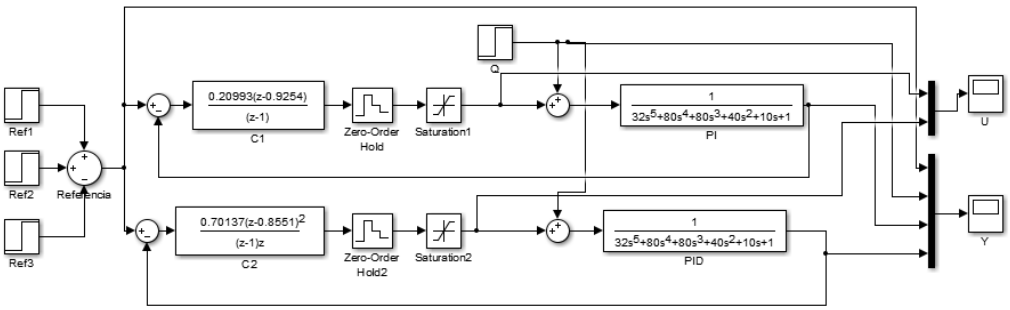


Figura 17: Diagrama de Blocos

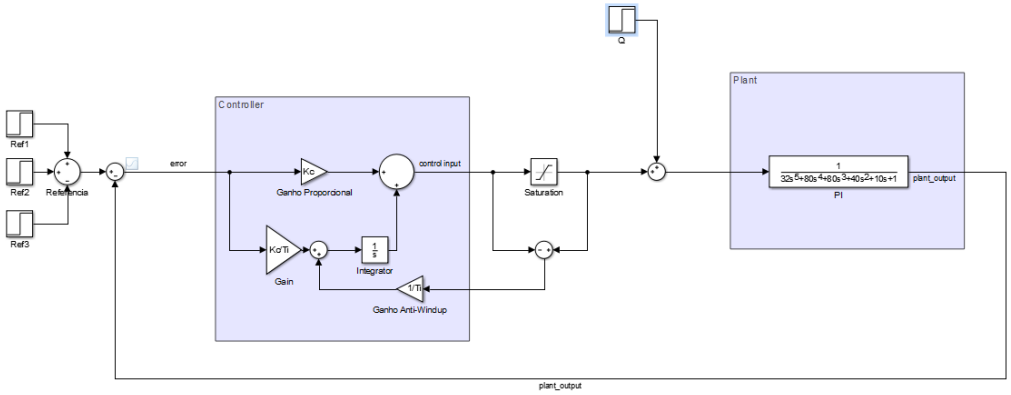


Figura 18: Diagrama de Blocos Anti-Windup PI

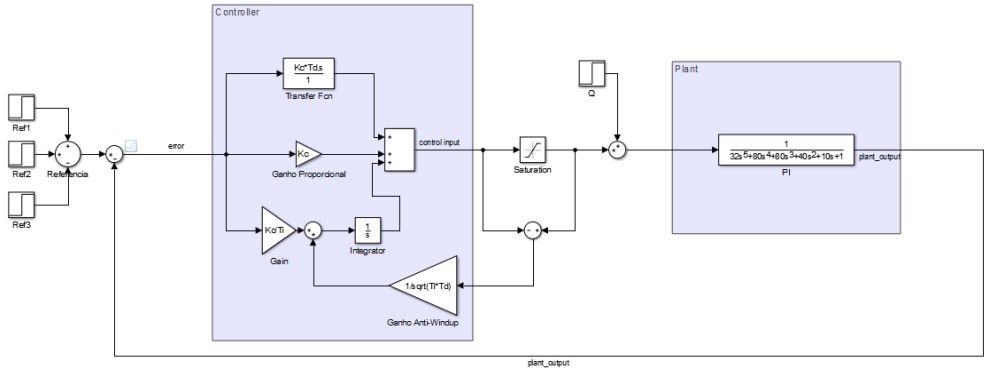


Figura 19: Diagrama de Blocos Anti-Windup PID

4.4.2 Simulação

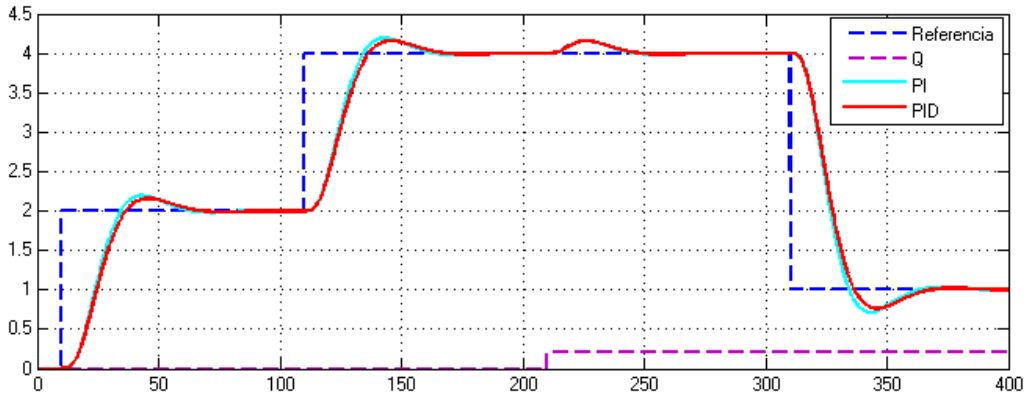


Figura 20: Simulação Saída para 3 mudanças de referência e perturbação

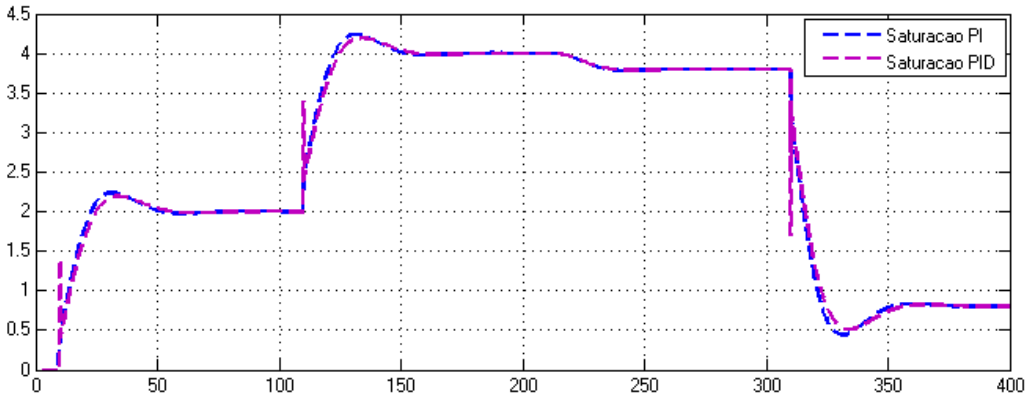


Figura 21: Simulação Sinal de Controle para 3 mudanças de referência

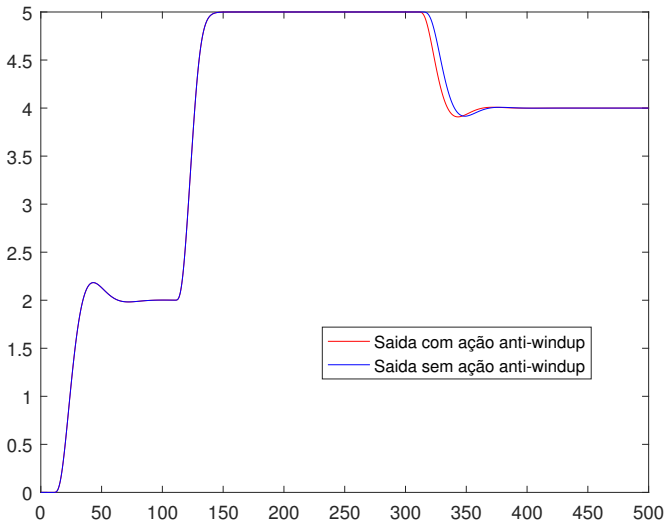


Figura 22: Sinal de saida da planta com e sem anti-windup

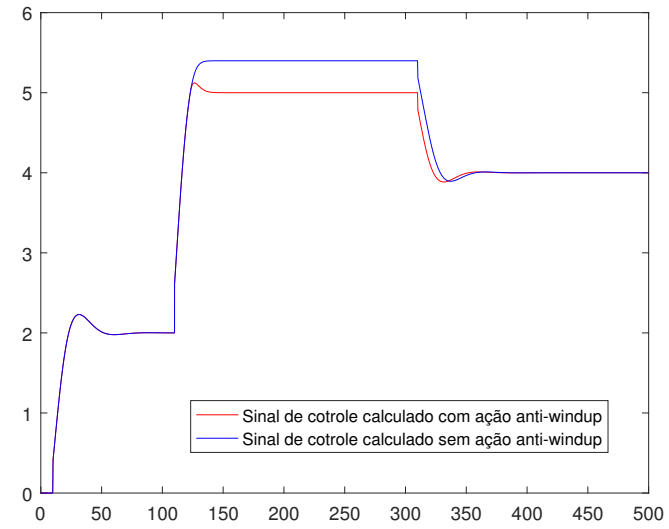


Figura 23: Sinal de controle com e sem anti-windup

4.5 Código Matlab

4.5.1 Código

```
%% Código de Simulação
% Condições iniciais
k_ini = delay1+1;
nit = 400/Ts;
u(1:nit) = 0; yp(1:k_ini) = 0;
y(1:k_ini) = 0; du(1:k_ini) = 0; erro(1:k_ini) = 0;
% Seleciona sintonia do controlador
controle = menu('CONTROLADOR','PI','PID');
% Referência e perturbação
yr(1:10/Ts) = 0; yr(10/Ts+1:110/Ts) = 2;
yr(110/Ts+1:310/Ts) = 5; yr(310/Ts+1:nit) = 1;
do(1:210/Ts) = 0; do(210/Ts+1:nit) = 0.2;
% Discretização da planta
nps = 1;
dps = [32 80 80 40 10 1]; % mdt = 1/(2*s+1)^5
[npz,dpz] = c2dm(nps,dps,Ts,'zoh');
switch controle
    case 1 % PI
        kp = 0.2021; ki = 0.0784; taw = kp/ki;
        % Simulação
        for k = k_ini:nit
        % Saída da planta original
              yp(k) =  -dpz(6)*yp(k-5)-dpz(5)*yp(k-4)-dpz(4)*yp(k-3)-
                        dpz(3)*yp(k-2)-dpz(2)*yp(k-1)+...
                        npz(6)*u(k-5)+npz(5)*u(k-4)+npz(4)*u(k-3)+
                        npz(3)*u(k-2)+npz(2)*u(k-1);
              y(k) = yp(k) + do(k);
        % Lei de controle
              erro(k) = yr(k)-y(k);
              up(k) = kp*erro(k);
              ui(k) = ui(k-1) + ki*Ts*(erro(k)+erro(k-1))/2  + (Ts/taw)*eaw(k-1)
              ub(k) = up(k) + ui(k);
              if ub(k) <= umin;
                 uf(k) = umin;
                 elseif ub(k) >= umax;
                 uf(k) = umax;
                 else
```

```
                uf(k) = ub(k);
            end
            eaw(k) = uf(k) - ub(k);
        end
    case 2 %PID
        kp = 0.181; ki = 0.0736; kd = 0.103; tf = 0.1;
        taw = sqrt(kp/ki*kp*kd);
        % Simulação
        for k = k_ini:nit
        % Saída da planta original
            yp(k) =  -dpz(6)*yp(k-5)-dpz(5)*yp(k-4)-dpz(4)*yp(k-3)-
                      dpz(3)*yp(k-2)-dpz(2)*yp(k-1)+...
                      npz(6)*u(k-5)+npz(5)*u(k-4)+npz(4)*u(k-3)+
                      npz(3)*u(k-2)+npz(2)*u(k-1);
            y(k) = yp(k) + do(k);
        % Lei de controle
            erro(k) = yr(k)-y(k);
            up(k) = kp*erro(k);
            ui(k) = ui(k-1) + ki*Ts*(erro(k)+erro(k-1))/2+ (Ts/taw)*eaw(k-1);
            ud(k) = (erro(k)-erro(k-1))*kd/Ts;
            ub(k) = up(k) + ui(k) + ud(k);
            if ub(k) <= umin;
               uf(k) = umin;
               elseif ub(k) >= umax;
               uf(k) = umax;
               else
               uf(k) = ub(k);
            end
            eaw(k) = uf(k) - ub(k);
        end
end
        % Resultados
        figure(1)
        t = 0:Ts:Ts*(nit-1);
        subplot(2,1,1),plot(t,y,'r',t,yr,'--k','linewidth',2),
                       ylabel('saída e referência'),xlabel('tempo (s)'),grid;
        subplot(2,1,2),plot(t,u,'b','linewidth',2),
                       ylabel('controle'),xlabel('tempo (s)'),grid;
```

4.5.2 Simulação do Código

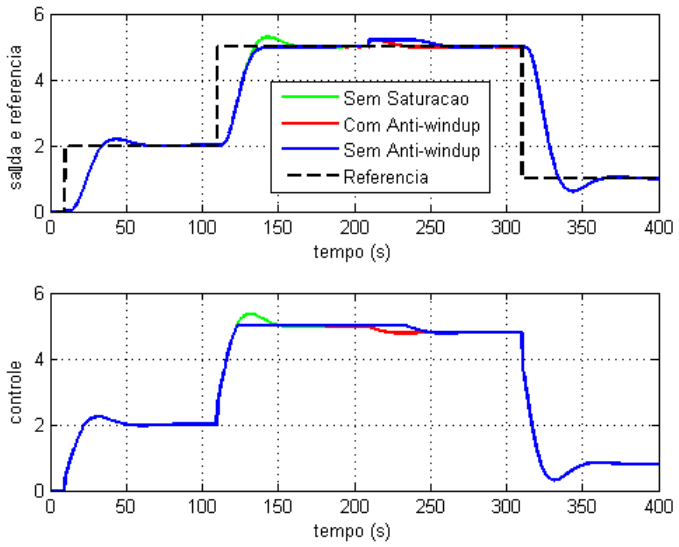


Figura 24: Simulação do sistema em MF com o controlador PI

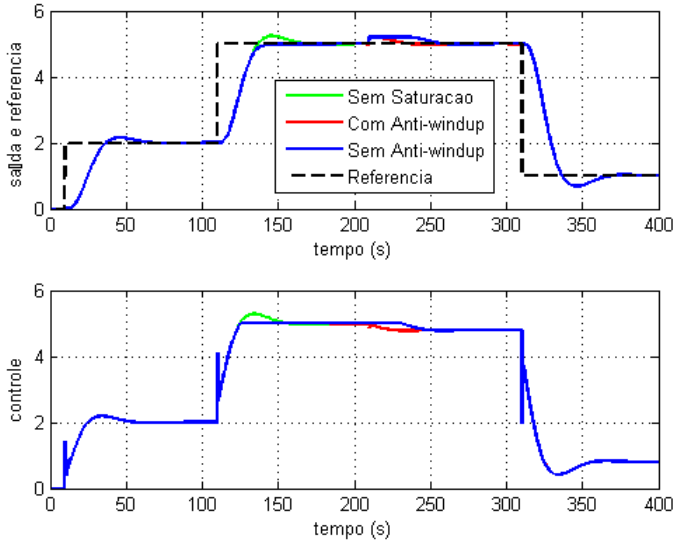


Figura 25: Simulação do sistema em MF com o controlador PID

5 Etapa 5: Implementação PID Novus (HIL) e Preditor de Smith

5.1 Preditor de Smith

Nesta etapa do trabalho da disciplina, foi implementado uma estrutura de controle para atender aos requisitos de desempenho para uma planta com atraso de transporte.

O modelo do processo de quinta ordem que foi analisado neste trabalho, pode ser entendido como um modelo de ordem reduzida mais um atraso (conforme discutido na primeira etapa). Assim, a estrutura do preditor de smith (figura 26) mostra-se vantajosa, já que esta elimina a dependência do atraso na equação característica do sistema em malha fechada.

Dessa forma, um novo projeto via lugar das raízes foi realizado utilizando apenas o modelo rápido (sem o atraso de transporte) das aproximações de ordem reduzida da planta.

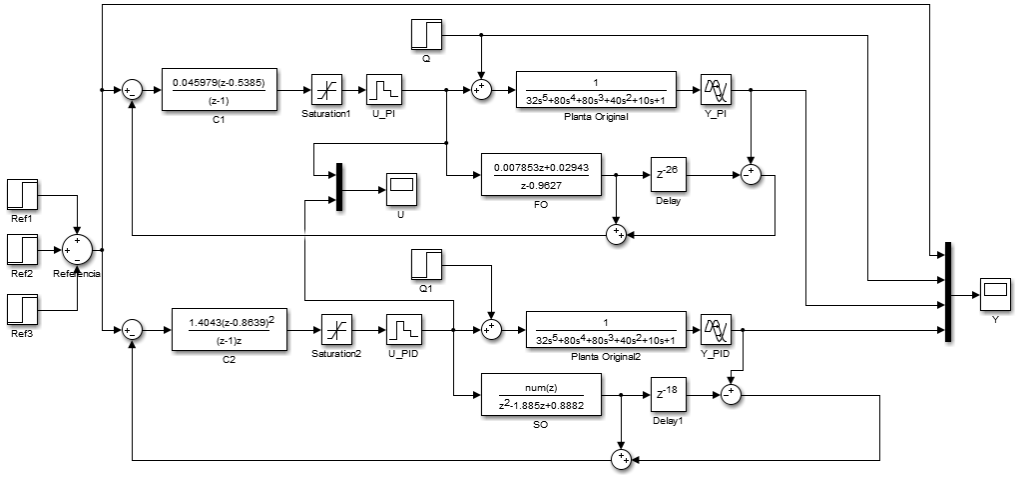


Figura 26: Diagrama de Blocos do Preditor de Smith (Projeto pelo Modelo de Primeira Ordem e pelo Modelo de Segunda Ordem)

5.1.1 Preditor Utilizando Modelo de Primeira Ordem

Utilizando a ferramenta computacional SISO Tools do Matlab, projetamos um controlador primário PI para o Preditor de Smith com base no modelo de primeira ordem sem o atraso de transporte.

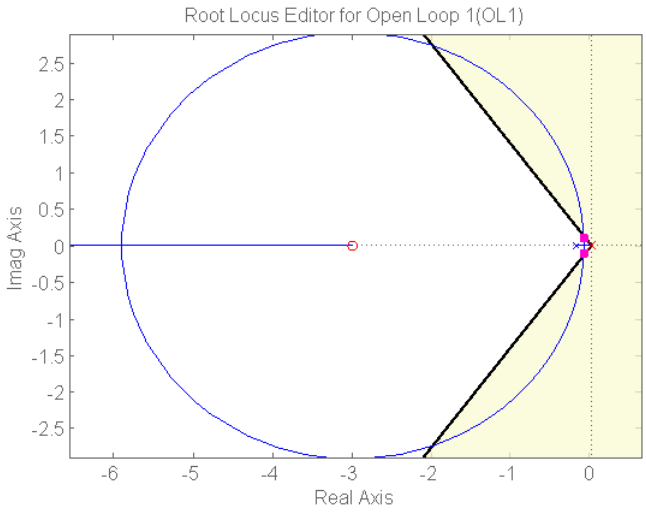


Figura 27: Projeto Lugar das Raízes para o Modelo de Primeira Ordem sem Atraso

Assim, chegamos as seguintes sintonias:

$$C_{PS-PI}(s) = \frac{0.035369\,(s+3)}{s}$$

Discretizando o controlador por Tustin com Ts=0.2s, obtemos:

$$C_{PS-PI}(z) = \frac{0.0045979\,(z-0.5385)}{z-1}$$

5.1.2 Preditor Utilizando Modelo de Segunda Ordem

Utilizando a ferramenta computacional SISO Tools do Matlab, projetamos um controlador primário PID para o Preditor de Smith com base no modelo de segunda ordem sem o atraso de transporte.

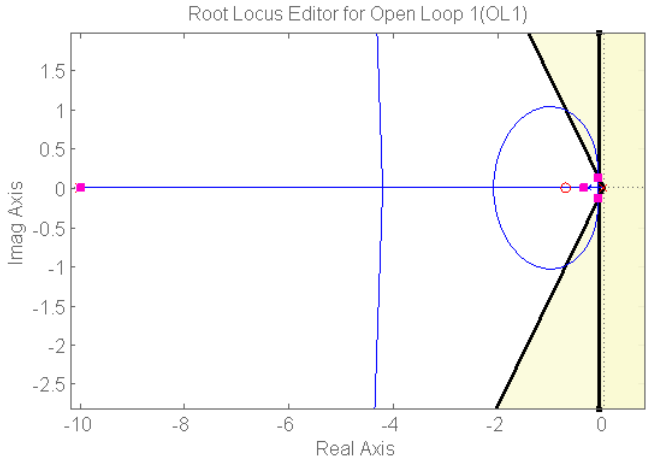


Figura 28: Projeto Lugar das Raízes para o Modelo de Segunda Ordem sem Atraso

Assim, chegamos as seguintes sintonias:

$$C_{PS-PID}(s) = \frac{2.4394\,(s+0.73)^2}{s\,(s+10)}$$

Discretizando o controlador por Tustin com Ts=0.2s, obtemos:

$$C_{PS-PID}(z) = \frac{1.4043\,(z-0.8639)^2}{z\,(z-1)}$$

5.1.3 Simulação Diagrama de Blocos

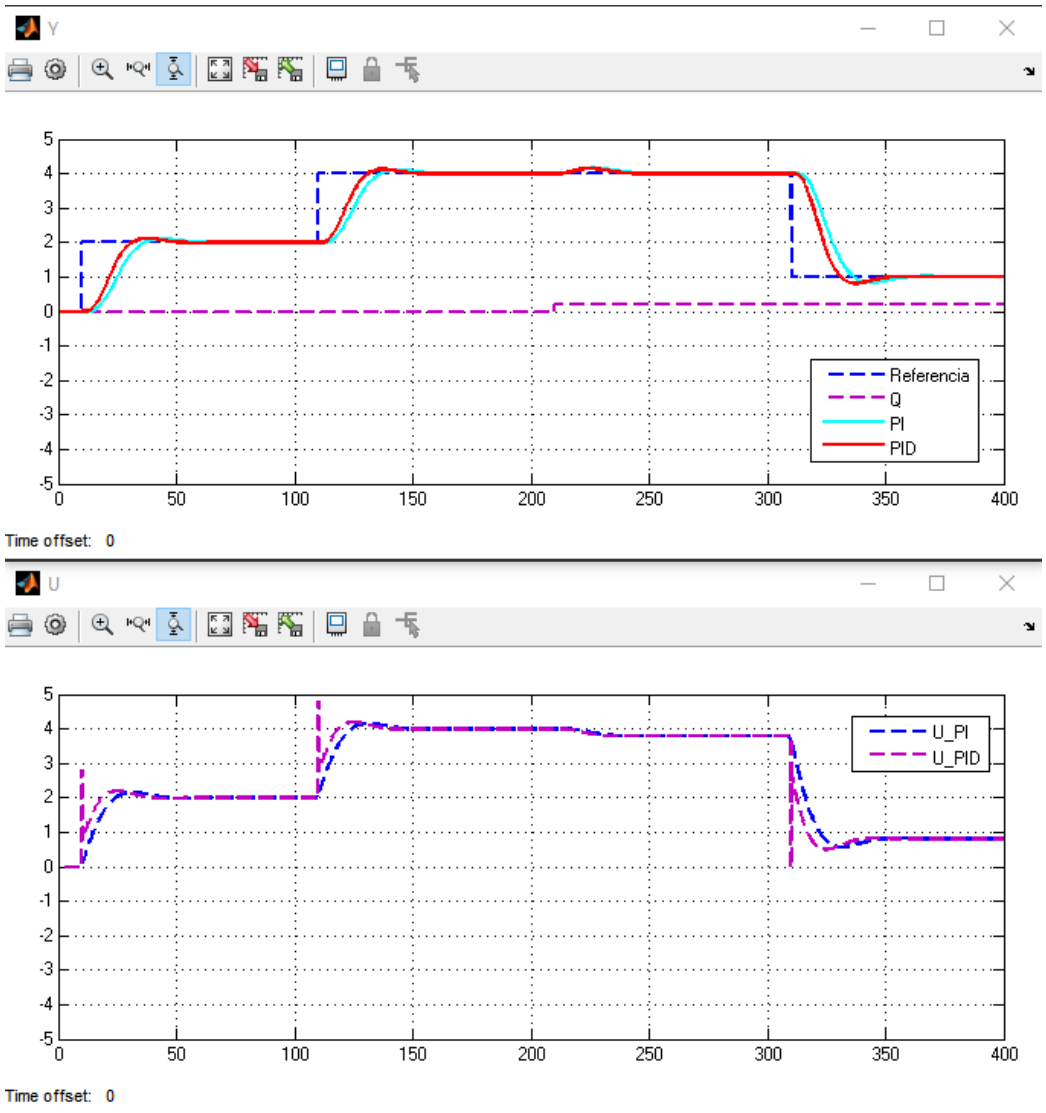


Figura 29: Simulação das Estruturas dos Preditores de Smith Através do Simulink para 3 Mudanças de Referência e uma Perturbação Degrau

5.1.4 Código Matlab

```
Ts=0.2
% Modelos Aproximados da Planta - Contínuo
fopdt = tf([1],[5.2637 1],'InputDelay',5.1585);
% Modelos Aproximados da Planta - Discretização
```

```
foz = c2d(fopdt,Ts)
n_fo = foz.num{1}; d_fo = foz.den{1}; delay1 = foz.InputDelay;
soz = c2d(sopdt,Ts)
n_so = soz.num{1}; d_so = soz.den{1}; delay2 = soz.InputDelay;
% Condições iniciais
nit = 400/Ts; umax = 4.9; umin = 0;
d = delay1; d2 = delay2;
% Parâmetros da Planta
nps = 1; dps = [32 80 80 40 10 1]; % mdt = 1/(2*s+1)^5
[npz,dpz] = c2dm(nps,dps,Ts,'zoh');
% Inicialização
up(1:nit) = 0; ui(1:nit) = 0; ud(1:nit) = 0; eaw(1:nit) = 0;
ub(1:nit) = 0; uf(1:nit) = 0; u(1:nit) = 0; % Sinal de Controle
e(1:nit) = 0;           % Erro
y(1:nit) = 0;           % Saída da Planta
epred(1:nit) = 0;       % Erro de Predição
gn(1:nit) = 0;          % Predição
pn(1:nit) = 0;          % Predição Atrasada
yt(1:nit) = 0;          % Y utilizado para calcular o erro
% Referência
r(1:110/Ts) = 2; r(110/Ts+1:310/Ts) = 4; r(310/Ts+1:nit) = 1;
% Perturbação
q(1:210/Ts) = 0; q(210/Ts+1:nit) = 0.2;
% Parâmetros do Controlador PI
kp = 0.0354; ki = 0.106;  taw = kp/ki*0.5; kd=0;
for k = d+2:nit
      u(k) = uf(k-1)+q(k);
      % Planta Original
        y(k) = -dpz(6)*y(k-5)-dpz(5)*y(k-4)-dpz(4)*y(k-3)...
            -dpz(3)*y(k-2)-dpz(2)*y(k-1)+npz(6)*u(k-5)+...
            npz(5)*u(k-4)+npz(4)*u(k-3)+npz(3)*u(k-2)+npz(2)*u(k-1);
      % Predição
        gn(k) = -d_fo(2)*y(k-1)+n_fo(1)*uf(k)+n_fo(2)*uf(k-1);
        pn(k) = gn(k-d);
        epred(k) = y(k) - pn(k);
        yt(k) = epred(k) + gn(k);
        e(k) = r(k) - yt(k);
      % Lei de Controle (Discretização por Tustin)
        up(k) = kp*e(k);
        ui(k) = ui(k-1) + ki*Ts*(e(k)+e(k-1))/2  + (Ts/taw)*eaw(k-1);
        ud(k) = (e(k)-e(k-1))*kd/Ts;
```

```
        ub(k) = up(k) + ui(k) + ud(k);
      % Saturação e Anti-windup
             if ub(k) <= umin;
                uf(k) = umin;
                elseif ub(k) >= umax;
                uf(k) = umax;
                else
                uf(k) = ub(k);
             end
             eaw(k) = uf(k) - ub(k);
end
% Parâmetros do Controlador PID
kp = 0.343; ki = 0.13; kd=0.21; tf = 0.1; taw = sqrt(kp/ki*kp*kd);
for k = d2+2:nit
      u(k) = uf(k-1)+q(k);
      % Planta Original
        y(k) = -dpz(6)*y(k-5)-dpz(5)*y(k-4)-dpz(4)*y(k-3)...
            -dpz(3)*y(k-2)-dpz(2)*y(k-1)+npz(6)*u(k-5)+...
            npz(5)*u(k-4)+npz(4)*u(k-3)+npz(3)*u(k-2)+npz(2)*u(k-1);
      % Predição
        gn(k) = -d_so(2)*y(k-1)-d_so(3)*y(k-2)+n_so(1)*uf(k)+...
                n_so(2)*uf(k-1)+n_so(3)*uf(k-2);
        pn(k) = gn(k-d2);
        epred(k) = y(k) - pn(k);
        yt(k) = epred(k) + gn(k);
        e(k) = r(k) - yt(k);
      % Lei de Controle (Discretização por Tustin)
        up(k) = kp*e(k);
        ui(k) = ui(k-1) + ki*Ts*(e(k)+e(k-1))/2  + (Ts/taw)*eaw(k-1);
        ud(k) = (e(k)-e(k-1))*kd/Ts;
        ub(k) = up(k) + ui(k) + ud(k);
      % Saturação
             if ub(k) <= umin;
                uf(k) = umin;
                elseif ub(k) >= umax;
                uf(k) = umax;
                else
                uf(k) = ub(k);
             end
             eaw(k) = uf(k) - ub(k);
end
```

5.1.5 Simulação do Código

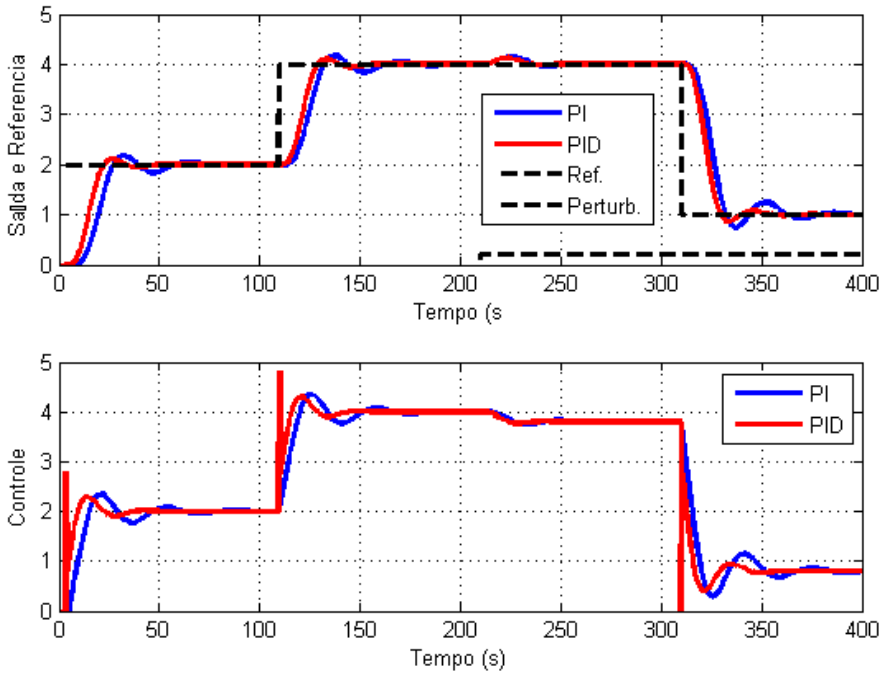


Figura 30: Simulação das Estruturas dos Preditores de Smith Através do Código do Matlab para 3 Mudanças de Referência e uma Perturbação Degrau

5.2 Implementação PID Novus (HIL)

5.2.1 Código Matlab

```
% -------------------------
% Controle PI - NOVUS
% Ambiente de Simulação: HIL
% Planta: 2a Ordem
% -------------------------
clear all; close all; clc;
% Condições Iniciais
nit = 600; ts = 0.2; umax = 4.9; umin = 0;
u(1:nit) = 0; y(1:nit) = 0; yr(1:nit) = 2.5;
% Discretização da Planta
nps = [1]; dps = [32    80    80    40    10     1];
[npz,dpz] = c2dm(nps,dps,ts,'zoh');
a1 = dpz(2);
a2 = dpz(3);
b0 = npz(2);
b1 = npz(3);

% Inicialização da Malha Fechada
inicializa_placa(5);
% Simulação
for t = 7:nit
% Recebe Controle
     u(t) = recebe_dado(1);
% Calcula Medição
       y(t) = - dpz(2)*y(t-1) - dpz(3)*y(t-2) - dpz(4)*y(t-3) -
                dpz(5)*y(t-4) - dpz(6)*y(t-5) + npz(2)*u(t-1) +
                npz(3)*u(t-2)+ npz(4)*u(t-3) + npz(5)*u(t-4) +
                npz(6)*u(t-5);
% Envia Saída
     envia_dado(1,y(t)); atraso_ms(1000*ts);
end
finaliza_placa;
% Resultados
figure(1)
t = 0:ts:(nit-1)*ts;
subplot(2,1,1),plot(t,yr,'k',t,y,'r','linewidth',2),
               ylabel('saída e referência'),xlabel('tempo (s)'),grid on;
subplot(2,1,2),plot(t,u,'b','linewidth',2),
               ylabel('controle'),xlabel('tempo (s)'),grid on;
```

5.2.2 Resultados

Inicialmente se calcula as constantes que se deve embarcar no Novus, PB e Ir, utilizando as constantes Kc e Ti que já tinham sido calculados para o PI anteriormente.

$$PB = \frac{100}{Kc}$$

$$Ir = \frac{60}{Ti}$$

Após se embarcar os valores corretos :

$$PB = \frac{100}{0.2021} = 494.80$$

$$Ir = \frac{60}{2.5793} = 23.26$$

Roda-se o código de matlab acima que simula a nossa planta para testar com o controlador da Novus calibrado.O resultado da simulação se encontra na figura abaixo.

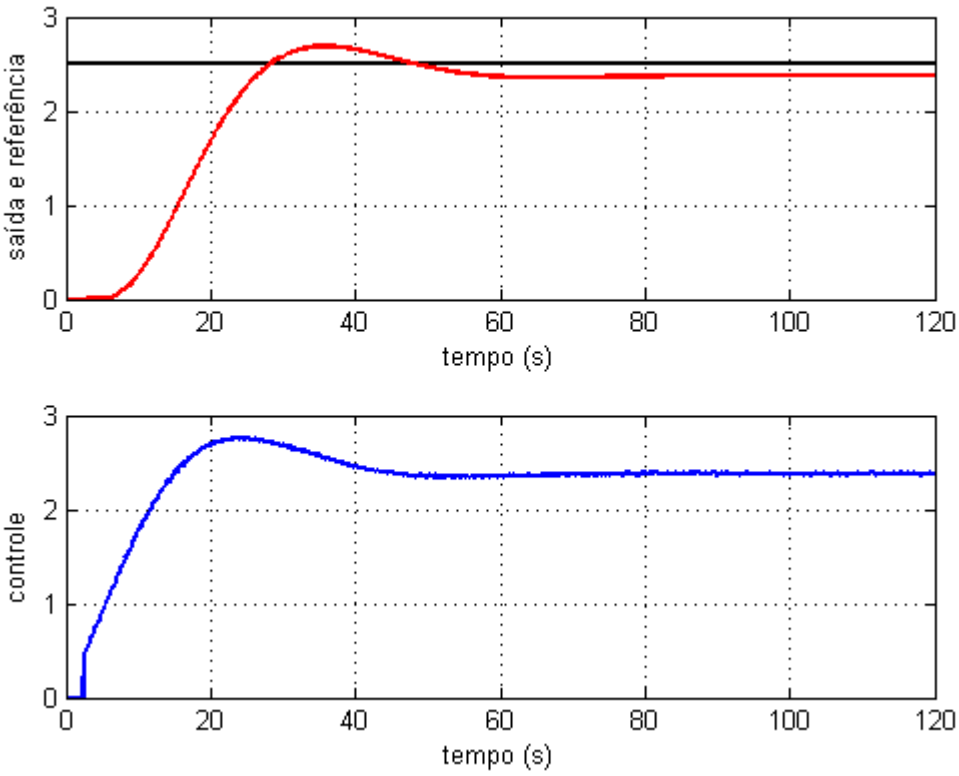


Figura 31: Resposta da planta utilizando HIL

Se comparar o PI da figura 20 com o da figura 31 se observa que se obteve a mesma resposta, porém com um pequeno erro no seguimento da referência devido as perdas no HIL.

6 Conclusão

6.1 Sobre o Trabalho

Nesta quinta etapa do trabalho, efetuamos os objetivos opcionais do trabalho: Implementação do controlador PID Novus (HIL), inserir atraso de transporte no sistema e aplicar o compensador preditor de Smith. Além disso, realizamos um compilado de todos os resultados obtidos no decorrer do curso.

Com relação aos resultados obtidos na simulação do preditor de Smith, no caso particular do controlador PID, verificou-se a presença de um *kick* no sinal de controle durante as mudanças de referência. O mesmo resultado já havia sido verificado no projeto via lugar das raízes da etapa 4 e nas sintonias do PID via métodos da literatura. Essa resposta abrupta do sinal de controle é causada pela ação derivativa do controlador. Uma solução para trabalhos futuros poderia ser a implementação de controladores com mais um grau de liberdade, configurando-se em uma estrutura PI+D, por exemplo.

6.2 Auto-avaliação do Grupo

O grupo se empenhou para cumprir o prazo de entrega e atender todos os requisitos solicitados nas especificações das etapas do trabalho. O desenvolvimento da parte técnica da atividade, assim como a elaboração de sua documentação foi fruto de um esforço colaborativo de todos os integrantes do grupo.