# 3 Implementação de um controlador PID digital (Matlab)

## *3.1 Algoritmo PID ideal*

Discretizando a equação do algoritmo PID de posição:

$$m(t) = k_c e(t) + \frac{k_c}{T_i}\int_0^t e(t)dt + k_c T_d \frac{de(t)}{dt} \quad \text{(3-1)}$$

obtém-se o sinal de controlo discreto, $m_k$, a partir de:

$$m_k = k_p e_k + k_i S_k + k_d (e_k - e_{k-1}) \quad \text{(3-2)}$$

onde:

$$S_k = S_{k-1} + e_k \quad \text{(3-3)}$$

e os novos parâmetros estão relacionados com os analógicos da seguinte forma:

$$\begin{cases} k_p = k_c \\ k_i = k_c \dfrac{\Delta T}{T_i} \\ k_d = k_c \dfrac{T_d}{\Delta T} \end{cases} \quad \text{(3-4)}$$

Pode-se também derivar o algoritmo de velocidade a partir de:

$$\Delta m_k = m_k - m_{k-1} \quad \text{(3-5)}$$

Este tem como saída o incremento (ou decremento) no valor da variável manipulada em cada instante ($\Delta m_k$) e não o valor absoluto desta variável ($m_k$):

$$\Delta m_k = e_k (k_p + k_i + k_d) - e_{k-1}(k_p + 2k_d) + e_{k-2} k_d \quad \text{(3-6)}$$

Uma implementação do algoritmo de posição, no formato de S-function (Matlab):

```
function [sys, x0, str, ts] = pid_pos (t, x, u, flag, ts, kc, Ti, Td)
persistent s ek_1

if flag == 0                    %inicialização
    sys = [0 0 1 1 0 1 1];
    x0 = [ ];
    str = [ ];
    ts = [-2 0];                %tempo de amostragem variável
    s=0;

elseif flag == 4                %Calcula próximo instante de amostragem
      ns = t / ts;                                  %ns nº de amostras
      sys = (1 + floor(ns + 1e-13*(1+ns)))*ts;  %momento próxima amostra

elseif flag == 3
    if t==0 ek_1=u(1); end;

    kp=kc; ki=kc*ts/Ti; kd=kc*Td/ts;          %Converte parametros

    ek=u(1);
    s = s + ek;
    m=kp*ek+ki*s+kd*(ek-ek_1);

    ek_1=u(1);
    sys = m;

else                            %por defeito não retorna nada
      sys = [ ];
end
```

Algoritmo 3-1: PID de posição em Matlab (S-function)

Repare-se que esta função assume como parâmetros de entrada kc, Ti e Td e converte-os para os parâmetros kp, ki e kd.

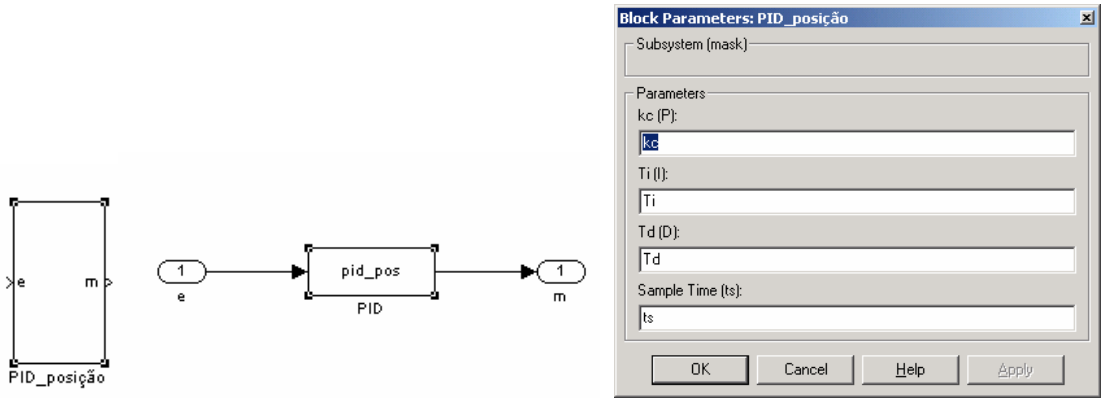



Fig. 3-1: Diagrama blocos do Controlador PID discreto (posição)

Criar o bloco PID de posição como descrito acima num modelo. A partir da janela principal do Matlab: **File->new->Model.**

Os blocos podem ser guardados numa biblioteca para uso futuro. A partir da janela do Modelo: **File->new->Library** abre uma janela de biblioteca. Pode-se agora arrastar o bloco **PID_posição** para a biblioteca.

Na biblioteca além dos blocos devem ser guardadas também as s-functions correspondentes. Para isso criar um directório, por ex. **pid_lib**, e guardar nesse directório a biblioteca com o nome **pid_lib**. Copiar agora a s-function **pid_pos.m** para esse mesmo directório.

Para que esse directório fique registado no caminho do Matlab, de modo que as s-functions estejam acessíveis a partir da janela principal do Matlab: **File->Set path**, e depois adicionar ao caminho o directório onde está a biblioteca: **pid_lib**.

Pode-se agora obter o controlador PID de velocidade a partir do controlador de posição, Algoritmo 3-1. Na janela da biblioteca:

#1 copiar o bloco **PID_Posição** e renomeá-lo para **PID_velocidade**.

#2 Abrir a máscara e mudar o nome da s-function para **pid_vel**

#3 Copiar o ficheiro com a s-function **pid_pos.m** para **pid_vel.m** e alterar o código para:

```
function [sys, x0, str, ts] = pid_vel (t, x, u, flag, ts, kc, Ti, Td)

persistent ek_    % passa a ter só uma variável persistente:
(...)
if flag == 0
        % apagar a inicialização da variável: s = 0
(...=
elseif (flag == 3)      % substituir por:
    if t==0 ek_=ones(2,1)*u(1); end          %ek_(1) -> ek-1, ek(2) -> ek-2

    kp=kc; ki=kc*ts/Ti; kd=kc*Td/ts;         %Converte parametros

    ek=u(1);
    dm=ek*(kp+ki+kd)-ek_(1)*(kp+2*kd)+ek_(2)*kd;

    ek_(2)=ek_(1);
    ek_(1)=ek;
    sys = dm;
```

Algoritmo 3-2: PID de velocidade (alterações ao Algoritmo 3-1 PID de posição )

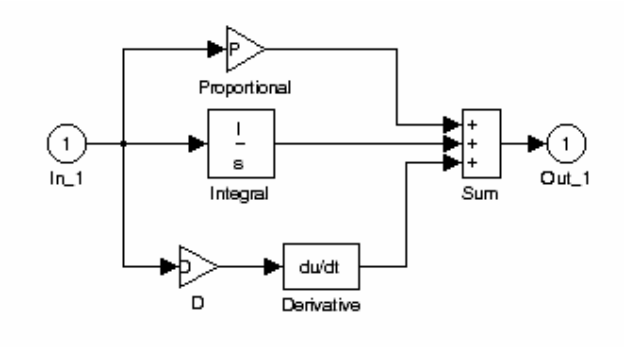
**Nota:** se não fôr possível efectuar alterações na biblioteca, é porque esta está protegida. Para a desproteger, na janela da biblioteca **Edit->Unlock Library.**

### 3.1.1 Resposta ao degrau do algoritmo PID

Vai-se agora analisar a resposta ao degrau dos controladores PID analógico e digital.

Primeiro copiar para a biblioteca o controlador analógico fornecido pela biblioteca do Simulink. Para isso aceder à biblioteca do simulink digitando **simulink <enter>** na janela principal do Matlab e arrastar o PID controller para a biblioteca **pid_lib**.

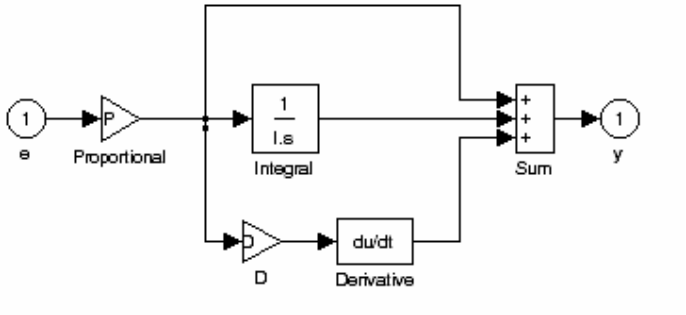
Agora ao ver por baixo da máscara a implementação deste controlador têm-se:




$$(3\text{-}7)\quad \begin{cases} P = k_c \\ I = \dfrac{k_c}{T_i} \\ D = T_d k_c \end{cases}$$

Fig. 3-2: PID analógico Matlab

No diagrama acima as constantes têm outra designação da já referida, sendo a correspondência dada por (3-7):




$$(3\text{-}8)\quad \begin{cases} P = k_c \\ I = T_i \\ D = T_d \end{cases}$$

Fig. 3-3: PID analógico segundo equação (3-1)

A versão da Fig. 3-2 é uma implementação diferente do controlador já visto, pois não segue a equação (3-1), visto que a constante (Kc = P) apenas é aplicada ao termo proporcional, e não a todos os 3 termos, e a constante de tempo integral está invertida.

Para alterar esta implementação para que respeite a equação (3-1) pode-se alterar o diagrama da Fig. 3-2 de acordo com a Fig. 3-3 e renomear o bloco para **PID_analógico)**. Repare-se que manipulando as constantes Porporcional, Integral e Derivativa como indicado na equação (3-7), ambos os modelos têm respostas idênticas.

Nesta disciplina vamos assumir que o modelo de PID é o dado pela equação (3-1) ou Fig. 3-3, pois os métodos de sintonia do PID estudados assumem este modelo.

Para comparar as diferentes implementações do algoritmo PID na resposta ao degrau desenhar o diagrama abaixo:

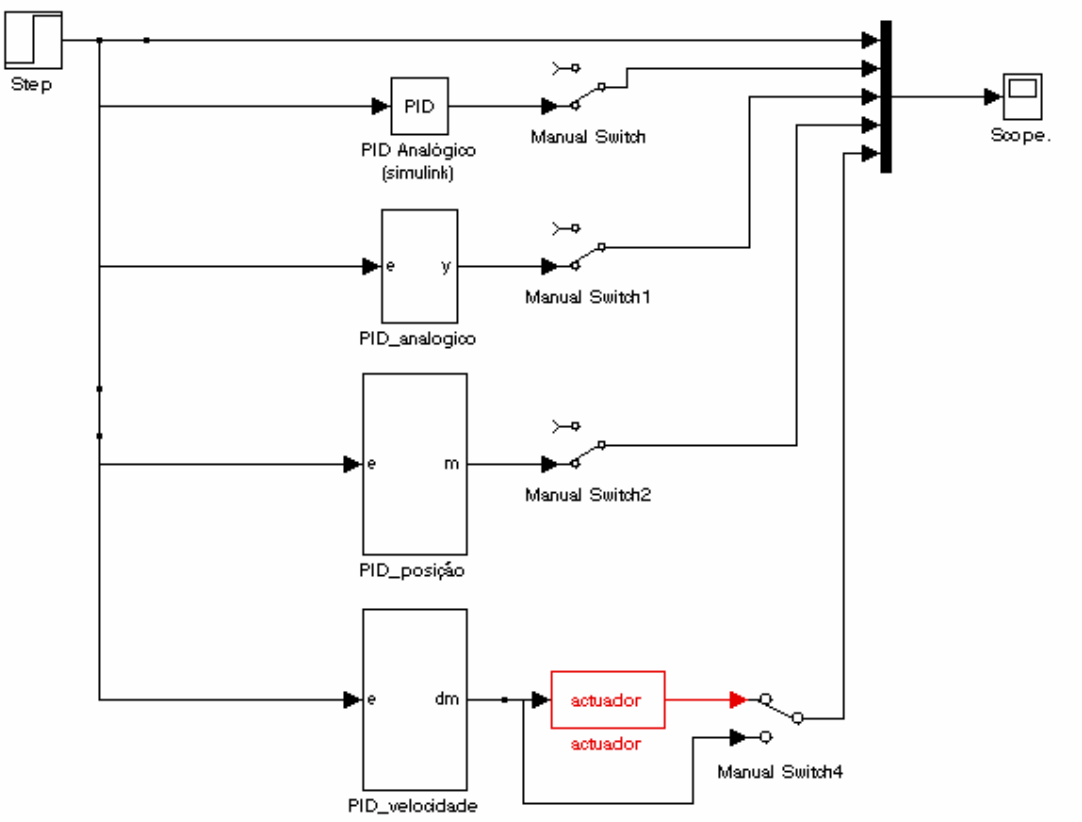



Fig. 3-4: Diagrama blocos para resposta ao degrau do controlador PID

Para já não inclua o bloco a vermelho: actuador. Os 4 manual switches permitem ligar o algoritmo correspondente de modo a apresentar no scope a saída deste.

Em cada bloco PID, colocar nos parâmetros Porporcional (Kc), Integral (Ti) e Derivativo (Td), respectivamente as variáveis: **Kc**; **Td** e **TI**. Nos blocos PID discretos colocar também

no parâmetro Sample Time a variável **ts**. No bloco step colocar no valor inicial do degrau 0 e no valor final do degrau a variável **uf**. O degrau ocorre no instante de tempo 1 segundo.

Estas variáveis deverão ser definidas no workspace do Matlab e permitem definir iguais parâmetros para todos os controladores do diagrama.

Gravar agora o modelo com o nome **pid00.mdl**. Este modelo **não** deve ser gravado no directório onde está a biblioteca **pid_lib**. Escolha outro directório, p. ex **STR_aula3**.

Neste directório vai-se também criar um script que permita lançar o modelo e definir os parâmetros do controlador. Este ficheiro vai chamar-se **pid_step.m**:

```
uf = 1              %Amplitude do degrau
Kc = 0.5            %define os parâmetros do controlador PID
Ti = 2
Td = 1
Ts = 0.25
pid00               %Abre modelo
```

Após executar a simulação deve-se obter no scope a seguinte figura:

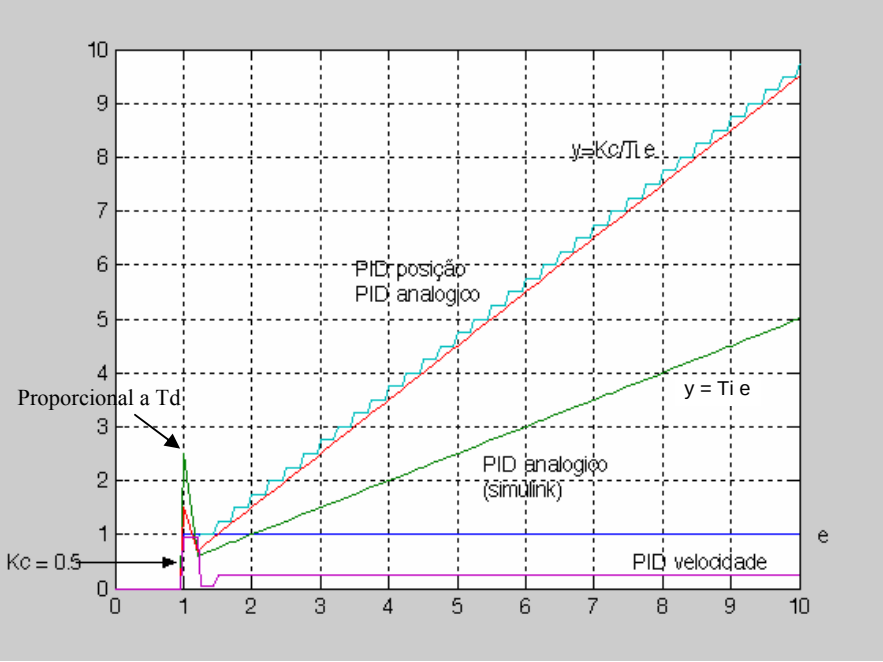



Fig. 3-5: Resposta ao degrau unitário do PID analógico e do discreto

Como se pode observar a resposta das implementações analógica e digital de posição são muito semelhantes. Se se pretender que a resposta do controlador digital seja mais rectilínea deve-se reduzir o intervalo de amostragem. Experimentar alterar Ts = 0.1 e executar outra vez a simulação.

A implementação analógica que acompanha a distribuição do simulink é uma rampa com menor declive, pois o declive da rampa neste caso é dado por $y = Ti\,e$, e não por $y = \frac{Kc}{Ti}e$.

Repare-se que o pico inicial na resposta é proporcional ao termo derivativo, e por isso cresce com Td, de modo que alterando Td para 0 e executando a simulação o pico não existe.

Assim o termo derivativo pode ser sintonizado para compensar uma resposta inicial do sistema demasiado amortecida, no entanto um valor demasiado grande pode provocar ligeiras oscilações na resposta do sistema, impedindo-o mesmo de estabilizar.

Já a resposta do PID_velocidade não é uma rampa, mas sim uma recta paralela ao degrau de entrada, pois constituí um incremento na posição do actuador e não a posição absoluta.

Para ensaiar o PID_velocidade vai-se simular um actuador. Este será implementado no formato de uma s-function, **actuador.m**, onde os parâmetros serão o intervalo de amostragem (ts), os valores de posição inicial (pini), mínimo (pmin) e máximo (pmax) do actuador:

```
Function [sys, x0, str, ts]=actuador (t, x, u, flag, ts, pmin, pmax, pini)

persistent act      %valor corrente do actuador [pmin, pmax].

if flag == 0
    sys = [0 0 1 1 0 1 1];
    x0 = [ ];
    str = [ ];
    ts = [-2 0];              %tempo de amostragem variável

    %Garante valores de posição máximo mínimo e inicial coerentes
    if pmin>pmax error ('Erro: Valor pmin tem de ser <= pmax!');
    elseif pini>pmax act = pmax;
    elseif pini<pmin act = pmin;
    else act=pini;                  %inicializa posiçãoo do actuador
    end

elseif flag == 4
      ns = t / ts;
      sys = (1 + floor(ns + 1e-13*(1+ns)))*ts;

elseif flag == 3

    act=act+u(1);                   %actualiza posição do actuador
    if act>pmax act = pmax;         %garantido que não ultrapassa
    elseif act<pmin act = pmin;     %os limites fisicos pmin e pmax
    end

    sys = act;

else
      sys = [ ];
end
```

Algoritmo 3-3: Actuador (S-function)

Mascarar o bloco do actuador de acordo com a figura seguinte:

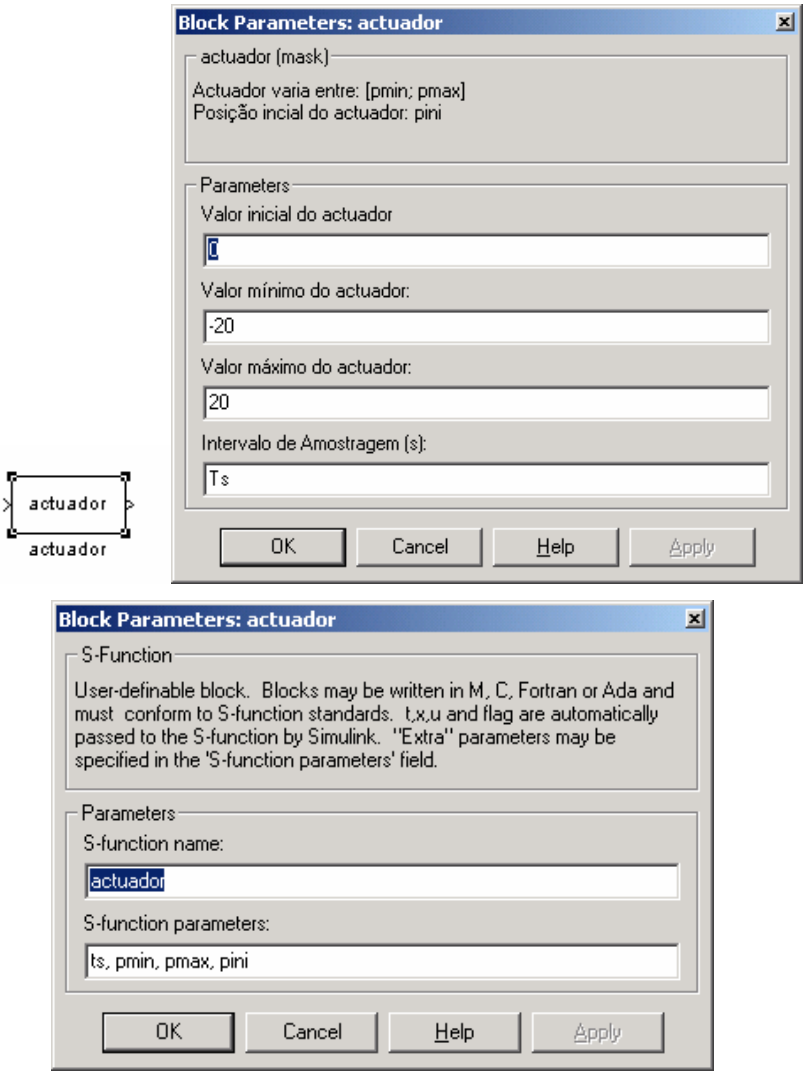



Fig. 3-6: Definição do actuador como s-function

Para já definir no actuador o valor inicial como 0, o mínimo como -20 e o máximo como 20 e o intervalo de amostragem a variável **Ts**.

**Nota:** Atenção que nos actuadores pode ser considerado um valor mínimo negativo e simétrico do máximo sendo a posição central 0, ou então um valor mínimo nulo e igual ao inicial.

Arrastar o bloco actuador para a biblioteca. Não esquecer que a s-function **actuador.m** tem de estar guardada no directório da biblioteca **pid_lib**.

Alternativamente o actuador poderia ser desenvolvido como uma montagem de blocos simulink e mascarando o sub-sistema tal como acima:

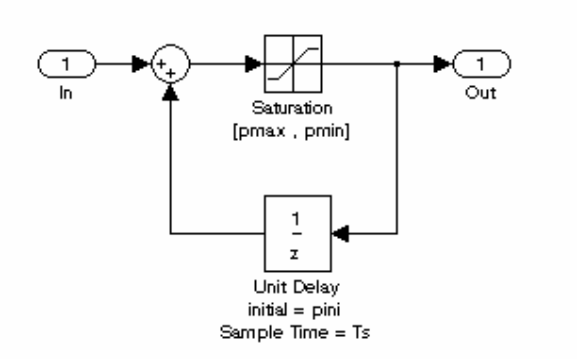



Fig. 3-7: Diagrama blocos do actuador

Colocar o actuador no modelo **pid00.mdl** na posição indicada na Fig. 3-4. Executar outra vez a simulação. Agora a resposta do PID_velocidade é igual ao PID_posição.

Se se pretender simular que os limites físicos do actuador foram ultrapassados, alterar o valor máximo do actuador para 5 e voltar a executar a simulação. Agora a resposta do PID_velocidade não ultrapassa esse valor:

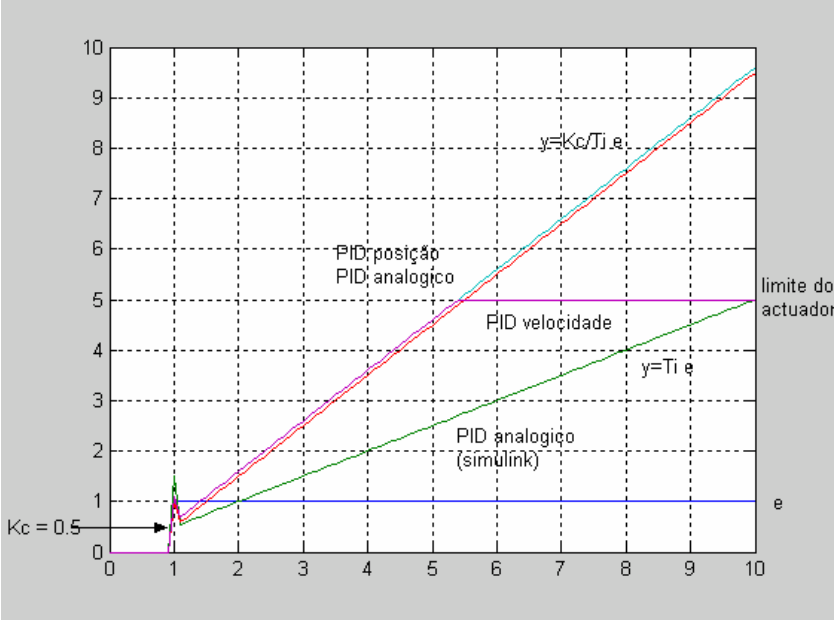



Fig. 3-8: Resposta ao degrau unitário do PID velocidade utilizando um actuador

### 3.1.2 Sintonia do controlador PID

A sintonia utilizada no exemplo anterior foi efectuada de acordo com o método de Ziegler Nichols para um controlador analógico e convertida para parâmetros equivalentes no controlador Digital, no entanto esta pode ser efectuada também directamente.

Se não for conhecida a função de transferência da plant esta poderá ser obtida através da curva de reacção do processo, isto é da resposta em **malha aberta** ao degrau.

Assumindo que se tem o seguinte sistema de 1ª ordem, com atraso de 0.6 segundos e como entrada um degrau de amplitude 2:

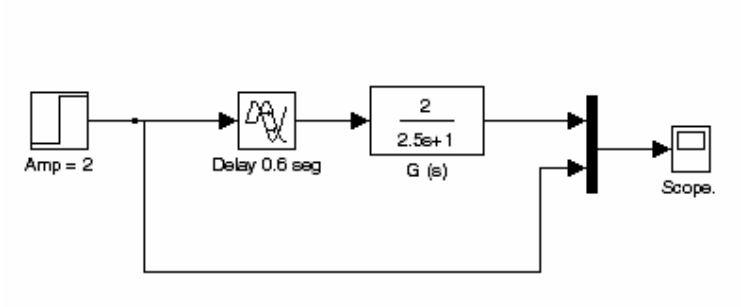



Fig. 3-9: Plant com atraso de 0.6 segundos e entrada de degrau com amplitude 2

Para processos de 1ª ordem, a curva de reacção do processo segue o padrão:

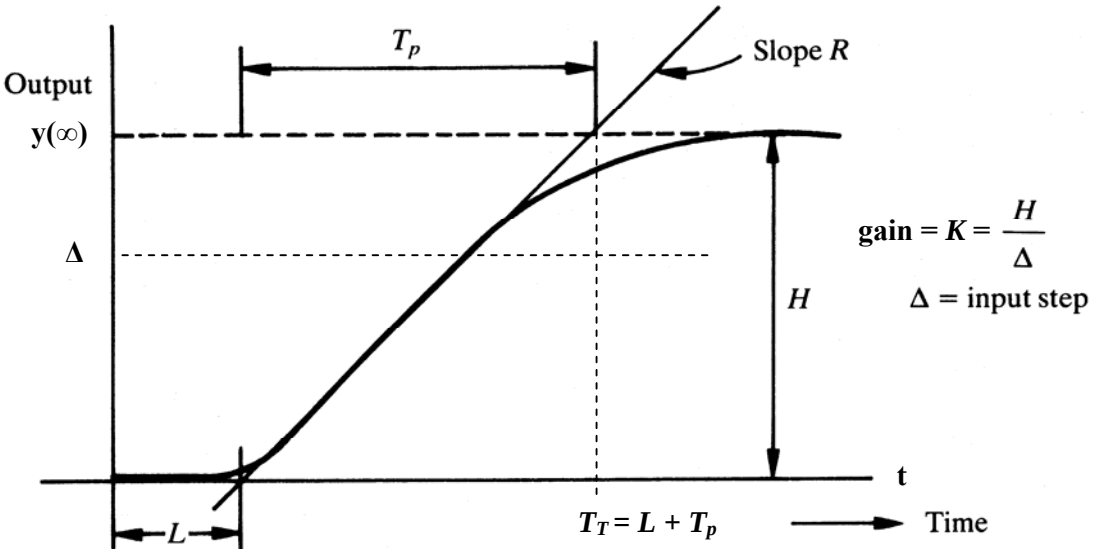



Fig. 3-10: **Curva de reacção do processo**

Neste caso particular será:

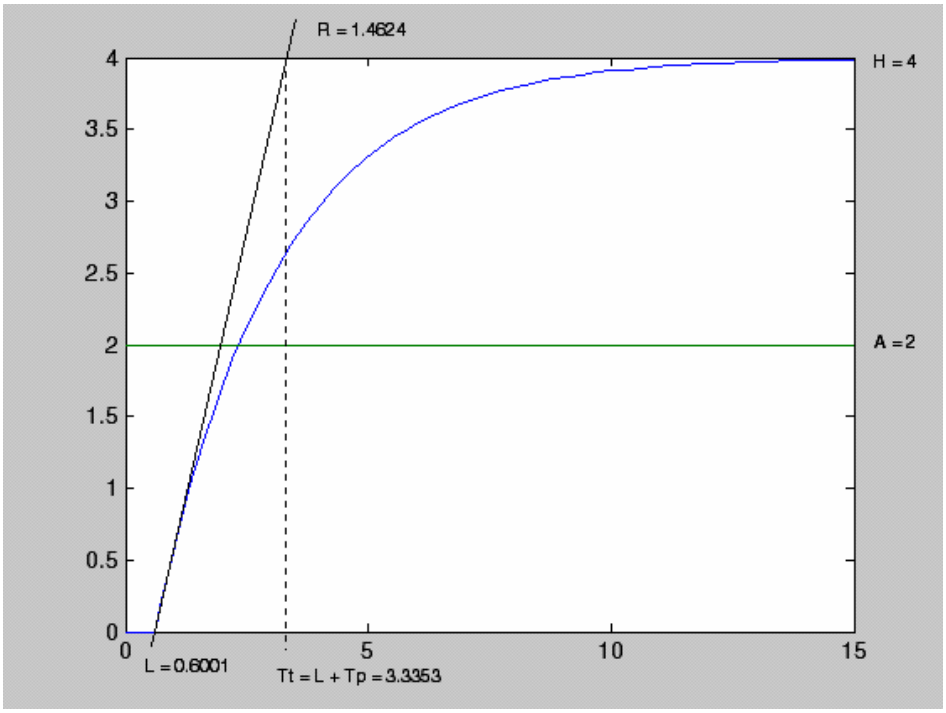



Fig. 3-11: Resposta ao degrau de amplitude 2 da Plant do diagrama anterior

O valor de L pode ser obtido fazendo com que os dados do scope sejam escritos para o workspace do Matlab, e verificando em que instante de tempo é que a resposta deixa de ser nula.

Já o tempo em que a recta com declive R intersecta a resposta final pode ser lido através da figura com o comando **ginput**, e resolvendo:

(3-9) $Tp = Tt - L = 3.3353 - 0.6001 = 2.7351$

Assim pode-se obter o declive da recta, isto é R:

(3-10) $$R = \frac{y_0 - y_1}{x_0 - x_1} = \frac{0-4}{0.6001-3.3353} = \frac{-4}{-2.7352} = 1.4624$$

E o ganho:

(3-11) $$k = \frac{H}{\Delta} = \frac{4}{2} = 2$$

Sendo a função de transferência dada por:

$$(3\text{-}12)\quad G(s) = \frac{k\, e^{-L s}}{T_p\, s + 1} = \frac{2\, e^{-0.6001 s}}{2.7352\, s + 1}$$

Obviamente que neste caso a função de transferência correcta é conhecida:

$$(3\text{-}13)\quad G(s) = \frac{k\, e^{-L s}}{T_p\, s + 1} = \frac{2\, e^{-0.6 s}}{2.5\, s + 1}$$

com L = 0.6 e:

$$(3\text{-}14)\quad R = \frac{y_0 - y_1}{x_0 - x_1} = \frac{0 - 4}{0.6 - 2.5 + 0.6} = \frac{-4}{-2.5} = 1.6$$

Assim vai-se comparar a sintonia do PID para a função de transferência estimada (3-12) e para a correcta, dada pela equação (3-14).

Agora a sintonia do PID pela regras de Ziegler-Nichols é dada por:

$$(3\text{-}15)\quad \begin{cases} k_c = \dfrac{1.2}{R\, L} \\ T_i = 2\, L \\ T_d = 0.5\, L \end{cases}$$

e de acordo com Goff:

$$(3\text{-}16)\quad T_s = 0.3\, L$$

usando a transformação para parâmetros digitais (3-4), Takahashi propôs a seguinte regra:

$$(3\text{-}17)\quad \begin{cases} k_p = \dfrac{1.2}{R\,(L + T_s)} \\ k_i = \dfrac{0.6\, T_s}{R\,(L + T_s/2)^2} \\ \dfrac{0.5}{R T_s} \le k_d \le \dfrac{0.6}{R T_s} \end{cases}$$

Para a função estimada com $T_s = 0.3 * 0.6001 = 0.18003$, têm-se então:

$$(3\text{-}18)\begin{cases} k_p = \dfrac{1.2}{1.4624*(0.6+0.18003)} = \mathbf{1.0520} \\ k_i = \dfrac{0.6*0.18003}{1.4624*(0.6+0.18003/2)^2} = \mathbf{0.1551} \\ \dfrac{0.5}{1.4624*0.18003} \le k_d \le \dfrac{0.6}{1.4624*0.18003} \Leftrightarrow \mathbf{1.8991} \le k_d \le \mathbf{2.279} \quad \text{(média 2.089)} \end{cases}$$

e para a correcta, com $T_s = 0.3 * 0.6 = 0.18$:

$$(3\text{-}19)\begin{cases} k_p = \dfrac{1.2}{1.6*(0.6+0.18)} = \mathbf{0.9615} \\ k_i = \dfrac{0.6*0.18}{1.6*(0.6+0.18/2)^2} = \mathbf{0.1418} \\ \dfrac{0.5}{1.6*0.18} \le k_d \le \dfrac{0.6}{1.6*0.18} \Leftrightarrow \mathbf{1.7361} \le k_d \le \mathbf{2.0833} \quad \text{(média 1.9097)} \end{cases}$$

**Para aplicar estes parâmetros nos controladores PID já desenvolvidos as s-function devem ser alteradas para receberem os parâmetros Kp, Ki e Kd e não os converter a partir de Kc, Ti e Td.**

Por isso antes de adicionar o controlador na malha como mostra a figura abaixo, podem ser criados 2 novos blocos para o algoritmo de posição e velocidade.

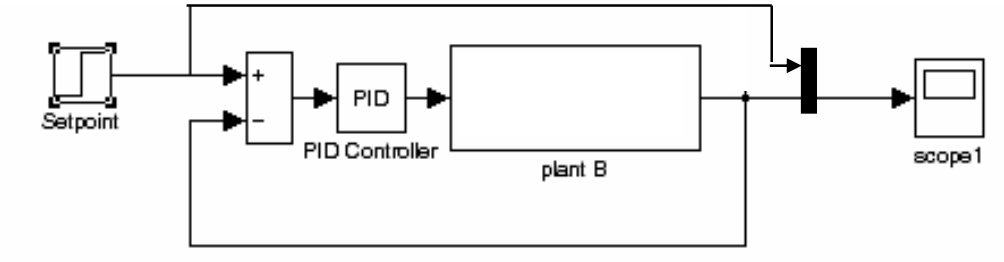



Fig. 3-12: Exemplo de Plant controlada por PID

Estes parâmetros também podem ser estimados analiticamente através da resposta em malha aberta, de acordo com:

$$(3\text{-}20)\quad T_T = T_p + L = \frac{\int_0^\infty [y(\infty) - y(\tau)]d\tau}{y(\infty)}$$

Isto é a área superior entre curva de resposta em malha aberta e a resposta final y(∞) dividida pela resposta final dá-nos o Tempo total (Tt). O atraso (L) pode ser medido verificando quando a resposta deixa de ser nula e assim pode-se calcular a constante de tempo Tp.

Se se tiver um vector com resposta em malha fechada (y) e outro com vectores com os tempos correspondentes (t) o seguinte script, **id1.m**, identifica (numericamente) a função de transferência de 1ª ordem na resposta ao degrau de amplitude A:

```
function [K, Tp, L, R] = id1(y, t, A)

% Atraso
I=find(y>0)
L=t(I(1)-1)

% Constante de tempo Tp de acordo com (3-20)
yf = y(length(y));
tfi = t(length(t));
Tt = ((yf*tfi)-trapz(t,y))/yf;
Tp = Tt - L;

% Declive
R = yf / Tp;  %Não necessário para este método

% Ganho
K = yf / A; % 1 pois e resposta ao degrau unitário
```

Algoritmo 3-4: Identificação de F.T. 1ª ordem

Os vectores podem ser retirados a partir do Scope, fazendo com que este grave as curvas numa variável do workspace: **Scope Parameters -> Data history**.

Se se pretender visualizar a resposta ao degrau da função de transferência estimada pode-se adicionar à função acima:

```
%Mostra resposta a FT estimada
[y,t]=step(tf(A*K, [T 1]));
y=[0; y];
t=[0; t+L];
plot(t,y)
hold on
grid
ylabel('Amplitude')
xlabel('time (s)')
title ('Estimated TF step response')
plot([L Tt], [0 yf], 'r')
hold off
```

Algoritmo 3-5: Mostra curva de resposta estimada

Para este caso os valores obtidos são:

```
K = 1.9991  T = 2.4928  L = 0.6000  (R = 1.6039)
```

Que é muito próximo da FT real. Pode-se também de acordo com (3-16) e (3-17) desenvolver um script que calcule os parâmetros para a versão discreta do PID: Kp, Ki, Kd e Ts:

```
function [Kp, Ki, Kd, Ts] = pid_id_d(y, t, A)

[K,T,L,R]=id1(y,t,A);
Ts = 0.3 * L;
Kp = 1.2 / (R * (L +Ts));
Ki = (0.6 * Ts) / (R * (L + Ts/2)^2);
Kd = (0.5/(R * Ts) + 0.6/(R * Ts))/2;   %Ponto medio de Kd
```

Algoritmo 3-6: Cálculo dos parâmetros para PID discreto

Também seria válido calcular os parâmetros para o PID analógico: Kc, Ti e Td, de acordo com (3-15) e convertê-los para os parâmetros do modelo discreto:

```
function [Kp, Ki, Kd, Ts] = pid_id_ad(y, t, A)

[K,T,L,R]=id1(y,t,2);

Kc=1.2/(R*L);
Ti=2*L;
Td=0.5*L;

Ts = 0.3 * L;
Kp = Kc;
Ki = Kc*Ts/Ti;
Kd = Kc*Td/Ts;
```

Algoritmo 3-7: Cálculo dos parâmetros para PID Analógico e conversão para discreto

Na figura abaixo tem-se a comparação da resposta controlada por PID de posição para a função de transferência estimada (a vermelho) e a para a correcta (a azul), sintonizado de acordo com (3-18) e (3-19):

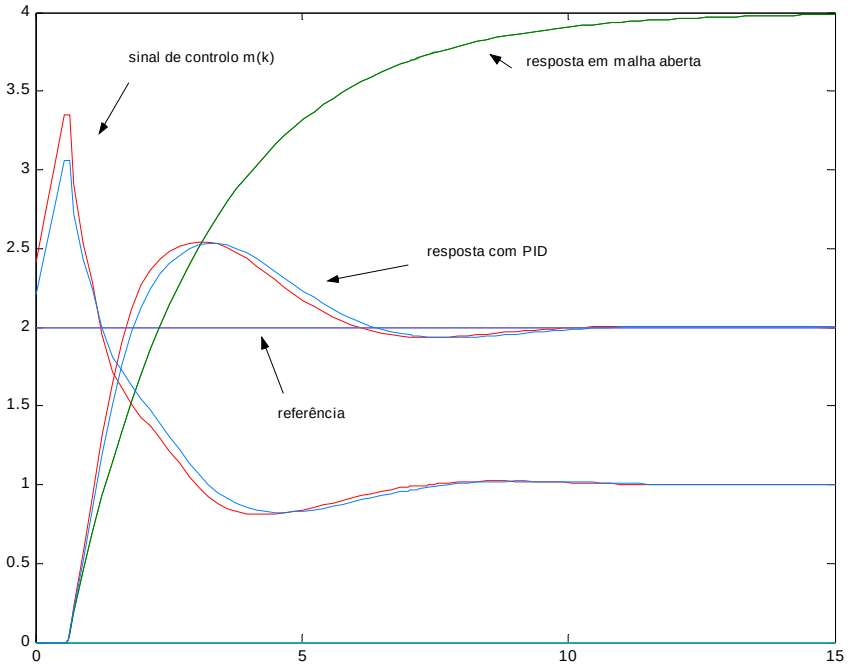



Fig. 3-13: Controlo com PID de posição sintonizado de acordo com (3-18) e (3-19)

Na figura seguinte tem-se a comparação da resposta controlada por PID de velocidade para a função de transferência estimada (a vermelho) e a para a correcta (a azul), sintonizado também de acordo com (3-18) e (3-19). Repare-se que o overshoot é menor.

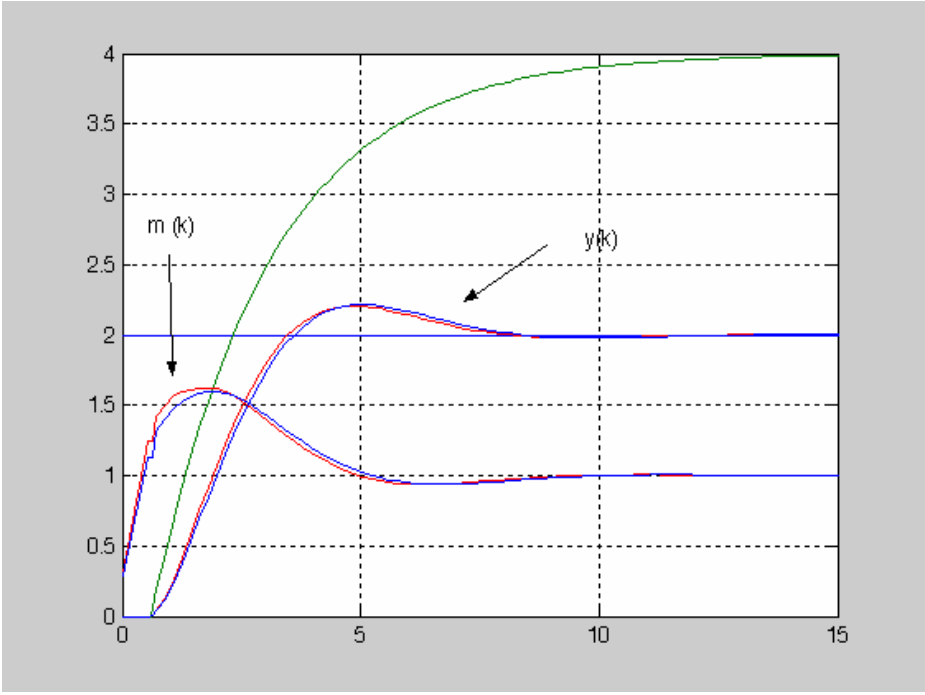
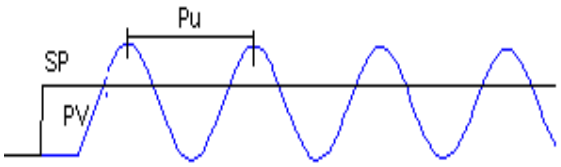



Fig. 3-14: Controlo com PID de velocidade sintonizado de acordo com (3-18) e (3-19)

Por outro lado se a função de transferência fôr de 2ª ou 3ª ordem pode ser usada uma variante do método de Ziegler-Nichols em **malha fechada**:

- 1: Fazer montagem em malha fechada com realimentação unitária positiva de um controlador P puro (isto é um ganho) em série com o processo. (Ou Desligar D e I no controlador PID)
- 2: Selecionar como referência um degrau
- 3: Aumentar o ganho P até que as oscilações tenham uma amplitude constante. Este ganho é chamado ganho critico e designado por Ku.
- 4: Identificar o Período das oscilações Pu.




- 5: Sintonizar o controlador de acordo com:

| | $K_c$ | $\tau_I$ | $\tau_D$ |
|---|---|---|---|
| **PI** | $0.45\ K_u$ | $P_u / 1.2$ | |
| **PID** | $K_u / 1.7$ ou $0.6\ K_u$ | $P_u / 2$ | $P_u / 8$ |

Alternativamente pode ser usada a **Tyreus-Luyben Tuning Chart**, também conhecida por TLC, que tem como objectivo reduzir efeitos oscilatórios e aumentar a robustez:

| | $K_c$ | $\tau_I$ | $\tau_D$ |
|---|---|---|---|
| **PI** | $K_u/3.2$ | $2.2\ P_u$ | |
| **PID** | $K_u/2.2$ | $2.2\ P_u$ | $P_u/6.3$ |

Veja-se um exemplo. Seja o processo de 2ª ordem dado por:

$$G(s) = \frac{k\, e^{-Ls}}{T_1 s^2 + T_2 s + x} = \frac{e^{-0.5s}}{2s^2 + s + x}$$

Que corresponde ao modelo simulink da figura seguinte:

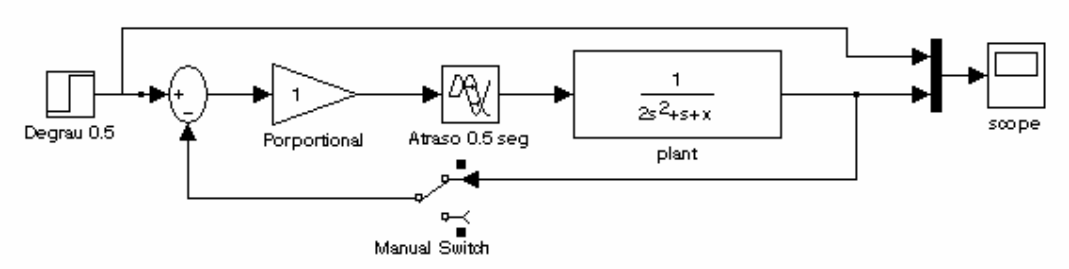



Fig. 3-15: Controlo com PID de processo de 2º ordem

Com x = 0, a resposta do processo G(s) em malha aberta a um degrau com amplitude 0.5 é instável, como mostra a figura abaixo, pois tem um polo duplo na origem:

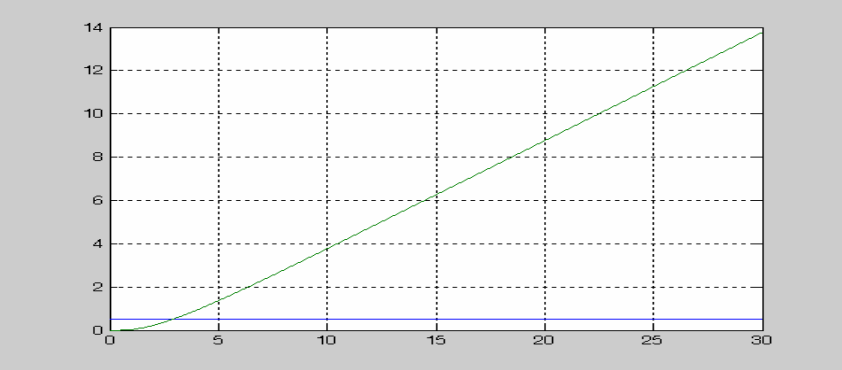



Fig. 3-16: Resposta em malha aberta do processo G(s) com x = 0

Fechando o interruptor na malha de realimentação e aumentando o ganho proporcional em série com o processo até P = 2.07 atinge-se o ganho critico Ku = P = 2.07. Pode-se agora medir o período de oscilação nessas condições Pu=7.05.

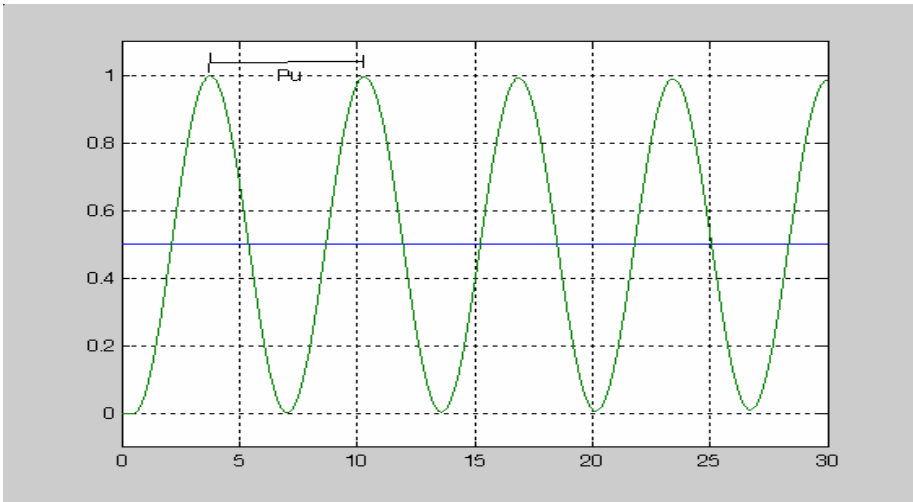



Fig. 3-17: Resposta em condições de ganho critico com Ku=2.07 e Pu = 7.05

Pode-se também escrever um script que faça esta análise a partir da saída acima e de Ku, retornando os parâmetros do controlador PID analógico. Posteriormente se necessário estes terão de ser convertidos para os parâmetros digitais:

```
function [Kc, Ti, Td, Pu] = pid_id_cl(y, t, Ku, TLC)
%Se TLC <> 0 usa Tyreus-Luyben chart para sintonia

%Procura 1º maximo
max=0;
for x=1:length(t)
    if y(x) < max break; end;
    max=y(x);
    x=x+1;
end

%Procura próximo mínimo
min=max;
for x=x:length(t)
    if y(x) > min break; end;
    min=y(x);
    x=x+1;
end

Pu=t(x);

if TLC == 0
    %Ziegler Nichols
    Kc=Ku/1.7; % ou Ku*0.6;
    Ti=Pu/2;
    Td=Pu/8;
else
    %Tyreus-Luyben
    Kc=Ku/2.2;
    Ti=Pu*2.2;
    Td=Pu/6.3;
end
```

Algoritmo 3-8: Cálculo dos parâmetros para PID Analógico em malha fechada

Para este caso os parâmetros estimados são:

Kc = 1.2176 Ti = 3.5250 Td = 0.8812 Pu = 7.0500

O intervalo de amostragem foi seleccionado para 0.01 segundos. A resposta apresenta-se a seguir:

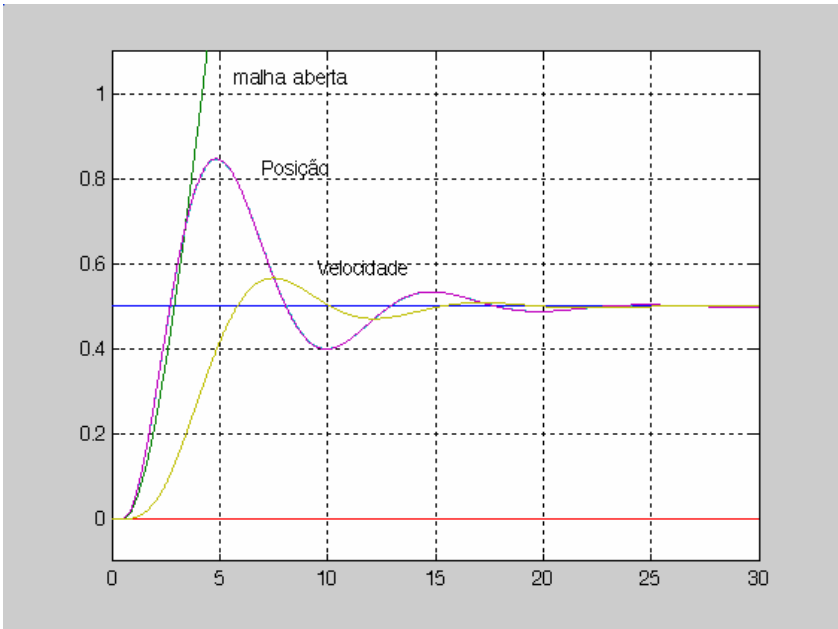



Fig. 3-18: Controlo com PID de posição sintonizado de acordo com Algoritmo 3-1

Modificando agora o processo, fazendo x=1, a resposta em malha aberta não é instável e não apresenta erro em regime estacionário:

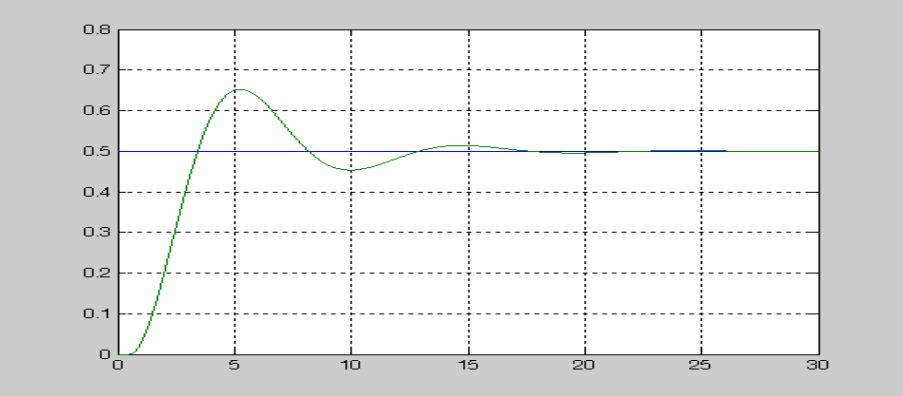



Fig. 3-19: Resposta em malha aberta do processo G(s) com x = 1

Para o ganho critico Ku=2.12 em malha fechada obtém-se Pu = 5.84, e os parâmetros estimados são:

Kc = 1.2471  Ti = 2.9200  Td = 0.7300

Com um intervalo de amostragem de 0.01 segundos, a resposta é:

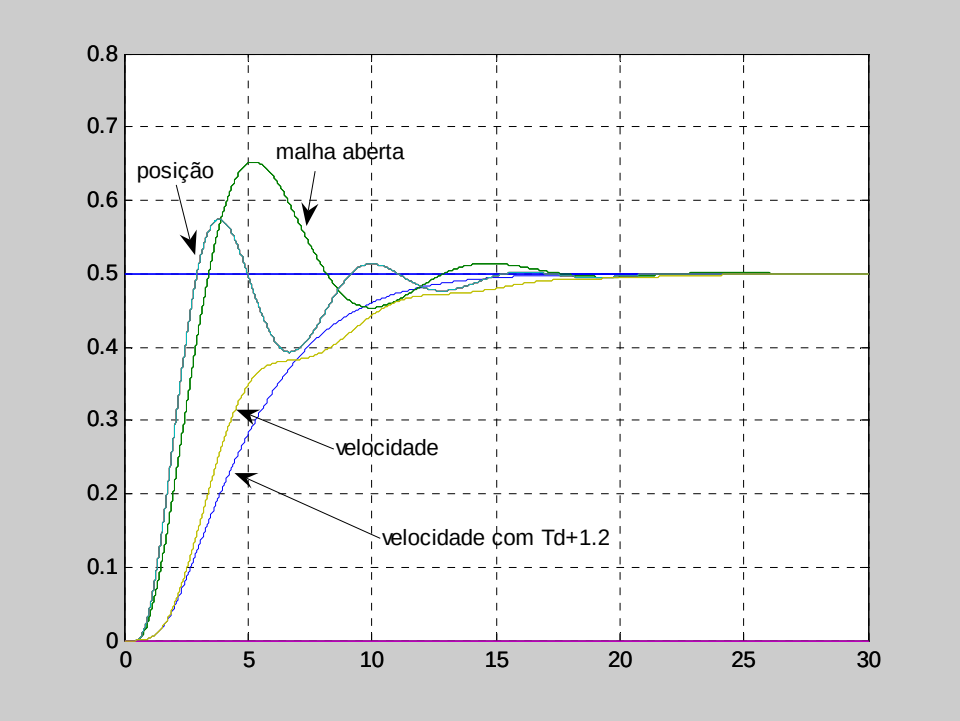



Fig. 3-20: Controlo com PID de posição sintonizado de acordo com Algoritmo 3-1

Como se pode observar a sintonia para o algoritmo de velocidade precisa ainda de alguma afinação. Somando 1.2 ao valor de Td encontra-se uma melhor afinação.

## *3.2 Formas alternativas do algoritmo PID*

### 3.2.1 Transferência sem saltos (bumpless transfer)

Seguidamente apresenta-se um algoritmo modificado de modo a possibilitar a transferência do controlo automático para o manual o mais suavemente possível. O controlo manual assumido é:

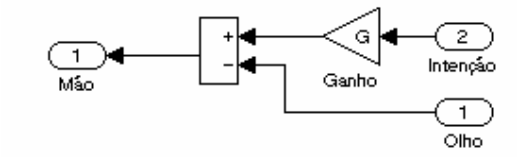



Fig. 3-21: Simulação de controlo manual

Onde a intenção corresponde ao sinal de referência ou setpoint desejado, o olho a uma leitura da saída do processo e a mão ao sinal de controlo a aplicar ao processo. O ganho pode ser variado de modo amplificar a referência reduzindo o erro em regime estacionário. Para este caso considere-se G = 1.5.

Neste caso a intenção ou a referência, é multiplicada por um factor de 1.5 de modo a eliminar o erro em estado estacionário sem afectar a estabilidade.

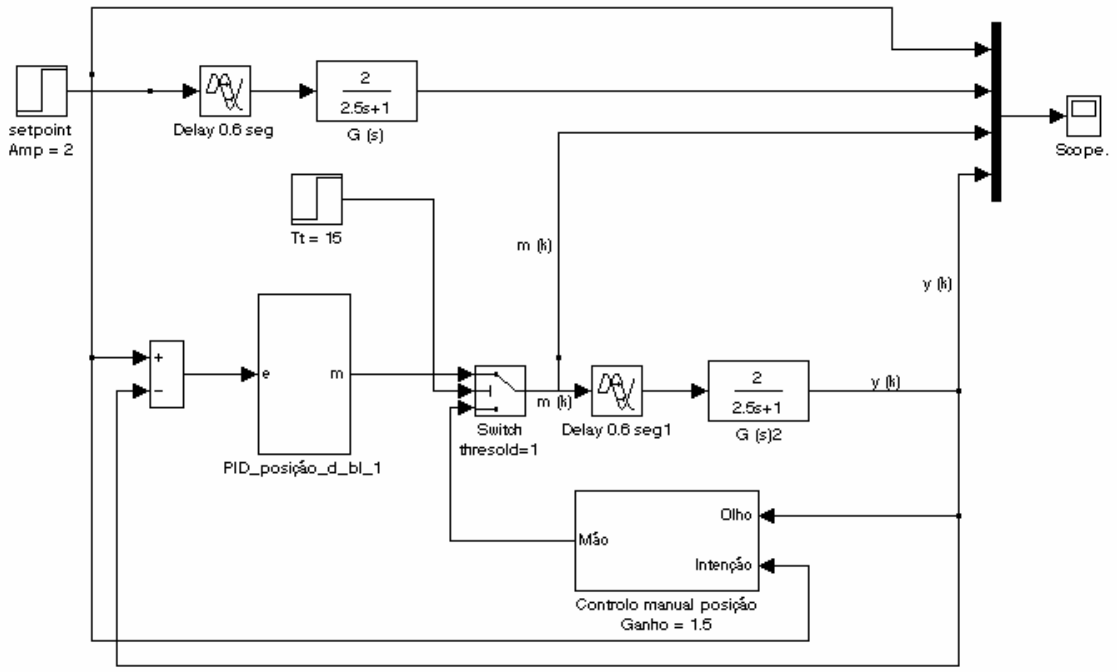



Fig. 3-22: Simulação de bumpless transfer para método 1

No instante de tempo t = 15, é efectuada a transição do modo manual para automático. As alterações ao algoritmo PID de posição para o método 1:

```
function [sys, x0, str, ts] = pid_bump_01 (t, x, u, flag, ts, kp, ki, kd, tt, MV)
persistent s ek_1
 (...)
if flag == 3              %tt é o instante de transição manual -> automático
        if t > tt-1       %Simula ligação do controlador PID neste instante
                ek_1 = u(1);
                s = 0;
        end

        ek=u(1);
        s = s + ek;
```

```
m = kp*ek + ki*s + kd*(ek-ek_1) + MV;
ek_1 = ek;
sys = m;
```

Algoritmo 3-9: PID bumpless método 1 (alterações ao Algoritmo 3-1 PID de posição )

A figura seguinte mostra a resposta do algoritmo PID de posição na transferência do modo de controlo manual para automático (que é diferente do método 1 com MV=0):

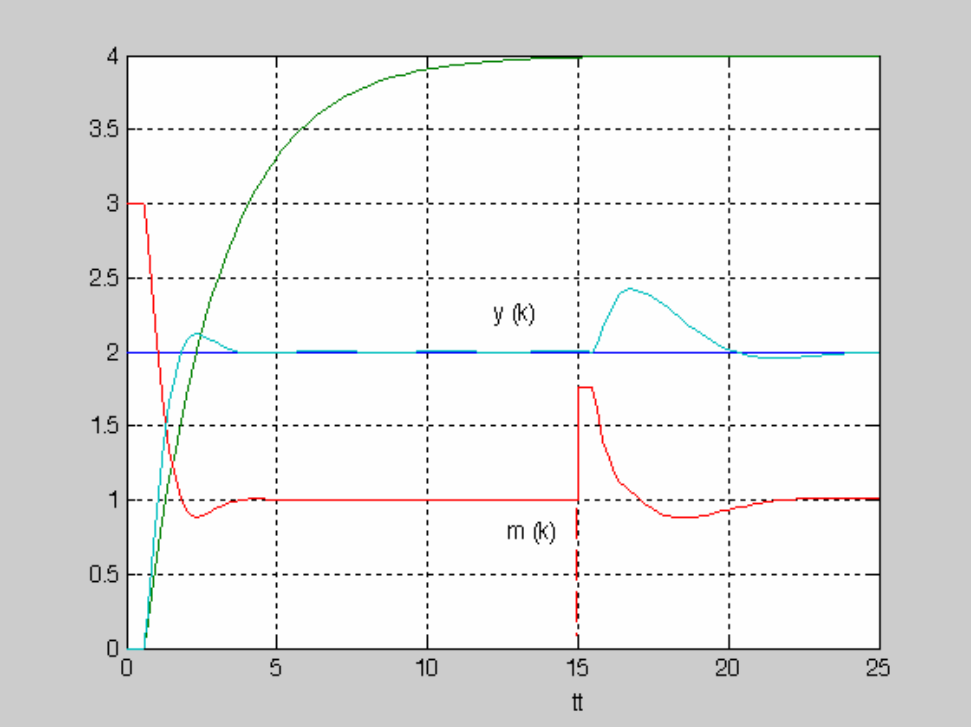



Fig. 3-23: Resposta sem bumpless transfer (t = 15 segundos)

Na figura seguinte encontra-se a resposta com o algoritmo modificado para possibilitar a transferência sem saltos pelo método 1. O valor de **MV = 1** é o ponto de operação do controlador $\mathbf{m_k}$, ou a posição do actuador, quando da transição e pode ser obtido através de ensaios.

A sintonia PID é efectuada de acordo com o já determinado em (3-19):

kp=0.9615 ki=0.1418 kd=(1.7361+2.0833)/2 ts=0.18

Como já referido o instante de transição é em: **tt=15**.

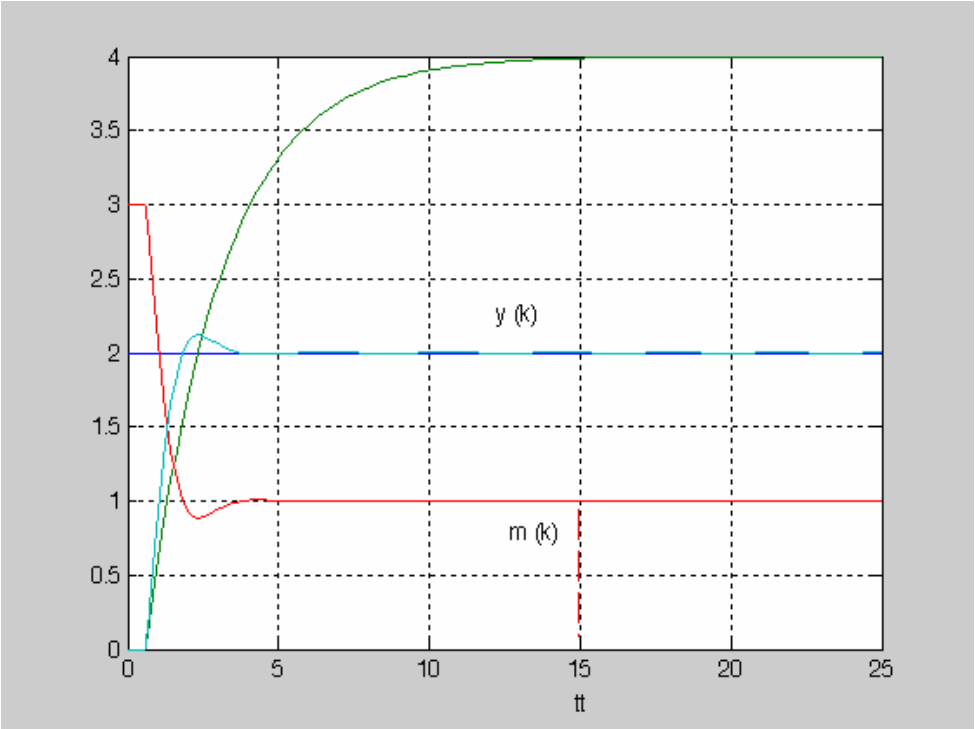



Fig. 3-24: Resposta bumpless transfer (método 1) em t = 15 segundos (MV = 1)

Se se variar MV a transição apresentará um salto.

A seguir vai ser ensaiado o algoritmo de velocidade.

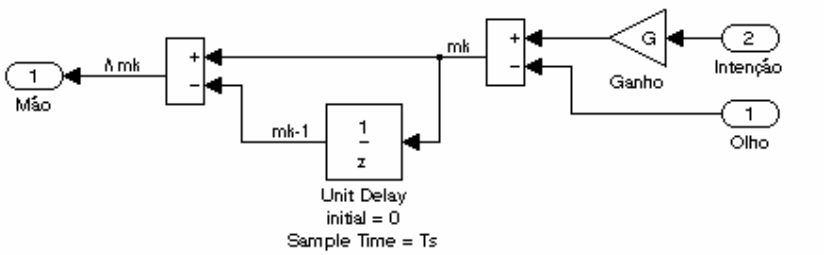



Fig. 3-25: Simulação de controlo manual

Repare-se que neste caso o controlo manual terá de ser alterado de modo que o actuador seja incrementado ou decrementado, isto é pretende-se a variação no sinal de controlo de acordo com (3-5). Então o controlo manual terá de ser redesenhado de acordo com a figura anterior. Deve ser fornecido também o intervalo de amostragem ao actuador de modo que o sinal $m_k$ seja atrasado um instante de amostragem.

Abaixo tem-se as alterações no modelo para utilizar o algoritmo PID de velocidade:

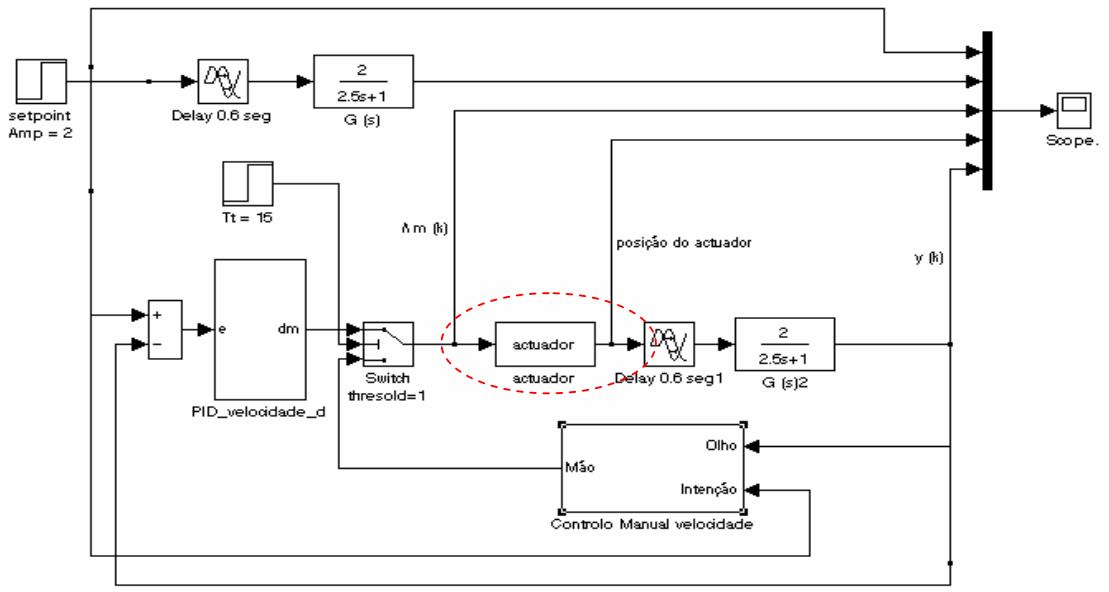



Fig. 3-26: Simulação de bumpless transfer para método 3 (algoritmo de velocidade)

A figura abaixo apresenta a resposta:

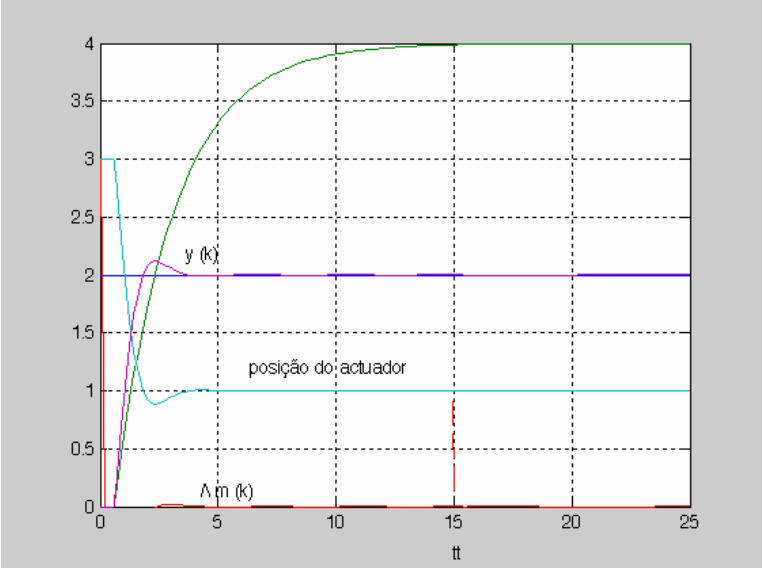



Fig. 3-27: Resposta bumpless transfer (método 3 algoritmo de velocidade) em t = 15 seg.

Assim consegue-se um transferência sem salto, sem ter de se ensaiar ou monitorizar constantes tais como MV.