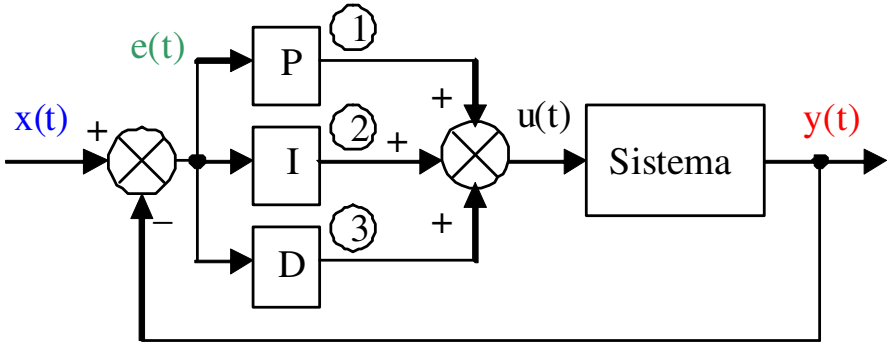
Tema 13

Implementación de un Regulador PID

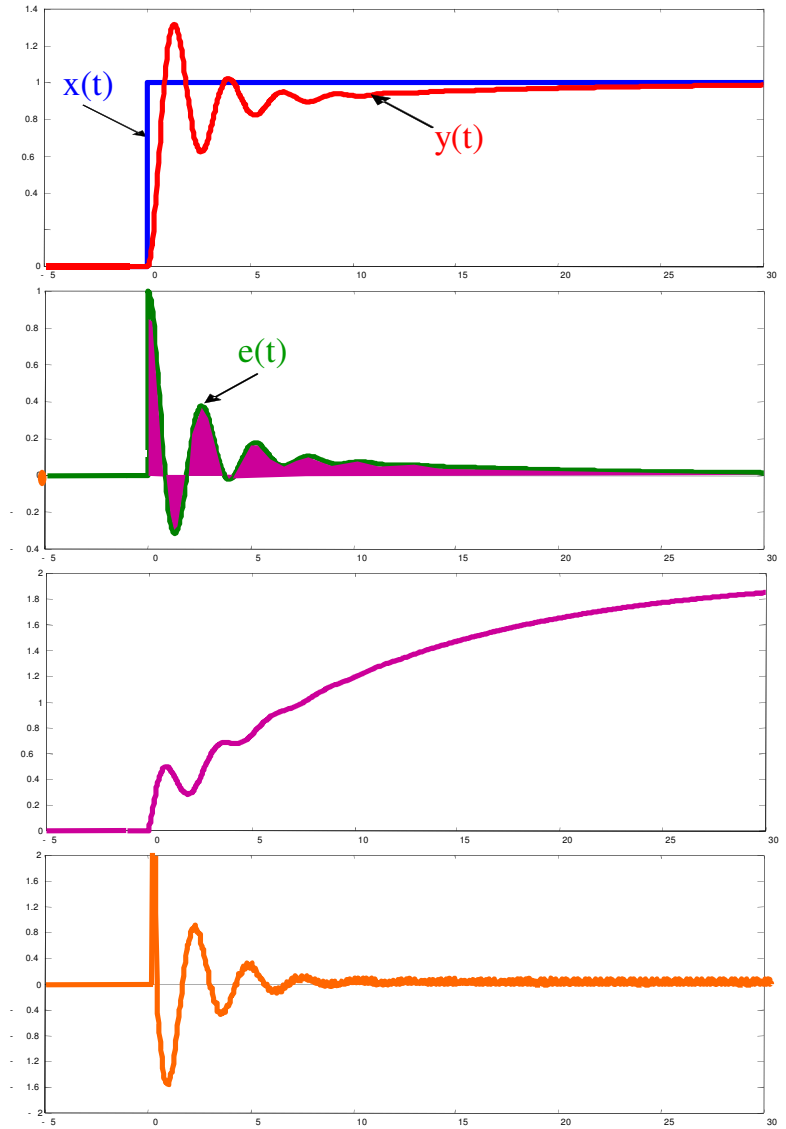
Acciones de Control Clásicas



1- Proporcional $e(t) = x(t) - y(t)$

2- Integral $\int_0^t e(t) \cdot dt$

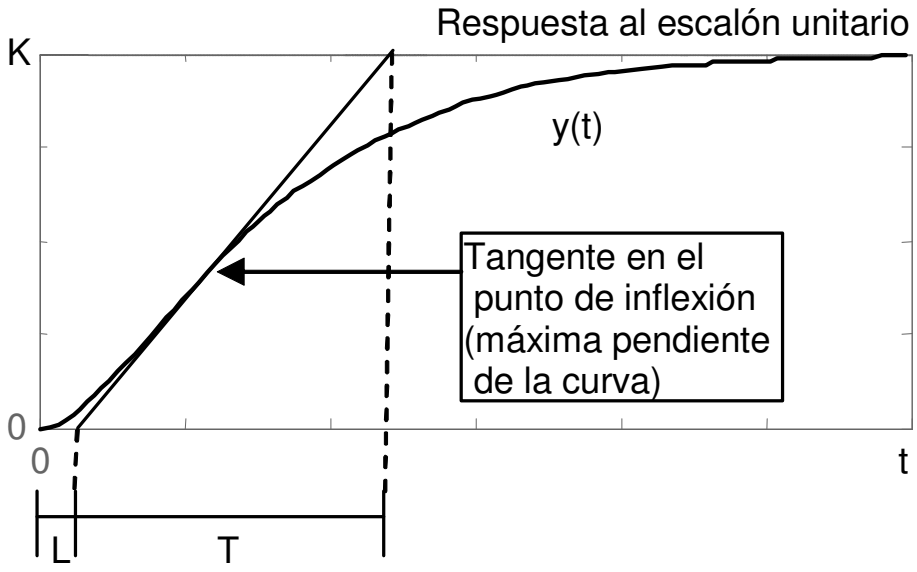
3- Diferencial $\frac{de(t)}{dt}$



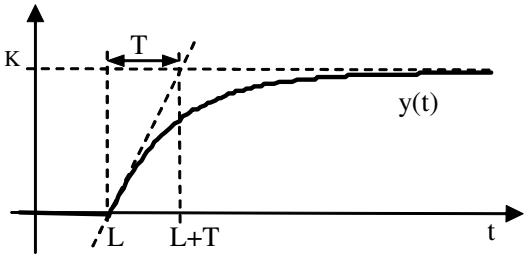
Diseño de un regulador PID

Se puede sintonizar un regulador PID sencillo para sistemas monovariables que presentan una respuesta en forma de "S" ante entradas escalón, sin necesidad de conocer su modelo matemático.

Uno de los métodos es el primero de los propuestos por Ziegler-Nichols cuyo objetivo es obtener una respuesta con una sobreoscilación aproximada del 25%.



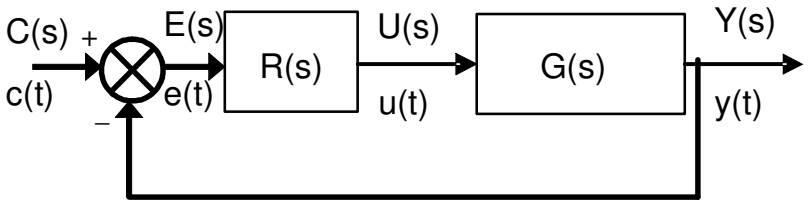
Ziegler-Nichols considera en realidad un sistema de primer orden con constante de tiempo T y retardo puro L:

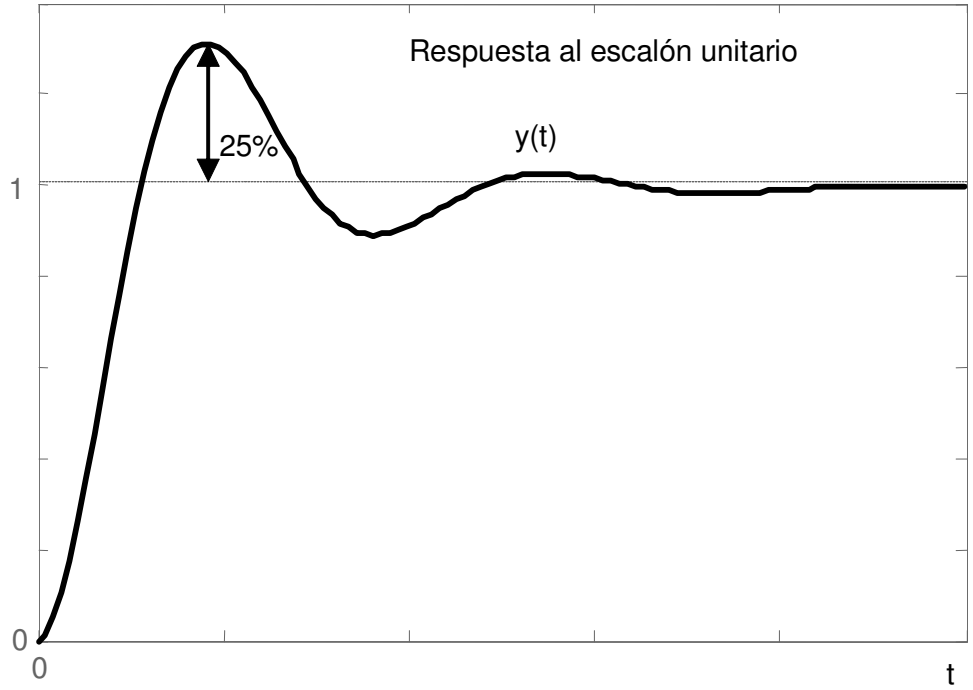


$$G(s) = e^{-L \cdot s} \frac{K}{T \cdot s + 1}$$

Diseño de un regulador PID

A partir de los valores de "K", "L" y "T" se pueden obtener los parámetros de un regulador P, PI o PID, para que el sistema en bucle cerrado tenga una respuesta con una sobreoscilación aproximada del 25%.





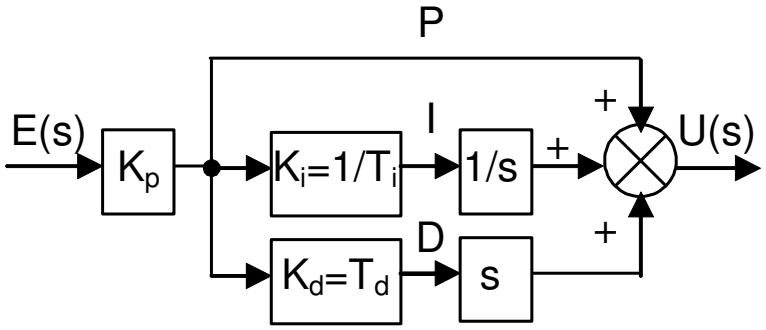
Nota: En el caso de que la realimentación no sea unitaria, H(s)≠1, se debe utilizar la respuesta al escalón unitario de G(s)·H(s) para determinar los valores de "K", "L" y "T".

Diseño de un regulador PID

Los parámetros para los distintos reguladores se obtiene a partir de la siguiente tabla:

TIPO	K_p	$K_i=1/T_i$	$K_d=T_d$
P	$T/(K\cdot L)$	0	0
PI	$0.9\cdot T/(K\cdot L)$	$0.3/L$	0
PID	$1.2\cdot T/(K\cdot L)$	$0.5/L$	$0.5\cdot L$

$$u(t) = K_p \cdot \left(e(t) + K_i \cdot \int e(t) \cdot dt + K_d \frac{de(t)}{dt} \right)$$



$$R(s) = \frac{U(s)}{E(s)} = K_p \cdot \left(1 + \frac{K_i}{s} + K_d \cdot s \right)$$

Cálculo Iterativo

$$u_k = K_p \cdot (e_k + K_i \cdot \text{integral} + K_d \cdot \text{derivada})$$

… → **Cálculo en el instante K**

$$\text{integral} = \text{integral} + e_{k-1} \cdot T_m$$

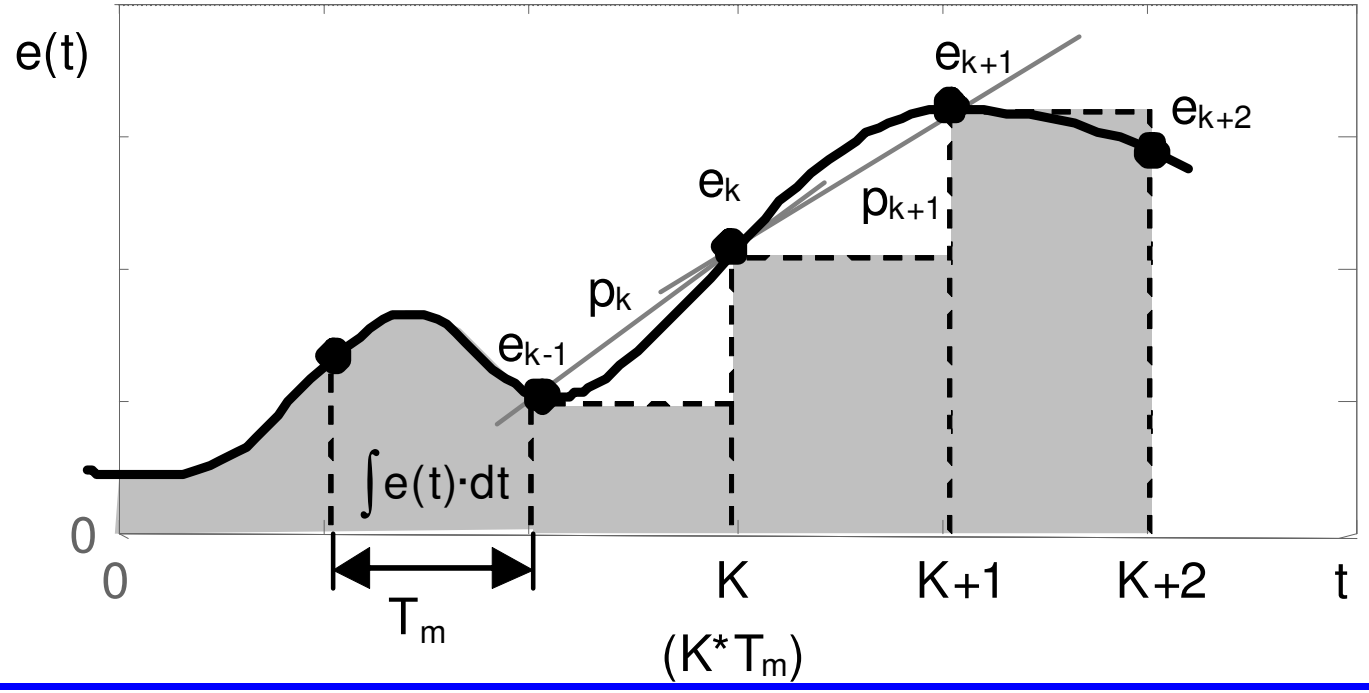
$$\text{derivada} = p_k = \frac{e_k - e_{k-1}}{T_m}$$

→ **Cálculo en el instante K+1**

$$\text{integral} = \text{integral} + e_k \cdot T_m$$

$$\text{derivada} = p_{k+1} = \frac{e_{k+1} - e_k}{T_m}$$

→ …



Implementación de un regulador PID

```
PROGRAM CONTROL_NIVEL
VAR
        rTM:REAL:=1.0;          (* Tiempo de muestreo en segundos *)
                                (* Debe coincidir con el "tiempo de ciclo" del programa *)
        rREF_NIVEL: REAL;       (* REFERENCIA de nivel *)
        rERROR:REAL:=0.0;       (* ERROR en este ciclo *)
        rERROR_OLD:REAL:=0.0;(* ERROR en el ciclo anterior *)
        rINTEGRAL:REAL:=0.0;    (* INTEGRAL del error *)
        rDERIVADA:REAL:=0.0;    (* DERIVADA del error *)
        rSPEED:REAL:=0.0;       (* CONSIGNA de velocidad para el motor de la bomba *)
END_VAR


(* Programa ciclico que calcula cada 1 segundos el algoritmo del PID *)
rNIVEL:=FU_rLEENIVEL(I_wSENNIV);            (*  Se actualiza aquí la variable global rNIVEL *)
rREF_NIVEL:=FU_rLEEREF(I_iREFNIV);          (* Lectura del valor de REFERENCIA *)

rERROR:= rREF_NIVEL-rNIVEL;                 (* Cálculo del error *)
rINTEGRAL:=rINTEGRAL+rERROR_OLD*rTM;        (* Cálculo de la integral del error *)
rDERIVADA:=(rERROR-rERROR_OLD)/rTM;         (* Cáculo de la derivada del error *)

rSPEED:=rKP*(rERROR+rKI*rINTEGRAL+rKD*rDERIVADA);  (* Consigna de velocidad *)

rERROR_OLD:=rERROR; (* Se guarda el valor del error para el próximo ciclo *)
```

Implementación de un regulador PID

```
(* Programa ciclico que calcula cada 1 segundos el algoritmo del PID *)
rNIVEL:=FU_rLEENIVEL(I_wSENNIV);                (*  Se actualiza aquí la variable global rNIVEL *)
rREF_NIVEL:=FU_rLEEREF(I_iREFNIV);              (* Lectura del valor de REFERENCIA *)
rERROR:= rREF_NIVEL-rNIVEL;                     (* Cálculo del error *)

IF xPID_ACTIVO THEN     (* Se dan las condiciones para el funcionamietno del PID *)
                IF rKI>0.0 THEN
                  rINTEGRAL:=rINTEGRAL+rERROR_OLD*rTM; (* Cálculo de la integral del error *)
                ELSE
                  rINTEGRAL:=0.0;(* Evita que la integral del error crezca cuando no se usa la acción integral *)
                END_IF;

                rDERIVADA:=(rERROR-rERROR_OLD)/rTM;          (* Cáculo de la derivada del error *)

                rSPEED:=rKP*(rERROR+rKI*rINTEGRAL+rKD*rDERIVADA);  (* Consigna de velocidad *)

                IF rSPEED > 100.0 THEN rSPEED:=100.0; END_IF;  (* Límites de velocidad 0->100 % *)
                IF rSPEED < 0.0 THEN rSPEED:=0.0; END_IF;
                Q_iSPEED:=10*REAL_TO_INT(rSPEED);            (* Se envía la consigna a la salida *)
ELSE
                rINTEGRAL:=0.0;(* Asegura que la integral vuelve a cero cuando el PID no está activo *)
                rSPEED:=0.0;
END_IF;

rERROR_OLD:=rERROR; (* Se guarda el valor del error para el próximo ciclo *)
```