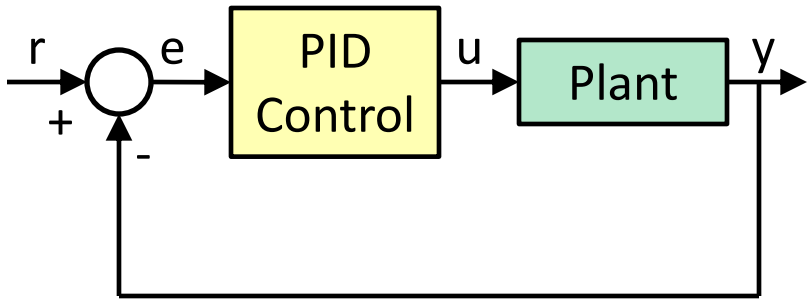
PID Control

- What is a PID control
  - Proportional + Integral + Derivative (PID)
  - Popular in the industry
    - By 1989, more than 90% PID
  - Easy to implement
  - It is quite robust
  - Applies to mechanical systems
    - Predominantly 2nd-order systems
  - Tuning algorithms not dependent on exact system model
  - Two popular tuning techniques
    - Step reaction curve experiment
    - Closed-loop “cycling” experiment under proportional control around the nominal operating point



PID Control

The PID action

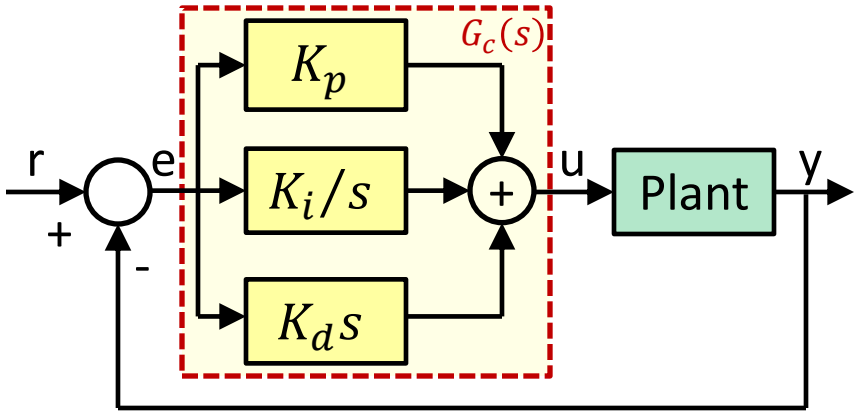
$$e = r - y$$

$$u = K_p e + K_i \int_0^t e + K_d \dot{e}$$

$$U(s) = K_p E(s) + \frac{K_i}{s} E(s) + K_d s E(s)$$

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

$$G_c(s) = \frac{U(s)}{E(s)}\bigg|_{IC=0} = K_p + K_i \frac{1}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

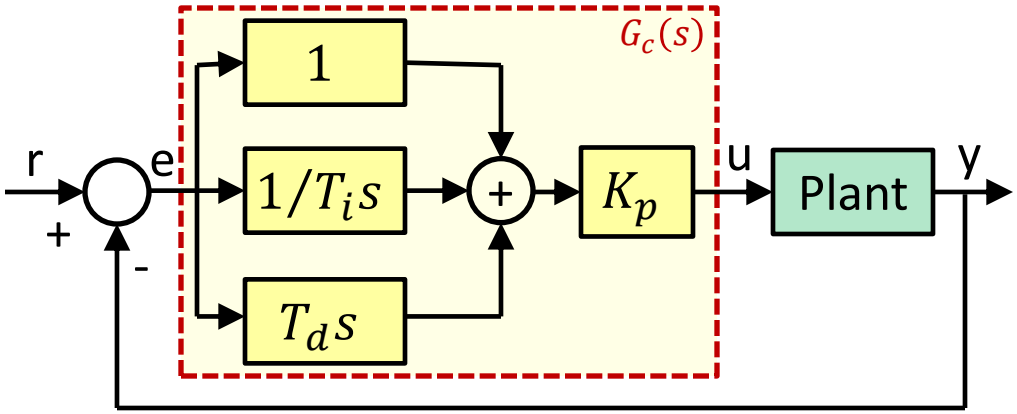


Equivalently,

Let $T_i = \frac{K_p}{K_i}$ and $T_d = \frac{K_d}{K_p}$

$$\Rightarrow u = K_p \left(e + \frac{1}{T_i} \int_0^t e + T_d \dot{e}\right)$$

$$U(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right) E(s)$$



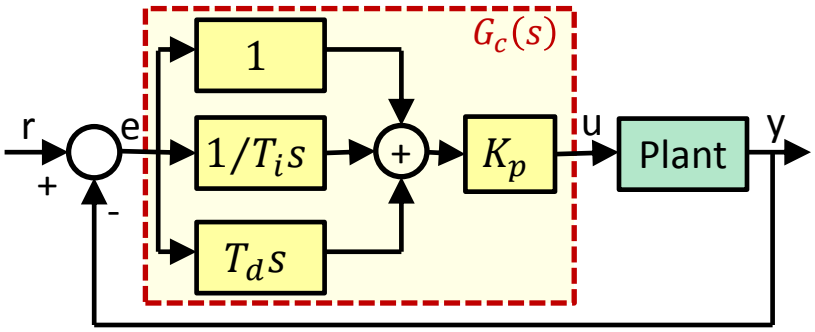
$$G_c(s) = \frac{U(s)}{E(s)}\bigg|_{IC=0} = K_p \left(1 + \frac{1}{T_i s} + T_d s\right) = K_p \left(\frac{T_d s^2 + s + 1/T_i}{s}\right)$$

PID Control

Example

Consider a third-order plant $G(s) = \frac{1}{(s+1)^3}$ with a proportional control $G_c(s) = K_p$.

Draw the step response of the closed-loop system for various values of K_p.



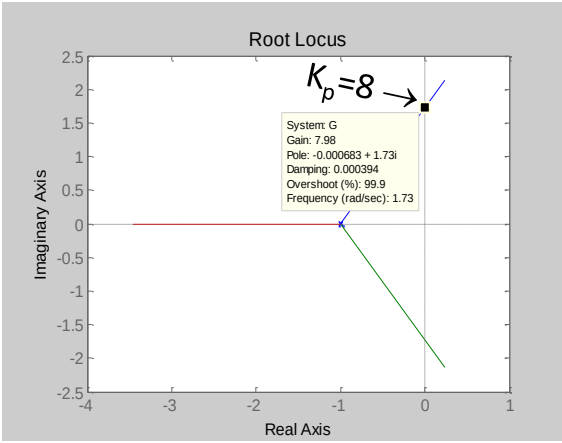
Matlab code:

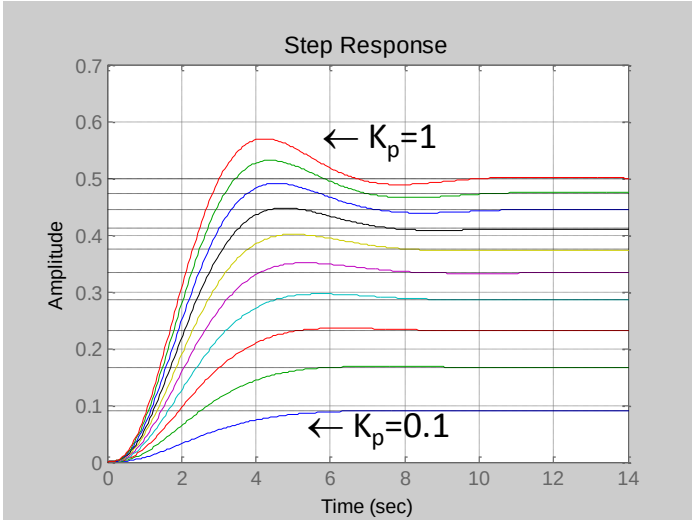
```
s=tf('s'); G=1/(s+1)^3;
for Kp=[0.1:0.1:1], Gcl=feedback(Kp*G,1);
   figure(1), step(Gcl), hold on; end
figure(2), rlocus(G,[0,15]);
```

As K_p increases:

- Response speed increases
- Overshoot increases (less stable)
- Steady-state error decreases
- System becomes unstable for

$$K_p > 8$$

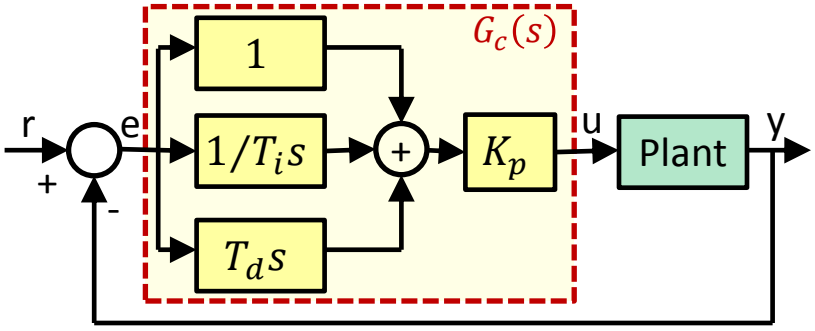




PID Control

Example (continued):

Now fix $K_p = 1$, apply a PI (proportional + integral) control, and draw the step response of the closed-loop system for various values of T_i.



Matlab code:

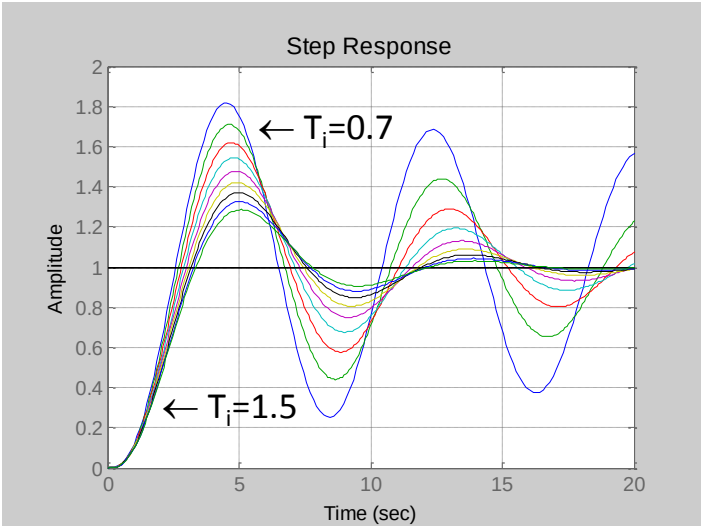
```
s=tf('s'); G=1/(s+1)^3; Kp=1;
for Ti=[0.7:0.1:1.5], Gc=Kp*(1+1/Ti/s);
   Gcl=feedback(G*Gc,1); step(Gcl), hold on; end
```

- Due to integrator action, the steady-state error to step command will be zero for any value of T_i, if the closed-loop system is stable
- If $T_i < 0.6$ the closed-loop system will not be stable

As T_i increases:

Response speed decreases

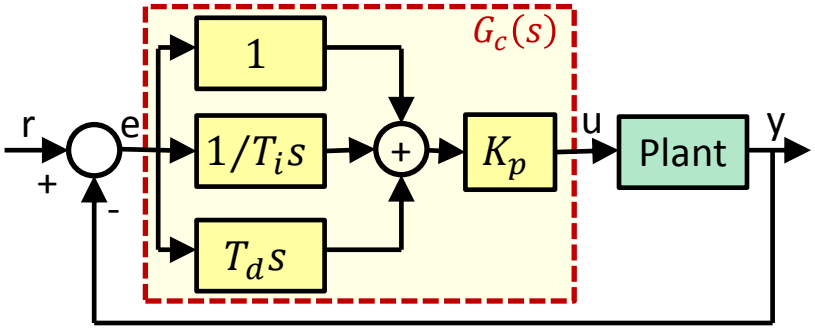
Overshoot decreases (less stable)



PID Control

Example (continued):

Now fix $K_p = T_i = 1$, apply a PID (proportional + integral + Derivative) control, and draw the step response of the closed-loop system for various values of T_d.

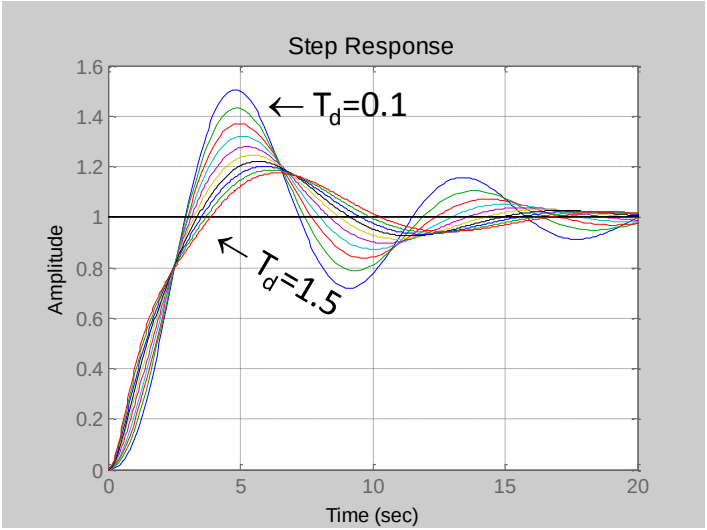


Matlab code:

```
s=tf('s'); G=1/(s+1)^3; Kp=1; Ti=1;
for Td=[0.1:0.2:2], Gc=Kp*(1+1/Ti/s+Td*s);
  Gcl=feedback(G*Gc,1); step(Gcl), hold on; end
```

As T_d increases:

- Response rise-time increases slightly (slower rise)
- Response settling-time does not change
- Overshoot decreases (more stable)



PID Control Implementation

PID control with low-pass filter

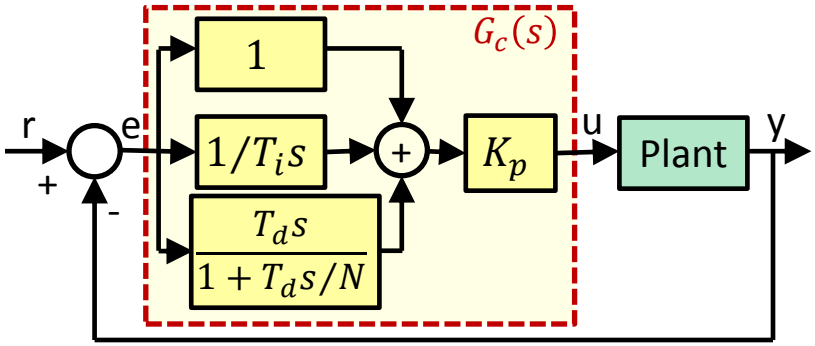
In practice pure derivative is not used, due to:

- Derivative kick (a jump in the control for a step input)
- Undesirable noise amplification

Need to approximate the derivative term so that the controller's TF is proper (i.e., the order of its numerator is less than or equal to the order of its denominator)

- Approximate the derivative term, as: $s \cong \frac{s}{\varepsilon s+1}, \quad \varepsilon \ll 1$
- Or equivalently, cascade the derivative term by a first-order low-pass filter: $T_d s \cong \frac{T_d s}{1+\frac{T_d}{N}s}$

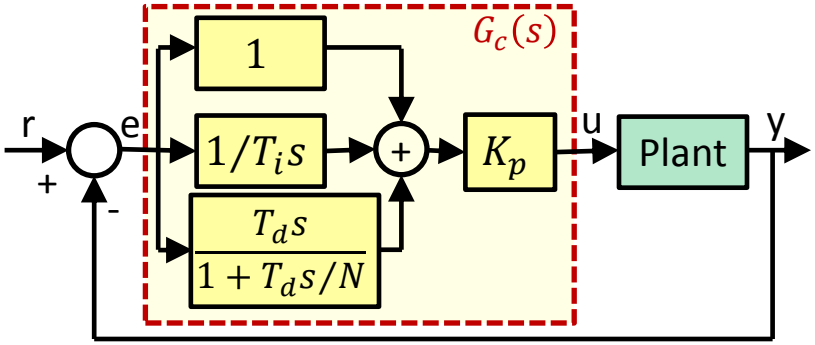
Approximate PID control: $U(s) = K_p\left(1 + \frac{1}{T_i s} + \frac{T_d s}{1+\frac{T_d}{N}s}\right)E(s)$



PID Control Implementation

Example (continued):

Now fix $K_p = T_i = T_d = 1$, apply an approximate PID (proportional + integral + Derivative with filter) control, and draw the step response of the closed-loop system for various values of N.



Matlab code:

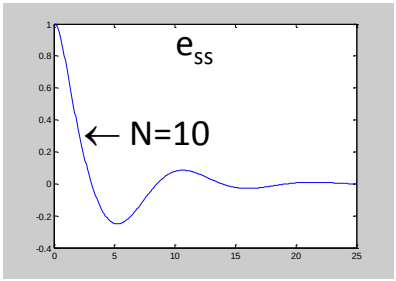
```
s=tf('s'); G=1/(s+1)^3; Kp=1; Ti=1; Td=1;
for N=[100,1000,10000,0.1:10,10],
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
   Gcl=feedback(G*Gc,1); step(Gcl), hold on; end, hold off;
figure, [y,t]=step(Gcl); err=1-y; plot(t,err);
```

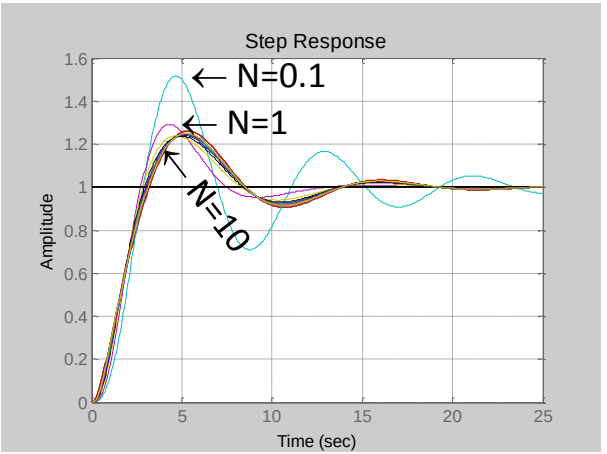
$N = \infty$ produces the original PID controller

$N = 10$ provides good PID approximation

As N decreases:

PID approximation degrades





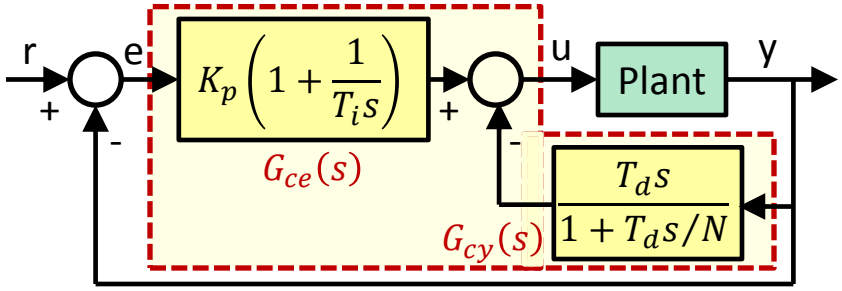
PID Control Implementation

PID control with derivative in the feedback loop

In practice the derivative is preferred in the feedback loop to produce smoother control for a step input

$$G_{ce}(s) = K_p\left(1 + \frac{1}{T_i s}\right)$$

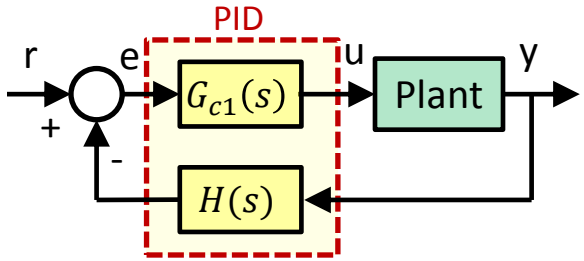
$$G_{cy}(s) = \frac{T_d s}{1 + T_d s/N}$$



Equivalently:

$$G_{c1}(s) = G_{ce}(s) = K_p\left(1 + \frac{1}{T_i s}\right)$$

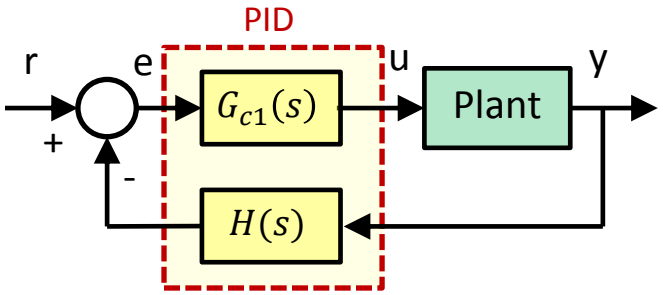
$$H(s) = 1 + \frac{G_{cy}(s)}{G_{ce}(s)} = \frac{(1+K_p/N)T_i T_d s^2 + K_p(T_i + T_d/N)s + K_p}{K_p(T_i s+1)(T_d s/N+1)}$$



PID Control Implementation

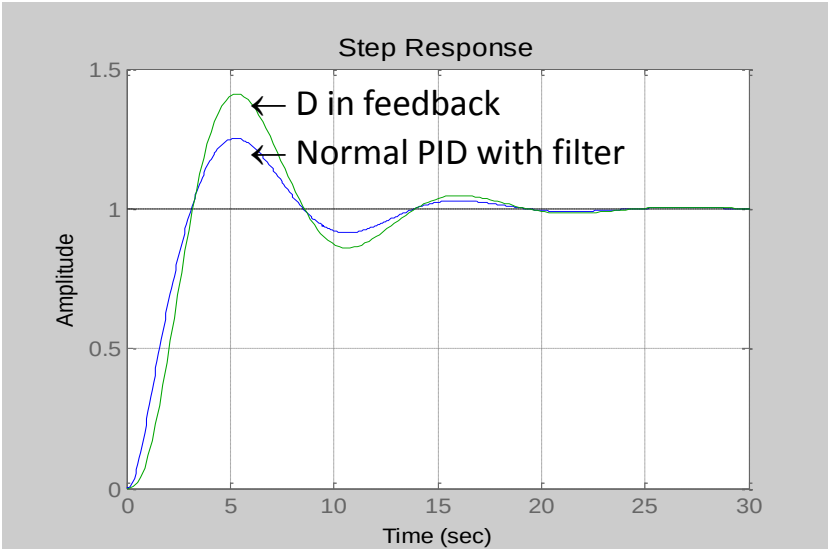
Example (continued):

Now with $K_p = T_i = T_d = 1, N = 10$, apply an filtered PID control with derivative in the feedback, and draw the step response of the closed-loop system.



Matlab code:

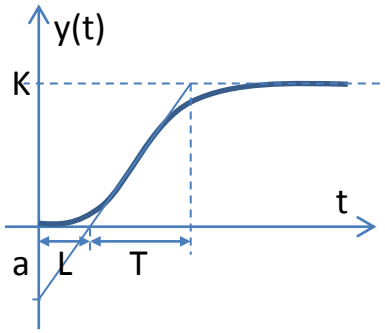
```
s=tf('s'); G=1/(s+1)^3; Kp=1; Ti=1; Td=1; N=10;
Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
Gcl1=feedback(G*Gc,1); Gc1=Kp*(1+1/Ti/s);
H=((1+Kp/N)*Ti*Td*s^2+Kp*(Ti+Td/N)*s+Kp)/(Kp*(Ti*s+1)*(Td/N*s+1));
Gcl2=feedback(G*Gc1,H); step(Gcl1,Gcl2);
```



PID Tuning

Ziegler-Nichols tuning formula

- For first-order plus dead-time (FOPDT) plant models: $G(s) = \frac{K}{1+sT} e^{-sL}$
  - Many plants can be approximately expressed by FOPDT model
  - FOPDT model of a system can be derived experimentally
  - System's step response is S-shaped, as



Tuning procedure:

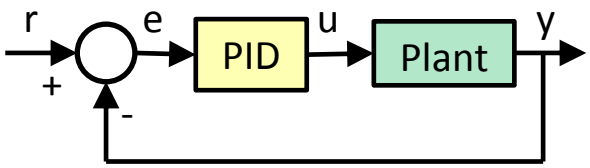
1. Step reaction curve experiment:
   - Measure the system's step-response, experimentally
   - Estimate the parameters K, L, and T (or $a = \frac{KL}{T}$)
   - Find the PID controller parameters from the Table:

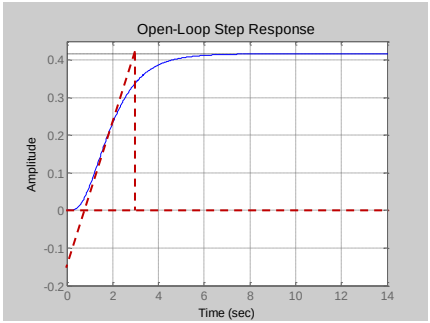
Controller Type	From Step Response		
	K_p	T_i	T_d
P	$1/a$		
PI	$0.9/a$	$3L$	
PID	$1.2/a$	$2L$	$L/2$

PID Tuning

Example:

- For the fourth-order plant $G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)}$, design a PID controller based on Ziegler-Nichols tuning rule.





Matlab code:

```
s=tf('s');
G=10/(s+1)/(s+2)/(s+3)/(s+4);
step(G); K=dcgain(G),
```

From step response: $K = 0.4167$, $L = 0.76$, $T = 1.96$, and $a = 0.1616$

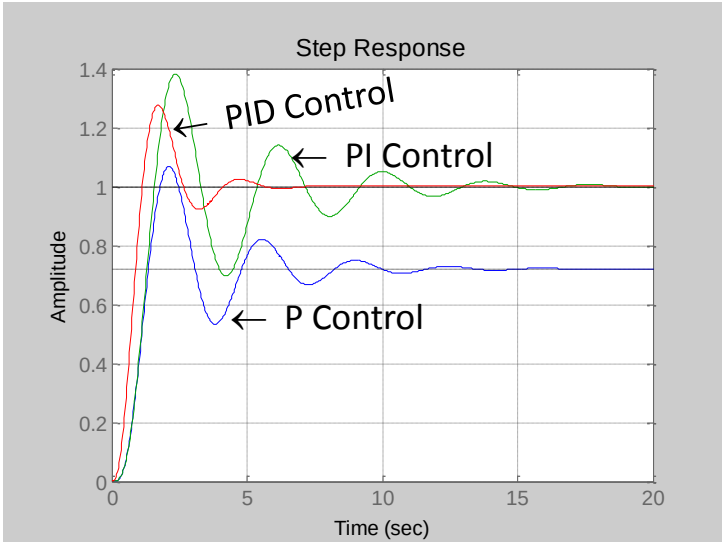
From Ziegler-Nichols' Table:

P Control: $K_p = \frac{1}{a} = 6.1895$

PI Control: $K_p = \frac{0.9}{a} = 6.1895$, $T_i = 3L = 2.28$

PID control: $K_p = \frac{1.2}{a} = 7.4274$, $T_i = 2L = 1.52$, $T_d = \frac{L}{2} = 0.38$

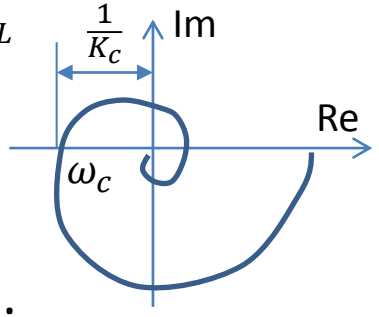
```
L=0.76; T=1.96; a=K*L/T;
Gcp=(1/a); Gcl1=feedback(G*Gcp,1);
Gcpi=(0.9/a)*(1+1/3/L/s); Gcl2=feedback(G*Gcpi,1);
Gcpid=(1.2/a)*(1+1/2/L/s+L/2*s); Gcl3=feedback(G*Gcpid,1);
step(Gcl1,Gcl2,Gcl3);
```



PID Tuning

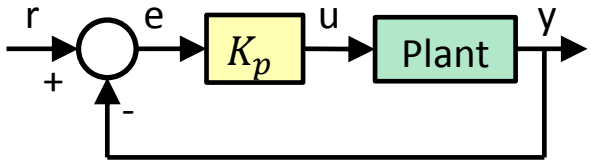
Ziegler-Nichols tuning formula

- First-order plus dead-time (FOPDT) plant models: $G(s) = \frac{K}{1+sT} e^{-sL}$
- System's Nyquist plot is as:



Tuning procedure:

2. Closed-loop cycling experiment with proportional control:
   - Draw the system's Nyquist plot, experimentally
   - Estimate the critical frequency and gain, ω_c and K_c, from the plot
     - Experimentally:
       - Increase gain K_p until system response becomes oscillatory (this is the critical gain K_c)
       - Estimate the frequency of oscillation as critical frequency ω_c
   - Find the PID controller parameters from the Table, where $T_c = \frac{2\pi}{\omega_c}$:
- Typically $T_i > 4T_d$

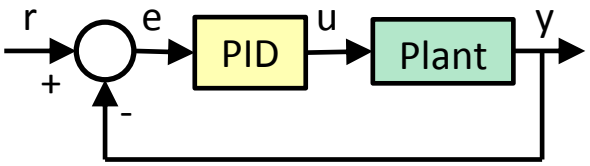


Controller Type	From Step Response		
	K_p	T_i	T_d
P	$0.5K_c$		
PI	$0.4K_c$	$0.8T_c$	
PID	$0.6K_c$	$0.5T_c$	$0.12T_c$

PID Tuning

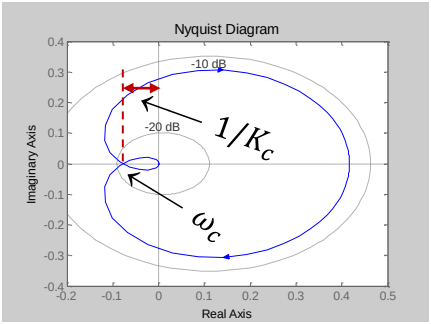
Example:

- For the fourth-order plant $G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)}$, design a PID controller based on Ziegler-Nichols tuning rule.



Matlab code:

```
s=tf('s');
G=10/(s+1)/(s+2)/(s+3)/(s+4);
nyquist(G); axis([-0.2,0.5,-0.4,0.4]);
[Gm,Pm,wcg,wcp]=margin(G)
```



From Nyquist plot: $K_c = G_m = 12.6$, $\omega_c = \omega_{cg} = 2.2361$, $T_c = \frac{2\pi}{\omega_c} = 2.8099$

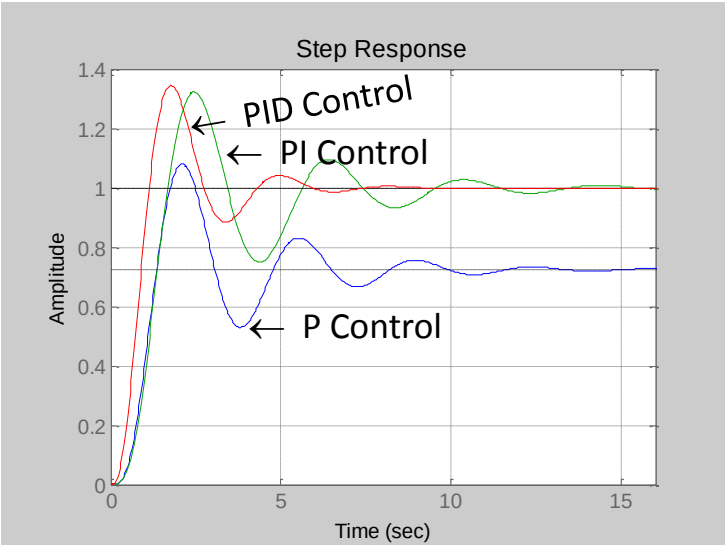
From Ziegler-Nichols' Table:

P Control: $K_p = 0.5K_c = 6.3$

PI Control: $K_p = 0.4K_c = 5.04$, $T_i = 0.8T_c = 2.2479$

PID control: $K_p = 0.6K_c = 7.56$, $T_i = 0.5T_c = 1.405$, $T_d = 0.12T_c = 0.3372$

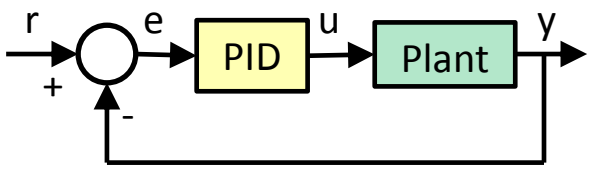
```
Kc=Gm; wc=wcg; Tc=2*pi/wc;
Gcp=0.5*Kc; Gcl1=feedback(G*Gcp,1);
Gcpi=0.4*Kc*(1+1/0.8/Tc/s); Gcl2=feedback(G*Gcpi,1);
Gcpid=0.6*Kc*(1+1/0.5/Tc/s+0.12*Tc*s);
Gcl3=feedback(G*Gcpid,1);
step(Gcl1,Gcl2,Gcl3);
```



Automatic Tuning of PID Controller

Example:

- For the fourth-order plant $G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)}$, design a PID controller based on Ziegler-Nichols tuning rule.

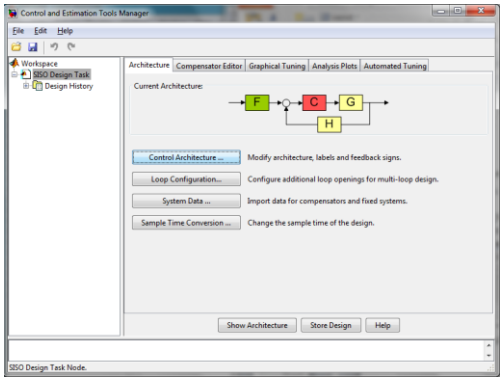


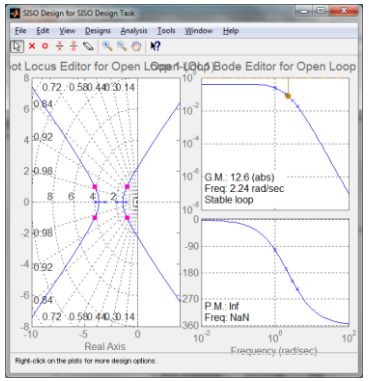
Automatic PID Tuning with SISOTool

1. Import system model into SISOTool

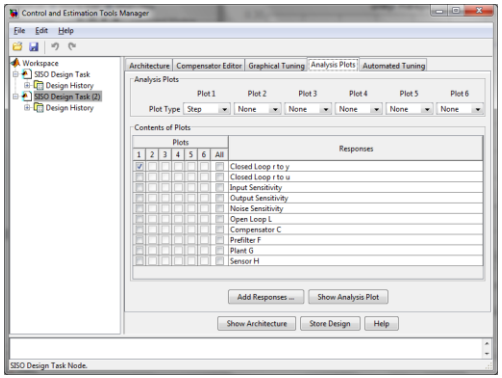
   Matlab code:

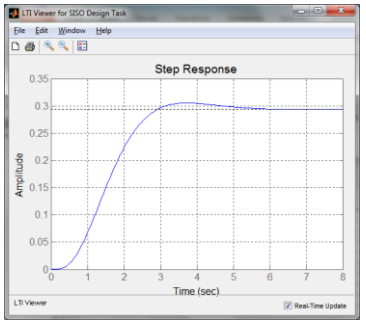
   ```
   s=tf('s'); G=10/(s+1)/(s+2)/(s+3)/(s+4);
   sisotool(G)
   ```





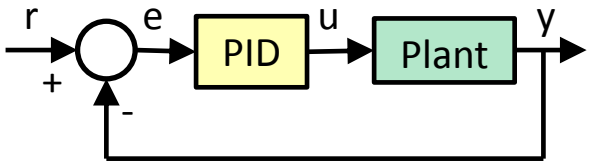
2. In CETM, from "Analysis Plot" tab, launch closed-loop step response



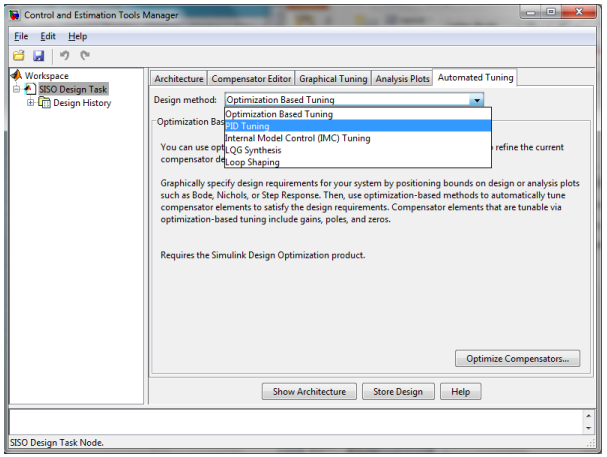


Automatic Tuning of PID Controller

Automatic PID Tuning with SISOTool …

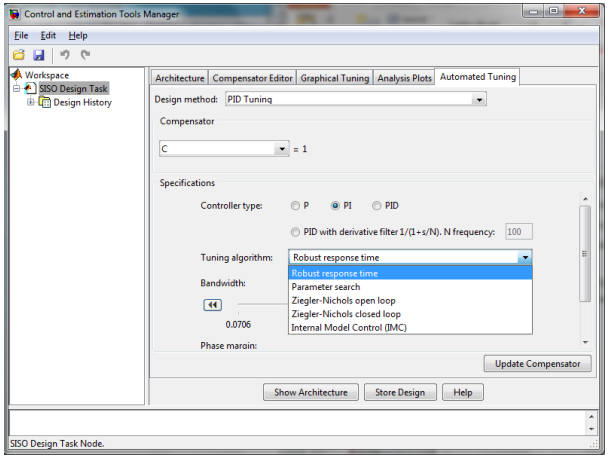


3. In CETM, from “Automated Tuning”tab, under “Optimization Based Tuning”, select “PID Tuning”



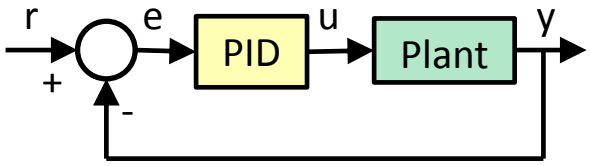
4. Choose the controller type:
   - P, PI, PID, PID with derivative filter
     - Here, choose “PID with derivative filter”

5. Select “Tuning algorithm”
   - Robust response time
   - Parameter search
   - Ziegler-Nichols open-loop
   - Ziegler-Nichols closed-loop
   - Internal Model Control (IMC)
     - Here, choose “Robust response time”

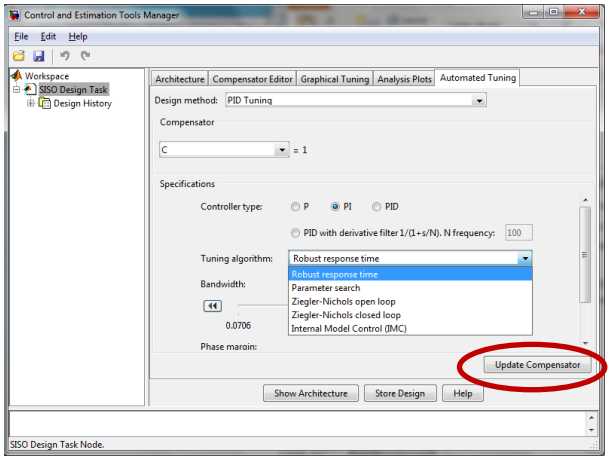


Automatic Tuning of PID Controller

Automatic PID Tuning with SISOTool …



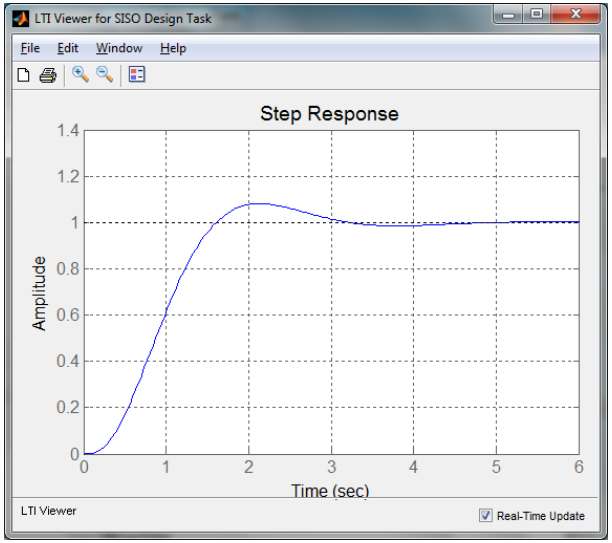
6. Click on “Update Compensator” button

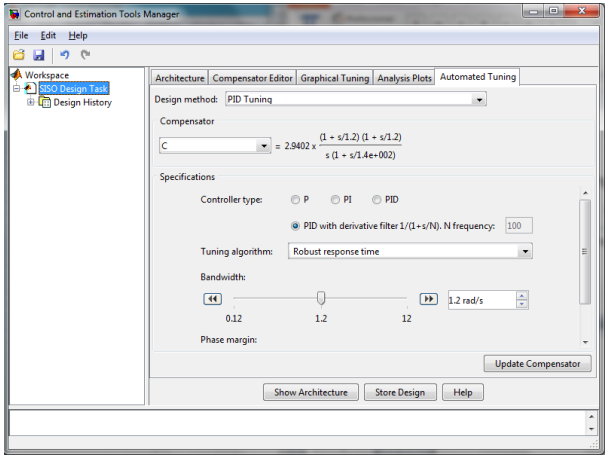


Tuned PID controller:

$$G_{cpid}(s) = 2.9402\,\frac{(1+\frac{s}{1.2})(1+\frac{s}{1.2})}{s(1+\frac{s}{1.4e^{0.002}})}$$

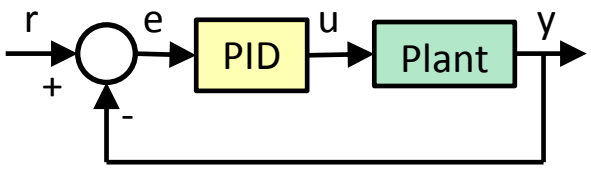
Closed-loop step response:





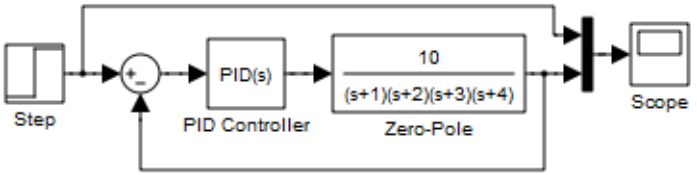
Automatic Tuning of PID Controller

Automatic PID Tuning with Simulink

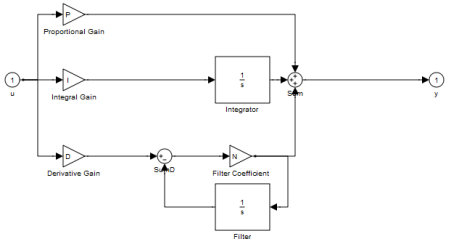


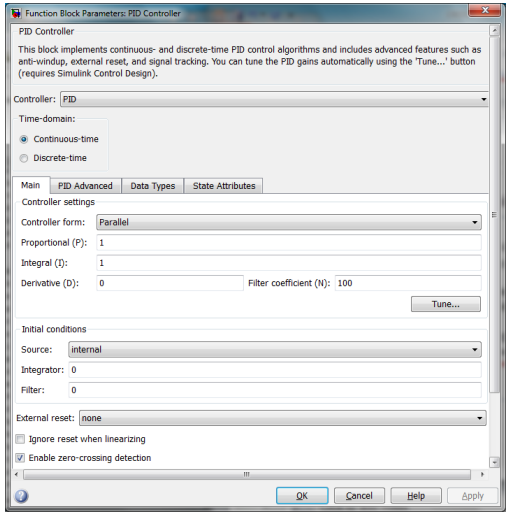
Example:

- For the fourth-order plant $G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)}$, design a PID controller using automated tuning in Simulink Control Design tool.



1. Build the system model in Simulink with a PID control block in a negative unity feedback structure
   - Add a “Step” input block and set its step-time = 0
   - Add a “Scope” and a “Mux” to view system response

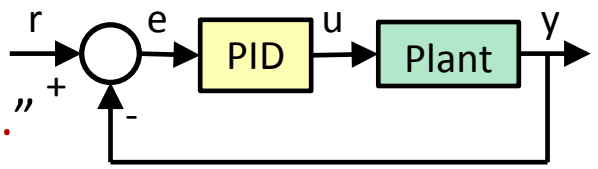




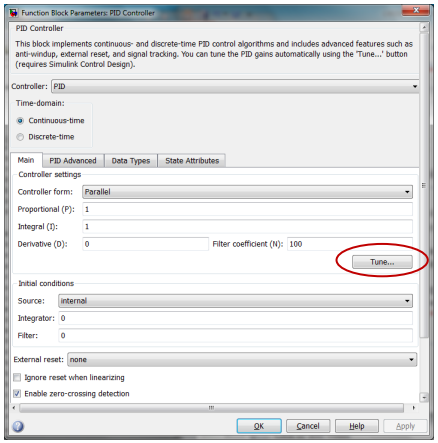
2. Double click on the PID block and choose:
   - Continuous-time or Discrete-time
     - Here, choose Continuous-time
   - PID, PI, PD, P, I
     - Here, choose “PID” (it includes derivative filter)

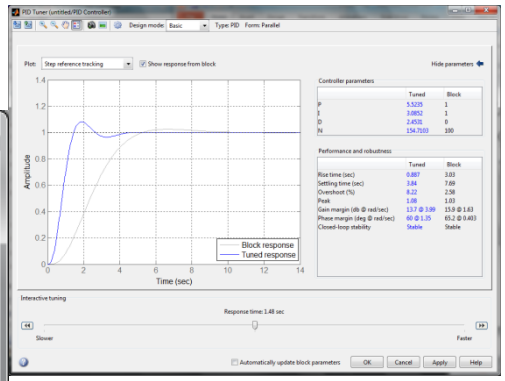
Automatic Tuning of PID Controller

Automatic PID Tuning with Simulink



3. In the PID block's parameter window, click on "Tune…" button
   - The "Step reference tracking" plot will appear in "PID Tuner" window
4. In "PID Tuner" window, press "Show parameters"
   - The window expands and shows the tuned parameter values
5. In "PID Tuner" window, you may
   - Adjust the response time with the slider
   - Select a different plot type
     - Step reference tracking
     - Step disturbance rejection
     - Open-loop Bode plot
     - Open-loop Nichols chart
   - Automatically update block parameters





Tuned PID controller:

$$G_{cpid}(s) = 3.1456 + \frac{2.0524}{s} + 0.7867\,\frac{8.9601}{1+\frac{8.9601}{s}}$$