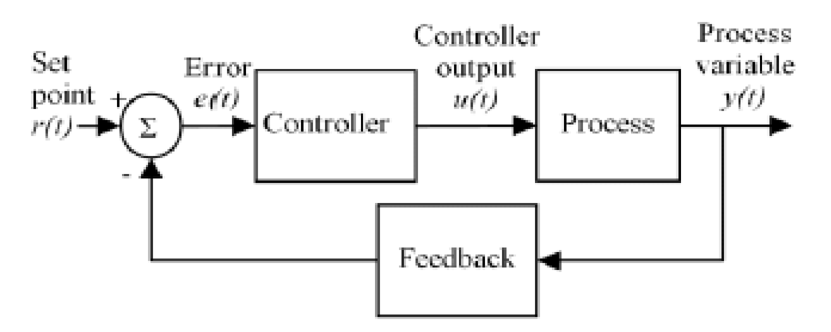


جَـامعة
المَـنارة
MANARA UNIVERSITY

Lecture (7-8)
PID Controller

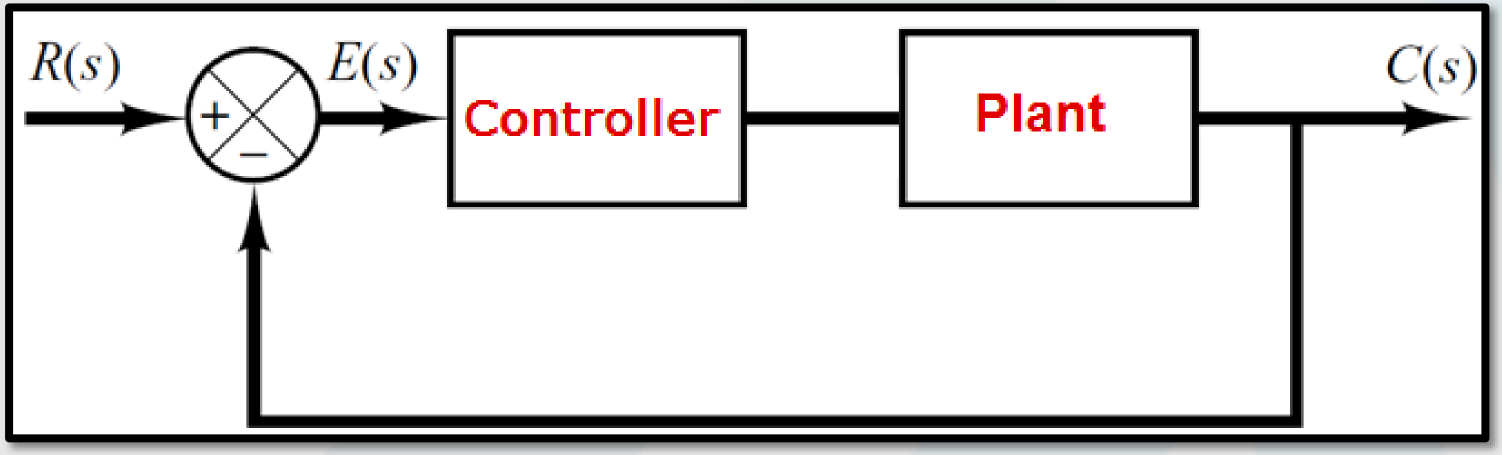
Mechatronics Engineering Department
Assistant Professor Isam Asaad

References

- Control Systems Course, professor Aniket Khandekar, Zeal college of engineering and Research, Pune.
- Gopal, M. - Control Systems_ Principles and Design 3rd edition-Tata McGraw Hill Publishing Co. Ltd. (2008)
- Modern Control Systems, Richard C. Dorf and Robert H. Bishop, Prentice Hall, 12th edition, 2010, ISBN-10: 0-13-602458-0
- Modelling, Dynamics and Control, University of Sheffield, John Anthony Rossiter.
- https://www.wevolver.com/article/mastering-pid-tuning-the-comprehensive-guide

Introduction

- This introduction will show you the characteristics of the each of proportional (P), the integral (I), and the derivative (D) controls, and how to use them to obtain a desired response.
- In this lecture, we will consider the following unity feedback system:



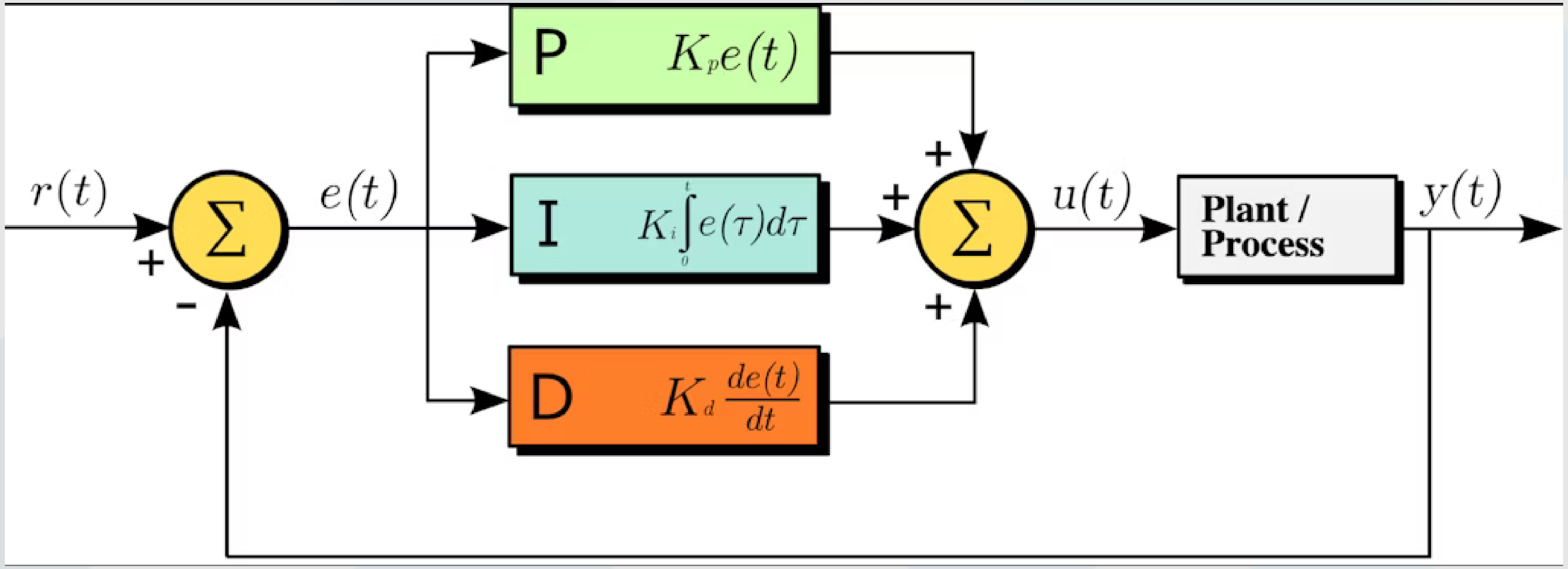
- **Plant**: A system to be controlled
- **Controller**: Provides the excitation for the plant; Designed to control the overall system behavior

The PID controller

- The transfer function of the PID controller looks like the following:

$$G_c(s) = K_p + K_D\, s + \frac{K_I}{s}$$

  - Kp = Proportional gain
  - K_I = Integral gain
  - Kd = Derivative gain



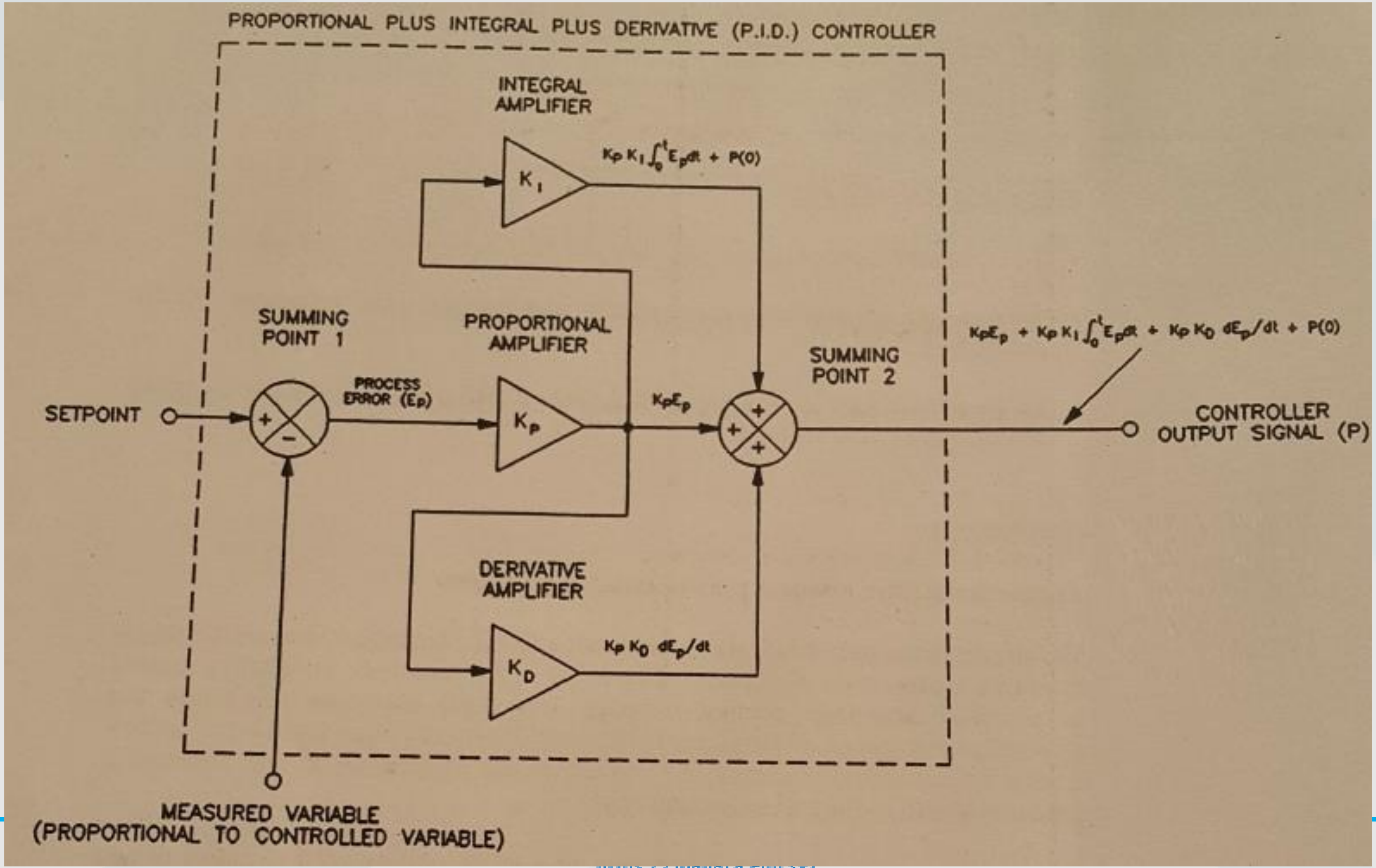
The PID controller

- First, let's take a look at how the PID controller works in a closed-loop system using the schematic shown above. The variable [E(s)] represents the tracking error, the difference between the desired input value [R(s)] and the actual output [C(s)].
- This error signal (e) will be sent to the PID controller, and the controller computes both the derivative and the integral of this error signal.
- The signal [U(s)] just past the controller is now equal to the proportional gain (Kp) times the magnitude of the error plus the integral gain (Ki) times the integral of the error plus the derivative gain (Kd) times the derivative of the error.

$$u(t) = K_p\, e(t) + K_I \int e(t).dt + K_D \frac{d\, e(t)}{dt}$$

- This signal (u) will be sent to the plant, and the new output will be obtained. This new output will be sent back to the sensor again to find the new error signal (e). The controller takes this new error signal and computes its derivative and its integral again.
- This process goes on and on.

The PID controller

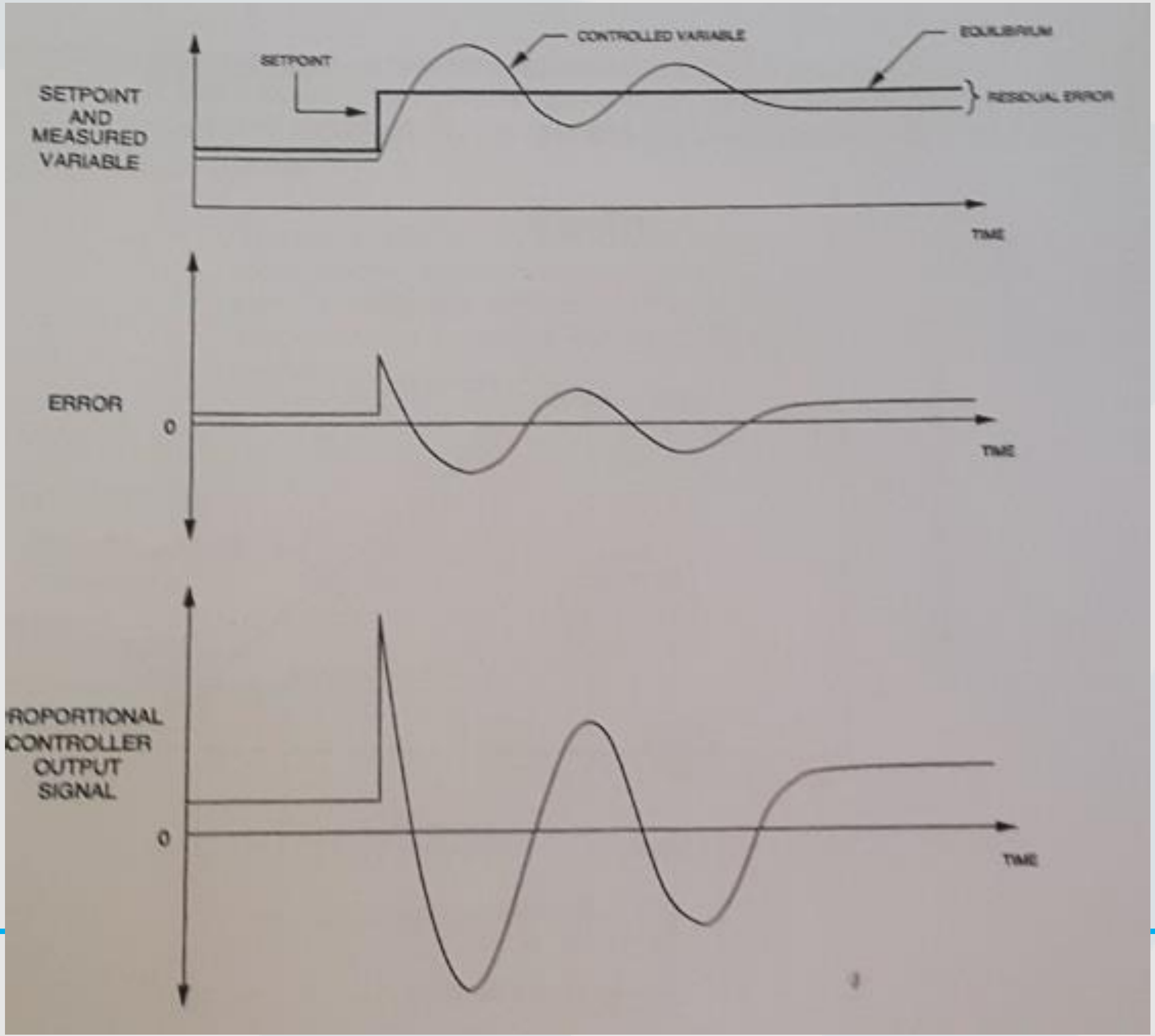


The characteristics of P, I, and D controllers

- A proportional controller (Kp) will have the effect of reducing the rise time and will reduce ,but never eliminate, the steady-state error.
- An integral control (Ki) will have the effect of eliminating the steady-state error, but it may make the transient response worse.
- A derivative control (Kd) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response.
- Effects of each of controllers Kp, Kd, and Ki on a closed-loop system are summarized in the table shown below.

	RISE TIME	OVERSHOOT	SETTLING TIME	Steady-State Response
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

Proportional Controller



$$P = K_p . E_p + P_o$$

where

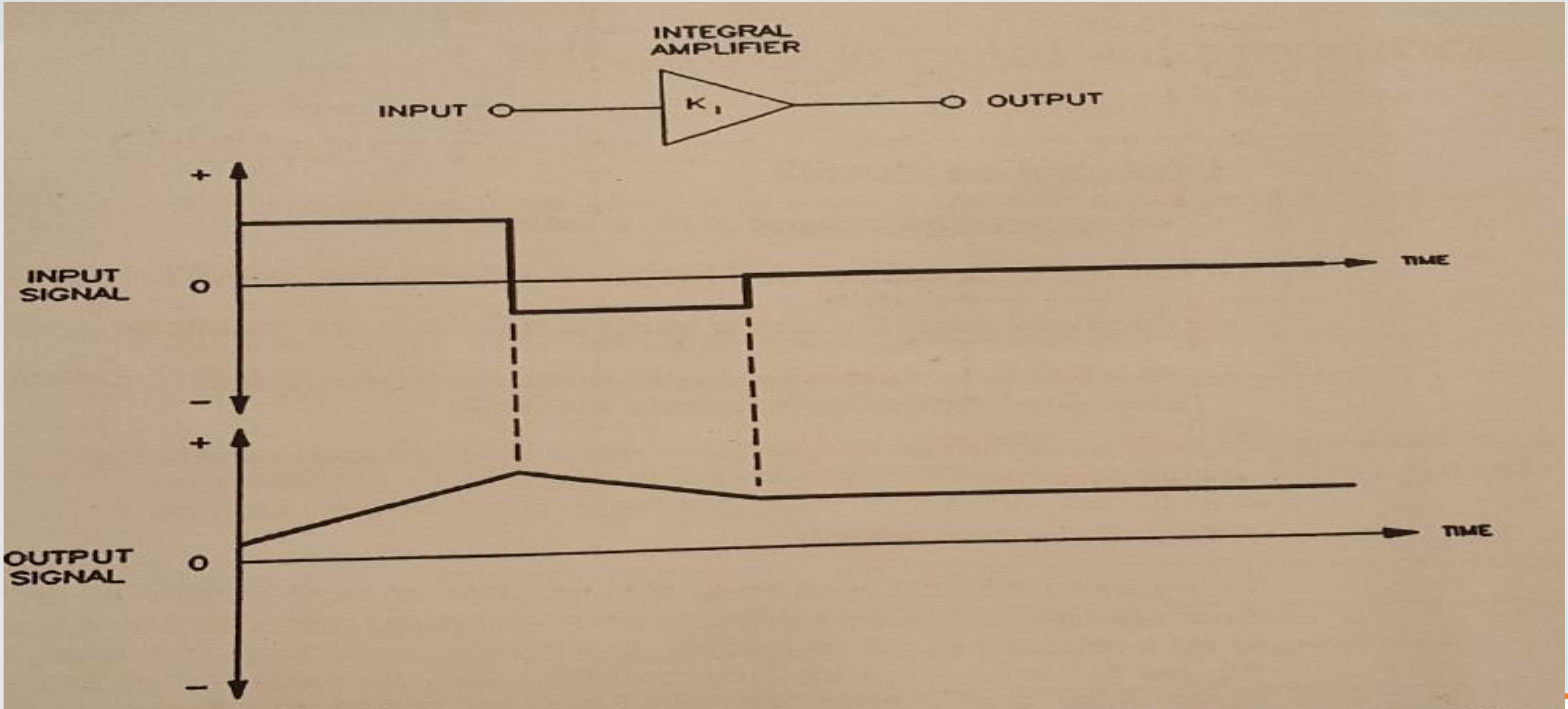
P = controller output, including error

P_o= controller output without error (%)

E_P = error signal in proportional control action

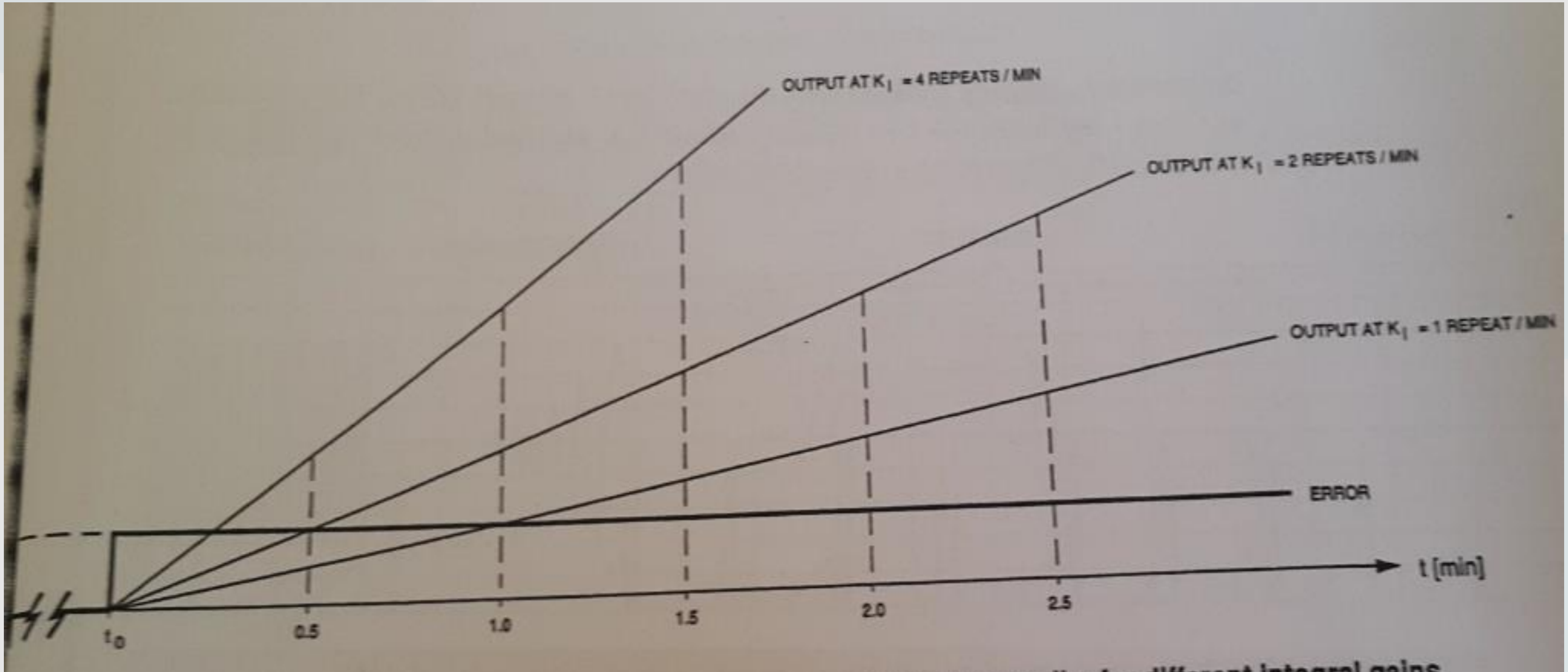
K_P = proportional controller gain or proportional constant between error and controller output.

Integral Controller



Example of the signal waveforms at the input and output of an integral amplifier.

Integral gain and integration time:

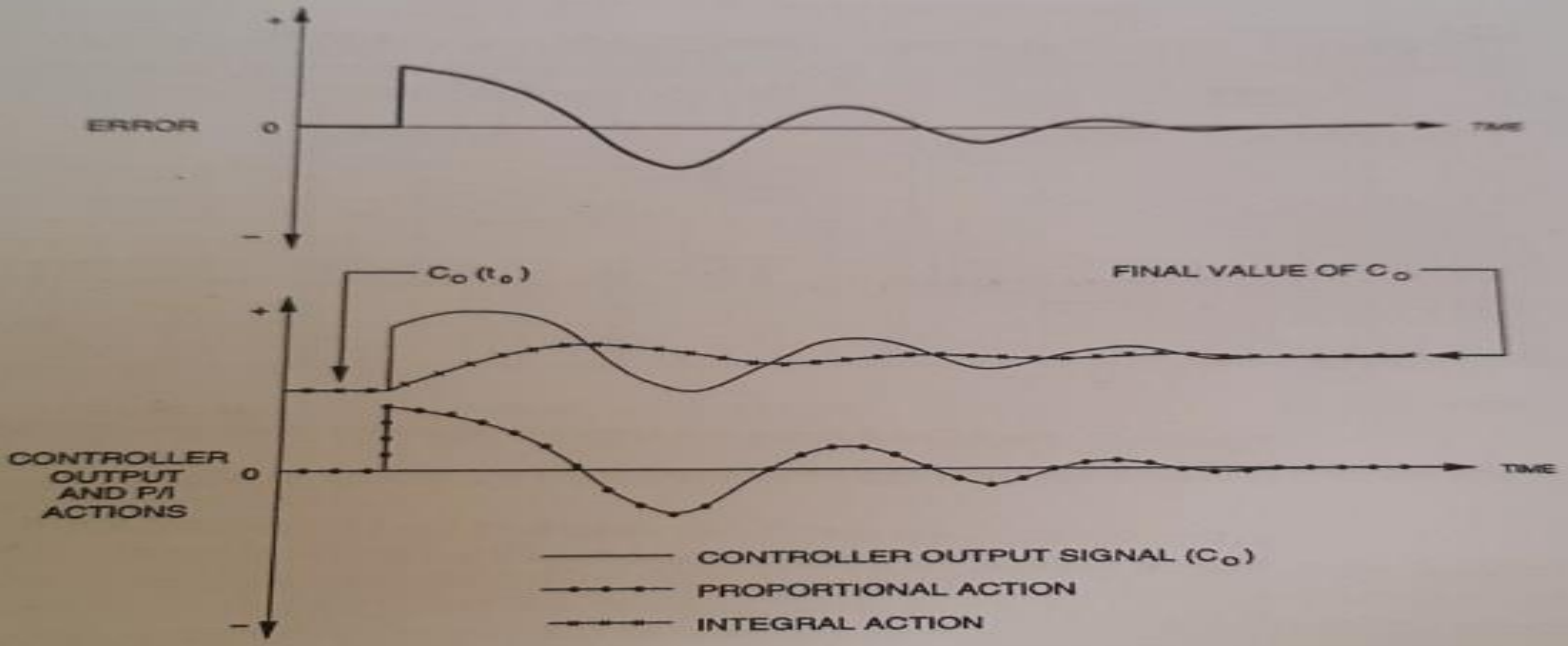


Open-loop step response of an integral controller for different integral gains.

T_i=1/Ki [min]

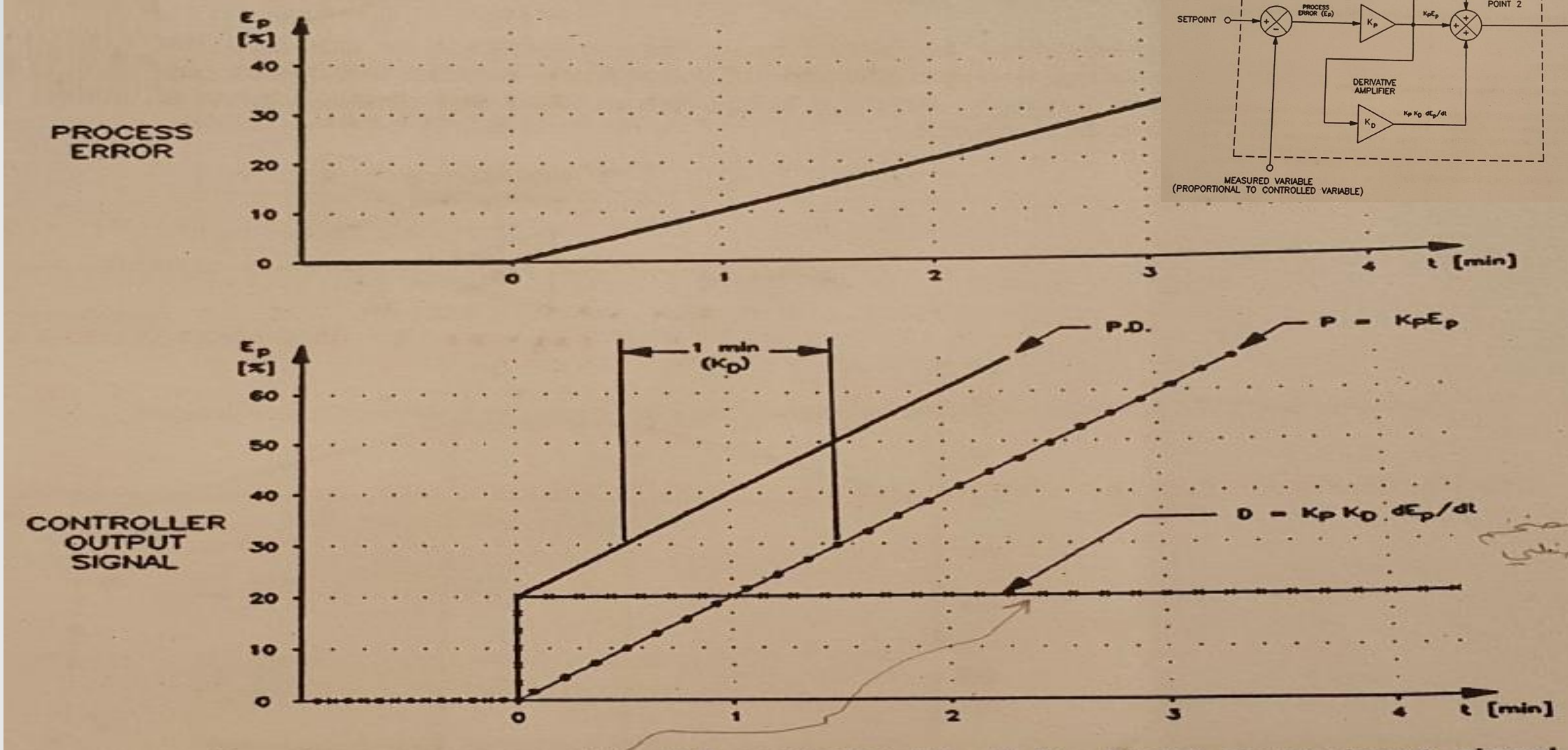
- In the context of integral control, windup refers to a situation where the integral term continues to accumulate an error even when the system has reached a saturation limit (maximum or minimum output). Windup can occur when the integral action is unable to correct the error due to constraints on the manipulated variable. This situation can lead to overshooting and prolonged settling times.

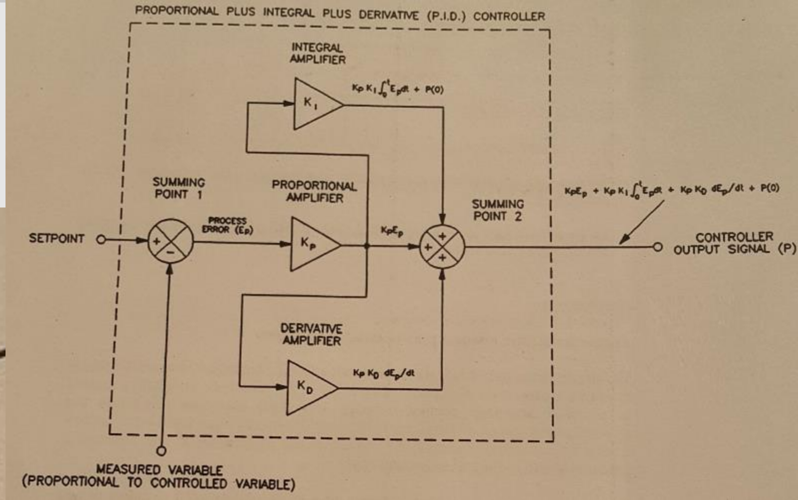
How a PID controller work:



. Example of what happens in a proportional-plus-integral control system when the setpoint is changed suddenly.

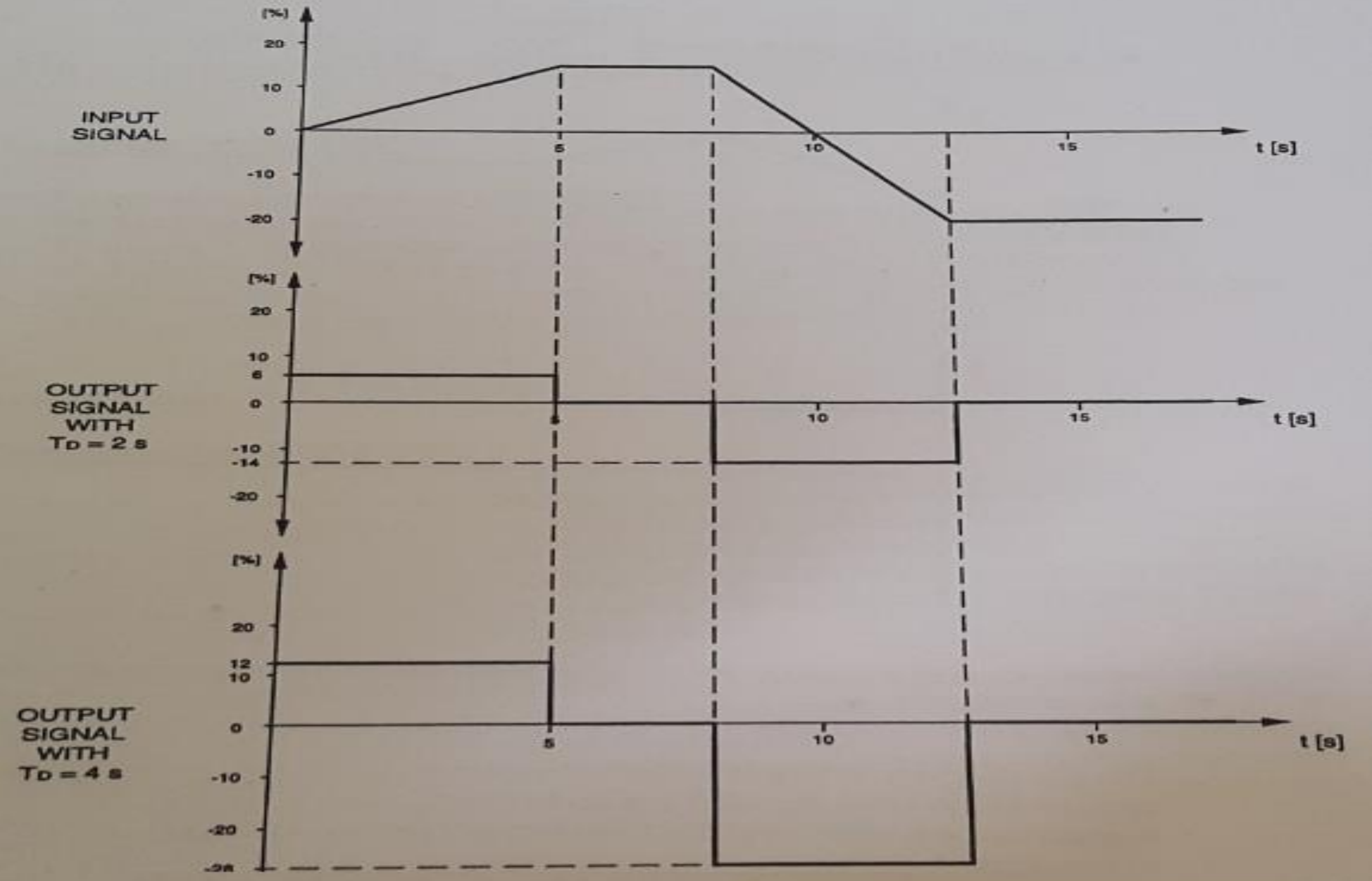
Derivative time:





Open-loop response of a P.D. controller ($K_p = 2$, $K_D = 1$ min) to an input signal changing at a rate of 10%/min.

يصبح التضخيم التفاضلي الكلي
$= K'_D = K_p * K_D = 2 * 1 = 2$ min

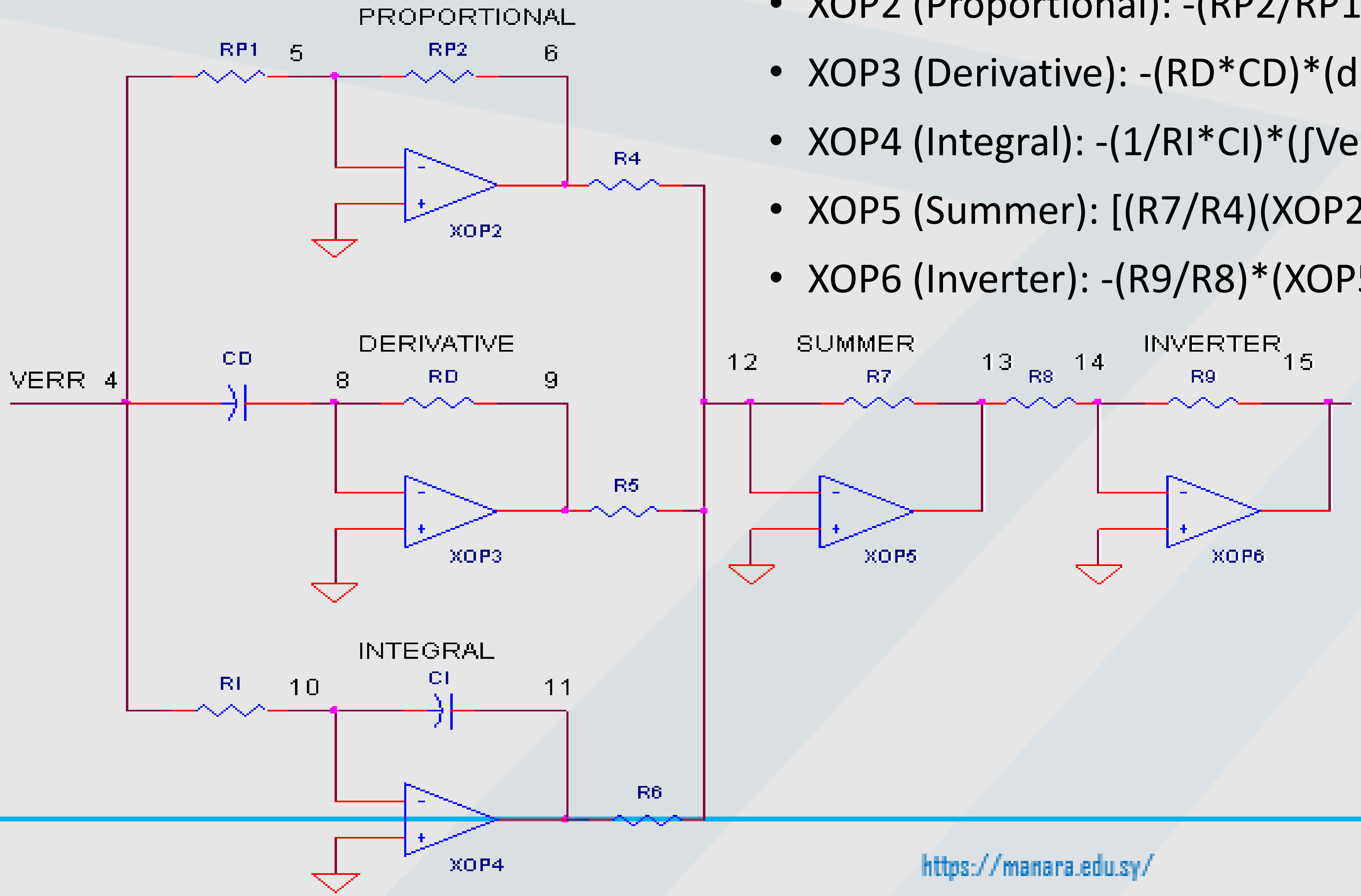


Output signal of a derivative mode section for two different derivative times T_D.

$K_d = T_d$ [min]

PID Circuit:

- XOP2 (Proportional): -(RP2/RP1)(Verr)
- XOP3 (Derivative): -(RD*CD)*(d(VERR)/dt)
- XOP4 (Integral): -(1/RI*CI)*(∫Verr dt)
- XOP5 (Summer): [(R7/R4)(XOP2) + (R7/R5)(XOP3) + (R7/R6)*(XOP4)]
- XOP6 (Inverter): -(R9/R8)*(XOP5)

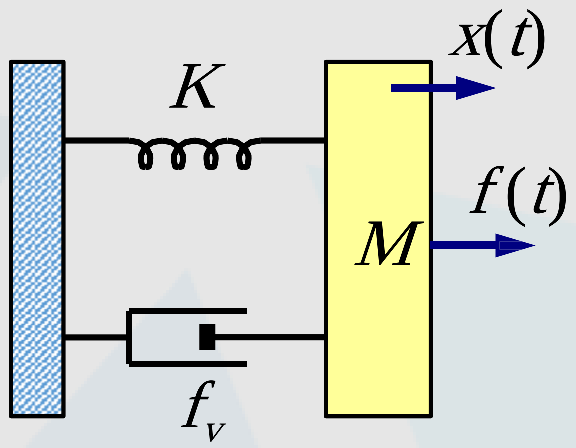


Example Problem

- Suppose we have a simple mass, spring, and damper problem.
- The modeling equation of this system is

$$M\frac{d^2x(t)}{dt^2} + f_v\frac{dx(t)}{dt} + Kx(t) = f(t)$$

$$\frac{X(s)}{F(s)} = \frac{1}{M s^2 + f_v s + K}$$

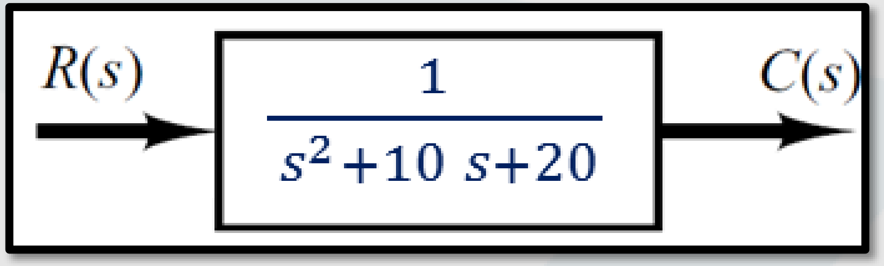


- Let
  - M = 1kg
  - f_v = 10 N.s/m
  - k = 20 N/m
  - F(s) = 1 = unit step

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

Open-loop step response

- Let's first view the open-loop step response.



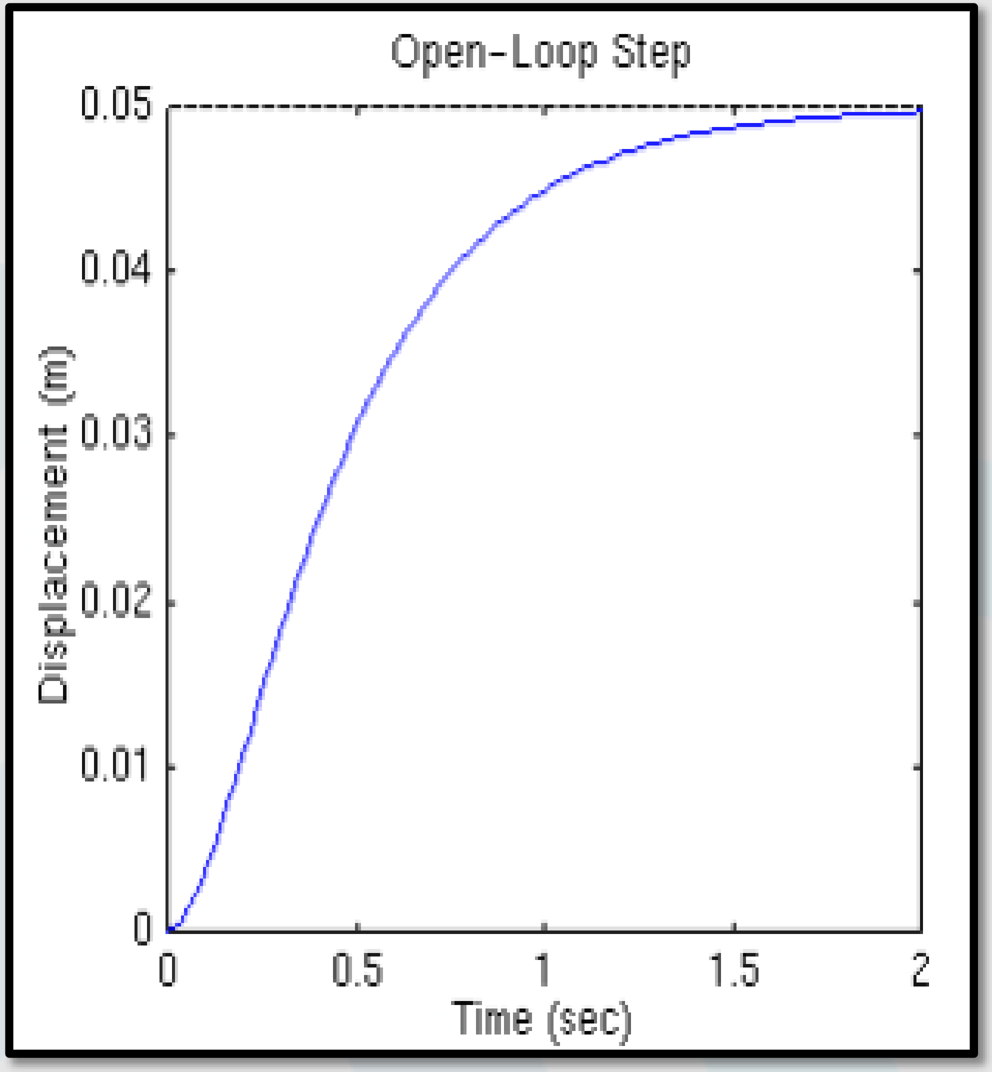
- Create a new m-file and add in the following code:

```
>> num=1;
>> den=[1 10 20];
>> step (num,den)
```

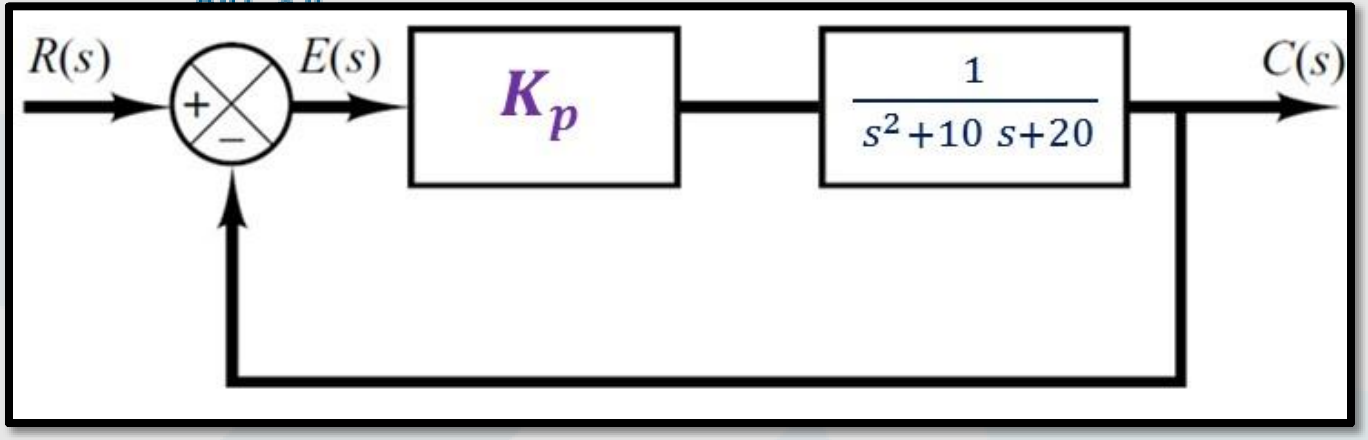
- Running this m-file in the Matlab command window should give you the plot shown below.

Open-loop step response

- The DC gain of the plant transfer function is 1/20, so 0.05 is the final value of the output to an unit step input. This corresponds to the steady-state error of 0.95, quite large indeed.
- Furthermore, the rise time is about one second, and the settling time is about 1.5 seconds.
- Let's design a controller that will reduce the rise time, reduce the settling time, and eliminates the steady-state error.



Closed Loop with P-Controller



- The closed-loop T.F is

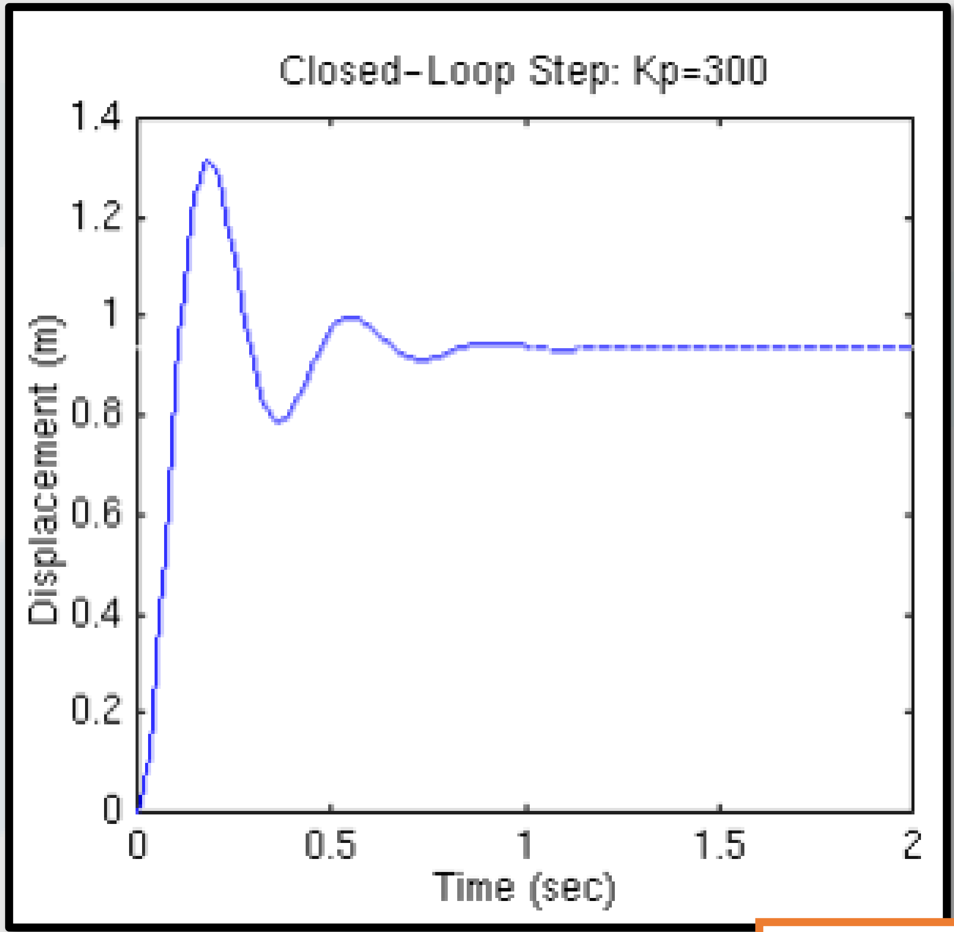
$$\frac{C(s)}{R(s)} = \frac{K_p}{s^2 + 10\,s + (20 + K_p)}$$

- Let the proportional gain (Kp) equals 300 and change the m-file to the following:

```
>> Kp =300;
>> num = [Kp];
>> den = [1 10 20+Kp];
>> t = 0:0.01:2;
>> step (num,den,t)
```

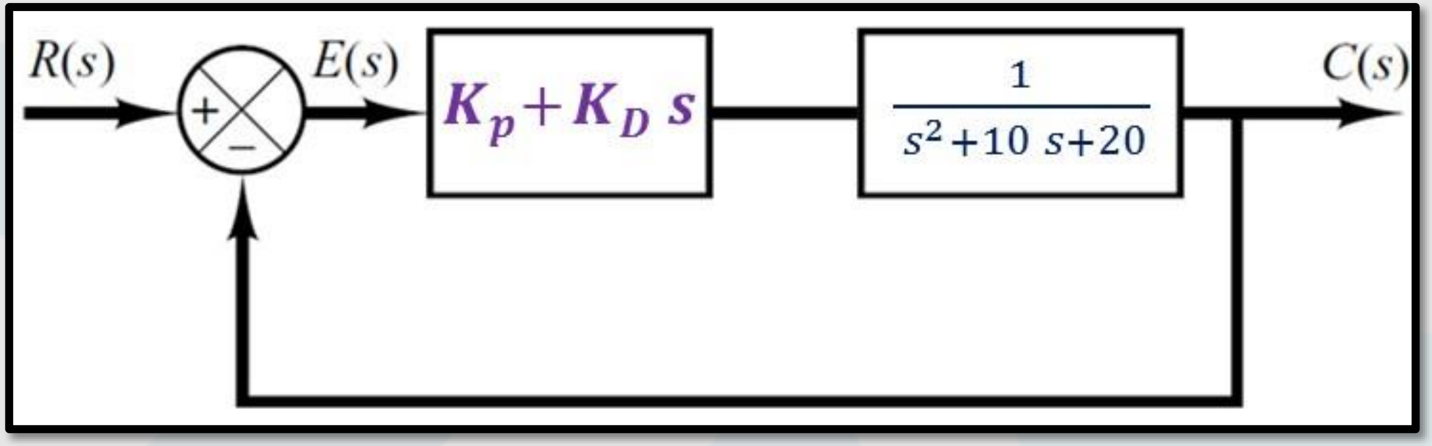
Closed Loop with P-Controller

- Running this m-file in the Matlab command window should give you the following plot.



The plot shows that the proportional controller reduced both the rise time and the steady-state error, increased the overshoot, and decreased the settling time by small amount.

Closed Loop with PD-Controller



- The closed-loop T.F is

$$\frac{C(s)}{R(s)} = \frac{K_p + K_D\,S}{s^2 + (10 + K_D)\,s + (20 + K_p)}$$

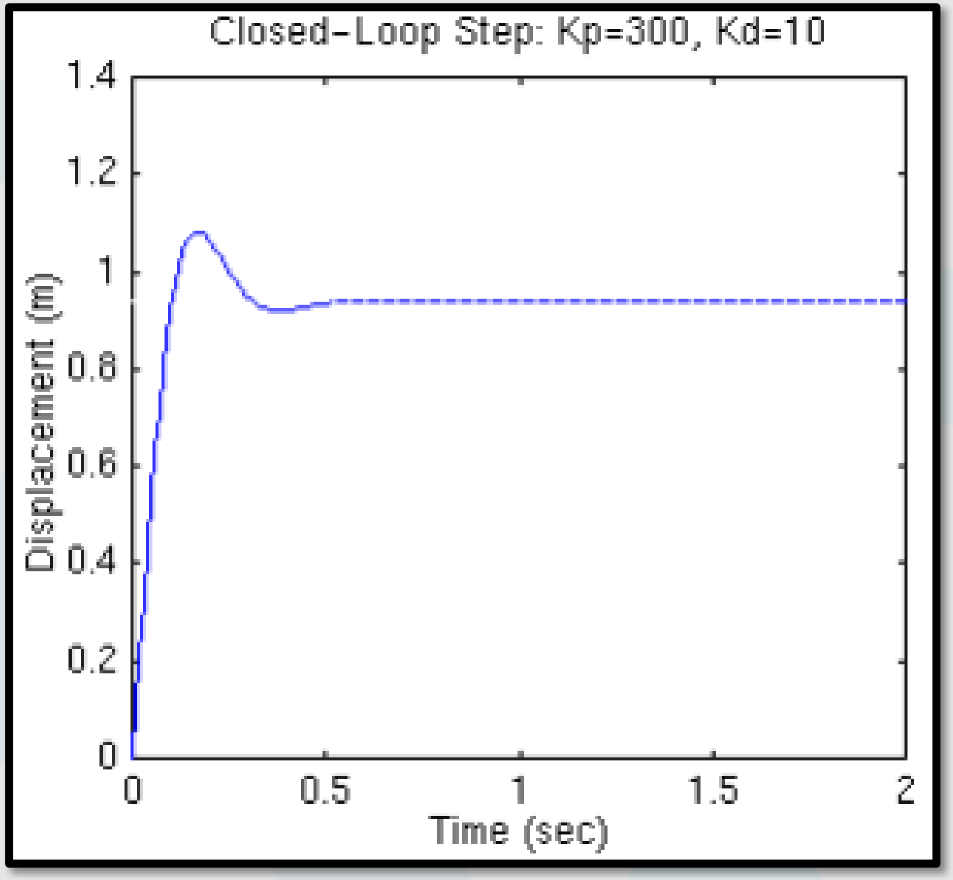
- Let Kp equals 300 and Kd equals 10, then change the m-file to the following:

```
>> Kp =300; KD=10;
>> num = [KD   Kp];
>> den = [1 10+KD 20+Kp];
>> t = 0:0.01:2;
>> step (num,den,t)
```

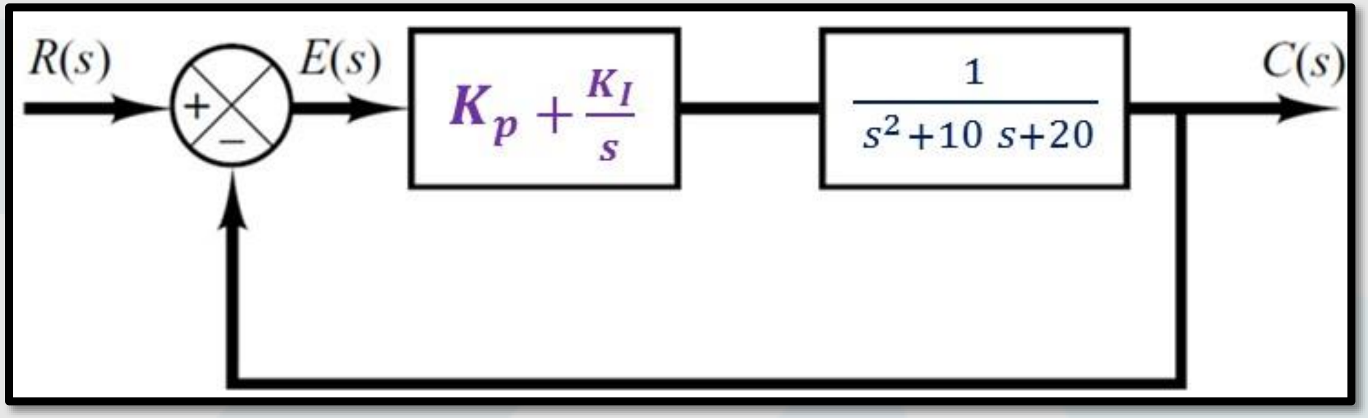
Closed Loop with PD-Controller

- ❑ Running this m-file in the Matlab command window should gives you the following plot.

The plot shows that the derivative controller reduced both the overshoot and the settling time, and had small effect on the rise time and the steady-state error.



Closed Loop with PI-Controller



- The closed-loop T.F is

$$\frac{C(s)}{R(s)} = \frac{K_p\, s + K_I}{s^3 + 10\, s^2 + (20 + K_p)\, s + K_I}$$

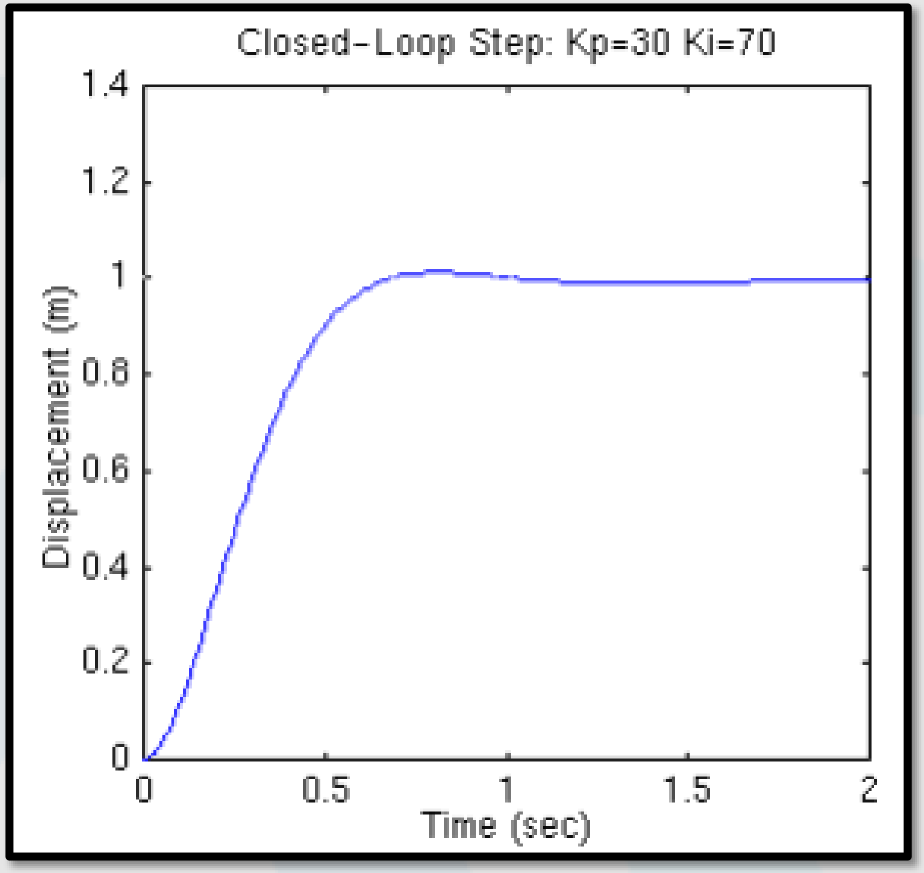
- Let Kp equals 300 and Ki equals 70, then change the m-file to the following:

```
>> Kp = 300; KI = 70;
>> num = [Kp  KI];
>> den = [1 10 20+Kp KI]
>> t = 0:0.01:2;
>> step (num,den,t)
```

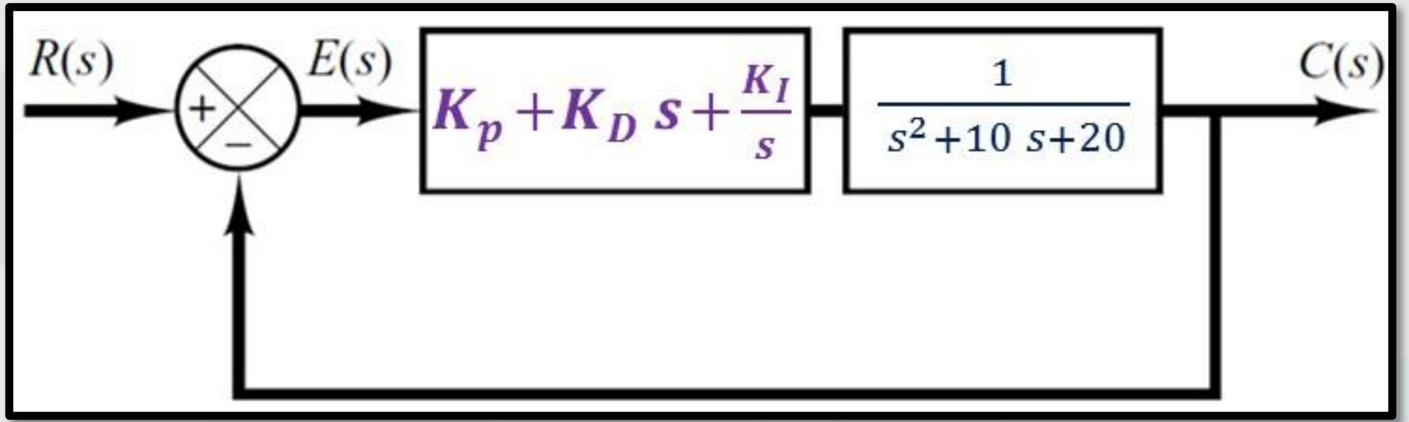
Closed Loop with PI-Controller

- Running this m-file in the Matlab command window should gives you the following plot.

- We have reduced the proportional gain (Kp) because the integral controller also reduces the rise time and increases the overshoot as the proportional controller does (double effect). The above response shows that the integral controller eliminated the steady-state error.



Closed Loop with PID-Controller



- The closed-loop T.F is

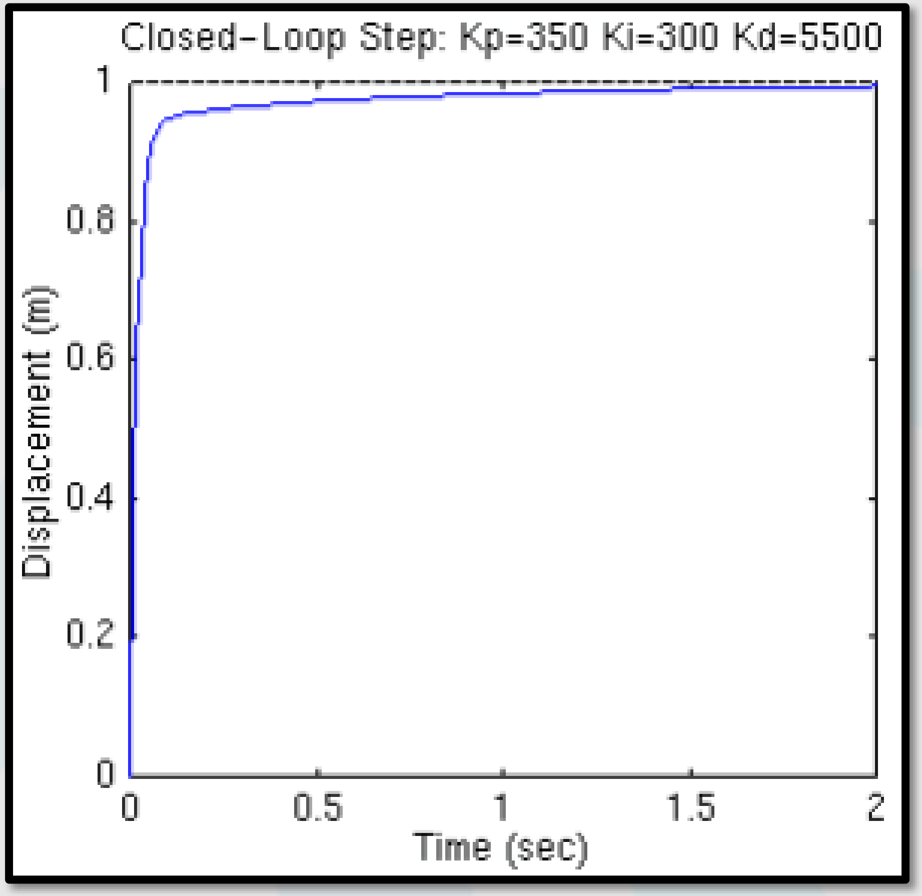
$$\frac{C(s)}{R(s)} = \frac{K_D s^2 + K_p s + K_I}{s^3 + (10 + K_D)s^2 + \left(20 + K_p\right) s + K_I}$$

- Let Kp equals 350, Kd equals 50 and Ki equals 300, then change the m-file to the following:

```
>> Kp = 350; KI = 300; KD = 50;
>> num = [KD   Kp   KI];
>> den = [1 10+KD 20+Kp KI];
>> t = 0:0.01:2;
>> step (num,den,t)
```

Closed Loop with PID-Controller

- Running this m-file in the Matlab command window should gives you the following plot.



Now, we have obtained the system with no overshoot, fast rise time, and no steady- state error.

PID Tuning:

- What is the process of PID tuning?
  - Choosing the proper values for P, I, and D is called "PID Tuning“.
  - Used to get a desired and stable response of the controlled variable.
- PID Tuning Methods:

  1) There are lots of methods.
  2) We will use three basic and common methods:
     1) Manual method.
     2) Ziegler-Nichols method.
        1) Open-Loop method.
        2) Closed-Loop method.
     3) Using MATLAB.

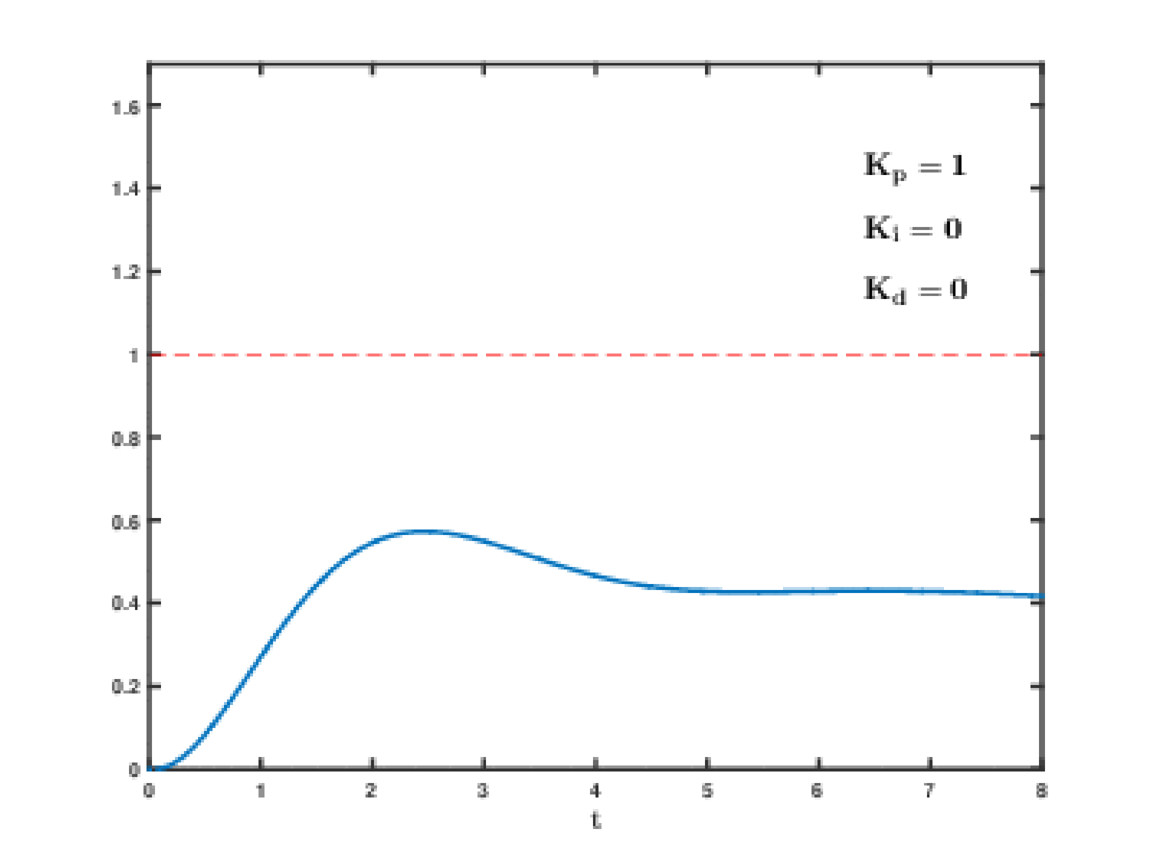
PID Tuning (Manual method):

- Step-by-step guide to manual PID tuning:

1. Set all gains to zero (Kp = 0, Ki = 0, Kd = 0).
2. Increase Kp until the system responds to setpoint changes with acceptable speed, but without excessive overshoot.
3. Increase Ki gradually to eliminate steady-state error. Watch for oscillations or instability.
4. If needed, introduce Kd to reduce overshoot and dampen oscillations. Be cautious, as too much derivative action can introduce noise sensitivity.
5. Fine-tune all parameters iteratively, making small adjustments and observing the system response.
6. Test the system with various setpoints and disturbances to ensure robust performance.

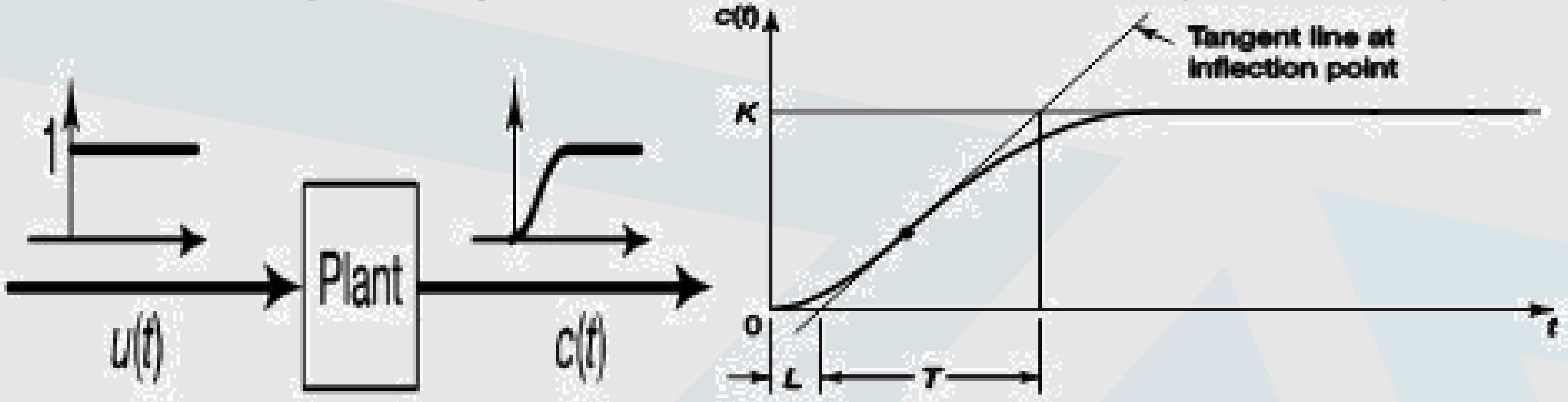
	RISE TIME	OVERSHOOT	SETTLING TIME	Steady-State Response
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

PID Tuning (Manual method):



	RISE TIME	OVERSHOOT	SETTLING TIME	Steady-State Response
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

PID Tuning -Ziegler-Nichols method(Open-Loop):



Ziegler–Nichols Tuning Rule Based on Step Response of Plant (First Method)

Type of Controller	K_p	T_i =1/Ki	T_d =Kd
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

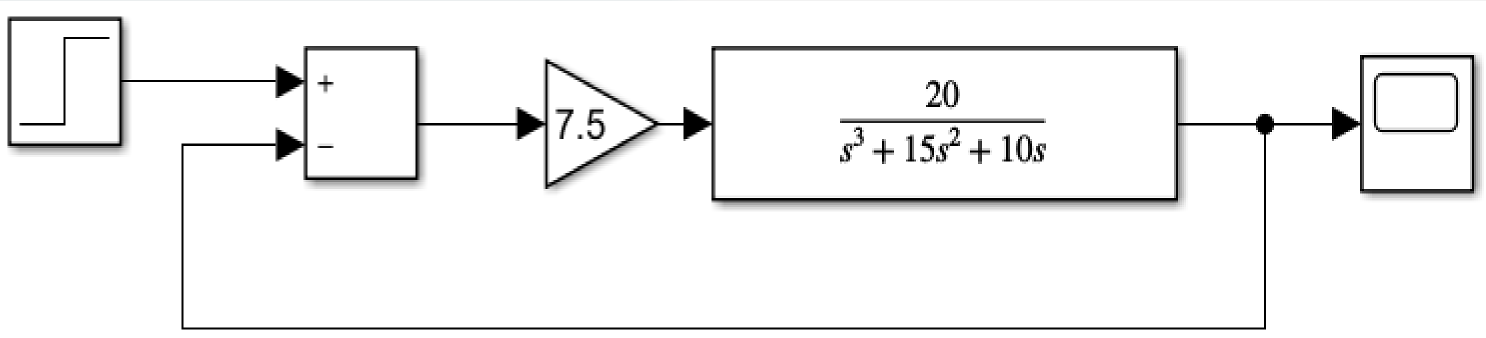
PID Tuning -Ziegler-Nichols method(Closed Loop):

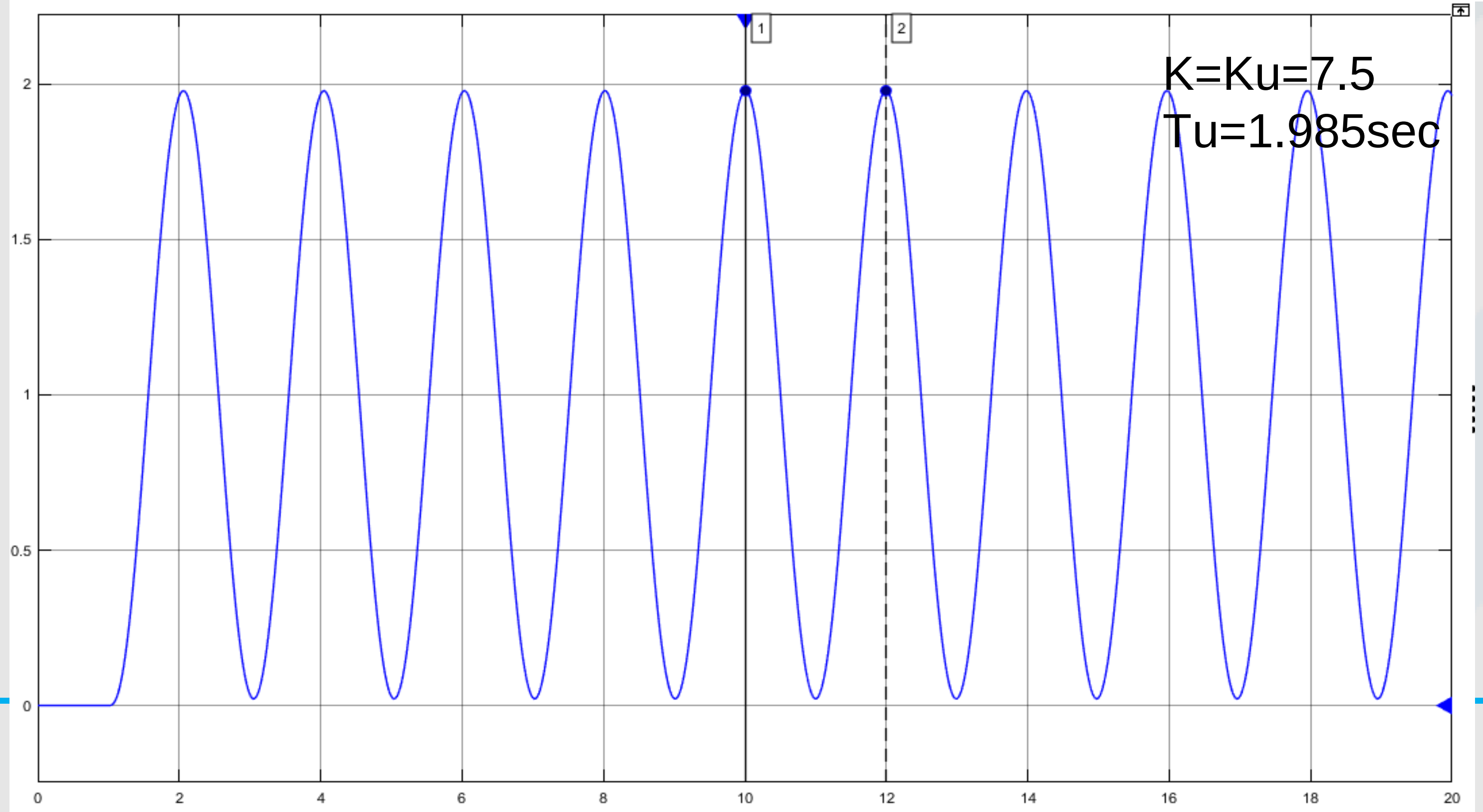
❑A practical walkthrough of the Ziegler-Nichols method:

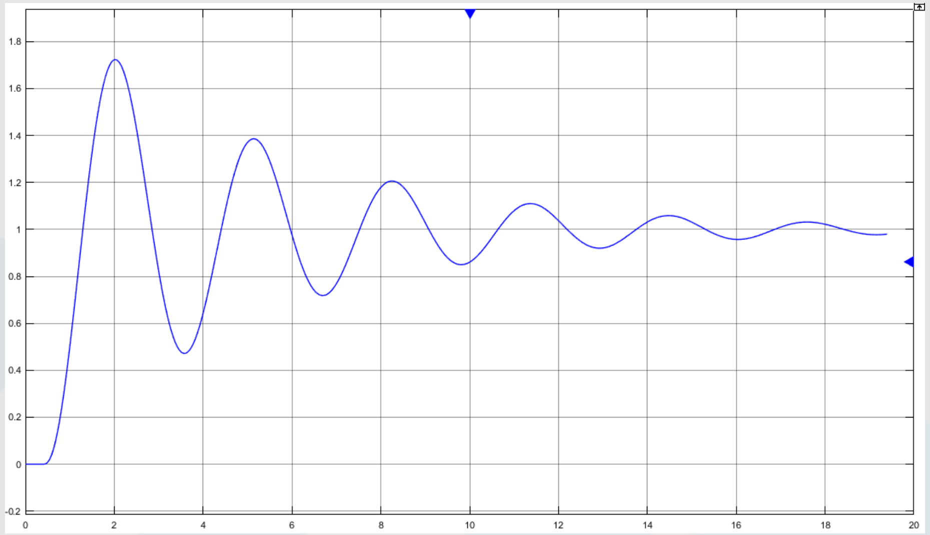
1. Disable integral and derivative actions (set Ti = ∞ and Td = 0).
2. Increase Kp=Kc until the system exhibits sustained oscillations. This gain is the ultimate gain (Ku)>>Calculate Ku manually or using Routh method.
3. Measure the period of oscillations (Tu).
4. Calculate PID parameters based on Ziegler-Nichols method:

Control type	Kp	Ki	Kd
P	0.50Ku	—	—
PI	0.45Ku	0.54Ku/Tu	—
PID	0.60Ku	1.2Ku/Tu	3KuTu/40

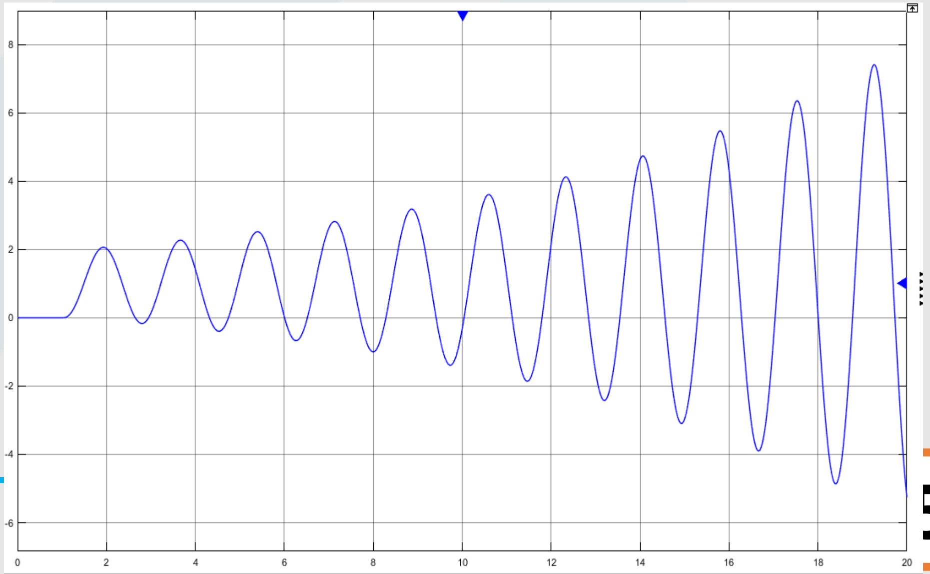
PID Tuning (Ziegler-Nichols method) Example1:







https://www.youtube.com/watch?v=MRA-yt22j5I



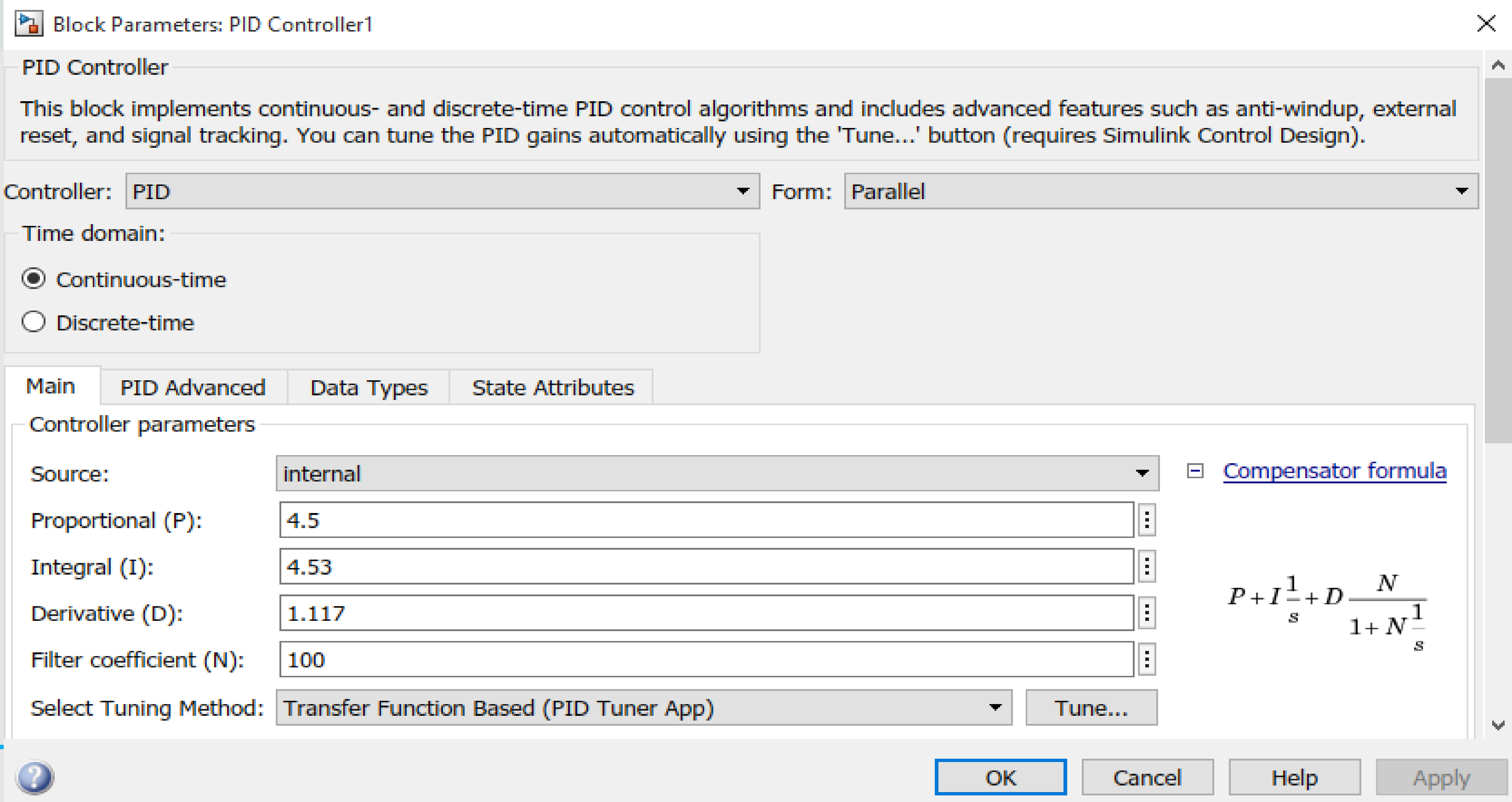
PID Tuning (Ziegler-Nichols method) Example1:

Manual Method:

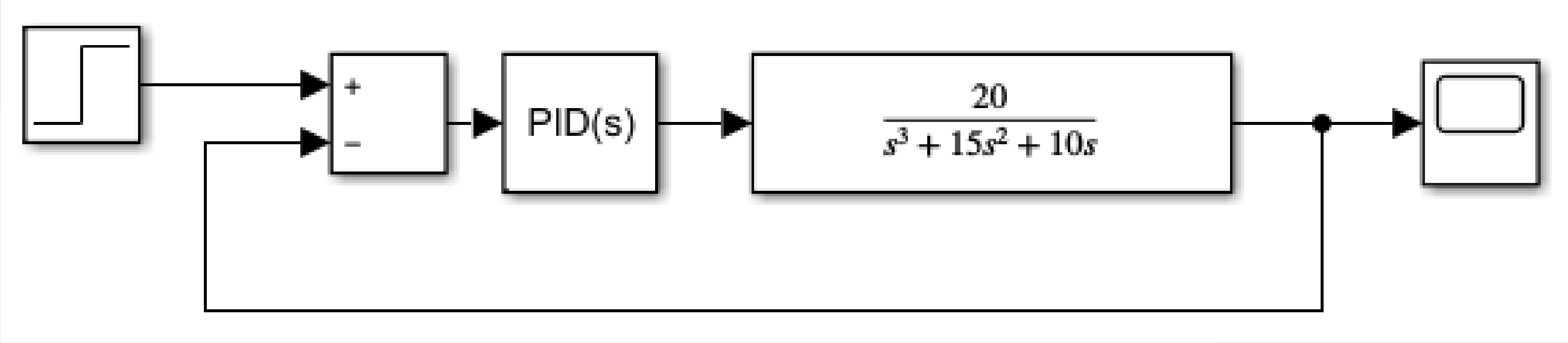
Kp = 0.60 x Kc➔Kp = 0.6 x 7.5➔Kp = 4.5
Ki=1.2xKc/Tu=1.2x7.5/1.985➔Ki=4.53
Kd=3x Kc x Tu/40=3x7.5x1.985/40➔Kd=1.117

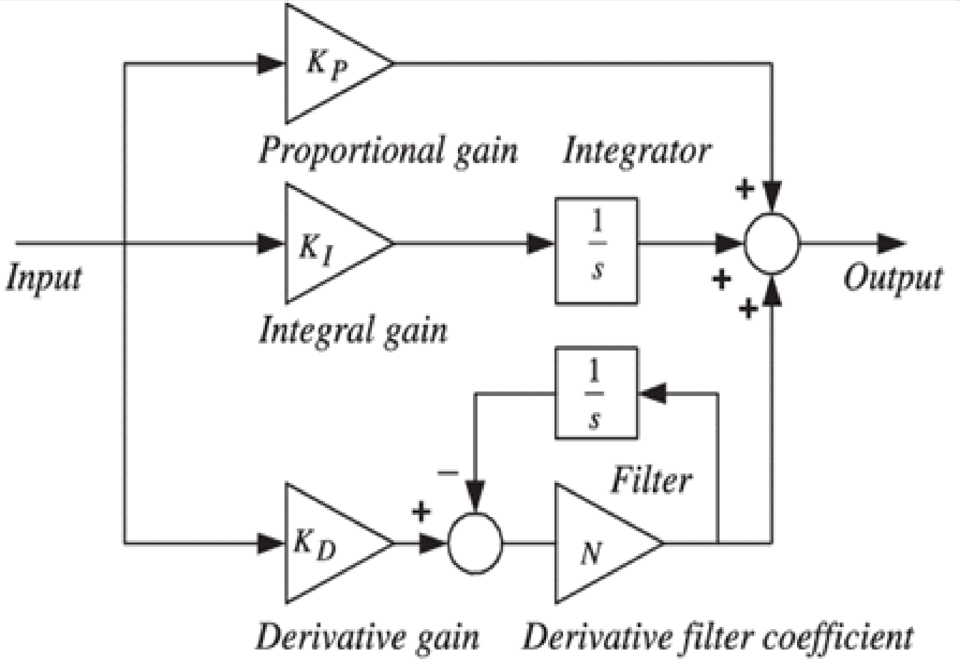
Control type	Kp	Ki	Kd
P	0.50Ku	—	—
PI	0.45Ku	0.54Ku/Tu	—
PID	0.60Ku	1.2Ku/Tu	3KuTu/40

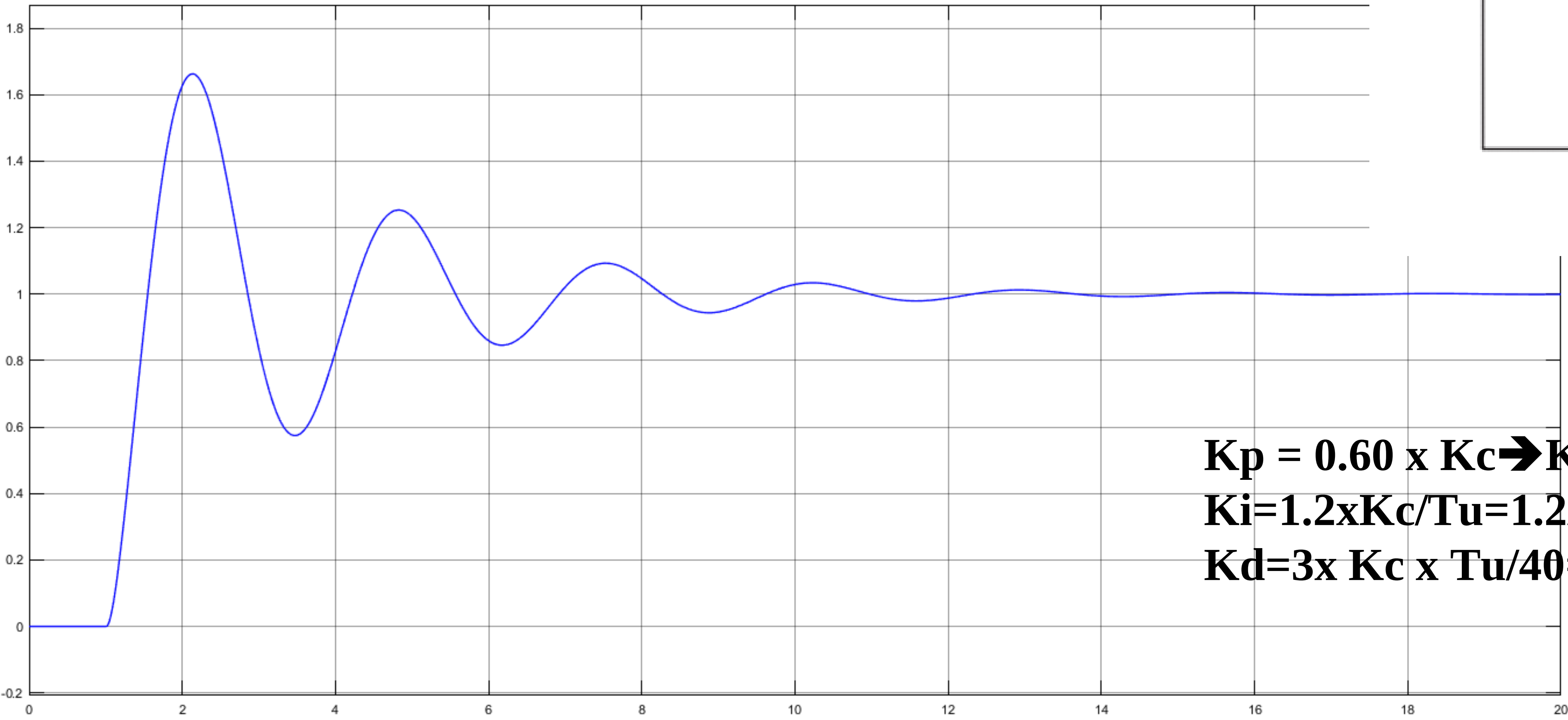
PID Tuning (Ziegler-Nichols method) Example1:



PID Tuning (Ziegler-Nichols method) Example1:

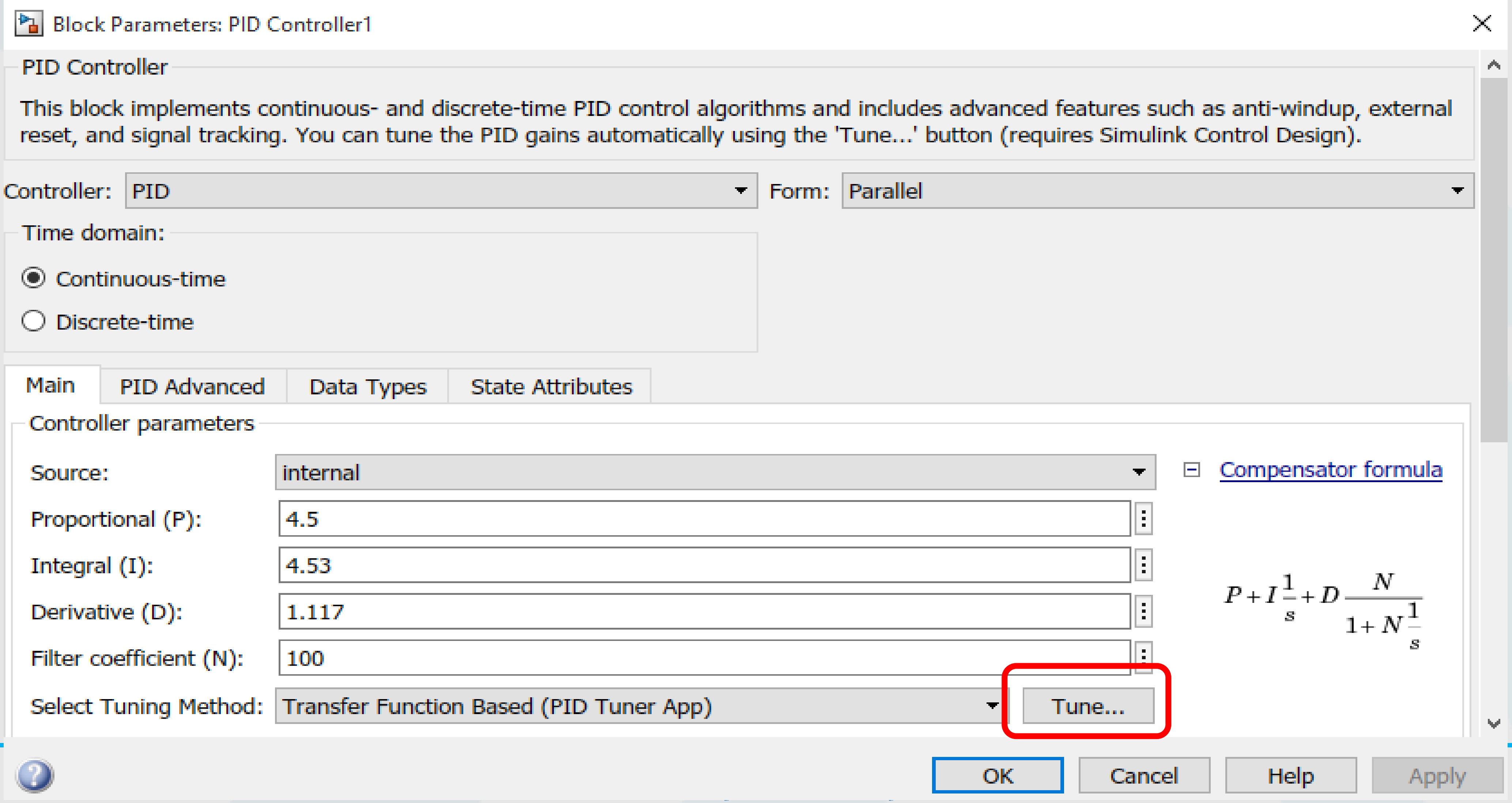






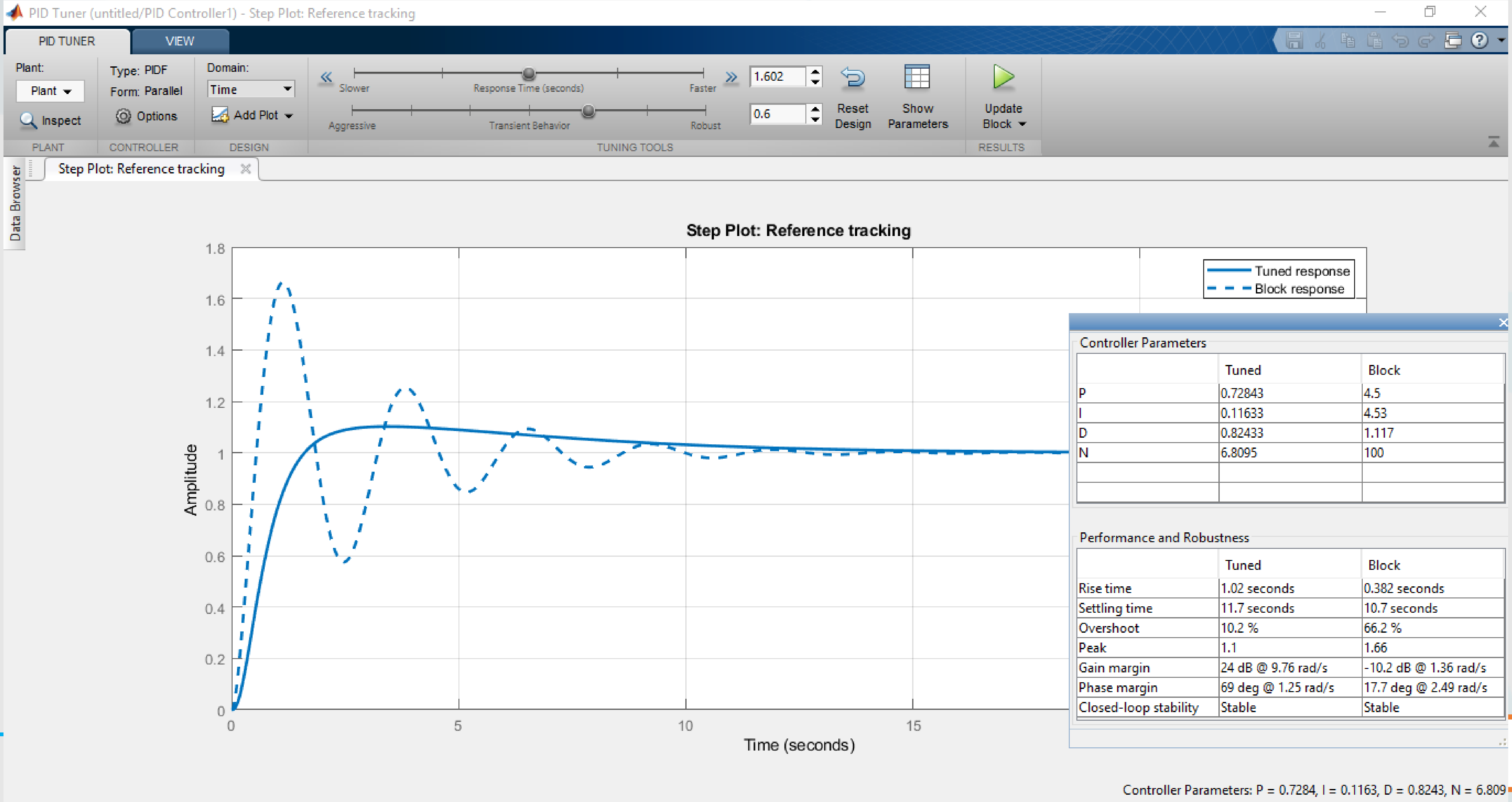
Kp = 0.60 x Kc➔Kp = 0.6 x 7.5➔Kp = 4.5
Ki=1.2xKc/Tu=1.2x7.5/1.985➔Ki=4.53
Kd=3x Kc x Tu/40=3x7.5x1.985/40➔Kd=1.117

PID Tuning (Ziegler-Nichols method) Example1:



$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

PID Tuning using MATLAB PID Tuner (Example1):



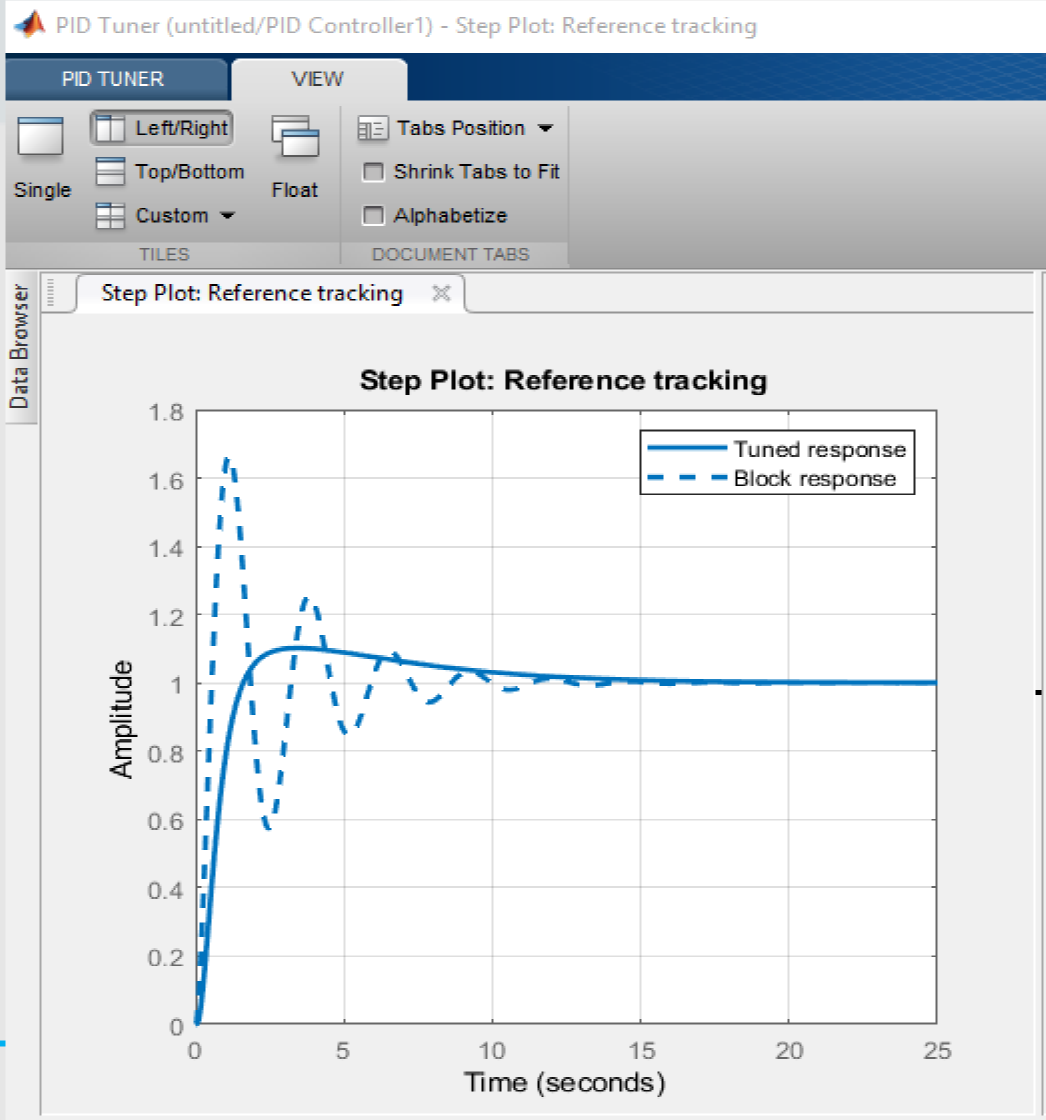
Controller Parameters

	Tuned	Block
P	0.72843	4.5
I	0.11633	4.53
D	0.82433	1.117
N	6.8095	100

Performance and Robustness

	Tuned	Block
Rise time	1.02 seconds	0.382 seconds
Settling time	11.7 seconds	10.7 seconds
Overshoot	10.2 %	66.2 %
Peak	1.1	1.66
Gain margin	24 dB @ 9.76 rad/s	-10.2 dB @ 1.36 rad/s
Phase margin	69 deg @ 1.25 rad/s	17.7 deg @ 2.49 rad/s
Closed-loop stability	Stable	Stable

PID Tuning using MATLAB PID Tuner(Example1):



Controller Parameters

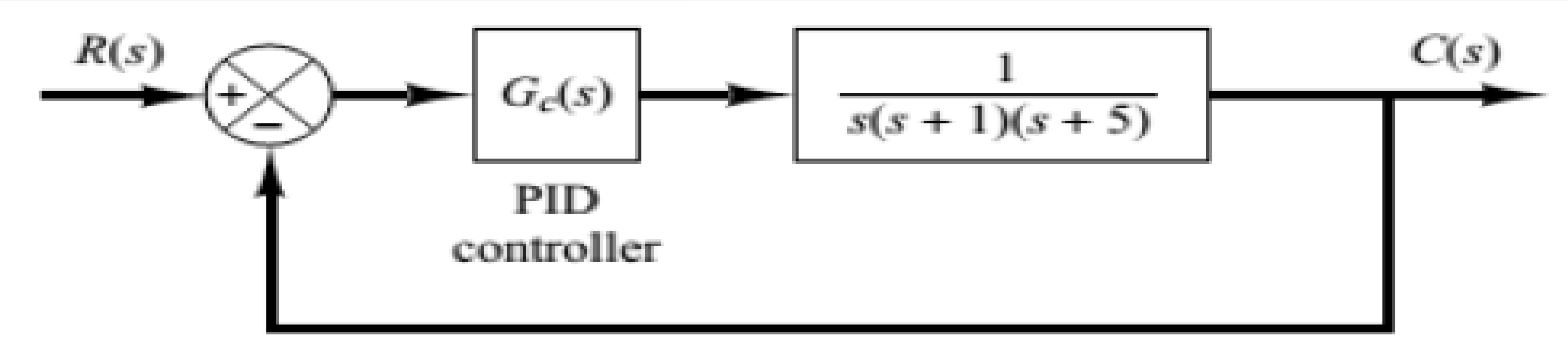
	Tuned	Block
P	0.72843	4.5
I	0.11633	4.53
D	0.82433	1.117
N	6.8095	100

Performance and Robustness

	Tuned	Block
Rise time	1.02 seconds	0.382 seconds
Settling time	11.7 seconds	10.7 seconds
Overshoot	10.2 %	66.2 %
Peak	1.1	1.66
Gain margin	24 dB @ 9.76 rad/s	-10.2 dB @ 1.36 rad/s
Phase margin	69 deg @ 1.25 rad/s	17.7 deg @ 2.49 rad/s
Closed-loop stability	Stable	Stable

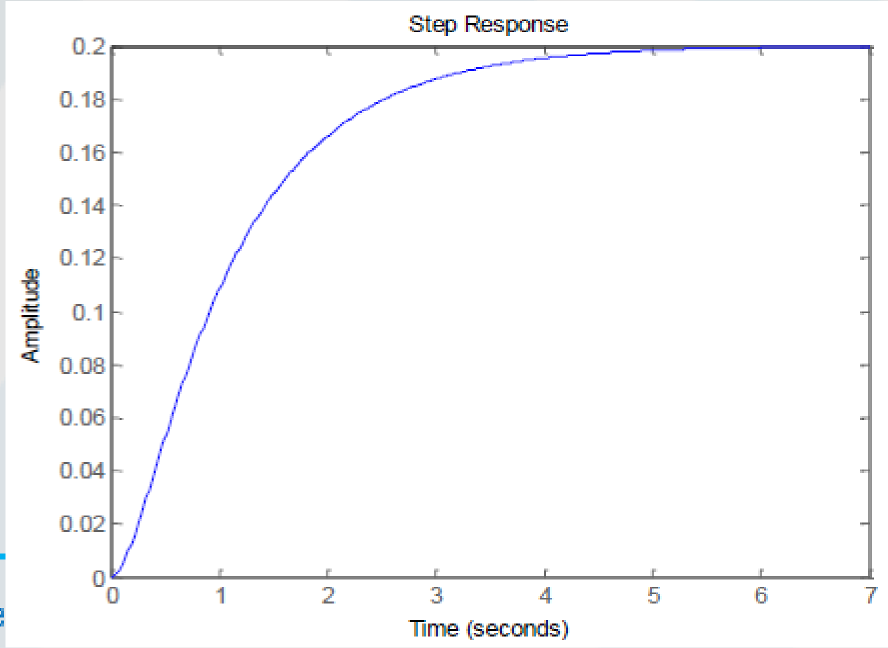
Controller Parameters: P = 0.7284, I = 0.1163, D = 0.8243, N = 6.809

PID Tuning (Ziegler-Nichols method) Example2:

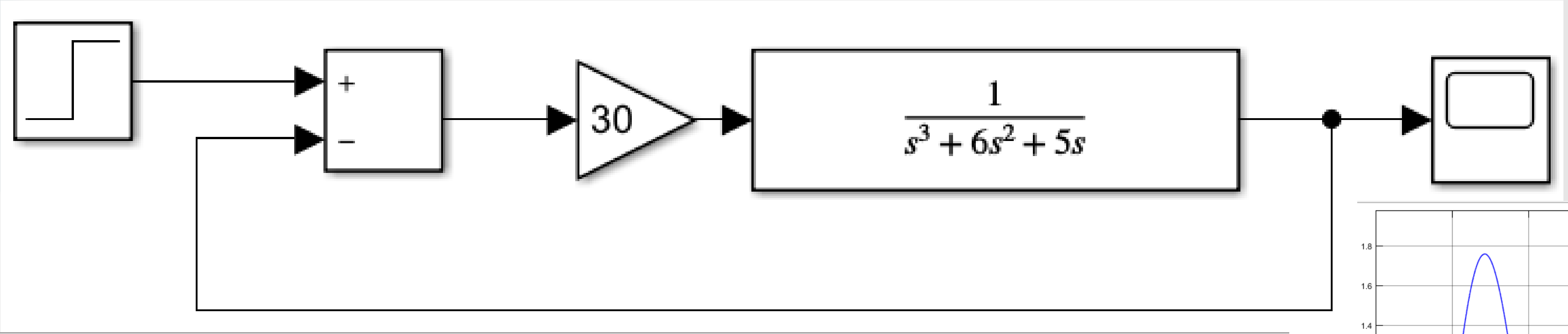


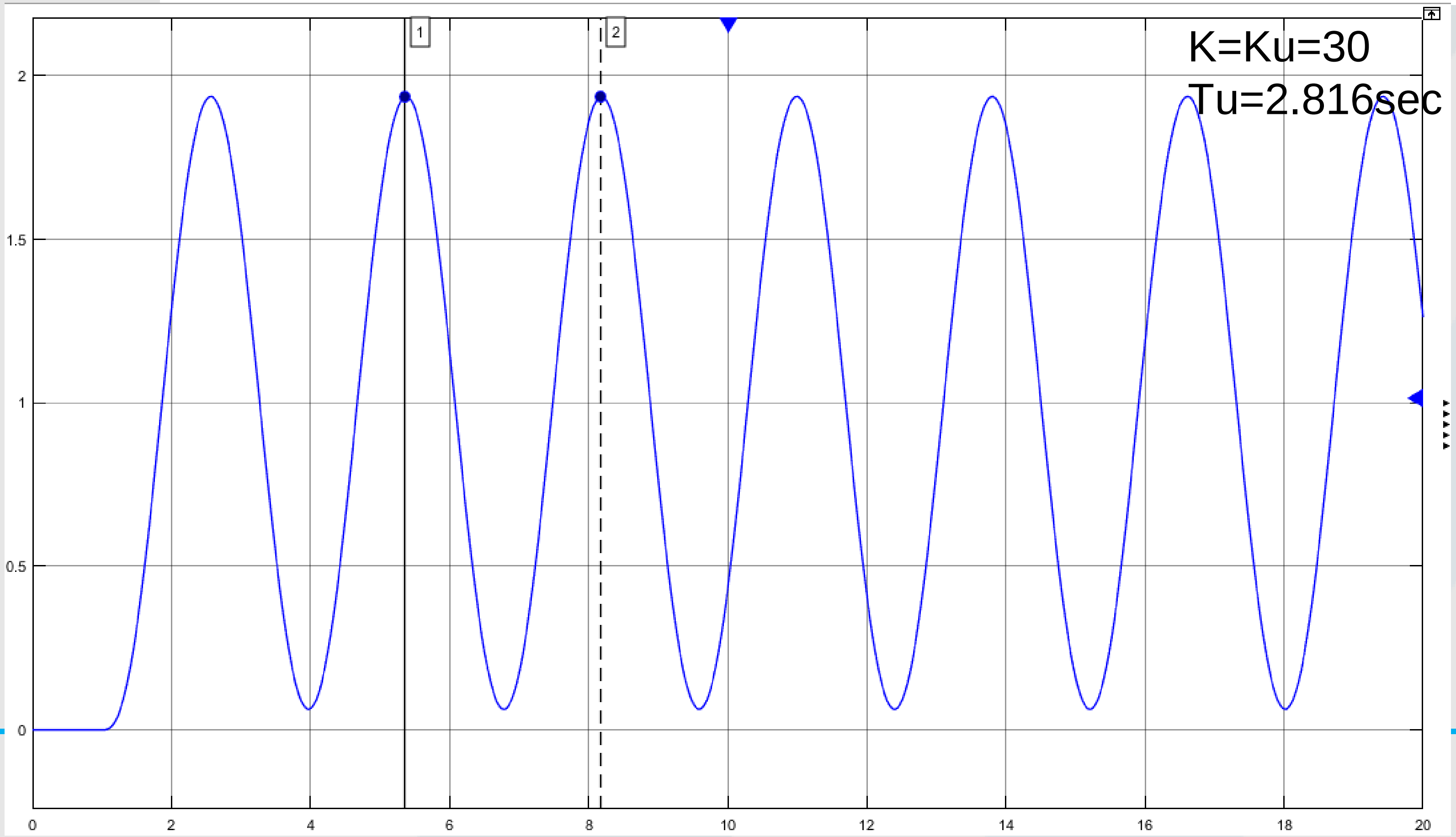
$$\frac{1}{s(s+1)(s+5)} = \frac{1}{s^3 + 6s^2 + 5s}$$

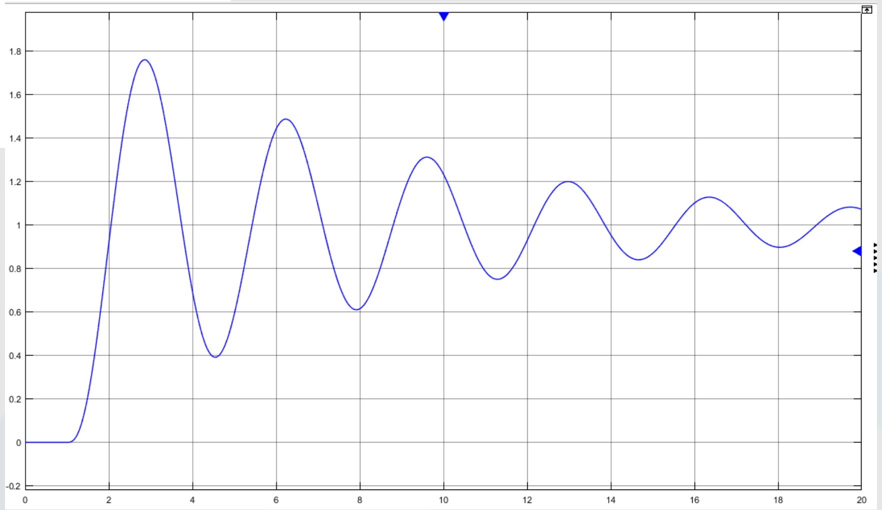
```
num=1;
den= [1 6 5];
plant= tf(num,den);
step(plant)
```

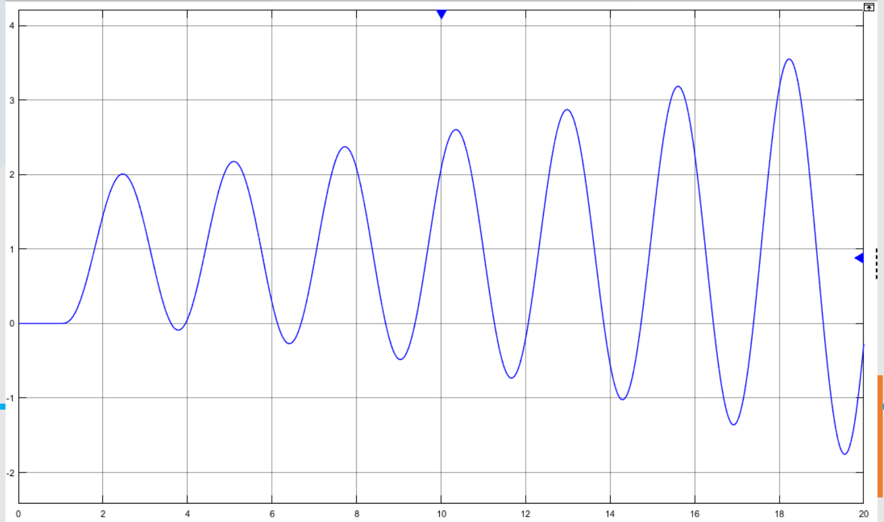


PID Tuning (Ziegler-Nichols method) Example2:









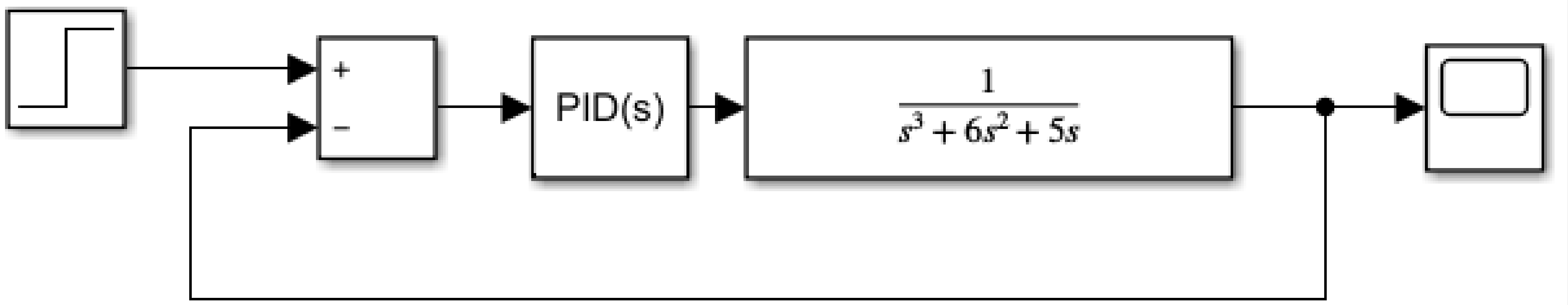
PID Tuning (Ziegler-Nichols method) Example2:

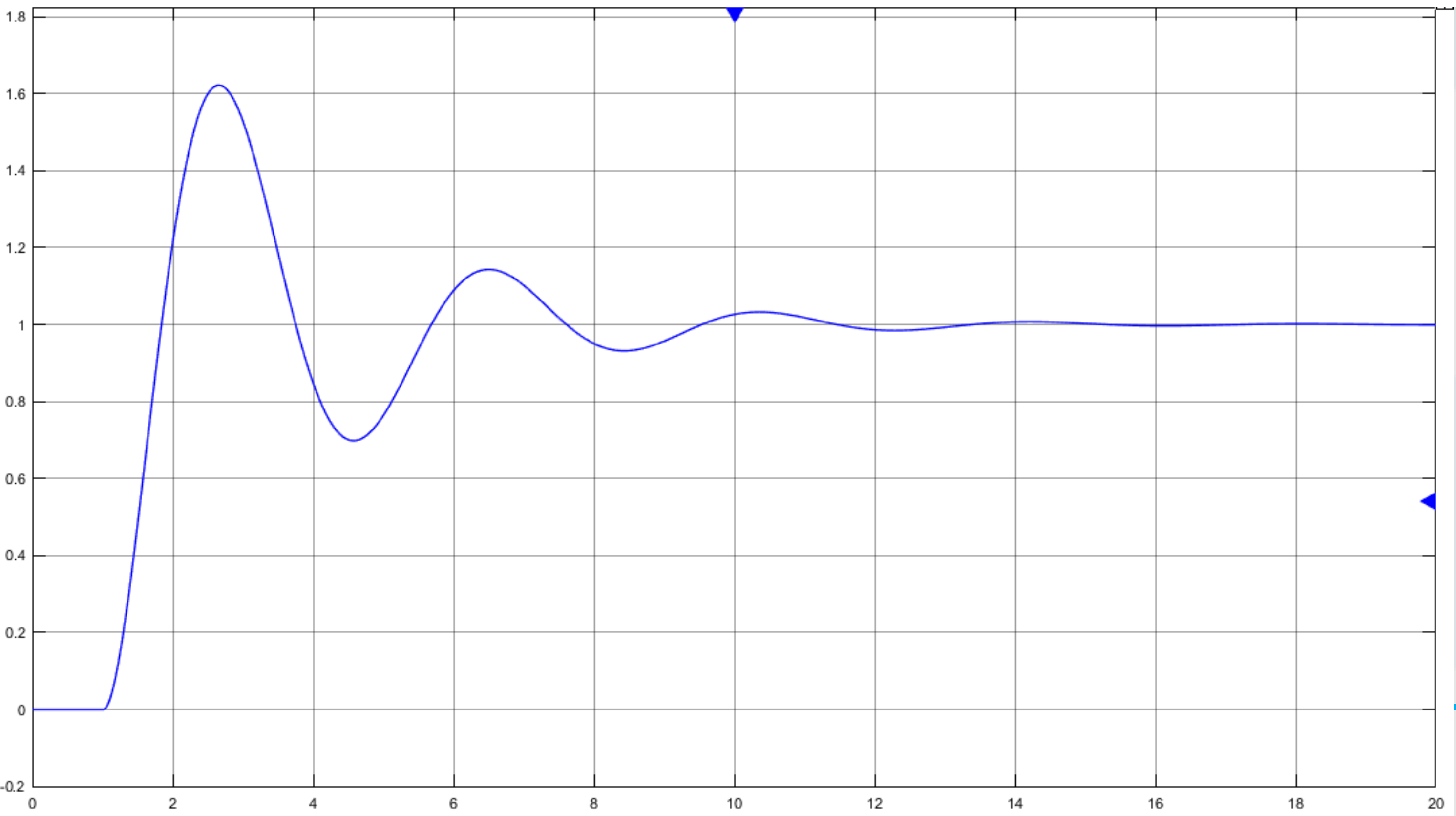
Manual Method:

Kp = 0.60 x Kc➔Kp = 0.6 x 30➔Kp = 18
Ki=1.2xKc/Tu=1.2x30/2.816➔Ki=12.78
Kd=3x Kc x Tu/40=3x30x2.816/40➔Kd=6.336

Control type	Kp	Ki	Kd
P	0.50Ku	—	—
PI	0.45Ku	0.54Ku/Tu	—
PID	0.60Ku	1.2Ku/Tu	3KuTu/40

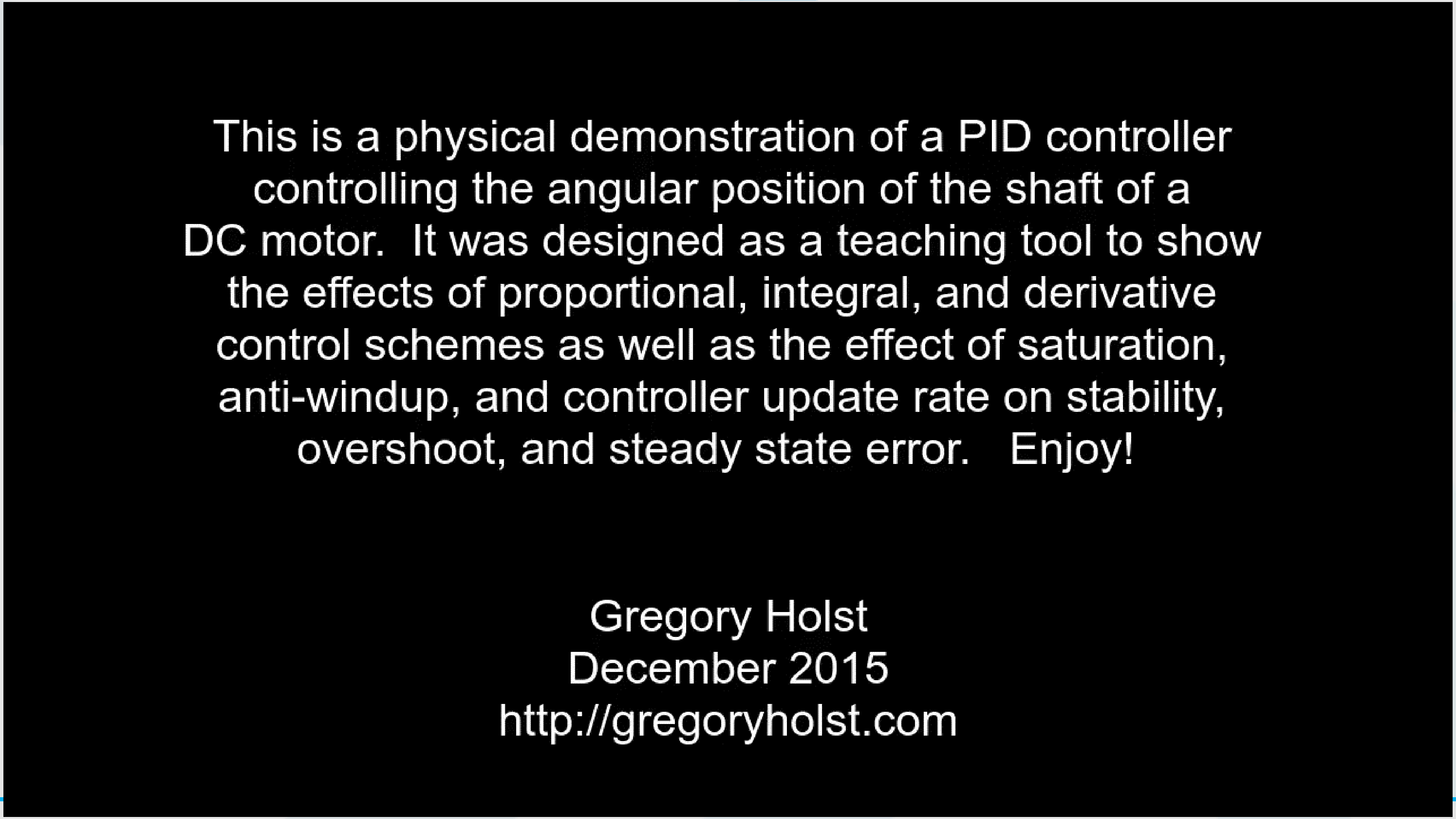
PID Tuning (Ziegler-Nichols method) Example2:





Kp = 0.60 x Kc➔Kp = 0.6 x 30➔Kp = 18
Ki=1.2xKc/Tu=1.2x30/2.816➔Ki=12.78
Kd=3x Kc x Tu/40=3x30x2.816/40➔Kd=6.336

Hardware Demo of a Digital PID Controller:



https://www.youtube.com/watch?v=fusr9eTceEo