

Lecture 16

PID controller

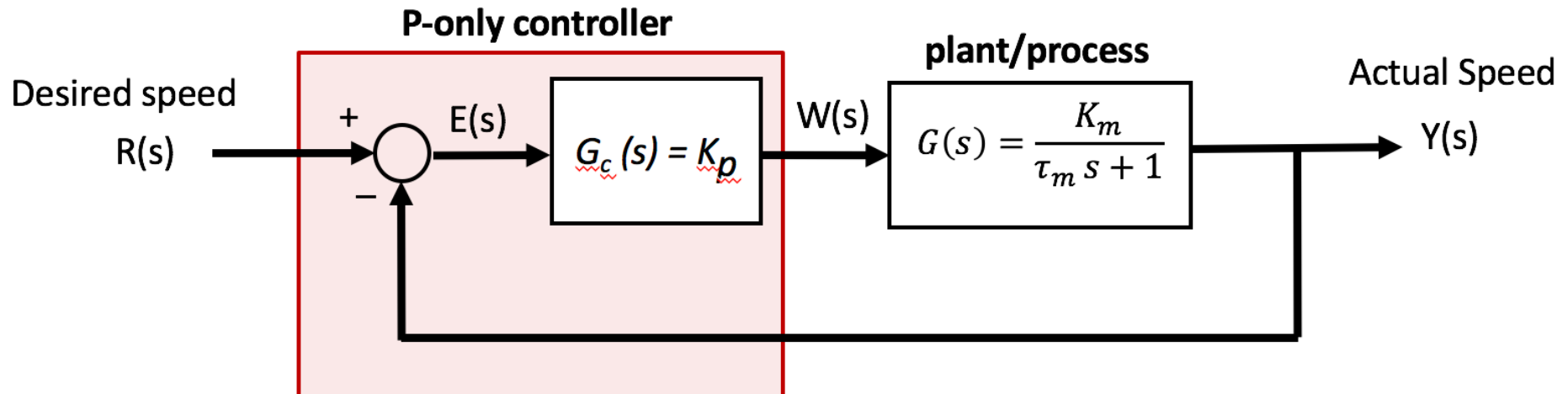
Prof Peter YK Cheung

Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/
E-mail: p.cheung@imperial.ac.uk

Limitations of Proportional-only (P) control

- ◆ In **proportional-only** control, the controller output is given by: $w(t) = K_p e(t)$
- ◆ Using P-only control is simple, but often insufficient because:
 1. If K_p is small, **error** $e(t)$ can be large (i.e. there is an offset error between set-point and controlled output variable, in this case, speed of motor).
 2. If K_p is large, the system may oscillate (i.e. become unstable).
 3. Even if the system is stable, it may take a long time to settle to its final output value or exhibition large overshoots.
 4. It may not have sufficient tolerance to perturbations or disturbances.

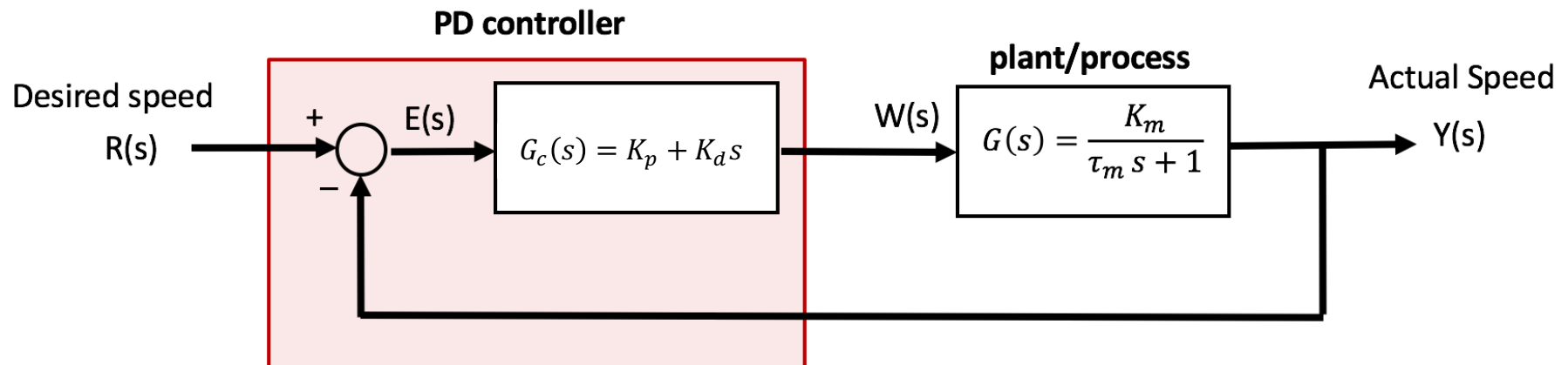


Proportional - Derivative (PD) Control

- ◆ We can add another term to include the rate of change of the error $\dot{e}(t)$. This is known as a **proportional-derivative** (PD) controller: $w(t) = K_p e(t) + K_d \dot{e}(t)$
- ◆ In computers, the **derivative term** $\dot{e}(t)$ is usually calculated by taking the **difference** between current error value $e[n]$ and the previous error value $e[n-1]$:

$$\text{differential term at time } n = K_d (e[n] - e[n - 1]) / \Delta t$$

- ◆ The main advantages of the PD controllers are:
 1. It can reduce the overshoot of a proportional-only controller response because PD controller takes into account the rate of change in error.
 2. It can also improve the system's tolerance to external disturbances.



Proportional - Integral (PI) Control

- ◆ Alternatively, we can add an **integral term** to the controller. This is known as a PI controller:

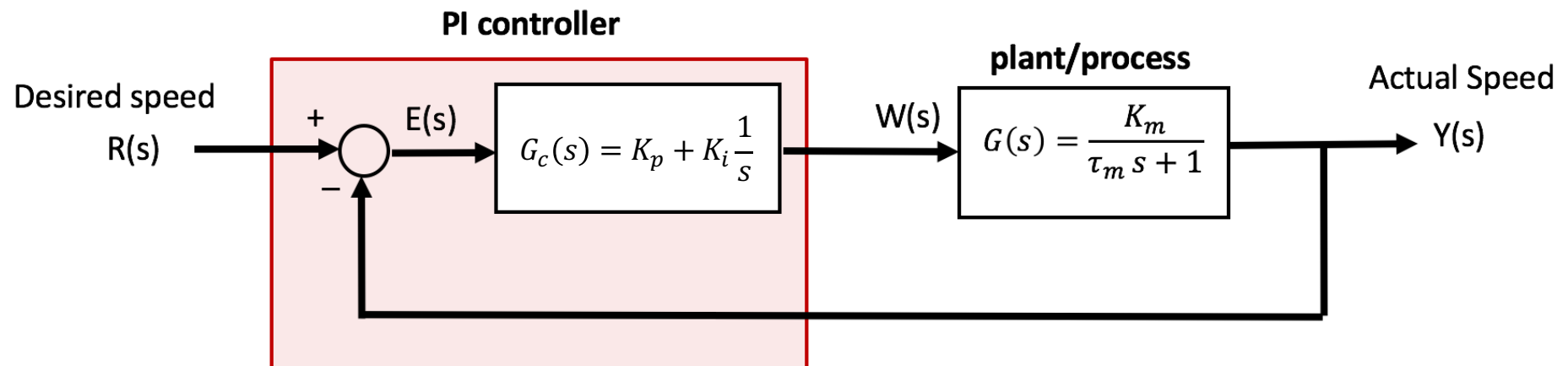
$$u(t) = K_p e(t) + K_i \int e(\tau) d\tau$$

- ◆ The integral term is implemented on a digital microprocessor as **summation** (Δt is the sampling interval):

$$\text{integral term at time } n = K_i \Delta t \sum_{k=0}^n e[n - k]$$

- ◆ The main advantages of the PI controller are:

1. It eliminates **steady-state error**.
2. It can help with **stability of the system**, especially if K_p is large.



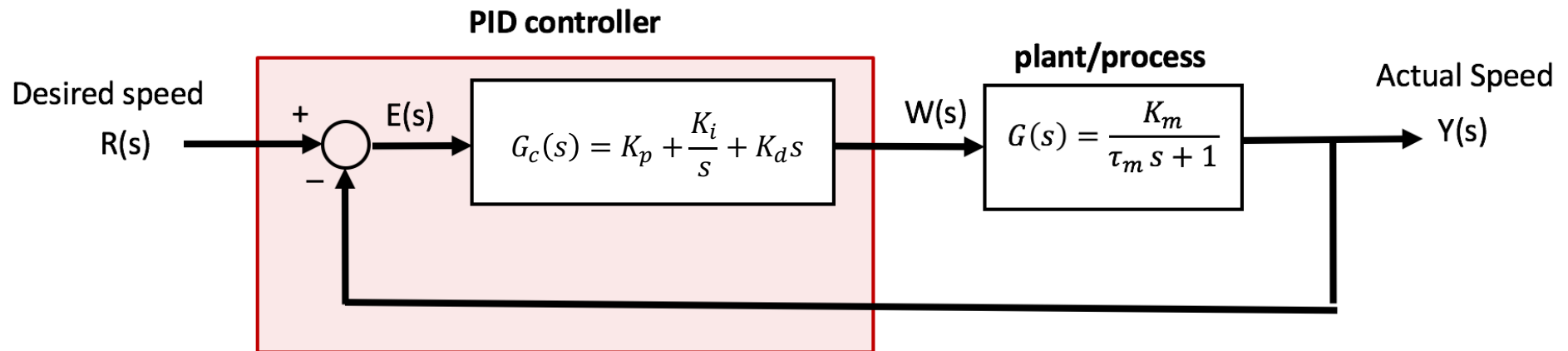
PID Control

- ◆ Finally, we can combine all three terms to form a PID controller:

$$u(t) = K_p e(t) + K_i \int e(\tau) d\tau + K_d \dot{e}(t)$$

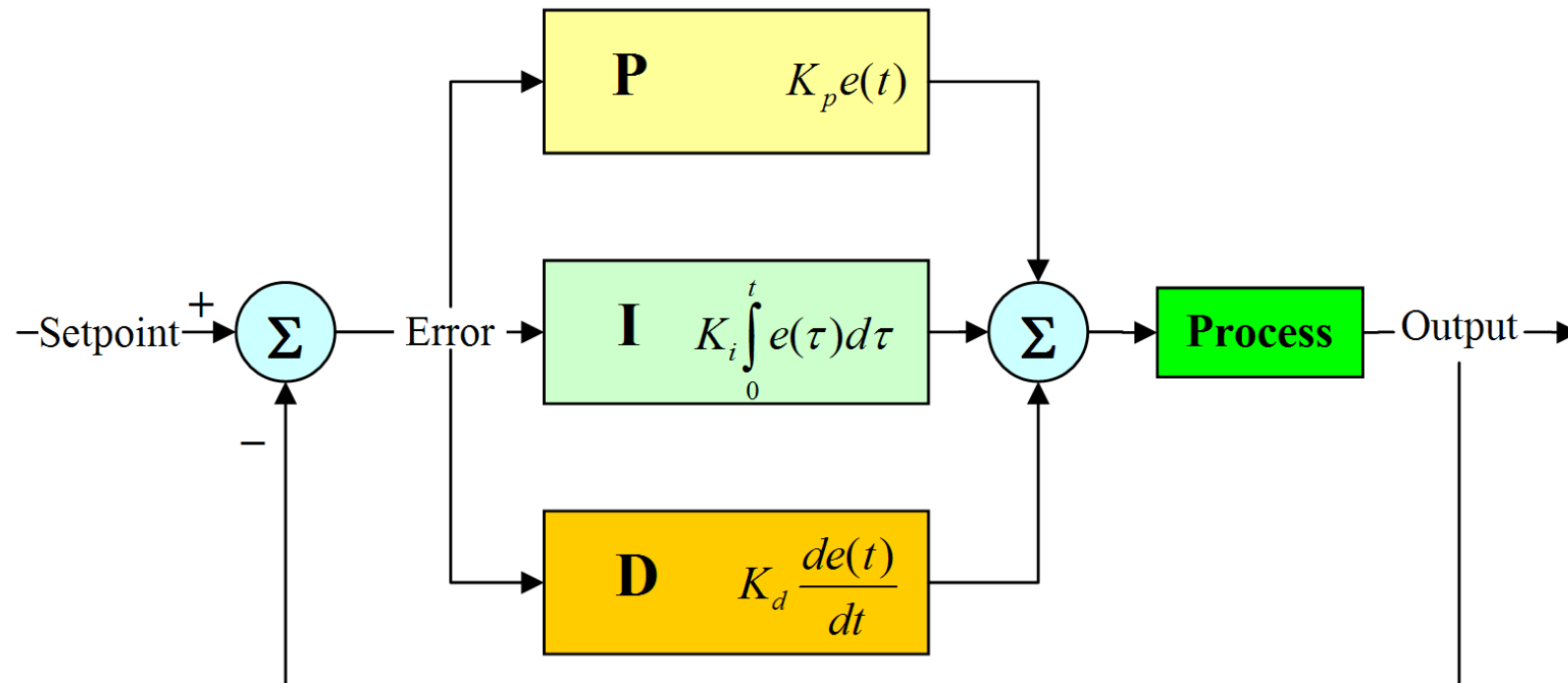
- ◆ This has the advantages of **ALL** three types of feedback control (P, I and D):

1. **Removal** of steady-state **error** due to I.
2. **Reduce** the amount of **overshoots** (due to be I and D).
3. **Improve** the **transient** response to make it faster (due to both I and D).
4. **Improve stability** of the system.
5. Better perturbation tolerance.



PID Controller block diagram

- ◆ Here is a schematic of an implementation of a typical PID controller.
- ◆ In practice, we often use PI (e.g. driving a motor), or PD (e.g. balancing two-wheel vehicle), and NOT all three terms.



Use PID controller to drive a car



Tuning of a PID Controller

- ◆ Choosing the correct values for K_p , K_d and K_i is known as **tuning** the controller.
- ◆ **Impact of various gains** on step response of a system:

PID Gain	Percent Overshoot	Settling Time	Steady-State Error
Increasing K_p	Increases	Minimal impact	Decreases
Increasing K_I	Increases	Increases	Zero steady-state error
Increasing K_D	Decreases	Decreases	No impact

- ◆ We will now consider two approaches to tuning the PID controller:
 1. Ziegler-Nichols method
 2. Trial-and-error manual tuning

Ziegler-Nichols method of tuning PID controller

1. Set K_d and K_i to **zero**.
2. Adjust K_p from 0 until the system starts to **oscillate** at certain frequency.
3. **Measure** the value $K_u = K_p$, and the oscillation period as T_u .
4. **Set** the various **gain factors** according to the follow formula and table:

$$u(t) = K_p(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \dot{e}(t))$$

Control Type	K_p	T_i	T_d
<i>P</i>	$0.5K_u$	-	-
<i>PI</i>	$0.45K_u$	$T_u/1.2$	-
<i>PD</i>	$0.8K_u$	-	$T_u/8$
<i>classic PID</i>	$0.6K_u$	$T_u/2$	$T_u/8$

Manual method of tuning PID controller

1. Set K_p , K_d and K_i to zero.
 2. Start with a small K_p , double it each time until the system starts to oscillate.
 3. Half the value of K_p .
 4. Start with a small K_d , double it each time until the system starts to oscillate.
 5. Half the value of K_d .
 6. Start with a small K_i , double it each time until the system starts to oscillate.
 7. Half the value of K_i .
-
- ◆ If you are only using PD or PI controller, skip the irrelevant steps.
 - ◆ Fine tune the various gain until you get the response you want.

Segway balancing with PID controller

- ◆ For the team project, we need to balance the Segway using feedback control.
- ◆ Instead of using motor speed as the control variable, we should use the **pitch angle** as the **control variable**.
- ◆ Available for us to use is the **pitch angle** (after passing through a **complementary filter**) p and the **rate of change of pitch angle** (from gyroscope alone) \dot{p} .
- ◆ Since we have \dot{p} available, the best controller to use is a PD controller, where the control variable is the pitch angle p .
- ◆ For **normal balancing** action, the **set-point $r(t)$ is 0**, i.e. upright position.
- ◆ The controller output value $w(t)$ is derived with all P, I and D terms of the pitch angle:

$$w(t) = K_p e(t) + K_d \dot{e}(t) + K_i \int e(\tau) d\tau$$

- ◆ $w(t)$ is used to drive the motors forward or backward in order to keep pitch angle $p = 0$, by producing PWM values to drive the two motors.
- ◆ To **move** the Segway forward or backward, **change the set-point $r(t)$** to some other values (a small positive or negative angle).
- ◆ To **turn** right or left, you need also to **adjust the ratio** of PWM duty cycle between the two motors.