Imperial College London

Lecture 16

PID controller

Prof Peter YK Cheung

Dyson School of Design Engineering

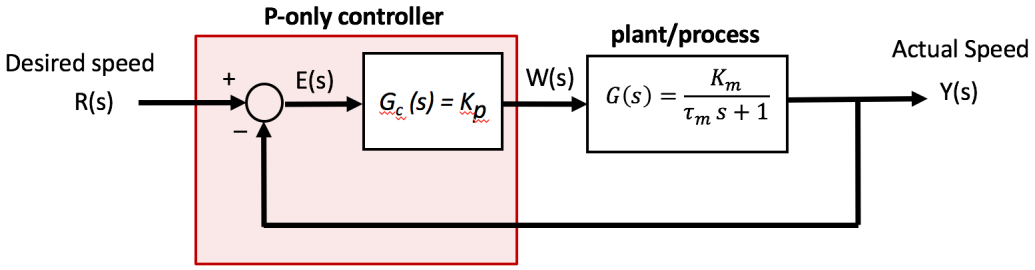
URL: www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/
E-mail: p.cheung@imperial.ac.uk

In this lecture, we will examine a very popular feedback controller known as the **proportional-integral-derivative (PID) control** method. This type of controller is widely used in industry, does not require accurate model of the plant or process being controlled and can be understood by most engineers without being a control expert.

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 Kp 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 Kp 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.



In the last lecture, we studied in some details, how proportional feedback control works. The controller for such a feedback scheme uses a proportional gain element *Kp* to amplify the error signal e(t) and produces a drive signal w(t) as input to the plant. This type of simple control system is found in many situations such as operational amplifiers you used in the first year, or a simple temperature controller you may find at home.

However, proportional-only (P) controller has a number of limitations. In order to produce the output y(t) to match that of the set-point (desired value) r(t) with minimal error e(t), the gain Kp needs to be high. However having a large Kp can make the system unstable, especially if the plant (or processor) is 2nd order or higher.

Even if the system is not unstable and eventually settle down to the steady-state value, the system output may exhibition large overshoot transient behaviour, similar to what you saw with the Bulb Box system when it is driven by a step signal. Since such a system is prone to oscillation, it may also take a long timer before it reaches the final steady-state condition.

Finally, using P-only control may not give us the tolerance to perturbation required.

These limitations are summarised in the slide above.

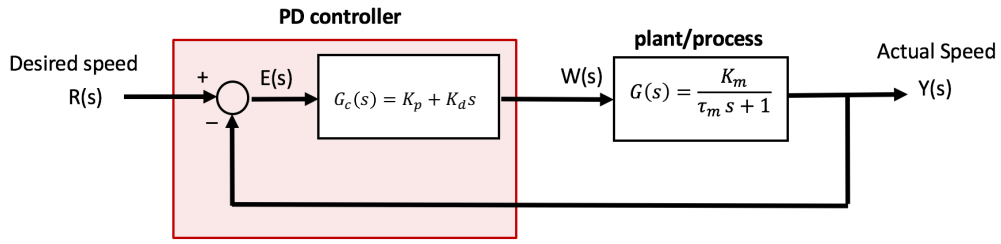
Proportional - Derivative (PD) Control

- 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.



The first improvement we can make is to add a **derivative term** to the controller. The output of the controller is now proportional to the error e(t) (with a gain of Kp) **and** to the derivative of the error, i.e. $\dot{e}(t)$ (with a gain of Kd). This is known as a **proportional-derivative controller** or PD control.
The time domain relationship of the controller is:

$$w(t) = K_p e(t) + K_d \dot{e}(t)$$

Remember that the Laplace Transform of (d/dt) is s. Therefore the transfer function (in s-domain) of the controller is:

$$G_c(s) = K_p + K_d s$$

How can one implement the derivative term on digital hardware (such as a Pyboard or a Raspberry Pi)? The derivative term is implemented as the **difference between two successive samples**:

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

Note that the term $1/\Delta t$ is often "absorbed" into the derivative gain Kd.

What does the derivative term do to the behaviour of the system? The derivative term acts as "**brake**" to the system. As the error is reducing (rate of error is negative), it reduces the drive to the plant (i.e. braking), and therefore prevents overshoot.

The derivative term can also make the system more tolerant to external disturbances.

Proportional - Integral (PI) Control

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

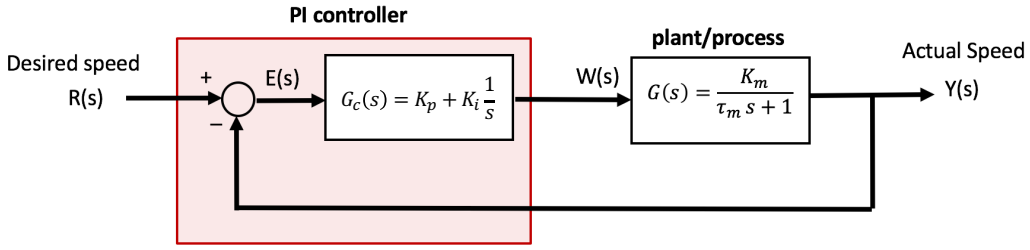
$$u(t) = K_p e(t) + K_i \int e(\tau) d\tau$$

- 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.



Another common improvement to the P-only control is the proportional-integral (PI) control. Here we add another term that is dependent on the integral of the error over time. The time domain description of the controller is:

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

Again in practical electronics and programming, we implement the **integral term as** a **summation**. Integration in calculus is the same as accumulation of area under the curve. Therefore in discrete domain, we simply sum the error from t = 0:

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

As before, Δt is often "absorbed" into the Ki term.

Remember that the Laplace transform of an integral is 1/s. Therefore the s-domain representation of the PI controller (i.e. the transfer function) is:

$$G_c(s) = K_p + K_i \frac{1}{s}$$

PI controller accumulates error to produce a constant output u(t) that is sufficient to drive the plant to give the desired output value y(t). When that happens, e(t) = 0, and yet the output y(t) = r(t) is maintained. It can be seen that this cannot be achieved with P-only controller. For example, in order to drive a DC motor at a certain speed, we need the input PWM duty cycle to be w(t) = *pwm_value*. With proportional gain we need an error e(t) = *pwm_value*/Kp. Unless Kp is infinite, e(t) is NOT 0.

However, with PI controller, the integral term will accumulate to *pwm_value* and now e(t) = 0 and w(t) will stay constant.

A PI controller will work with the P term gain Kp = 0 (i.e. I-only control), but its response to a step input would be slower. The integral action will take time to reach the desired value. However, PD controller WILL NOT work with D-only!

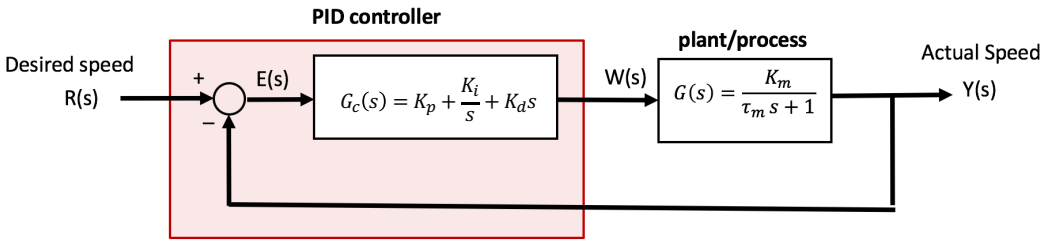
PID Control

- 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.



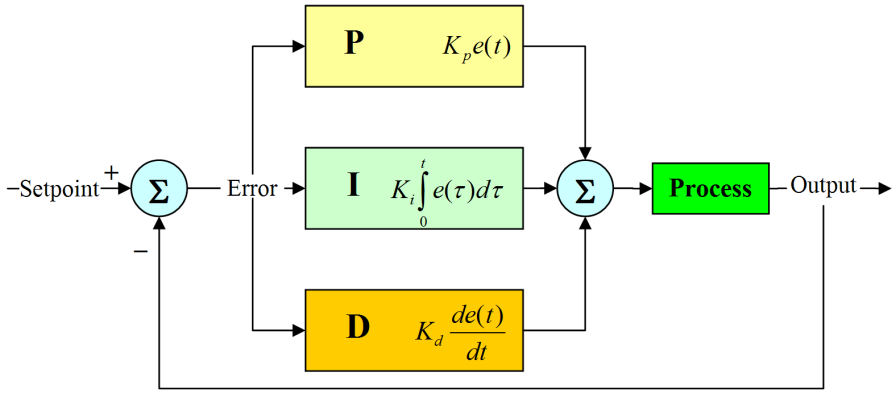
Finally, we can combine all three terms (P, I and D) together to make a PID controller. This has all the benefits of proportional control, integral control and derivative control. Not that PID control is a generalised version of all the other form with the appropriate gains set to some constant or 0.

The time domain formulation of a PID controller is:

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

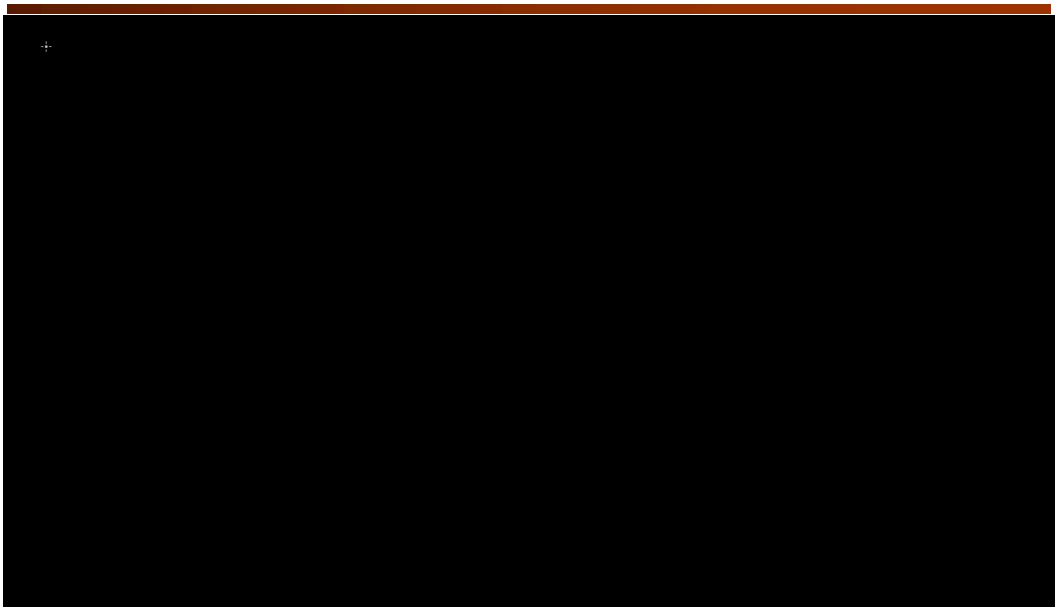
PID Controller block diagram

- 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.



In practical implementations, we can compute the three terms separately and then add them together as shown in the block diagram here.

Use PID controller to drive a car



Here is a nice video explaining the idea of PID control without mathematics. The example used is that of driving a car by stepping on the accelerator pedal. The control variable is speed of the car. The actuation variable w(t) is the rate of change of angle of the pedal.

You can watch this video via this link:

https://www.youtube.com/watch?v=XfAt6hNV8XM

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

PID control structure is therefore very general. Designing a controller for a given plant or process requires us to determine the correct value for Kp, Ki and Kd. This process is known as "**tuning**".

Before we consider how to tune a PID controller, you should know the effect of each of increasing Kp, Ki and Kd on the feedback system behaviour in terms of overshoot, settling time and steady-state error (i.e. e(t) value after the system settle to final condition).

This is shown in the table above.

We will next consider two common approaches to tuning a PID controller.

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
P	$0.5K_u$	-	-
PI	$0.45K_u$	$T_u/1.2$	-
PD	$0.8K_u$	-	$T_u/8$
classic PID	$0.6K_u$	$T_u/2$	$T_u/8$

The first method was introduced by J. G. Ziegler & N. B. Nichols back in 1942. The steps are explained in the slide above.

Note that the Z-N tuning method may not provide you with the optimum performance or best disturbance tolerance. It however does provide a guide on how to set the different gain values for a reasonably good control system.

Also, in this formulation, the gain terms are all expressed in terms of Ku (Kp is factorised to outside the brackets), and the oscillation period Tu (Ti and Td are specified in terms of Tu).

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.

A good alternative to the Z-N tuning method is one described here. This does not involved measuring the oscillation period, or just setting gain values based on measured parameters.

Instead it uses a trial-and-error approach: in turn pushing Kp, then Kd and Ki to the limit until the system becomes unstable, and dial each back to half way or smaller.

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.

Finally, let me relate what you have learned in feedback control to the team project. The task at hand is to balance the Segway so that it can stand upright without falling over. The system (the plant) is inherently unstable (i.e. will NOT balance on its own). Feedback control is necessary for the Segway to self balance.

The variable under control is NOT speed of the Segway, but its pitch angle. To do that, we will use a PID controller.

The pitch angle is measured and is compared to the set-point, which is zero if the Segway is upright. This error is multiplied by Kp and used as part of the PWM value to drive the motors in the direction that corrects the error.

The derivative control is NOT achieved by differentiating the error e(t). Instead we will use the gyroscope reading for y-axis, which measures the rate of change of the pitch angle (i.e. $\dot{p}$). We multiple $\dot{p}$ with Kd to get the derivative term. Finally we integrate the value *p* and multiply this with Ki to get the integral term. Now we add these together to provide the drive to the motors. Remember that we need to limit this value to +/- 100 (duty cycle cannot exceed 100, and +/- indicates direction of motor).
To move the Segway forward or backward, all you need is to change the **set-point** (target pitch angle). This is actually how it works on a real Segway.

Finally **turning** can be achieved by changing the ratio of the PWM values driving the left and the right motor. Having the two wheels at different speed will make the Segway turn.