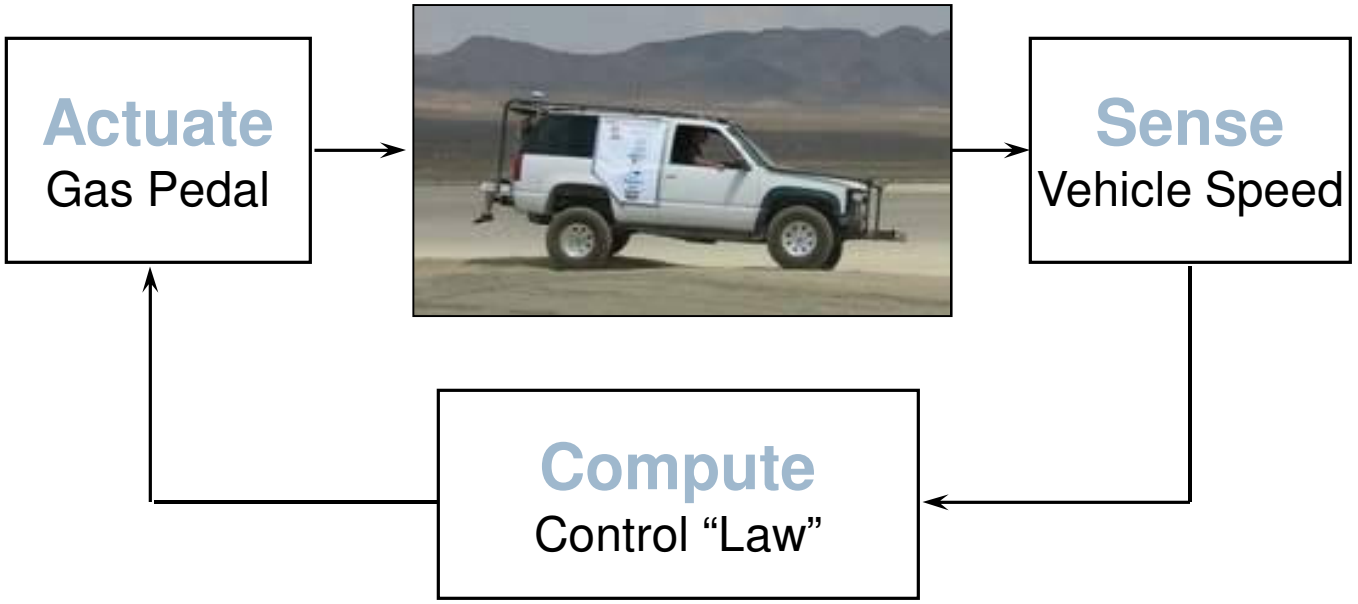
Robotics Group Project - 5CCS2RGP

Lecture 3: Feedback control

Hongbin Liu, King's College London, 2014

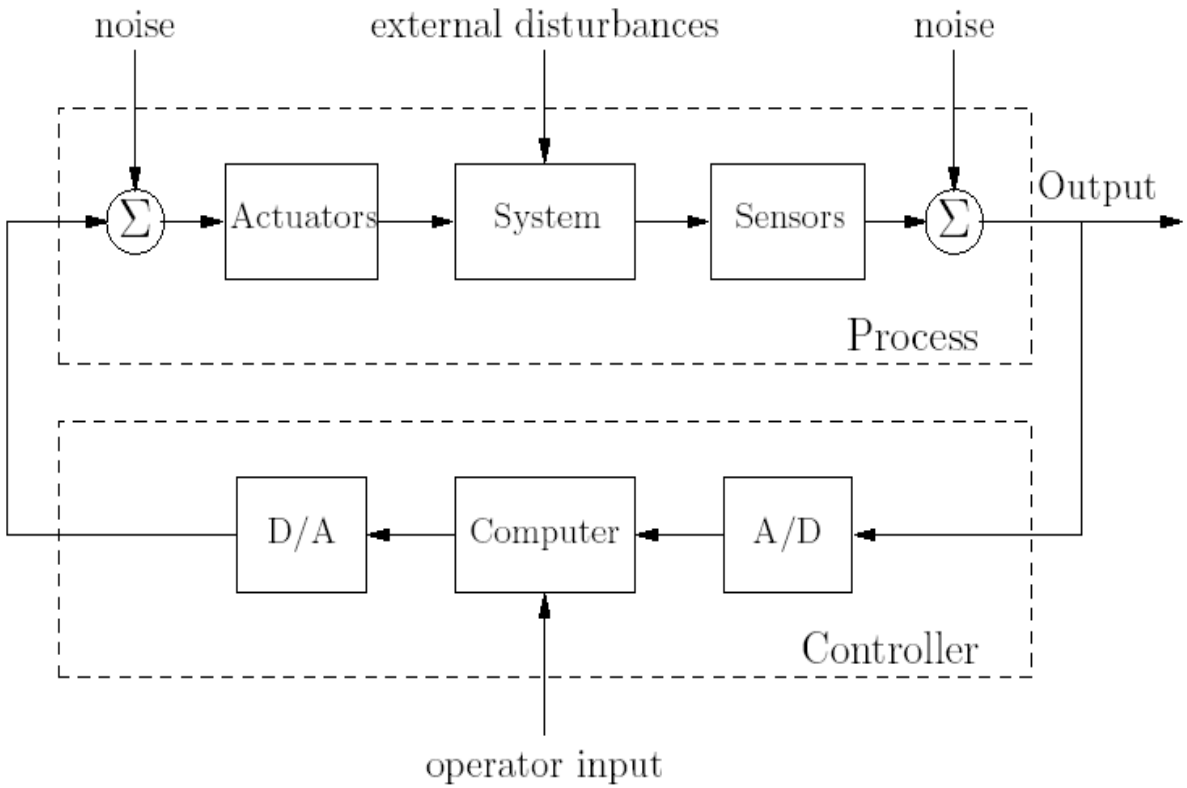
What is Feedback?



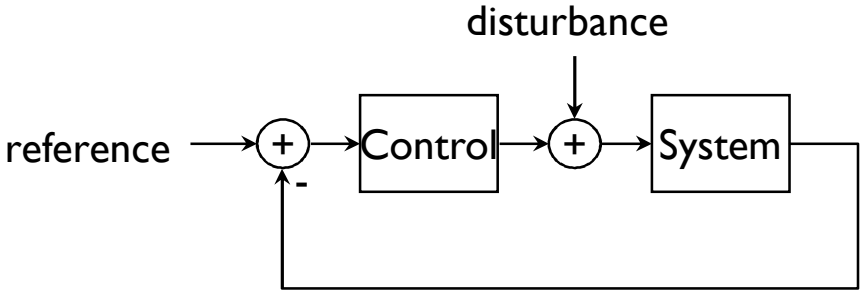
Control = Sensing + Computation + Actuation

In Feedback "Loop"

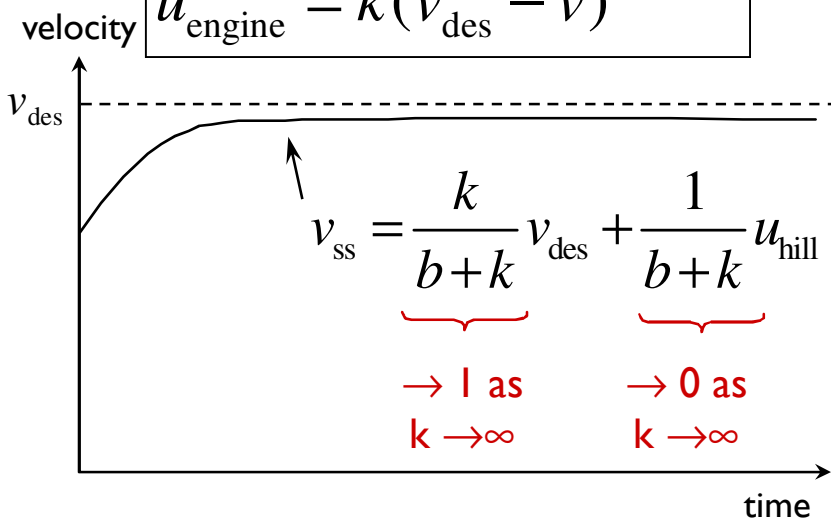
A modern Feedback Control System



Example : Cruise Control



$$m\dot{v} = -bv + u_{\text{engine}} + u_{\text{hill}}$$
$$u_{\text{engine}} = k(v_{\text{des}} - v)$$



- Stability/performance
  - Steady state velocity approaches desired velocity as k $\rightarrow \infty$
  - Smooth response; no overshoot or oscillations
- Disturbance rejection
  - Effect of disturbances (hills) approaches zero as k $\rightarrow \infty$

PID control

PID control has become almost universally used in industrial control. These controllers have proven to be robust and extremely beneficial in the control of many important applications.

PID stands for:

P (*Proportional*)

I (*Integral*)

D (*Derivative*)

Controlling a Simple System

- Consider a simple system: $\dot{x} = Fx + u$
- The setpoint x_{set} is the desired value.
  - The controller responds to error: $e = x - x_{set}$
- The goal is to set u to reach $e = 0$.

The intuition behind control

- Use action *u* to push back toward error *e* = 0
  - error *e* depends on state *x* (via sensors *y*)
- What does pushing back do?
  - Depends on the structure of the system
  - Velocity versus acceleration control
- How much should we push back?
  - What does the magnitude of *u* depend on?

Velocity or acceleration control?

- If error reflects **x**, does **u** affect **x**′ or **x**′′ ?

- Velocity control: $\mathbf{u} \rightarrow \mathbf{x}'$ (Lego robot)
- Acceleration control: $\mathbf{u} \rightarrow \mathbf{x}''$ (rocket)

The Bang-Bang Controller

- Push back, against the *direction* of the error
  - with constant action *u*
- Error is $e = x - x_{set}$

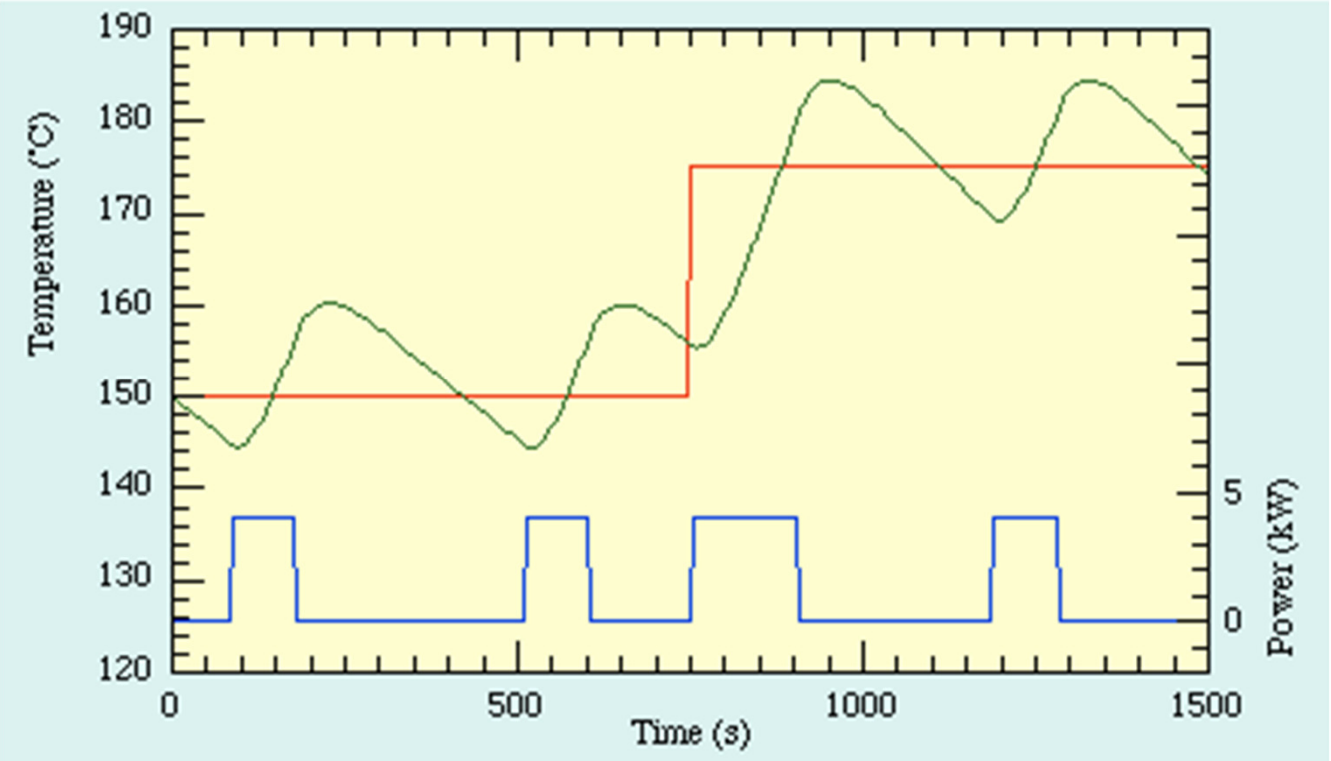
$$e < 0 \Rightarrow u := on \Rightarrow \dot{x} = F(x, on) > 0$$
$$e > 0 \Rightarrow u := off \Rightarrow \dot{x} = F(x, off) < 0$$

- To prevent chatter around e = 0,

$$e < -\varepsilon \Rightarrow u := on$$
$$e > +\varepsilon \Rightarrow u := off$$

Bang-Bang Control in Action

- Good for reaching the setpoint
- Not very good for staying near it



Proportional Control

- Push back, *proportional* to the error. $u = -ke + u_b$
  - set u_b so that $\dot{x} = F(x_{set}, u_b) = 0$ at desired state x
- For a linear system, we get exponential convergence.

$$x(t) = Ce^{-\alpha t} + x_{set}$$

- The controller gain *k* determines how quickly the system responds to error.

Velocity Control

- You want to drive your car at velocity $v_{set.}$
- You issue the motor command $u = pos_{accel}$
- You observe velocity v_{obs}.

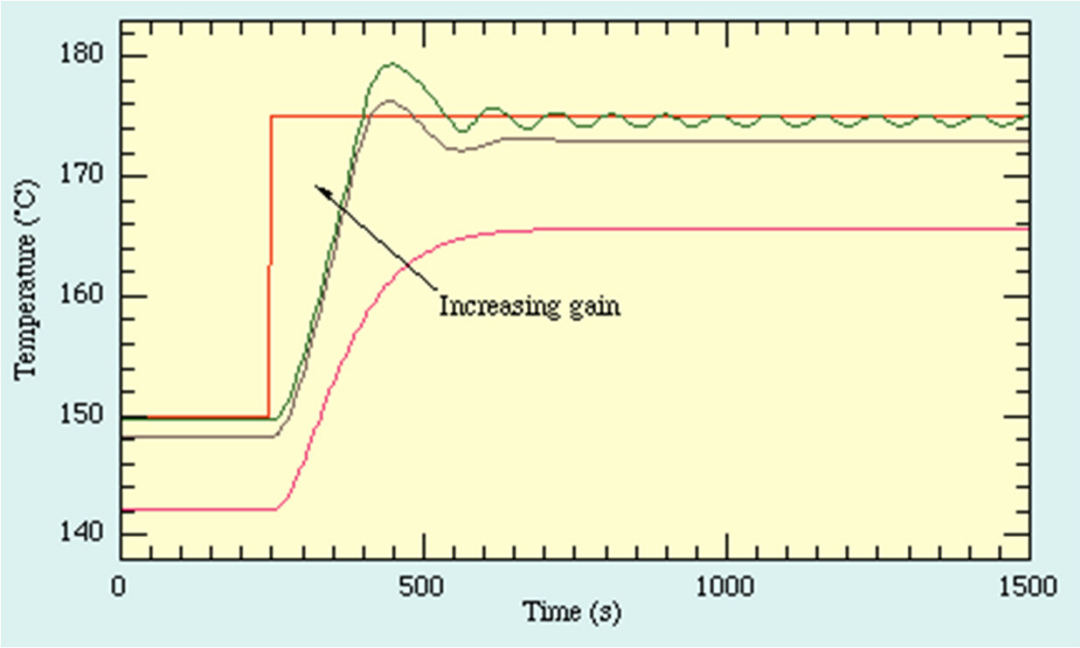
- Define a first-order controller:

$$u = -k(v_{obs} - v_{set}) + u_b$$

  - k is the controller gain.

Proportional Control in Action

- Increasing gain approaches setpoint faster
- Can leads to overshoot, and even instability
- Steady-state offset



Steady-State Offset

- Suppose we have continuing disturbances:

$$\dot{x} = F(x,u) + d$$

- The P-controller cannot stabilize at e = 0.
  - Why not?

Steady-State Offset

- Suppose we have continuing disturbances:

$$\dot{x} = F(x,u) + d$$

- The P-controller cannot stabilize at e = 0.
  - if u_b is defined so $F(x_{set},u_b) = 0$
  - then $F(x_{set},u_b) + d \neq 0$, so the system changes
- Must adapt u_b to different disturbances d.

Adaptive Control

- Sometimes one controller isn't enough.
- We need controllers at different time scales.

$$u = -k_P e + u_b$$

- This can eliminate steady-state offset.

$$\dot{u}_b = -k_I e, \text{where } k_I << k_P$$

  - Why?

Adaptive Control

- Sometimes one controller isn't enough.
- We need controllers at different time scales.

$$u = -k_P e + u_b$$

- This can eliminate steady-state offset.

$$\dot{u}_b = -k_I e, \text{where}: k_I << k_P$$

  - Because the slower controller adapts u_b.

Integral Control

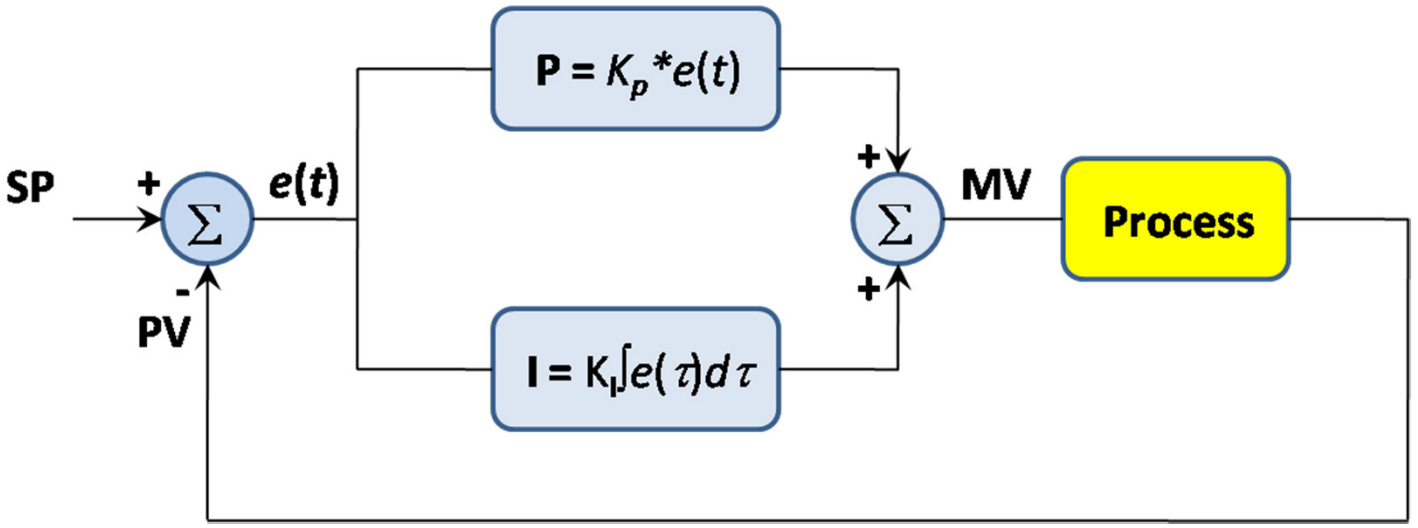
- The adaptive controller $\dot{u}_b = -k_I e$ means

- Therefore $u_b(t) = -k_I \int_0^t e dt + u_b$

- The Proportional-Integral (PI) Controller.

$$u(t) = -k_P e(t) - k_I \int_0^t e dt + u_b$$

Proportional-Integral (PI) Controller



Derivative Control

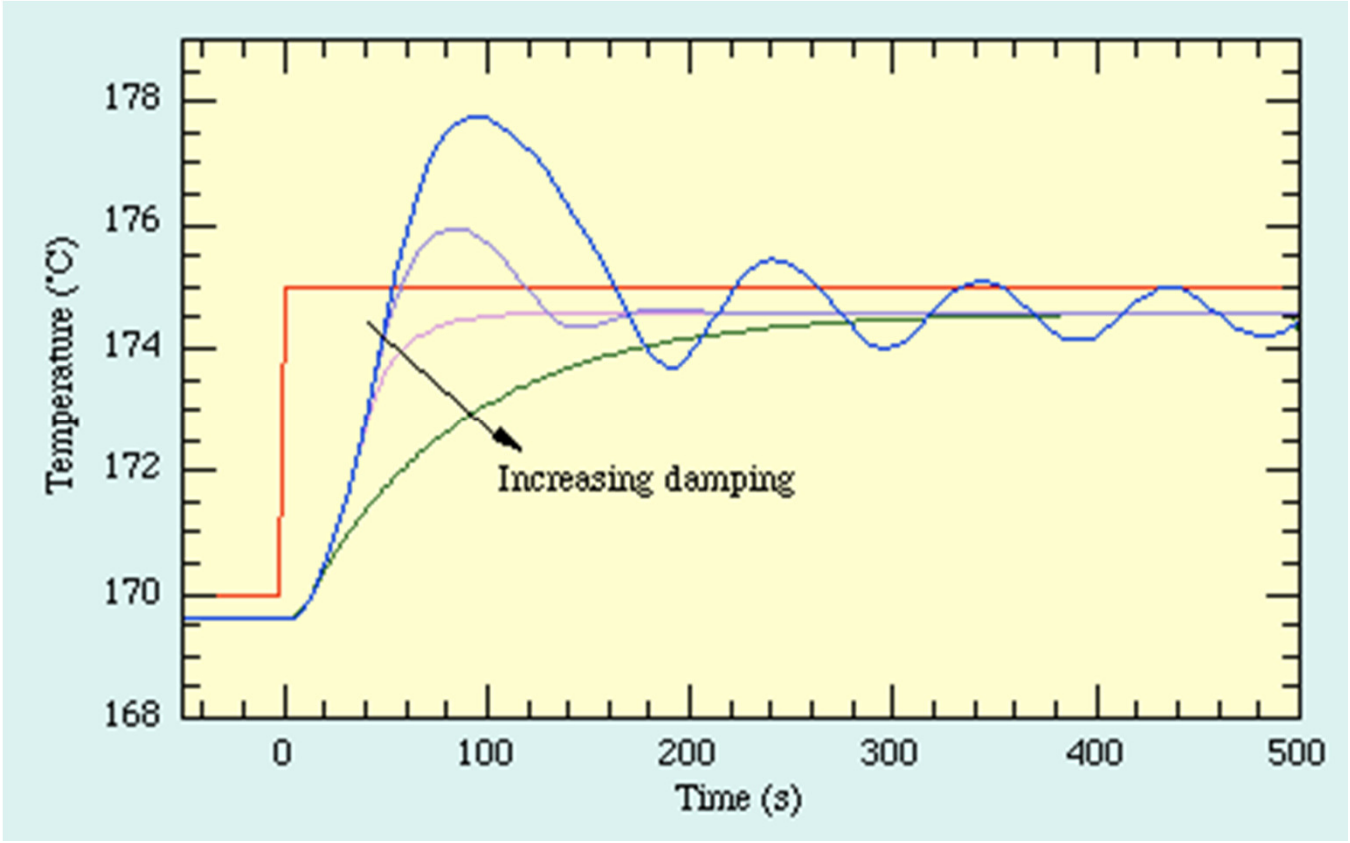
- Damping friction is a force opposing motion, proportional to velocity.
- Try to prevent overshoot by damping controller response.

- Estimating a derivative from measurements is fragile, and amplifies noise.

$$u = -k_P e - k_D \dot{e}$$

Derivative Control in Action

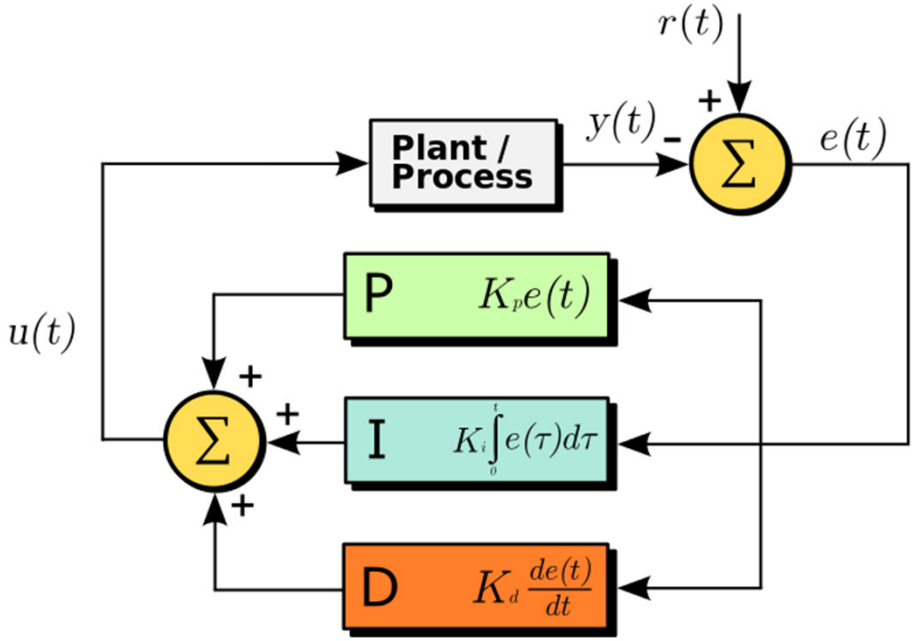
- Damping fights oscillation and overshoot
- But it's vulnerable to noise



The PID Controller

- A weighted combination of Proportional, Integral, and Derivative terms.

$$u(t) = -k_P e(t) - k_I \int_0^t e dt - k_D \dot{e}(t)$$



The PID Controller

where

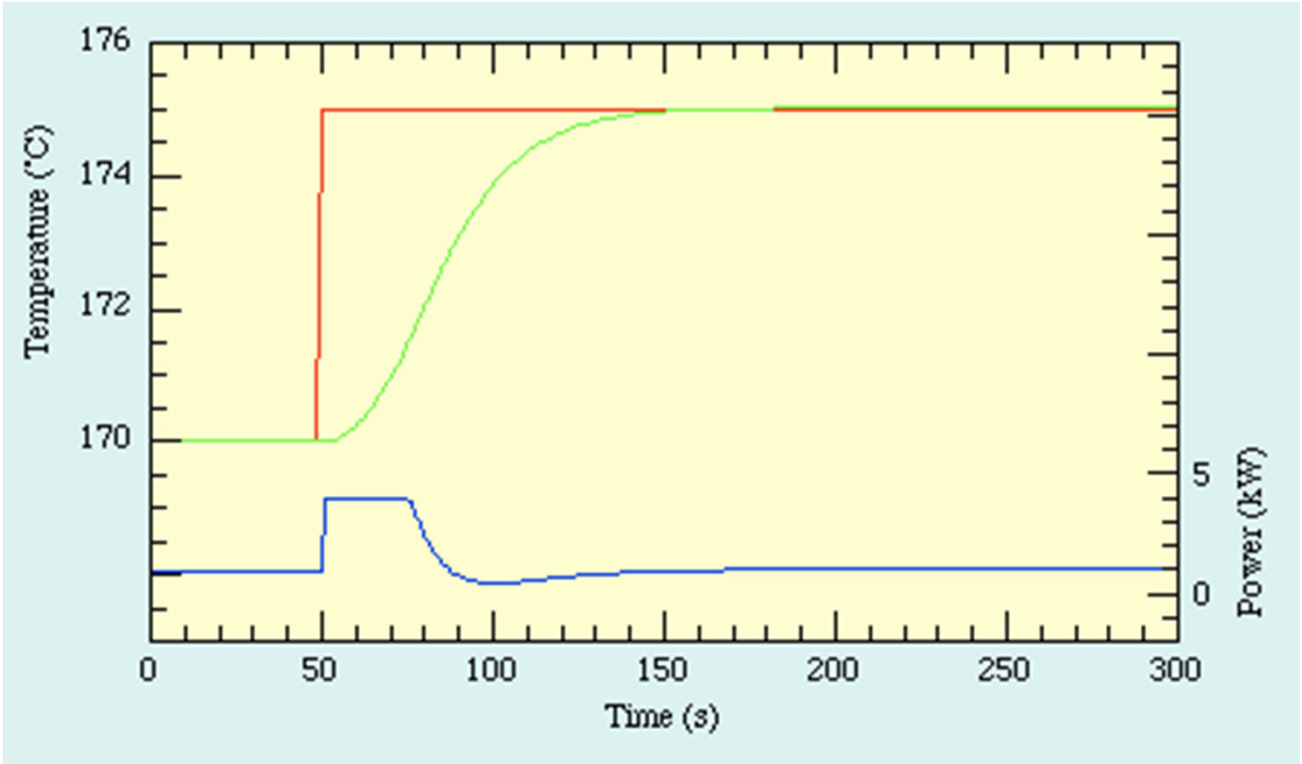
- Kp: Proportional gain, a tuning parameter :
- Ki: Integral gain, a tuning parameter :
- Kd: Derivative gain, a tuning parameter :
- e: Error :
- t: Time or instantaneous time (the present)

Stopping Controller

- Desired stopping point: $x = 0$.
  - Current position: x
  - Distance to obstacle: $d = |x| + \varepsilon$

PID Control in Action

- But, good behavior depends on good tuning!



Historical Note

Early feedback control devices implicitly or explicitly used the ideas of proportional, integral and derivative action in their structures. However, it was probably not until Minorsky's work on ship steering* published in 1922, that rigorous theoretical consideration was given to PID control.

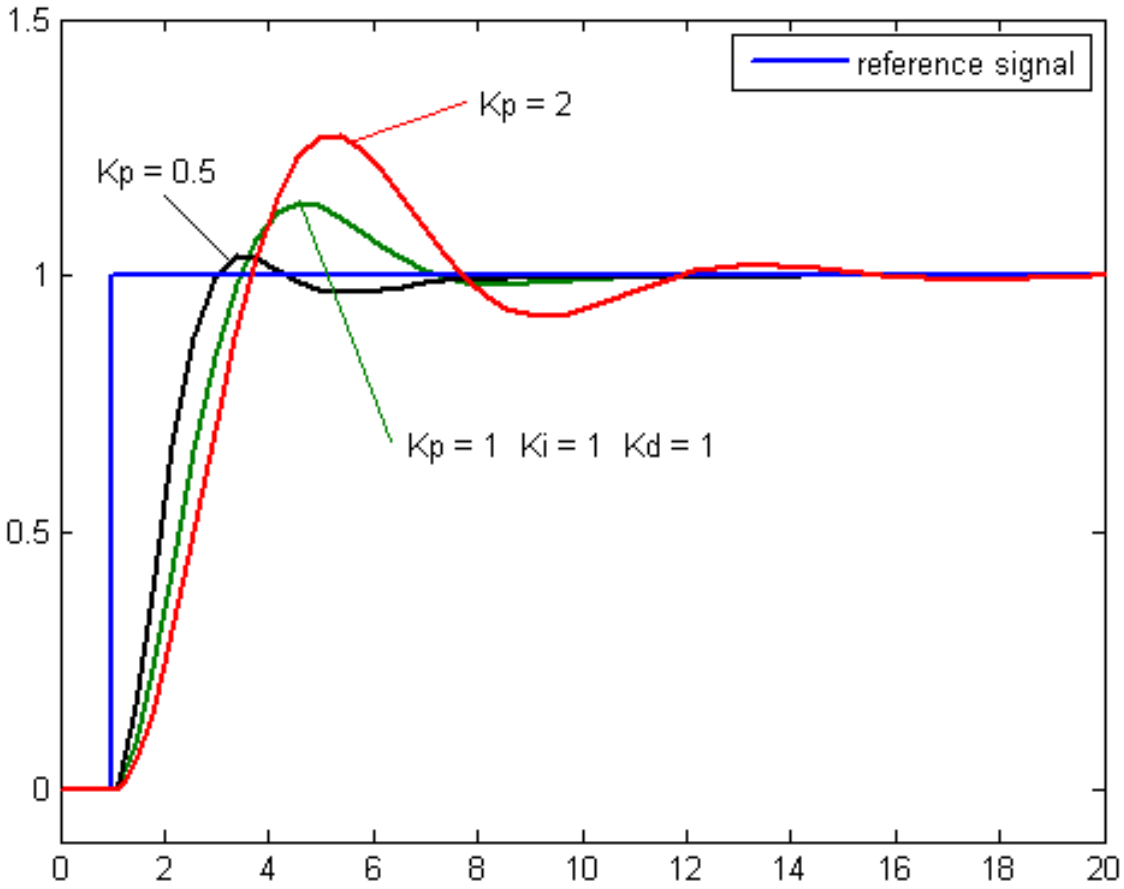
This was the first mathematical treatment of the type of controller that is now used to control almost all industrial processes.

* Minorsky (1922) "Directional stability of automatically steered bodies", *J. Am. Soc. Naval Eng.*, 34, p.284.

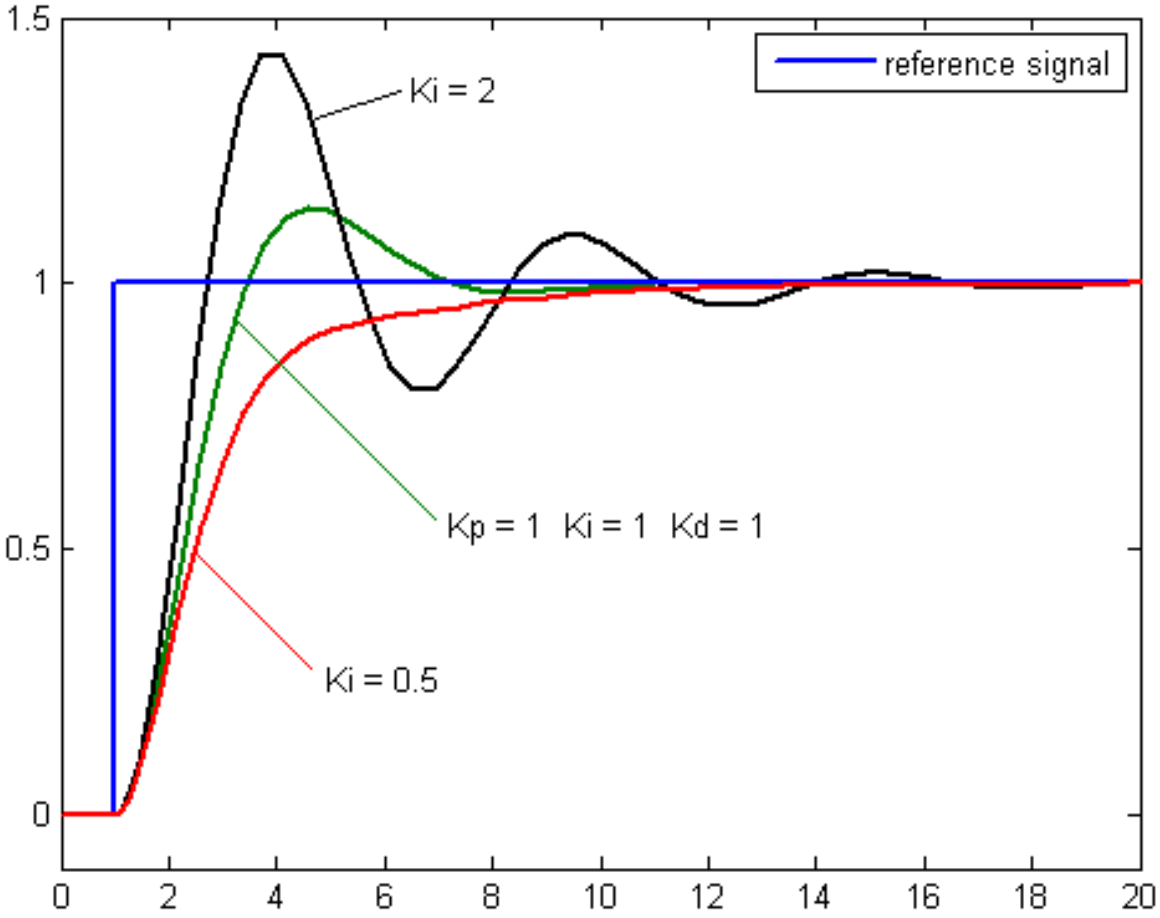
Pseudocode

```
previous_error = 0
integral = 0
start:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
```

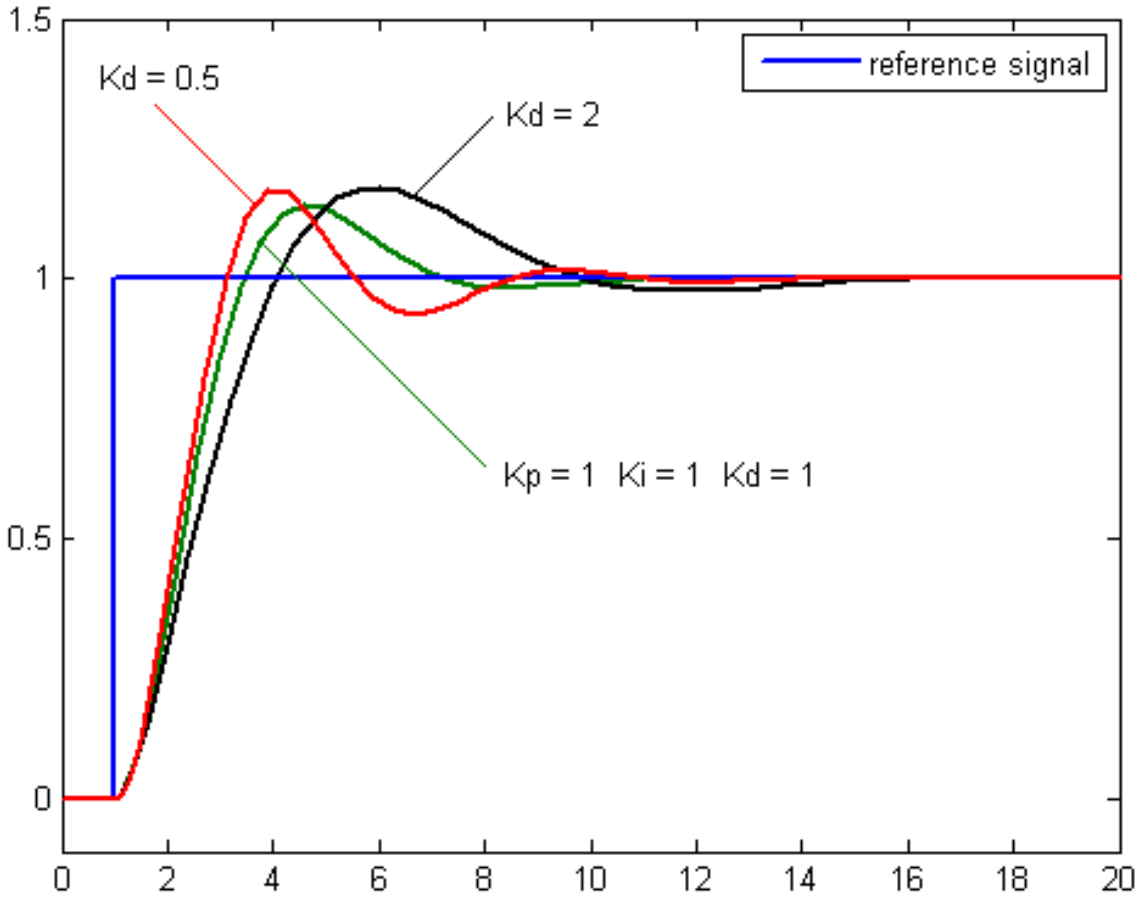
Kp


reference signal
Kp = 2
Kp = 0.5
Kp = 1 Ki = 1 Kd = 1
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20

Ki


reference signal
Ki = 2
Kp = 1 Ki = 1 Kd = 1
Ki = 0.5
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20

Kd


reference signal
Kd = 0.5
Kd = 2
Kp = 1 Ki = 1 Kd = 1
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20

Tuning method

- **Manual tuning**

Effects of increasing a gaineter independently					
gaineter	Rise time	Overshoot	Settling time	Steady-state error	Stability
Kp	Decrease	Increase	Small change	Decrease	Degrade
Ki	Decrease	Increase	Increase	Eliminate	Degrade
Kd	Minor change	Decrease	Decrease	No effect in theory	Improve if Kd small

Tuning method

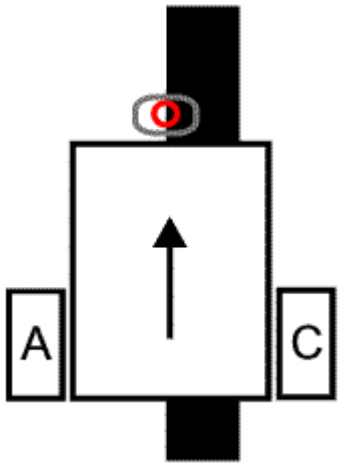
- **Ziegler–Nichols method**

Ziegler–Nichols method giving K' values (loop times considered to be constant and equal to dT)			
Control Type	**Kp**	**Ki'**	**Kd'**
P	$0.50K_c$	0	0
PI	$0.45K_c$	$1.2K_p dT / P_c$	0
PD	$0.80K_c$	0	$K_p P_c / (8dT)$
PID	$0.60K_c$	$2K_p dT / P_c$	$K_p P_c / (8dT)$

heuristic tuning method is formally known as the Ziegler–Nichols method. As in the method above, the and gains Ki, Kd are first set to zero. Kp is increased until it reaches the ultimate gain, Kc, at which the output of the loop starts to oscillate. Kc and the oscillation period Pc are used to set the gains as shown in the table

Lego Mindstorms PID Control Tutorial

- http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html



Tuning method

Loop Tuning - Twiddle

Initial Gain =[0, 0 , 0], dGain=[1, 1, 1], n=0, best_error =test (gains)

```
while sum(dgains)>tol:
    for i=1:number of gains
        gains [i]=gains [i]+dgains[i]
        err=test (gains)
        if err< best_error
            best_error=err
            dgains[i]=dgains[i] *1.1
        else :
            gains[i]=gains[i] - 2.0*dgains[i]
            err=test(gains)
            if err<best_error:
              best_error=err
              dgains[i]=dgains[i]*1.1
            else:
              gains[i]=gains[i]+dgains[i]
              dgains[i]=dgains[i]*0.9
    n=n+1
return gains
```