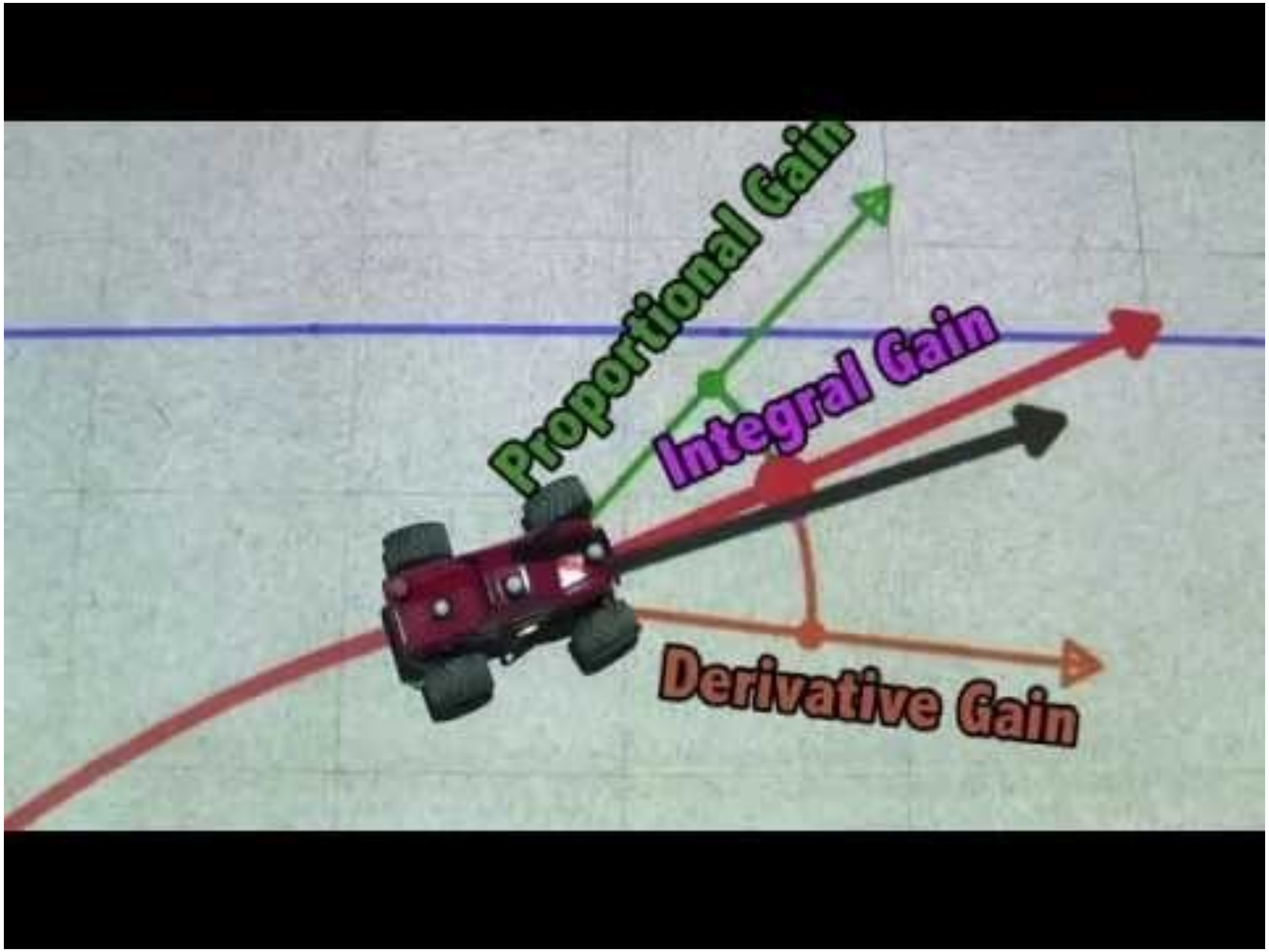
CMPE 185 Autonomous Mobile Robots

Navigation and Control Part 2

Dr. Wencen Wu
Computer Engineering Department
San Jose State University

PID Controller Explained



P Control

- Let us start with a simple controller: P controller
- Proportional Feedback Control (P Control)
  - Uses the error between the desired and measured sate to determine the control signal

- If $x_{desired}$ is the desired state, and x is the actual state, we define the error as

$$e = x_{desired} - x$$

- The control signal u is calculated as $u = K_P e$

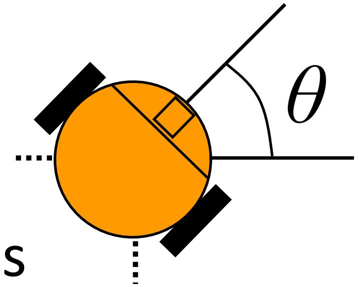
where K_P is called the proportional gain

P Control – A Simple Example

- Consider the orientation control of a mobile robot

$$\dot{\theta} = w$$

$$\theta_{k+1} = \theta_k + w\Delta t$$



- The control signal u is the angular velocity w, and is calculated as

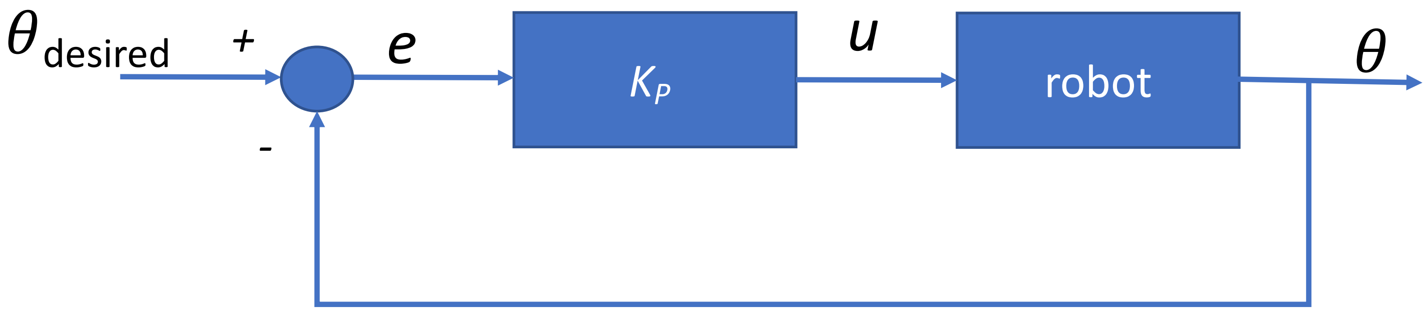
$$u = K_P(\theta_{desired} - \theta)$$

- Note:
  - If $\theta_{desired} = \theta$, the control signal is 0
  - If $\theta_{desired} < \theta$, the control signal is negative, resulting in an decrease in θ
  - If $\theta_{desired} > \theta$, the control signal is positive, resulting in an increase in θ
  - The magnitude of the increase/decrease depends on K_P

P Control – A Simple Example

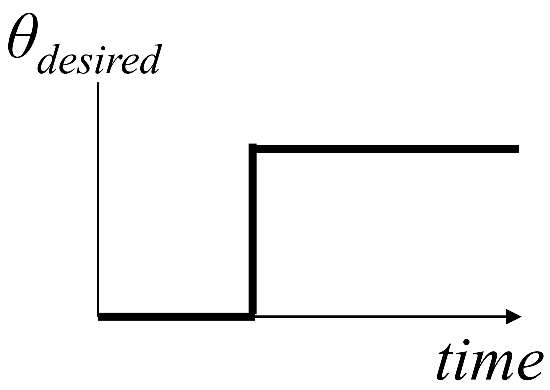
- Block Diagram

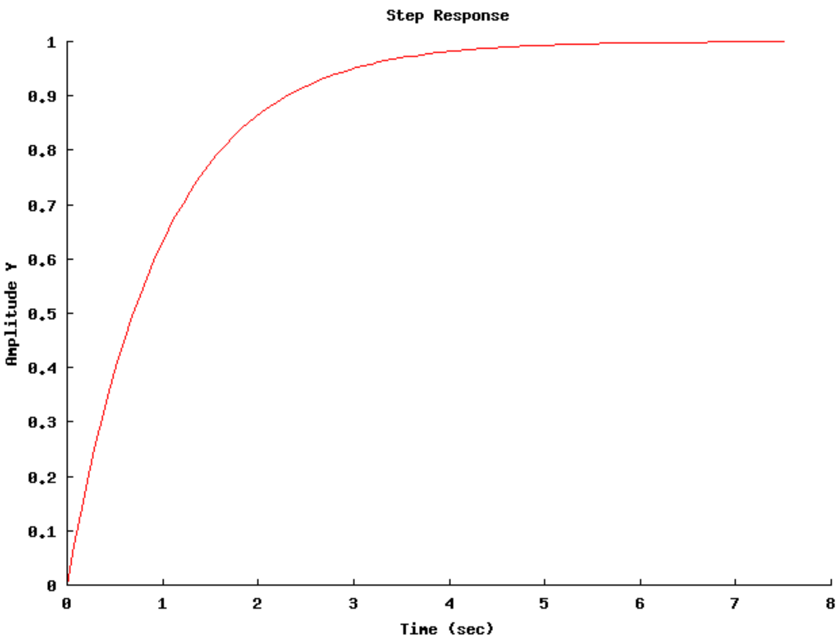
$$u = K_P(\theta_{desired} - \theta)$$



P Control – A Simple Example

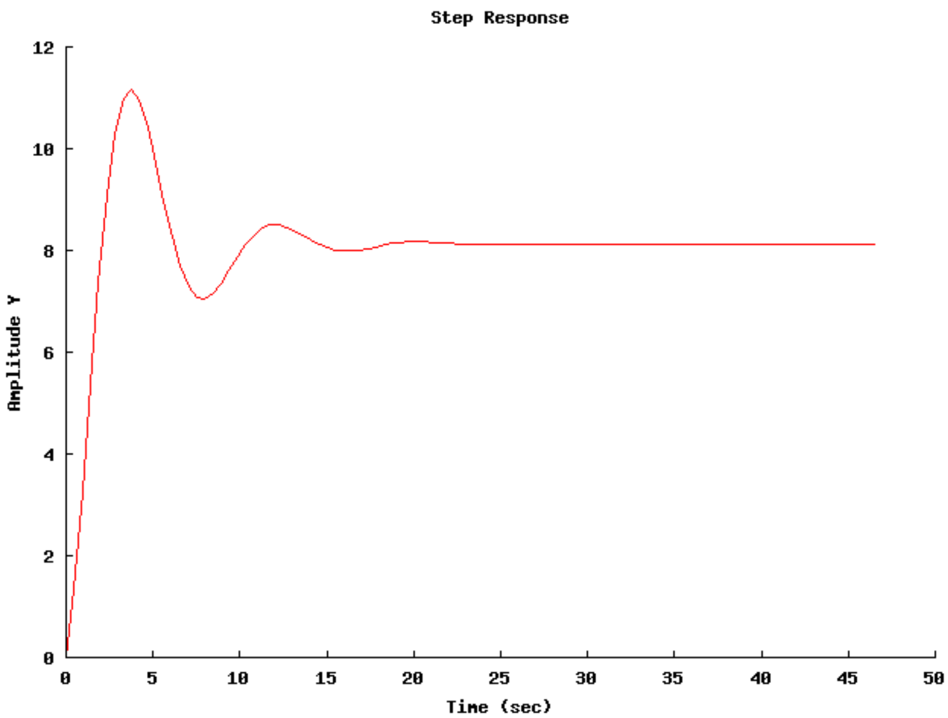
- Time Domain Response of Step Response
- Step from $\theta_{\text{desired}} = 0$ to $\theta_{\text{desired}} = 1$





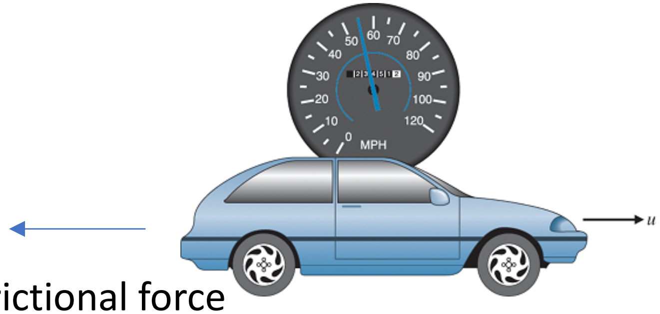
P Control – A Simple Example

- Time Domain Response of Step Response
- Step from $\theta_{\text{desired}} = 0$ to $\theta_{\text{desired}} = 8$



Another Example: Cruise Controllers

- Make a car drive at a desired, reference speed *r*
- Newton's Second Law: $F = ma$
- State: velocity x
- Input: gas/brake u
- Dynamics:



bx: frictional force

$$m\dot{x} = cu - bx$$

$$\dot{x} = \frac{c}{m}u - \gamma x$$

$$\gamma = \frac{b}{m}$$

c = electro-mechanical transmission coefficient

Cruise Controllers

- Assume that we measure the velocity $y = x$
- The control signal should be a function of

$$r - y (= e)$$

- What properties should the control signal have?
  - Small *e* gives small *u*
  - *u* should not be "jerky"
  - *u* should not depend on us knowing *c* and *m* exactly

- Car model: $$\dot{x} = \frac{c}{m}u - \gamma x$$

- Want:

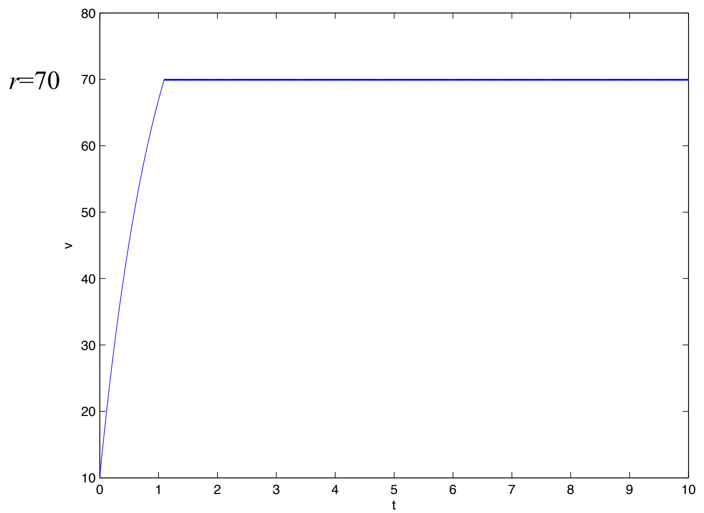
$$x \rightarrow r \text{ as } t \rightarrow \infty \quad (e = r - x \rightarrow 0)$$

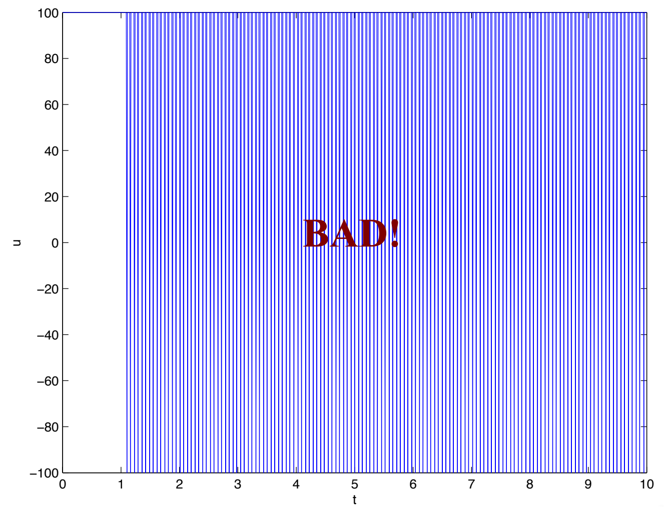
Bang-Bang Control

- **Attempt 1:** Bang-Bang control

$$u = \begin{cases} u_{max} & if\ e > 0 \\ -u_{max} & if\ e < 0 \\ 0 & if\ e = 0 \end{cases}$$

Bumpy ride
Burns out actuators





- Problem: the controller over-reacts to small errors

Proportional Control

- **Attempt 2:** P Control

$$u(t) = K_P e(t)$$

- Intuition: if *e(t)* > 0, the goal velocity is larger than the current velocity. So, command a larger acceleration

- Small error yields small control signals

- Nice and smooth

Proportional Control

- At steady state (x does not change any more)

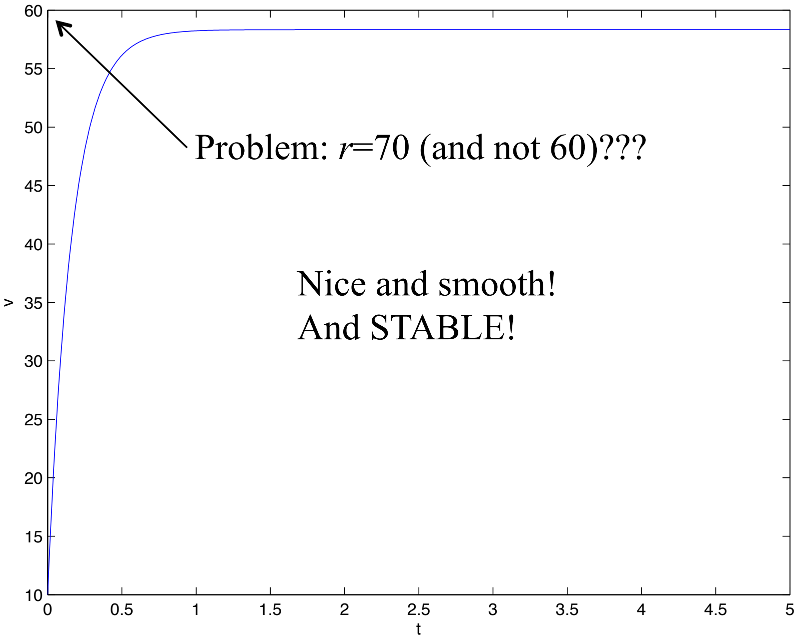
$$\dot{x} = 0 = \frac{c}{m}u - \gamma x$$

$$= \frac{c}{m}k(r - x) - \gamma x$$

$$\rightarrow (ck + m\gamma)x = ckr$$

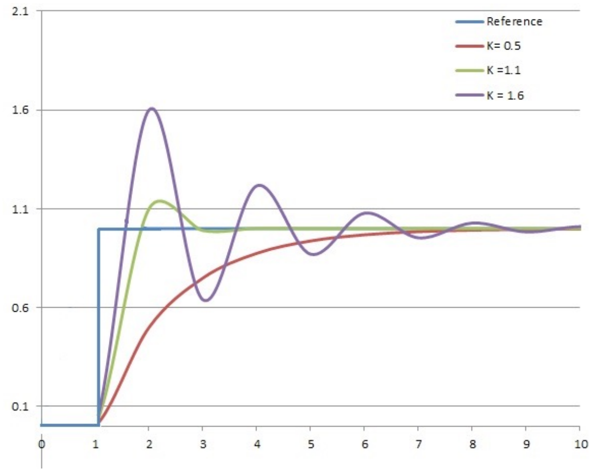
$$\boxed{x = \frac{ck}{ck + m\gamma}r \; < r}$$

$$\dot{x} = \frac{c}{m}u - \gamma x$$



Proportional Control

- We want to drive error to zero quickly
  - This implies large gains
- We want to get rid of steady-state error
  - If we're close to desired output, proportional output will be small. This makes it hard to drive steady-state error to zero.
  - This implies large gains.
- What's wrong with really large gains?
  - Oscillations



PI Controller

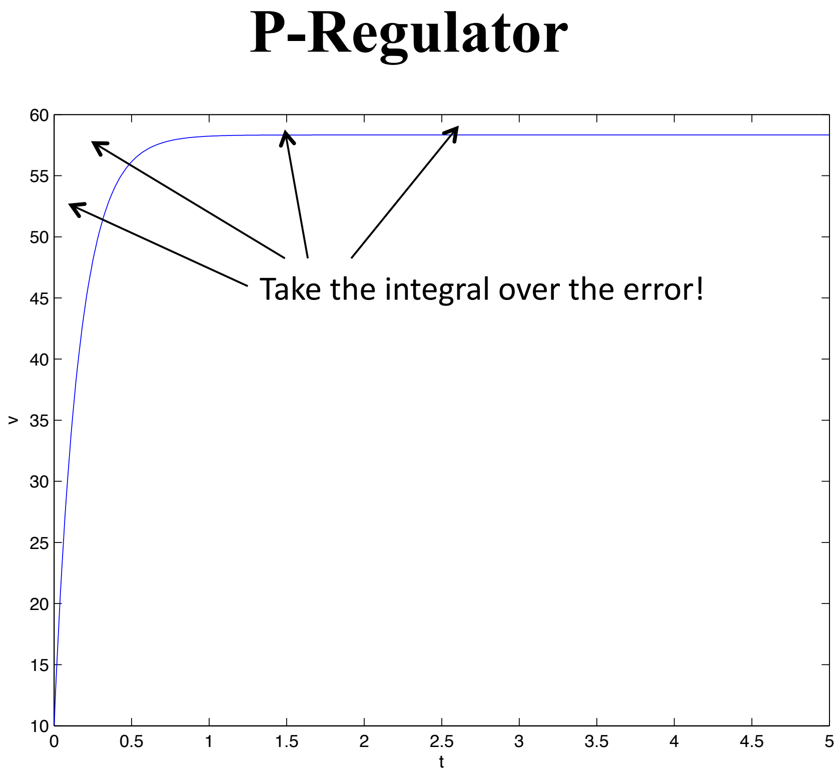
- **Attempt 3:** PI-Controllers

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau$$

- If we have error for a long period of time, it argues for additional correction

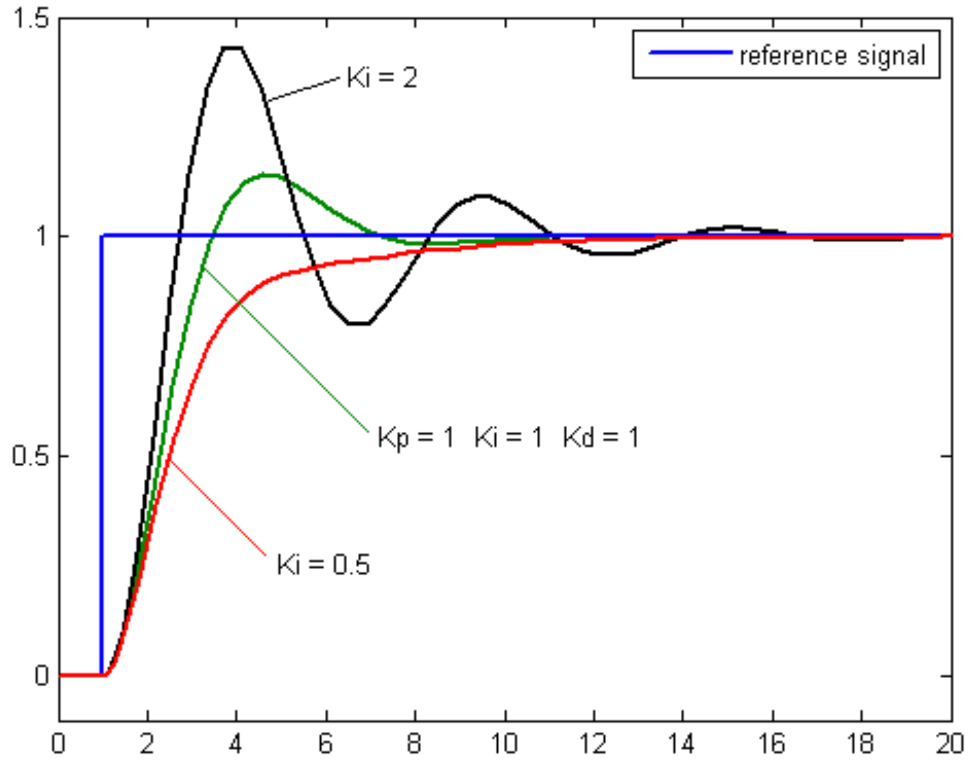
- The integral term in the controller is the sum of the instantaneous error over time and gives the accumulated offset

- Force average error to zero (in steady state)



PI Controller

- Pros:
  - accelerates the movement of the process towards setpoint
  - eliminates the residual steady-state error
- Cons:
  - may result in overshooting the setpoint



Derivative Controller

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

$$K_D \dot{e}(t)$$

- Derivative control is “happy” the error is not changing
  - Things not getting better, but not getting worse either

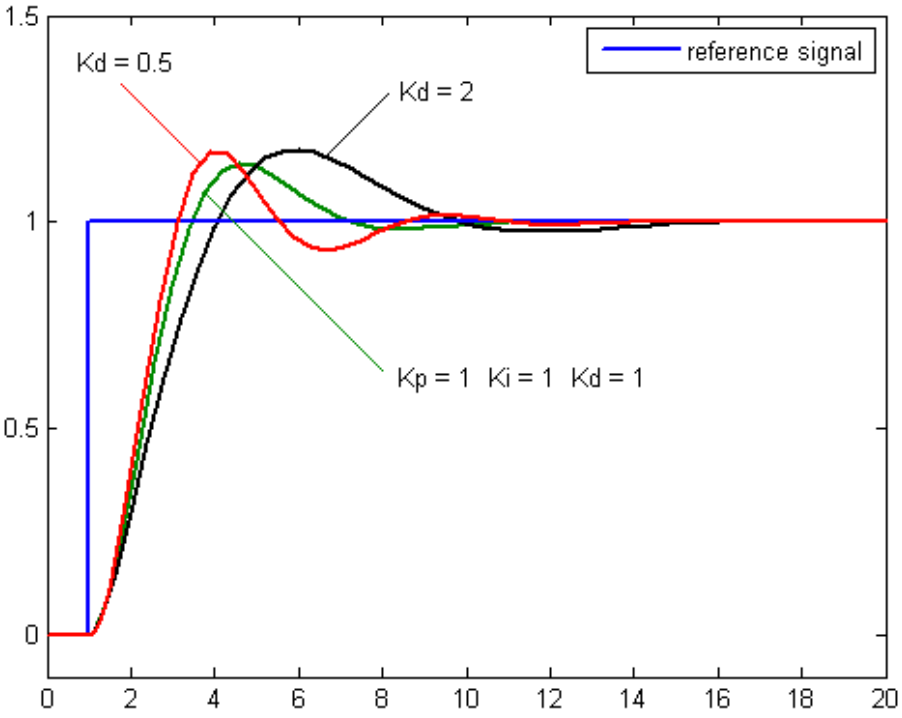
- Estimating a derivative from measurements is fragile, and amplifies noise
- The Derivative term is rarely used along

PD Controller

- **Attempt 4:** PD controller

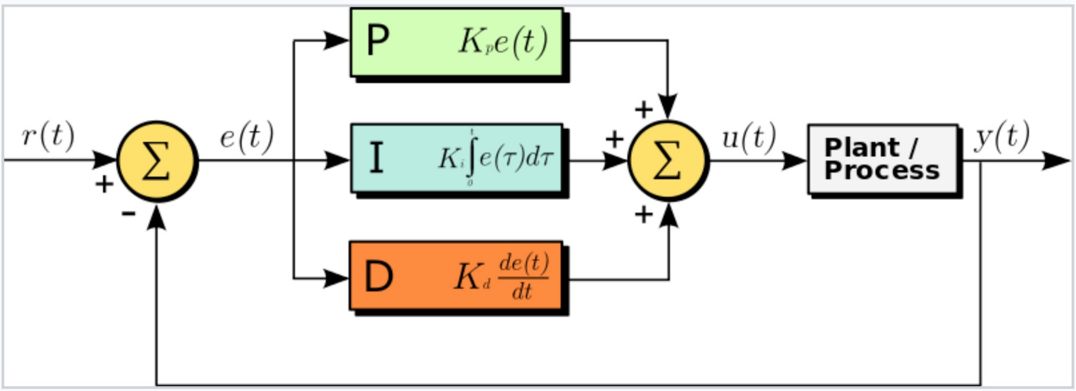
$$u(t) = K_P e(t) + K_D \dot{e}(t)$$

- Combine P and D terms
  - D term helps us avoid oscillation, allowing us to have bigger P terms
    - Faster response
    - Less oscillation



PID Control

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$



PID Control

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt}$$

- **P**: contributes to stability, medium-rate responsiveness
- **I**: tracking and disturbance rejection, slow-rate responsiveness. May cause oscillations
- **D**: fast-rate responsiveness. Sensitive to noise
- **PID**: by far the most used low-level controller.
  - However, stability is not guaranteed

PID Control

- Note: we often won't use all three terms
  - Each type of term has downsides
  - Use only the terms you need for good performance

- Feedback has a remarkable ability to fight uncertainty in model parameters!

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$

PID Controller Parameter Tuning

- If the parameters of the PID controller are chosen incorrectly, the controlled process input can be unstable, i.e., its output diverges, with or without oscillation.

- **Where do PID gains come from?**
  - Analysis
    - Carefully model system in terms of underlying physics and PID controller gains
    - Compute values of PID controller so that system is 1) stable and 2) performs well
  - Empirical experimentation
    - Hard to make models accurate enough: many parameters
    - Often, easy to tune by hand.

PID Controller Parameter Tuning

- Parameter tuning is very important for PID controller
  - **Manual tuning**

1. Increase P term until performance is adequate or oscillation begins
2. Increase D term to dampen oscillation
3. Go to 1 until no improvements possible.
4. Increase I term to eliminate steady-state error.

  - **Ziegler-Nichols method**
  - **Software**
  - …

Characteristics of P, I, and D Gains

Closed Loop Response	Rise Time	Overshoot	Settling Time	Steady State Error
Increase K_P	Decrease	Increase	Small change	Decrease
Increase K_I	Decrease	Increase	Increase	Eliminate
Increase K_D	Small change	Decrease	Decrease	Small change

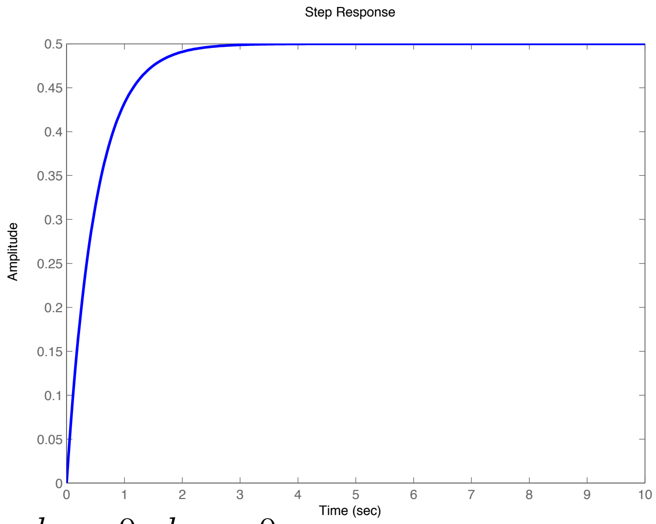
By properly tuning the PID gains

Cruise Controller

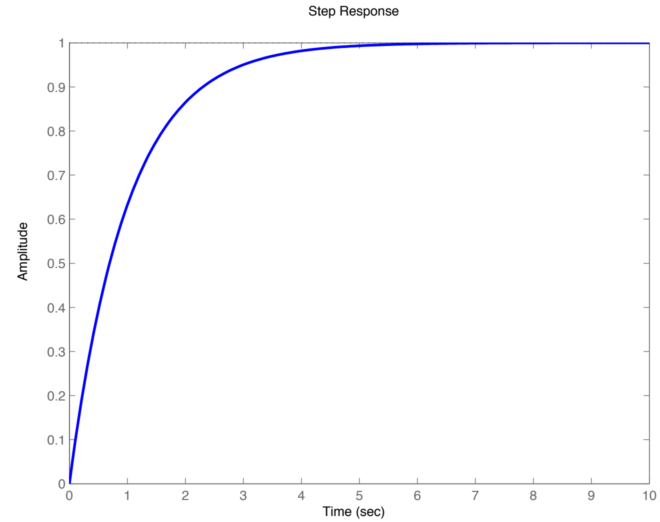
- Let's consider the simplified model

$$\dot{x} = \frac{c}{m}u - \gamma x \qquad\qquad c = 1, m = 1, \gamma = 0.1, r = 1$$

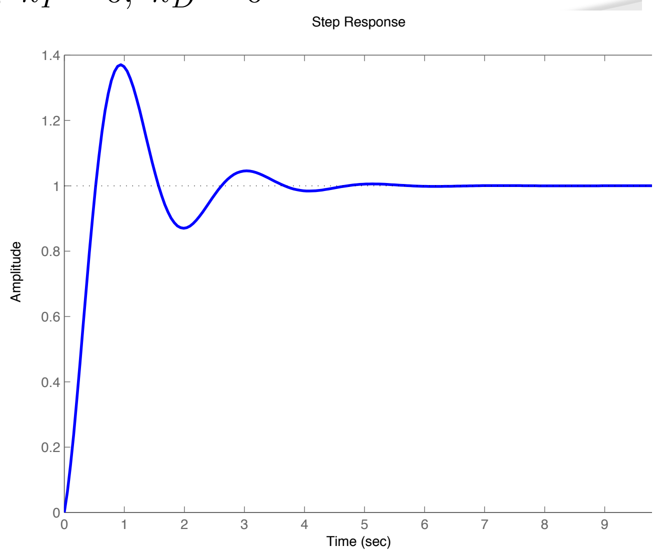
Cruise Controller



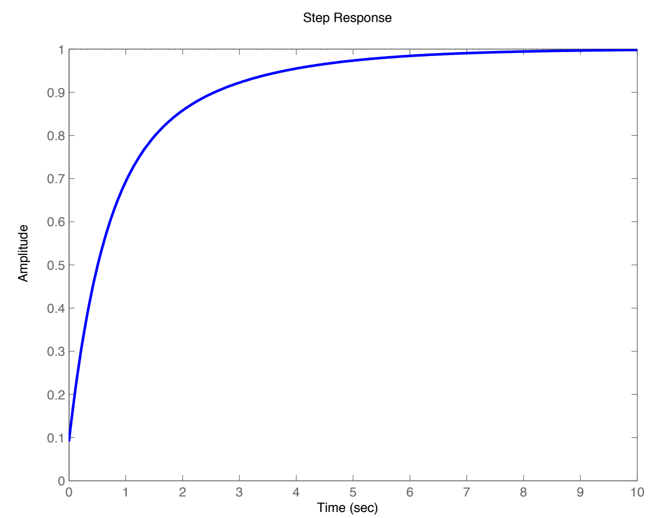
$k_P = 1,\ k_I = 0,\ k_D = 0$



$k_P = 1,\ k_I = 1,\ k_D = 0$



$k_P = 1,\ k_I = 10,\ k_D = 0$



$k_P = 1,\ k_I = 1,\ k_D = 0.1$

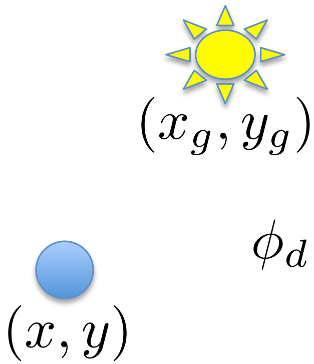
Example: Go To Goal

- How to drive a robot to a goal location?

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases}$$

- Heading error: $e = \theta_{\mathrm{d}} - \theta$

- In this case, the desired heading θ_{d} is time-varying



$$\phi_d = \arctan\left(\frac{y_g - y}{x_g - x}\right)$$

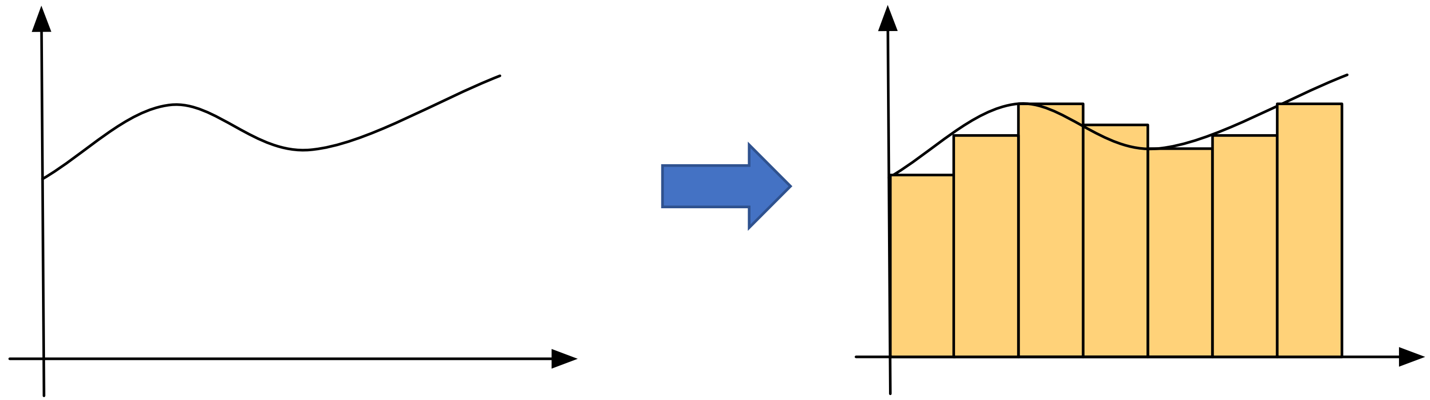
- Control: ω = PID(e)

PID Controller Implementation

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$

$\Delta t\ (sample\ time)$

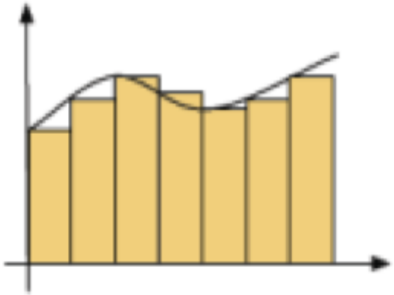
$$\dot{e} \approx \frac{e_{new} - e_{old}}{\Delta t}$$



PID Controller Implementation

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$

$$\Delta t \ (sample\ time) \qquad \dot{e} \approx \frac{e_{new} - e_{old}}{\Delta t}$$



$$\int_0^t e(\tau)d\tau \approx \sum_{k=0}^{N} e(k\Delta t)\Delta t = \Delta t E$$

$$\Delta t E_{new} = \Delta t \sum_{k=1}^{N+1} e(k\Delta t) = \Delta t e\big((N+1)\Delta t\big) + \Delta t E_{old}$$

$$E_{new} = E_{old} + e$$

PID Controller Implementation

- Each time the controller is called

```
read e;
e_dot=e-old_e;
E=E+e;
u=kP*e+kD*e_dot+kI*E;
old_e=e;
```

Note: The coefficients now include the sample time and must be scaled accordingly

- Thank You!