



# Motion Control and PID

[autorob.org](http://autorob.org)

EECS 398  
Intro. to Autonomous Robotics

ME/EECS 567   ROB 510  
Robot Modeling and Control

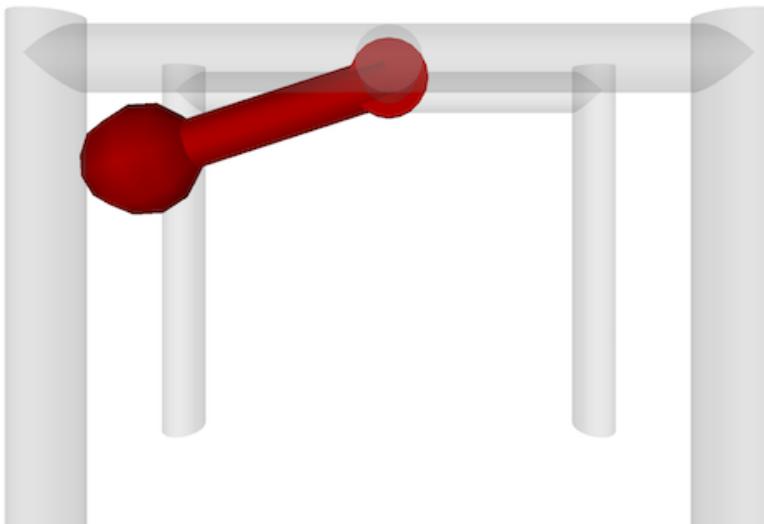
Fall 2018



# Administrivia

- Project 1 (“Path Planning”) due next Monday (Sep 24)
- Class meeting cancelled next Monday (Sep 24)
- Project 2 (“Pendularm”) out next Monday, due October 8th
- Quiz 1 in class next Wednesday (Sep 26)
- Prof. Jenkins will not have office hours next week

# Project 2: Pendularm



[https://raw.githubusercontent.com/autorob/kineval-stencil/master/project\\_pendularm/pendularmI.html](https://raw.githubusercontent.com/autorob/kineval-stencil/master/project_pendularm/pendularmI.html)

```
function init() {...
  pendulum = { // pendulum object
    length:2.0,
    mass:2.0,
    angle:Math.PI/2,
    angle_dot:0.0};

  gravity = 9.81; // Earth gravity
  t = 0; dt = 0.05; // init time
  pendulum.control = 0; // motor

  // next lecture: PID control
  pendulum.desired = -Math.PI/2.5;
  pendulum.desired_dot = 0;
  pendulum.servo =
    {kp:0, kd:0, ki:0};
  ...
}
```

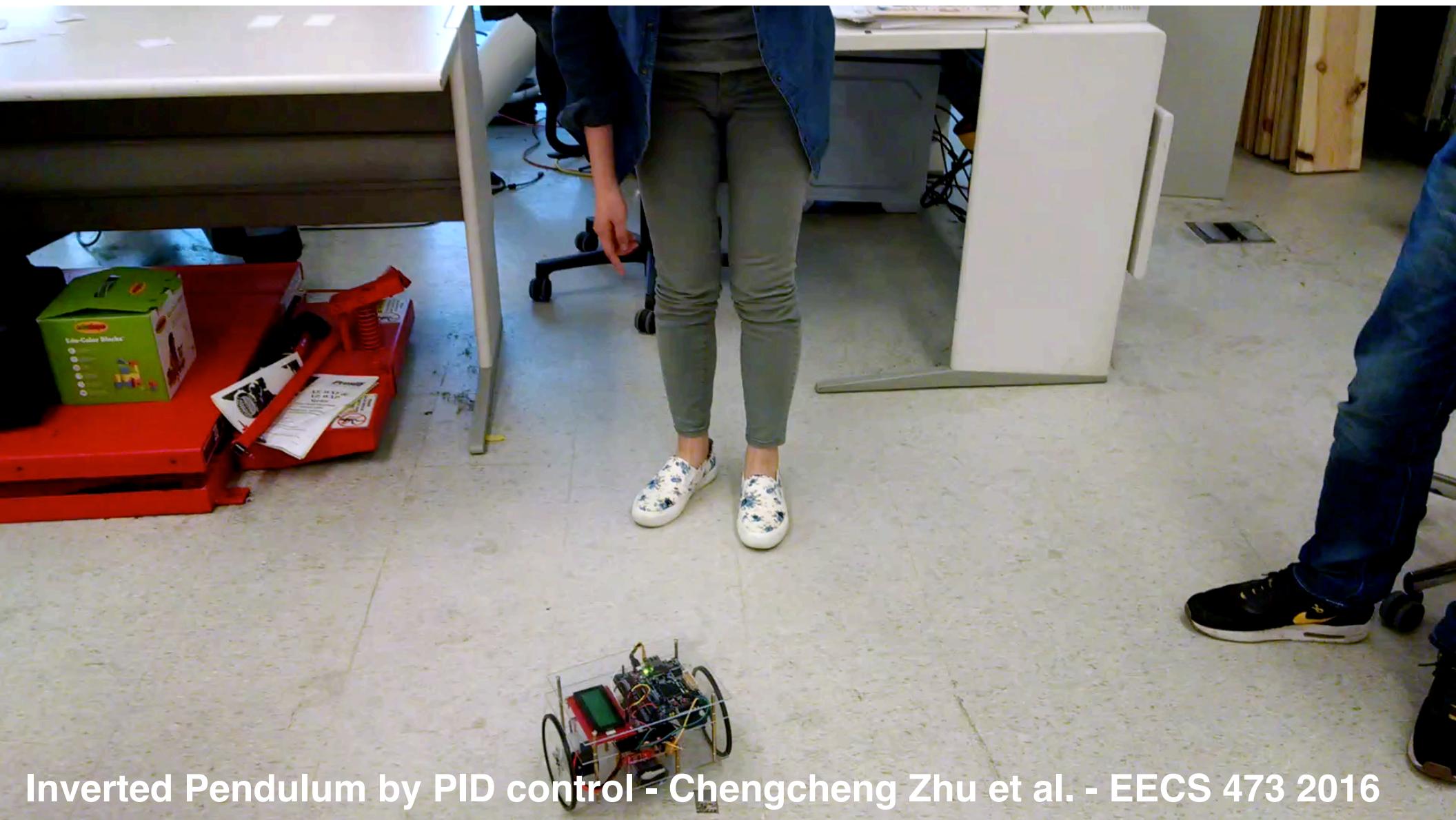
```
function animate() { ...
if (numerical_integrator === "euler") {
    // STENCIL: Euler integrator }
else if (numerical_integrator === "verlet") {
    // STENCIL: basic Verlet integration }
else if (numerical_integrator === "velocity verlet") {
    // STENCIL: velocity Verlet }
else if (numerical_integrator === "runge-kutta") {
    // STENCIL: Runge-Kutta 4 integrator }
else { }
// set the angle of the pendulum
pendulum.geom.rotation.y = pendulum.angle;
t = t + dt; // advance time
...
}

function pendulum_acceleration(p,g) {
    // STENCIL: return acceleration from equations of motion }
```

What can you do with  
pendularm?



Inverted Pendulum by PID control - Chengcheng Zhu et al. - EECS 473 2016



Inverted Pendulum by PID control - Chengcheng Zhu et al. - EECS 473 2016



Monster Inverted Pendulum - K. French et al. - ROB 550 - Fall 2016

Building up to more...

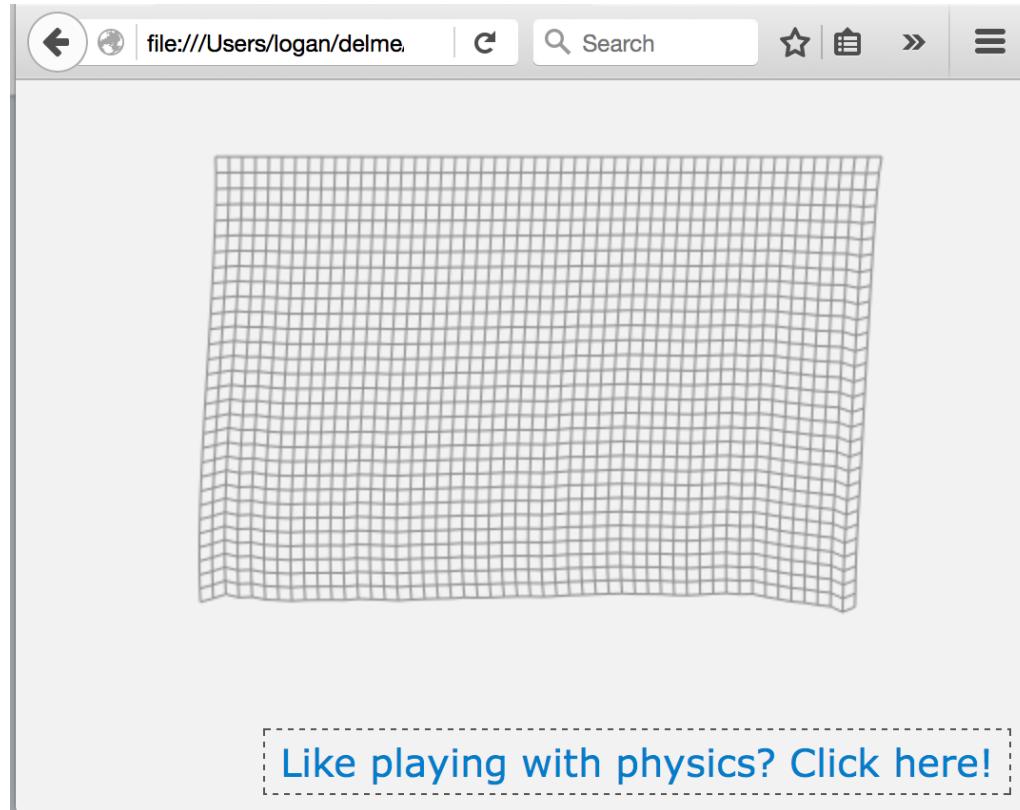


x16

RoBob Ross - Rabideau, Ekins, Rao, Jones - EECS 467 - Winter 2017

Let's start with something fun

# Verlet Simulated Mass-Spring Cloth

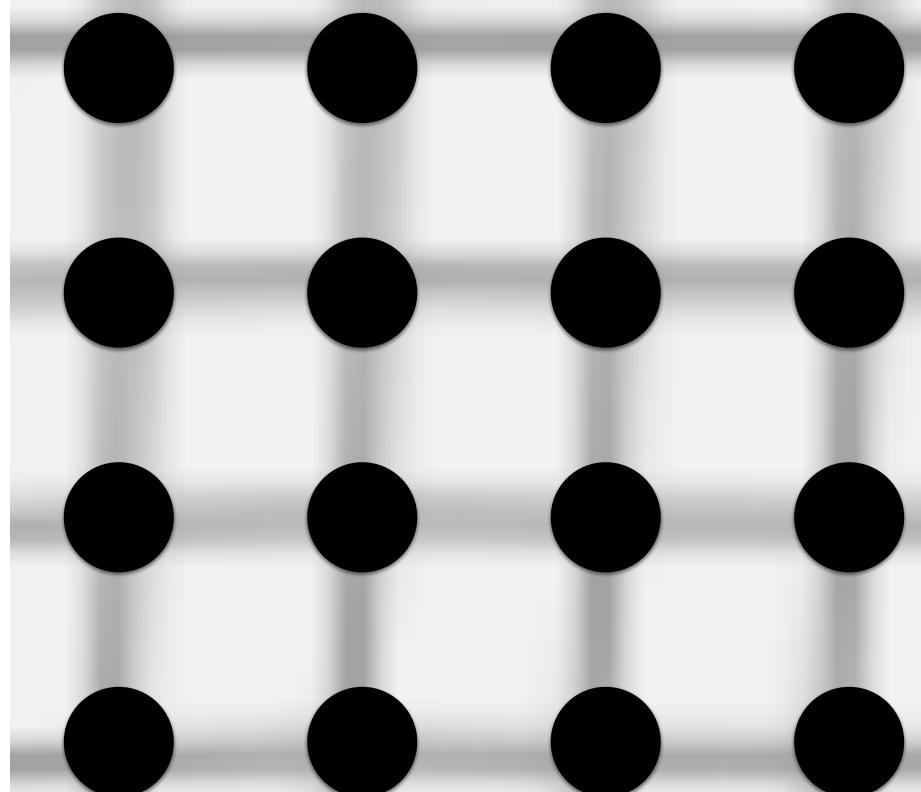


- Cloth represented as a network of point-mass nodes
- Neighboring nodes connected by high stiffness springs

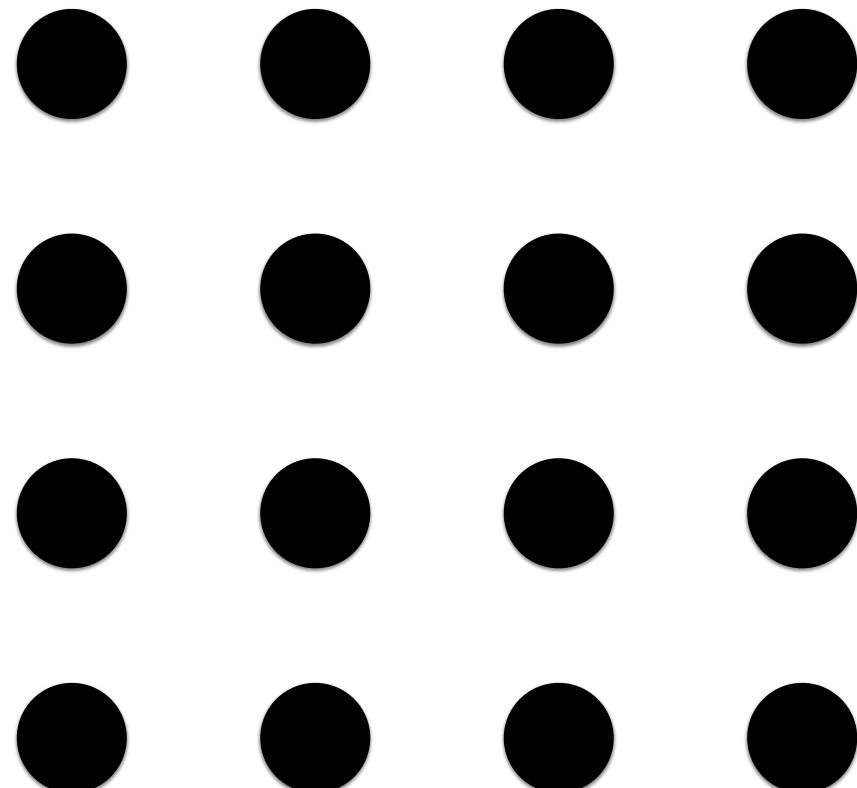
<http://codepen.io/dissimulate/pen/KrAwx>

<https://github.com/Dissimulate/Tearable-Cloth>

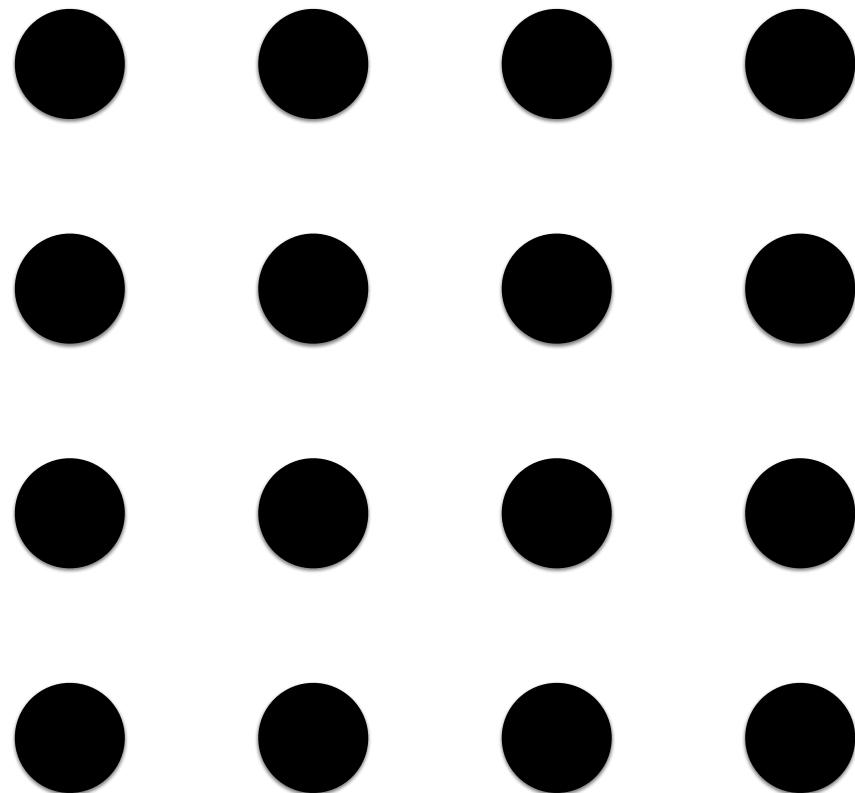
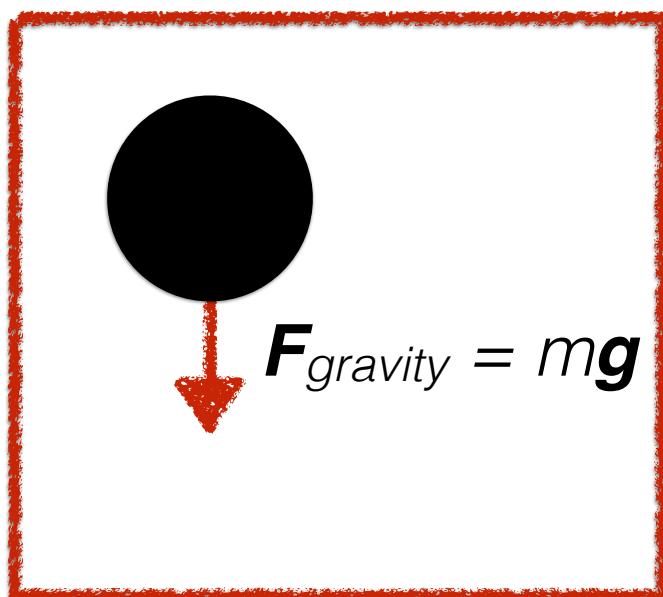
# Start with point-mass particles



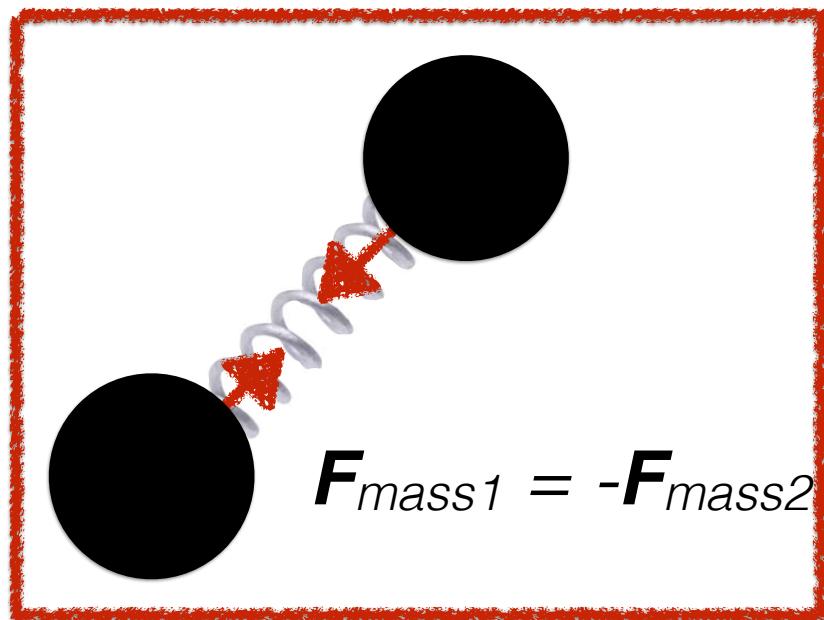
# Start with point-mass particles



# Apply gravity

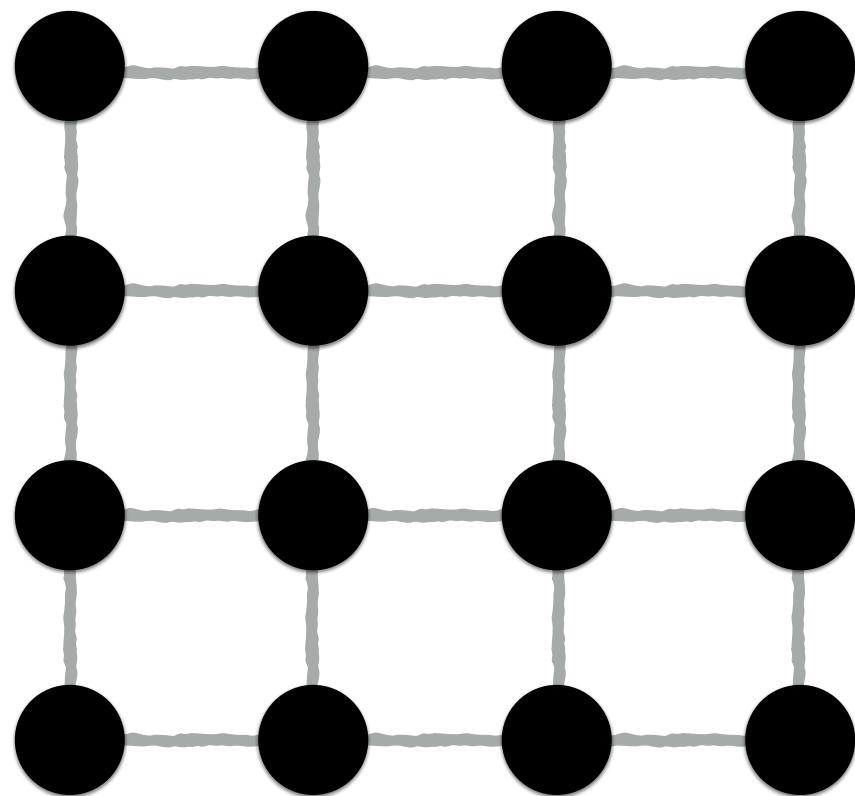


# Enforce spring force constraints

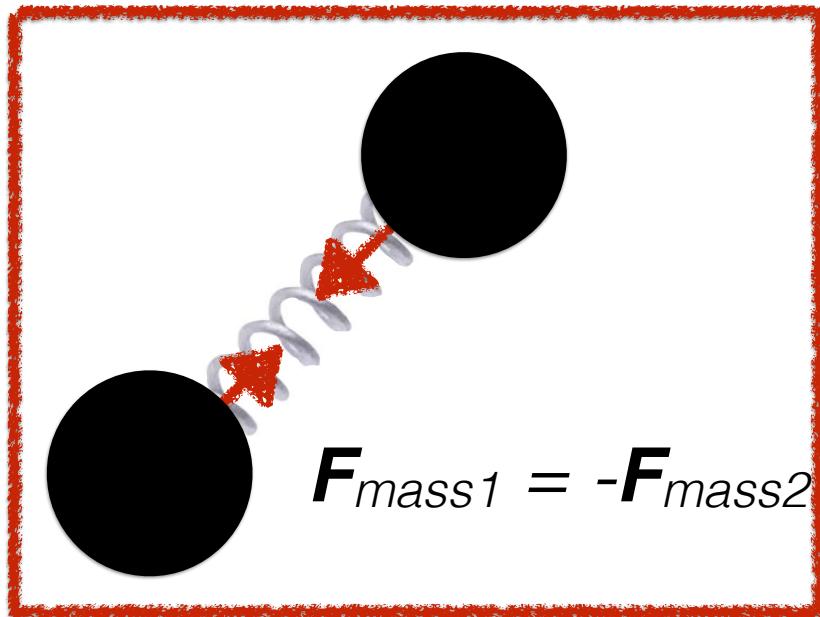


$$\mathbf{F}_{\text{mass1}} = -\mathbf{F}_{\text{mass2}}$$

spring with stiffness 1  
and rest length  $r$



# Enforce spring force constraints



spring with stiffness 1  
and rest length  $r$

enforced through constraint relaxation  
(multiple iterations of Gauss-Seidel)

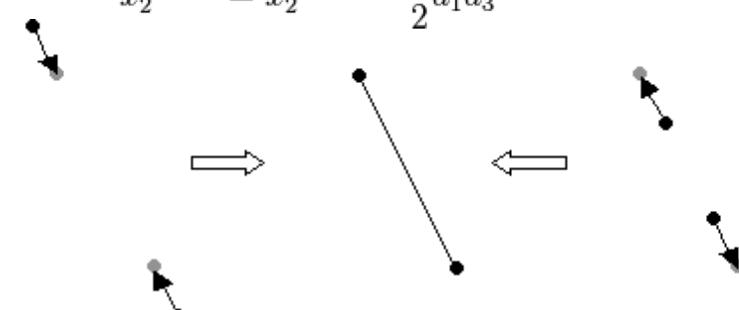
$$d_1 = x_2^{(t)} - x_1^{(t)}$$

$d_2 = \|d_1\|$  Distance between particles

$$d_3 = \frac{d_2 - r}{d_2}$$
 Distance from rest

$$x_1^{(t+\Delta t)} = \tilde{x}_1^{(t+\Delta t)} + \frac{1}{2} d_1 d_3$$
 Correction towards

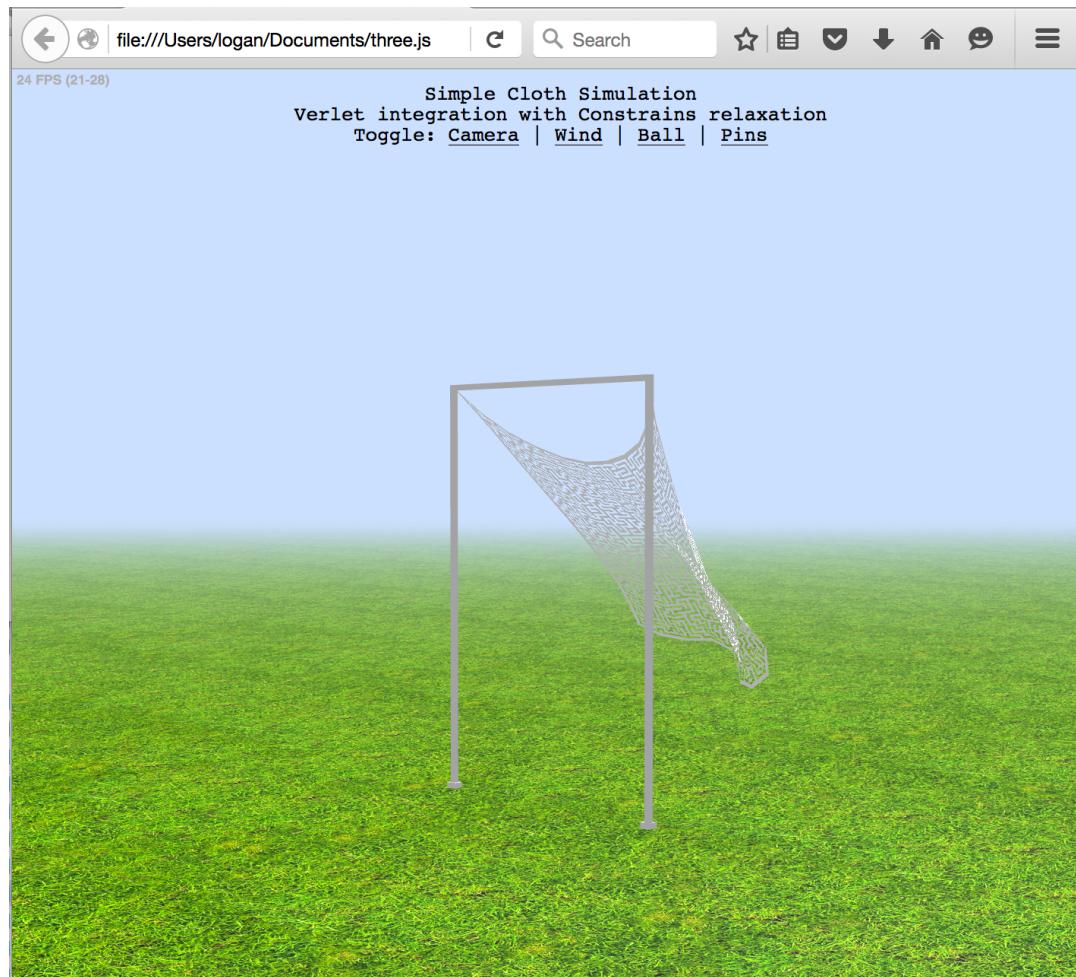
$$x_2^{(t+\Delta t)} = \tilde{x}_2^{(t+\Delta t)} - \frac{1}{2} d_1 d_3$$
 rest length



Dist. too large

Correct distance

Dist. too small

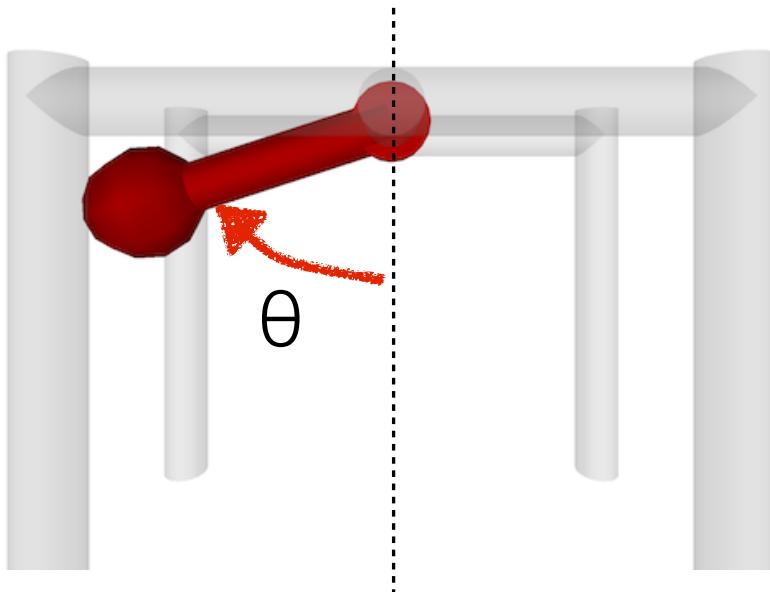


[http://threejs.org/examples/webgl\\_animation\\_cloth](http://threejs.org/examples/webgl_animation_cloth)

# Pendulum Revisited

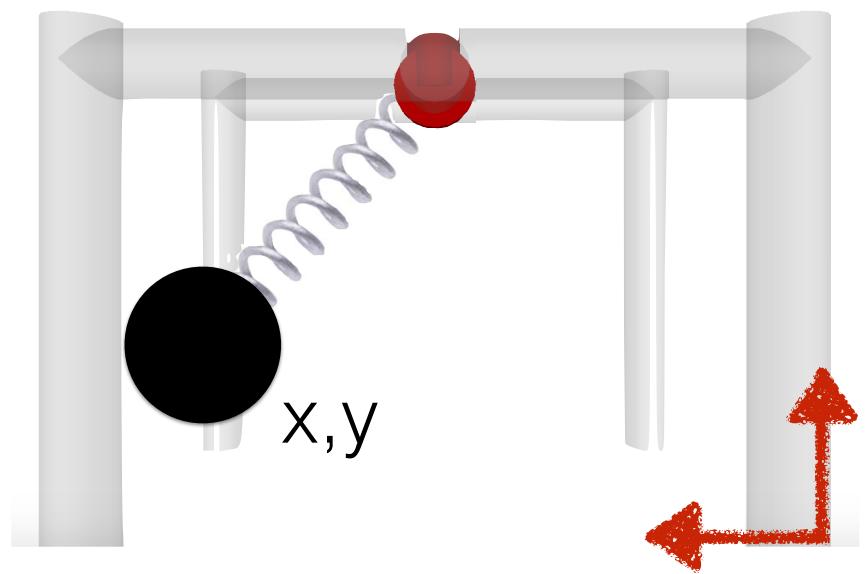
# Pendulum Revisited

## Generalized coordinates



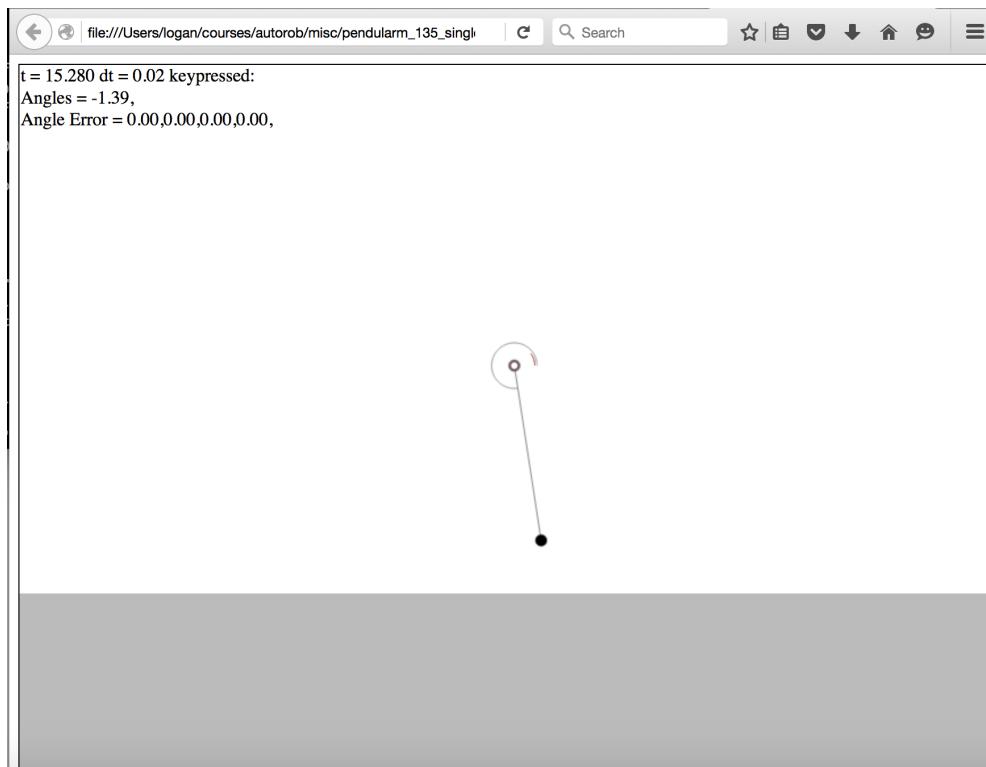
Simulate the motion of DOFs  
(pendulum angle) directly

## Maximal coordinates

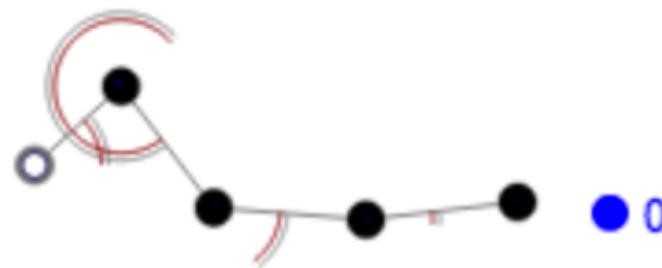


Simulate the motion of individual  
bodies (pendulum bob location)  
corrected by constraints (spring)

# Maximal Pendularm

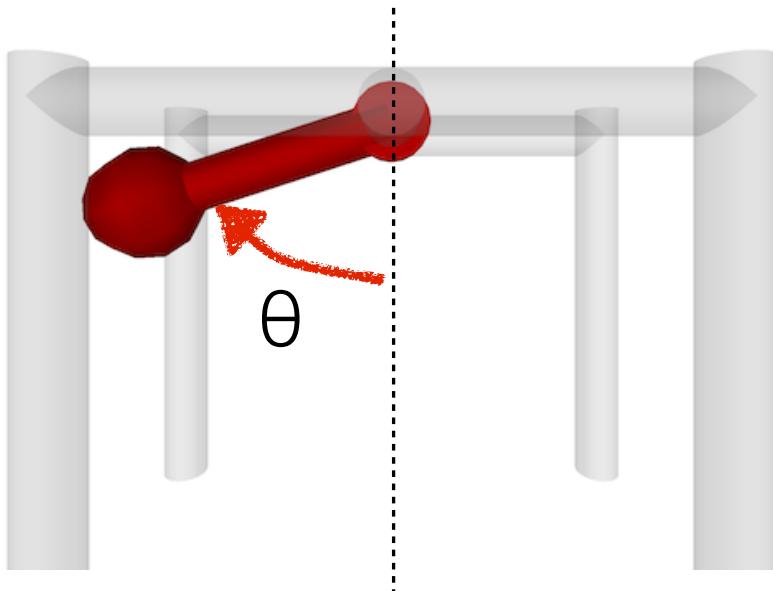


# Maximal N-Link Pendularm



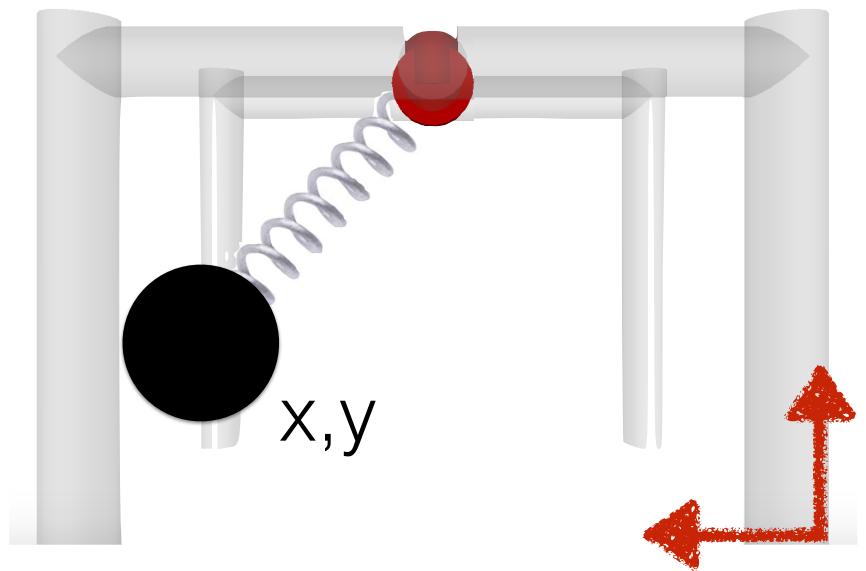
# Why use generalized coordinates?

## Generalized coordinates



Simulate the motion of DOFs  
(pendulum angle) directly

## Maximal coordinates



Simulate the motion of individual  
bodies (pendulum bob location)  
corrected by constraints (spring)

# Motor force expressed wrt DOFs

Equation of motion  
(with rotational inertia  $I$ )

$$I\ddot{\theta} = -mgl \sin(\theta) + \tau$$

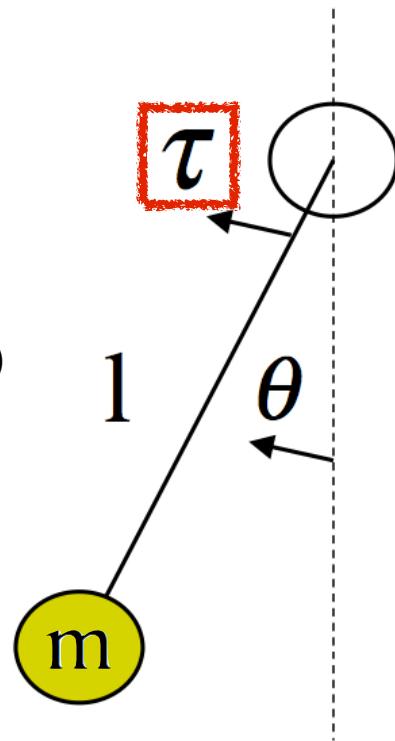
with Parallel Axis Theorem ( $I=mr^2$ )

$$\ddot{\theta} = -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2}$$

Numerical integration over time

$$\theta_{t+dt} = \theta_t + \dot{\theta}_t dt$$

$$\dot{\theta}_{t+dt} = \dot{\theta}_t + \ddot{\theta}_t dt$$

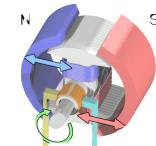


Motor produces torque  
(angular force)

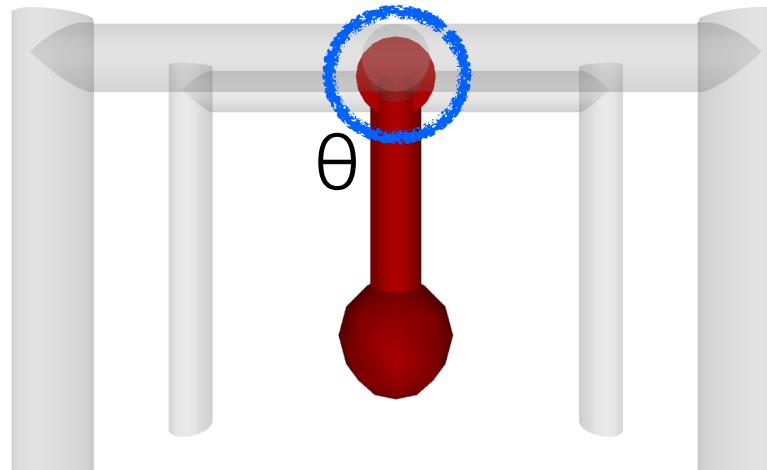
Angle expresses pendulum  
range of motion

Pendulum of length  $l$  with  
point mass  $m$

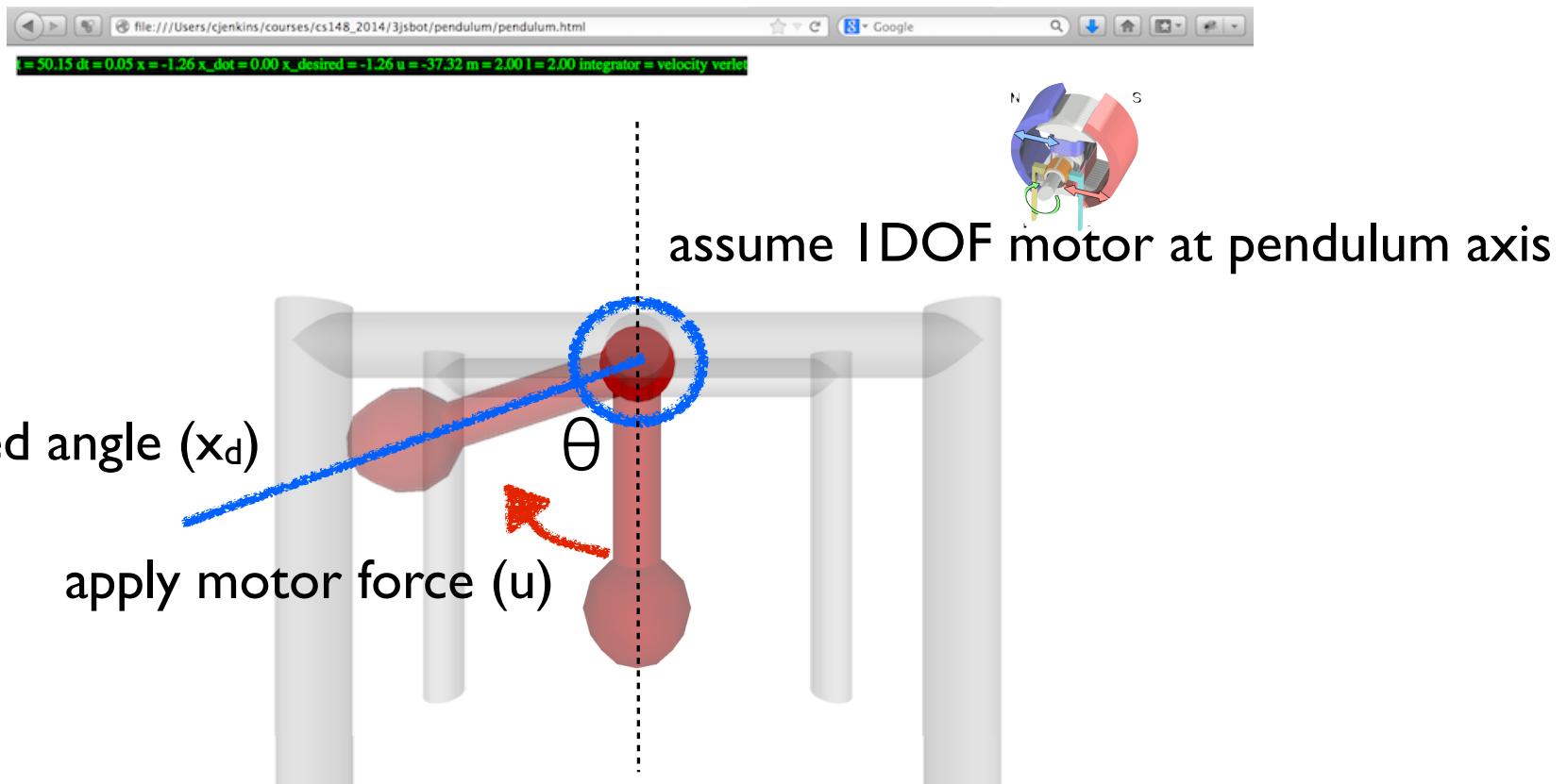
# Let's apply some motor forces

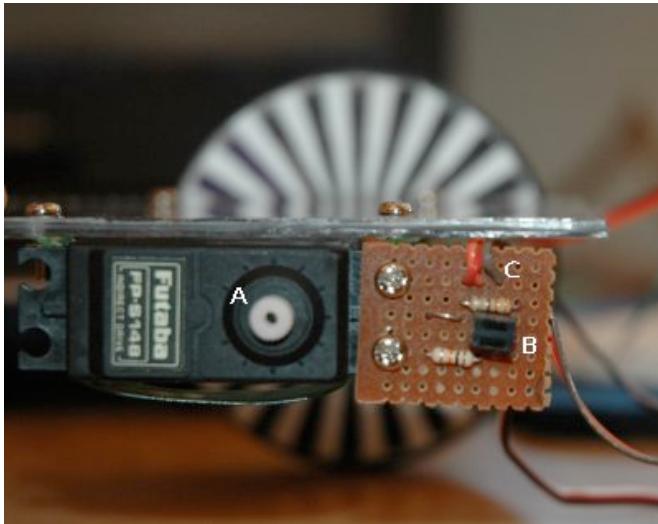


assume 1DOF motor at pendulum axis



# Can we control the pendulum?



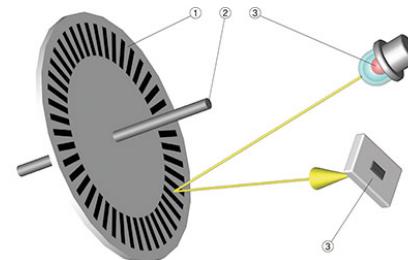
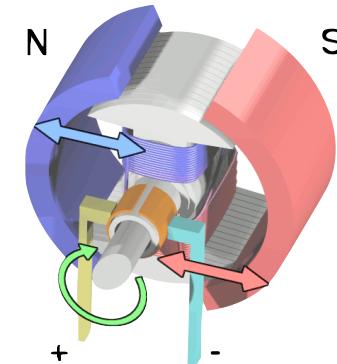


Servos have:

- actuators to produce motion
- proprioception to sense pose
- controller to regulate current to motor

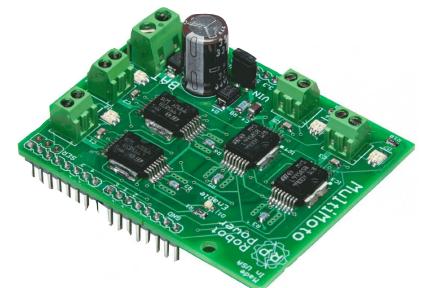


Actuator  
(electric)



Proprioception  
(optical encoder)

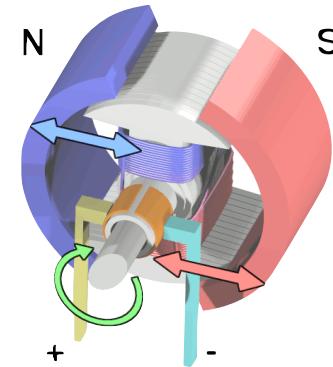
Motor Controller  
(4 channel H-Bridge)



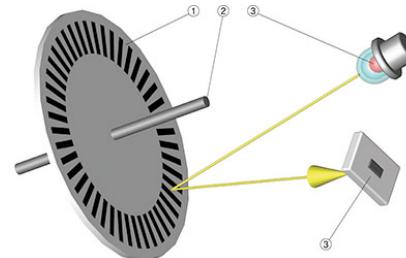


## WHAT IS INSIDE A SERVO?

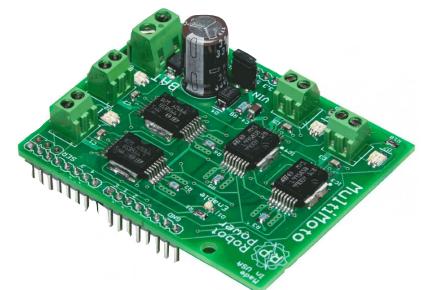
Actuator  
(electric)



Proprioception  
(optical encoder)



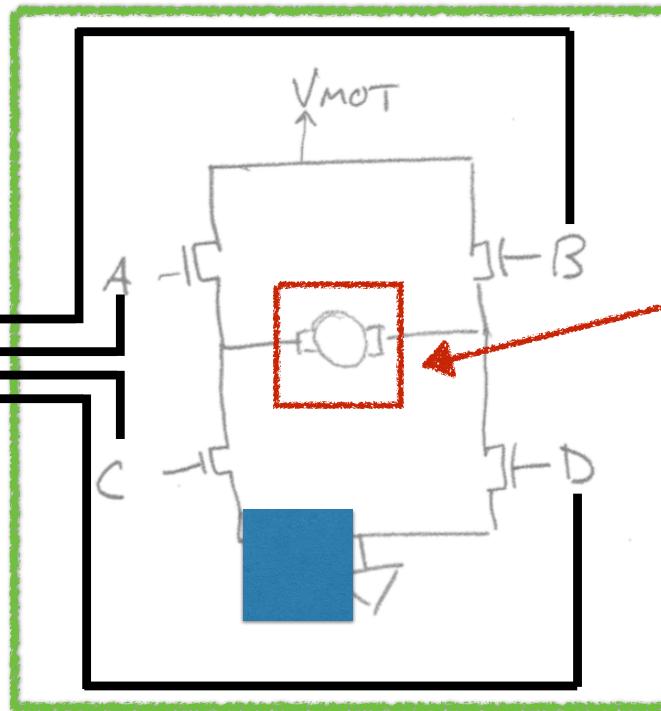
Motor Controller  
(4 channel H-Bridge)



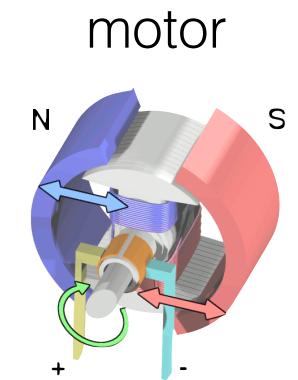
**WHAT IS INSIDE A SERVO?**

# Control Signal

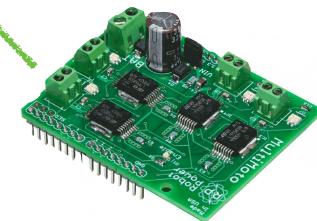
PWM signal sent  
to H-bridge  
controller.



H-bridge sends  
proportional  
current to motor.



Current actuates  
motor motion  
about axis.

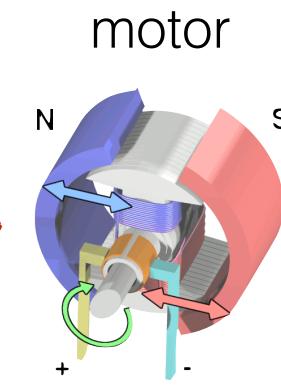
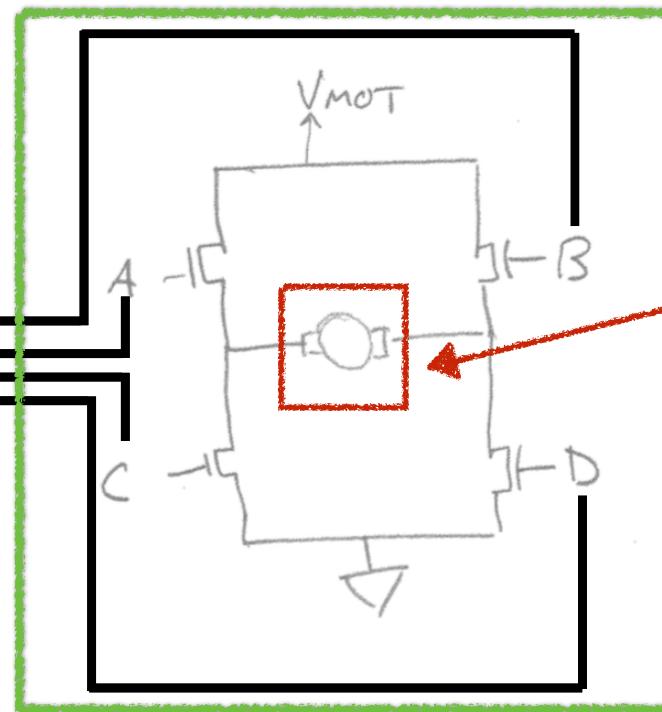
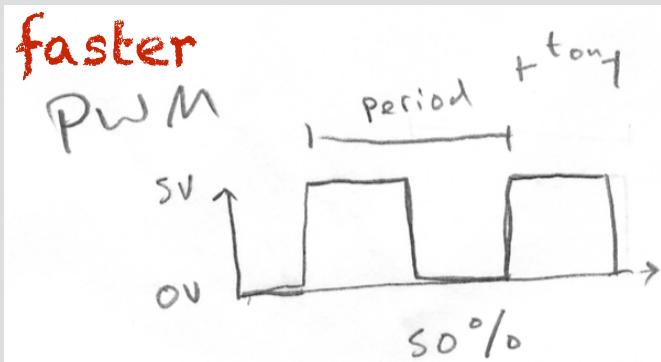


motor controller

PWM  
(Pulse Width Modulation)

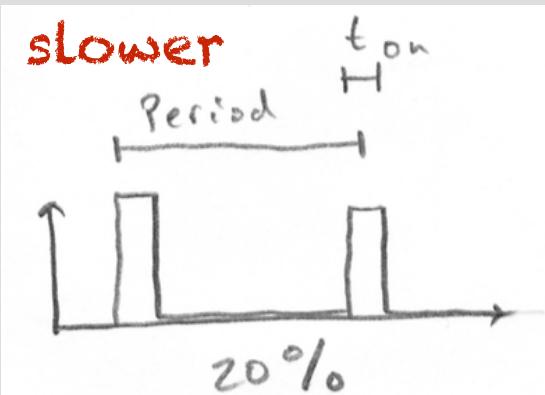
# Control Signal

PWM signal sent  
to H-bridge  
controller.



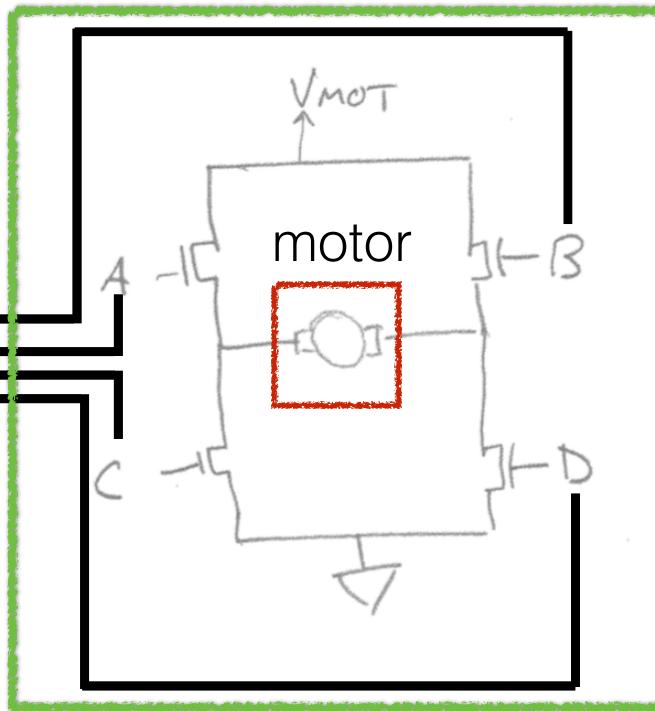
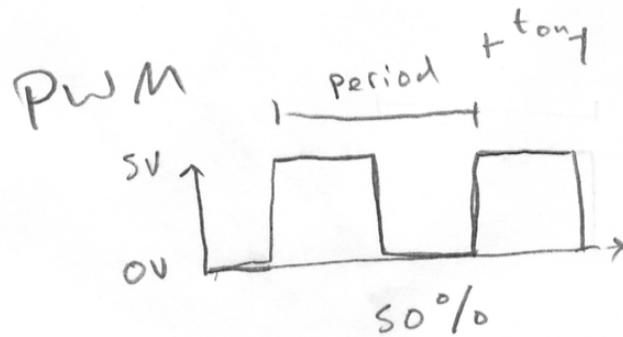
PWM has a periodic duty cycle on each line.

Percentage of time in high voltage determines speed of motor



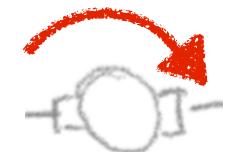
# Control Signal

motor controller



**spin forward:**

A and D



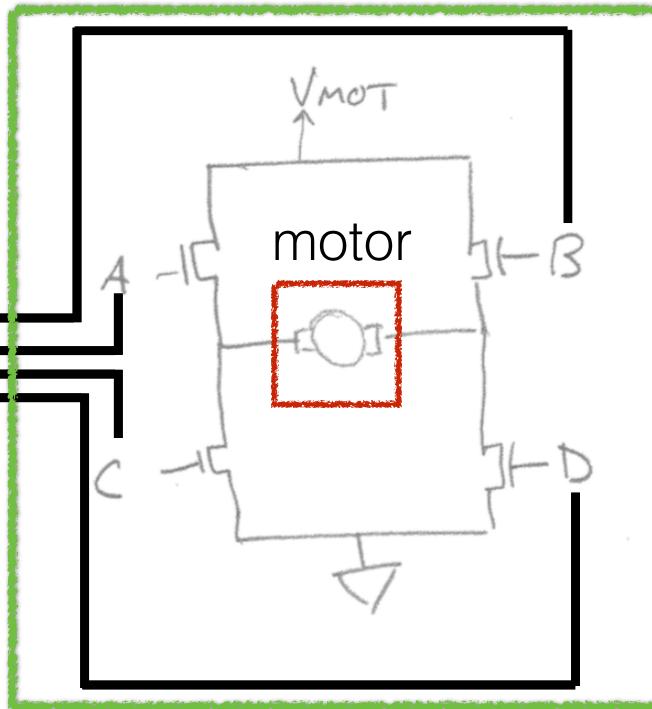
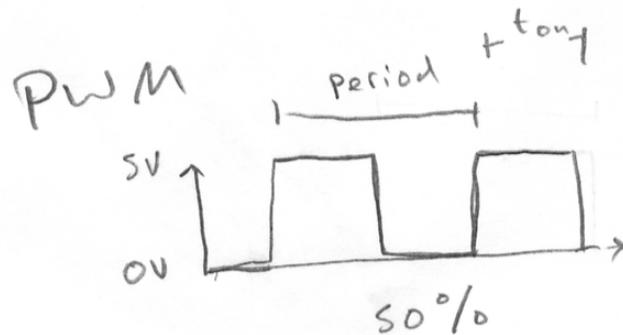
**spin backward:**

B and C



# Control Signal

motor controller



what if A and C?

**spin forward:**  
A and D

**spin backward:**  
B and C

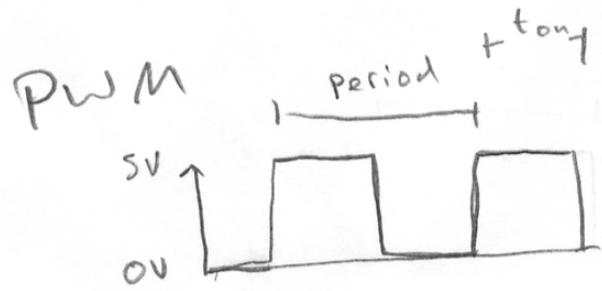
**what if none are on?**

**what if C and D?**

**what if A and B?**

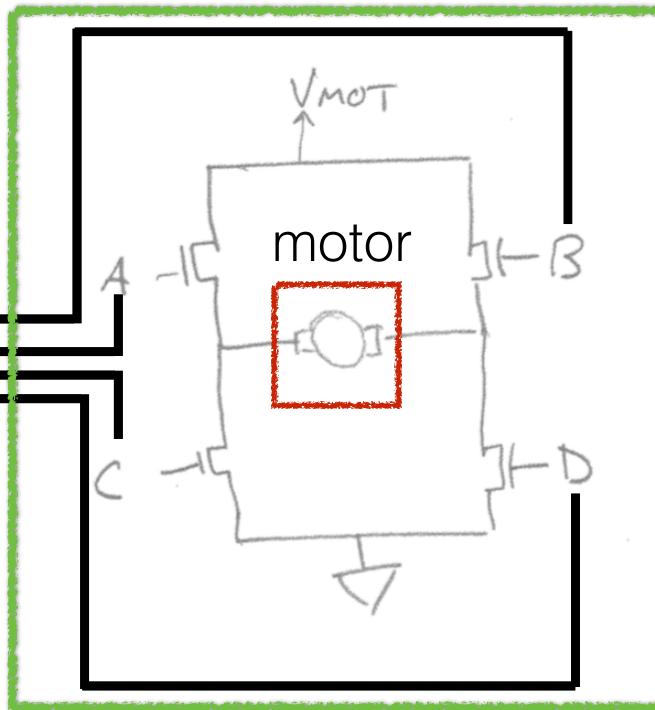
# Control Signal

motor controller



**what if A and C?**

No more  
robot.  
(our hardware prevents  
this)



**spin forward:**  
A and D

**spin backward:**  
B and C

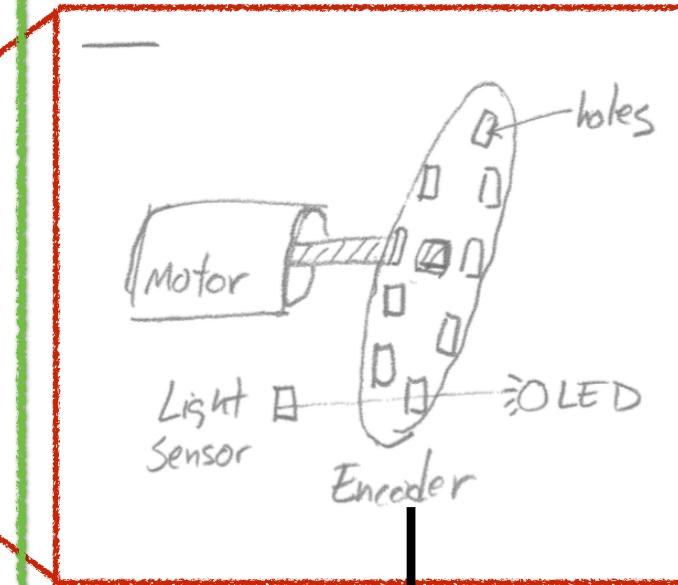
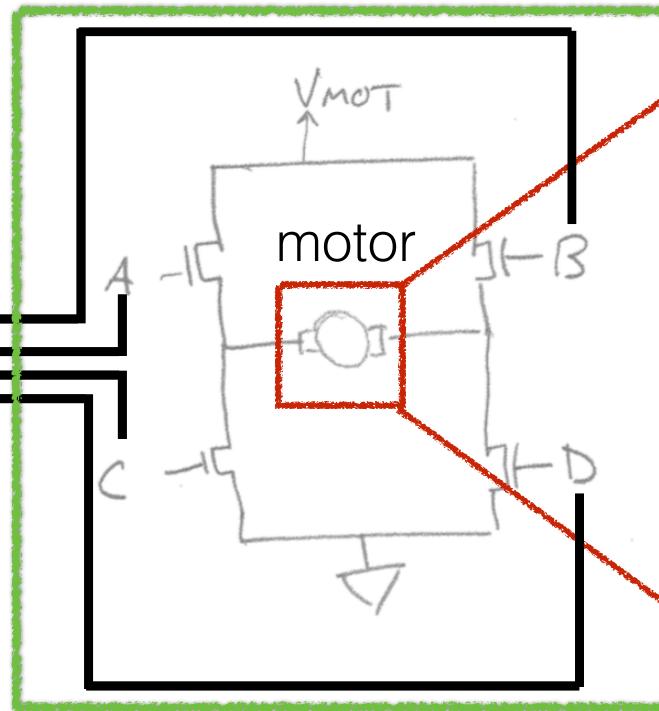
**what if none are on?**  
open circuit  
no current -> no torque

**what if C and D?**  
negative torque -> brake

**what if A and B?**  
negative torque -> brake

# Servo Control

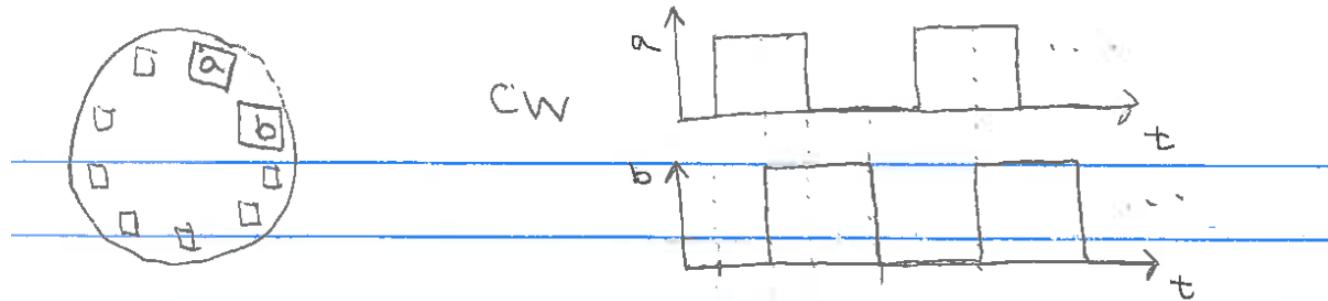
motor controller



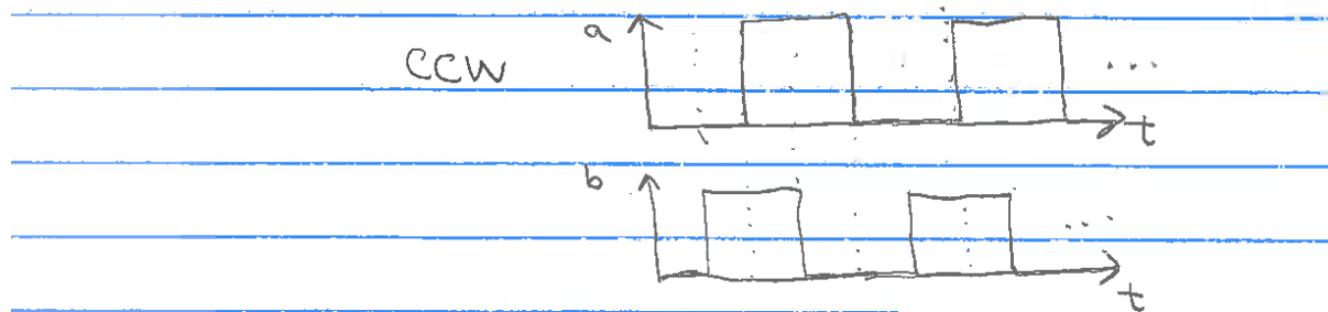
servo controller  
generates control signal

encoder sends joint  
state to back to  
servo controller

# Quadrature encoders



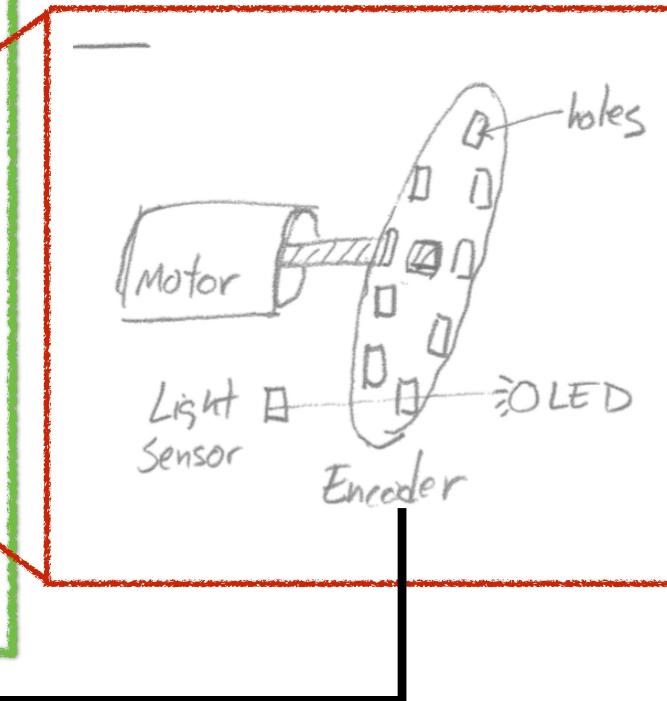
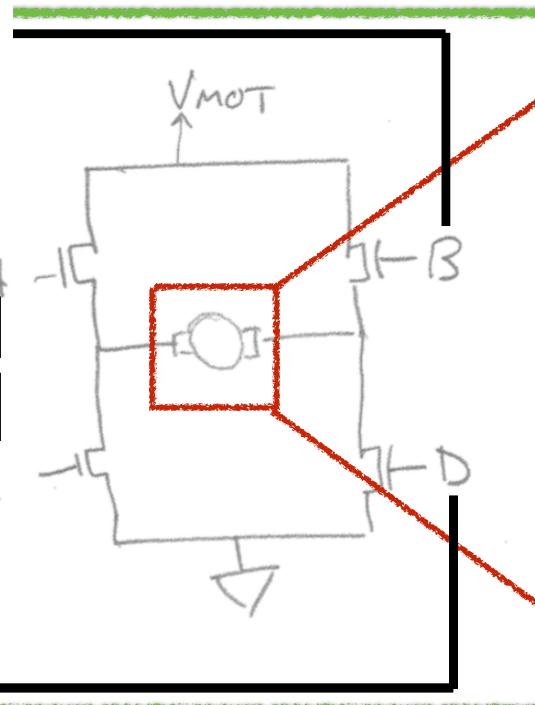
Bit seq: 00 10 11 01 00 10 11 01 ...



Bit seq 00 01 11 10 00 01 11 10 ...

## HOW DO WE IMPLEMENT A SERVO CONTROLLER?

### Servo Control



# HOW DO WE IMPLEMENT A SERVO CONTROLLER?

## Servo Control

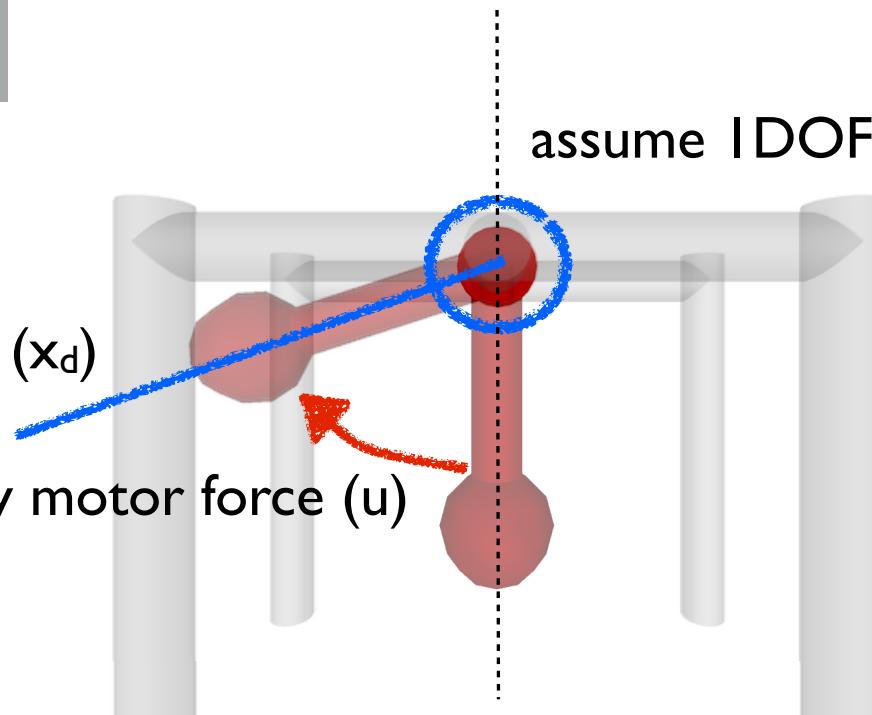
Users/cjenkins/courses/cs148\_2014/3jsbot/pendulum/pendulum.html

26 x\_dot = 0.00 x\_desired = -1.26 u = -37.32 m = 2.00 l = 2.00 integrator = velocity verlet

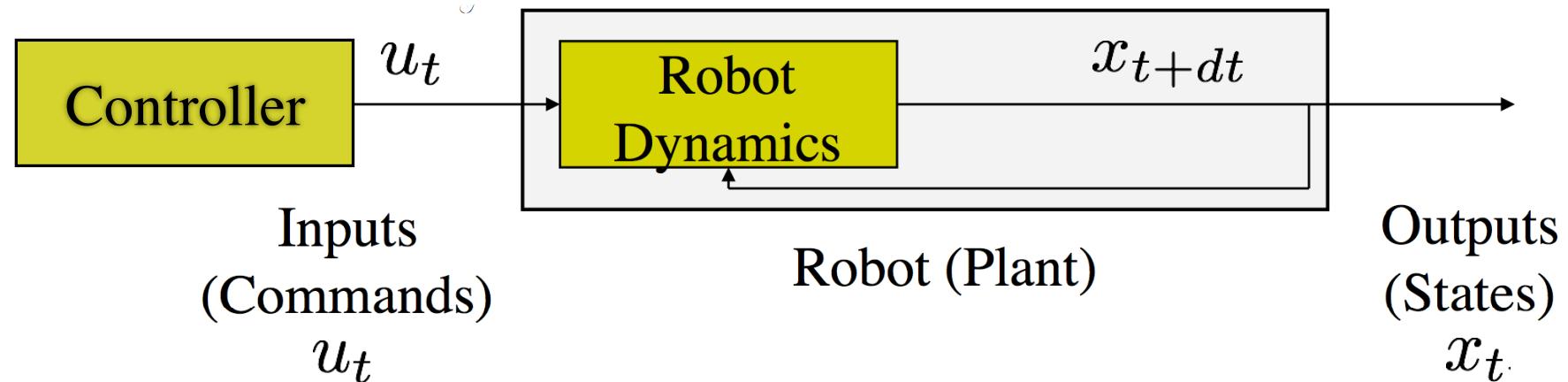
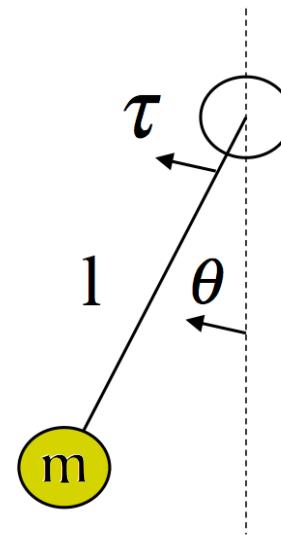
assume 1DOF motor at pendulum axis

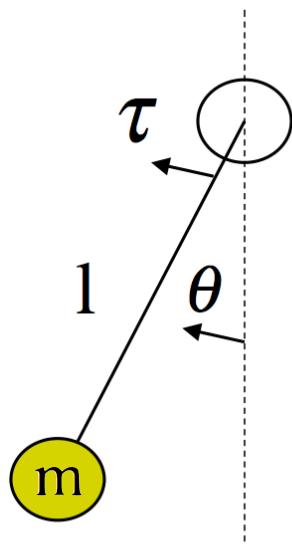
maintain desired angle ( $x_d$ )

apply motor force (u)



# Servo Control

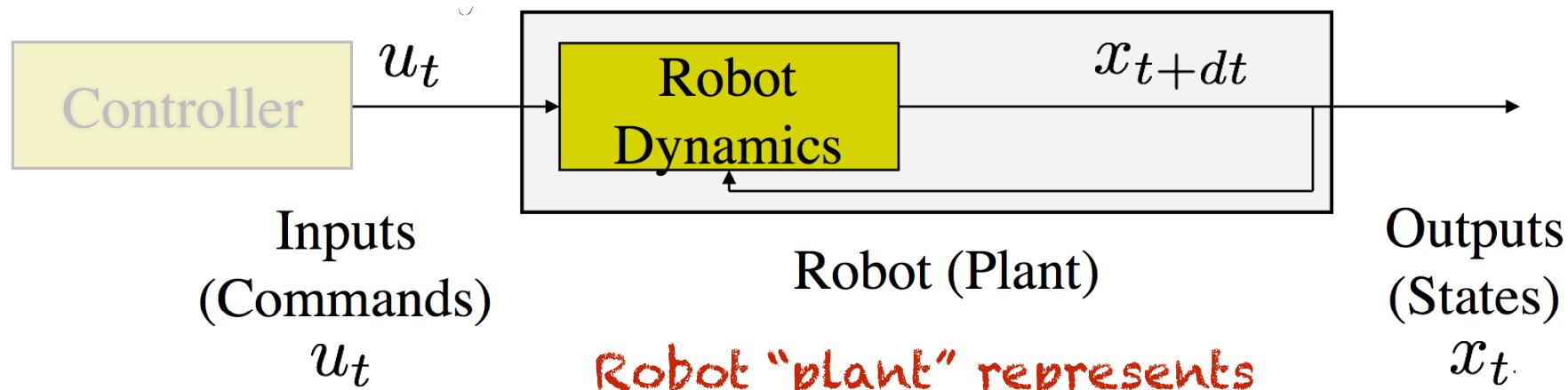




$$\ddot{\theta} = -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2}$$

Motor command used  
within equations of motion

$$\dot{x} = f(t, x, u)$$

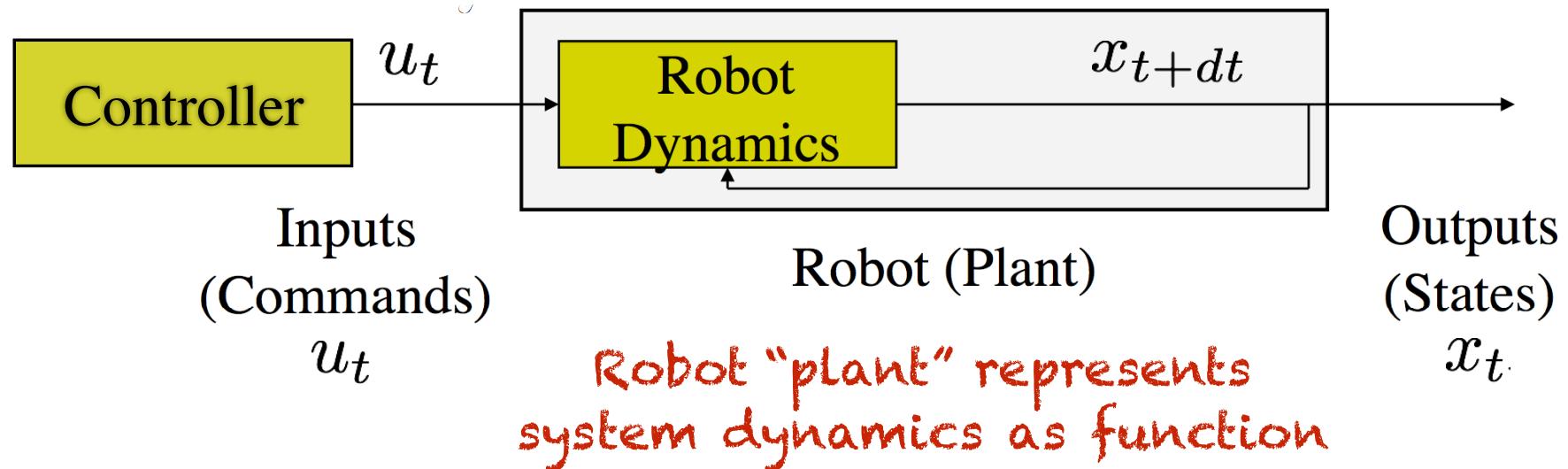


Robot "plant" represents  
system dynamics as function

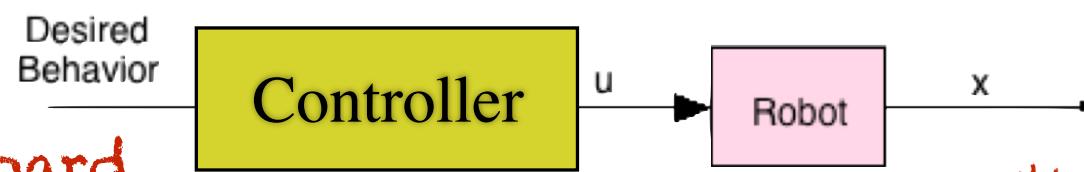
$x_t$

Servo controller generates  
motor commands based  
on desired state

$$u = \pi(t, x, x_d) \quad \dot{x} = f(t, x, u)$$



## Open Loop Control



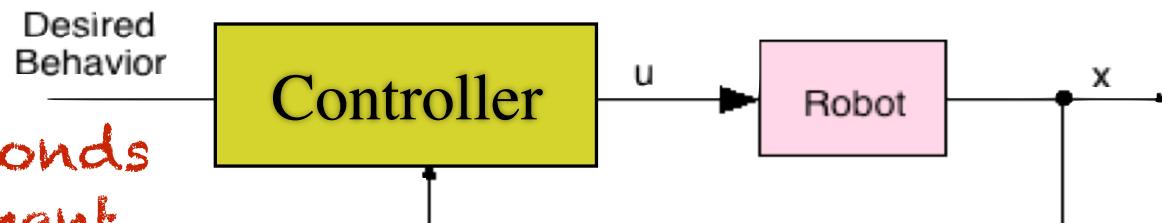
Acts without regard  
to environment

possible controller parameters

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t) = \pi(\alpha, t)$$

possible time dependence

## Closed Loop Control

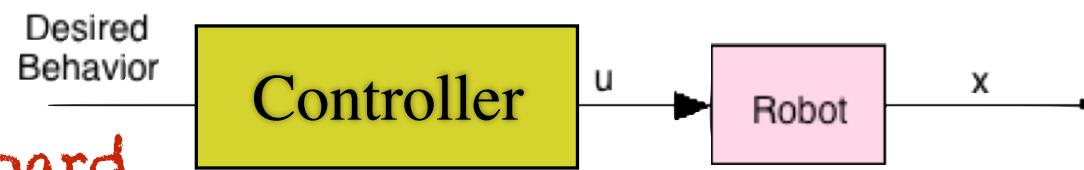


Control responds  
to environment

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$

# When to use open Loop vs closed Loop?

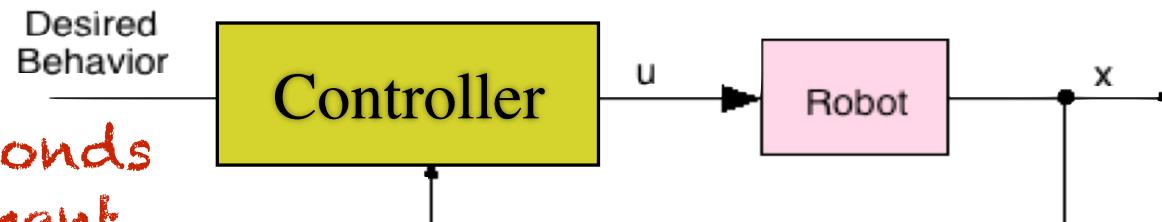
## Open Loop Control



Acts without regard  
to environment

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t) = \pi(\alpha, t)$$

## Closed Loop Control

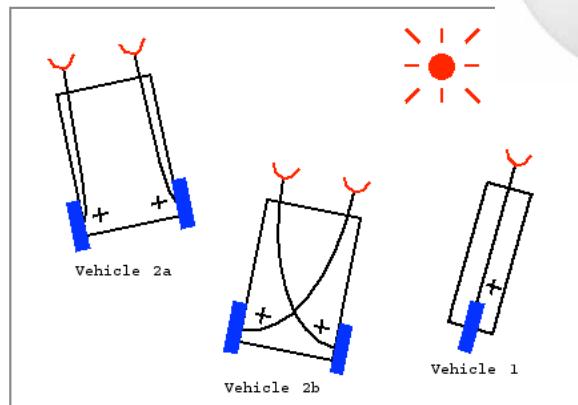


Control responds  
to environment

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$



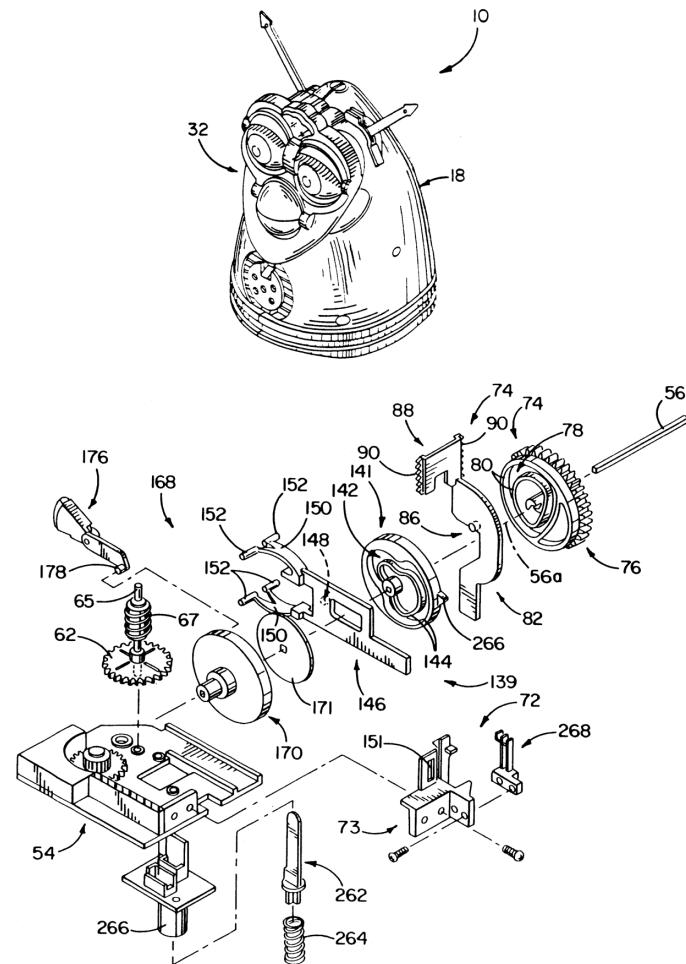
Which uses  
open Loop?





[www.youtube.com/user/silwolf](http://www.youtube.com/user/silwolf)

# Open loop: Furby



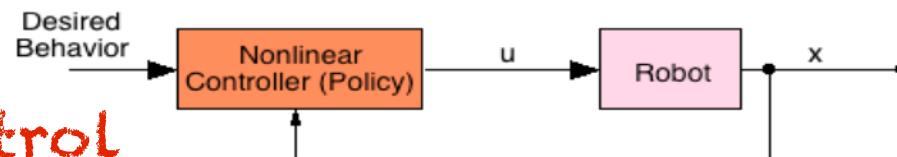
Hasbro



Disney

# Types of Feedback Control

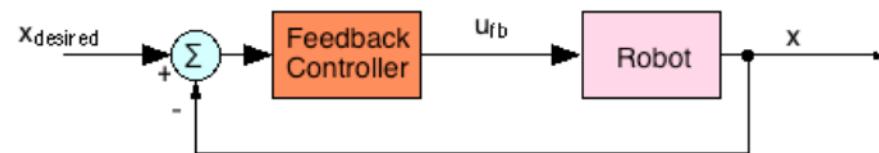
## Feedback Control



$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$

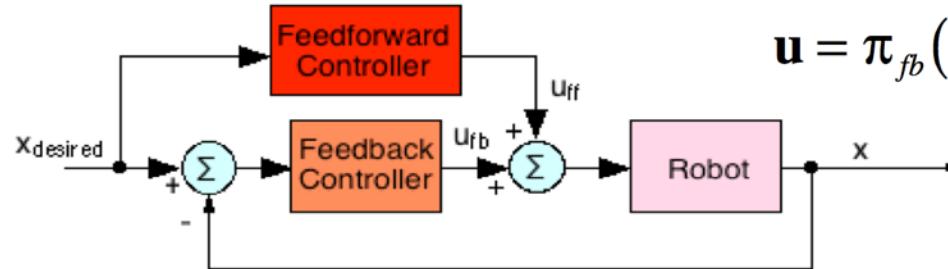
PID Control

## Negative Feedback Control



$$\mathbf{u} = \pi(\mathbf{x} - \mathbf{x}_{des}, \alpha, t)$$

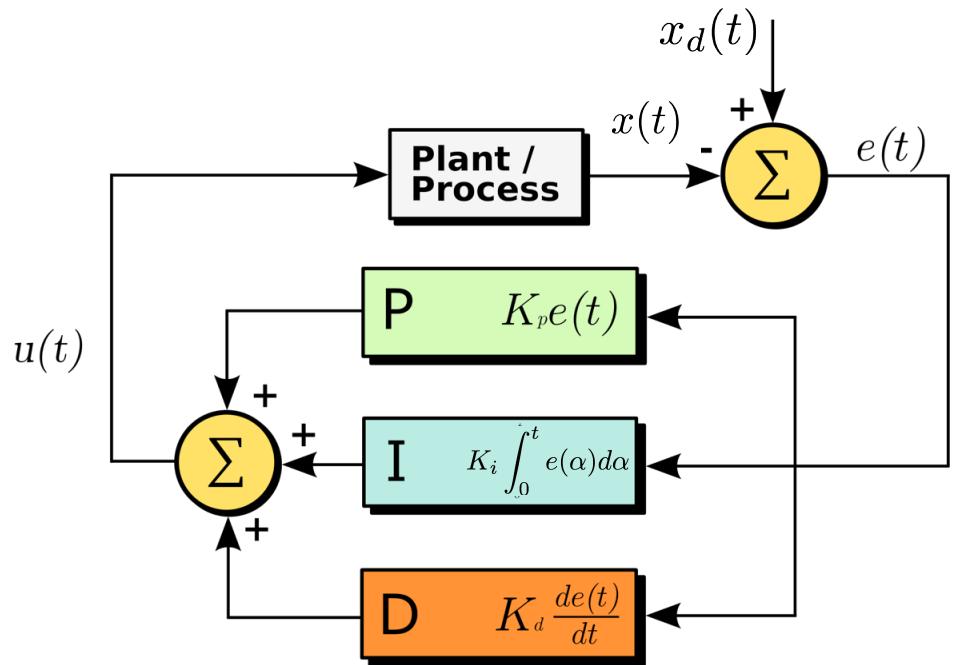
## Neg. Feedback & Feedforward Control



$$\mathbf{u} = \pi_{fb}(\mathbf{x} - \mathbf{x}_{des}, \alpha, t) + \pi_{ff}(\mathbf{x}_{des}, \alpha, t)$$

# PID Control

- Proportional-Integral-Derivative Control
- Sum of different responses to error
- Based on the mass spring and damper system
- Feedback correction based on the current error, past error, and predicted future error



# PID Control

for pendulum ( $x$  is  $\theta$ )

Error signal:

$$e(t) = x_{desired}(t) - x(t)$$

Control signal:

$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha) d\alpha + K_d \frac{d}{dt} e(t)$$

**P**  $K_p e(t)$

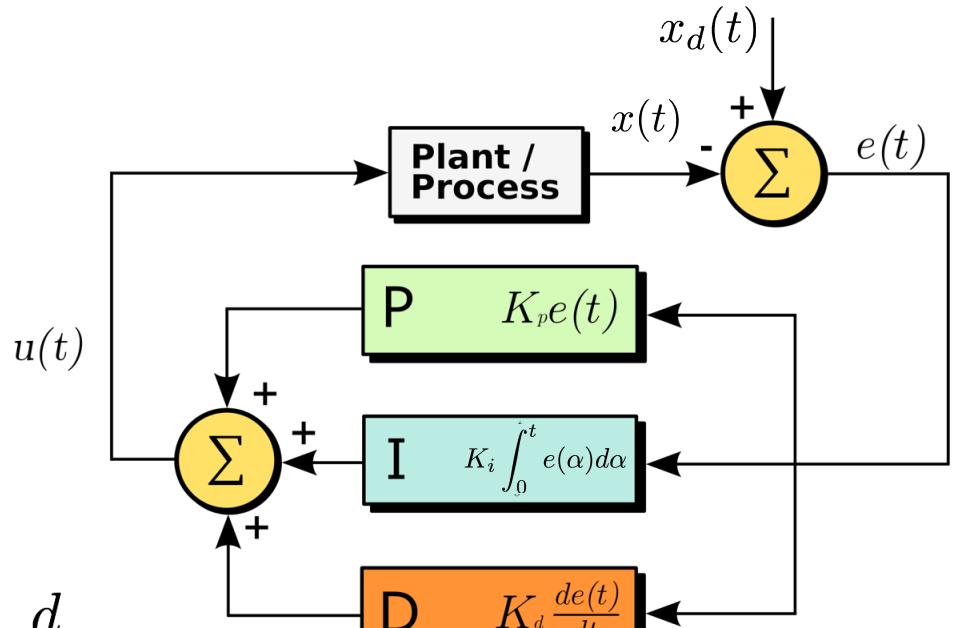
**I**  $K_i \int_0^t e(\alpha) d\alpha$

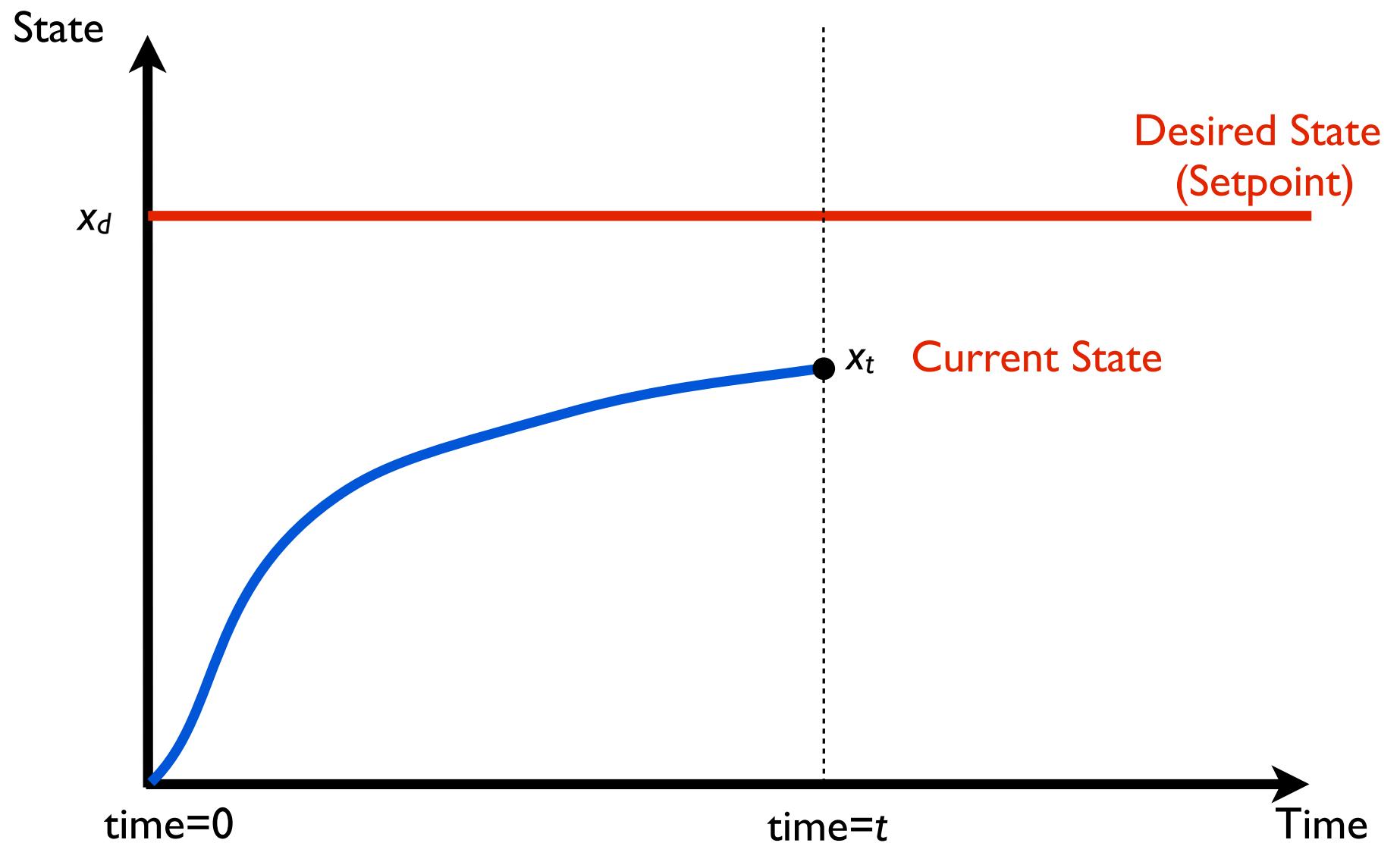
**D**  $K_d \frac{de(t)}{dt}$

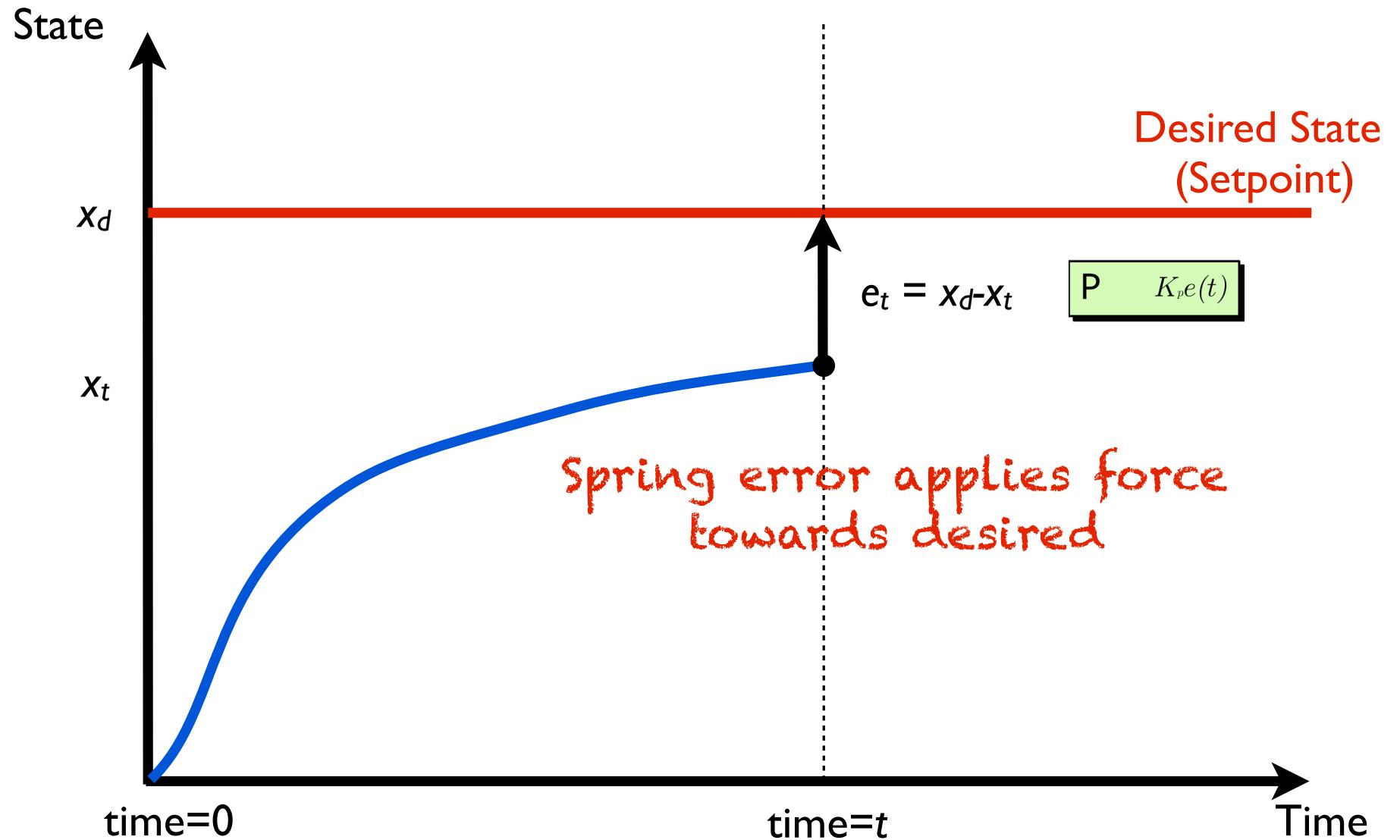
Current

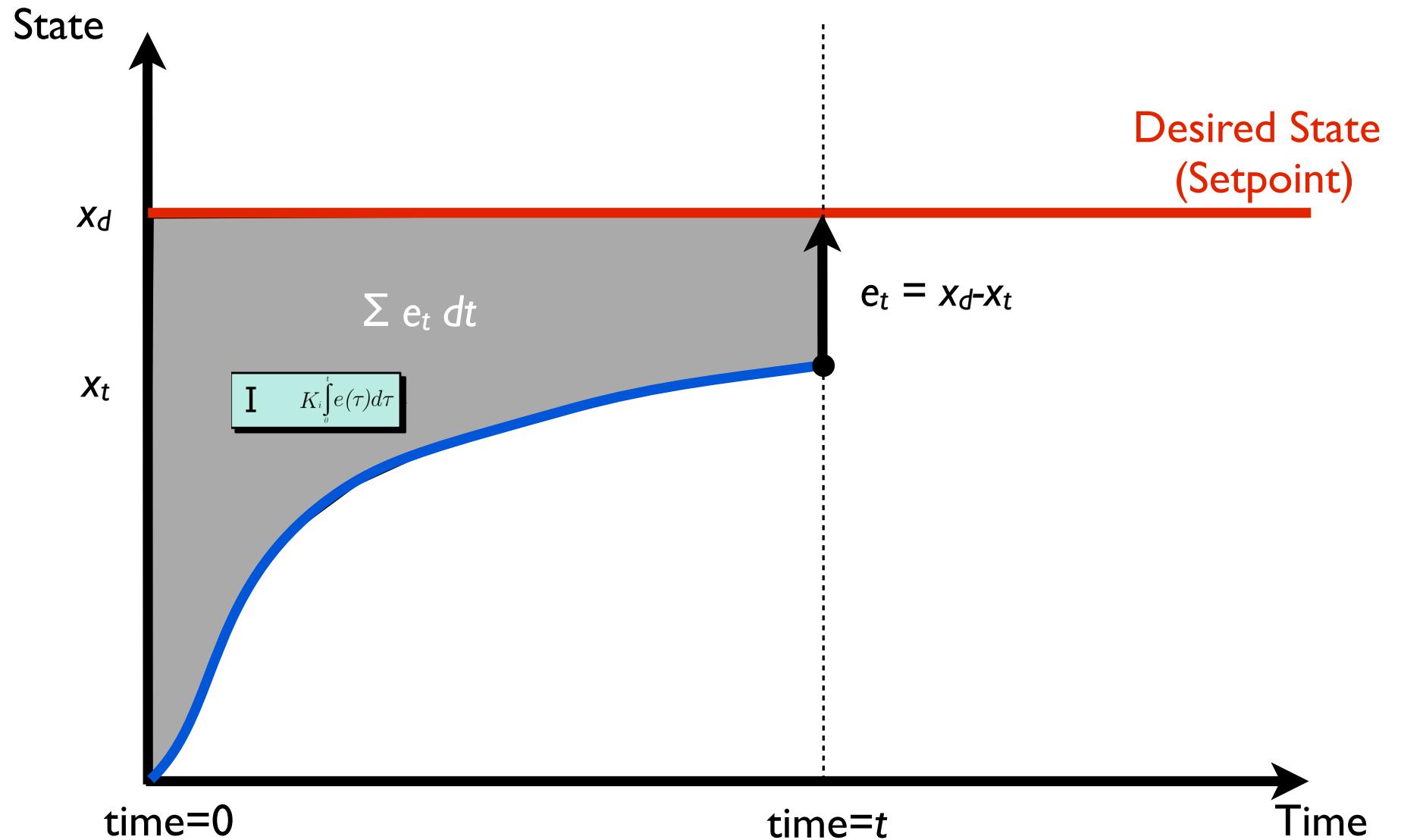
Past

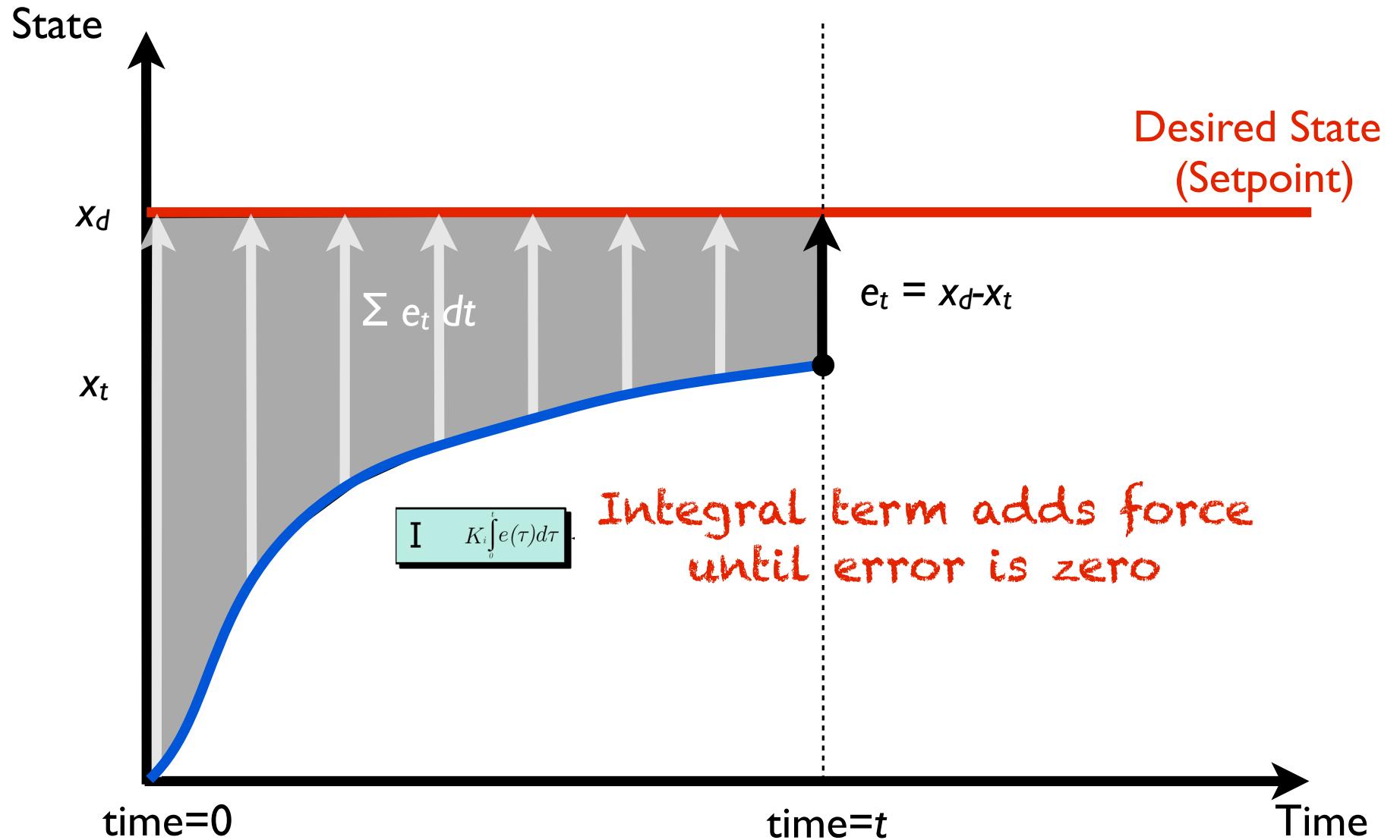
Future

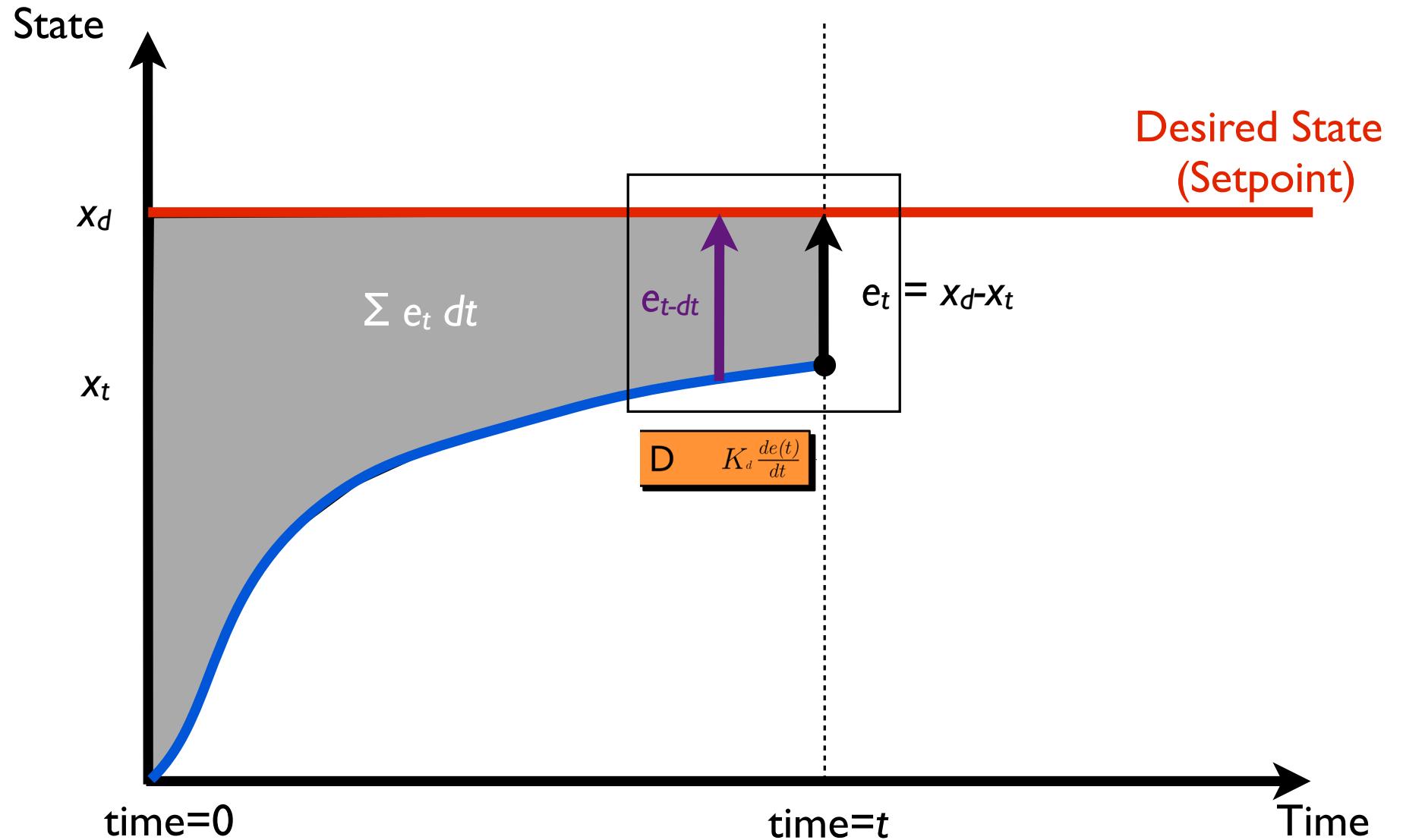


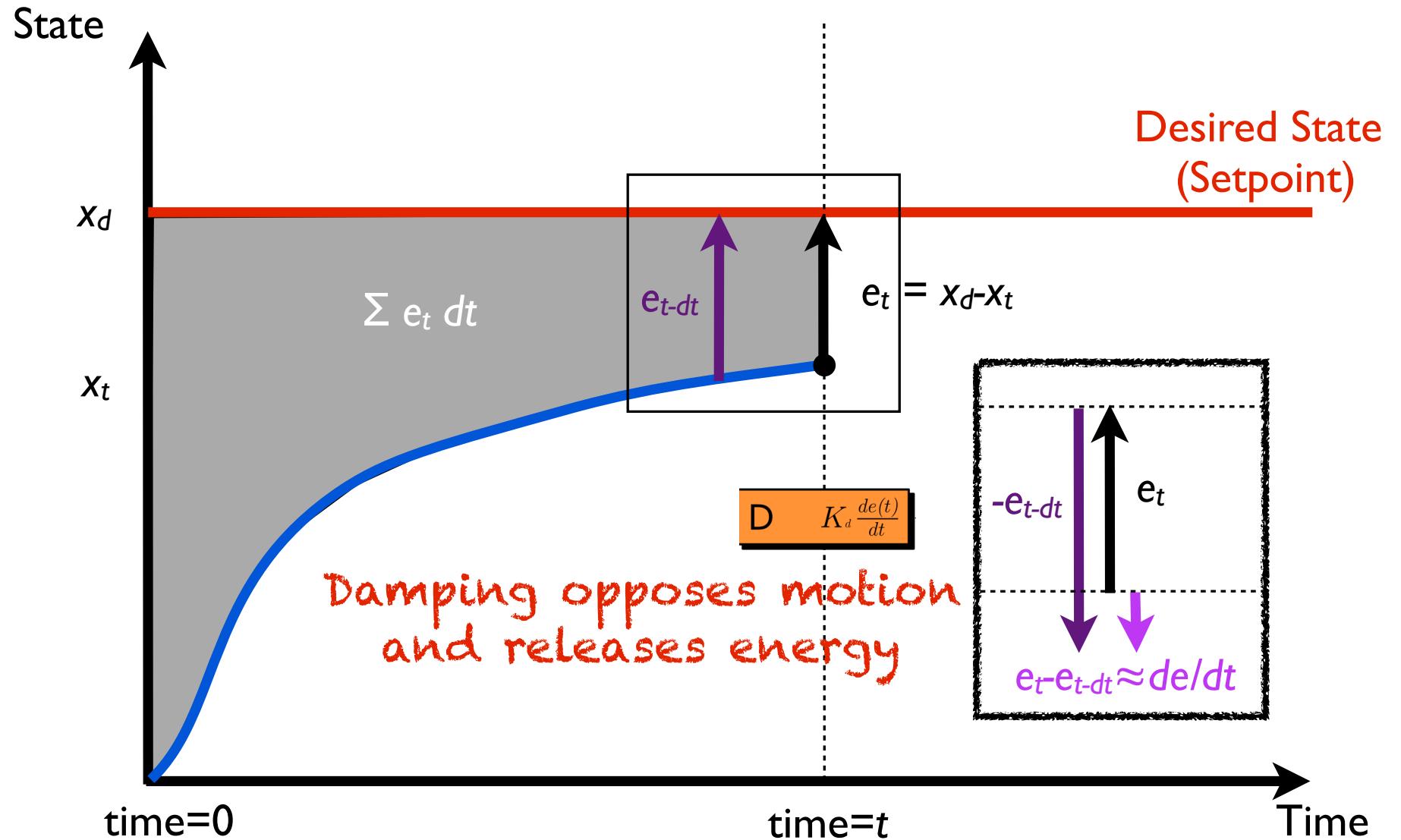


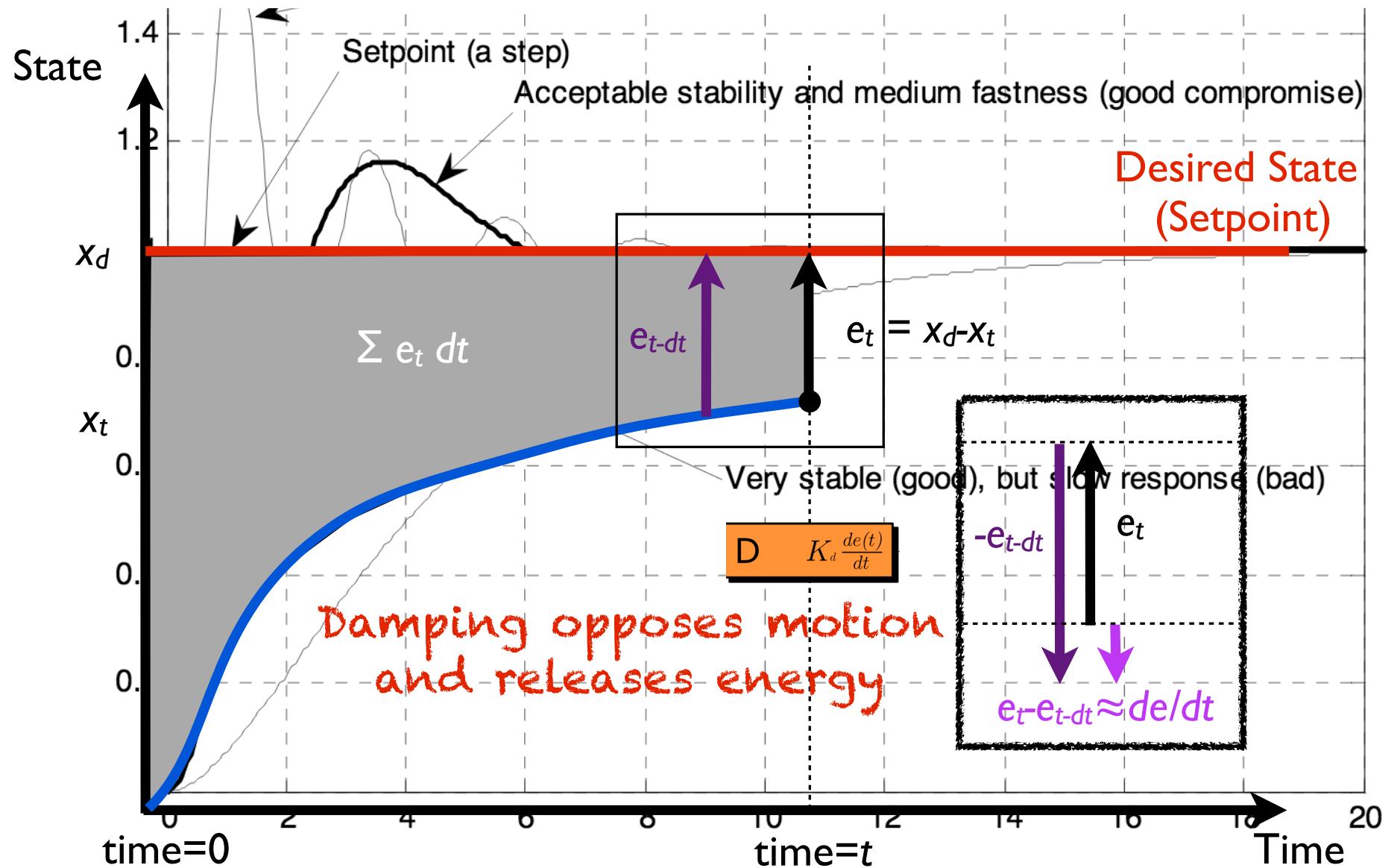




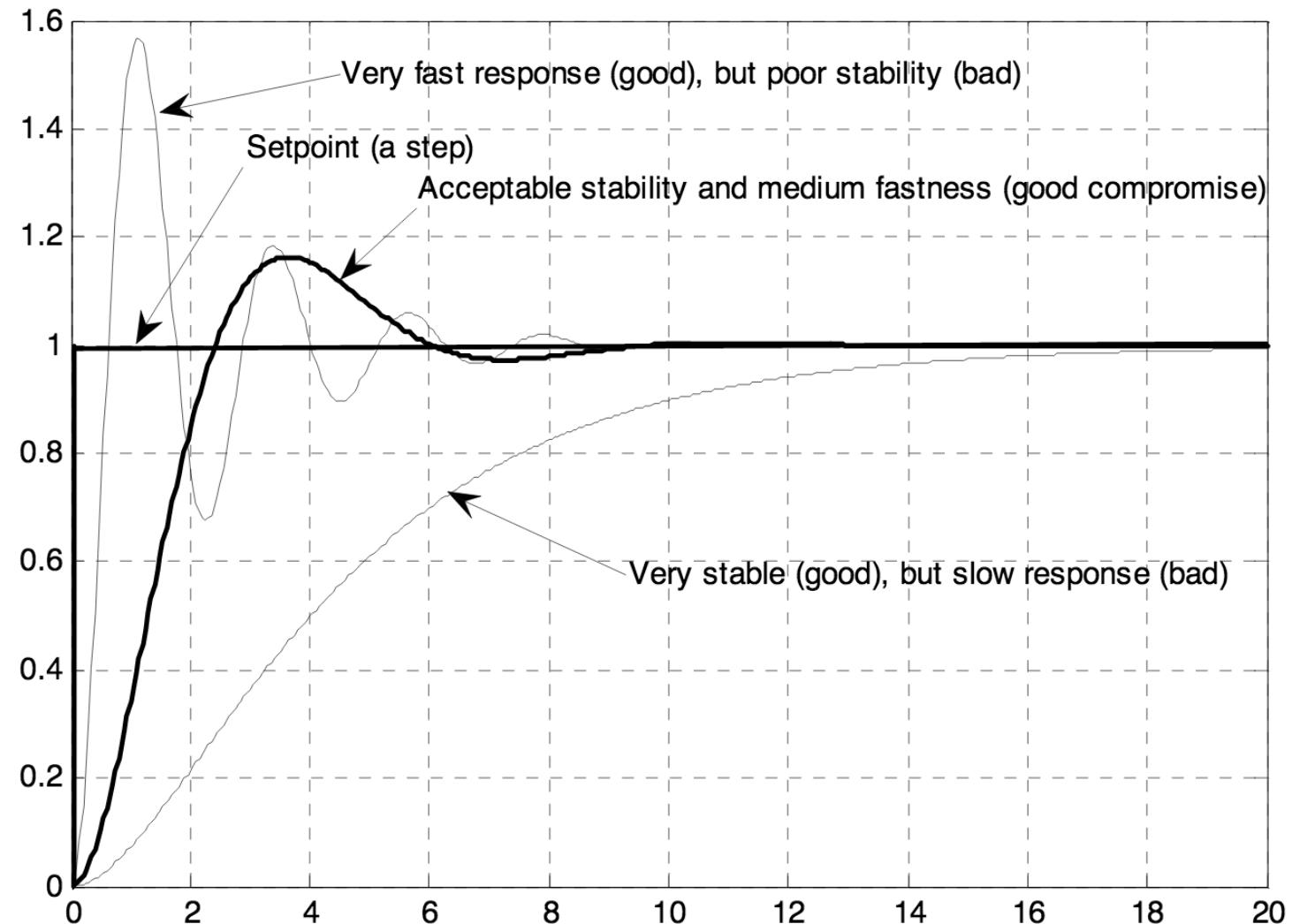








# PID Convergence



PID as a  
spring and damper model

# PID Control

Error signal:

$$e(t) = x_{desired}(t) - x(t)$$

Control signal:

$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha) d\alpha + K_d \frac{d}{dt} e(t)$$

**P**  $K_p e(t)$

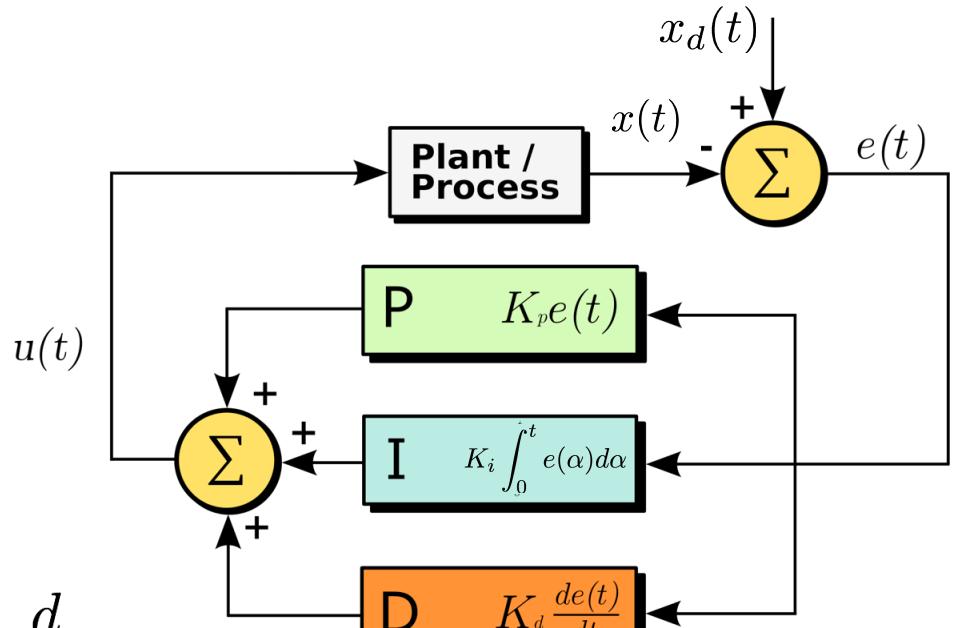
**I**  $K_i \int_0^t e(\alpha) d\alpha$

**D**  $K_d \frac{de(t)}{dt}$

Current

Past

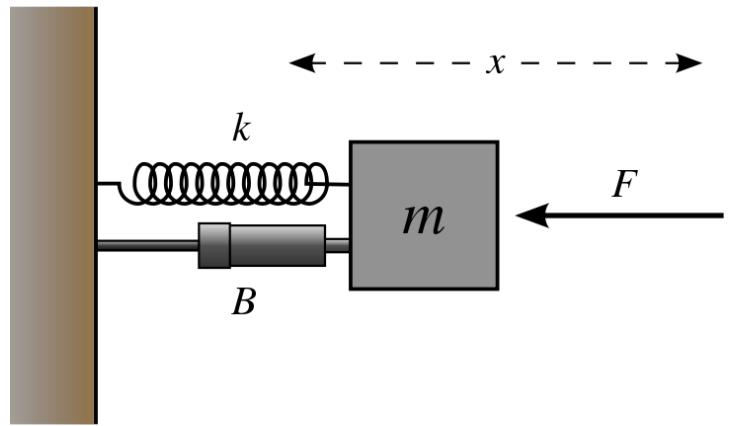
Future



# Hooke's Law

P  $K_p e(t)$

- Describes motion of mass spring damper system as



$$F = -kx$$

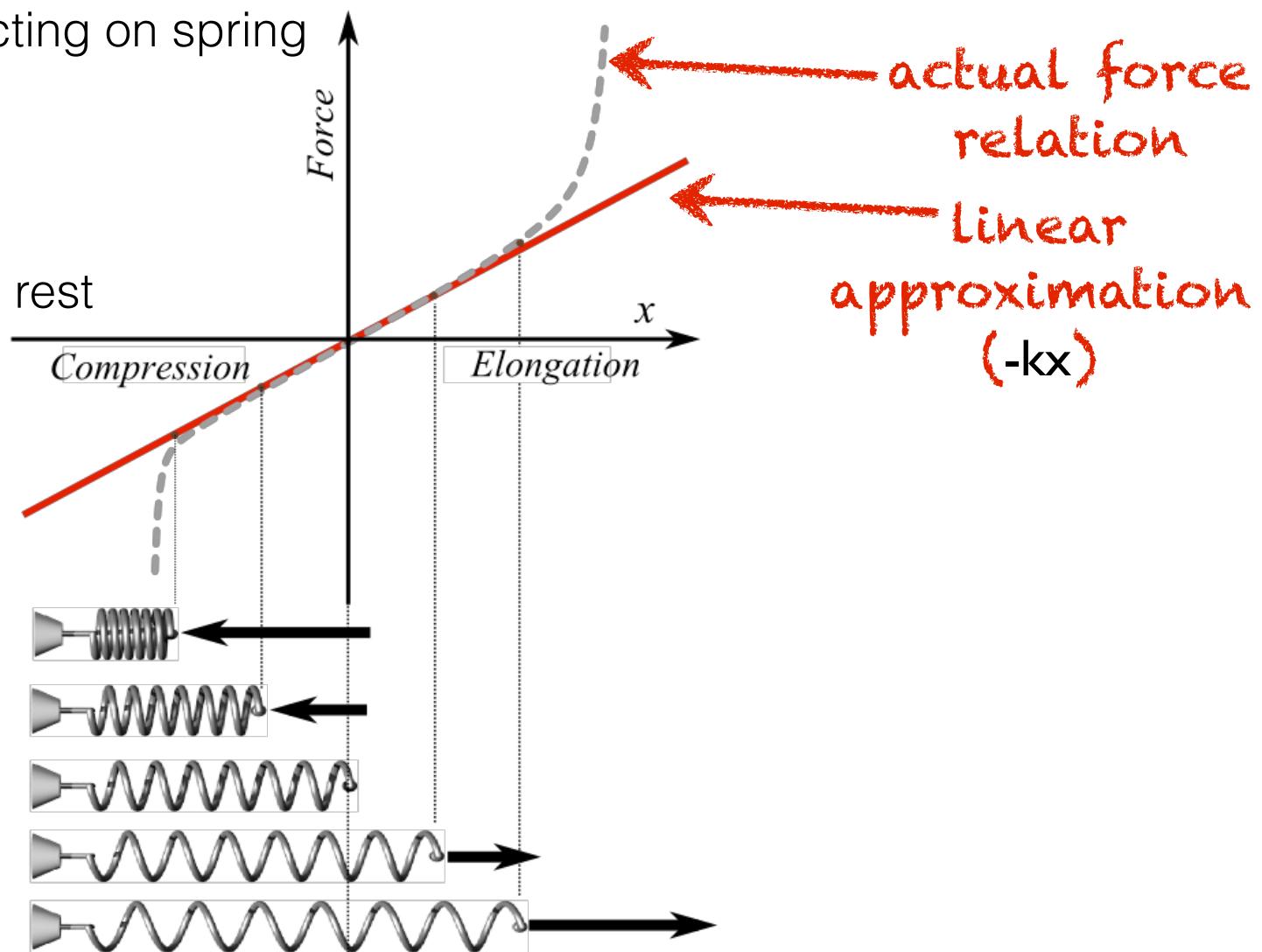
force moving  
spring towards rest

spring  
stiffness

distance from  
rest displacement

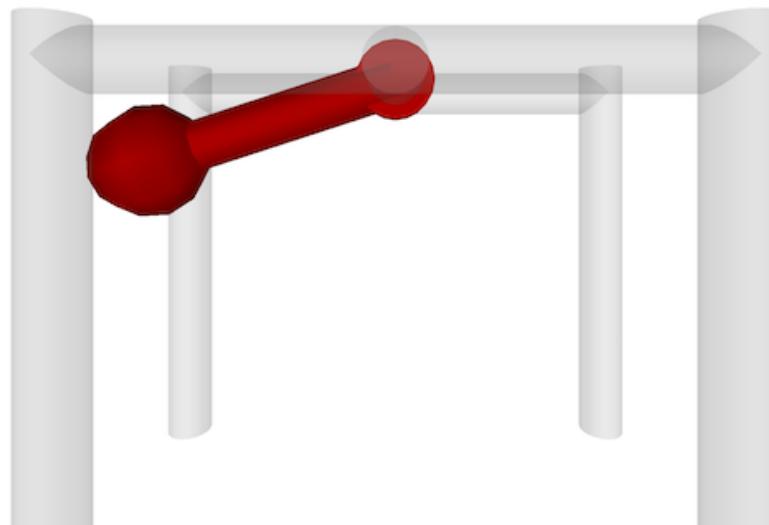
Vertical: Force acting on spring

Horizontal: Position from rest





# Let's see what happens



overstiff  
understiff

# PID Control

Error signal:

$$e(t) = x_{desired}(t) - x(t)$$

Control signal:

$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha) d\alpha + K_d \frac{d}{dt} e(t)$$

**P**  $K_p e(t)$

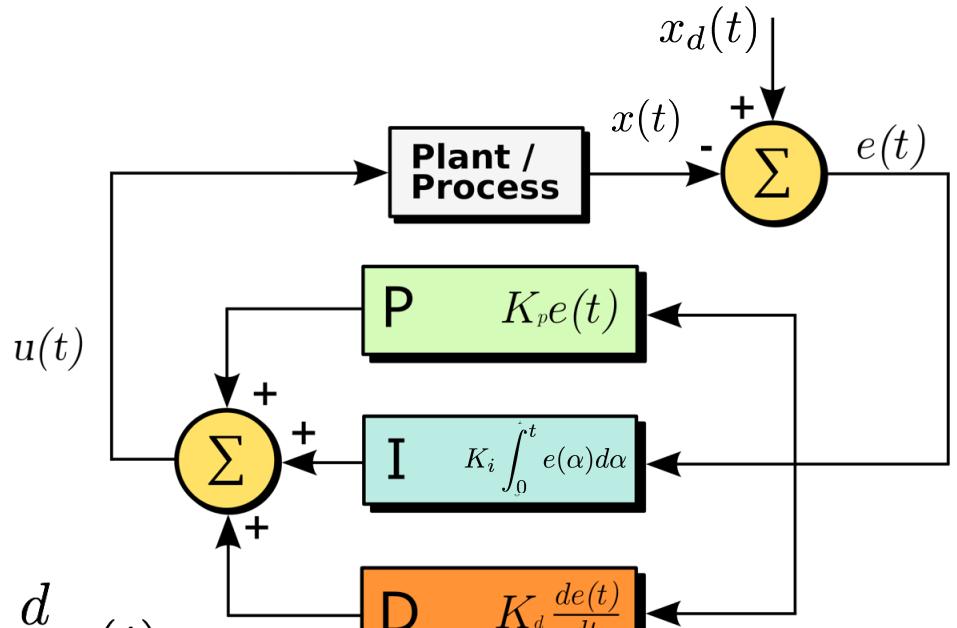
**I**  $K_i \int_0^t e(\alpha) d\alpha$

**D**  $K_d \frac{de(t)}{dt}$

Current

Past

Future

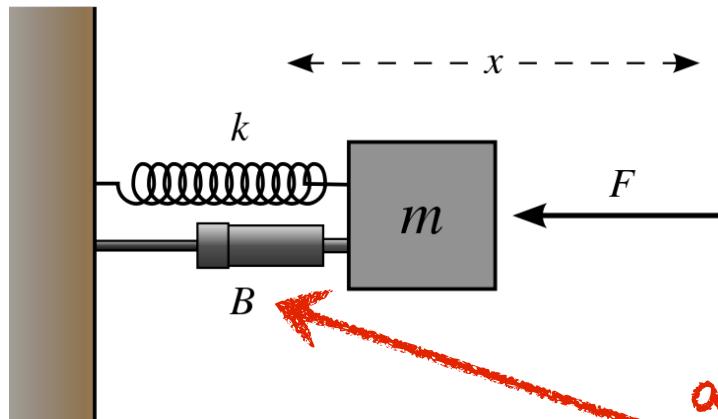


# Spring and Damper

$$P \quad K_p e(t)$$

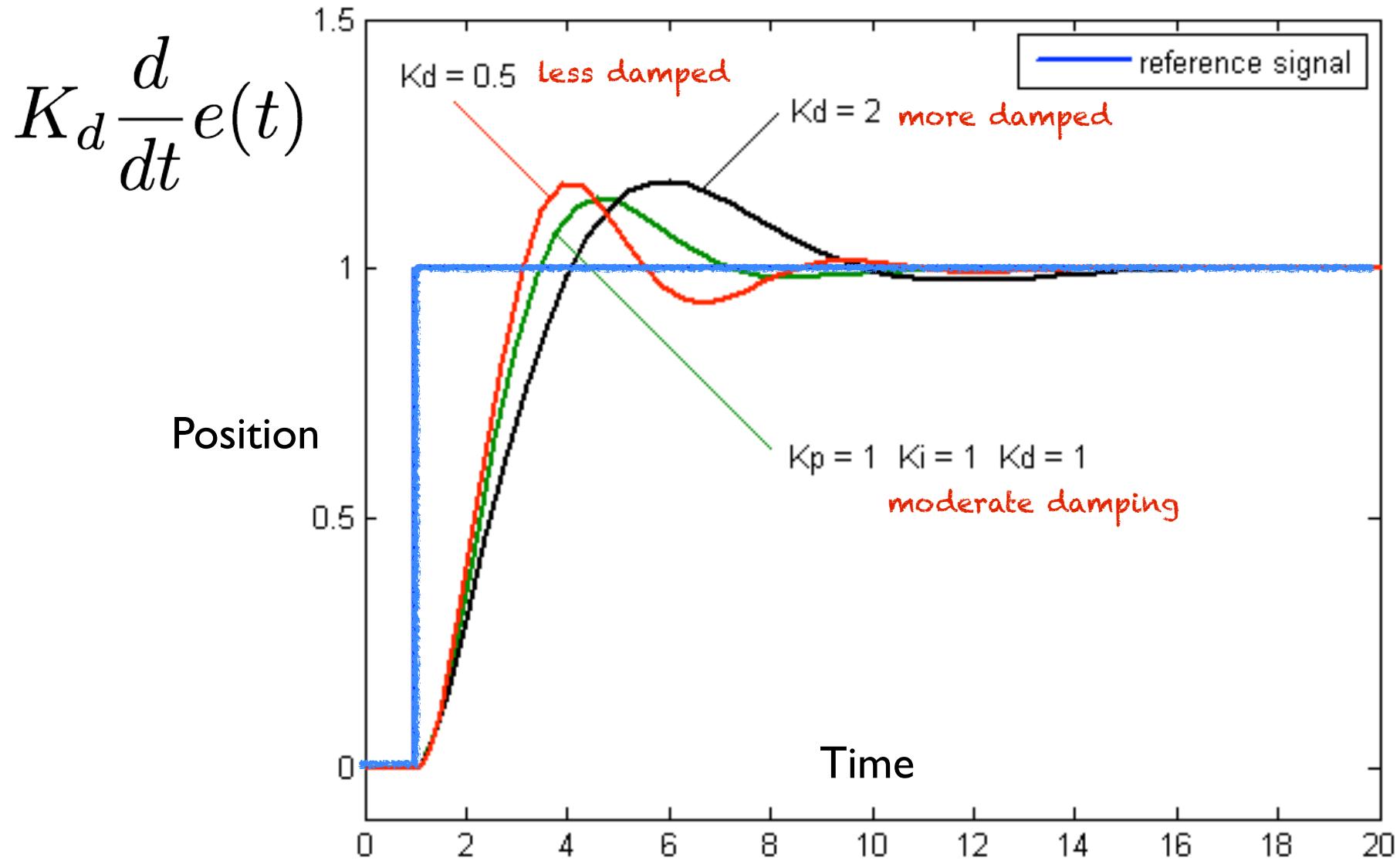
$$D \quad K_d \frac{de(t)}{dt}$$

$$F = -kx + -b\dot{x}$$

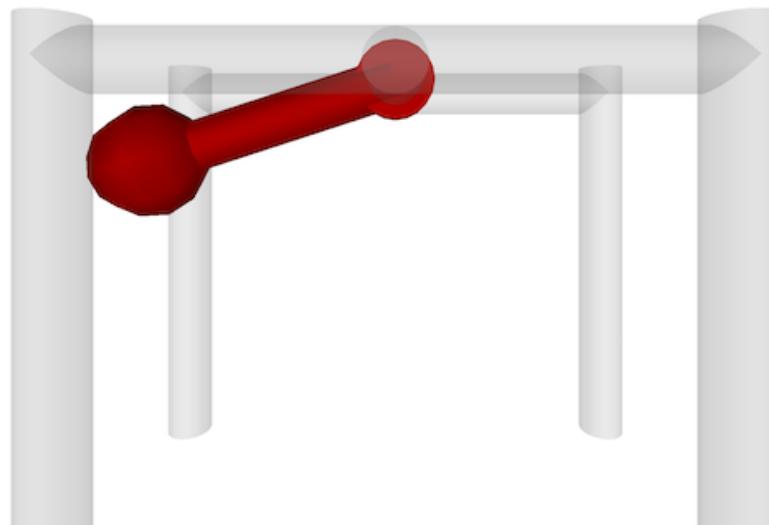


assuming constant set point,  
velocity is derivative of error

add damper to  
release energy



# Let's see what happens



steady state

# PID Control

Error signal:

$$e(t) = x_{desired}(t) - x(t)$$

Control signal:

$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha) d\alpha + K_d \frac{d}{dt} e(t)$$

**P**  $K_p e(t)$

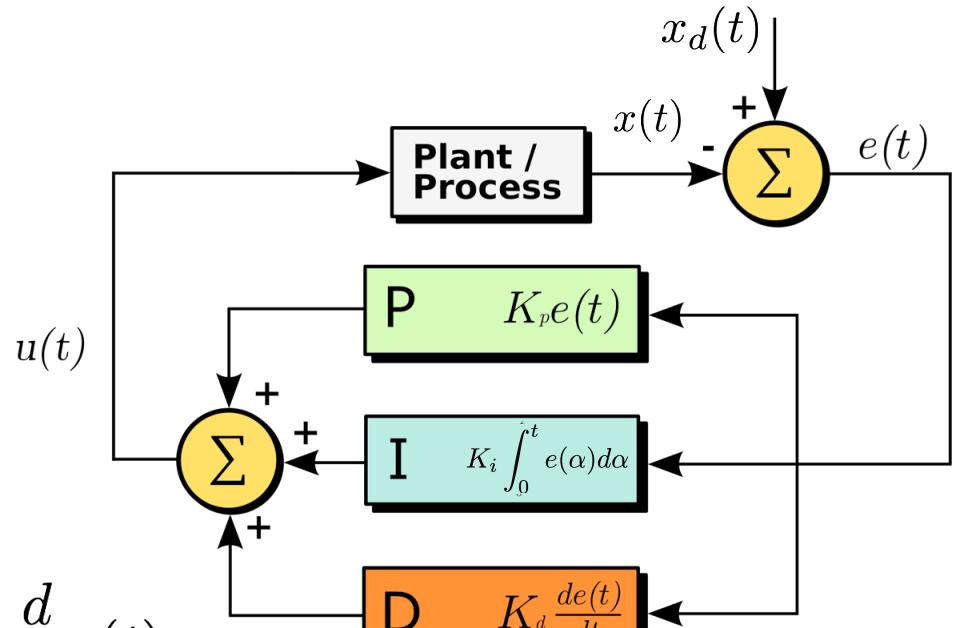
**I**  $K_i \int_0^t e(\alpha) d\alpha$

**D**  $K_d \frac{de(t)}{dt}$

Current

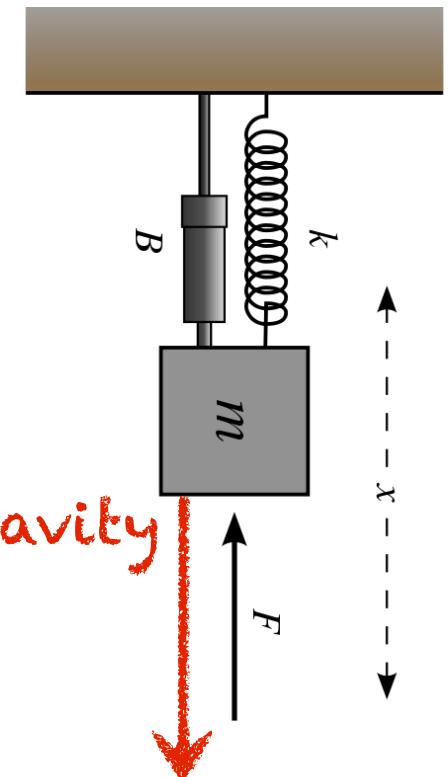
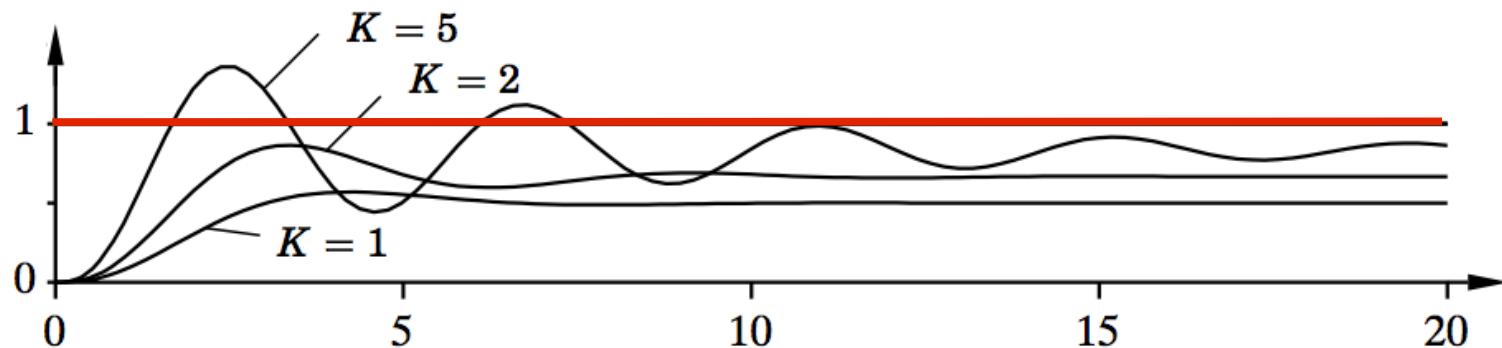
Past

Future



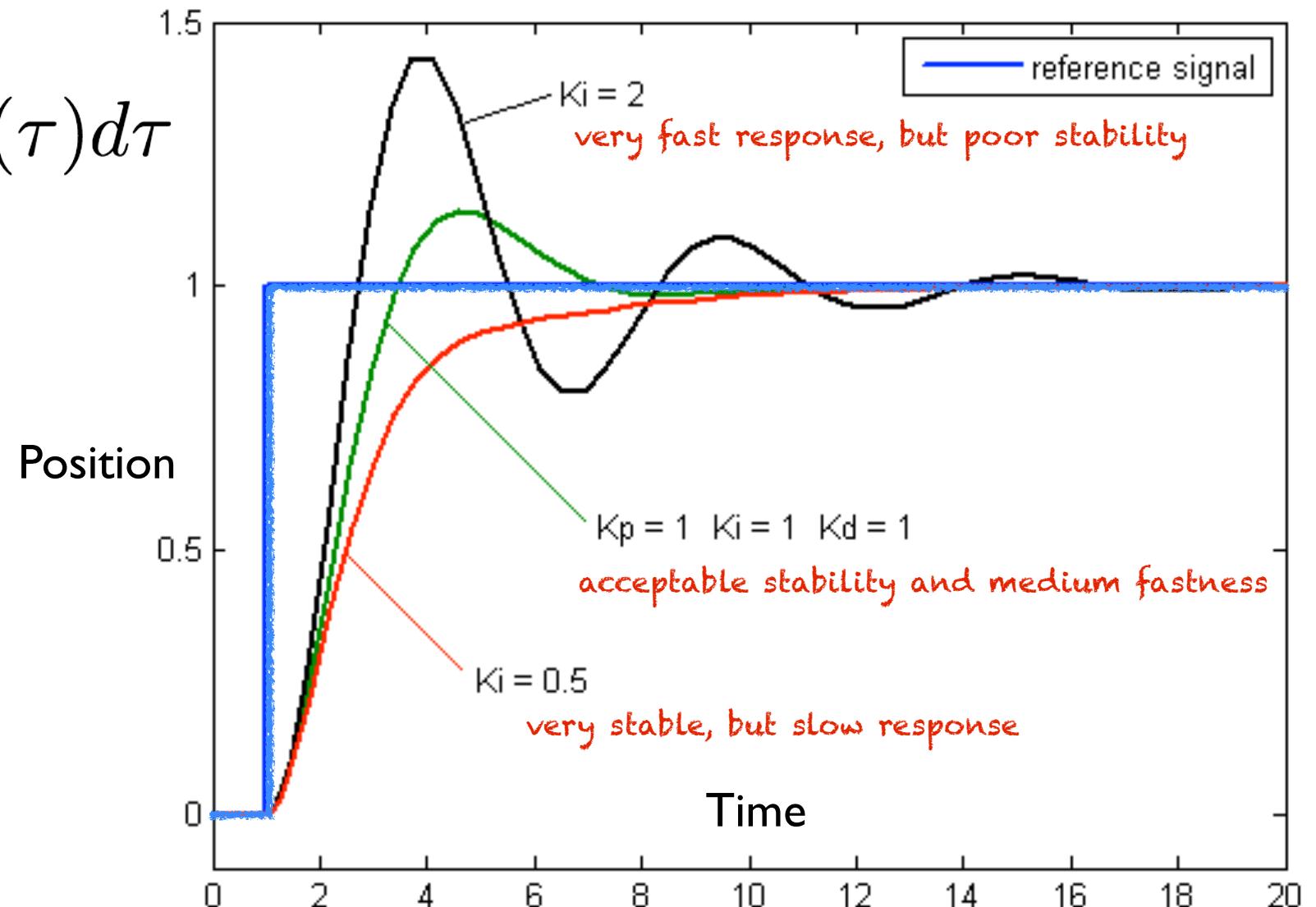
# Steady state error

- Steady state error occurs when the system rests at equilibrium before reaching desired state
- Cause could be an significant external force, weak motor, low proportional gain, etc.
- PID integral term compensates by accumulating and acting against error toward convergence

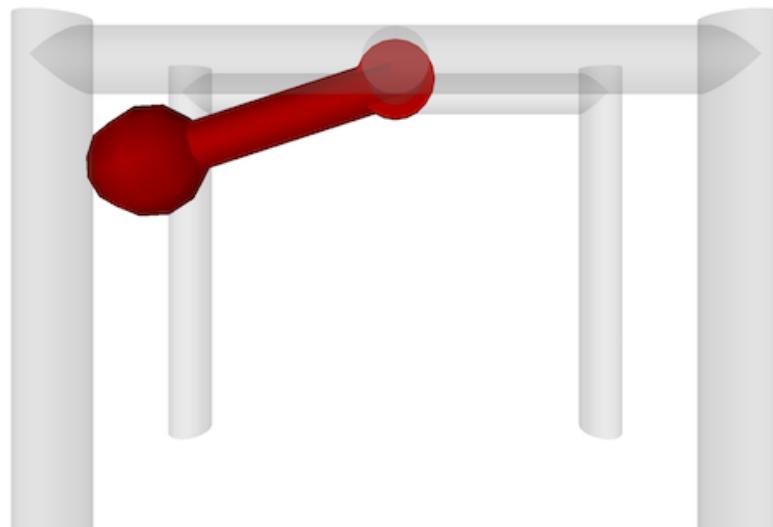


$$K_i \int_0^t e(\tau) d\tau$$

$$I = K_i \int_0^t e(\tau) d\tau$$



# Let's see what happens



pid

# Gain tuning

- Implementing PID algorithm will not necessarily produce a good controller
- Selection of the gains will greatly affect the performance of the controller
- PID gain tuning is more of an art than a science. Choose carefully.

$$u(t) = \boxed{K_p} e(t) + \boxed{K_i} \int_0^t e(\alpha) d\alpha + \boxed{K_d} \frac{d}{dt} e(t)$$

P       $K_p e(t)$

I       $K_i \int_0^t e(\alpha) d\alpha$

D       $K_d \frac{de(t)}{dt}$

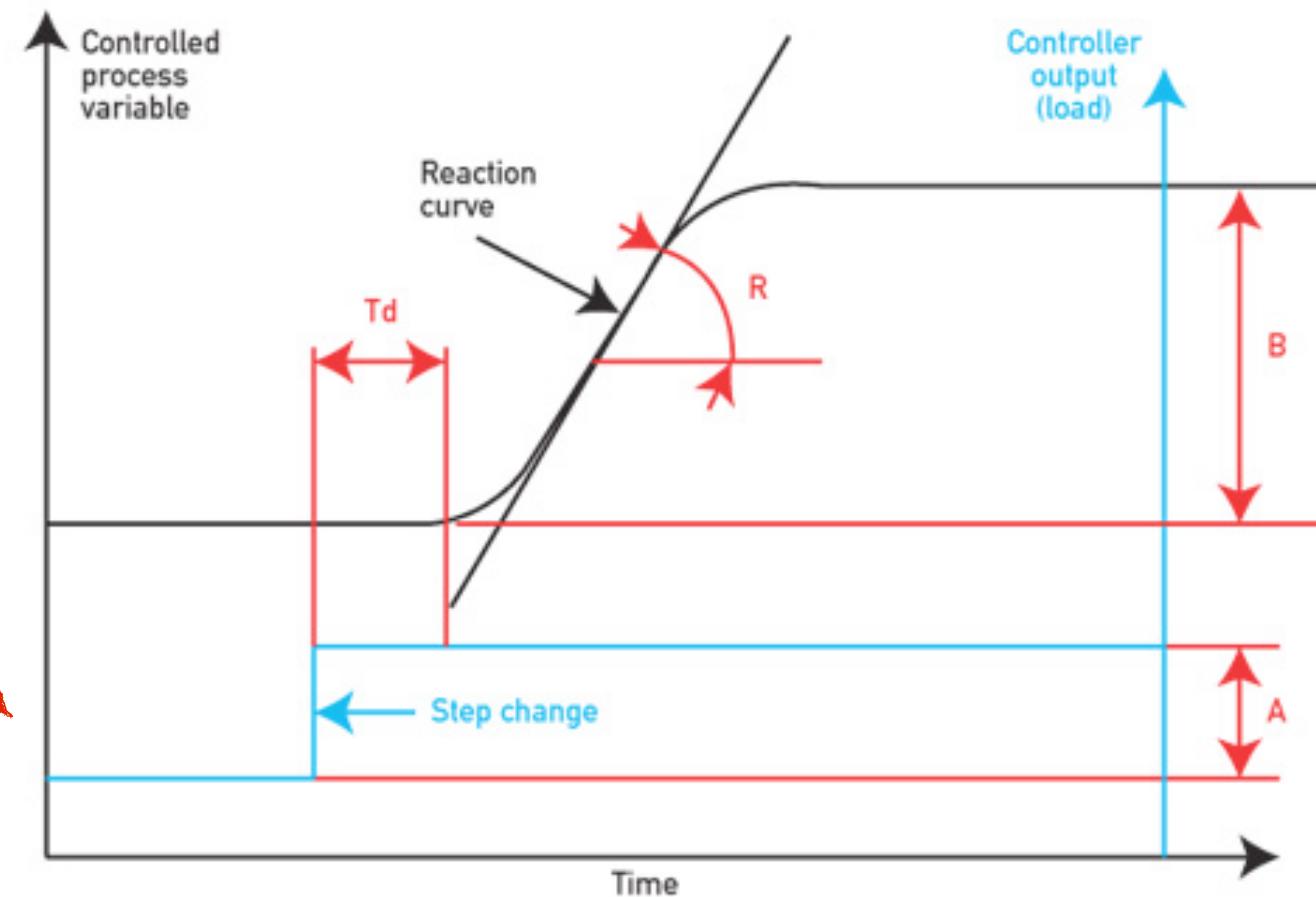
# PID controller

From Wikipedia, the free encyclopedia

## Effects of *increasing* a parameter independently<sup>[14]</sup>

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability <sup>[11]</sup>
$K_p$	Decrease	Increase	Small change	Decrease	Degrade
$K_i$	Decrease	Increase	Increase	Eliminate	Degrade
$K_d$	Minor change	Decrease	Decrease	No effect in theory	Improve if $K_d$ small

# Ziegler-Nichols PID tuning



# My approach to PID tuning

(take it or leave it)

- Start with all gains at zero :  $K_i = K_d = K_p = 0$
- Increase spring gain  $K_p$  until system roughly meets desired state
  - overshooting and oscillation about the desired state can be expected
- Increase damping gain  $K_d$  until the system is consistently stable
  - damping releases energy, but system will have steady state error
- Increase integral gain  $K_i$  until the system consistently reaches desired
- Refine gains as needed to improve performance; Test from different states

# Putting it all together (Pendulum)

desired pendulum state

$$x_d = [\theta_d, \dot{\theta}_d = 0]$$



Error

$$e_t = x_d - x_t$$

generate motor force  
towards desired angle

PID Controller

$$u_t = K_p e_t + K_i \sum_{\alpha=0}^t e_\alpha d\alpha + K_d \frac{d}{dt} e_t$$

$$u_t = \tau$$

Pendulum Dynamics

$$\ddot{\theta} = -\frac{g}{l} \sin(\theta) + \frac{\tau}{ml^2}$$

$$x_{t+dt}$$

feedback state  
for next iteration

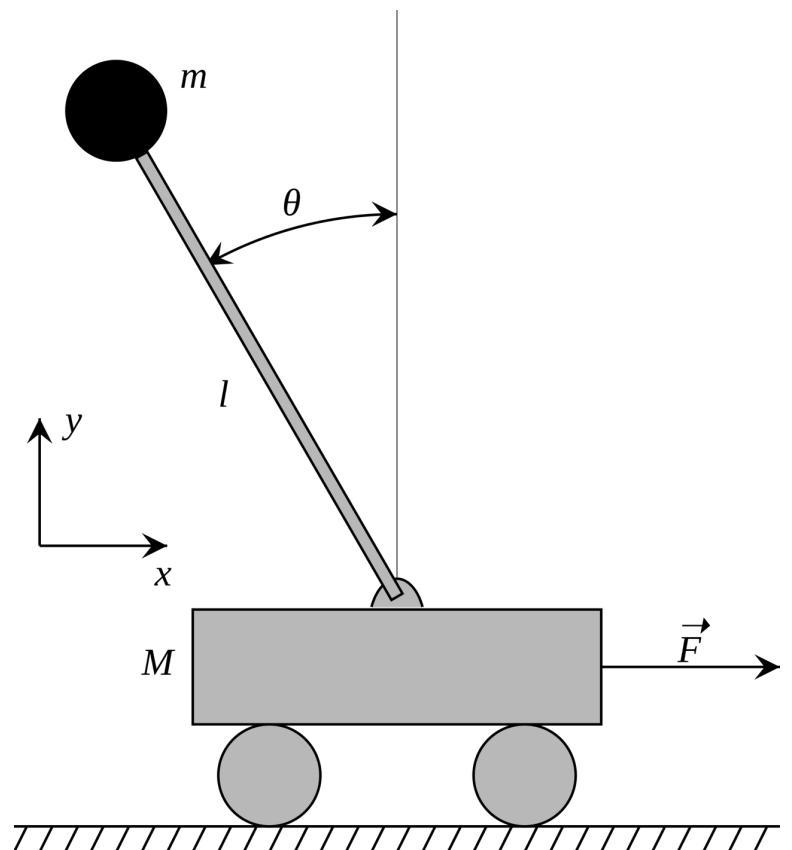
integrate over time to next state

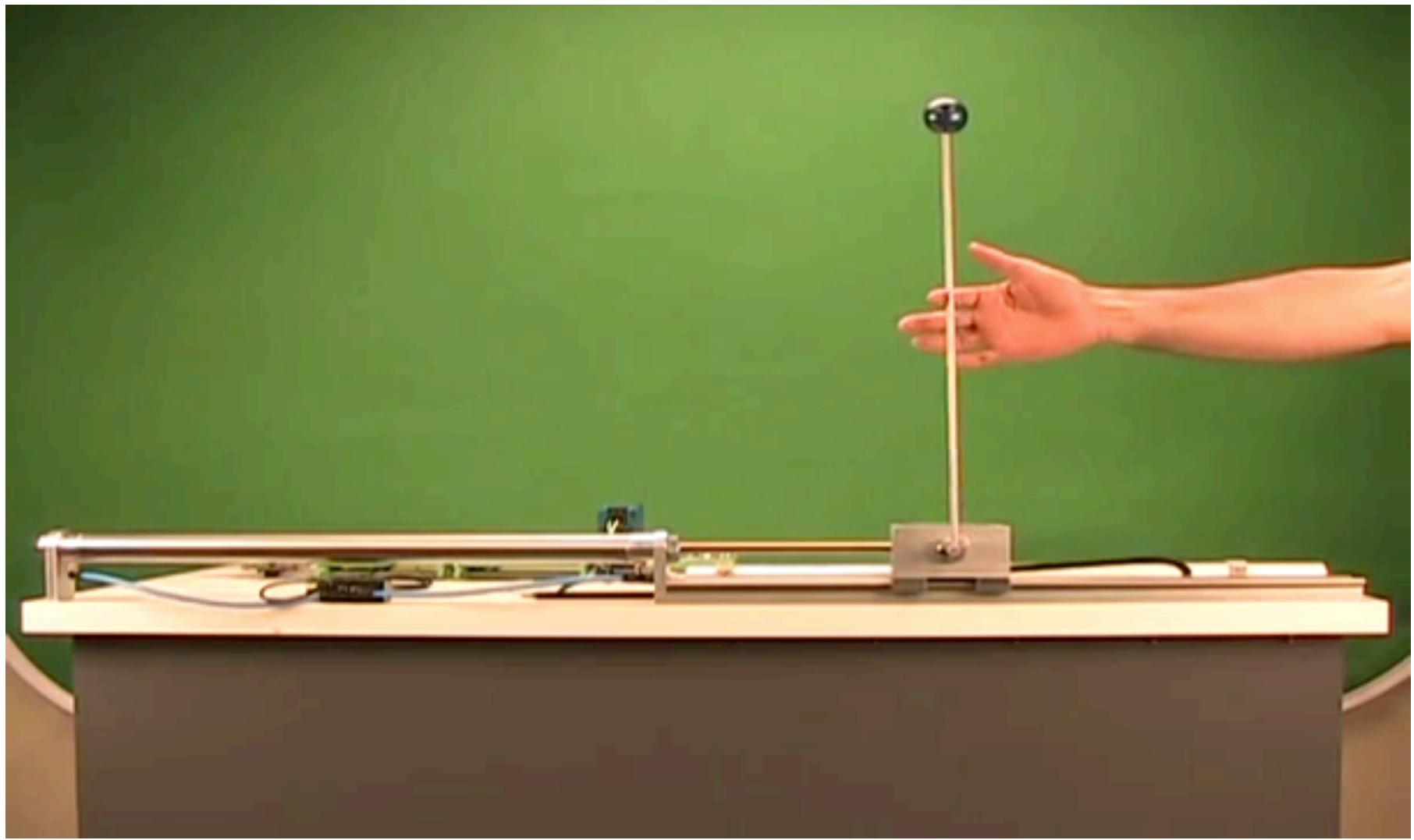
Other types of systems?

# Cart Pole Balancer

$$(M+m)\ddot{x} - m\ell\ddot{\theta}\cos\theta + m\ell\dot{\theta}^2 \sin\theta = F$$

$$\ell\ddot{\theta} - g \sin\theta = \ddot{x} \cos\theta$$

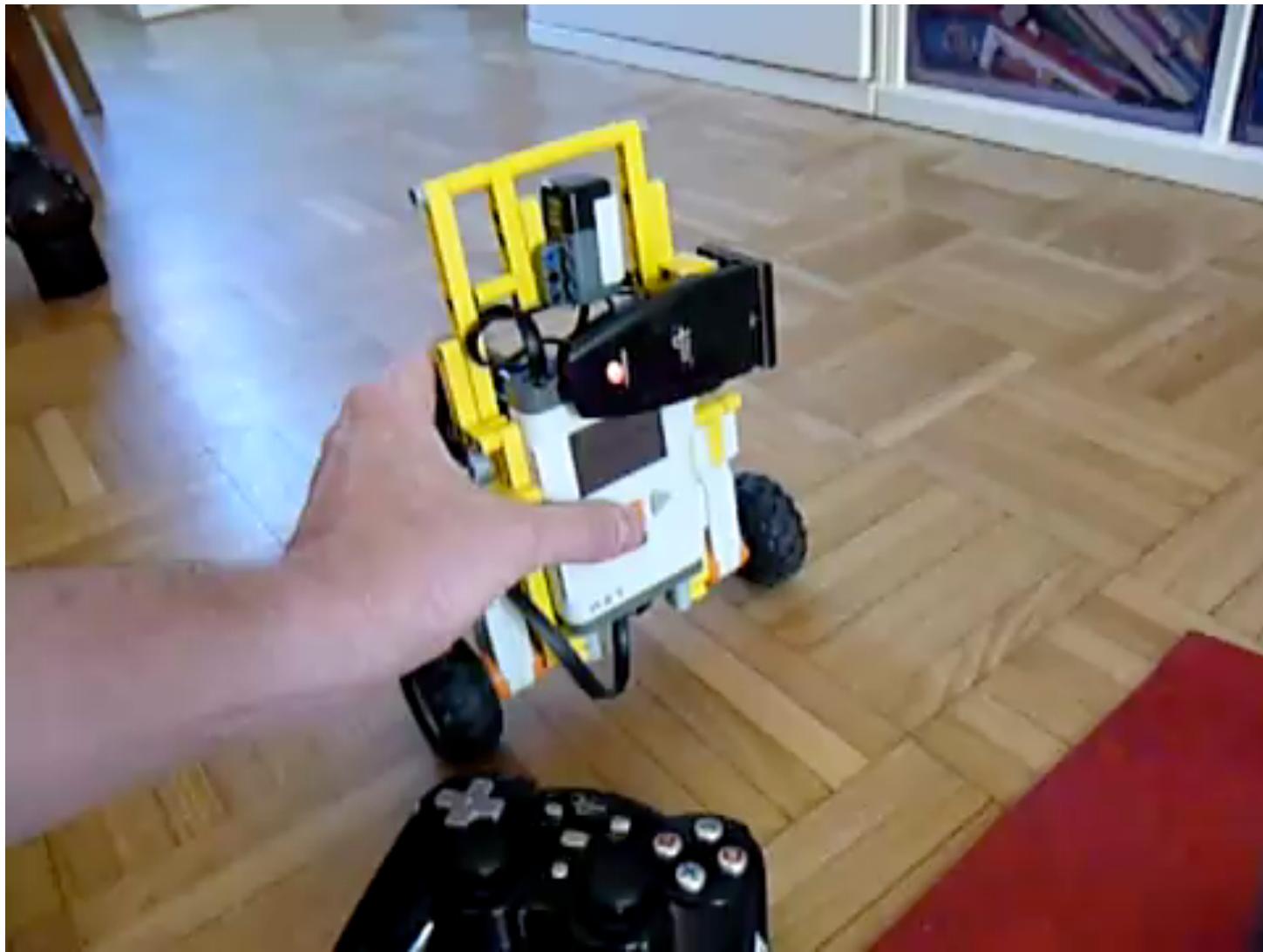




Enfield Technologies

# Differential Drive Inverted Pendulum





<https://www.youtube.com/watch?v=hpCwdu0e3i0>

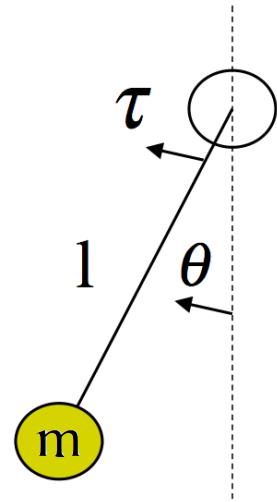
techbricks.nl

# Robot links are rigid bodies

## Pendulum as a particle

Mass assumed to be at a single point

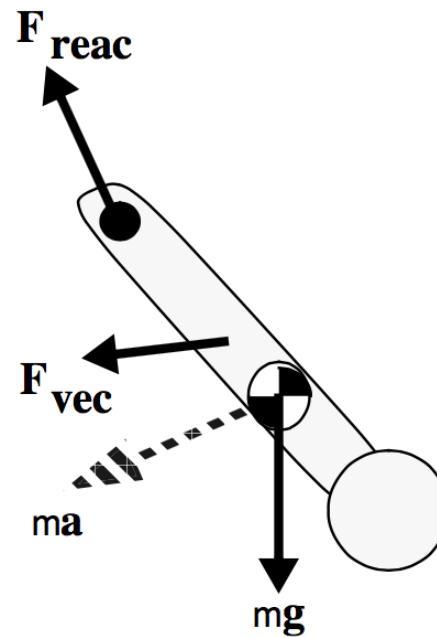
Only position of particle considered



## Pendulum as a rigid body

Mass distributed about the body's center of mass

Must account for position and orientation of body



SD/FAST (the original!) <http://support.ptc.com/support/sdfast/index.html>

# Newton-Euler Equations of Motion

- describe combined translational & rotational dynamics of a rigid body
- Newton's equation:  $F = m\ddot{x}$ 
  - expresses the force acting at the center of mass for accelerating body (note,  $\mathbf{x}$  is a 3 dimensional vector)
- Euler's equation:  $\tau = I_{cm}\dot{\omega} + \omega \times I_{cm}\omega$ 
  - expresses the torque acting on a rigid body given an angular velocity ( $\omega$ ), angular acceleration ( $\dot{\omega}$ ) given 3-by-3 inertia matrix

# Newton-Euler Equations of Motion

- For a single rigid body, Newton-Euler equations can be
  - expressed with respect to the body's center of mass

$$\begin{pmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{pmatrix} = \begin{pmatrix} m\mathbf{I}_3 & 0 \\ 0 & \mathbf{I}_{cm} \end{pmatrix} \begin{pmatrix} \mathbf{a}_{cm} \\ \boldsymbol{\alpha} \end{pmatrix} + \begin{pmatrix} 0 \\ \boldsymbol{\omega} \times \mathbf{I}_{cm} \boldsymbol{\omega} \end{pmatrix}$$

- and with respect to an arbitrary frame of reference

$$\begin{pmatrix} \mathbf{F} \\ \boldsymbol{\tau}_p \end{pmatrix} = \begin{pmatrix} m\mathbf{I}_3 & -m[\mathbf{c}]^\times \\ m[\mathbf{c}]^\times & \mathbf{I}_{cm} - m[\mathbf{c}]^\times [\mathbf{c}]^\times \end{pmatrix} \begin{pmatrix} \mathbf{a}_p \\ \boldsymbol{\alpha} \end{pmatrix} + \begin{pmatrix} m[\boldsymbol{\omega}]^\times [\boldsymbol{\omega}]^\times \mathbf{c} \\ [\boldsymbol{\omega}]^\times (\mathbf{I}_{cm} - m[\mathbf{c}]^\times [\mathbf{c}]^\times) \boldsymbol{\omega} \end{pmatrix}$$

# Newton-Euler Equations of Motion

- For a single rigid body, Newton-Euler equations can be
  - expressed with respect to the body's center of mass

$$\begin{pmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{pmatrix} = \begin{pmatrix} m\mathbf{I}_3 & 0 \\ 0 & \mathbf{I}_{cm} \end{pmatrix} \begin{pmatrix} \mathbf{a}_{cm} \\ \boldsymbol{\alpha} \end{pmatrix} + \begin{pmatrix} 0 \\ \boldsymbol{\omega} \times \mathbf{I}_{cm} \boldsymbol{\omega} \end{pmatrix}$$

- and with respect to an arbitrary frame of reference

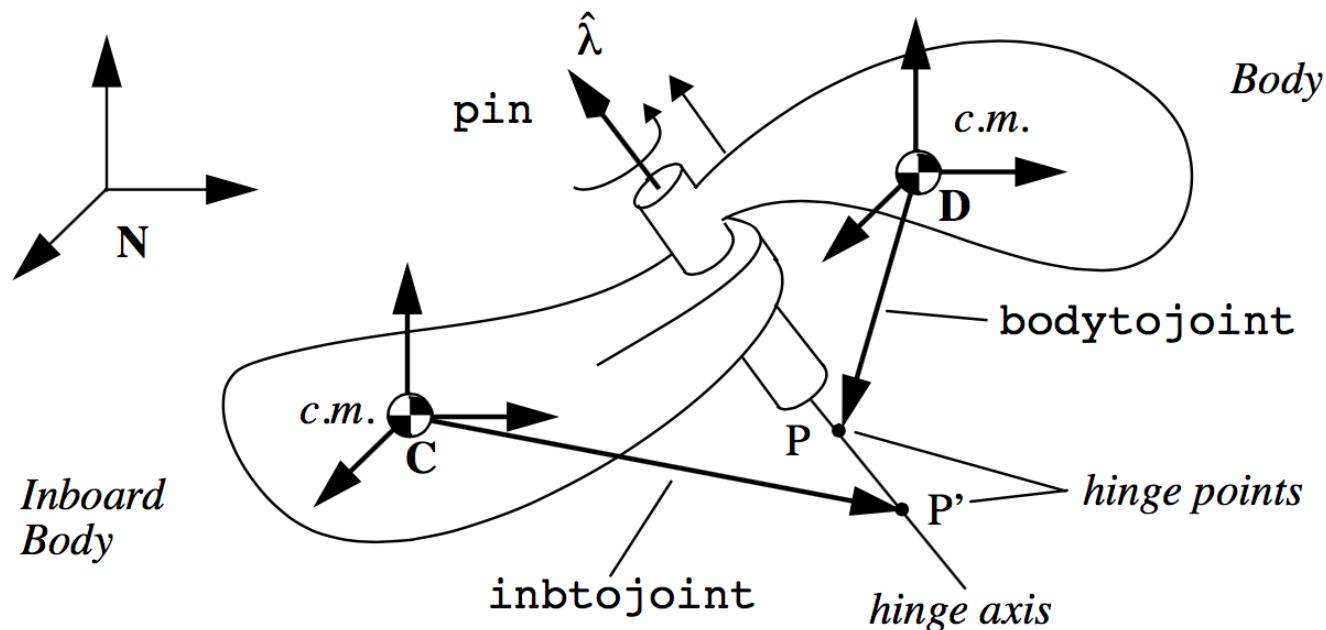
$$\begin{pmatrix} \mathbf{F} \\ \boldsymbol{\tau}_p \end{pmatrix} = \begin{pmatrix} m\mathbf{I}_3 & -m[\mathbf{c}]^\times \\ m[\mathbf{c}]^\times & \mathbf{I}_{cm} - m[\mathbf{c}]^\times [\mathbf{c}]^\times \end{pmatrix} \begin{pmatrix} \mathbf{a}_p \\ \boldsymbol{\alpha} \end{pmatrix} + \begin{pmatrix} m[\boldsymbol{\omega}]^\times [\boldsymbol{\omega}]^\times \mathbf{c} \\ [\boldsymbol{\omega}]^\times (\mathbf{I}_{cm} - m[\mathbf{c}]^\times [\mathbf{c}]^\times) \boldsymbol{\omega} \end{pmatrix}$$

Cross product by skew-symmetric matrix

$$[\boldsymbol{\omega}]^\times \equiv \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

# Joints between rigid bodies

Assembled Cylinder Joint (1-D Translational Plus Coaxial 1-D Rotational Joint)



SD/FAST (the original!) <http://support.ptc.com/support/sdfast/index.html>

# General Equations of Motion

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})^T \mathbf{g}$$

- Dynamics for rigid body systems can be expressed generally in configuration with respect to generalized coordinates
- Configuration expressed as a vector of joint state variables

$$\mathbf{q} = \{q_1, q_2, \dots, q_N\}$$

$$q_i = \begin{cases} \theta_i, & \text{if joint } i \text{ is revolute} \\ d_i, & \text{if joint } i \text{ is prismatic} \end{cases}$$

# General Equations of Motion

$$\boxed{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})^T \mathbf{g}$$



Vector of motor forces  
(N-dimensional for all DOFs)

# General Equations of Motion

$$\tau = \boxed{\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})^T \mathbf{g}$$

↑  
Vector of motor forces  
(N-dimensional for all DOFs)

“mass times acceleration”

M: Inertia matrix

(NxN positive definite)

• •  
q,  $\dot{q}$ ,  $\ddot{q}$ : state/velocity/acceleration  
in generalized coordinates  
(each N-dimensional vectors)

# General Equations of Motion

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \boxed{\mathbf{G}(\mathbf{q})} + \boxed{\mathbf{F}(\dot{\mathbf{q}})} + \boxed{\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}} + \mathbf{J}(\mathbf{q})^T \mathbf{g}$$



"mass times acceleration"

$\mathbf{M}$ : Inertia matrix

( $N \times N$  positive definite)

$\mathbf{q}$ ,  $\dot{\mathbf{q}}$ ,  $\ddot{\mathbf{q}}$ : state/velocity/acceleration  
in generalized coordinates  
(each  $N$ -dimensional vectors)

gravitational  
forces      frictional  
forces      Centripetal and  
Coriolis forces

Be careful with Centripetal force  
<https://youtu.be/m0bSJkrDYH8>



# General Equations of Motion

$$\tau = M(\mathbf{q})\ddot{\mathbf{q}} + G(\mathbf{q}) + F(\dot{\mathbf{q}}) + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + J(\mathbf{q})^T g$$

Vector of motor forces  
(N-dimensional for all DOFs)

“mass times acceleration”

M: Inertia matrix

(NxN positive definite)

• •  
q,  $\dot{q}$ ,  $\ddot{q}$ : state/velocity/acceleration  
in generalized coordinates  
(each N-dimensional vectors)

gravitational  
forces      frictional  
forces      Centripetal and  
Coriolis forces

“effect of manipulator payload”

J: Jacobian 6xN matrix  
(discussed later)

g: Cartesian wrench vector of  
linear and angular forces  
[ $f_x, f_y, f_z, m_x, m_y, m_z$ ]

# Approaches to EOM derivation

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})^T \mathbf{g}$$

- Lagrangian (energy-based) from last lecture
- Newton-Euler recursive forward-backward algorithm
  - equivalent to Lagrangian formulation, but very different approach
  - based on balanced forces between 2 links at joint articulation point
  - recursively propagate state variables from base to endeffector ...
  - then return recursively with update to forces

Just a few options

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

2. Start with the terminal conditions

$$\mathbf{f}_{n+1} = 0, \quad \boldsymbol{\tau}_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $\mathbf{f}_i$  and  $\boldsymbol{\tau}_i$  for  $i$  decreasing from  $n$  to 1.

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

**Step 0: start from base link**

2. Start with the terminal conditions

$$\mathbf{f}_{n+1} = 0, \quad \boldsymbol{\tau}_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $\mathbf{f}_i$  and  $\boldsymbol{\tau}_i$  for  $i$  decreasing from  $n$  to 1.

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\omega_i = (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i$$

Step 1: compute angular velocity  
for link<sub>i</sub> from link<sub>i-1</sub>

2. Start with the terminal conditions

$$\mathbf{f}_{n+1} = 0, \quad \boldsymbol{\tau}_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $\mathbf{f}_i$  and  $\boldsymbol{\tau}_i$  for  $i$  decreasing from  $n$  to 1.

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

**Step 2: compute angular acceleration for Link<sub>i</sub> from Link<sub>i-1</sub>**

$$\omega_i = (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i$$
$$\alpha_i = (R_{i-1}^i)^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i$$

2. Start with the terminal conditions

$$f_{n+1} = 0, \quad \tau_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $f_i$  and  $\tau_i$  for  $i$  decreasing from  $n$  to 1.

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\omega_i = (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i$$

$$\alpha_i = (R_{i-1}^i)^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i.$$

**Step 3: compute acceleration at center of mass**

$$\mathbf{a}_{c,i} = (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,ci} + \omega_i \times (\omega_i \times \mathbf{r}_{i,ci})$$

2. Start with the terminal conditions

$$f_{n+1} = 0, \tau_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $f_i$  and  $\tau_i$  for  $i$  decreasing from  $n$  to 1.

# Newton-Euler recursion

(from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\omega_i = (R_{i-1}^i)^T \omega_{i-1} + \mathbf{b}_i \dot{q}_i$$

$$\alpha_i = (R_{i-1}^i)^T \alpha_{i-1} + \mathbf{b}_i \ddot{q}_i + \omega_i \times \mathbf{b}_i \dot{q}_i.$$

**Step 4: compute acceleration at end of link**

$$\mathbf{a}_{e,i} = (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,i+1} + \omega_i \times (\omega_i \times \mathbf{r}_{i,i+1})$$

$$\mathbf{a}_{c,i} = (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,ci} + \omega_i \times (\omega_i \times \mathbf{r}_{i,ci})$$

2. Start with the terminal conditions

$$f_{n+1} = 0, \quad \tau_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $f_i$  and  $\tau_i$  for  $i$  decreasing from  $n$  to 1.

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\omega_i = (R_{i-1}^i)^T \omega_{i-1} + \mathbf{b}_i \dot{q}_i$$

$$\mathbf{a}_{e,i} = (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,i+1} + \omega_i \times (\omega_i \times \mathbf{r}_{i,i+1})$$

$$\alpha_i = (R_{i-1}^i)^T \alpha_{i-1} + \mathbf{b}_i \ddot{q}_i + \omega_i \times \mathbf{b}_i \dot{q}_i.$$

2. Start with the terminal conditions

$$f_{n+1} = 0, \quad \tau_{n+1} = 0$$

and use (9.148) and (9.149) to compute  $f_i$  and  $\tau_i$  for  $i$  decreasing from  $n$  to 1.

$$(9.163)$$

**Step 5: recurse to child link and repeat steps 1-4 for each link, until reaching endeffector**

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\begin{aligned}\omega_i &= (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i \\ \alpha_i &= (R_{i-1}^i)^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i \\ \mathbf{a}_{e,i} &= (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,i+1} + \omega_i \times (\omega_i \times \mathbf{r}_{i,i+1}) \\ \mathbf{a}_{c,i} &= (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,ci} + \omega_i \times (\omega_i \times \mathbf{r}_{i,ci})\end{aligned}$$

2. Start with the terminal conditions

$$\mathbf{f}_{n+1} = 0, \quad \boldsymbol{\tau}_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $\mathbf{f}_i$  and  $\boldsymbol{\tau}_i$  for  $i$  decreasing from  $n$  to 1.

**Step 6: start from endeffector Link**

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\begin{aligned} \omega_i &= (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i \\ \alpha_i &= (R_{i-1}^i)^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i \\ \mathbf{a}_{e,i} &= (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,i+1} + \omega_i \times (\omega_i \times \mathbf{r}_{i,i+1}) \\ \mathbf{a}_{c,i} &= (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,ci} + \omega_i \times (\omega_i \times \mathbf{r}_{i,ci}) \end{aligned}$$

2. Start with the terminal conditions

$$\mathbf{f}_{n+1} = 0, \quad \boldsymbol{\tau}_{n+1} = 0 \quad (9.163)$$

and use (9.148) and (9.149) to compute  $\mathbf{f}_i$  and  $\boldsymbol{\tau}_i$  for  $i$  decreasing from  $n$  to 1.

$$\mathbf{f}_i = R_i^{i+1} \mathbf{f}_{i+1} + m_i \mathbf{a}_{c,i} - m_i \mathbf{g}_i$$

**Step 7: compute force exerted by link<sub>i-1</sub> on link<sub>i</sub>**

# Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\begin{aligned}\omega_i &= (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i \\ \alpha_i &= (R_{i-1}^i)^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i \\ \mathbf{a}_{e,i} &= (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,i+1} + \omega_i \times (\omega_i \times \mathbf{r}_{i,i+1}) \\ \mathbf{a}_{c,i} &= (R_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,ci} + \omega_i \times (\omega_i \times \mathbf{r}_{i,ci})\end{aligned}$$

2. Start with the terminal conditions

$$\mathbf{f}_{n+1} = 0, \quad \tau_{n+1} = 0$$

and use (9.148) and (9.149) to compute  $\mathbf{f}_i$  and  $\tau_i$  for  $i$  decreasing from  $n$  to 1.

$$\mathbf{f}_i = R_i^{i+1} \mathbf{f}_{i+1} + m_i \mathbf{a}_{c,i} - m_i \mathbf{g}_i$$

**Step 8: compute torque exerted by link<sub>i-1</sub> on link<sub>i</sub>**

$$\boxed{\tau_i = R_i^{i+1} \tau_{i+1} - \mathbf{f}_i \times \mathbf{r}_{i,ci} + (R_i^{i+1} \mathbf{f}_{i+1}) \times \mathbf{r}_{i+1,ci} + \alpha_i + \omega_i \times (I_i \omega_i)}$$

# Newton-Euler recursion (from Spong textbook)

- Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, \mathbf{a}_{c,0} = 0, \mathbf{a}_{e,0} = 0 \quad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute  $\omega_i$ ,  $\alpha_i$  and  $\mathbf{a}_{c,i}$  for  $i$  increasing from 1 to  $n$ .

$$\omega_i = (\mathbf{R}_{i-1}^i)^T \omega_{i-1} + \mathbf{b}_i \dot{q}_i \quad \mathbf{a}_{e,i} = (\mathbf{R}_{i-1}^i)^T \mathbf{a}_{e,i-1} + \dot{\omega}_i \times \mathbf{r}_{i,i+1} + \omega_i \times (\omega_i \times \mathbf{r}_{i,i+1})$$

**Step 9: recurse to parent link and repeat steps 7-8 for each link, until reaching base**

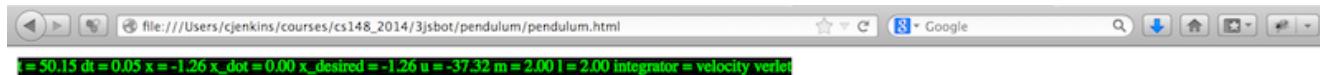
- Start with

$$f_{n+1} = 0, \quad \tau_{n+1} = 0 \quad (9.163)$$

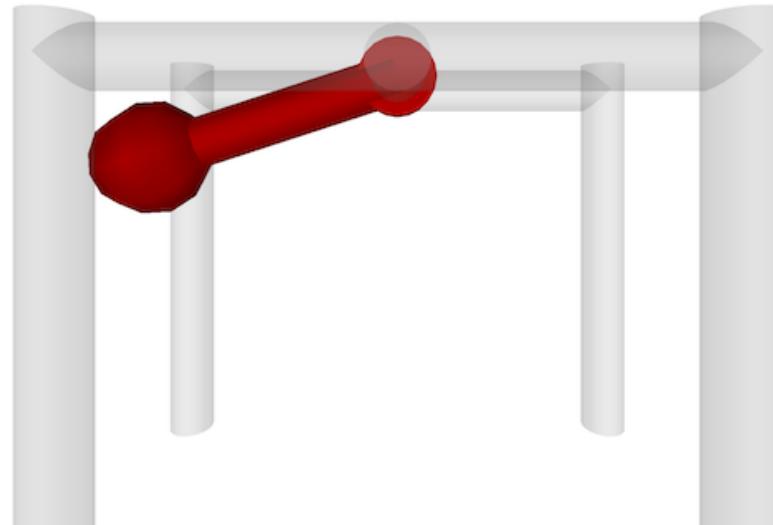
and use (9.148) and (9.149) to compute  $f_i$  and  $\tau_i$  for  $i$  decreasing from  $n$  to 1.

$$\begin{aligned} f_i &= \mathbf{R}_i^{i+1} f_{i+1} + m_i \mathbf{a}_{c,i} - m_i \mathbf{g}_i \\ \tau_i &= R_i^{i+1} \tau_{i+1} - f_i \times \mathbf{r}_{i,ci} + (R_i^{i+1} f_{i+1}) \times \mathbf{r}_{i+1,ci} + \alpha_i + \omega_i \times (I_i \omega_i). \end{aligned}$$

# Let's see what happens



What happens when the pendulum changes?  
(mass, length, gravity)



bigger  
different

# PD as a Linear System

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -\frac{g}{l} \sin(x_2) \\ x_1 \end{pmatrix} + \tau \begin{pmatrix} 1/m l^2 \\ 0 \end{pmatrix} \xrightarrow{\text{linearization}} \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -\frac{g}{l} x_2 \\ x_1 \end{pmatrix} + \tau \begin{pmatrix} 1/m l^2 \\ 0 \end{pmatrix} \text{ or}$$

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -\frac{g}{l} x_2 \\ x_1 \end{pmatrix} + \tau \begin{pmatrix} 1/m l^2 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{g}{l} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \tau \begin{pmatrix} 1/m l^2 \\ 0 \end{pmatrix} \text{ or}$$

$[x_1 \ x_2]^T$  vector of position and velocity

assuming pendulum small angle approximation:  $\sin \theta \approx \theta$

# PID as a Linear System

- Linearized System

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{g}{l} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \tau \begin{pmatrix} \frac{1}{ml^2} \\ 0 \end{pmatrix}$$

general form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

- The Integral Controller introduces a new state:

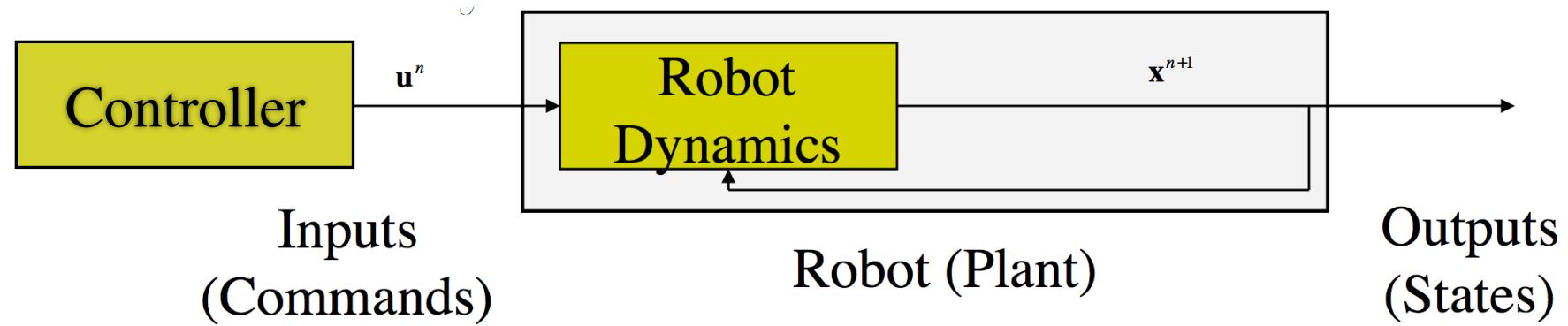
$$\dot{x}_3 = k_I(x_d - x_2)$$

- The new (linearized) system becomes;

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{g}{l} & 0 \\ 1 & 0 & 0 \\ 0 & -k_I & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} \frac{\tau}{ml^2} \\ 0 \\ k_i x_d \end{pmatrix} \quad u = \tau = k_p(x_d - x_2) + k_D(0 - x_1) + k_I x_3$$

PID does not consider  
system dynamics

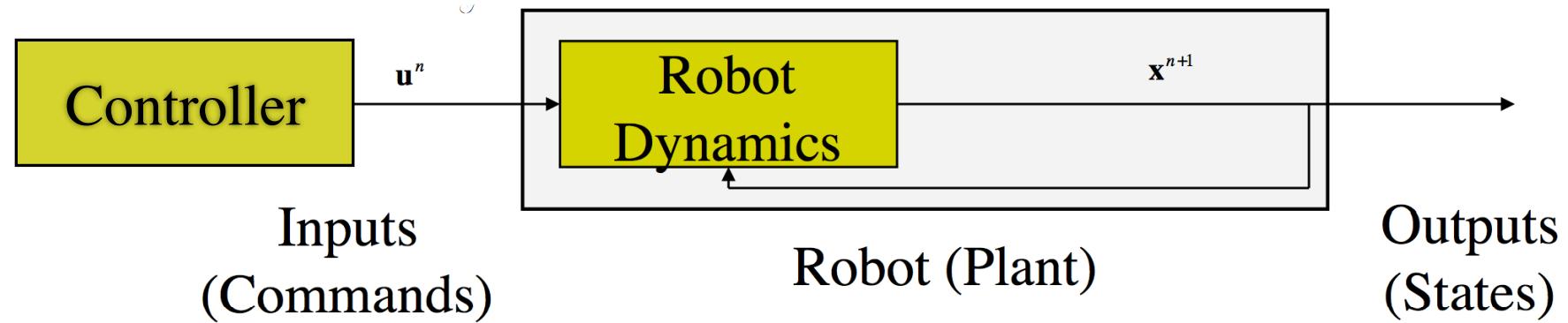
$$u = \pi(t, x, x_d) \quad \dot{x} = f(t, x, u)$$



How can dynamics be accounted for in control?

*forward model*

$$u = \pi(t, x, x_d) \quad \dot{x} = f(t, x, u)$$

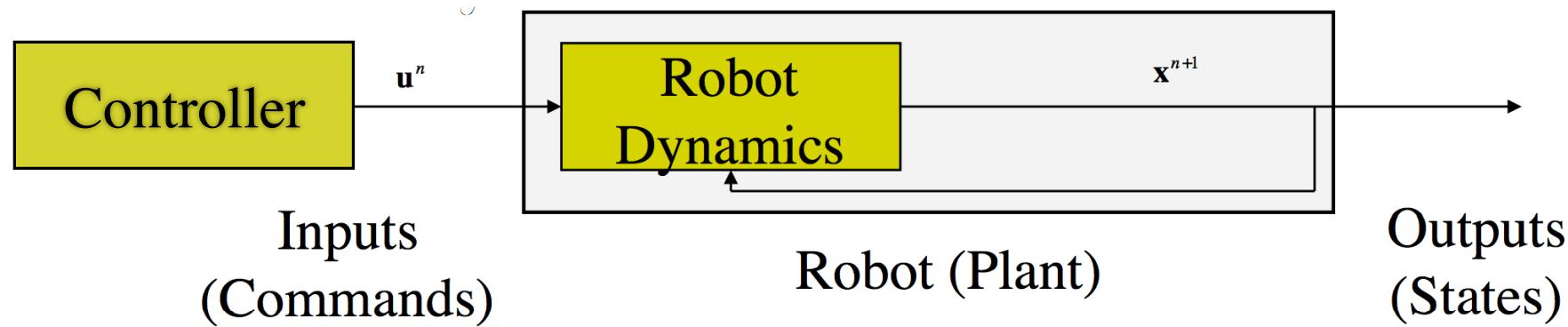


inverse model, if  $\dot{x} = f(t, x, \pi(t, x, x_d))$



forward model

$$u = \pi(t, x, x_d) \quad \dot{x} = f(t, x, u)$$



cost to  
desired  
state

# Optimal Control

Search over controls  $u$  that minimize cost

$$J = \Phi [ \mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f ] + \int_{t_0}^{t_f} \mathcal{L} [ \mathbf{x}(t), \mathbf{u}(t), t ] \, dt$$

subject to the first-order dynamic constraints

$$\dot{\mathbf{x}}(t) = \mathbf{a} [ \mathbf{x}(t), \mathbf{u}(t), t ],$$

the algebraic *path constraints*

$$\mathbf{b} [ \mathbf{x}(t), \mathbf{u}(t), t ] \leq \mathbf{0},$$

and the *boundary conditions*

$$\phi [ \mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f ] = 0$$

dynamics model

## Linear quadratic control [\[edit\]](#)

---

A special case of the general nonlinear optimal control problem given in the previous section is the *linear quadratic* (LQ) optimal control problem. The LQ problem is stated as follows. Minimize the *quadratic* continuous-time cost functional

$$J = \frac{1}{2} \mathbf{x}^T(t_f) \mathbf{S}_f \mathbf{x}(t_f) + \frac{1}{2} \int_{t_0}^{t_f} [\mathbf{x}^T(t) \mathbf{Q}(t) \mathbf{x}(t) + \mathbf{u}^T(t) \mathbf{R}(t) \mathbf{u}(t)] dt$$

Subject to the *linear* first-order dynamic constraints

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t) \mathbf{x}(t) + \mathbf{B}(t) \mathbf{u}(t),$$

and the initial condition

$$\mathbf{x}(t_0) = \mathbf{x}_0$$

A particular form of the LQ problem that arises in many control system problems is that of the *linear quadratic regulator* (LQR) where all of the matrices (i.e.,  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{Q}$ , and  $\mathbf{R}$ ) are *constant*, the initial time is arbitrarily set to zero, and the terminal time is taken in the limit  $t_f \rightarrow \infty$  (this last assumption is what is known as *infinite horizon*). The LQR problem is stated as follows. Minimize the infinite horizon quadratic continuous-time cost functional

$$J = \frac{1}{2} \int_0^\infty [\mathbf{x}^T(t) \mathbf{Q} \mathbf{x}(t) + \mathbf{u}^T(t) \mathbf{R} \mathbf{u}(t)] dt$$

Subject to the *linear time-invariant* first-order dynamic constraints

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t),$$

and the initial condition

$$\mathbf{x}(t_0) = \mathbf{x}_0$$

LQR is common form of optimal control

Next class:

Coordinate Systems and  
Linear Algebra review

