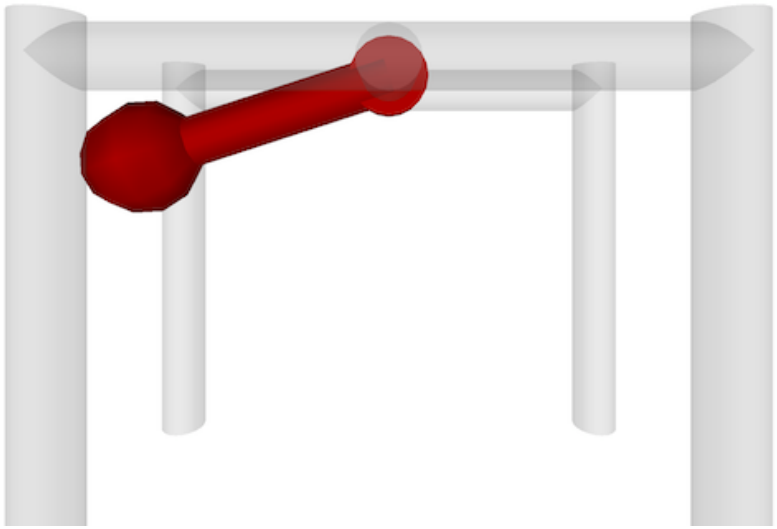
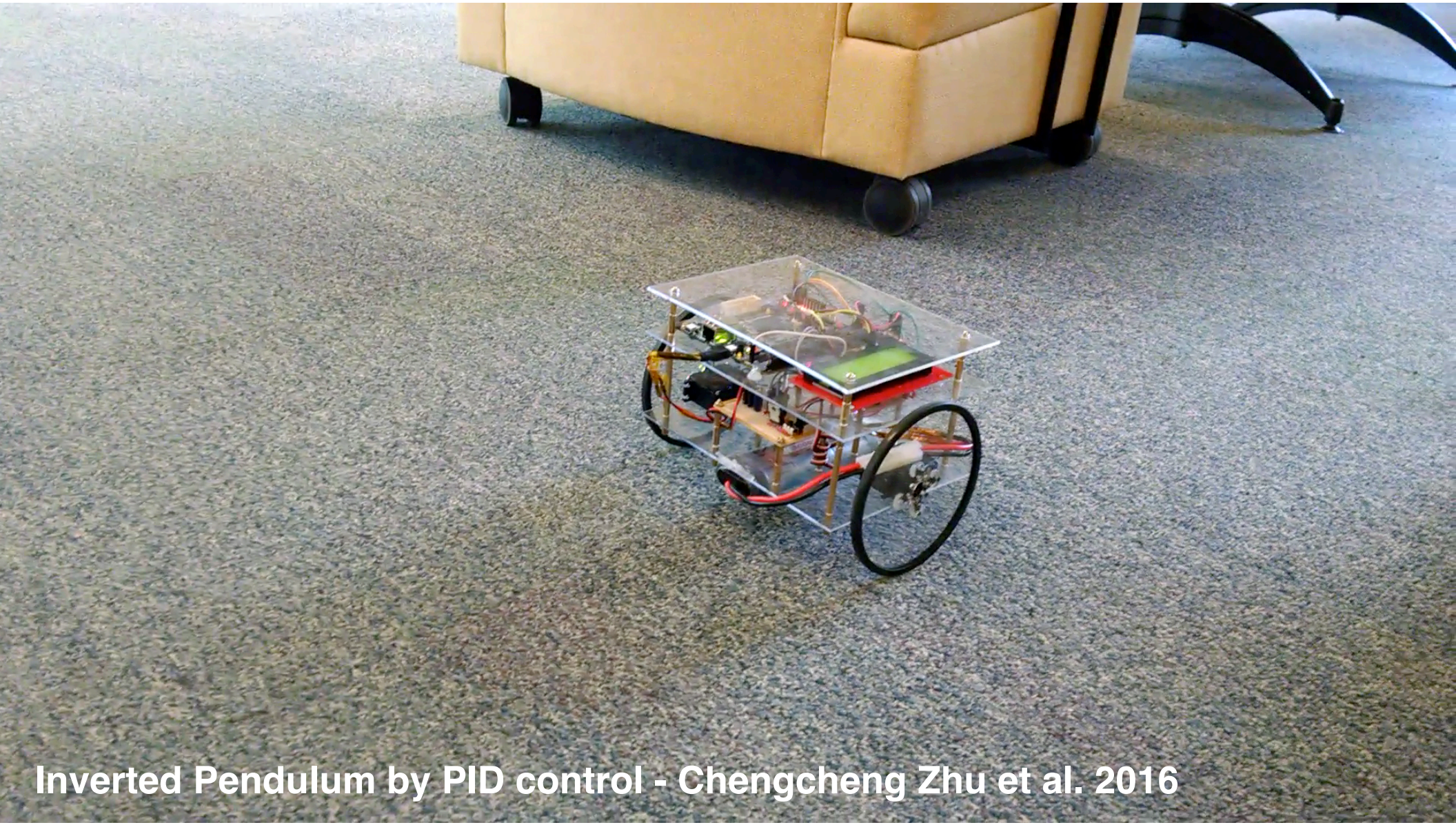

PID Motion Control
autorob.github.io

Project 2: Pendularm



https://raw.githubusercontent.com/ohseejay/kineval-stencil-f16/master/project_pendularm/pendularm1.html

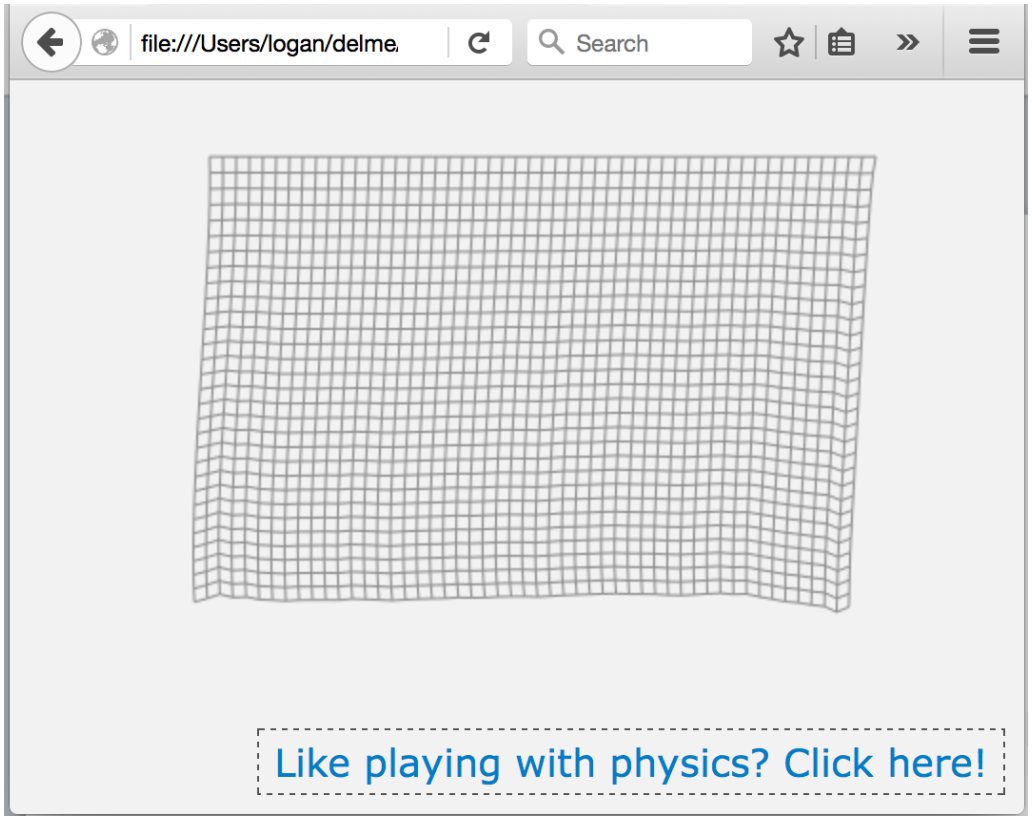
```
function init() {...
   pendulum = { // pendulum object
        length:2.0,
        mass:2.0,
        angle:Math.PI/2,
        angle_dot:0.0};

    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};
...}
```


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


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

Let’s start with something fun

Verlet Simulated Mass-Spring Cloth

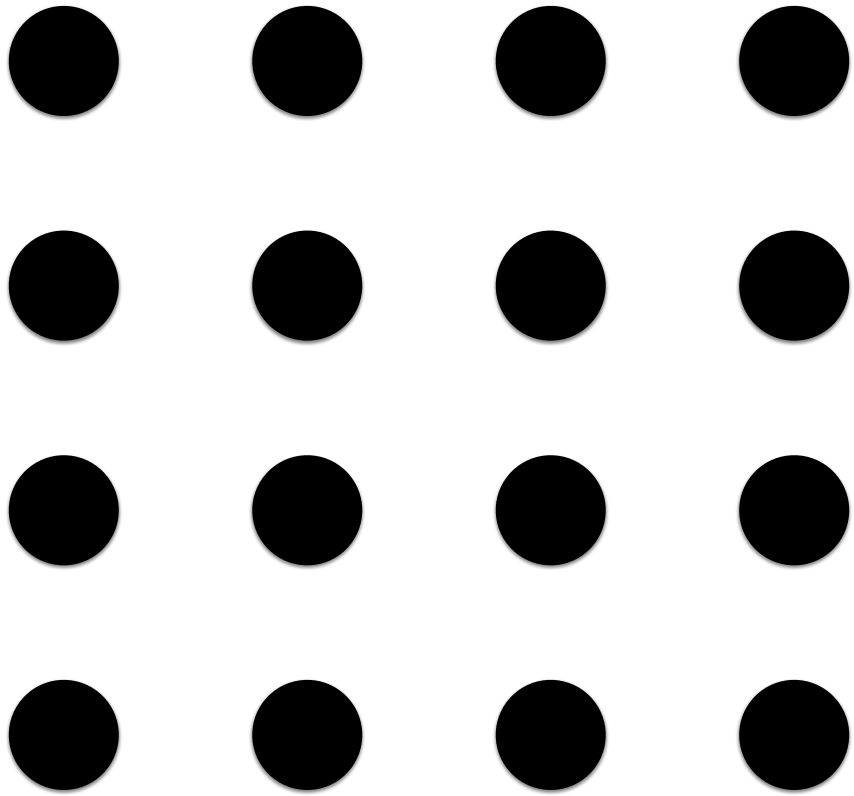


- Cloth represented as a network of point-mass nodes
- Neighboring nodes connected by high stiffness springs

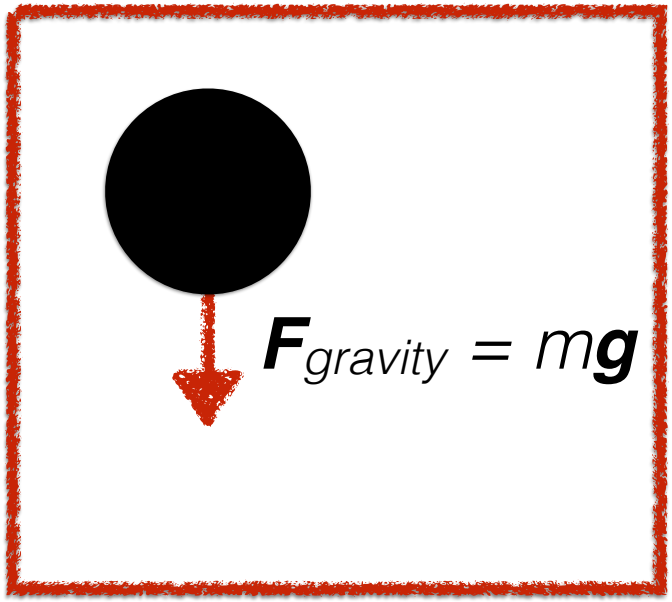
http://codepen.io/dissimulate/pen/KrAwx

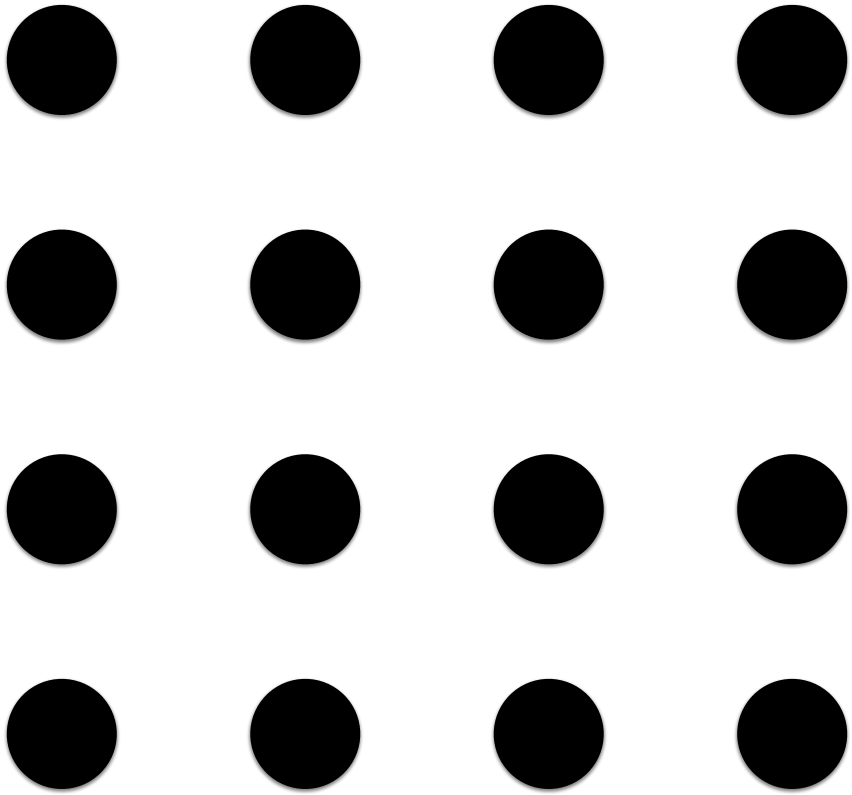
https://github.com/Dissimulate/Tearable-Cloth

Start with point-mass particles

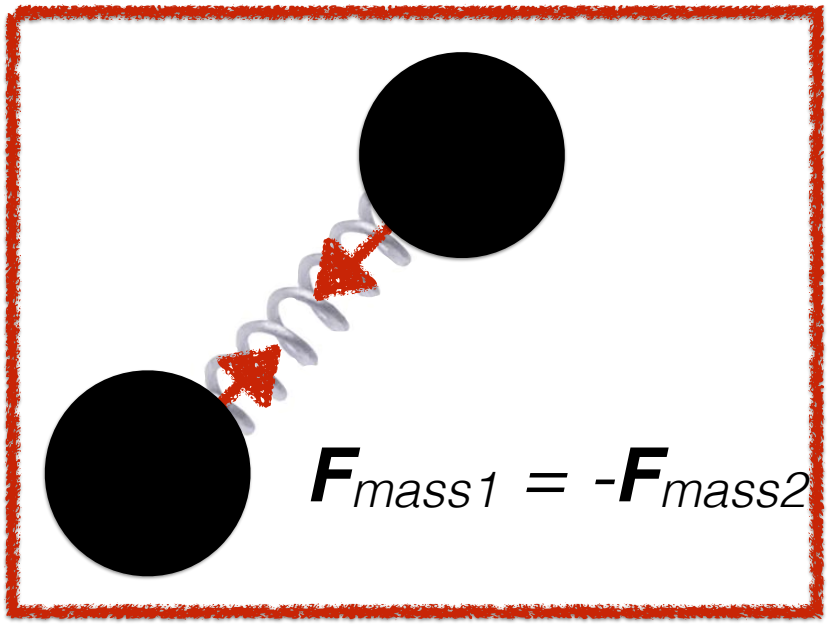


Integrate with gravity

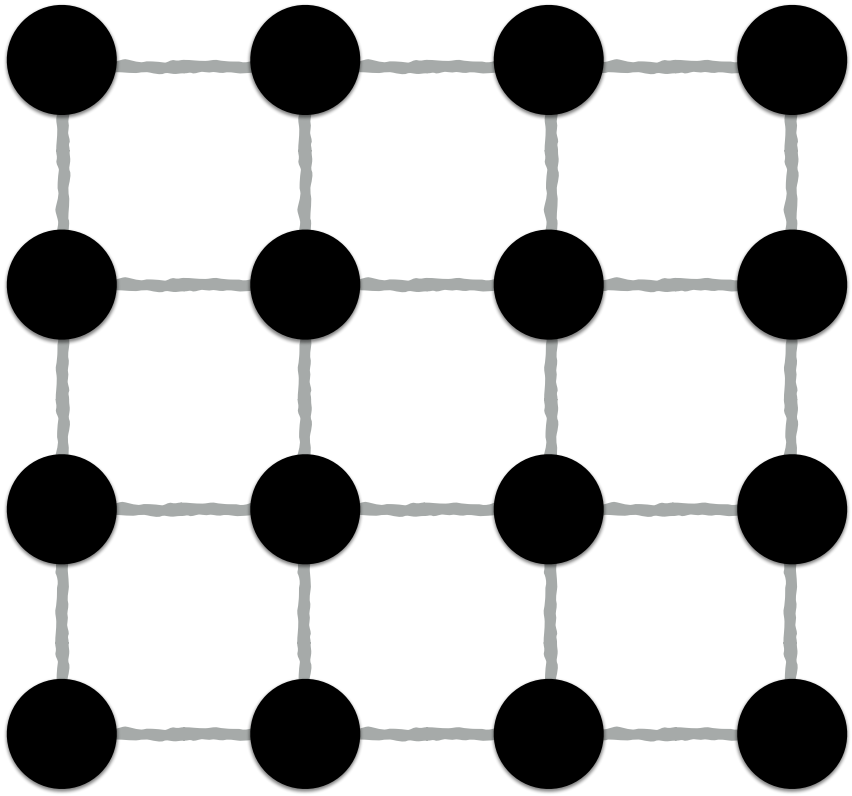




Enforce spring force constraints

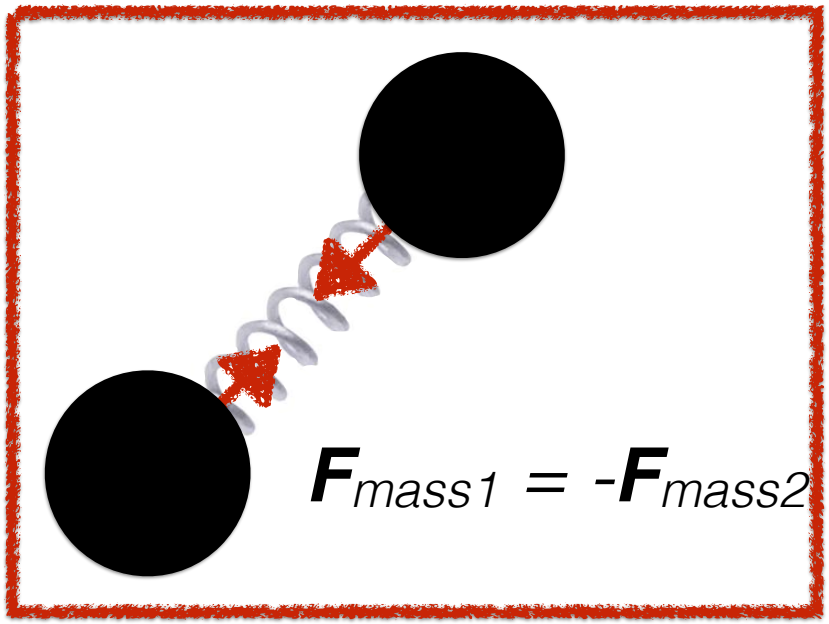


spring with stiffness 1
and rest length *r*



Jakobsen, http://www.gamasutra.com/resource_guide/20030121/jacobson_01.shtml

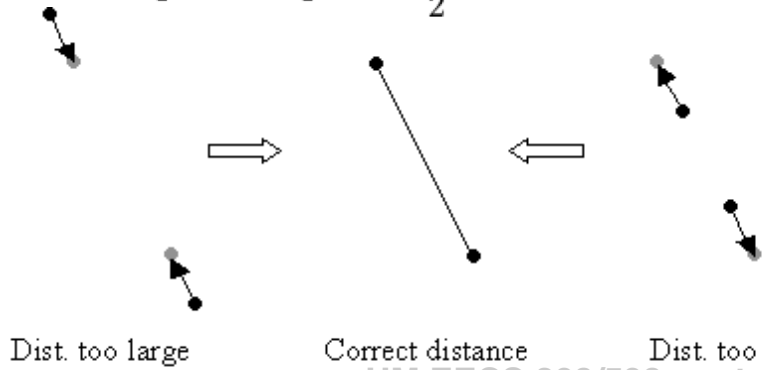
Enforce spring force constraints

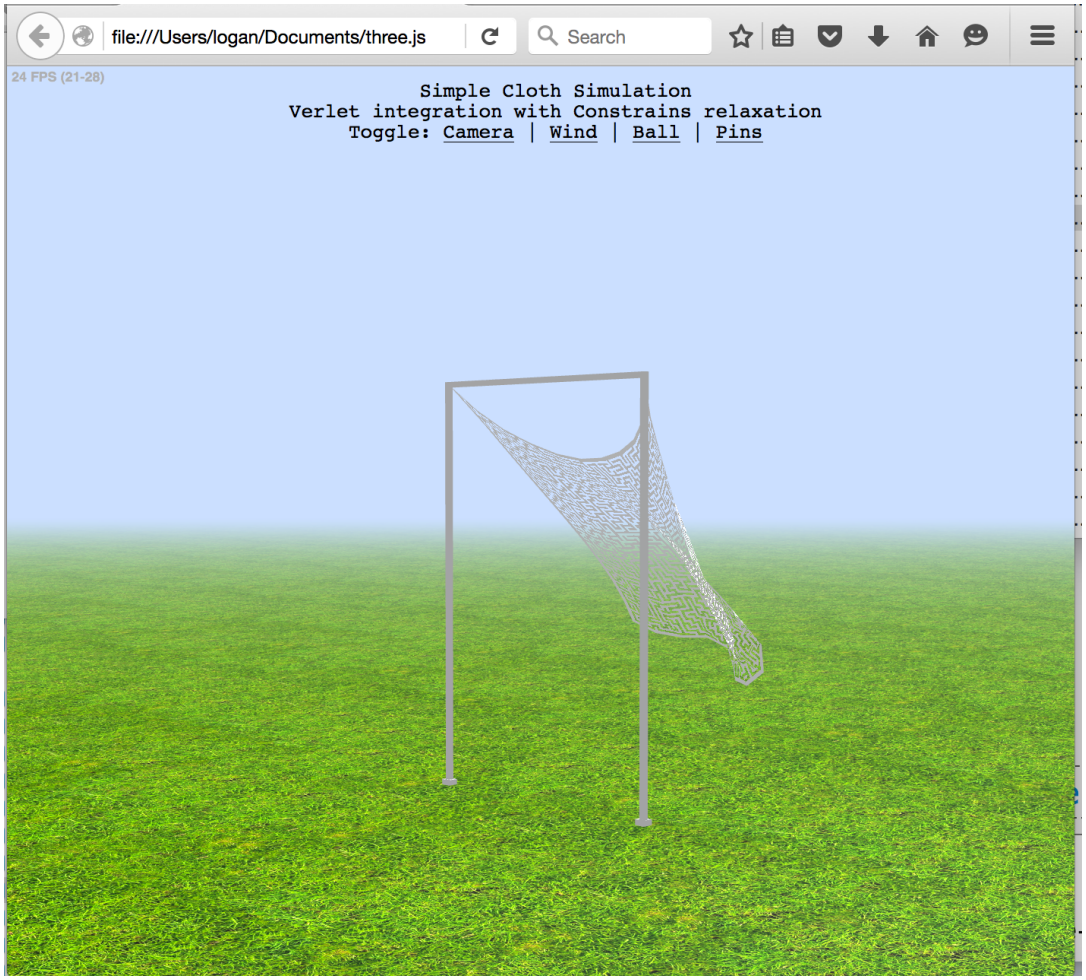


spring with stiffness 1
and rest length *r*

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
$$x_1^{(t+\Delta t)} = \tilde{x}_1^{(t+\Delta t)} + \frac{1}{2} d_1 d_3$$
$$x_2^{(t+\Delta t)} = \tilde{x}_2^{(t+\Delta t)} - \frac{1}{2} d_1 d_3$$ Correction towards rest length

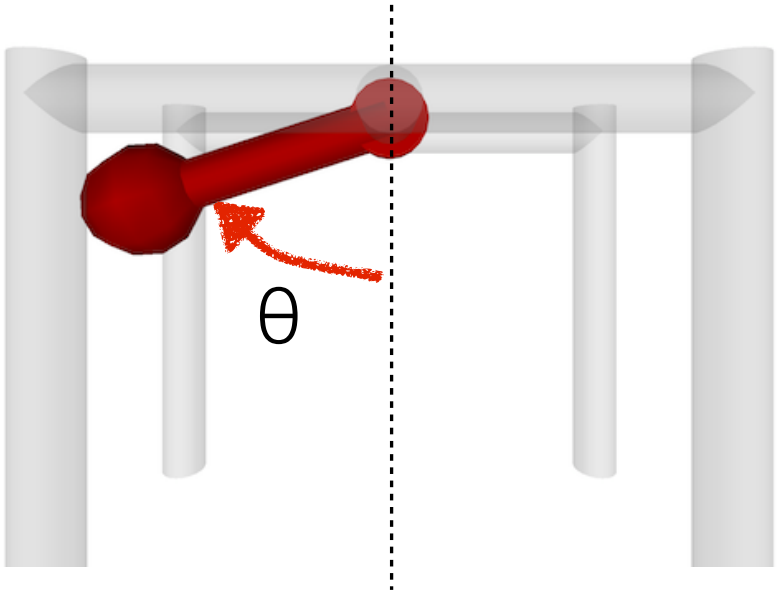




http://threejs.org/examples/webgl_animation_cloth

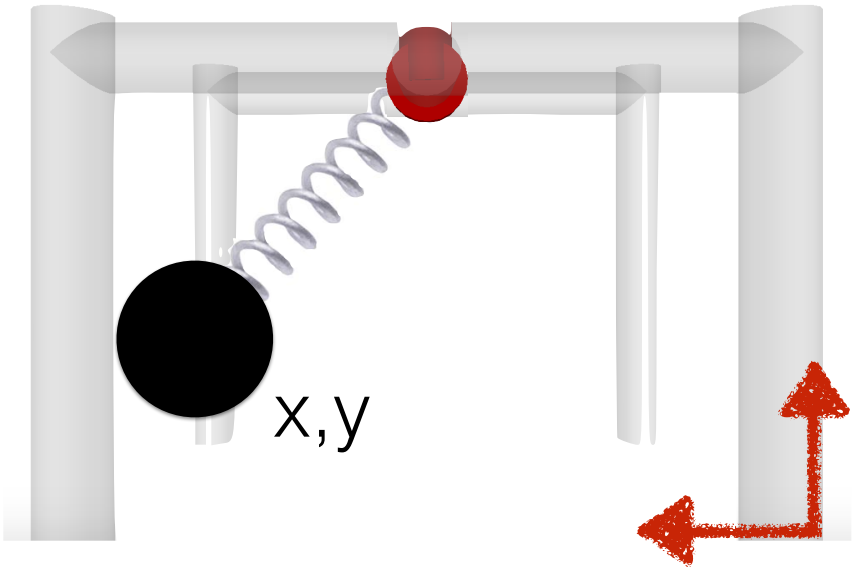
Pendulum Revisited

Generalized coordinates



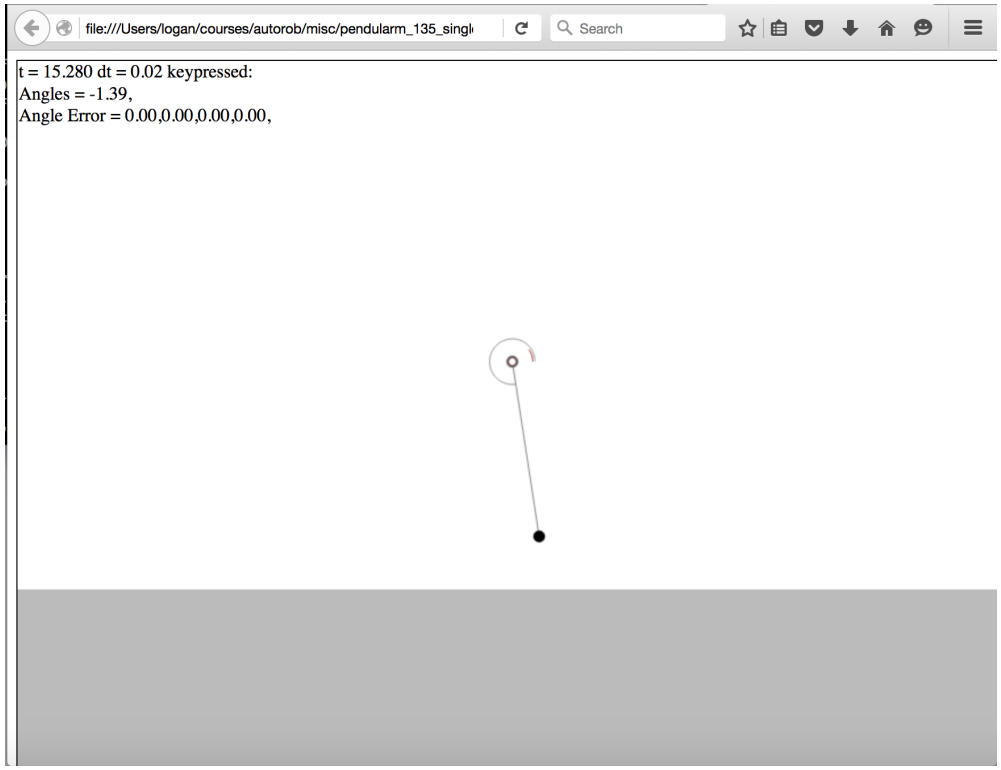
Simulate the motion of DOFs (pendulum angle) directly

Maximal coordinates

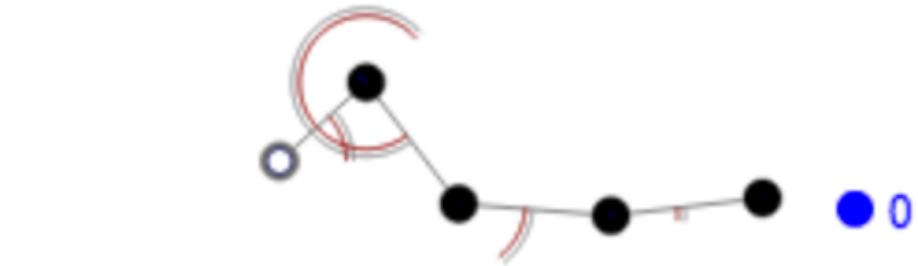


Simulate the motion of individual bodies (pendulum bob location) corrected by constraints (spring)

Maximal Pendularm

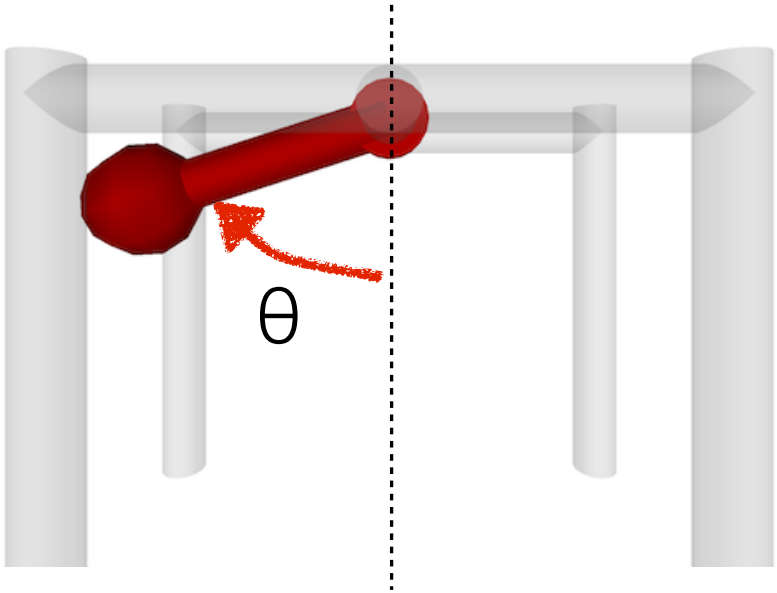


Maximal N-Link Pendularm



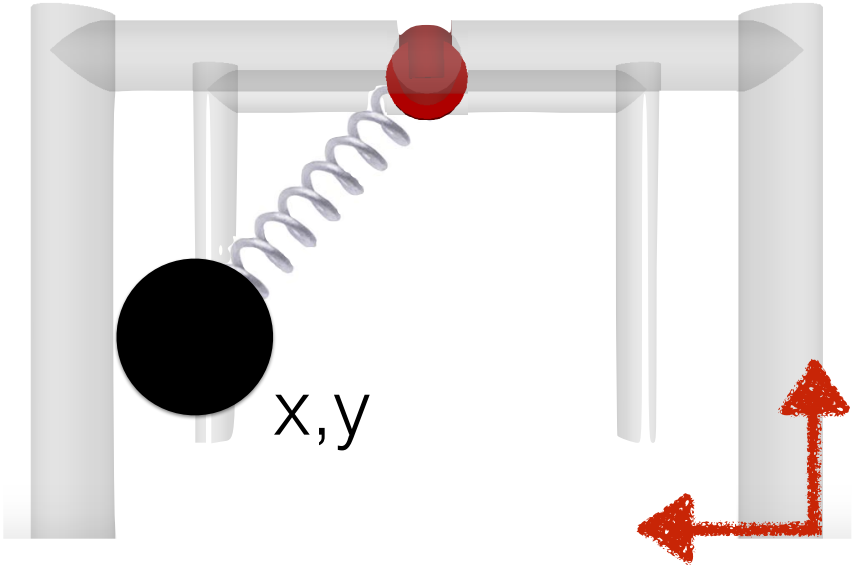
Why use generalized coordinates?

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*)

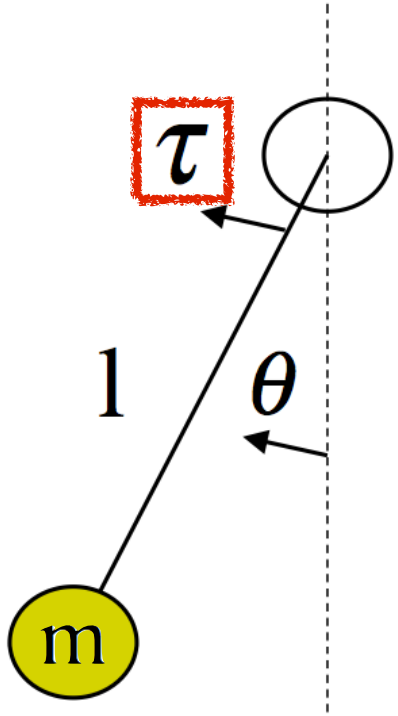
$$I\ddot{\theta} = -mgl\sin(\theta) + \tau$$

with Parallel Axis Theorem (I=mr^2)

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

Numerical integration over time

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

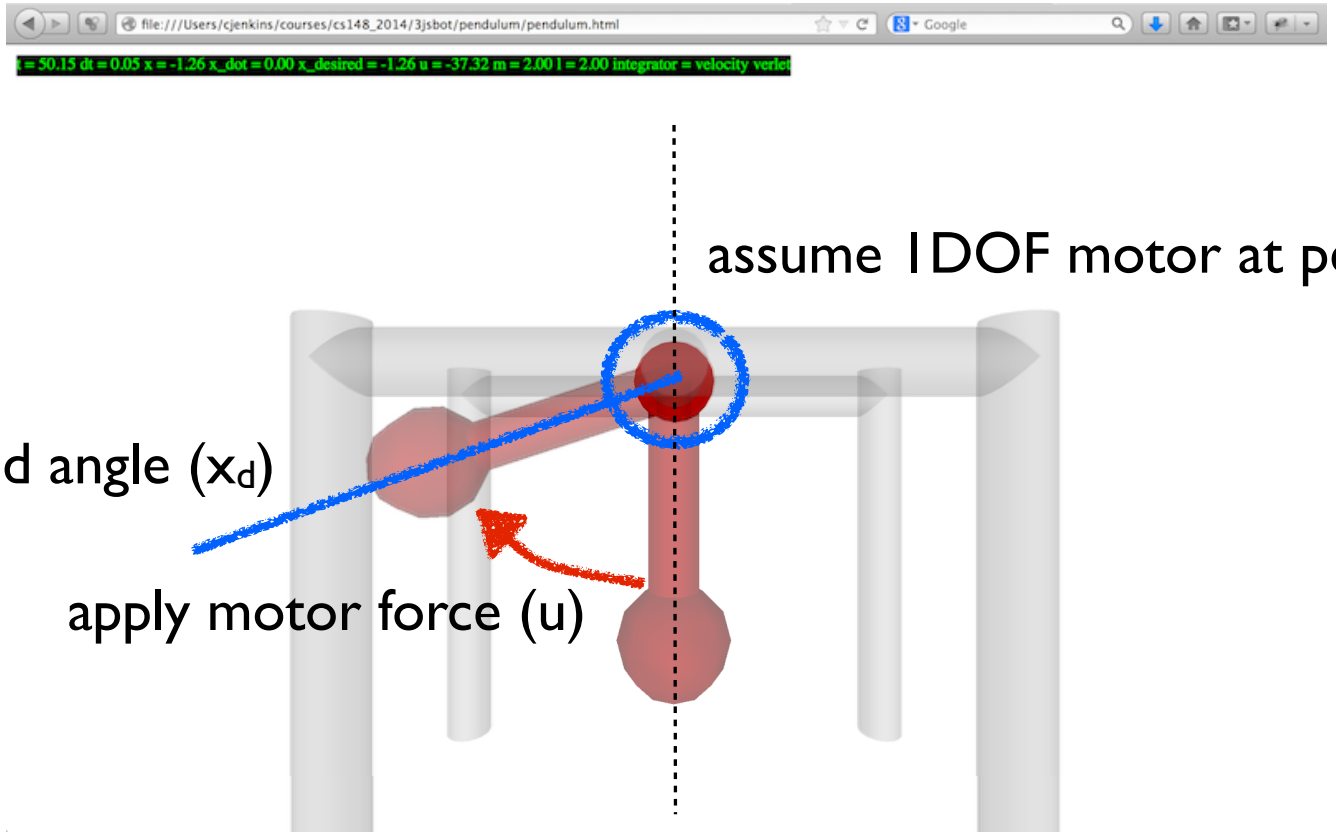


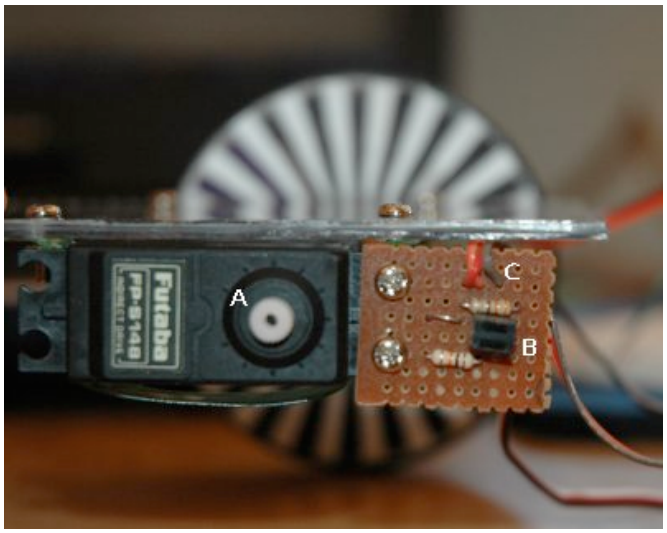
Motor produces torque
(angular force)

Angle expresses pendulum
range of motion

Pendulum of length l with
point mass m

Can we control the pendulum?

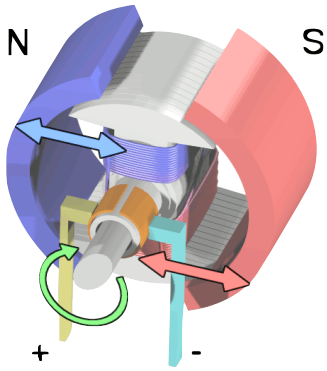


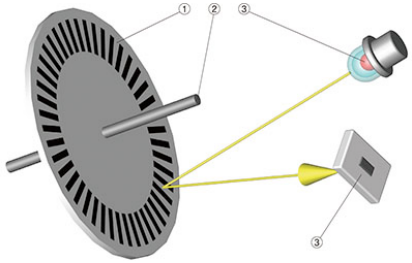


Servos have:

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

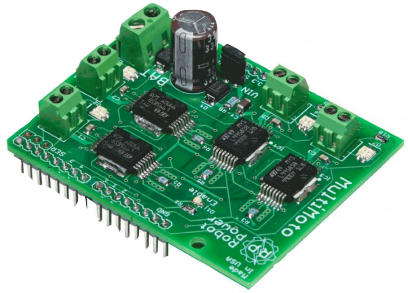
Actuator (electric)

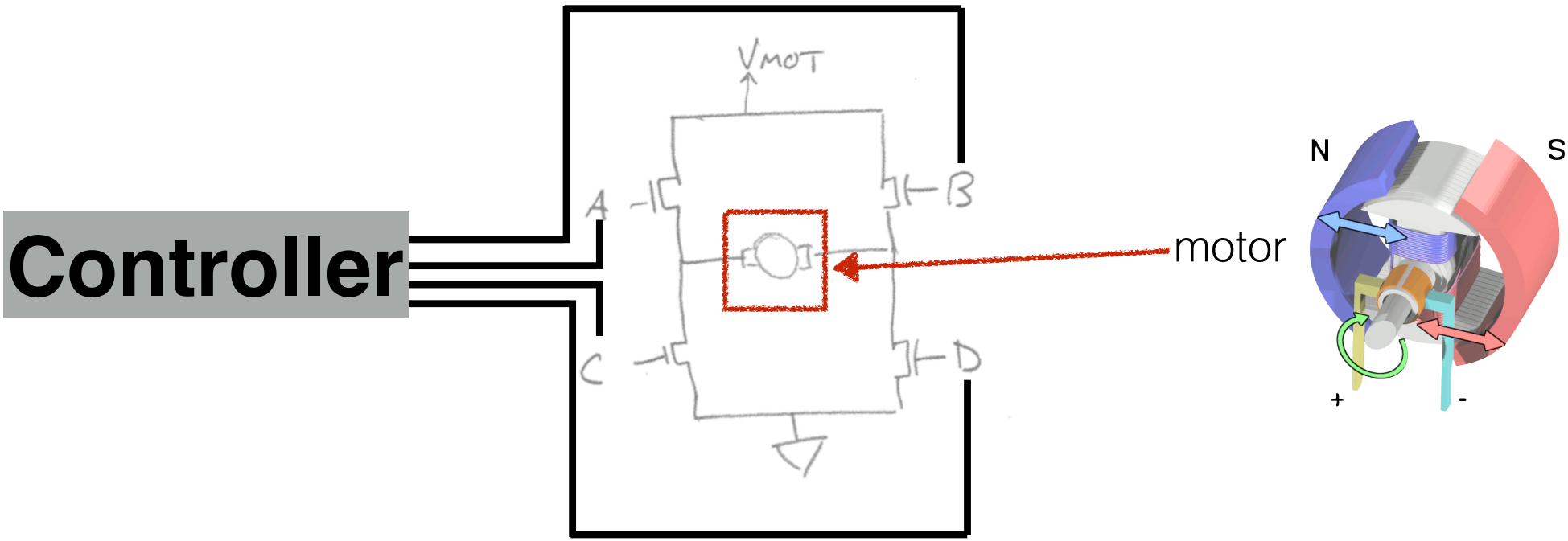




Proprioception (optical encoder)

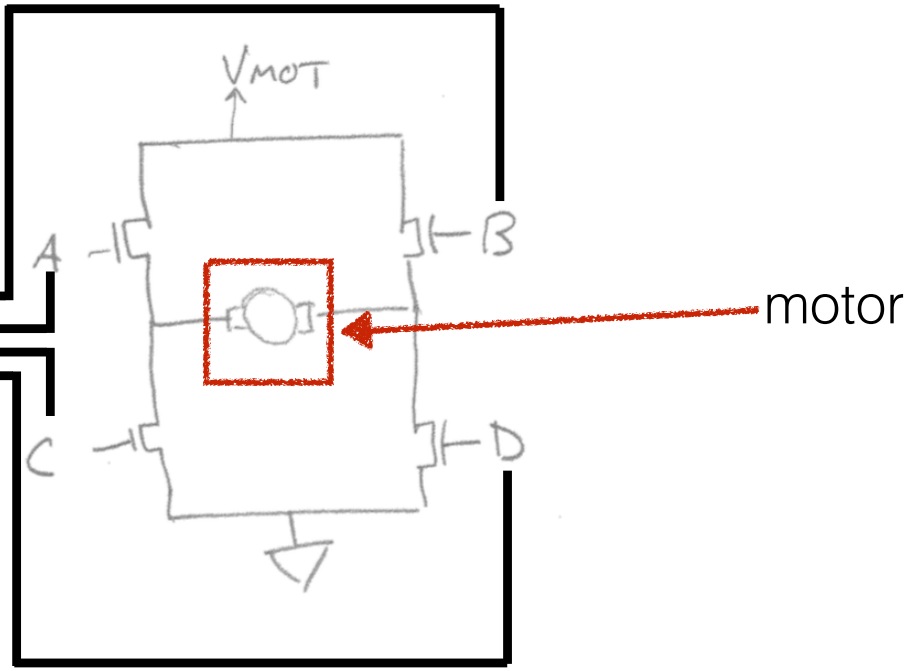
Controller (4 channel H-Bridges)

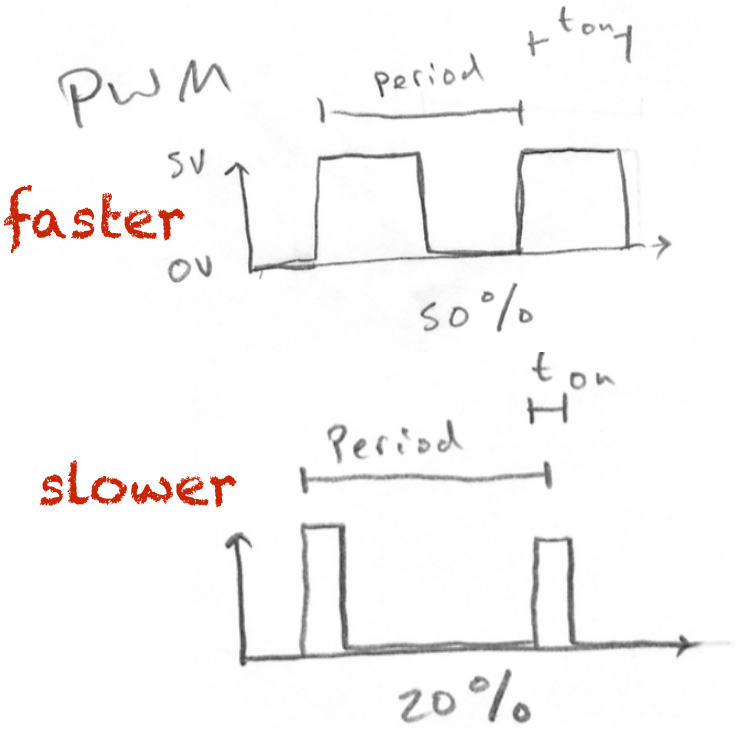




controller sends control PWM signal
to H-bridge.
H-bridge sends proportional current to motor.
Current actuates motor

Controller



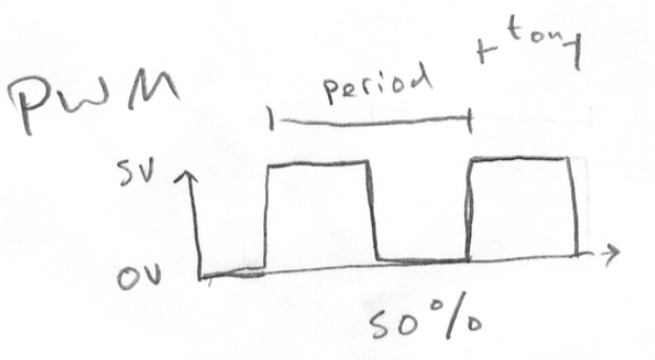


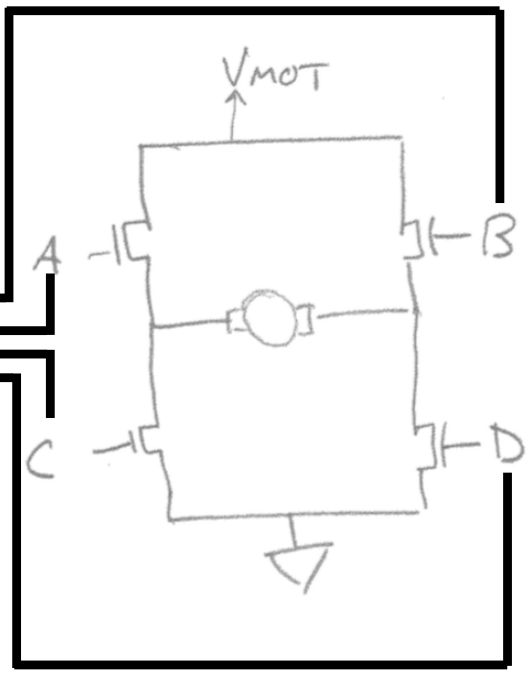
PWM (Pulse width modulation)

PWM has a periodic duty cycle.

Percentage of time in high voltage determines speed of motor

Controller





[redacted]

spin forward:
A and D

spin backward:
B and C

what if none are on:?

[redacted]

what if C and D?

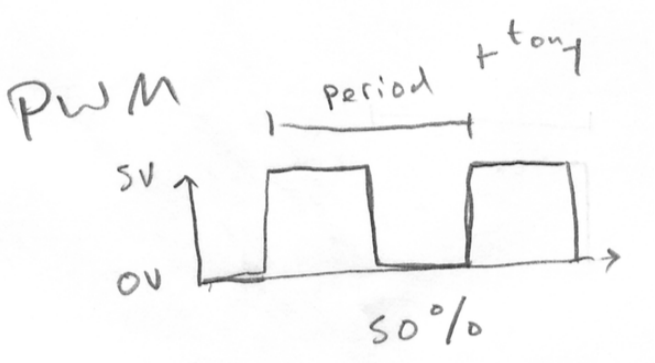
[redacted]

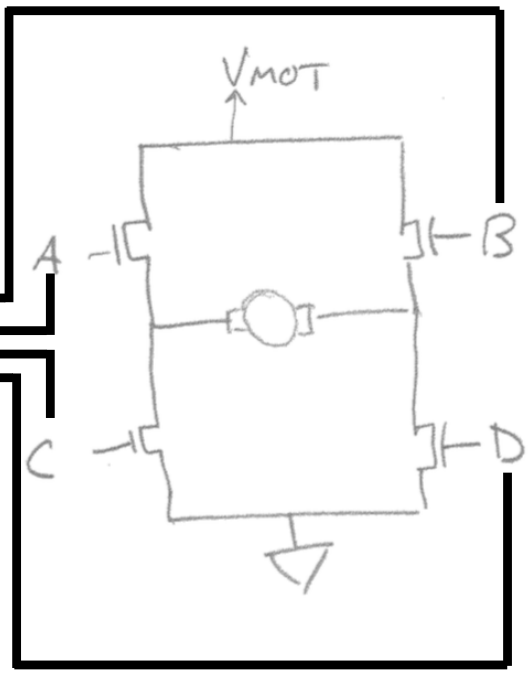
what if A and B?

[redacted]

what if A and C?

Controller





spin forward:
A and D

spin backward:
B and C

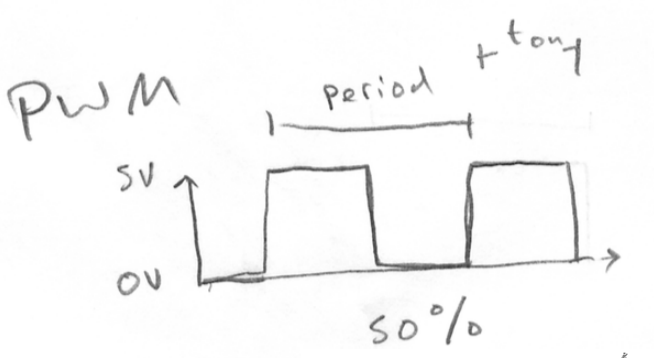
what if none are on:?
open circuit
no current -> no torque

what if C and D?
negative torque -> brake

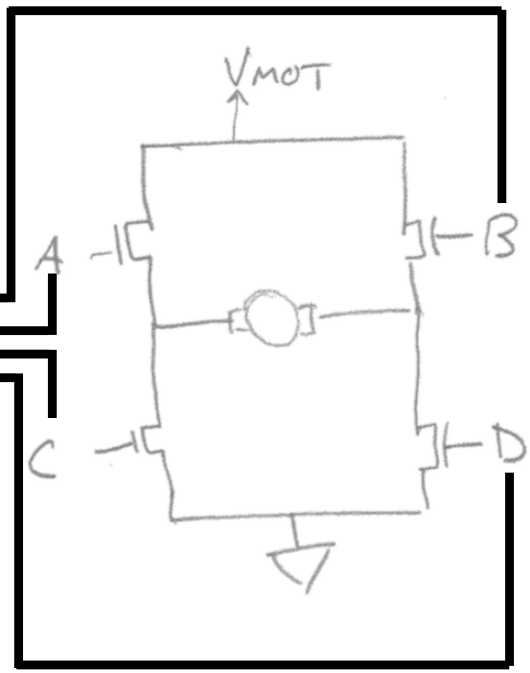
what if A and B?
negative torque -> brake

what if A and C?

Controller



No more robot, (our hardware prevents this)



spin forward:
A and D

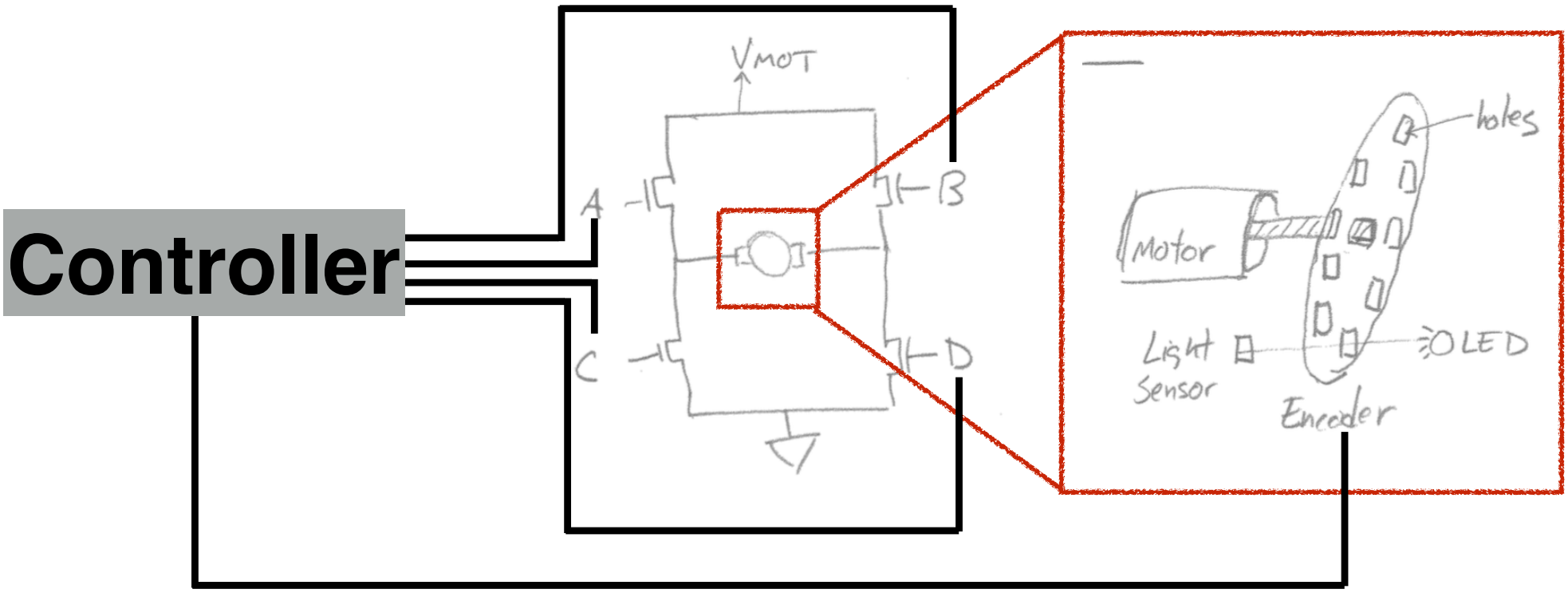
spin backward:
B and C

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

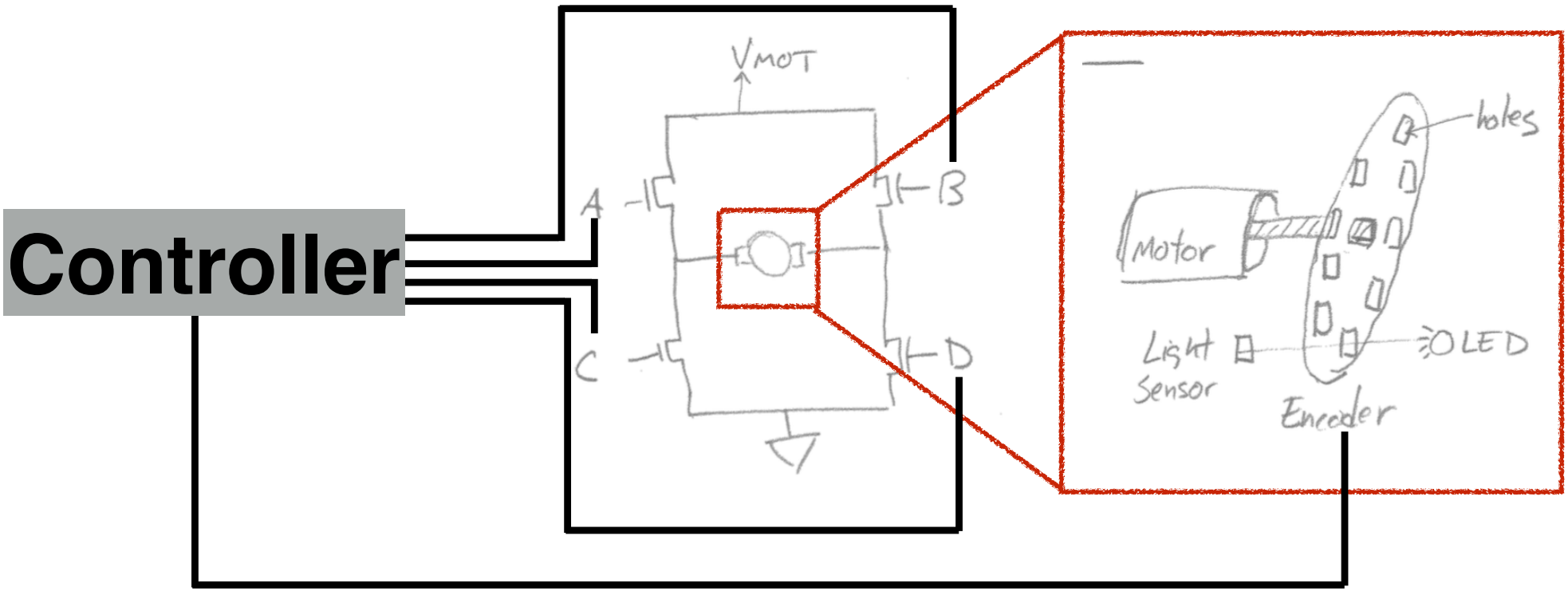
what if C and D?
negative torque -> brake

what if A and B?
negative torque -> brake

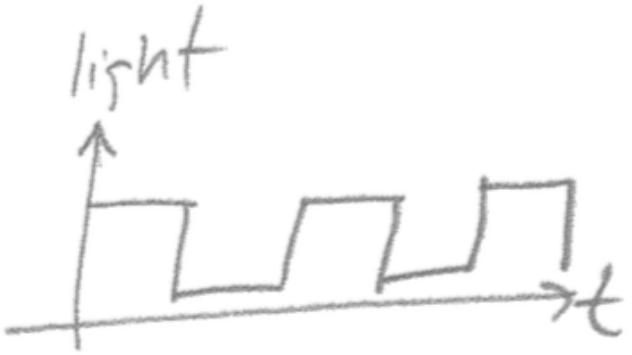
what if A and C?



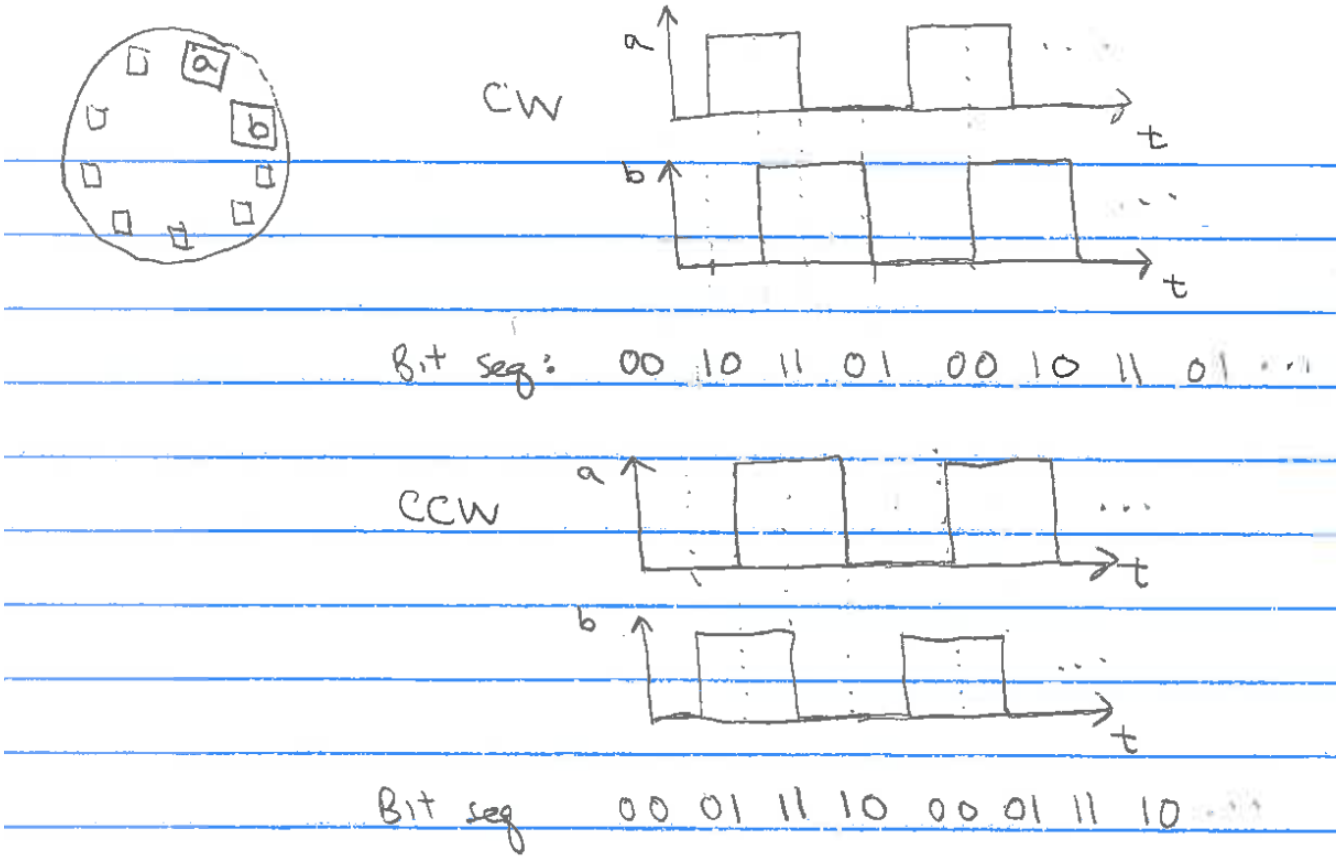
encoder sends joint state to back to controller



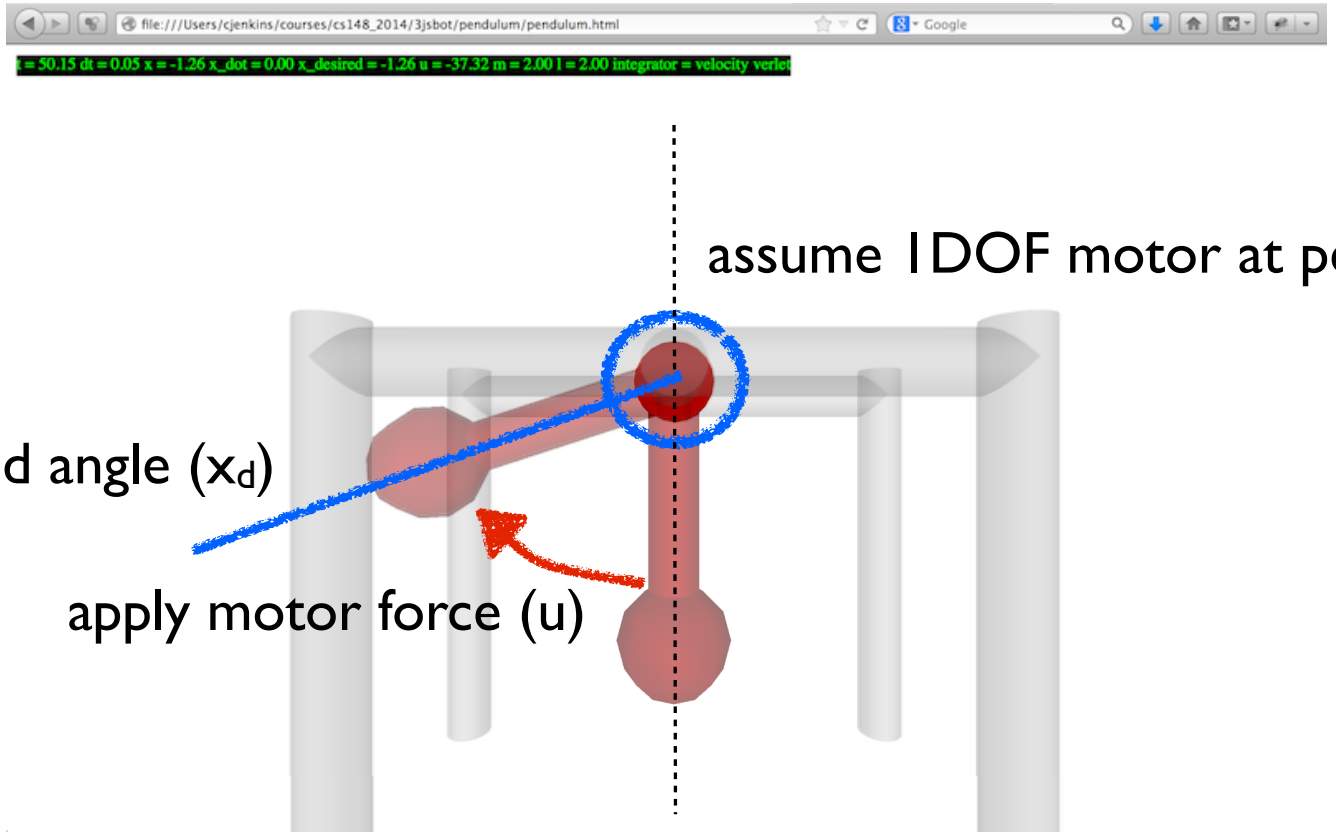
as motor turns, encoder goes between sensed and occluded light



Quadrature encoders

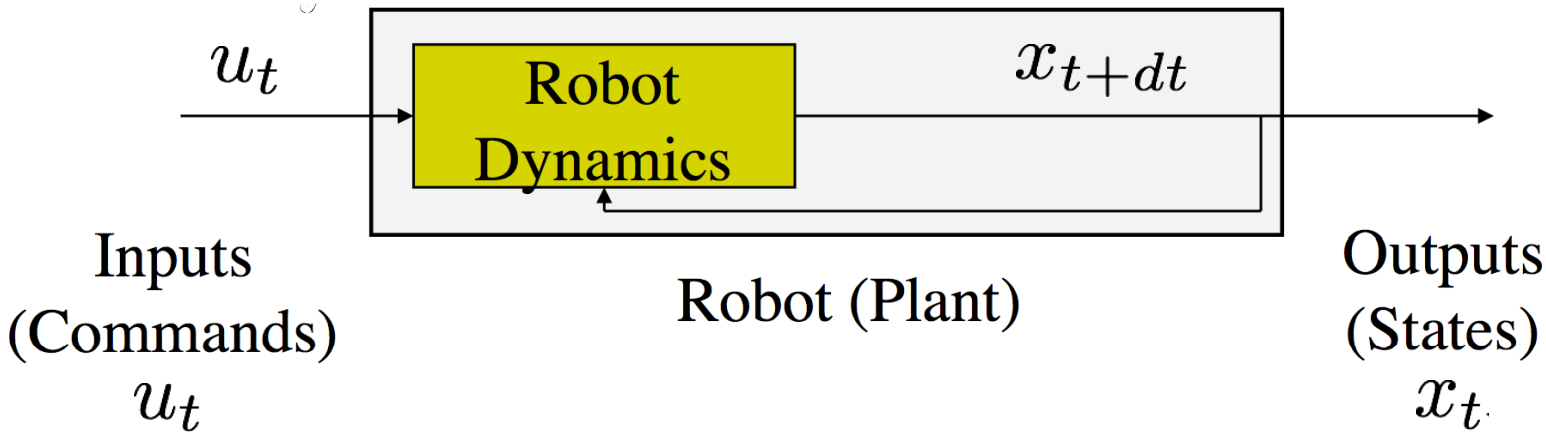


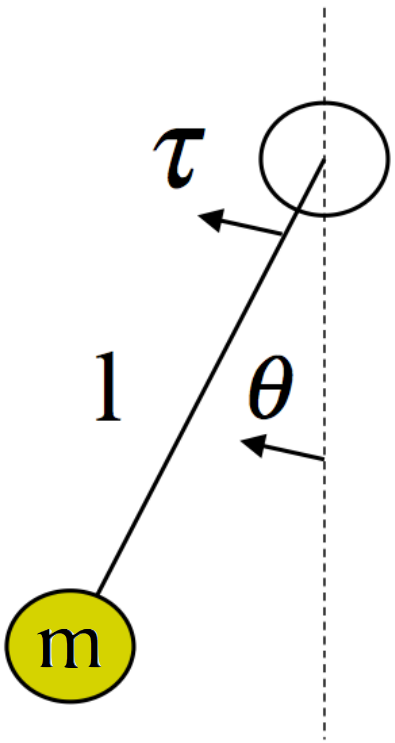
Can we control the pendulum?



Robot "plant" represents system dynamics as function

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

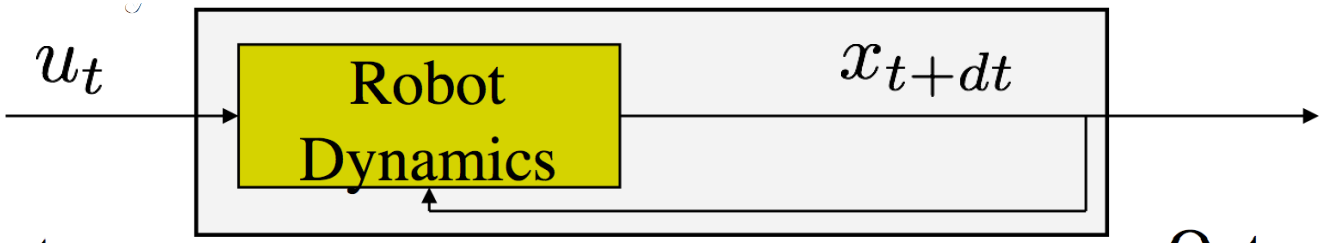




$$\ddot{\theta} = -\frac{g}{l}\sin(\theta) + \frac{\tau}{ml^2}$$

Motor command used within equations of motion

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



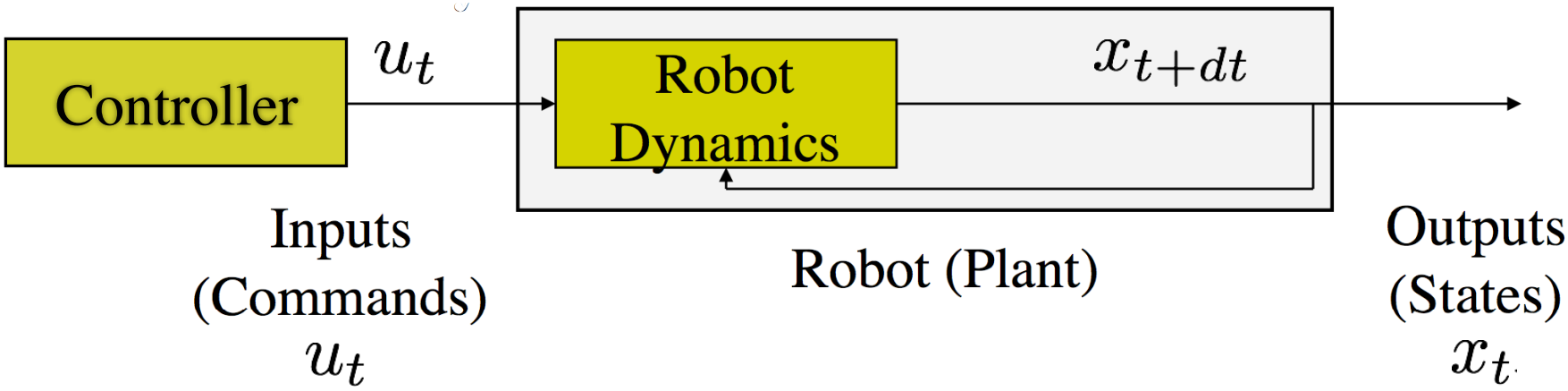
Inputs (Commands) u_t

Robot (Plant)

Outputs (States) x_t

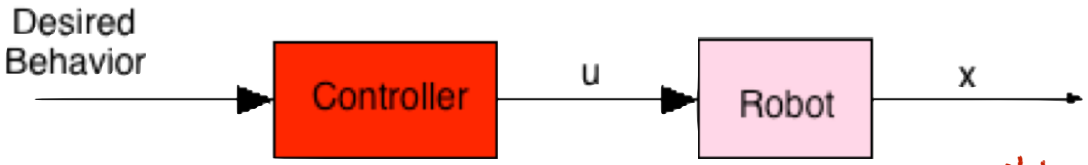
Controller generates motor commands based on desired state

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



Acts without regard to environment

Open Loop Control

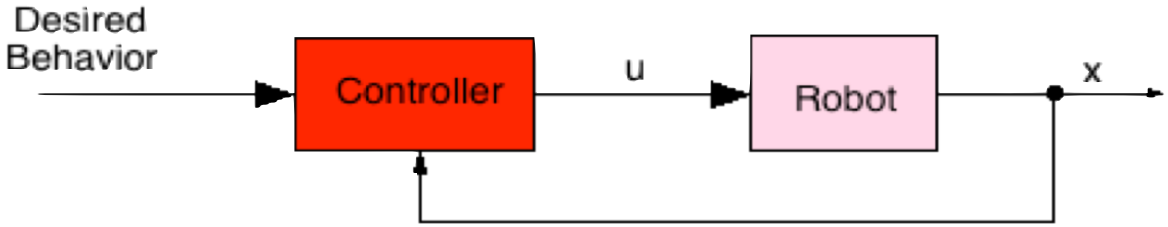


possible controller parameters

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

possible time dependence

Closed Loop Control

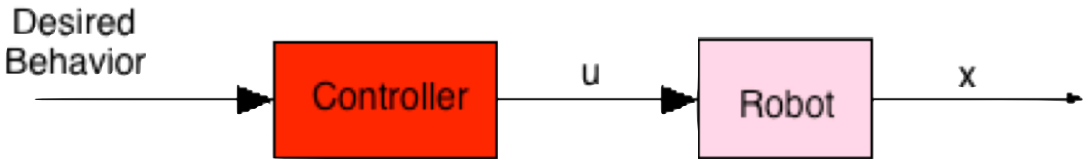


$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$

Control responds to environment

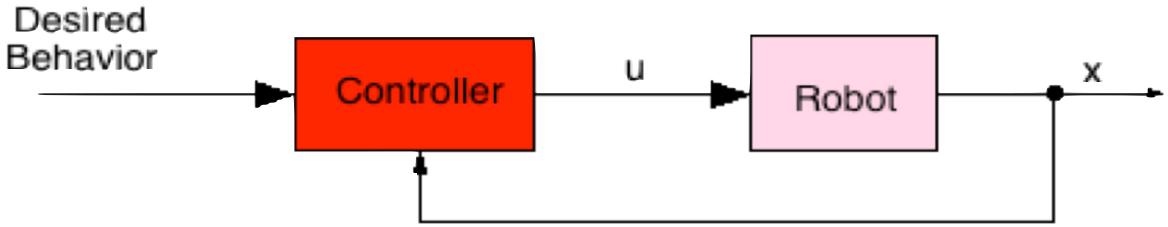
When to use open loop vs closed loop?

Open Loop Control

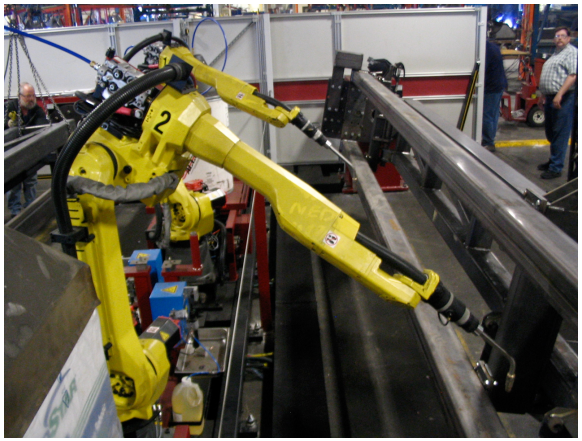


$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t) = \pi(\alpha, t)$$

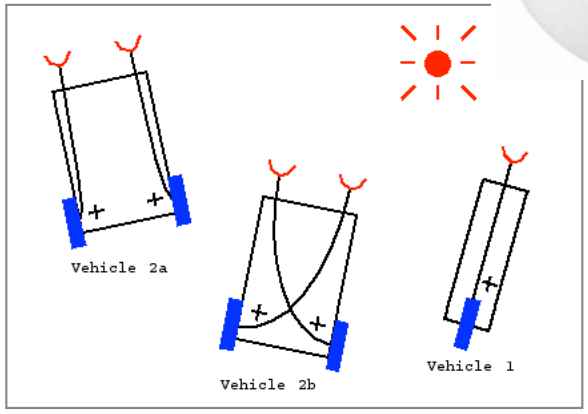
Closed Loop Control



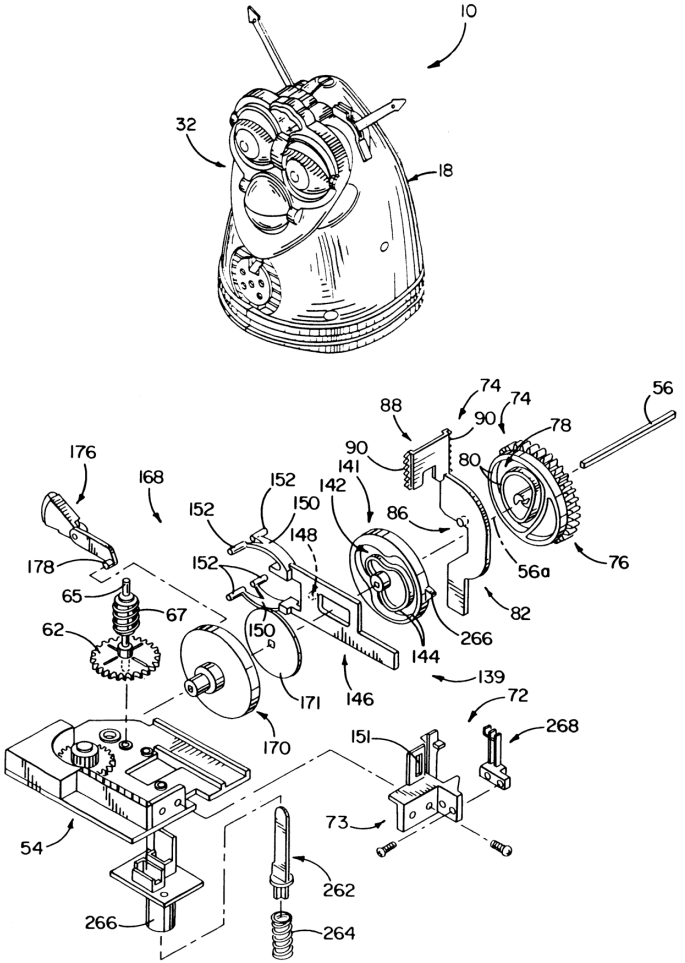
$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$



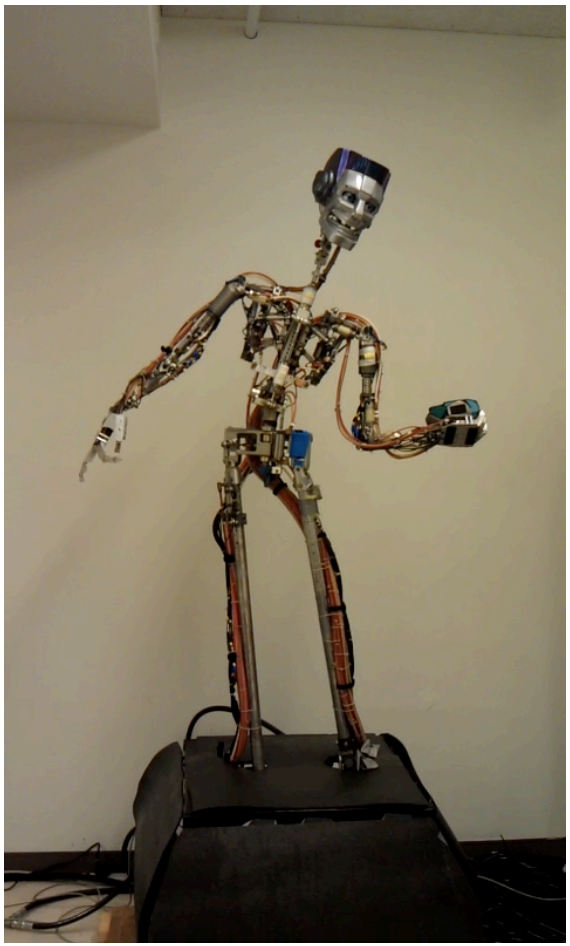
Which uses open loop?



Open loop: Furby



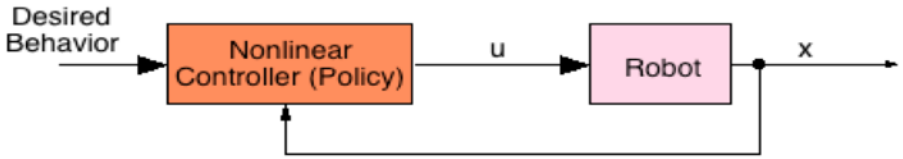
Hasbro



Disney

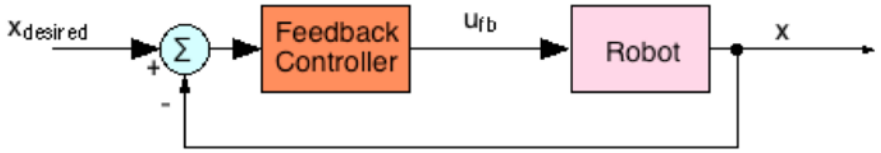
Types of Feedback Control

Feedback Control



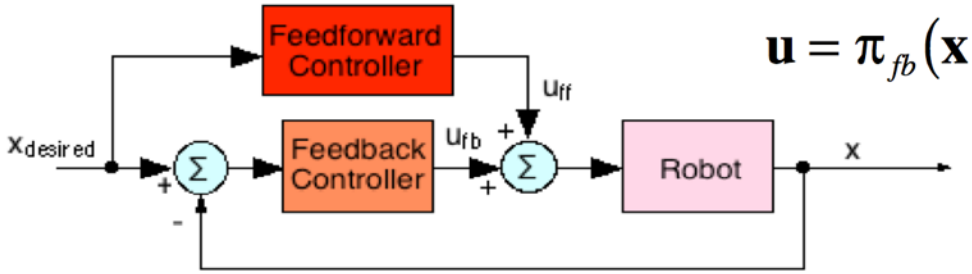
$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$

Negative Feedback Control



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

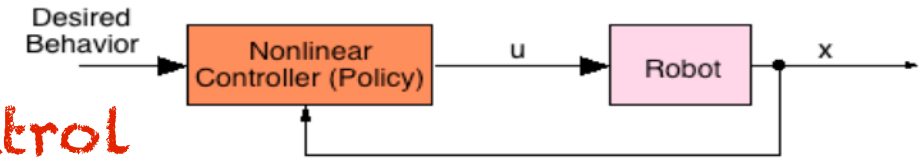
Neg. Feedback & Feedforward Control



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

Types of Feedback Control

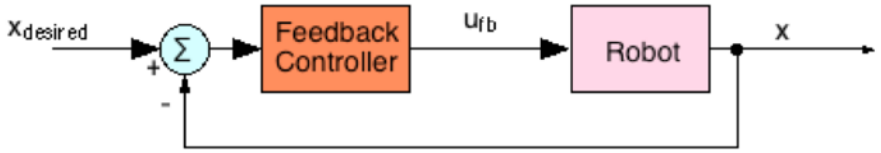
Feedback Control



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

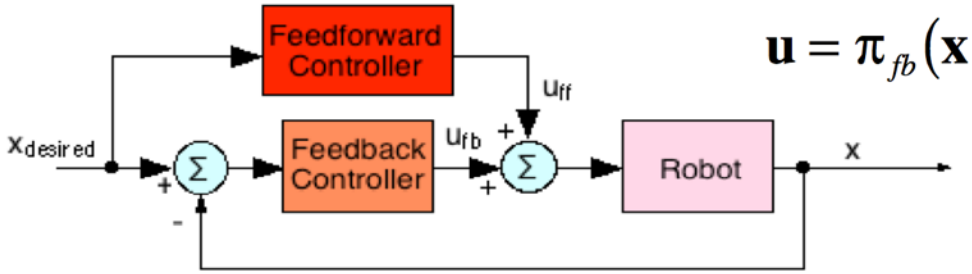
PID Control

Negative Feedback Control



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

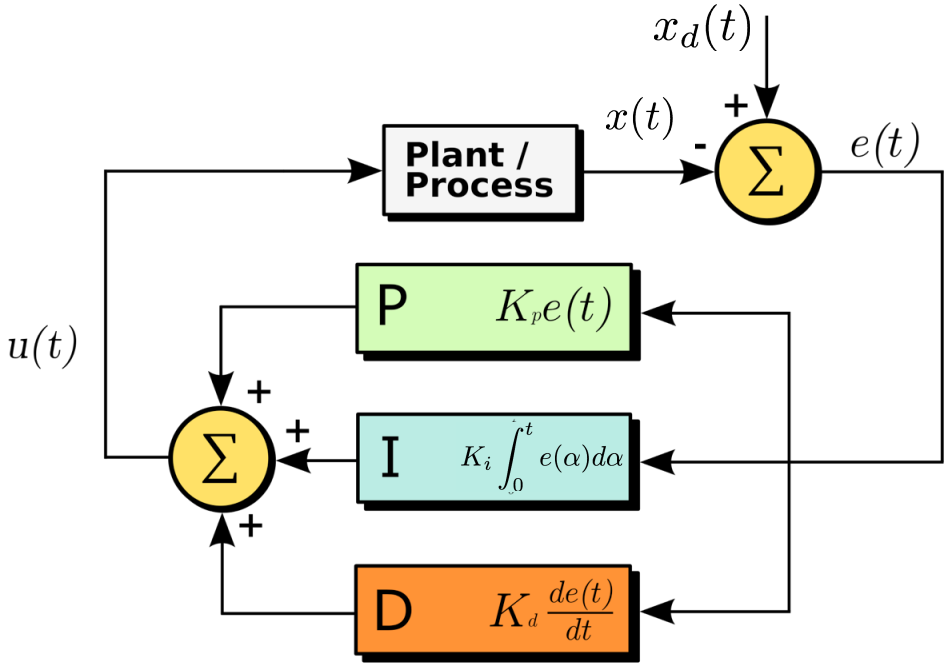
Neg. Feedback & Feedforward Control



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

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

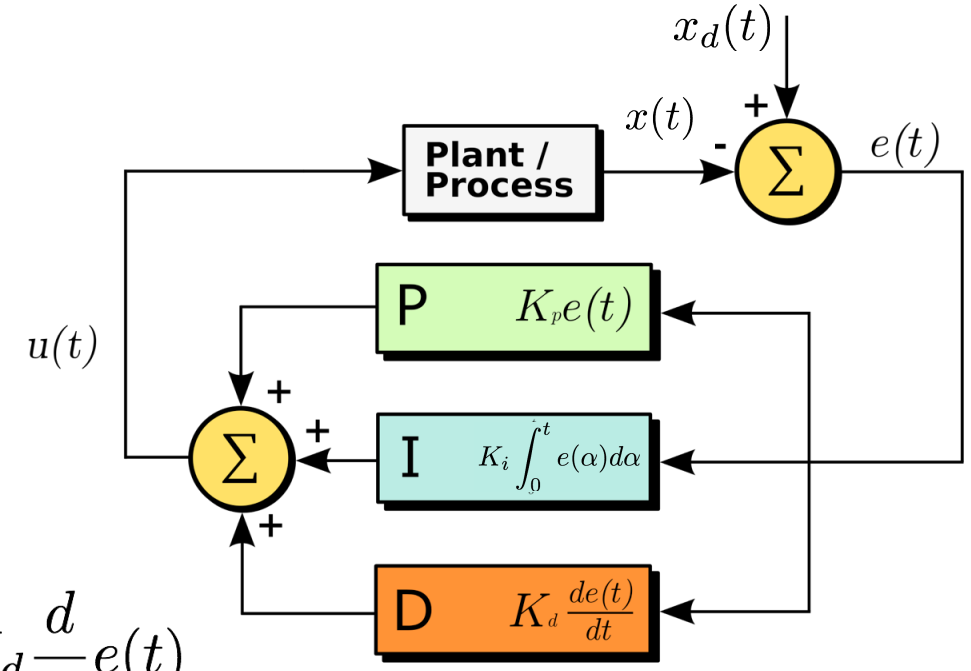


PID Control

for pendulum (x is Θ)

Error signal:

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



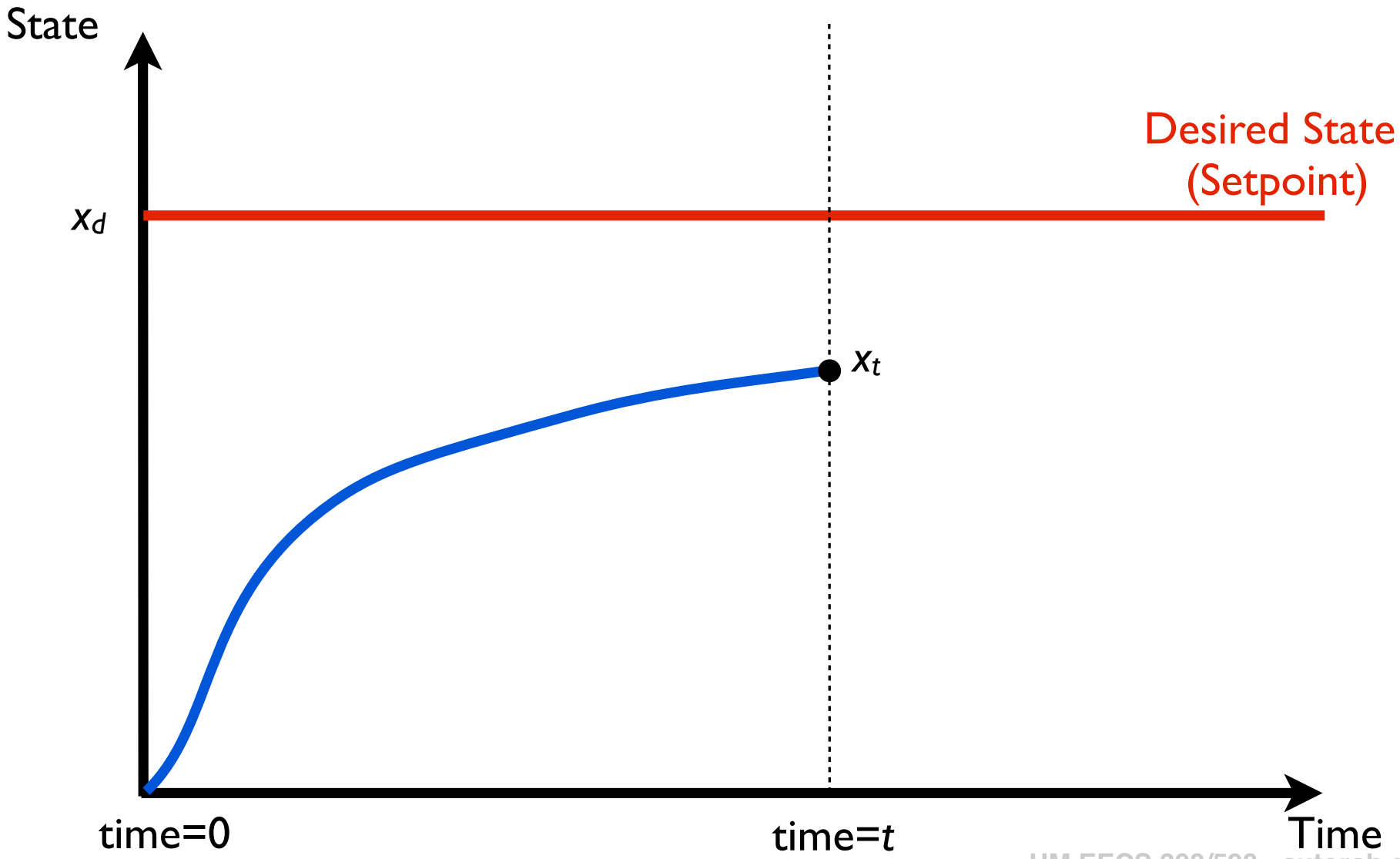
Control signal:

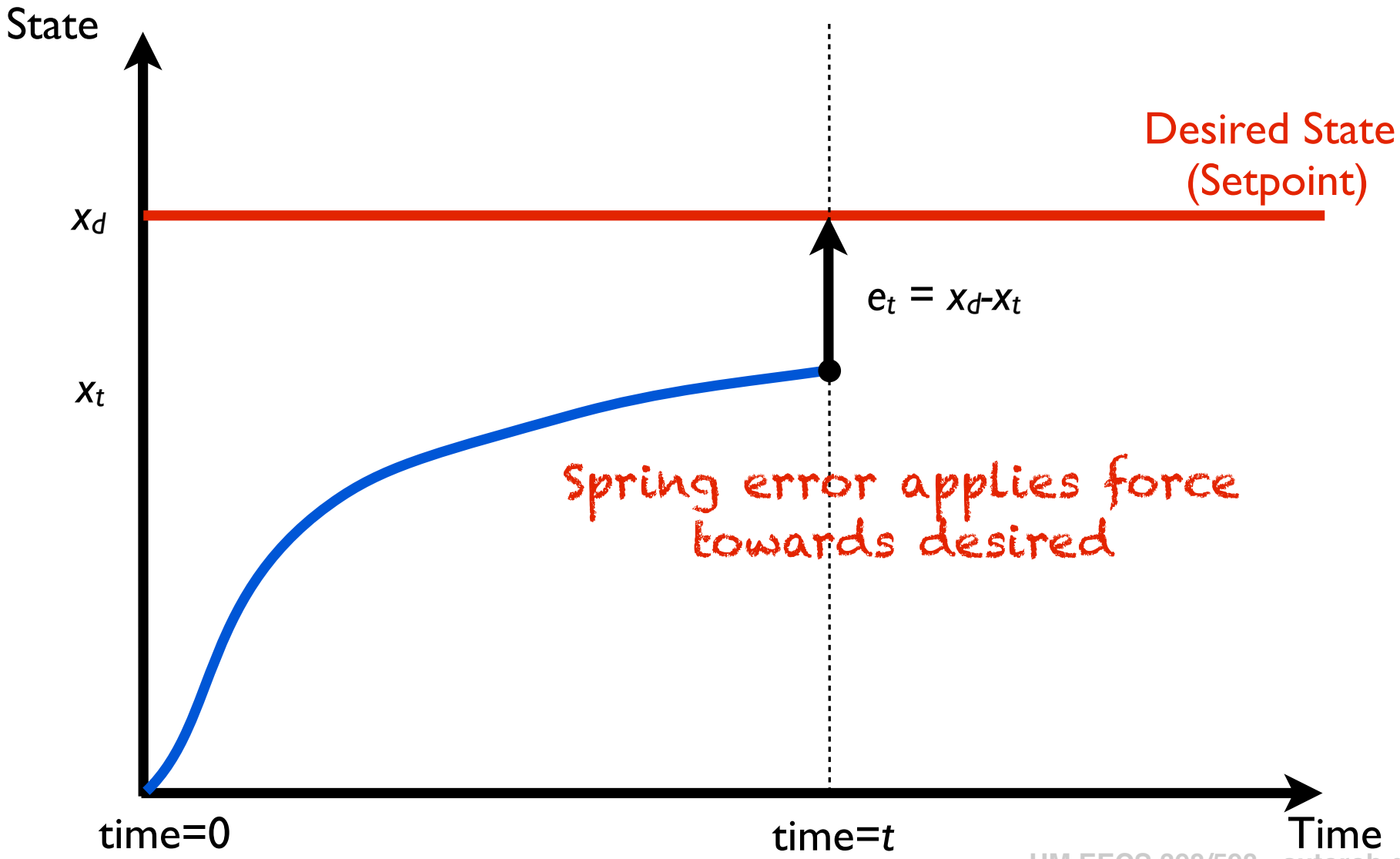
$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha)d\alpha + K_d \frac{d}{dt} e(t)$$

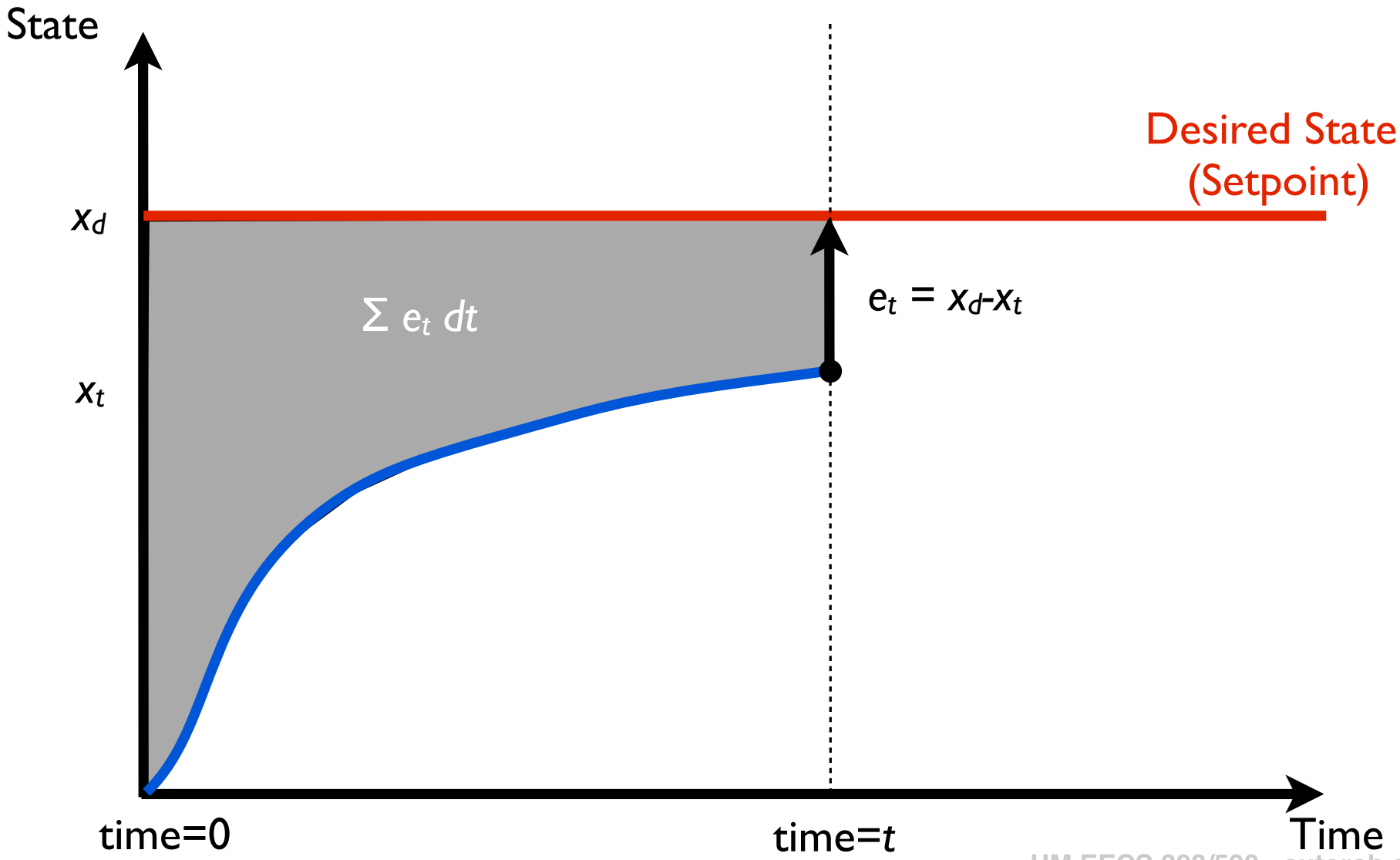
P $K_p e(t)$ — Current

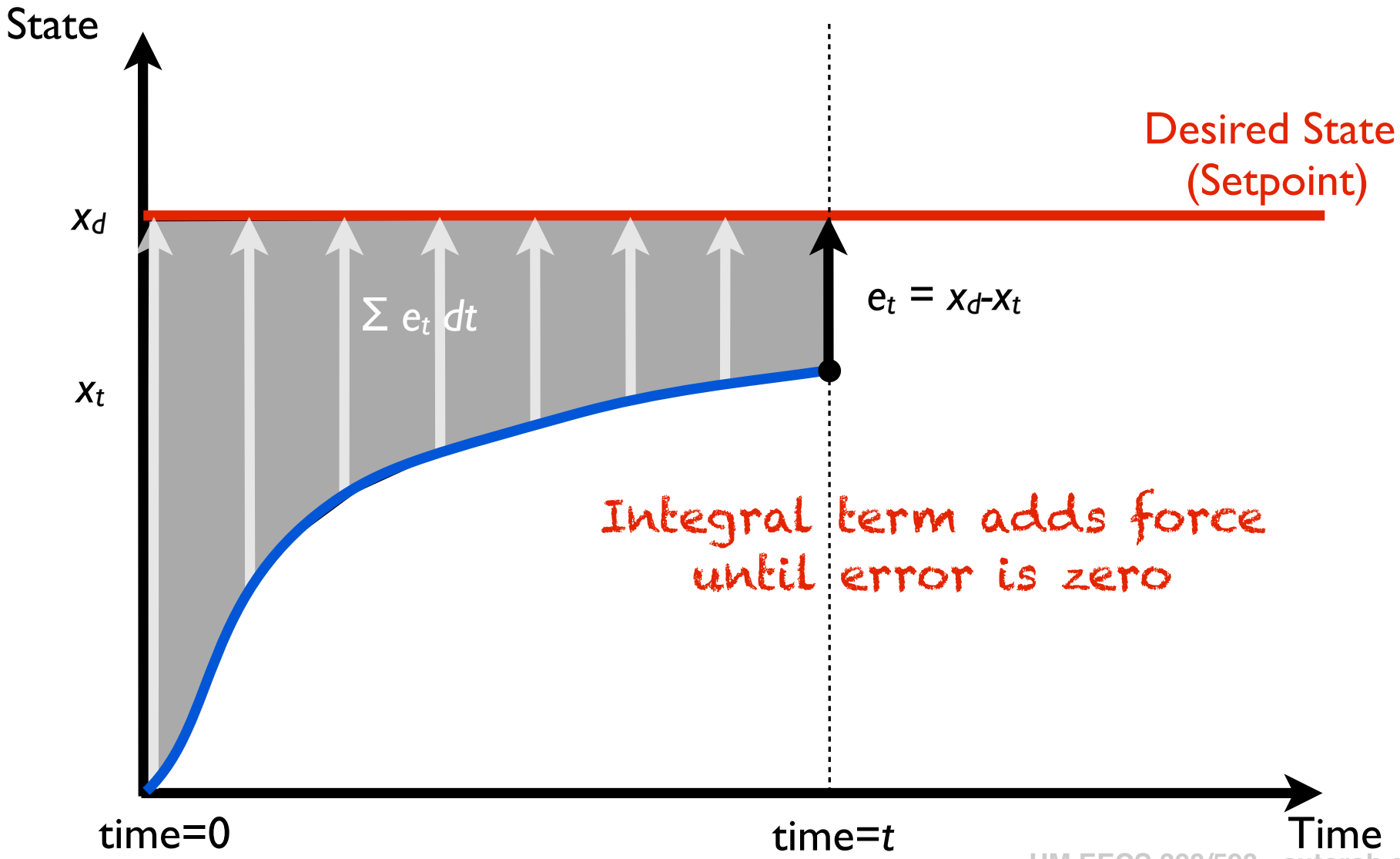
I $K_i \int_0^t e(\alpha)d\alpha$ — Past

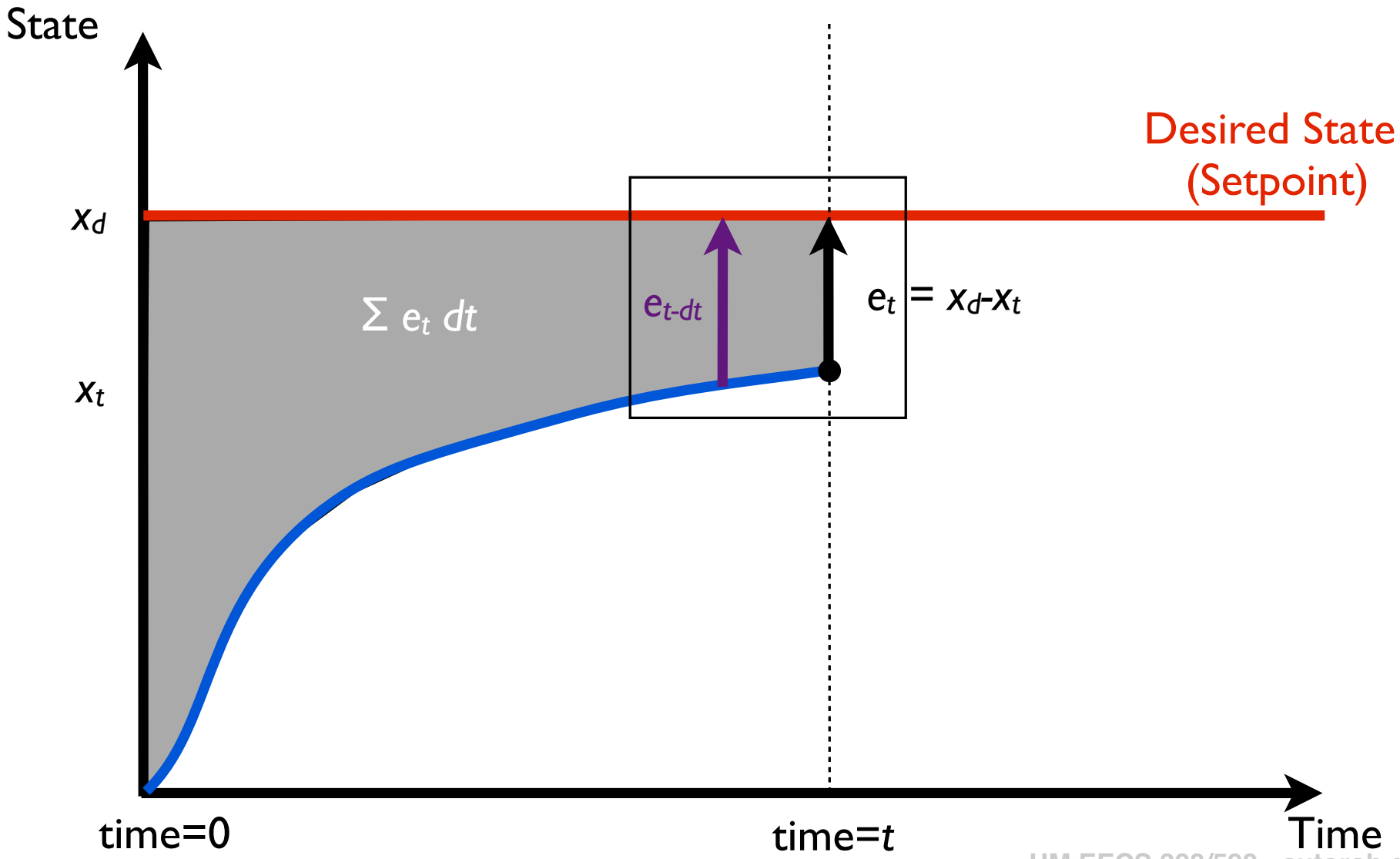
D $K_d \frac{de(t)}{dt}$ — Future

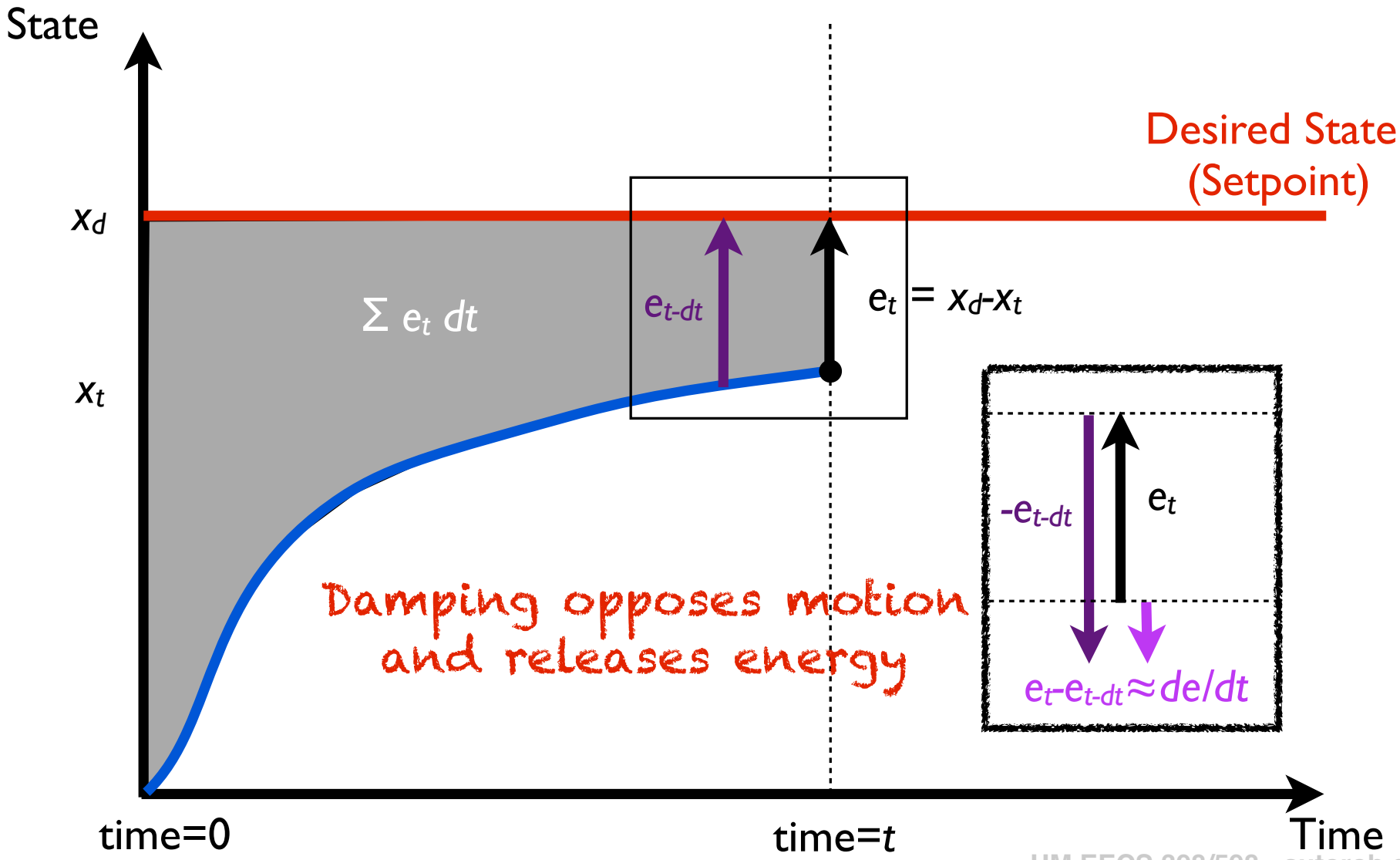

State
x_d
Desired State
(Setpoint)
x_t
time=0
time=t
Time


State
x_d
x_t
Desired State
(Setpoint)
$e_t = x_d - x_t$
Spring error applies force
towards desired
time=0
time=t
Time

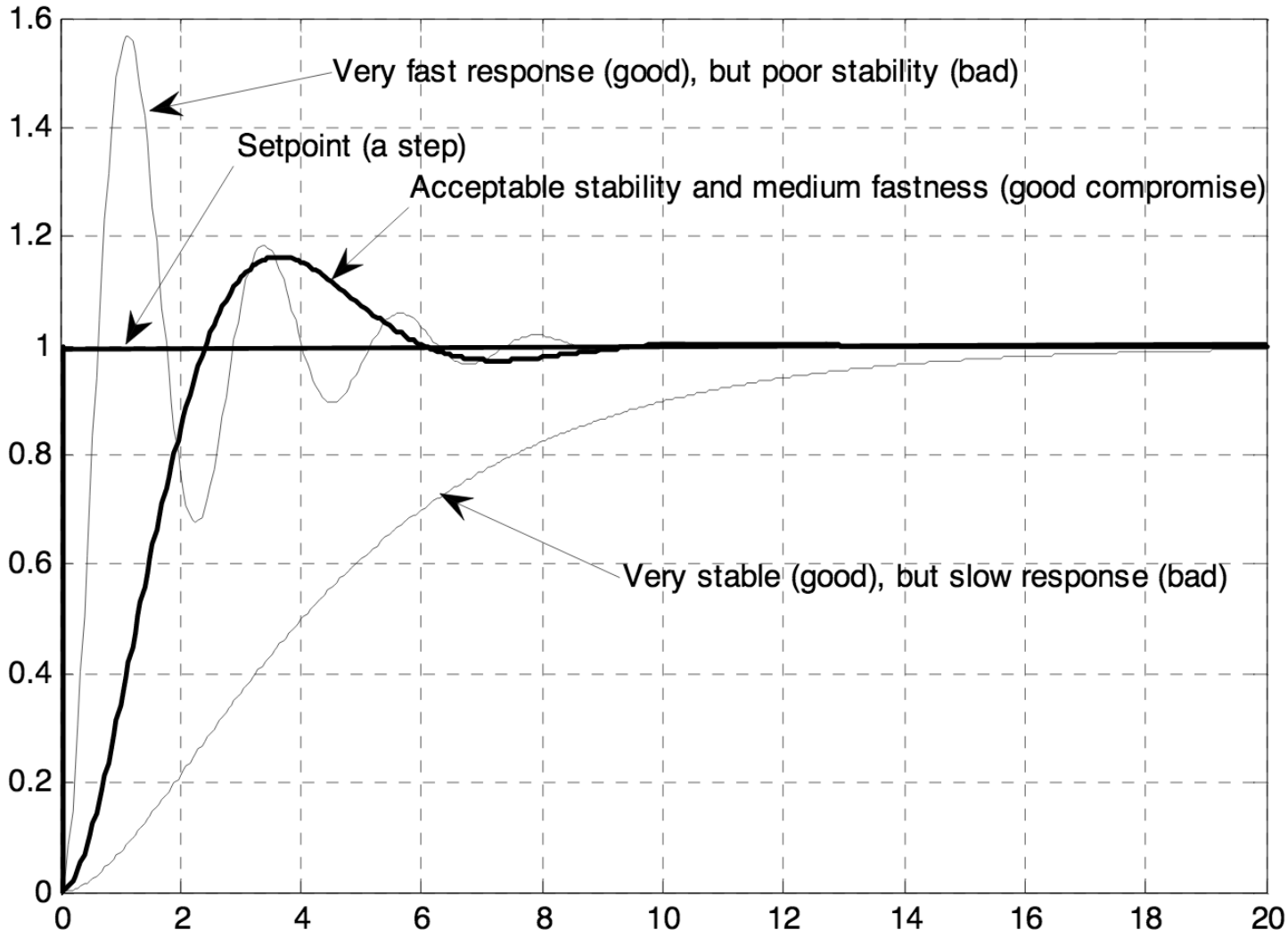

State
x_d
x_t
$\Sigma\ e_t\ dt$
Desired State
(Setpoint)
$e_t = x_d - x_t$
time=0
time=t
Time


State
x_d
x_t
Desired State
(Setpoint)
$\Sigma\ e_t\ dt$
$e_t = x_d - x_t$
Integral term adds force
until error is zero
time=0
time=t
Time


State
x_d
x_t
$\Sigma\ e_t\ dt$
e_{t-dt}
$e_t = x_d - x_t$
Desired State
(Setpoint)
time=0
time=t
Time


State
x_d
x_t
Desired State
(Setpoint)
$\Sigma\ e_t\ dt$
e_{t-dt}
$e_t = x_d - x_t$
$-e_{t-dt}$
e_t
$e_t - e_{t-dt} \approx de/dt$
Damping opposes motion
and releases energy
time=0
time=t
Time

PID Converenge

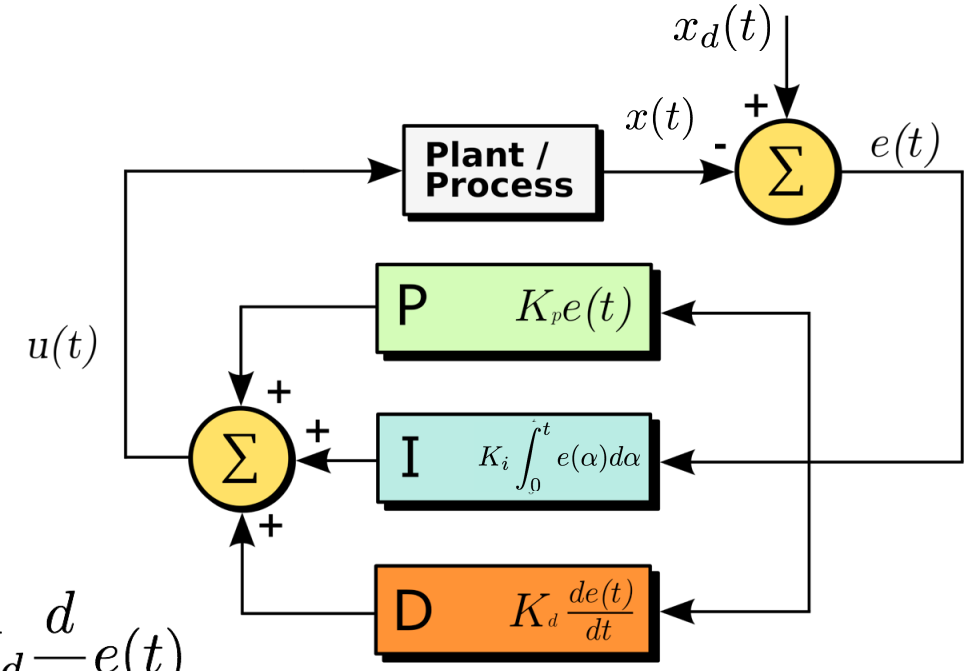


http://home.hit.no/~hansha/documents/control/theory/tuning_pid_controller.pdf

PID Control

Error signal:

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



Control signal:

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

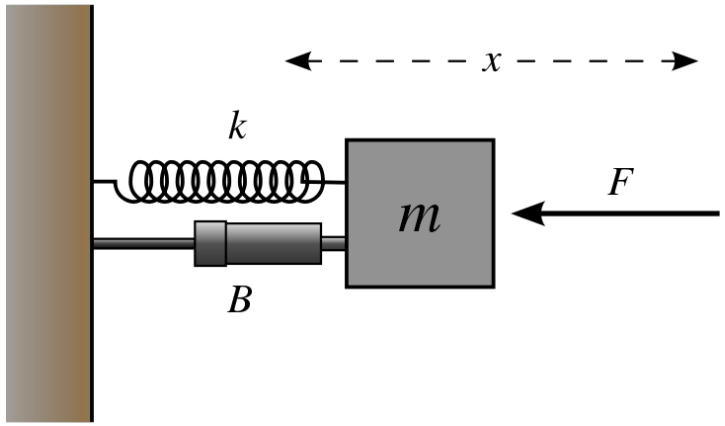
P $K_p e(t)$ — Current

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

D $K_d \frac{de(t)}{dt}$ — Future

Hooke's Law

- Describes motion of mass spring damper system as



$$F = -kx$$

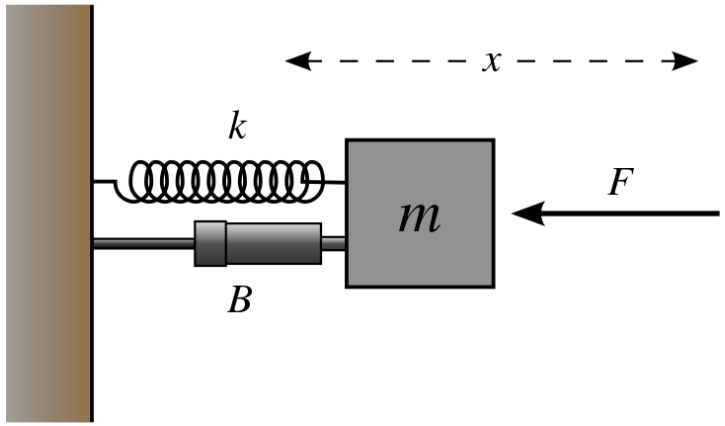
force moving spring towards rest

spring stiffness

distance from rest displacement

Hooke's Law

- Describes motion of mass spring damper system as



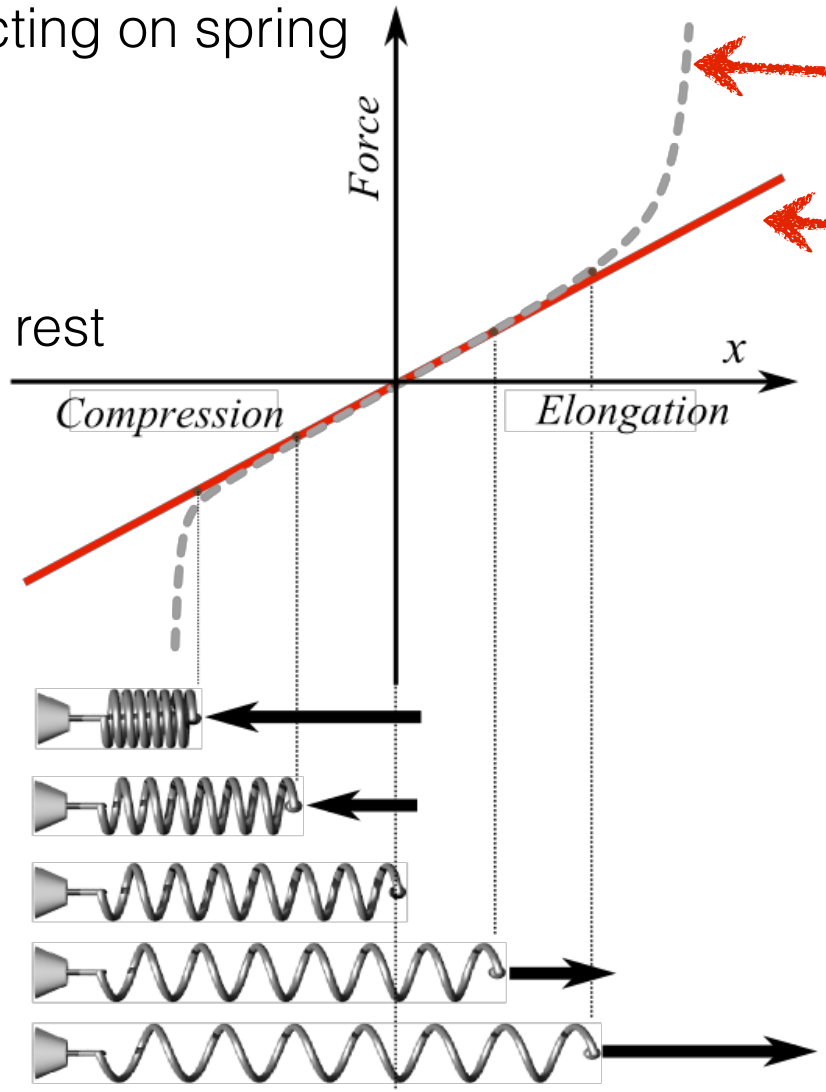
$$F = -kx$$

Proportional Control ("Position Error")

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

Vertical: Force acting on spring

Horizontal: Position from rest



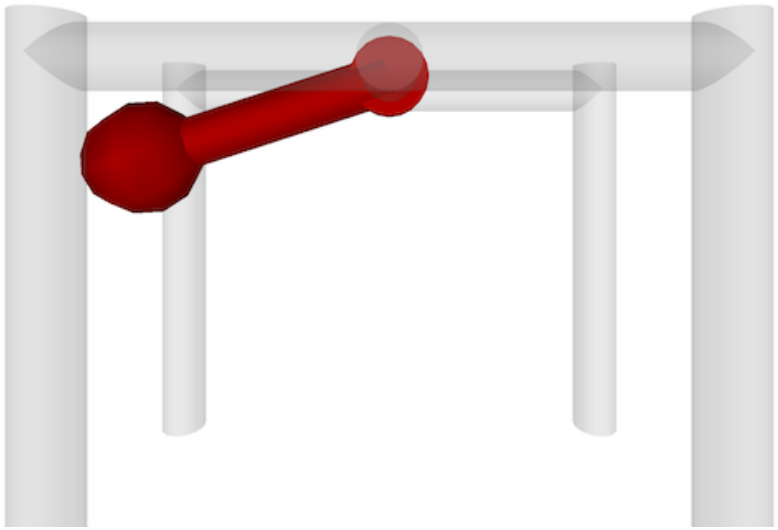
actual force relation

linear approximation (-kx)

$K_p e(t)$


overstiff (k=1.6)
reasonable (k=1.1)
understiff (k=0.5)
Reference
K= 0.5
K =1.1
K = 1.6
Position
Time
2.1
1.6
1.1
0.6
0.1
0
1
2
3
4
5
6
7
8
9
10

Let's see what happens

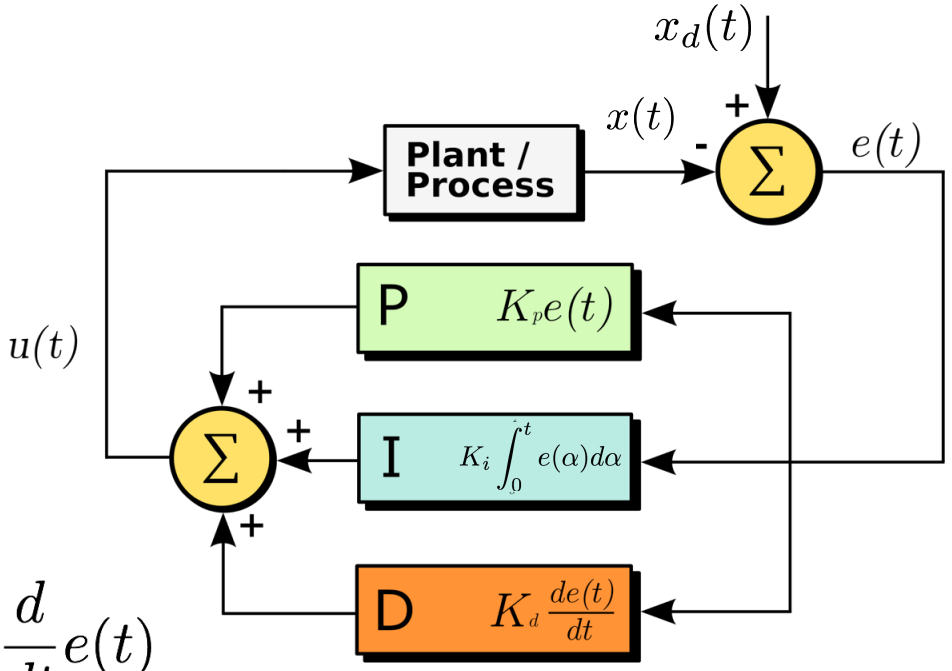


overstiff
understiff

PID Control

Error signal:

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



Control signal:

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

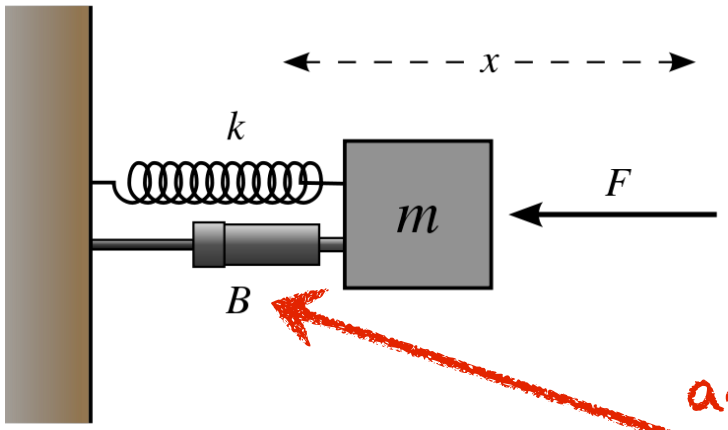
P $K_p e(t)$ — Current

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

D $K_d \frac{de(t)}{dt}$ — Future

Spring and Damper

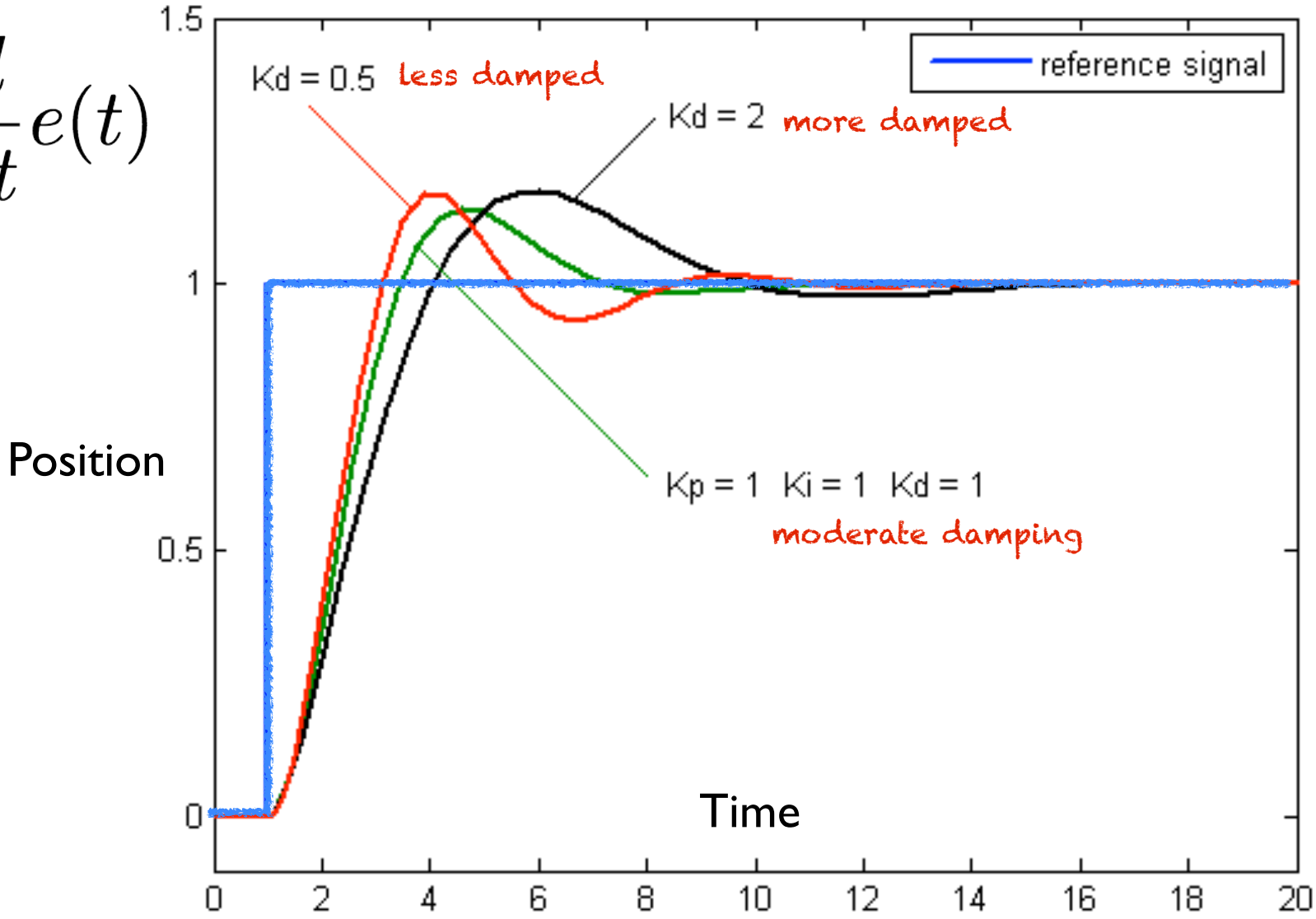
$$F = -kx + -b\dot{x}$$



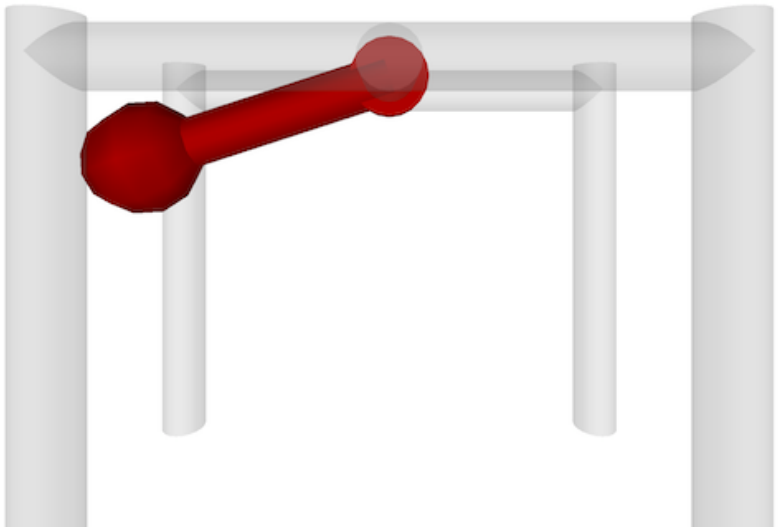
assuming constant set point, velocity is derivative of error

add damper to release energy

$$K_d \frac{d}{dt} e(t)$$



Let's see what happens

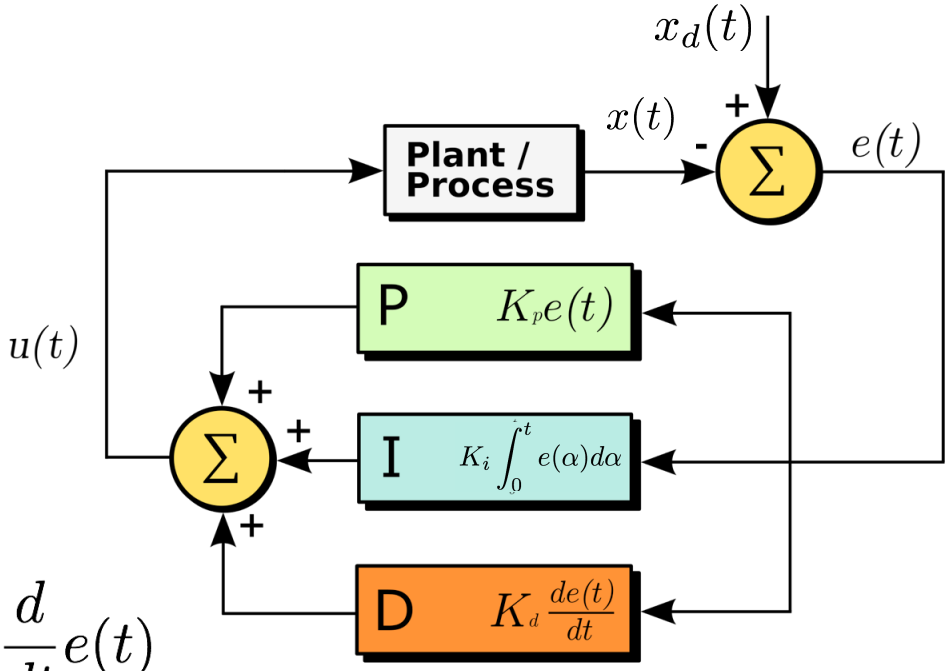


steadystate

PID Control

Error signal:

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



Control signal:

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

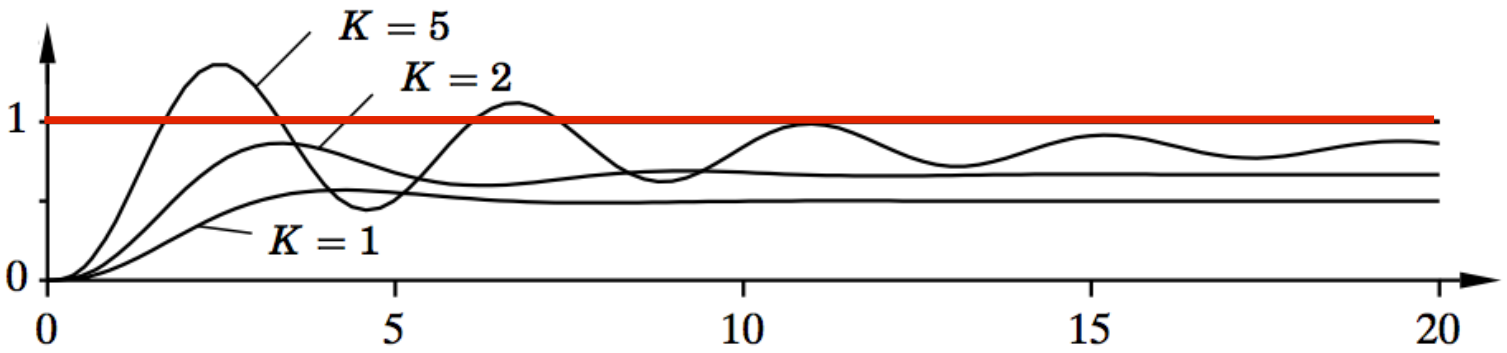
P $K_p e(t)$ — Current

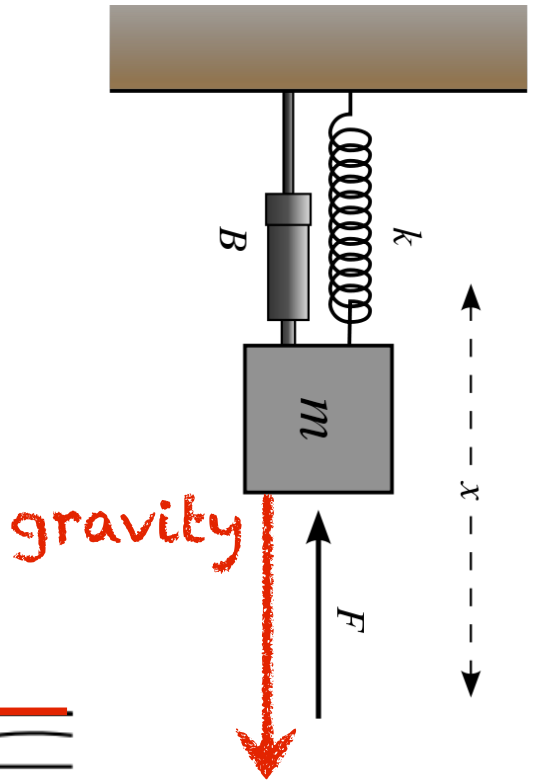
I $K_i \int_0^t e(\alpha)d\alpha$ — Past

D $K_d \frac{de(t)}{dt}$ — Future

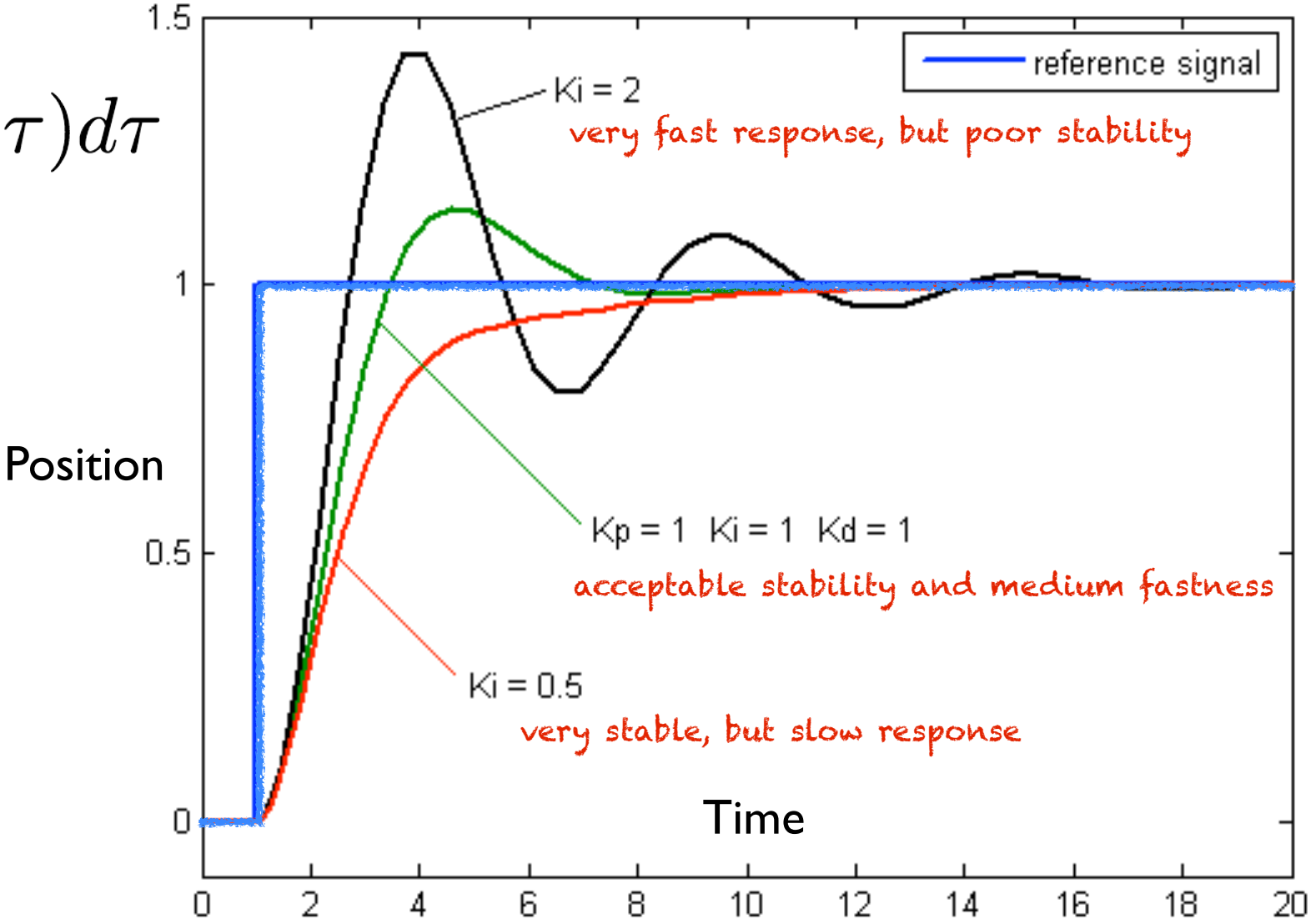
Steady state error

- 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

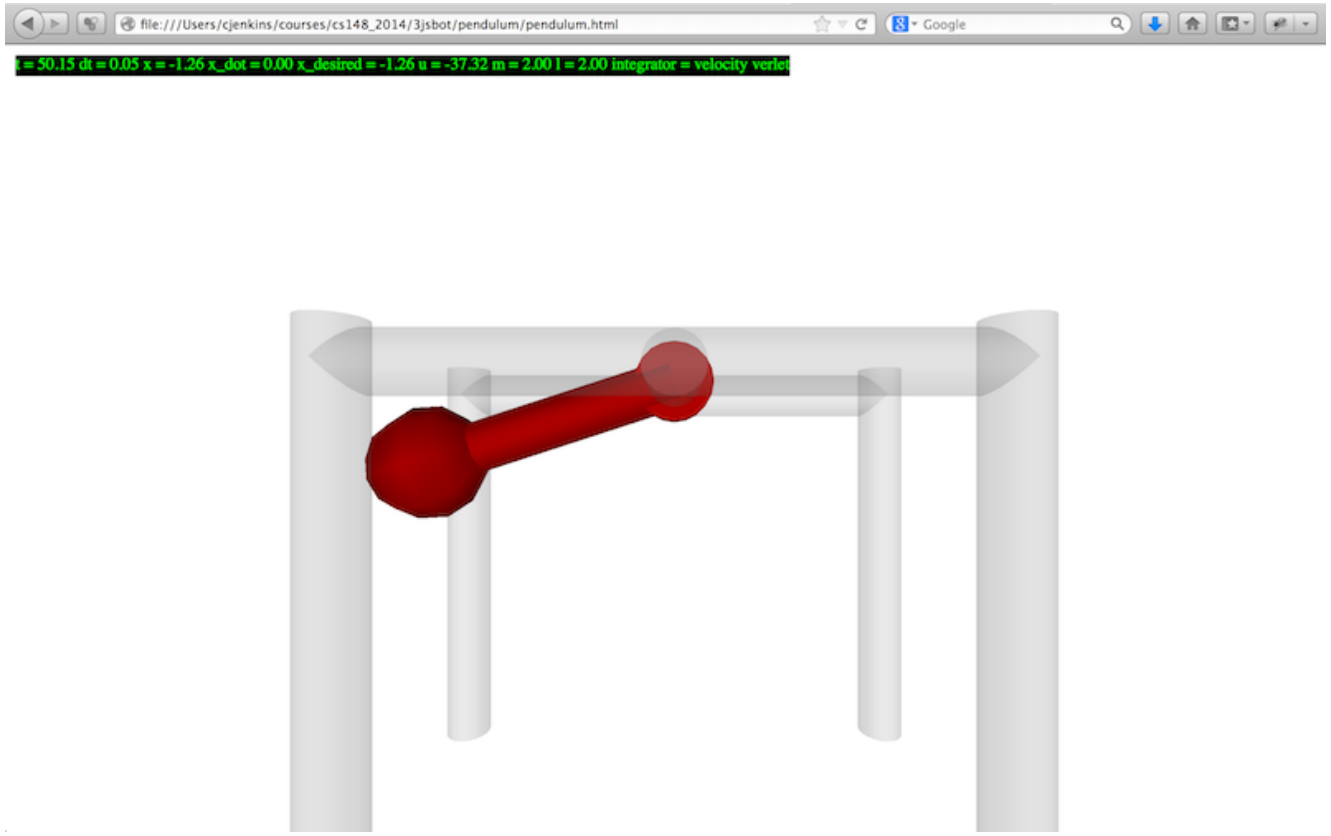




$$K_i \int_0^t e(\tau)d\tau$$



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.

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

P $K_p e(t)$

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

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

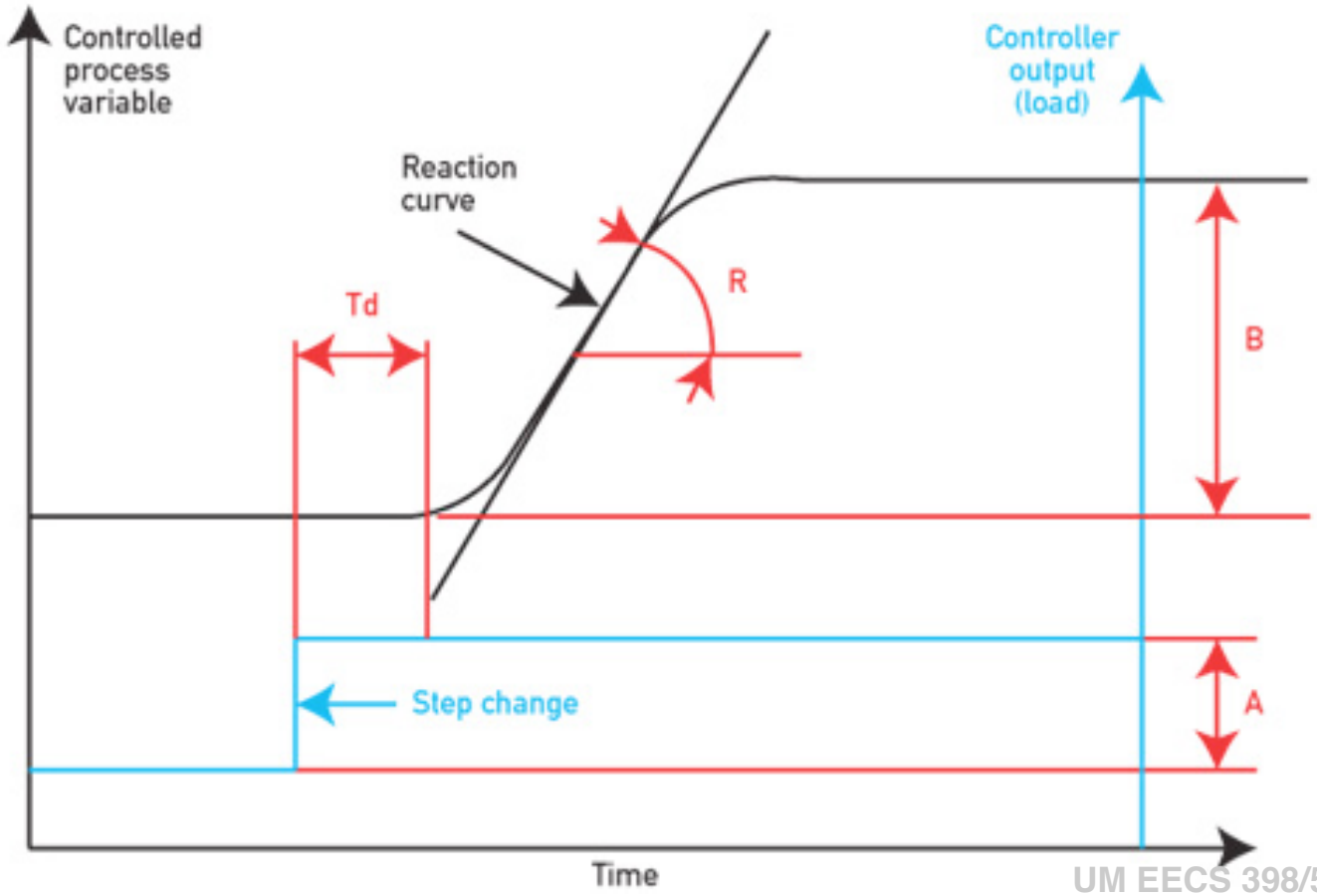
PID controller

From Wikipedia, the free encyclopedia

Effects of *increasing* a parameter independently[14]

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability[11]
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



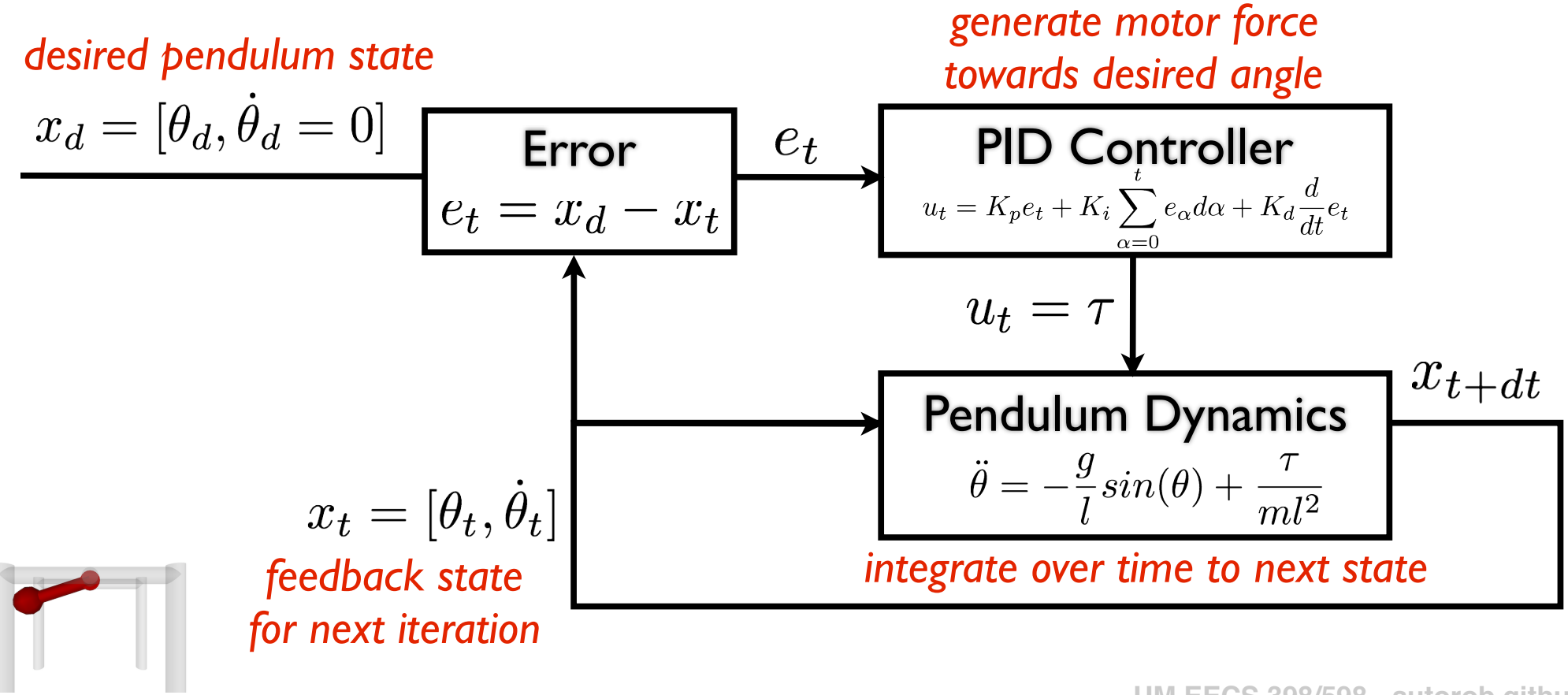
Explore on your own

My approach to PID tuning

(take it or leave it)

- Start will 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 (Pendularm)

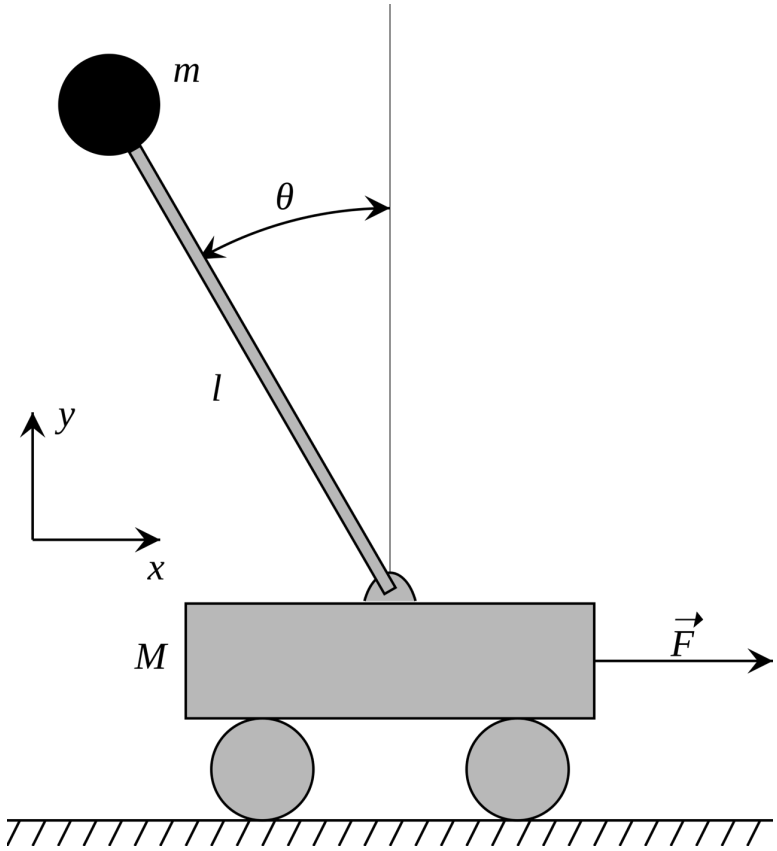


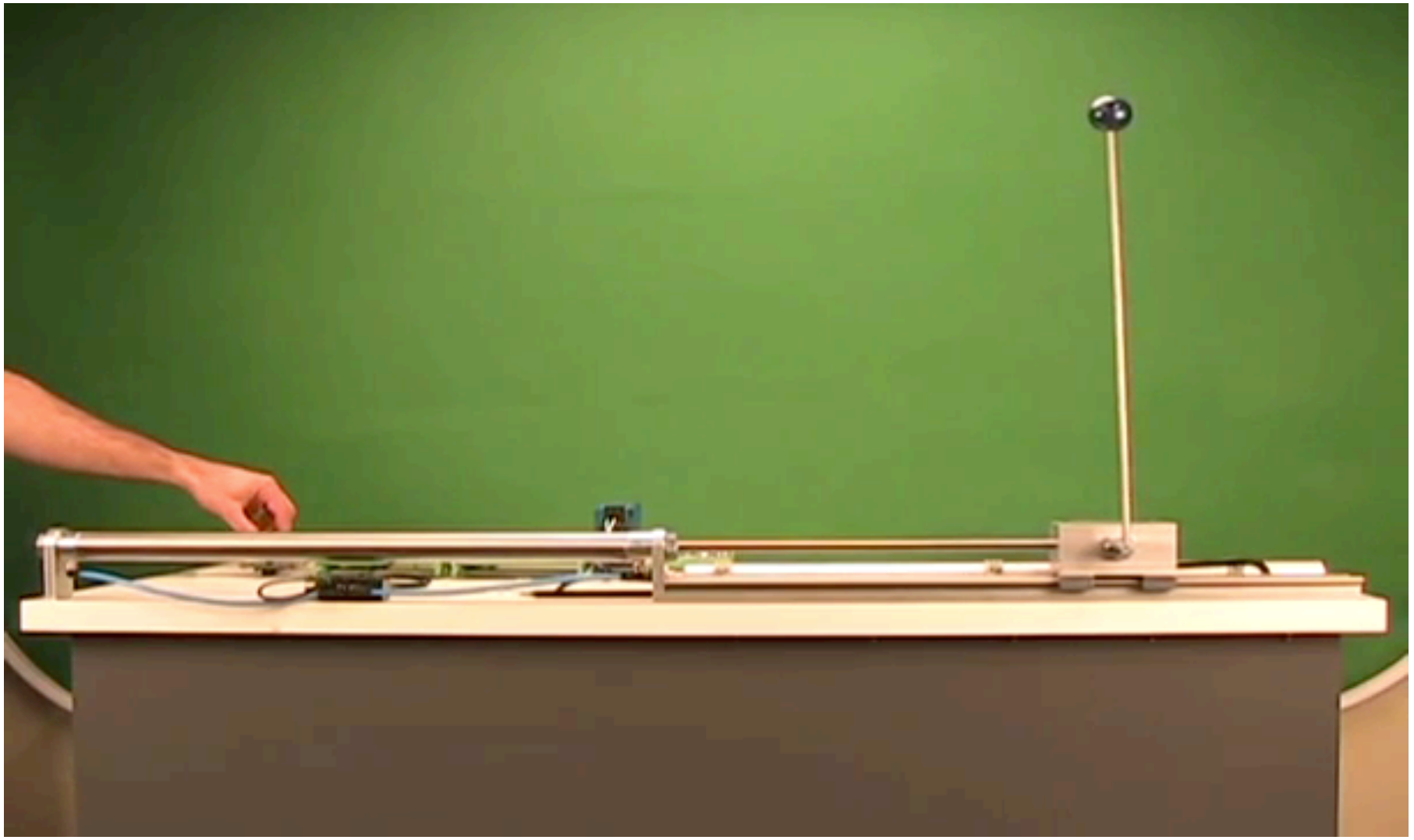
Other types of systems?

Cart Pole Balancer

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

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

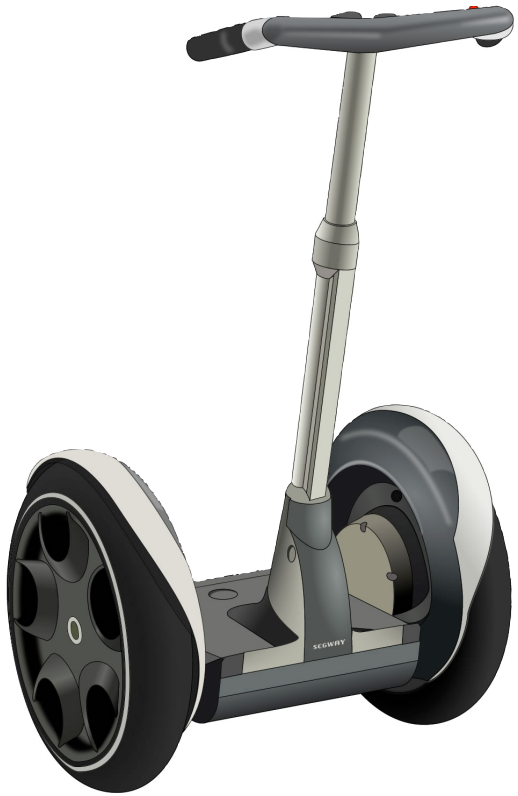




https://www.youtube.com/watch?v=a4c7AwHFkT8

Enfield Technologies

Differential Drive Inverted Pendulum



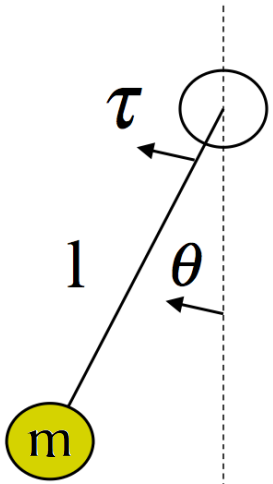
https://www.youtube.com/watch?v=hpCwdu0e3i0

techbricks.nl

Robot links are rigid bodies

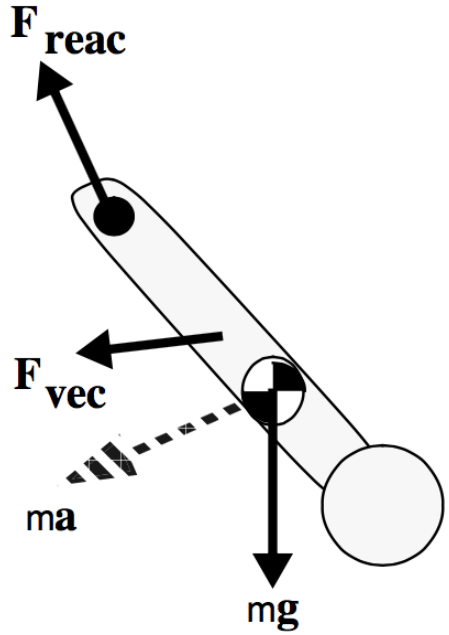
Pendulum as a particle

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, ***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 (ω), 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

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

  - and with respect to an arbitrary frame of reference

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

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

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

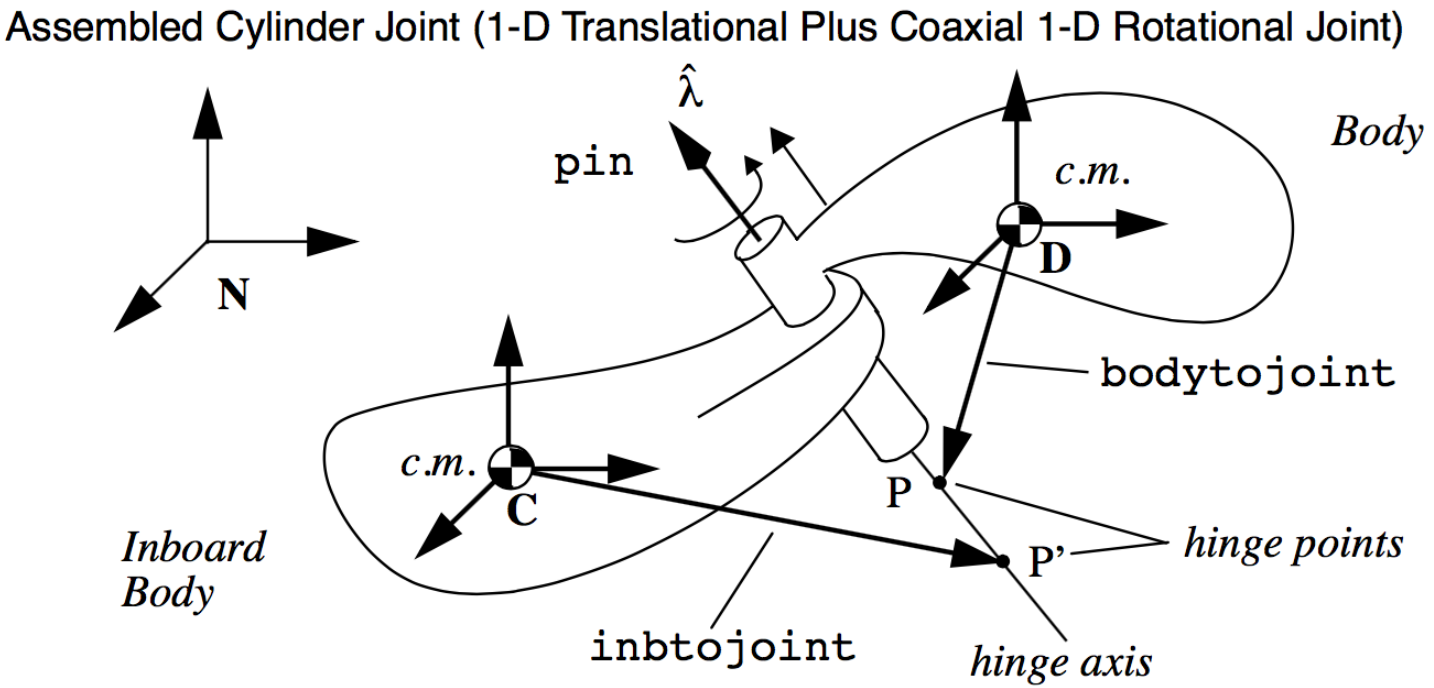
Cross product by skew-symmetric matrix

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

  - and with respect to an arbitrary frame of reference

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

Joints between rigid bodies



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

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

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

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

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

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

gravitational forces

frictional forces

Centripetal and Coriolis forces

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

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

"mass times acceleration"
M: Inertia matrix
(NxN positive definite)
q, $\dot{q}$, $\ddot{q}$: state/velocity/acceleration
in generalized coordinates
(each N-dimensional vectors)

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

General Equations of Motion

gravitational forces | frictional forces | Centripetal and Coriolis forces

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

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)

"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
- Newton-Euler recursive forward-backward algorithm
  - equivalent to Lagrangian formulation, but very different approach
  - based on balanced forces between links connected by joints
  - propagate state variables from base to endeffector, then return with forces

Just a few options

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0, \alpha_0 = 0, \boldsymbol{a}_{c,0} = 0, \boldsymbol{a}_{e,0} = 0 \tag{9.162}$$

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

2. Start with the terminal conditions

$$\boldsymbol{f}_{n+1} = 0, \qquad \boldsymbol{\tau}_{n+1} = 0 \tag{9.163}$$

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

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0, \alpha_0 = 0, \boldsymbol{a}_{c,0} = 0, \boldsymbol{a}_{e,0} = 0 \tag{9.162}$$

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

$$\boldsymbol{\omega}_i = (R^i_{i-1})^T \boldsymbol{\omega}_{i-1} + \boldsymbol{b}_i \dot{q}_i$$

compute angular velocity for link$_i$ from link$_{i-1}$

2. Start with the terminal conditions

$$\boldsymbol{f}_{n+1} = 0, \qquad \boldsymbol{\tau}_{n+1} = 0 \tag{9.163}$$

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

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0, \alpha_0 = 0, \boldsymbol{a}_{c,0} = 0, \boldsymbol{a}_{e,0} = 0 \tag{9.162}$$

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

$$\boldsymbol{\omega}_i = (R^i_{i-1})^T \boldsymbol{\omega}_{i-1} + \boldsymbol{b}_i \dot{q}_i$$

$$\boldsymbol{\alpha}_i = (R^i_{i-1})^T \boldsymbol{\alpha}_{i-1} + \boldsymbol{b}_i \ddot{q}_i + \boldsymbol{\omega}_i \times \boldsymbol{b}_i \dot{q}_i.$$

compute angular acceleration for link$_i$ from link$_{i-1}$

2. Start with the terminal conditions

$$\boldsymbol{f}_{n+1} = 0, \qquad \boldsymbol{\tau}_{n+1} = 0 \tag{9.163}$$

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

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0, \alpha_0 = 0, \boldsymbol{a}_{c,0} = 0, \boldsymbol{a}_{e,0} = 0 \tag{9.162}$$

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

$$\boldsymbol{\omega}_i = (R^i_{i-1})^T \boldsymbol{\omega}_{i-1} + \boldsymbol{b}_i \dot{q}_i$$

$$\boldsymbol{\alpha}_i = (R^i_{i-1})^T \boldsymbol{\alpha}_{i-1} + \boldsymbol{b}_i \ddot{q}_i + \boldsymbol{\omega}_i \times \boldsymbol{b}_i \dot{q}_i$$

compute acceleration at COM

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

2. Start with the terminal conditions

$$\boldsymbol{f}_{n+1} = 0, \qquad \boldsymbol{\tau}_{n+1} = 0 \tag{9.163}$$

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

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0, \alpha_0 = 0, \boldsymbol{a}_{c,0} = 0, \boldsymbol{a}_{e,0} = 0 \tag{9.162}$$

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

compute acceleration at end of link

$$\boldsymbol{\omega}_i = (R^i_{i-1})^T \boldsymbol{\omega}_{i-1} + \boldsymbol{b}_i \dot{q}_i$$

$$\boldsymbol{\alpha}_i = (R^i_{i-1})^T \boldsymbol{\alpha}_{i-1} + \boldsymbol{b}_i \ddot{q}_i + \boldsymbol{\omega}_i \times \boldsymbol{b}_i \dot{q}_i$$

$$\boldsymbol{a}_{e,i} = (R^i_{i-1})^T \boldsymbol{a}_{e,i-1} + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i,i+1} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i,i+1})$$

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

2. Start with the terminal conditions

$$\boldsymbol{f}_{n+1} = 0, \qquad \boldsymbol{\tau}_{n+1} = 0 \tag{9.163}$$

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

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\omega_0 = 0, \alpha_0 = 0, a_{c,0} = 0, a_{e,0} = 0 \qquad (9.162)$$

and solve (9.151), (9.155), (9.161) and (9.160) (in that order!) to compute ω_i, α_i and $a_{c,i}$ for i increasing from 1 to n.

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

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

$$a_{e,i} = (R^i_{i-1})^T a_{e,i-1} + \dot{\omega}_i \times r_{i,i+1} + \omega_i \times (\omega_i \times r_{i,i+1})$$

$$a_{c,i} = (R^i_{i-1})^T a_{e,i-1} + \dot{\omega}_i \times r_{i,ci} + \omega_i \times (\omega_i \times r_{i,ci})$$

2. Start with the terminal conditions

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

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

$$f_i = R^{i+1}_i f_{i+1} + m_i a_{c,i} - m_i g_i$$

compute force exerted by link$_{i-1}$ on link$_i$

Newton-Euler recursion (from Spong textbook)

1. Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0, \alpha_0 = 0, \boldsymbol{a}_{c,0} = 0, \boldsymbol{a}_{e,0} = 0 \qquad (9.162)$$

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

$$\boldsymbol{\omega}_i = (R^i_{i-1})^T \boldsymbol{\omega}_{i-1} + \boldsymbol{b}_i \dot{q}_i$$

$$\boldsymbol{\alpha}_i = (R^i_{i-1})^T \boldsymbol{\alpha}_{i-1} + \boldsymbol{b}_i \ddot{q}_i + \boldsymbol{\omega}_i \times \boldsymbol{b}_i \dot{q}_i$$

$$\boldsymbol{a}_{e,i} = (R^i_{i-1})^T \boldsymbol{a}_{e,i-1} + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i,i+1} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i,i+1})$$

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

2. Start with the terminal conditions

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

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

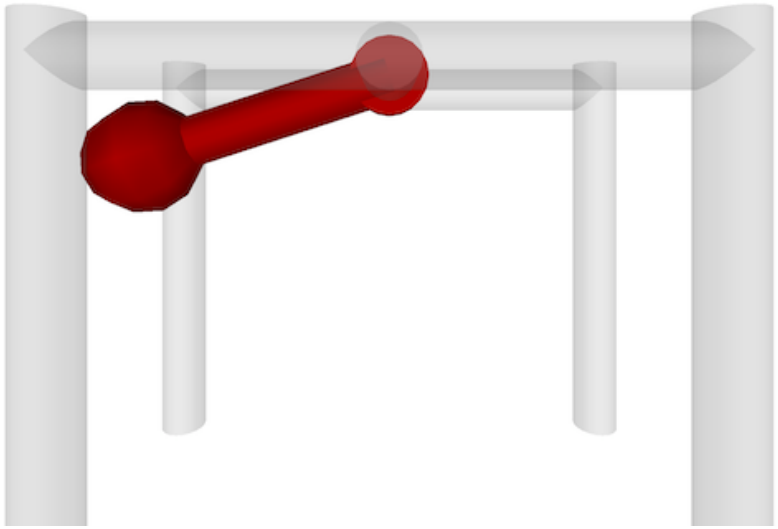
$$\boldsymbol{f}_i = R^{i+1}_i \boldsymbol{f}_{i+1} + m_i \boldsymbol{a}_{c,i} - m_i \boldsymbol{g}_i$$

$$\boldsymbol{\tau}_i = R^{i+1}_i \boldsymbol{\tau}_{i+1} - \boldsymbol{f}_i \times \boldsymbol{r}_{i,ci} + (R^{i+1}_i \boldsymbol{f}_{i+1}) \times \boldsymbol{r}_{i+1,ci} + \boldsymbol{\alpha}_i + \boldsymbol{\omega}_i \times (I_i \boldsymbol{\omega}_i)$$

compute torque exerted by link$_{i-1}$ on link$_i$

Let's see what happens

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



bigger
different

PD as a Linear System

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

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

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

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

PID as a Linear System

- Linearized System

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

general form

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

- The Integral Controller introduces a new state:

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

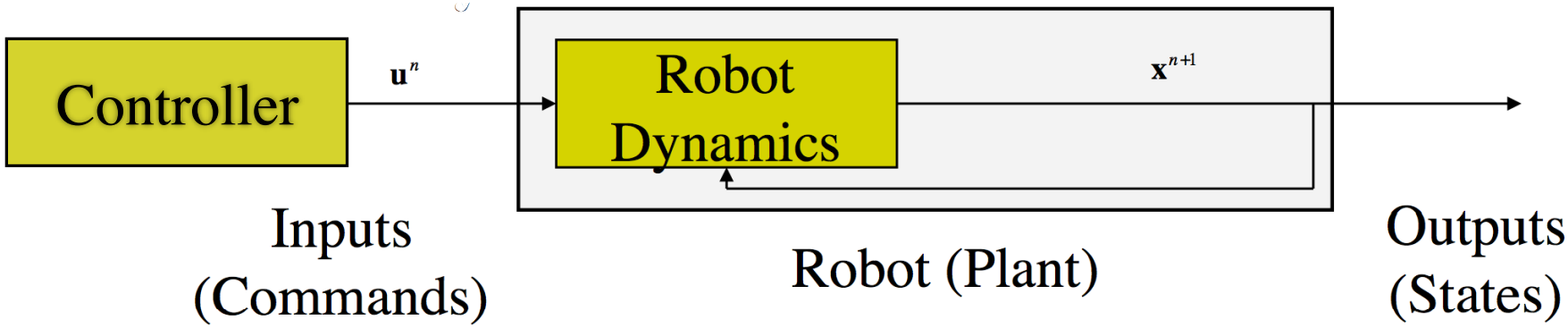
- The new (linearized) system becomes;

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

$$u=\tau=k_P(x_d-x_2)+k_D(0-x_1)+k_I x_3$$

PID does not consider system dynamics

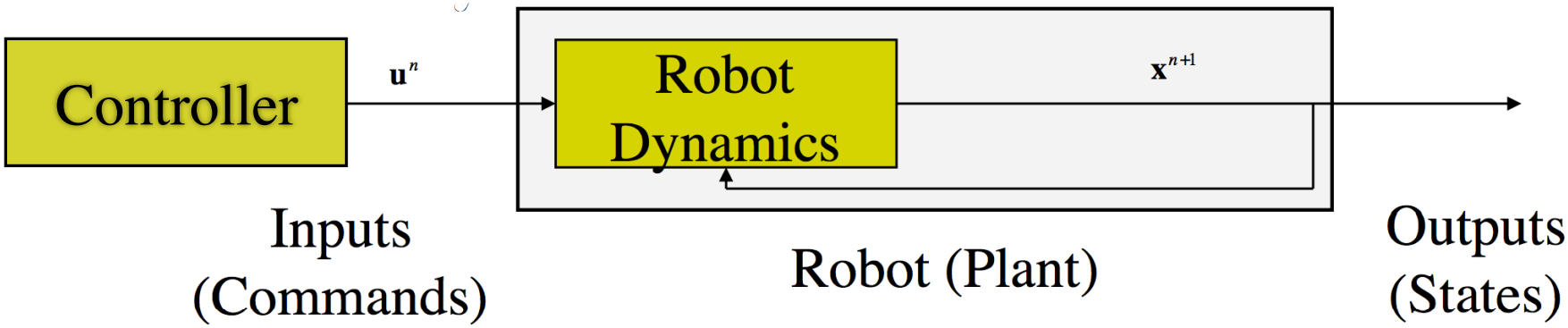
$$u = \pi(t, x, x_d) \qquad \dot{x} = f(t, x, u)$$



How can dynamics be accounted for in control?

forward model

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

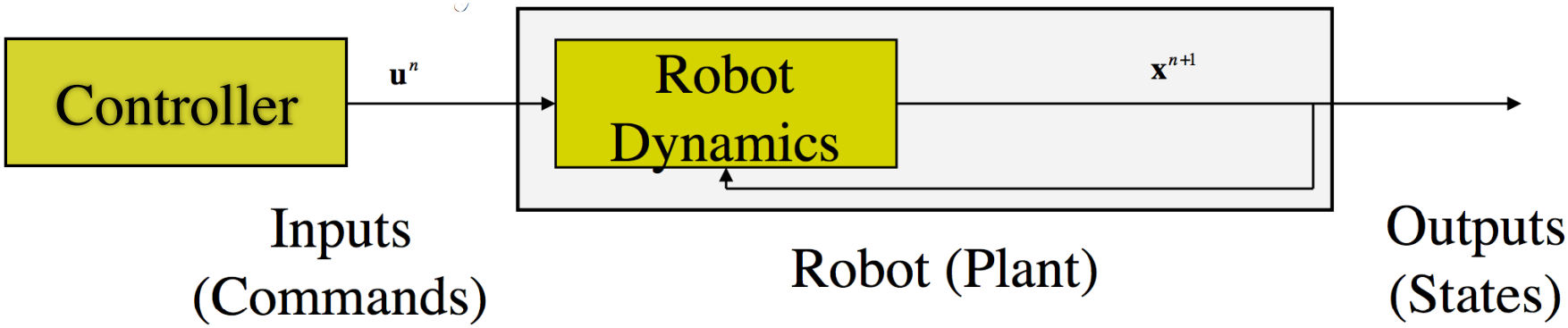


inverse model, if $\dot{x} = f(t, x, \pi(t, x, x_d))$

$u = \pi(t, x, x_d)$

forward model

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



Optimal Control

cost to desired state

Search over controls u that minimize cost

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

dynamics model

subject to the first-order dynamic constraints

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

the algebraic *path constraints*

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

and the boundary conditions

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

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

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

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

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

LQR is common form of optimal control

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 $t_f \to \infty$ (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

$$J = \tfrac{1}{2}\int_0^{\infty} [\,\mathbf{x}^{\mathrm{T}}(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^{\mathrm{T}}(t)\mathbf{R}\mathbf{u}(t)\,]\,\mathrm{d}\,t$$

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

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

Next class:

Coordinate Systems