Motion Control and PID

EECS 367
Intro. to Autonomous Robotics

ROB 511
Robot Operating Systems

Fall 2020

autorob.org


DIRECTORY

Michael Jackson - “Smooth Criminal” - https://youtu.be/h_D3VFfhvs4

https://youtu.be/7osiCOjdjBQ

https://youtu.be/7osiCOjdjBQ

https://youtu.be/KiGGTmuBujs


Ignika Smooth criminal
https://youtu.be/lXkYz0AfcAk


Biped Robotics Lab @ Michigan - https://youtu.be/-35xJfFMDkE

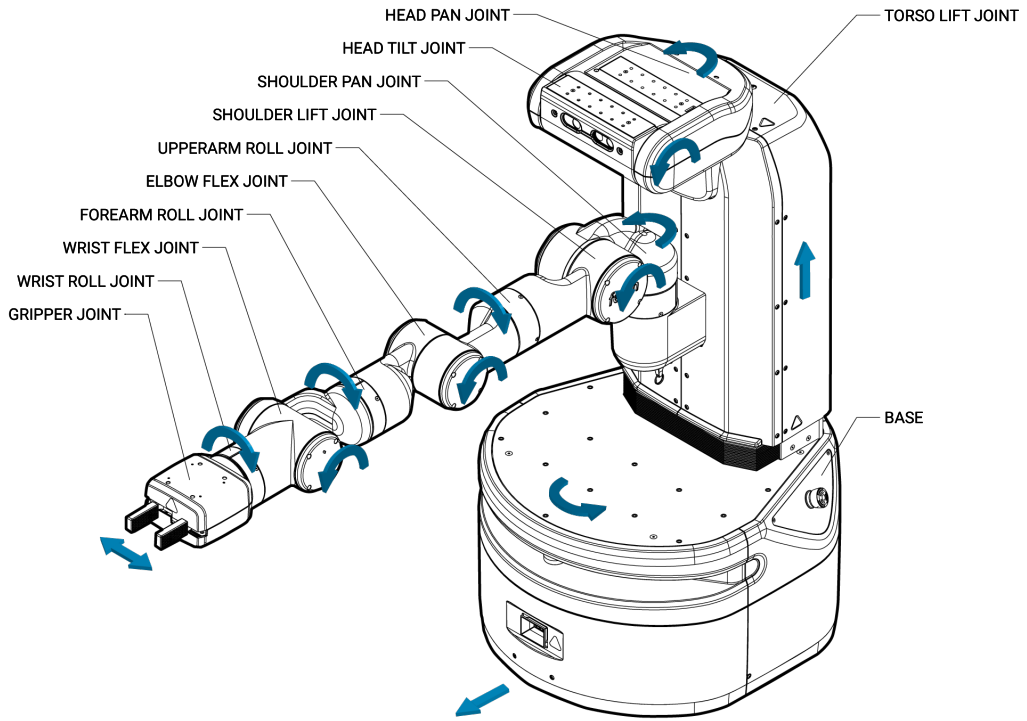

Biped Robotics Lab @ Michigan - https://youtu.be/0gauVSUJzd0

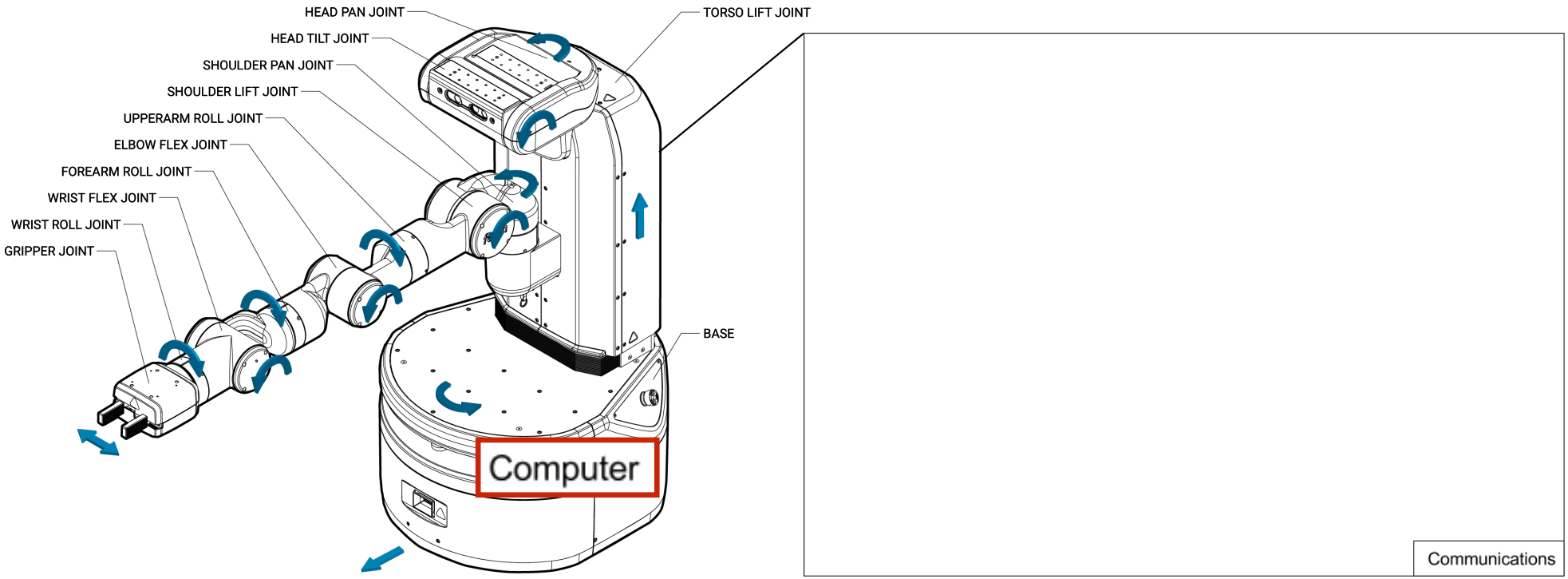
Control must be fast and often

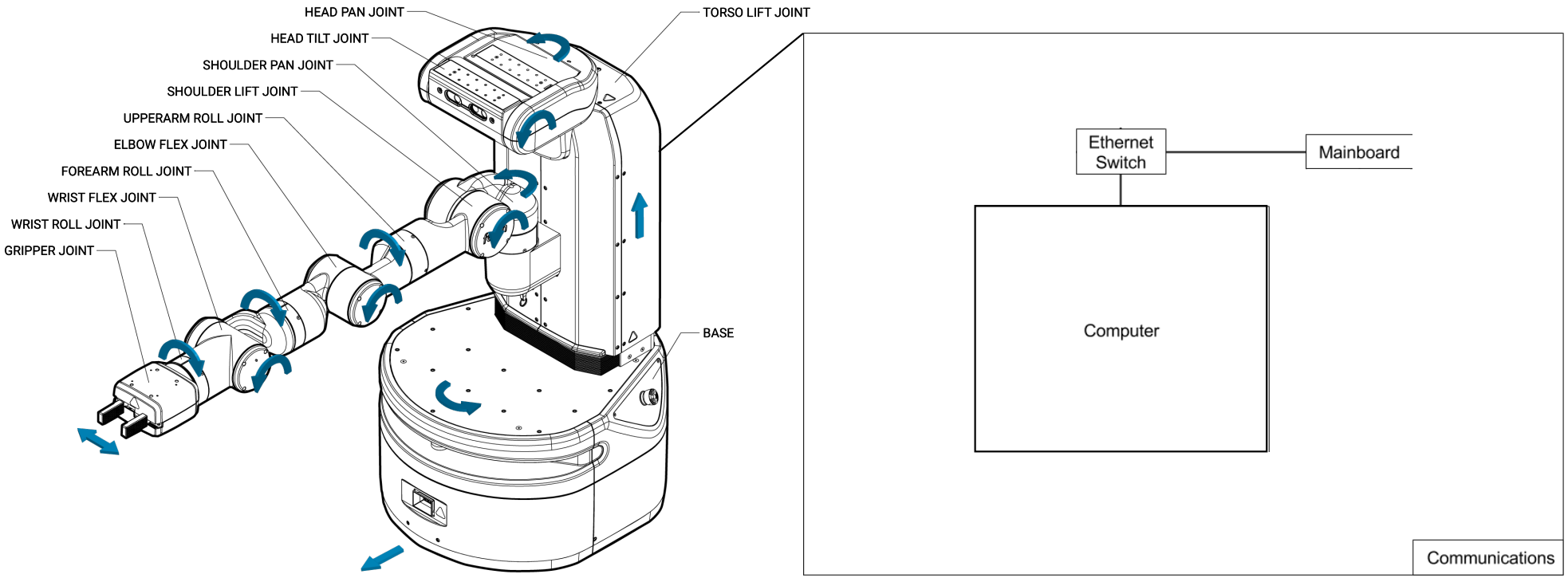

https://youtu.be/8o6FroUWkpE

Communication inside a robot is like its central nervous system

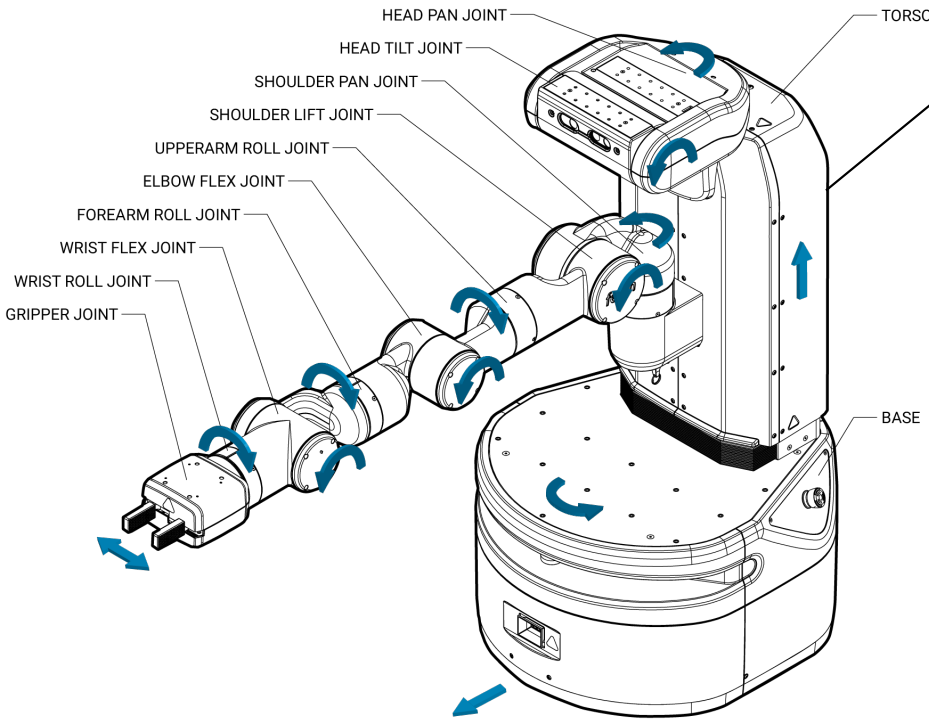
Communication inside a robot is like its central nervous system

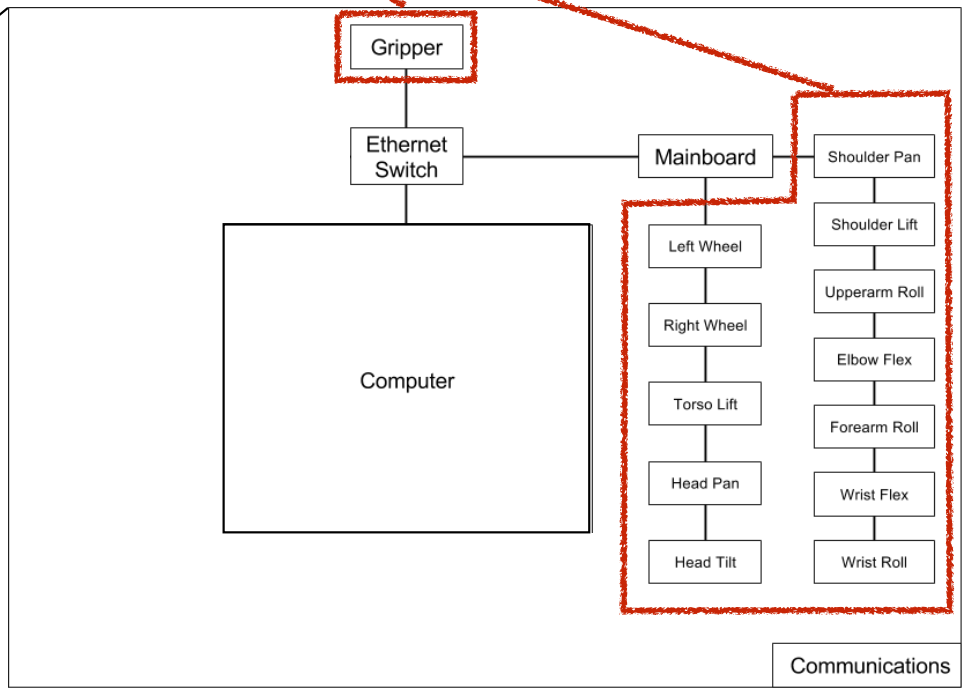


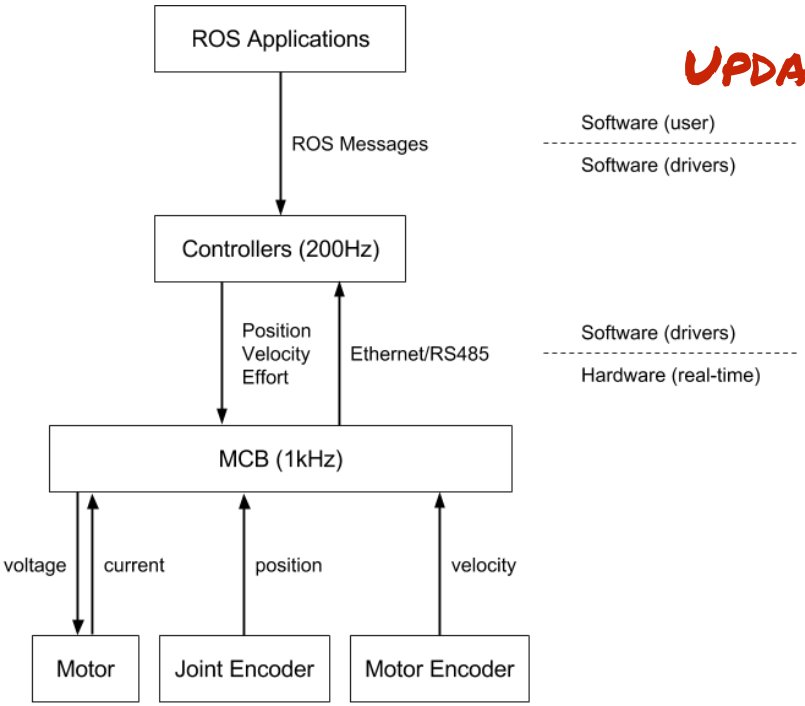


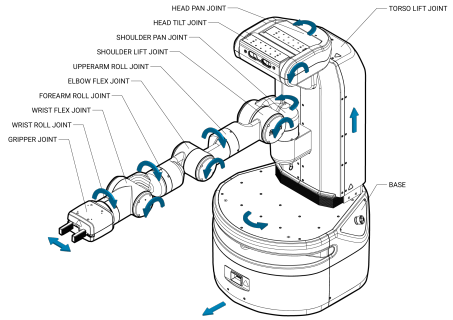

HEAD PAN JOINT
HEAD TILT JOINT
SHOULDER PAN JOINT
SHOULDER LIFT JOINT
UPPERARM ROLL JOINT
ELBOW FLEX JOINT
FOREARM ROLL JOINT
WRIST FLEX JOINT
WRIST ROLL JOINT
GRIPPER JOINT
TORSO LIFT JOINT
BASE
Ethernet Switch
Mainboard
Computer
Communications

Update joint control 1,000 times per second

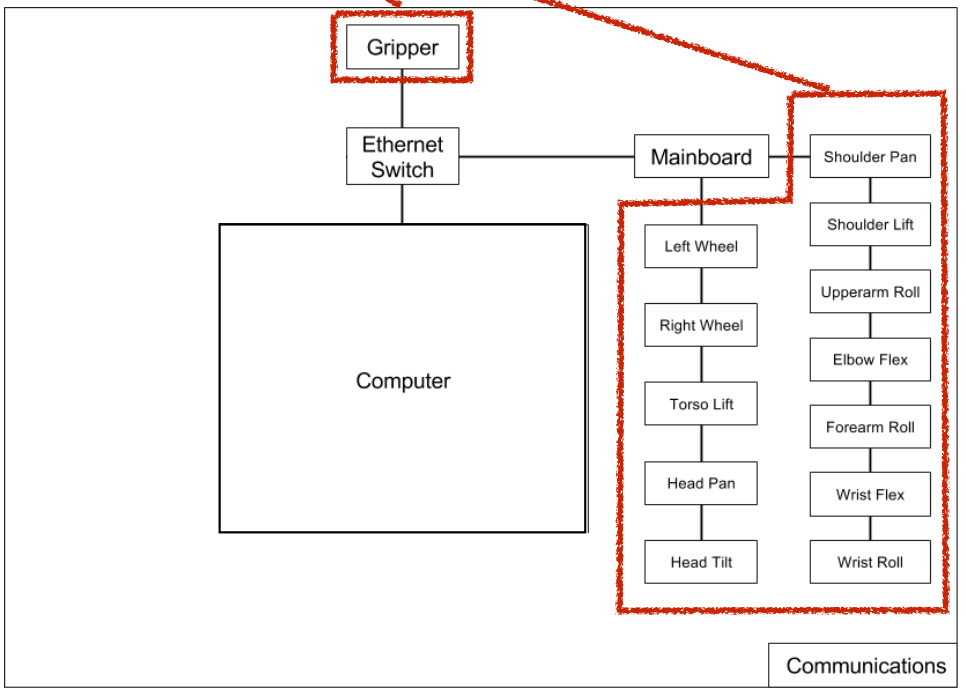


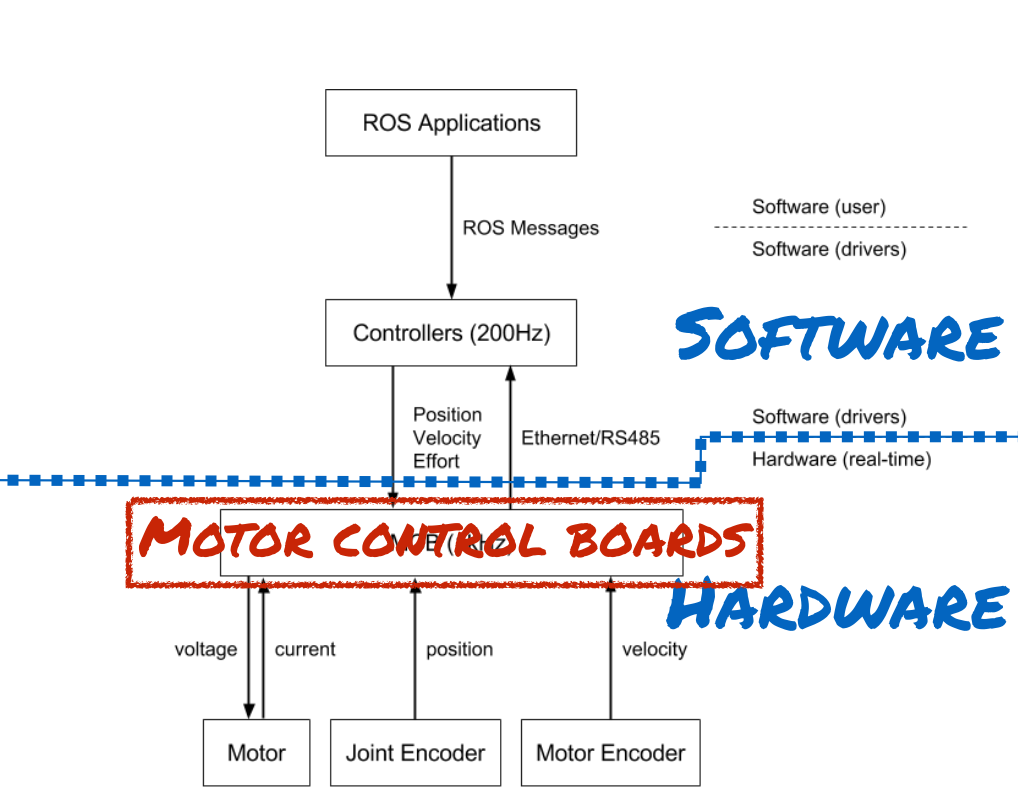


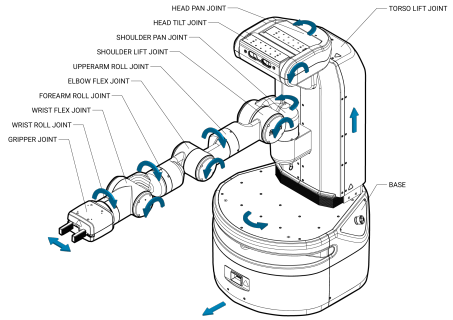




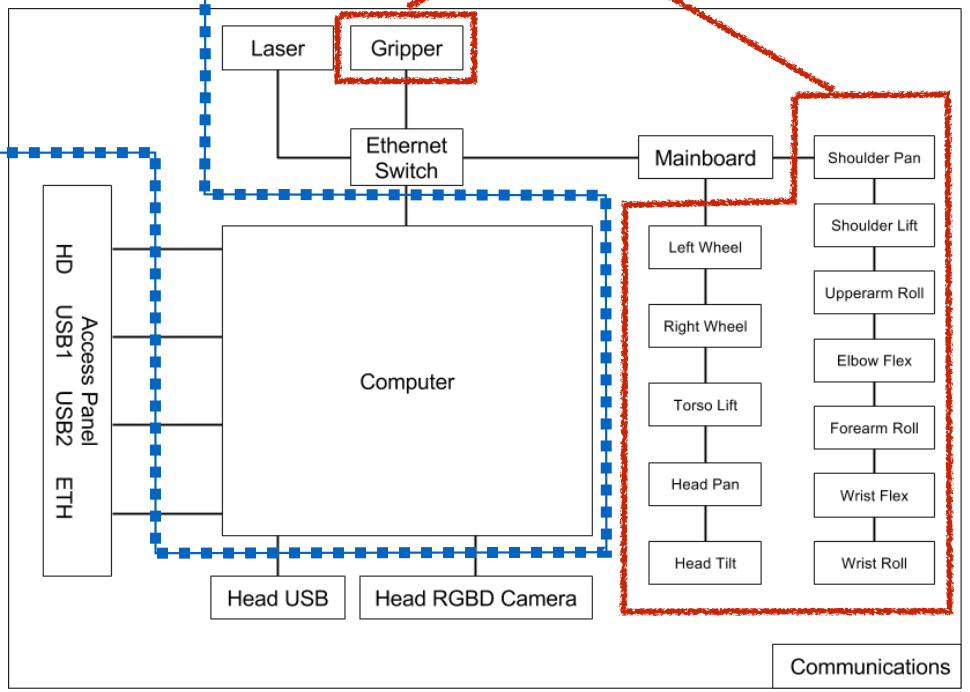
Update joint control 1,000 times per second







UPDATE JOINT CONTROL 1,000 TIMES PER SECOND



ROS Applications

ACTIONS

ROS Messages

Software (user)

Software (drivers)

Controllers (200Hz)

SOFTWARE

SETPOINTS

Position
Velocity
Effort

Ethernet/RS485

Software (drivers)

Hardware (real-time)

MOTOR COMMANDS

MCB (1kHz)

HARDWARE

voltage

current

position

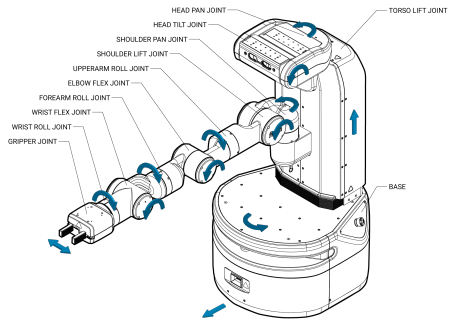
velocity

PROPRIOCEPTION

Motor

Joint Encoder

Motor Encoder



MOTOR CONTROL BOARDS UPDATE JOINT CONTROL 1,000 TIMES PER SECOND

Laser

Gripper

Ethernet Switch

Mainboard

Shoulder Pan

Shoulder Lift

Upperarm Roll

Elbow Flex

Forearm Roll

Wrist Flex

Wrist Roll

Left Wheel

Right Wheel

Torso Lift

Head Pan

Head Tilt

HD
USB1
USB2
ETH
Access Panel

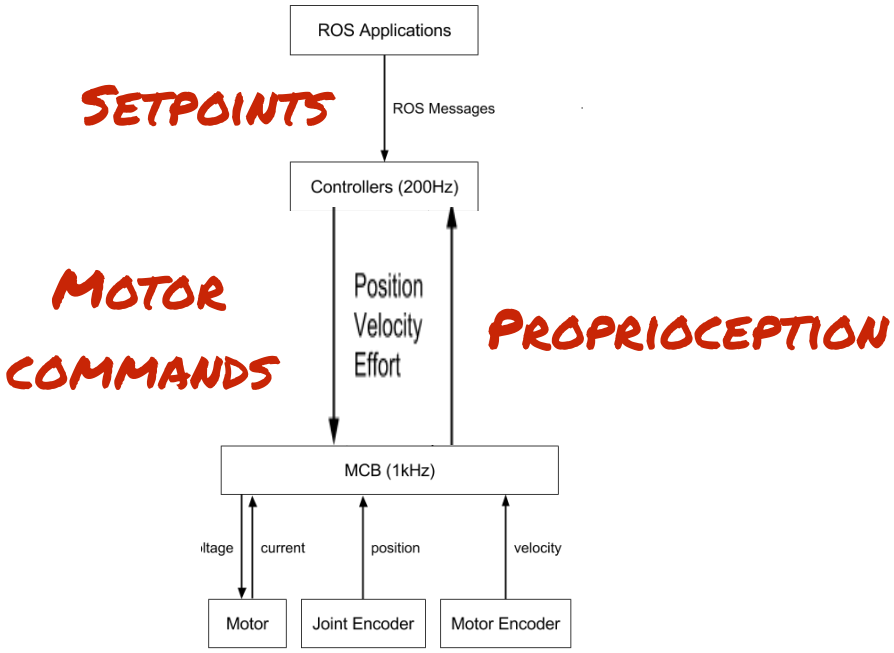
Computer

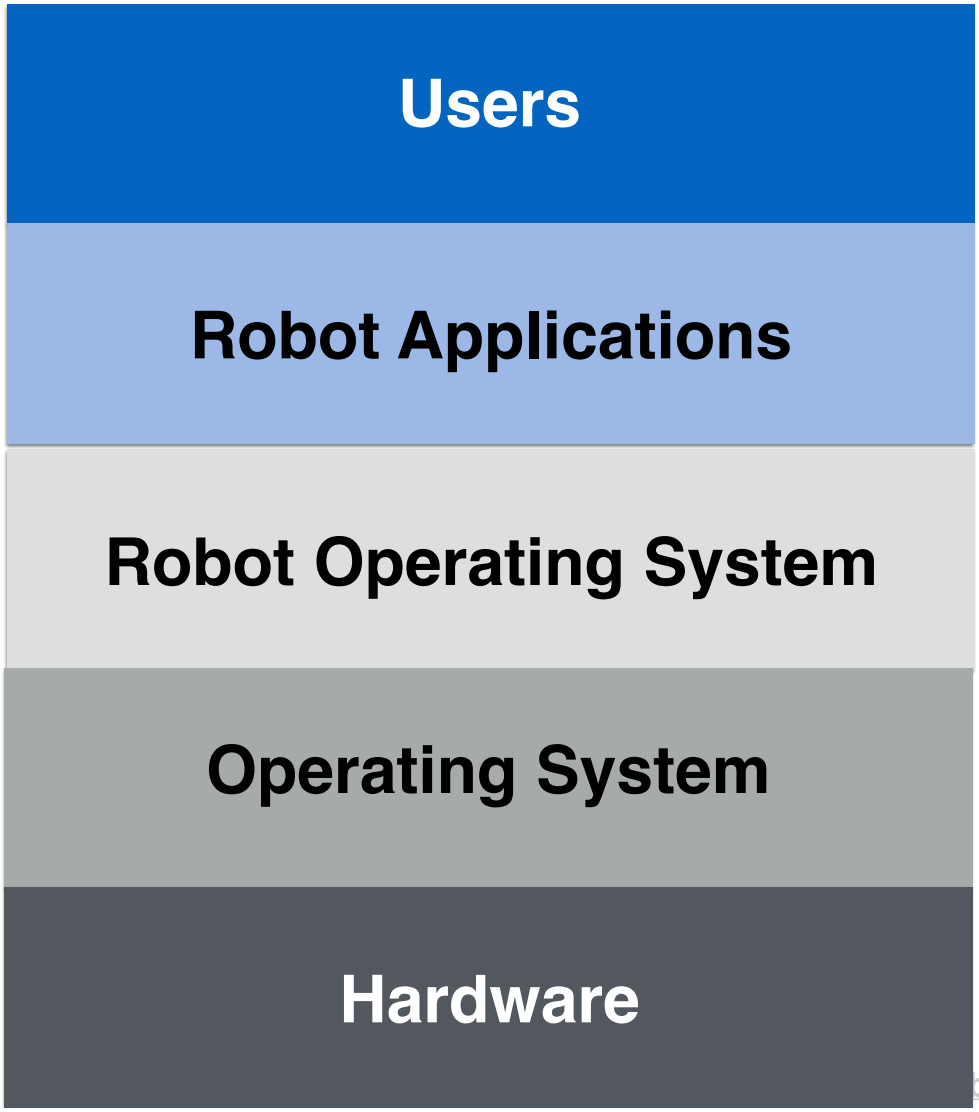
Head USB

Head RGBD Camera

Communications

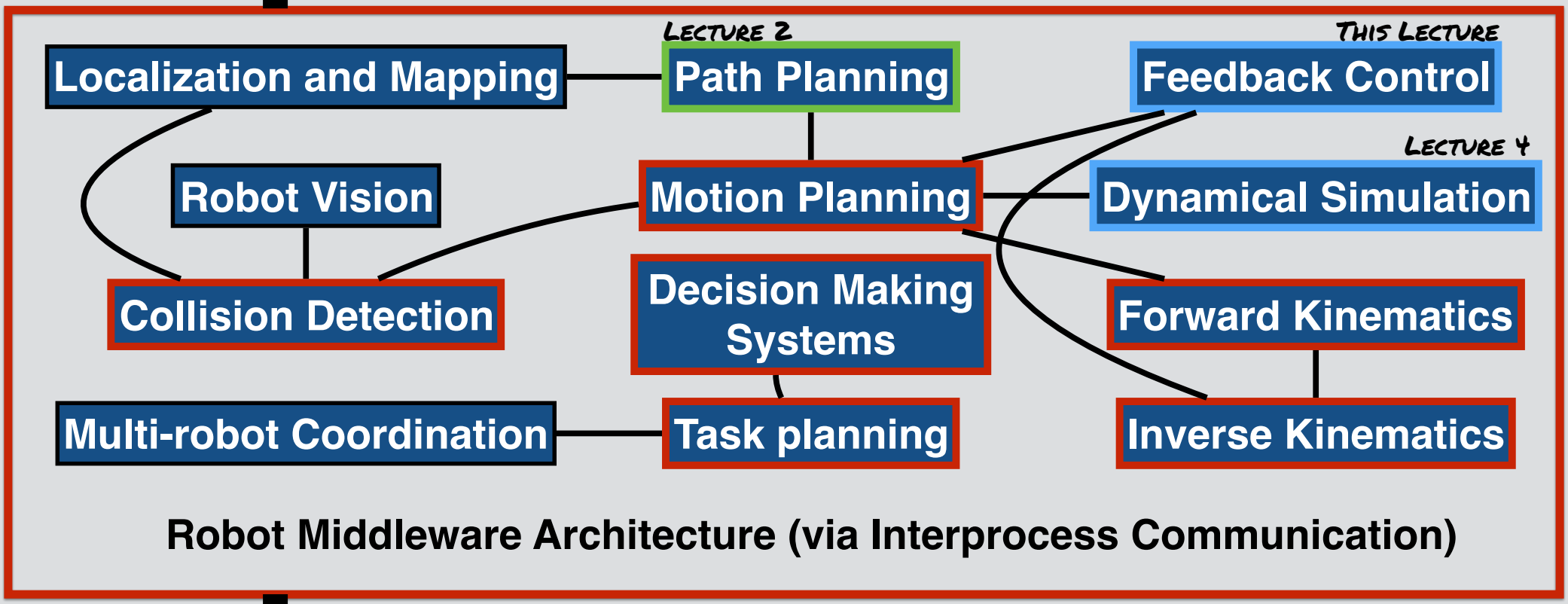
Motion control:
process to achieve and maintain a desired state (or setpoint) for a joint through applying motor commands based on current proprioceived state



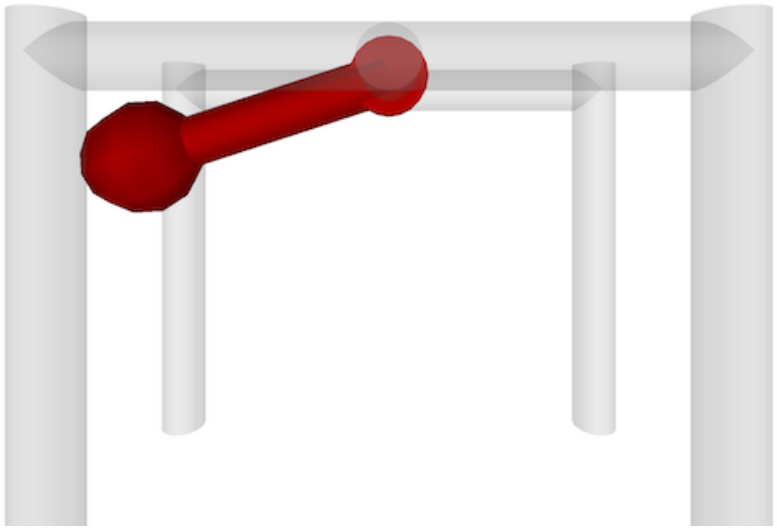


Robot Operating System

COVERED AT BREADTH IN AUTOROB



Project 2: Pendularm



Lecture 4

Dynamical Simulation

Equations of motion: Simple pendulum

Integrators: Euler, Verlet, Velocity Verlet, RK4

This Lecture

Feedback Control

PID Control of 1 DoF robot

We will put it all together later in this lecture

What can you do with pendularm?

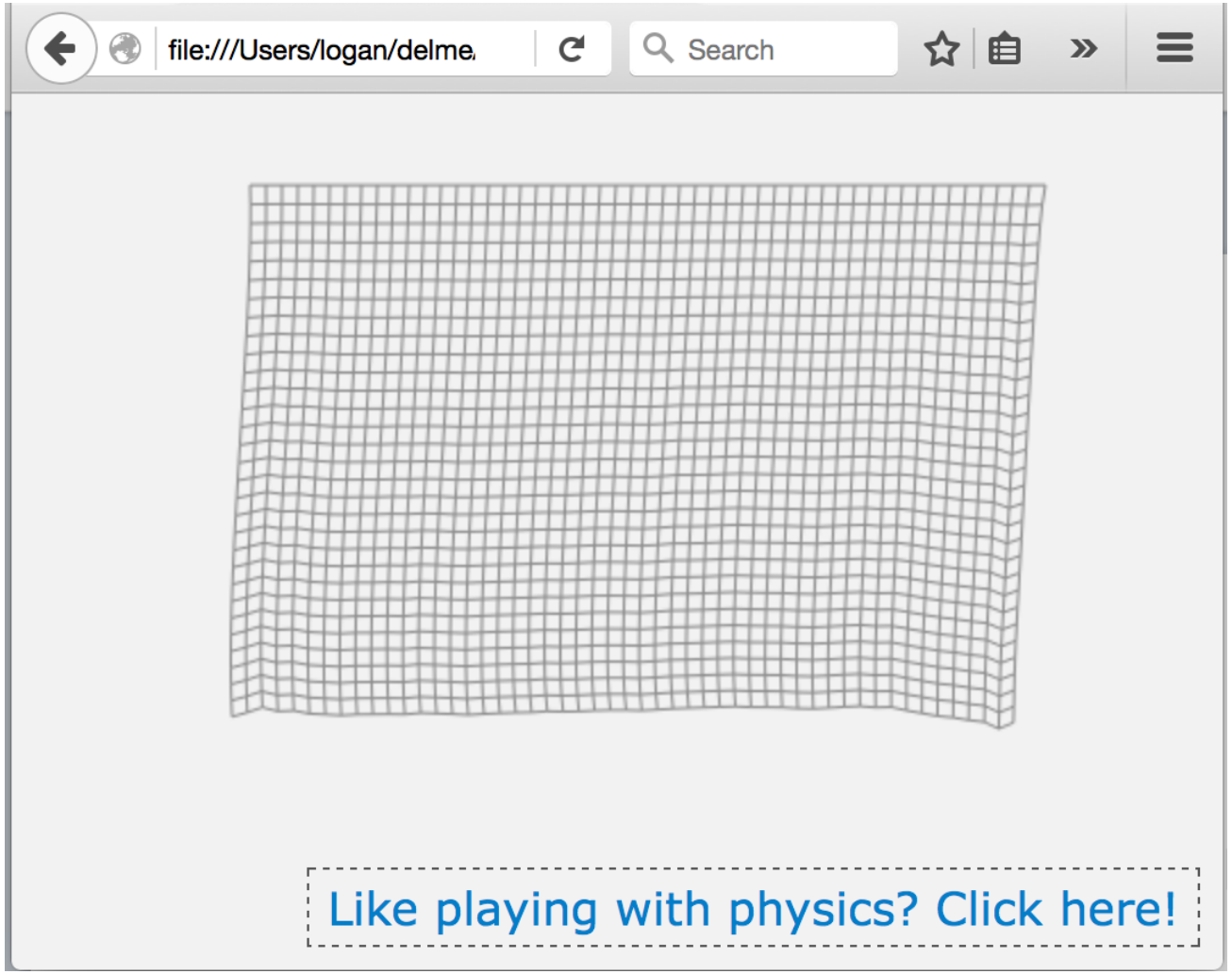

Inverted Pendulum by PID control - Chengcheng Zhu et al. - EECS 473 2016


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

Building up to more…

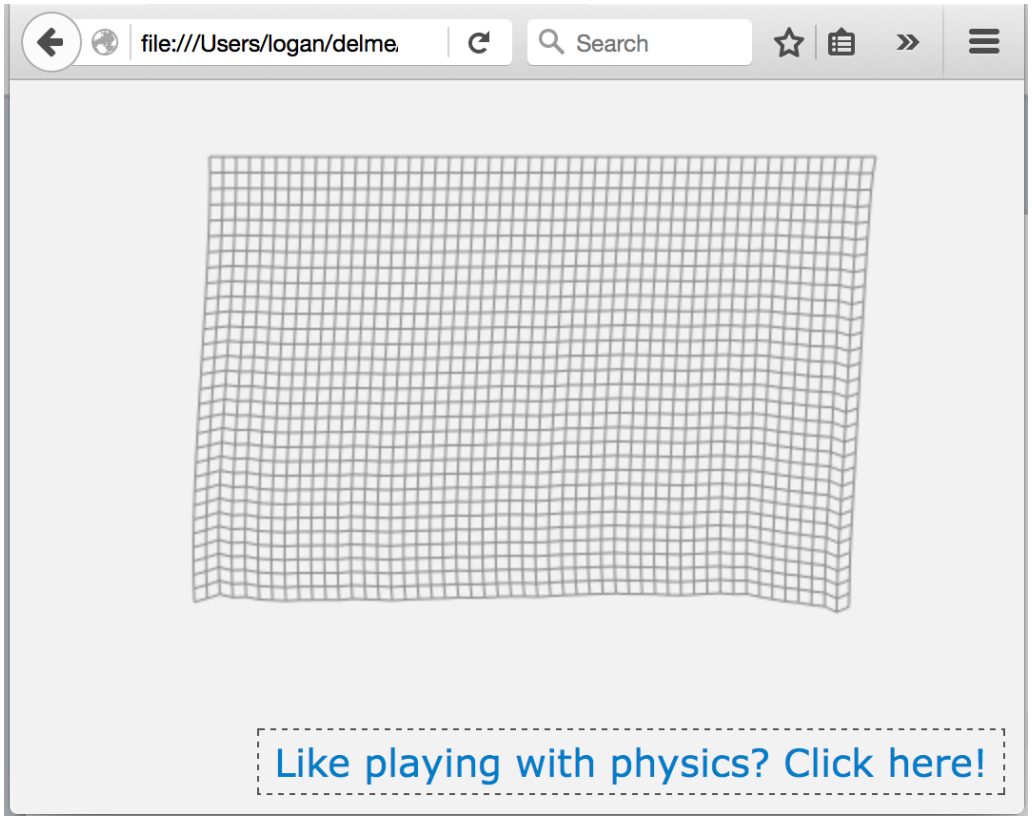

x16
RoBob Ross - Rabideau, Ekins, Rao, Jones - EECS 467 - Winter 2017

Let’s start with something fun



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

Verlet Simulated Mass-Spring Cloth

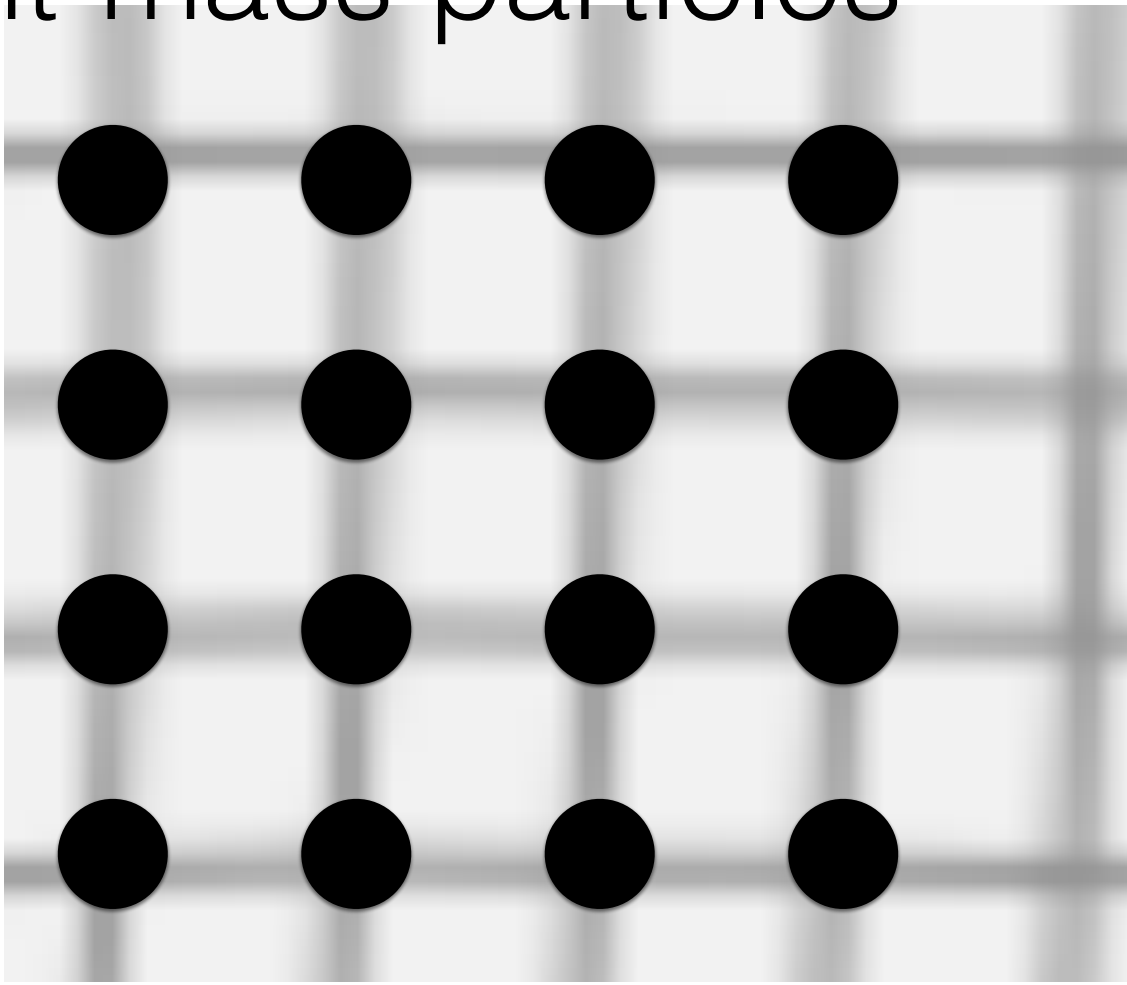


- Cloth represented as a network of point-mass nodes
- Neighboring nodes connected by high stiffness springs

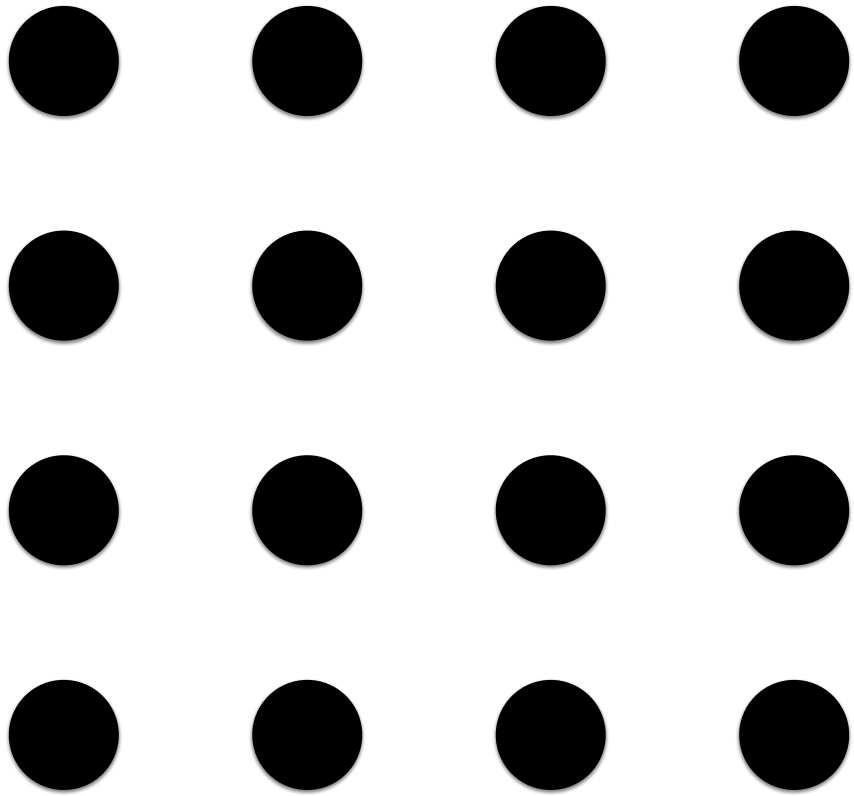
http://codepen.io/dissimulate/pen/KrAwx

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

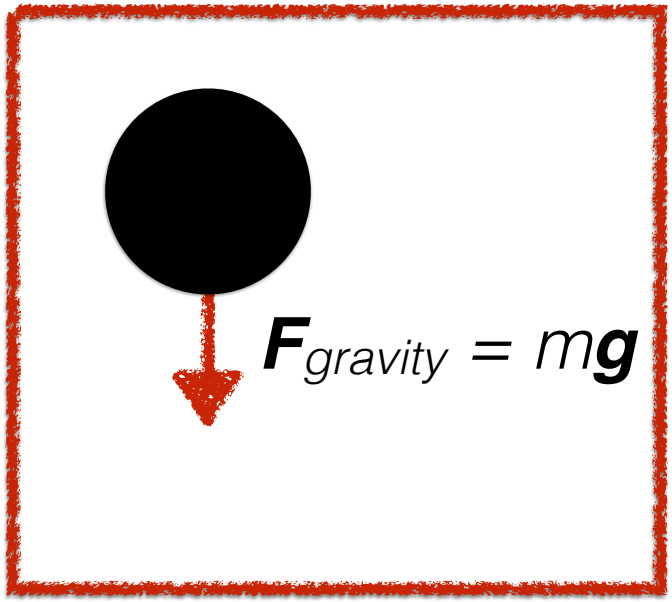
Start with point-mass particles

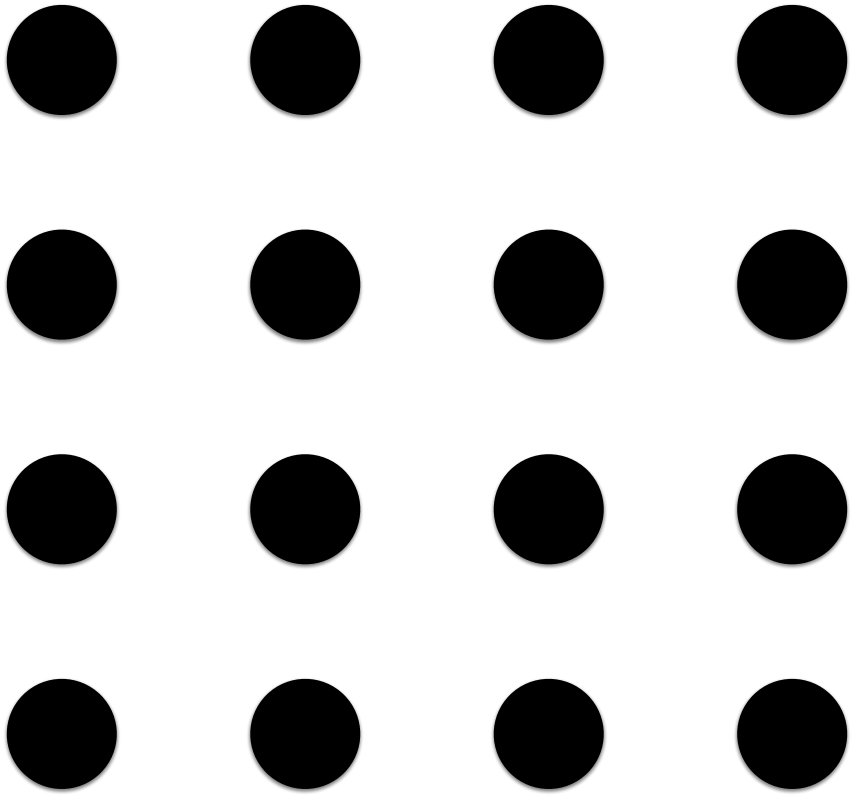


Start with point-mass particles

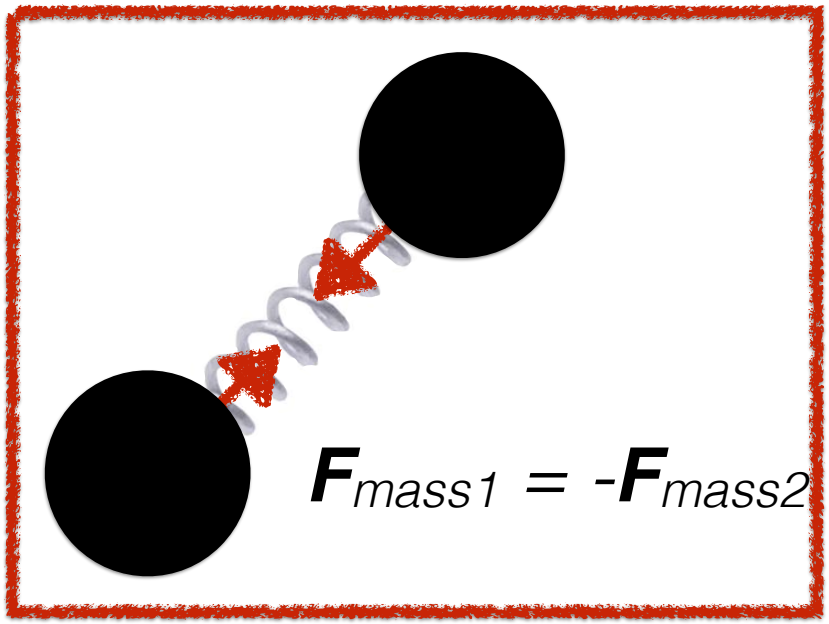


Apply gravity

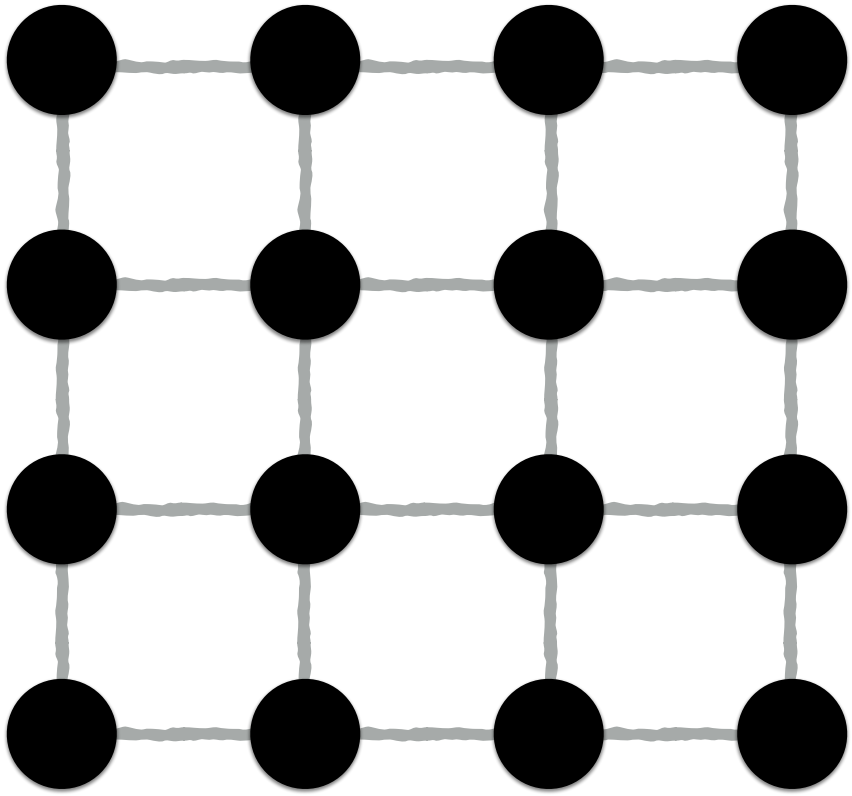




Enforce spring force constraints

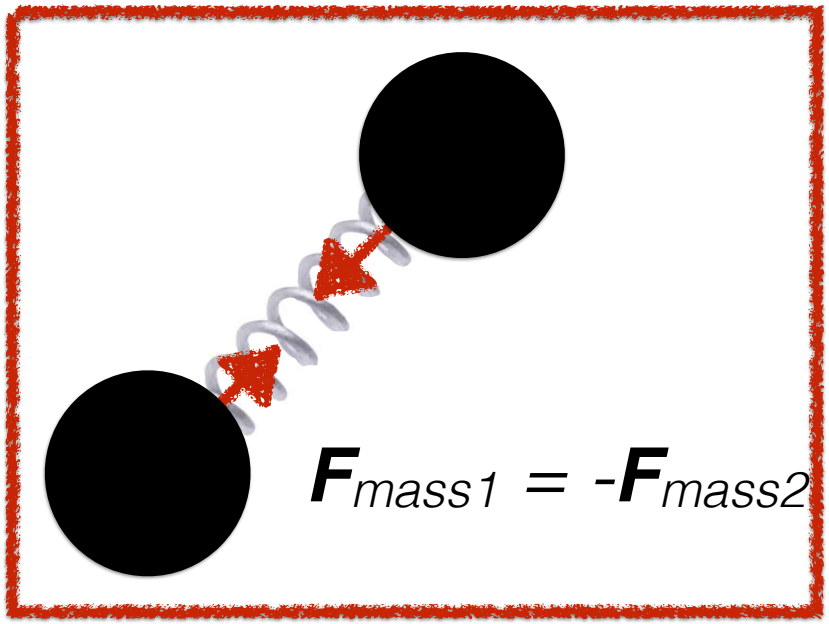


spring with stiffness 1
and rest length *r*



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

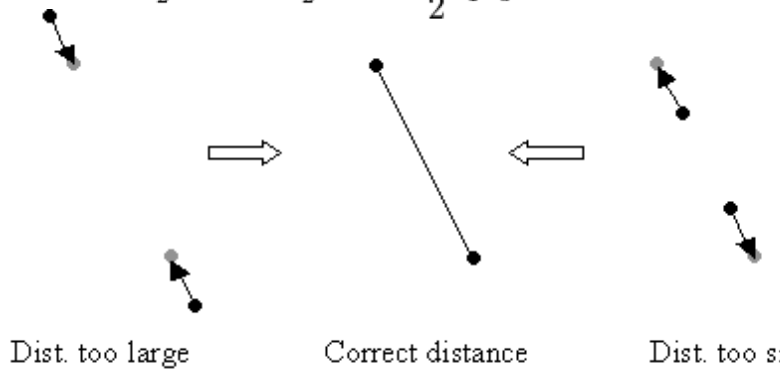
Enforce spring force constraints

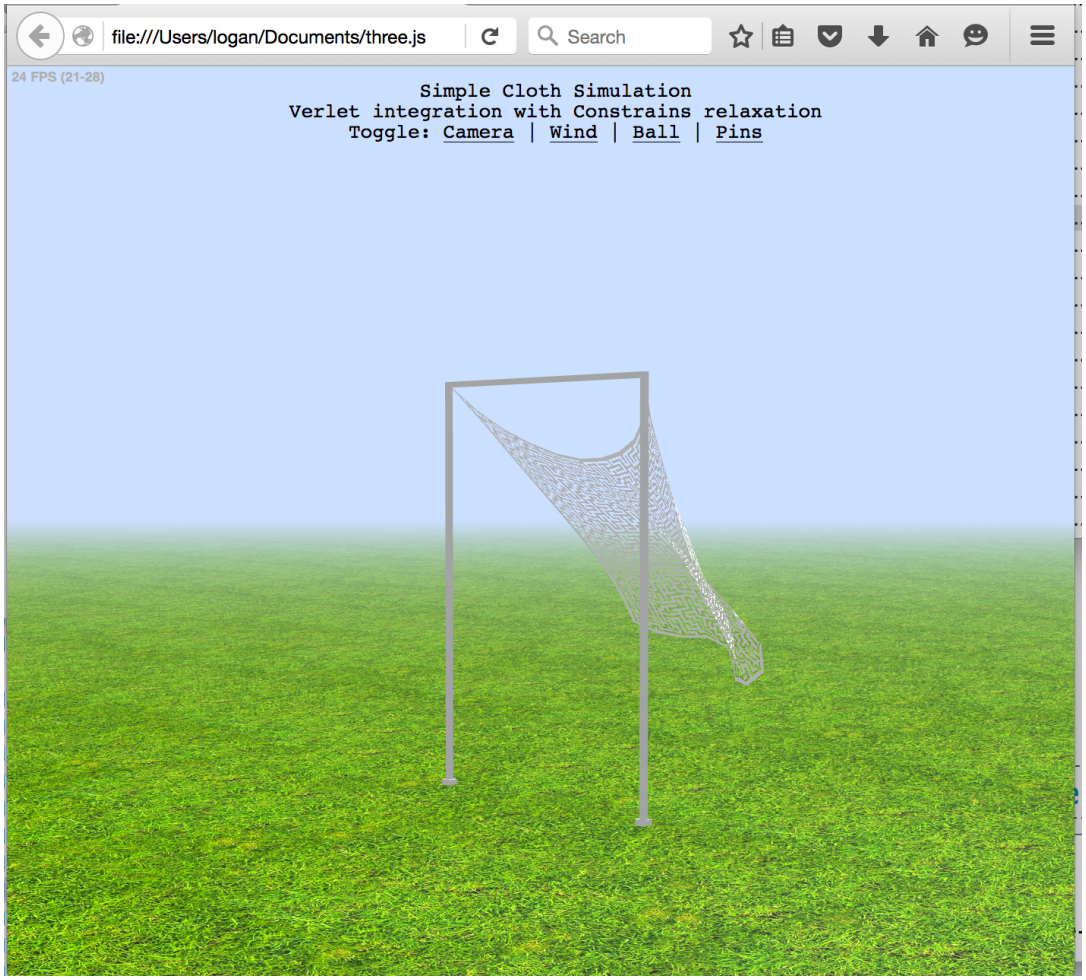


spring with stiffness 1
and rest length *r*

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



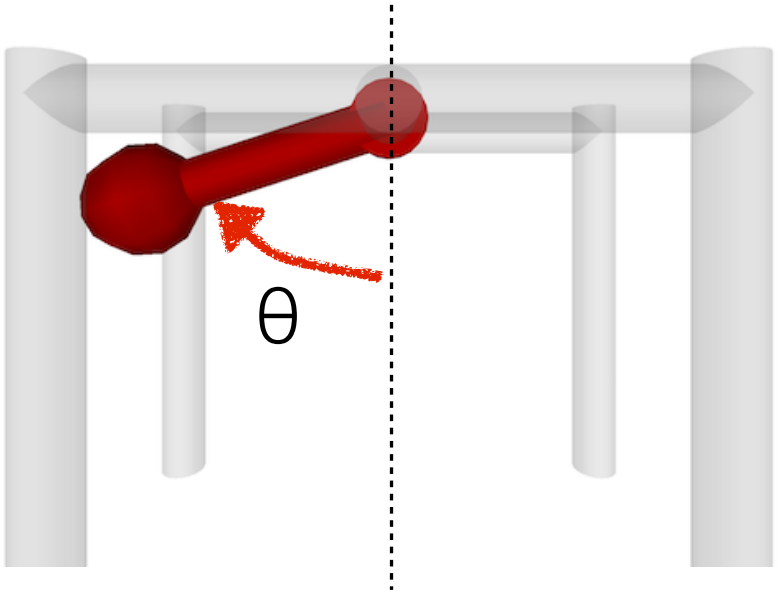


http://threejs.org/examples/webgl_animation_cloth

Pendulum Revisited

Pendulum Revisited

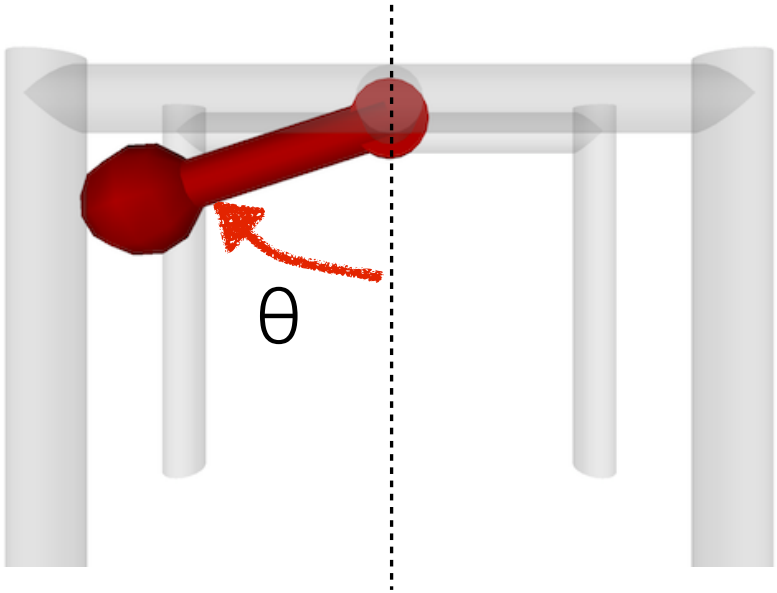
Generalized coordinates



Simulate the motion of DOFs
(pendulum angle) directly

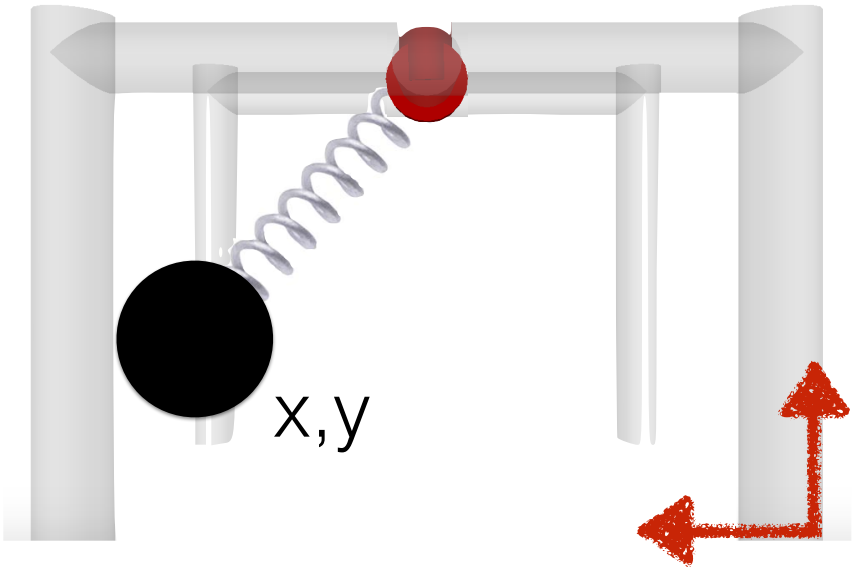
Pendulum Revisited

Generalized coordinates



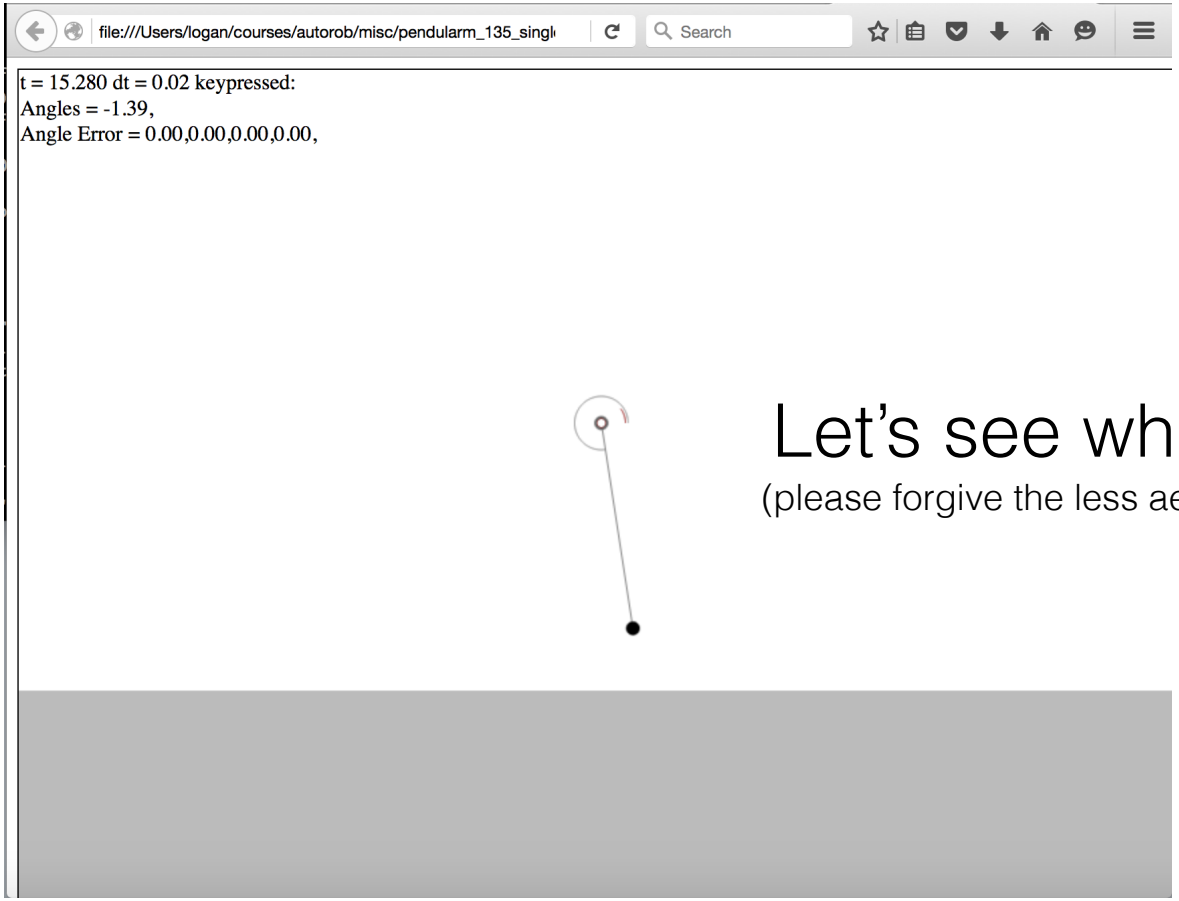
Simulate the motion of DOFs (pendulum angle) directly

Maximal coordinates



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

Maximal Pendularm



Let's see what happens

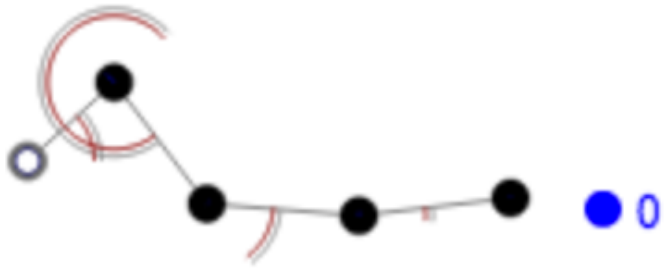
(please forgive the less aesthetically pleasing UI)

Maximal double pendulum?

Maximal double pendulum?

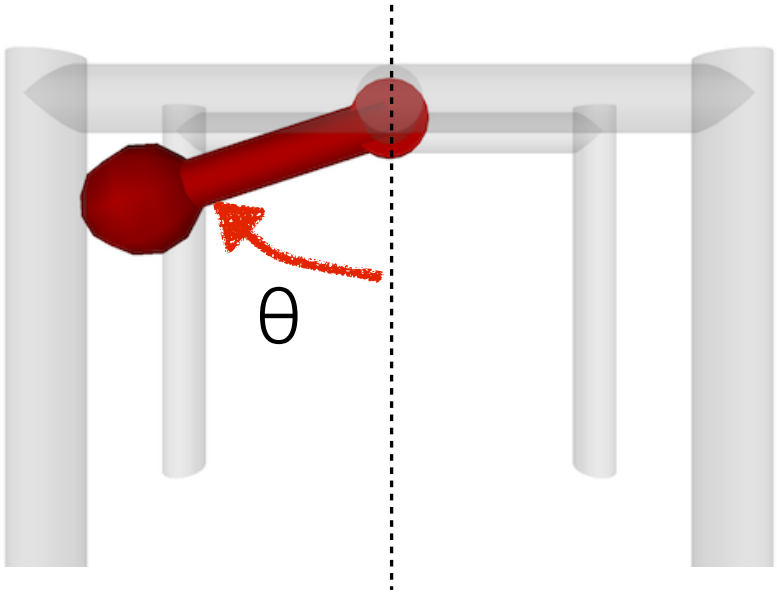
I can do even better…
Maximal 5-Link Pendularm

Maximal 5-Link Pendularm



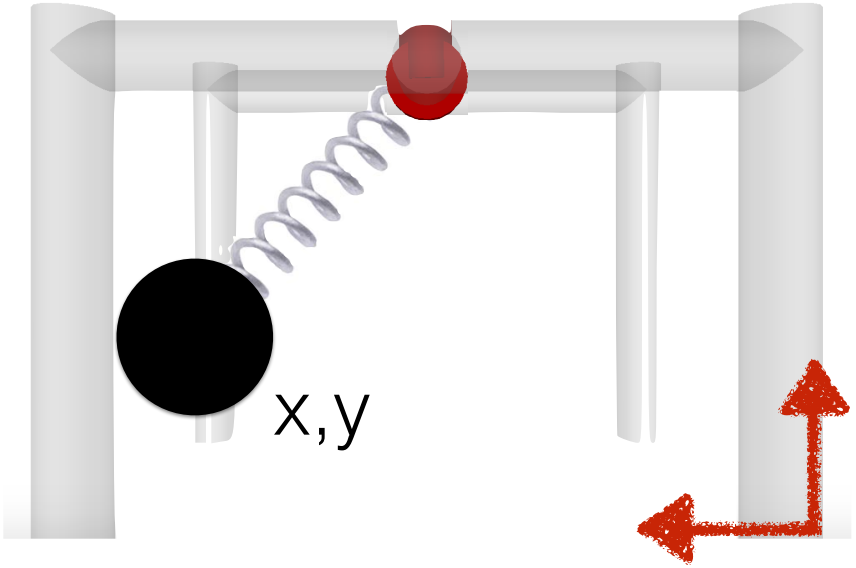
Why use generalized coordinates?

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)

Why use generalized coordinates?

Equation of motion
(with rotational inertia *I*)

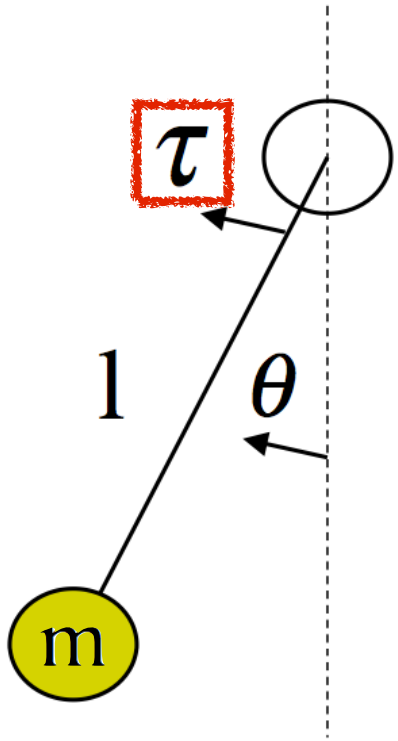
$$I\ddot{\theta} = -mgl\sin(\theta) + \tau$$

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

Motor force expressed via DOFs

Equation of motion
(with rotational inertia *I*)

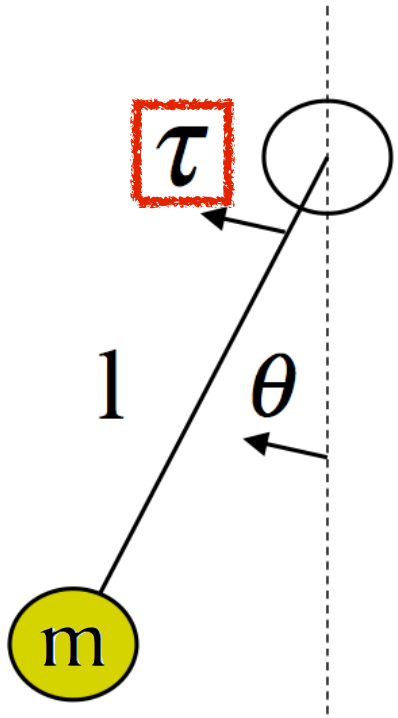
$$I\ddot{\theta} = -mgl\sin(\theta) + \tau$$

with Parallel Axis Theorem (I=mr^2)

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

Numerical integration over time

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

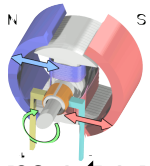


Motor produces torque
(angular force)

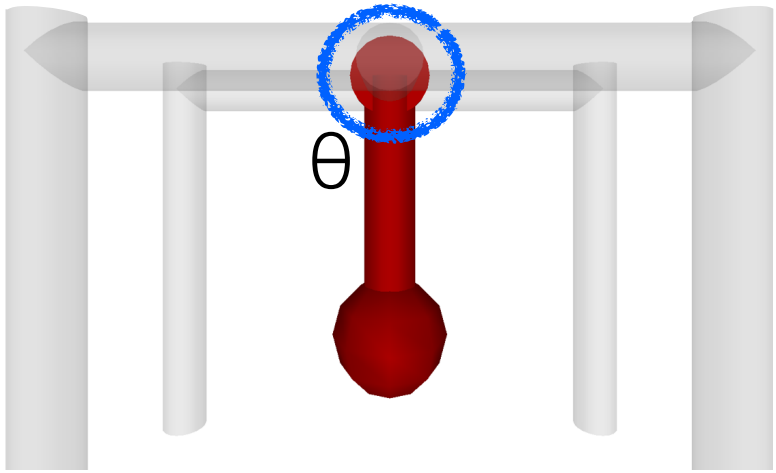
Angle expresses pendulum
range of motion

Pendulum of length l with
point mass m

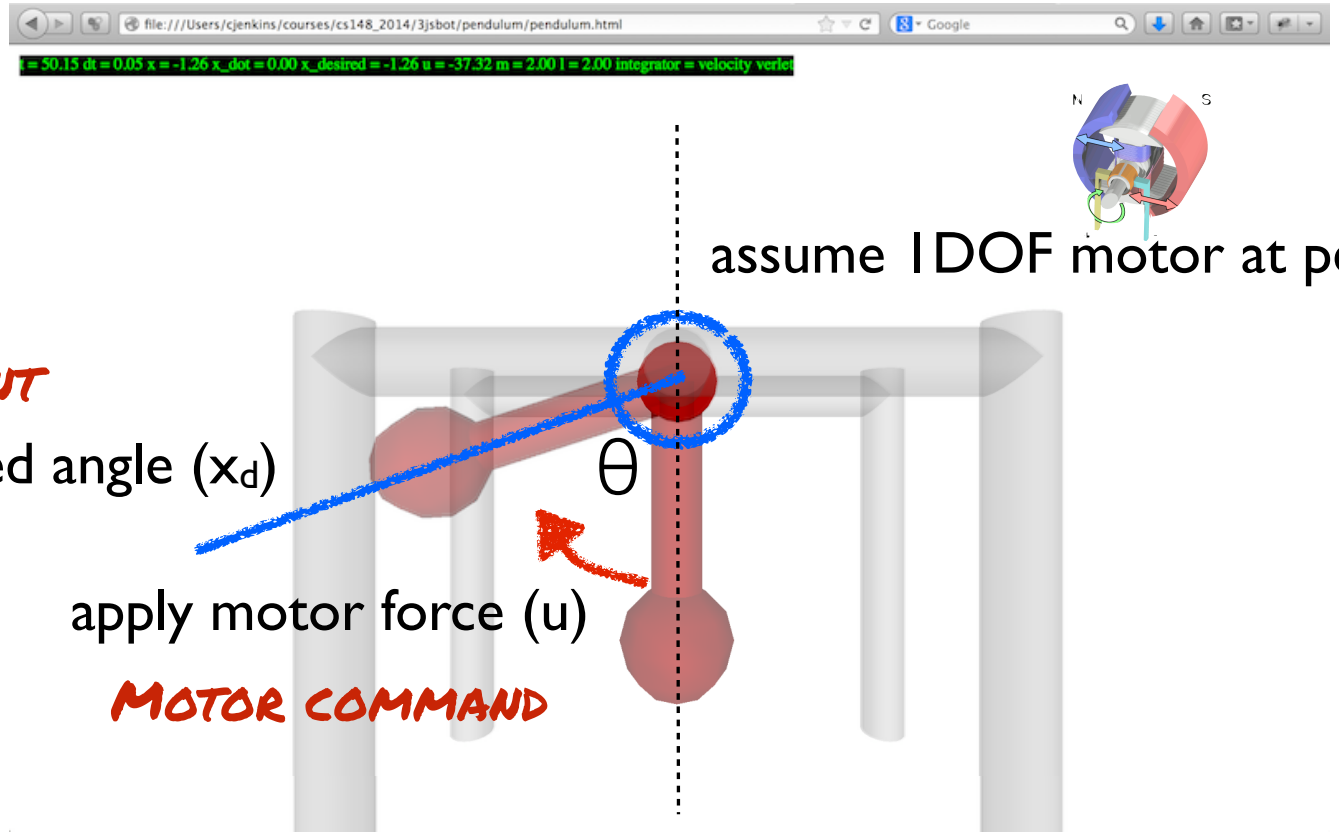
Motor force expressed via DOFs

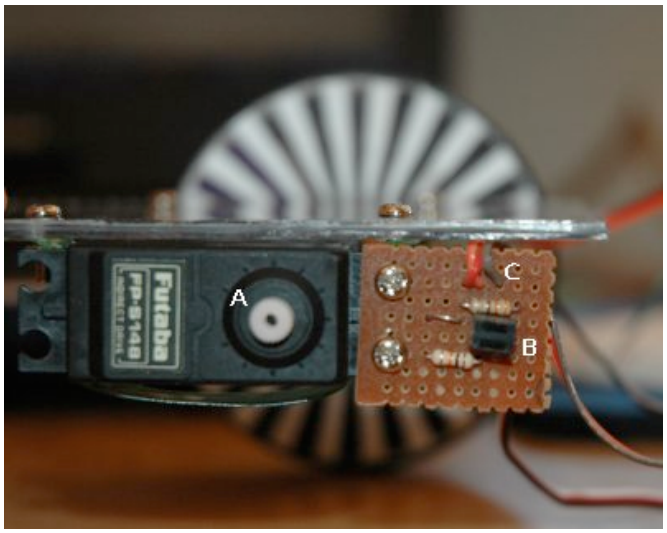


assume 1DOF motor at pendulum axis



Can we control the pendulum?

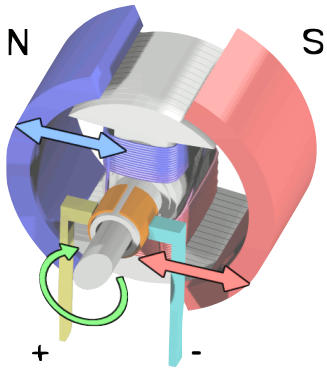


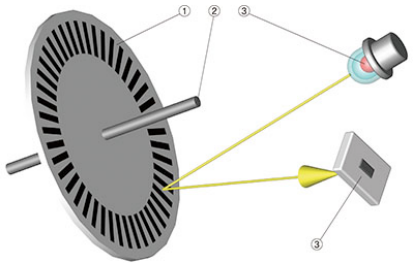


Servos have:

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

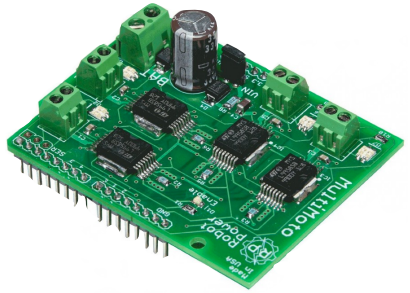
Actuator (electric)





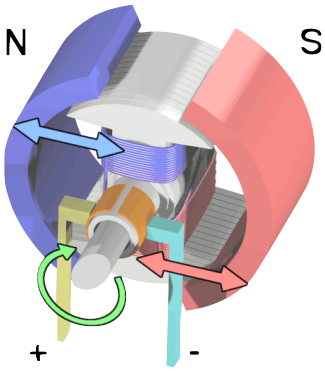
Proprioception (optical encoder)

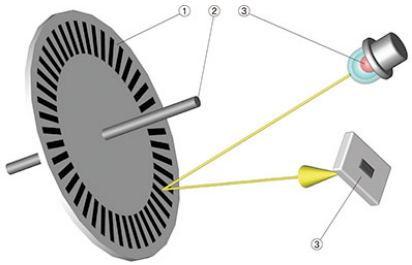
Motor Controller (4 channel H-Bridge)



What is inside a servo?

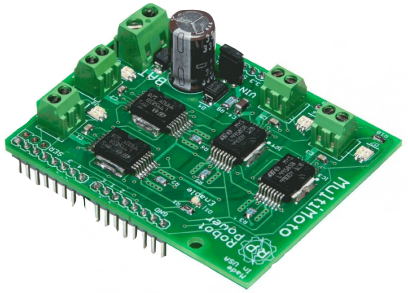
Actuator
(electric)





Proprioception
(optical encoder)

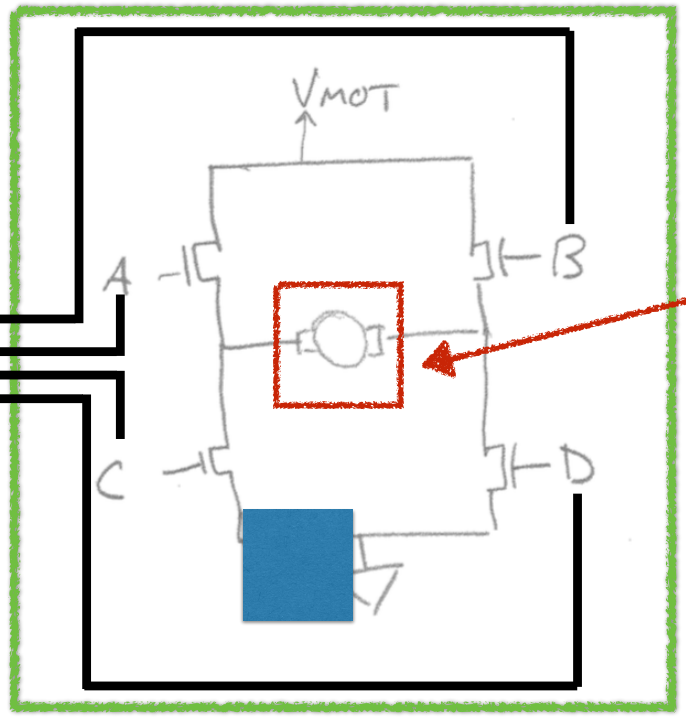
Motor Controller
(4 channel H-Bridge)



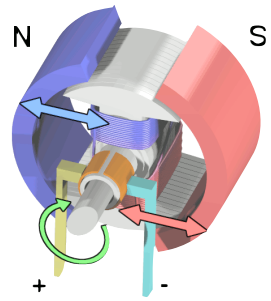
What is inside a servo?

Control Signal

PWM signal sent to H-bridge controller.

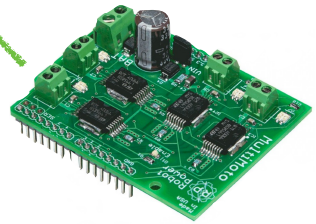


motor



Current actuates motor motion about axis.

H-bridge sends proportional current to motor.

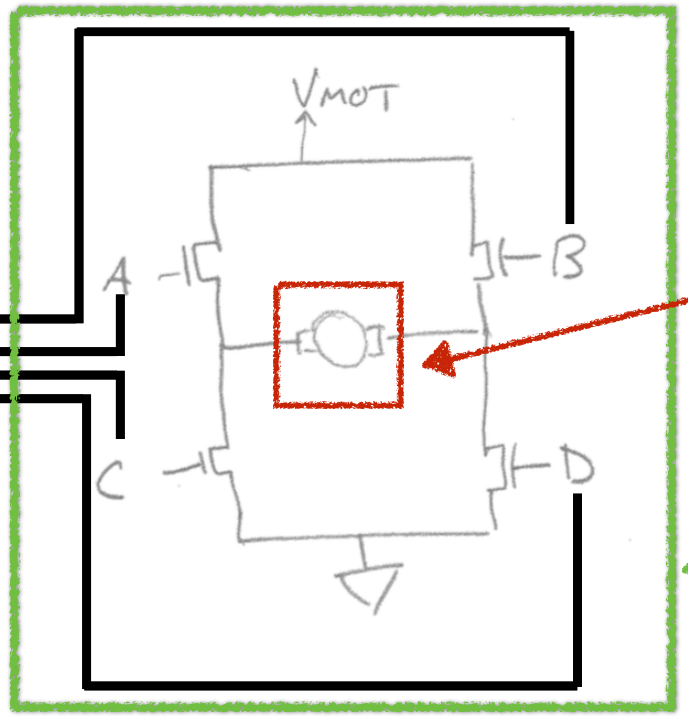


motor controller

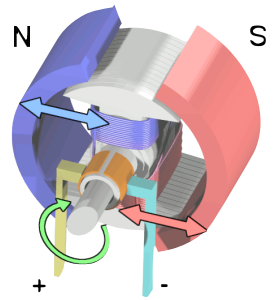
PWM
(Pulse Width Modulation)

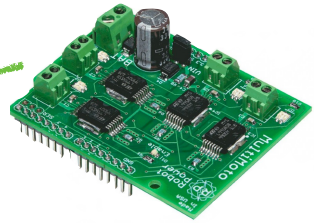
Control Signal

PWM signal sent to H-bridge controller.

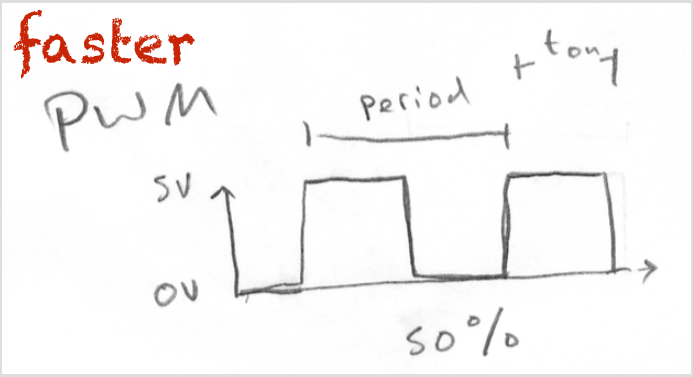


motor



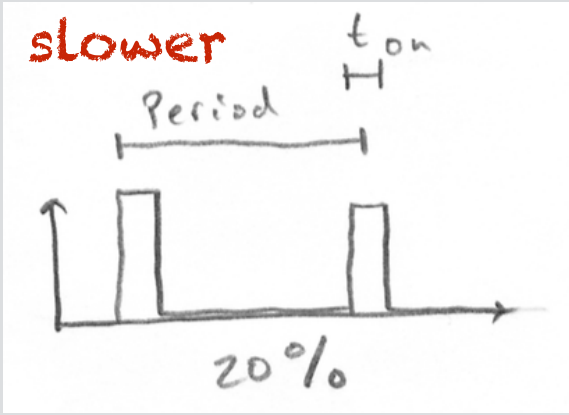


motor controller



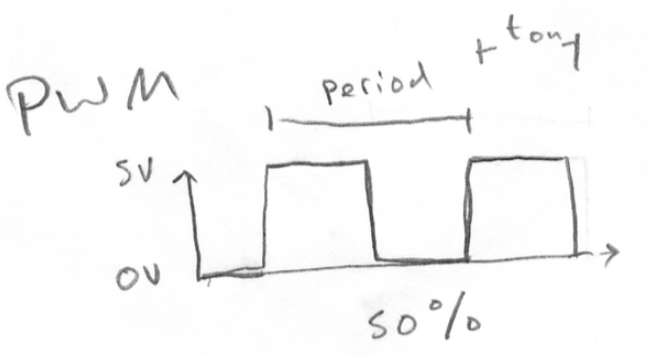
PWM has a periodic duty cycle on each line.

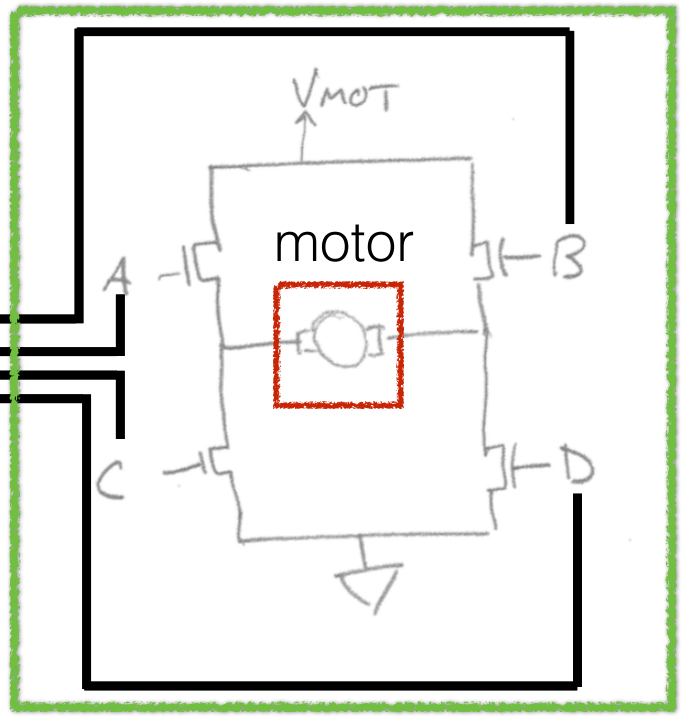
Percentage of time in high voltage ("on" state) determines speed of motor



motor controller

Control Signal



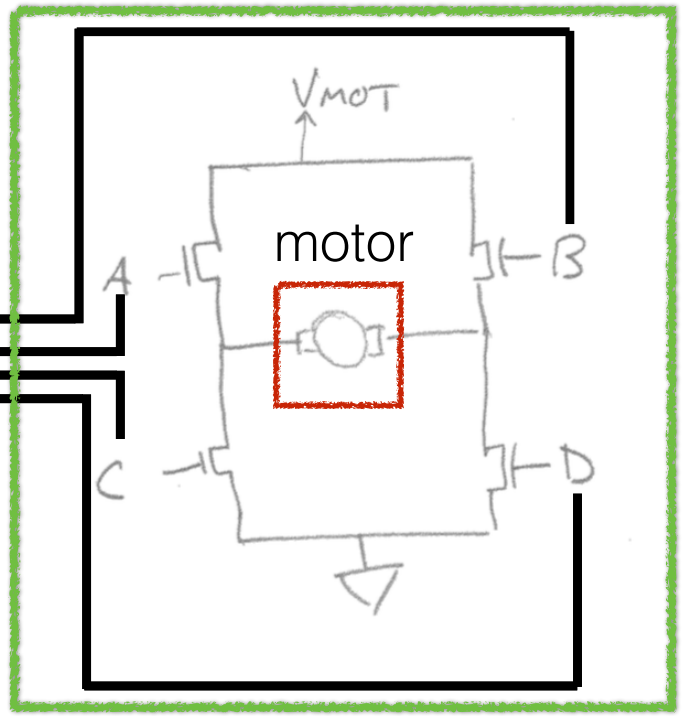


H-bridge has four switches that are "on" or "off"

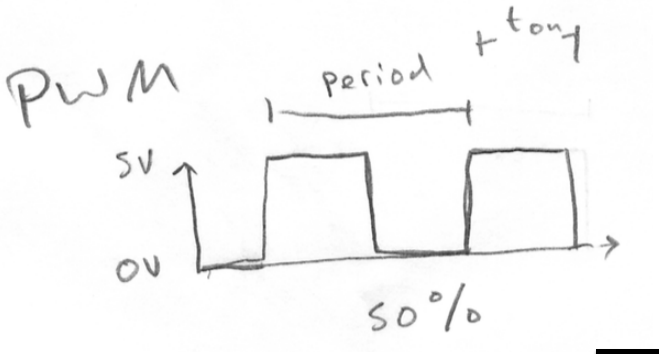
spin forward:
A and D on

spin backward:
B and C on

motor controller



Control Signal



spin forward:
A and D

spin backward:
B and C

what if none are on?

[redacted]

what if C and D?

[redacted]

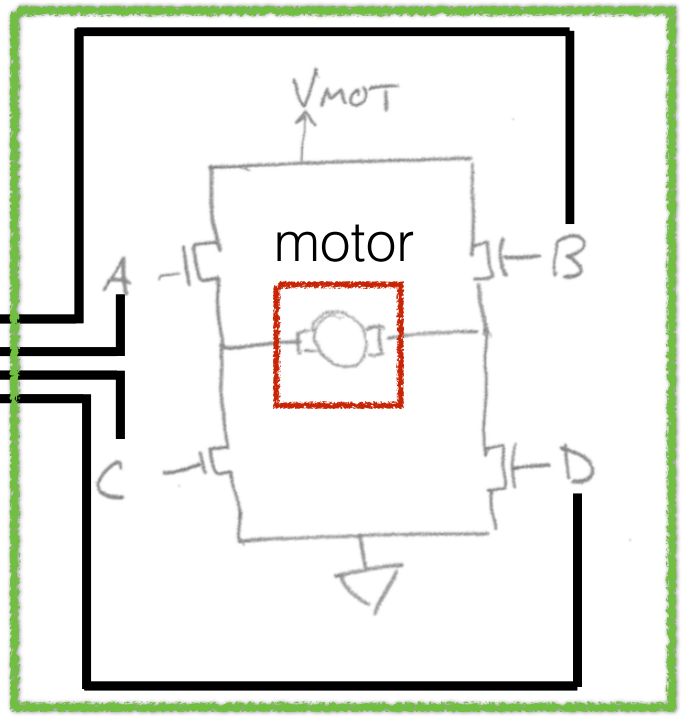
what if A and B?

[redacted]

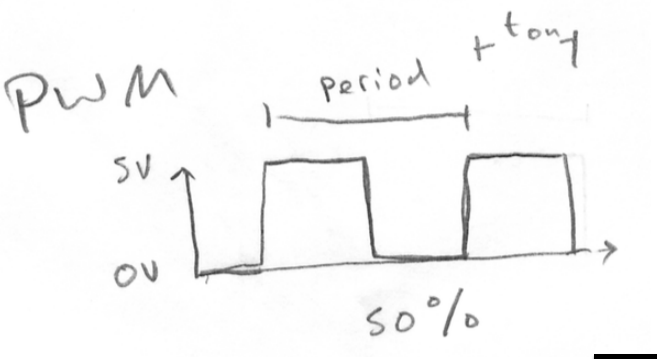
what if A and C?

[redacted]

motor controller



Control Signal



spin forward:
A and D

spin backward:
B and C

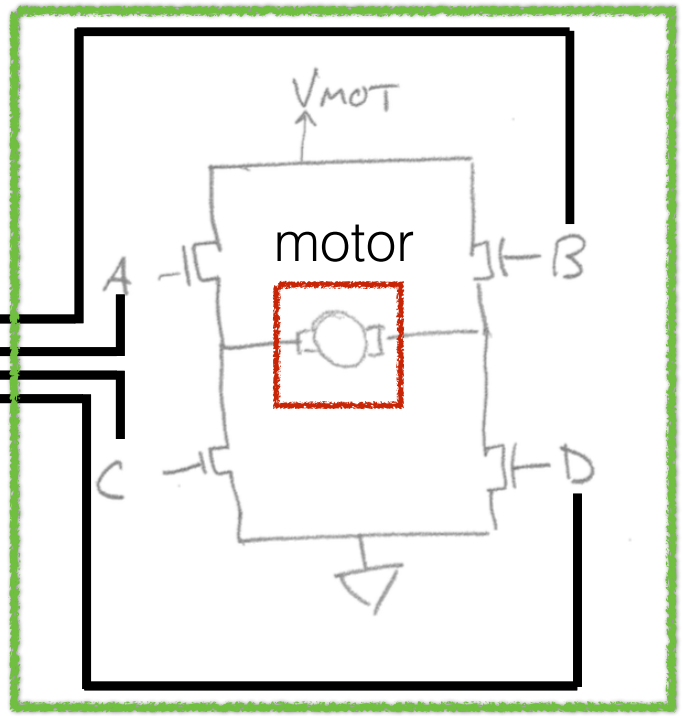
what if none are on?
open circuit
no current -> no torque

what if C and D?
negative torque -> brake

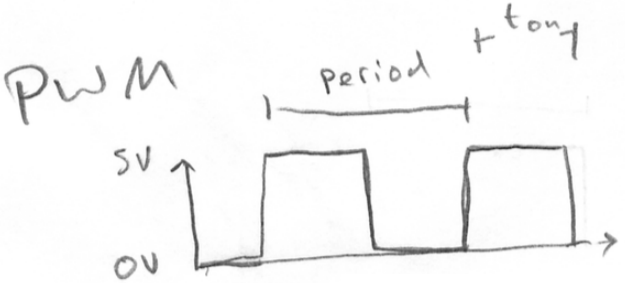
what if A and B?
negative torque -> brake

what if A and C?

motor controller



Control Signal



No more robot, (our hardware prevents this)

what if A and C?

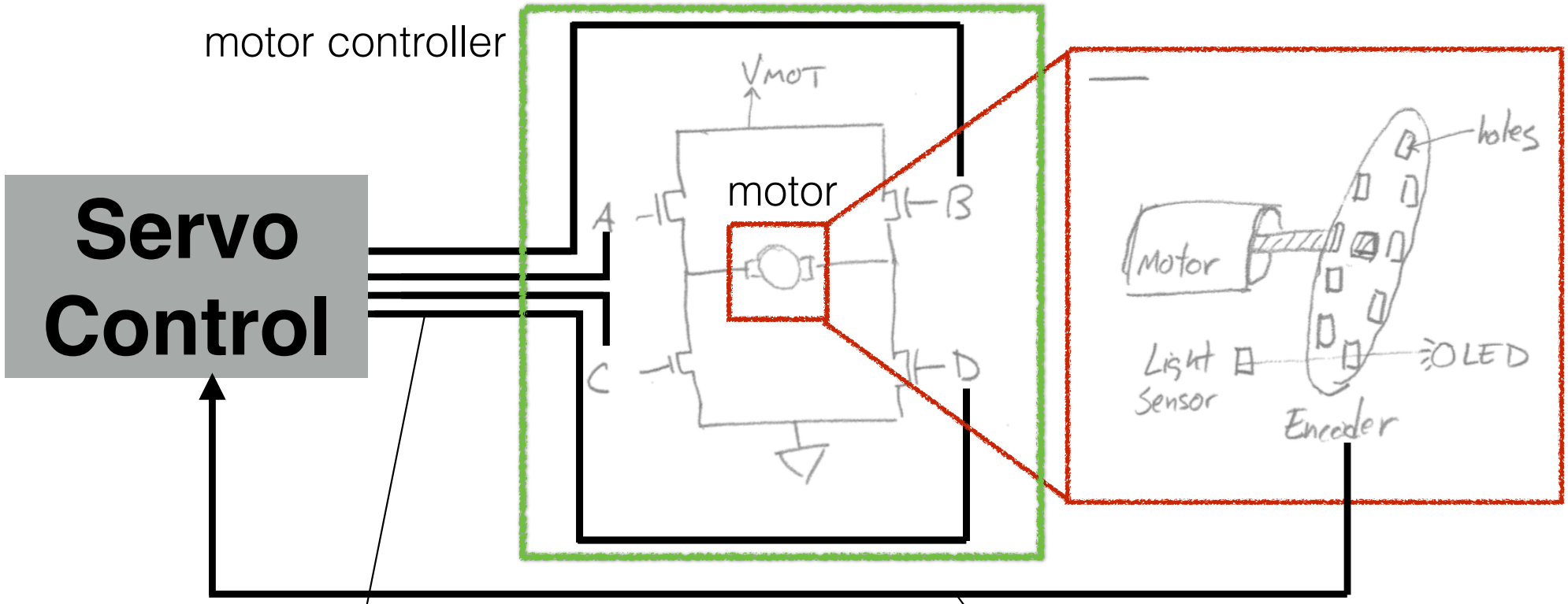
spin forward:
A and D

spin backward:
B and C

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

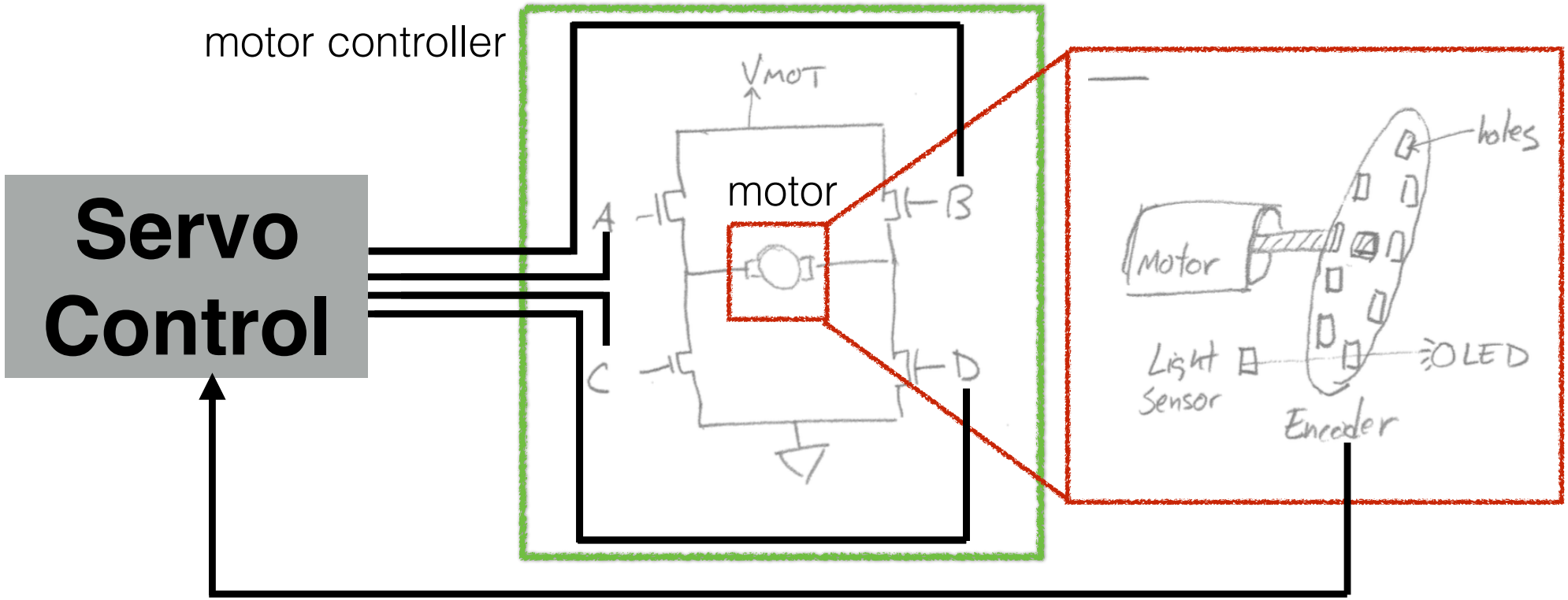
what if C and D?
negative torque -> brake

what if A and B?
negative torque -> brake

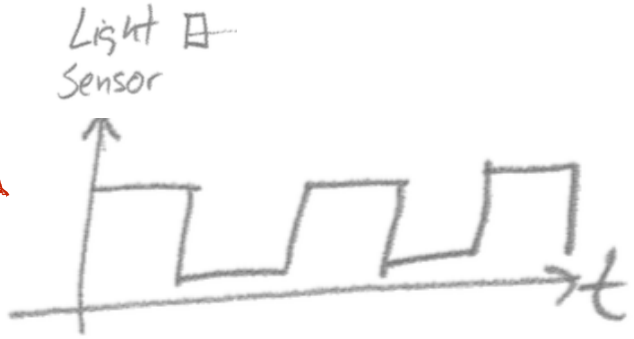


servo controller generates control signal

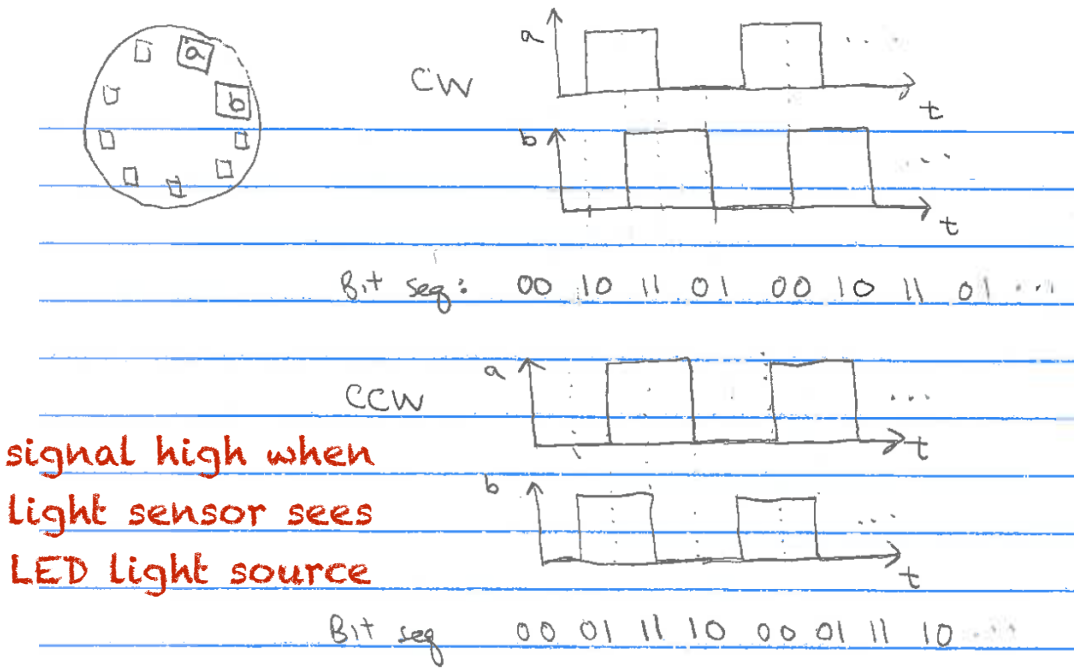
encoder sends joint state to back to servo controller

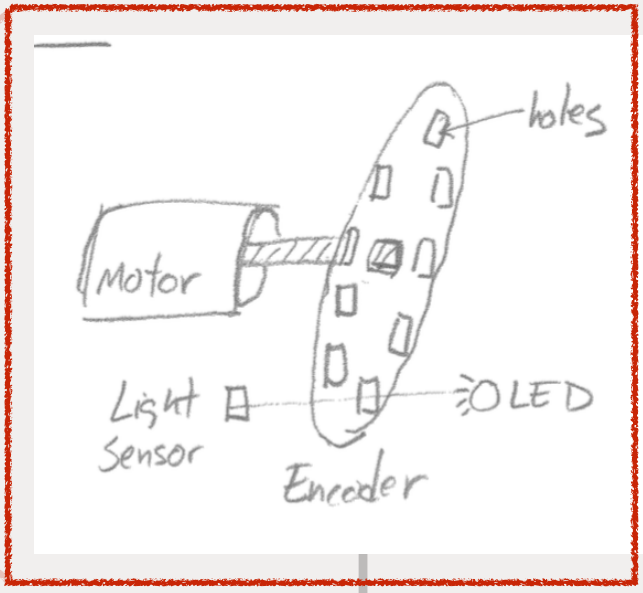


as motor turns,
encoder counts transitions between
sensed and occluded light



Quadrature Encoder

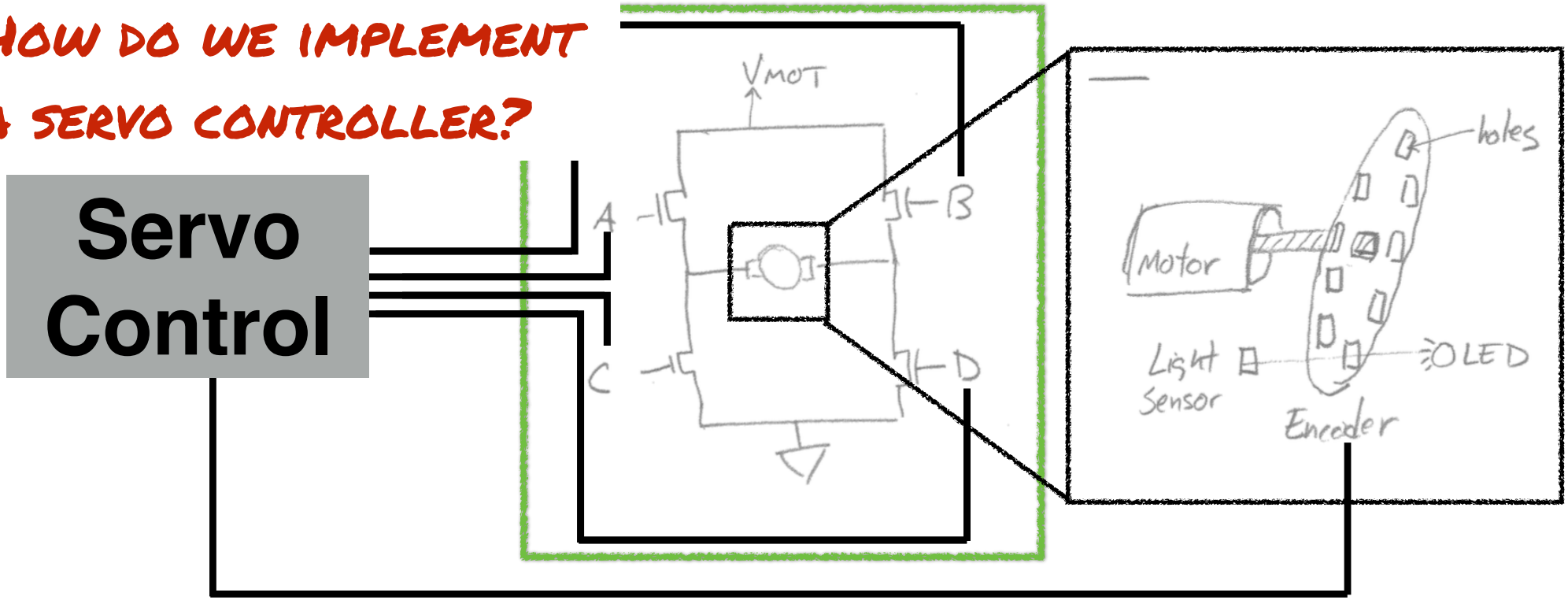




Encoders measure how much a motor has spun.

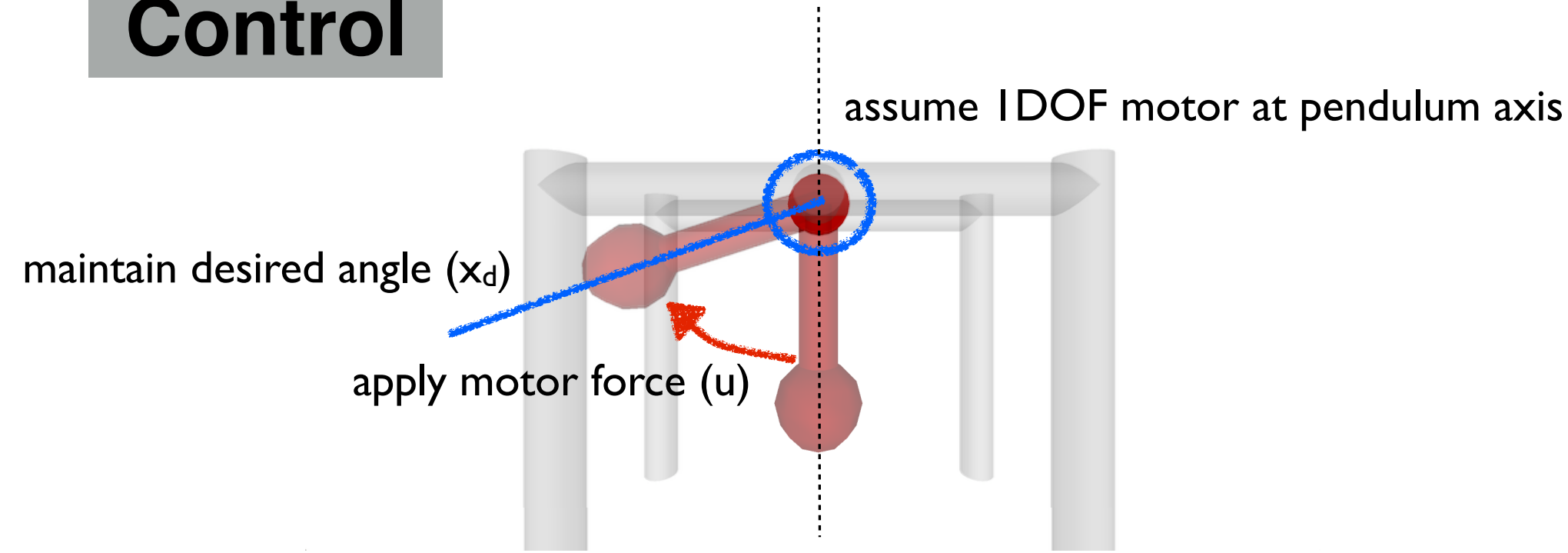
Quadrature encoders are commonly used. Why?

How do we implement a servo controller?

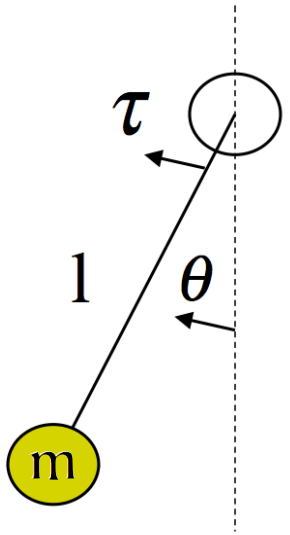


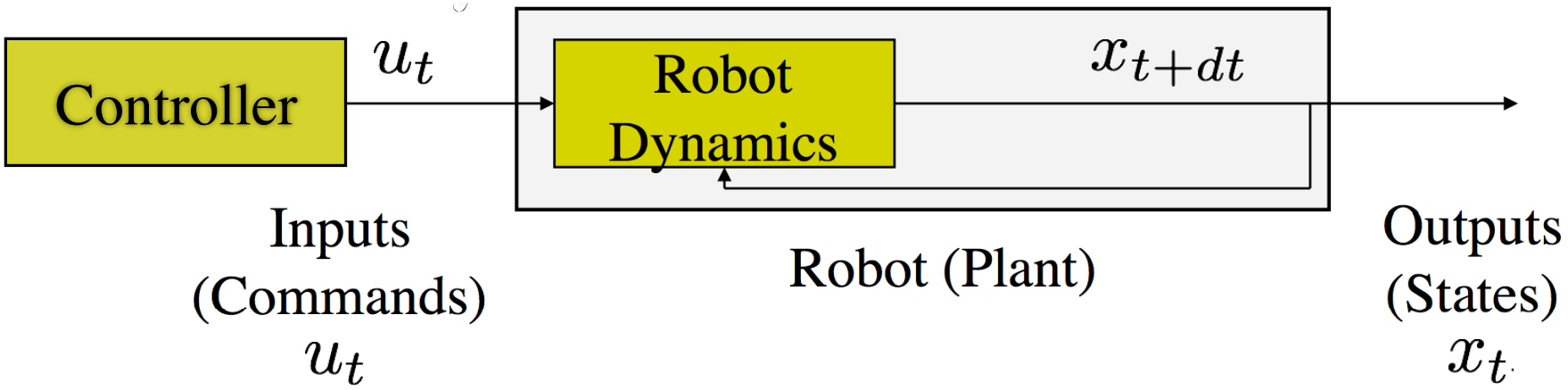
How do we implement a servo controller?

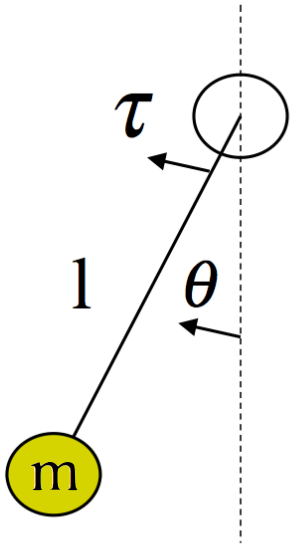
Servo Control



Servo Control



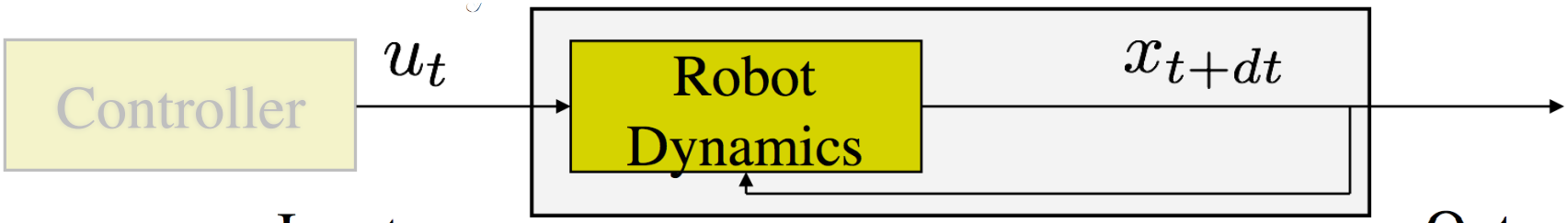




$$\ddot{\theta} = -\frac{g}{l}\sin(\theta) + \frac{\tau}{ml^2}$$

Motor command used within equations of motion

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



Inputs (Commands) u_t

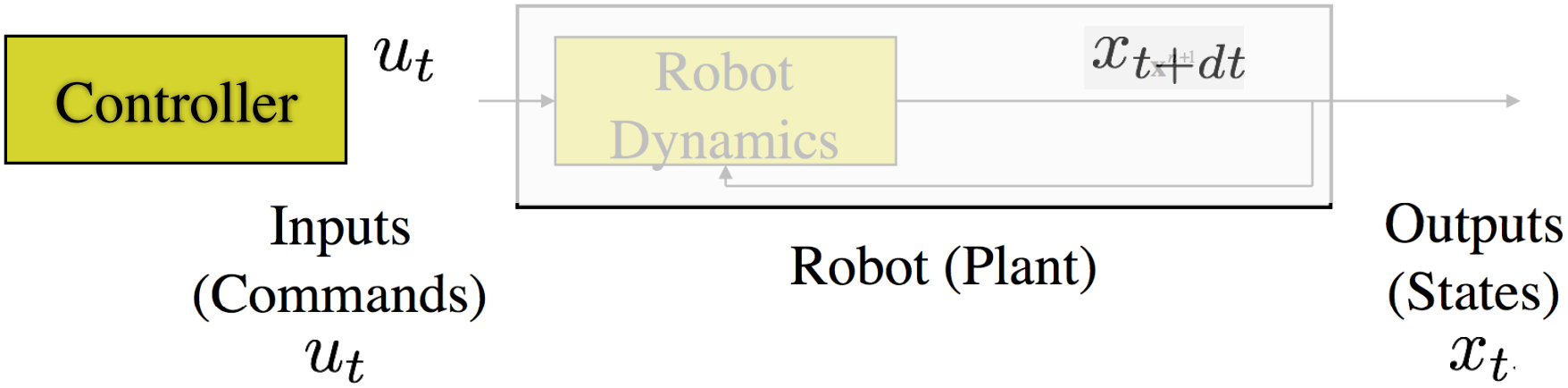
Robot (Plant)

Outputs (States) x_t

Robot "plant" represents system dynamics as function

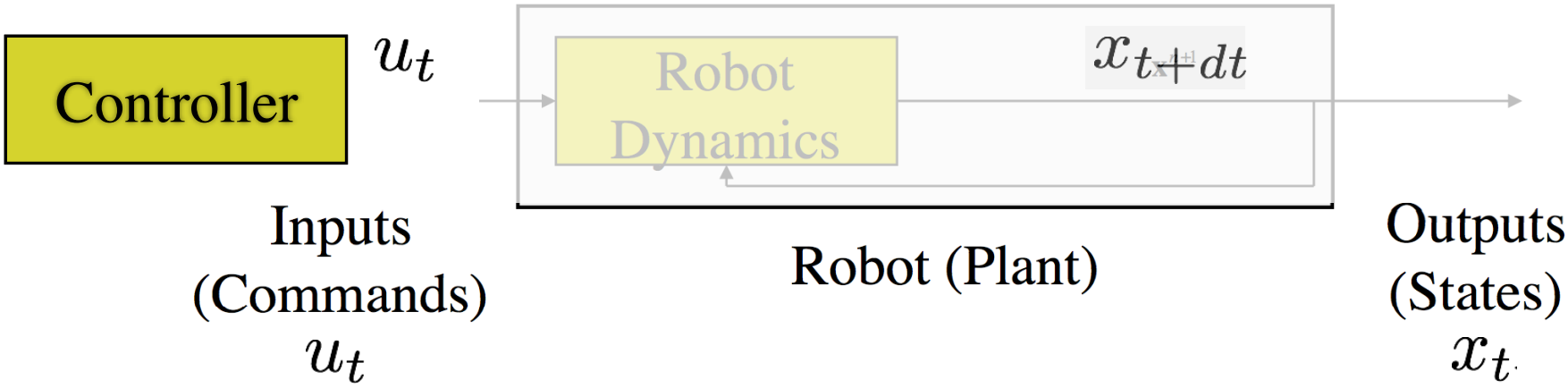
Servo controller generates motor commands based on desired state

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



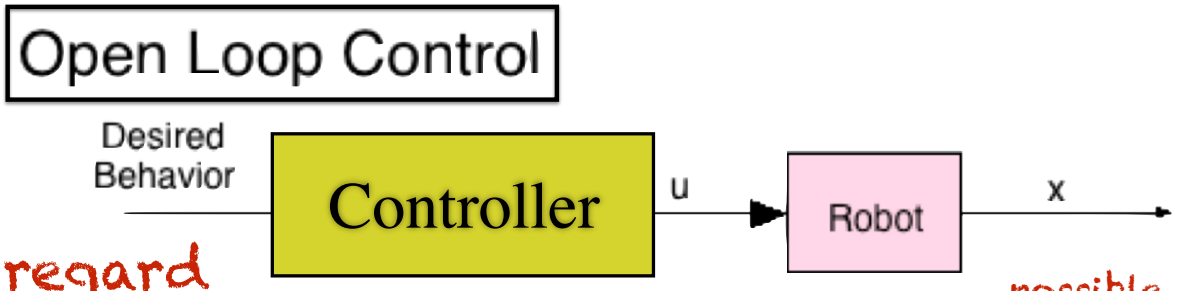
Servo controller generates motor commands based on desired state

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



π is a "control policy"

Control Policies

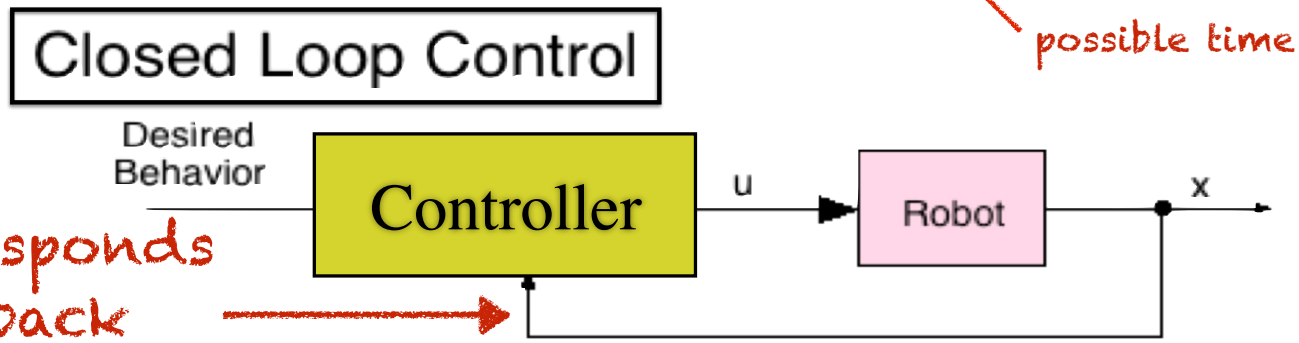


Acts without regard to environment

possible controller parameters

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

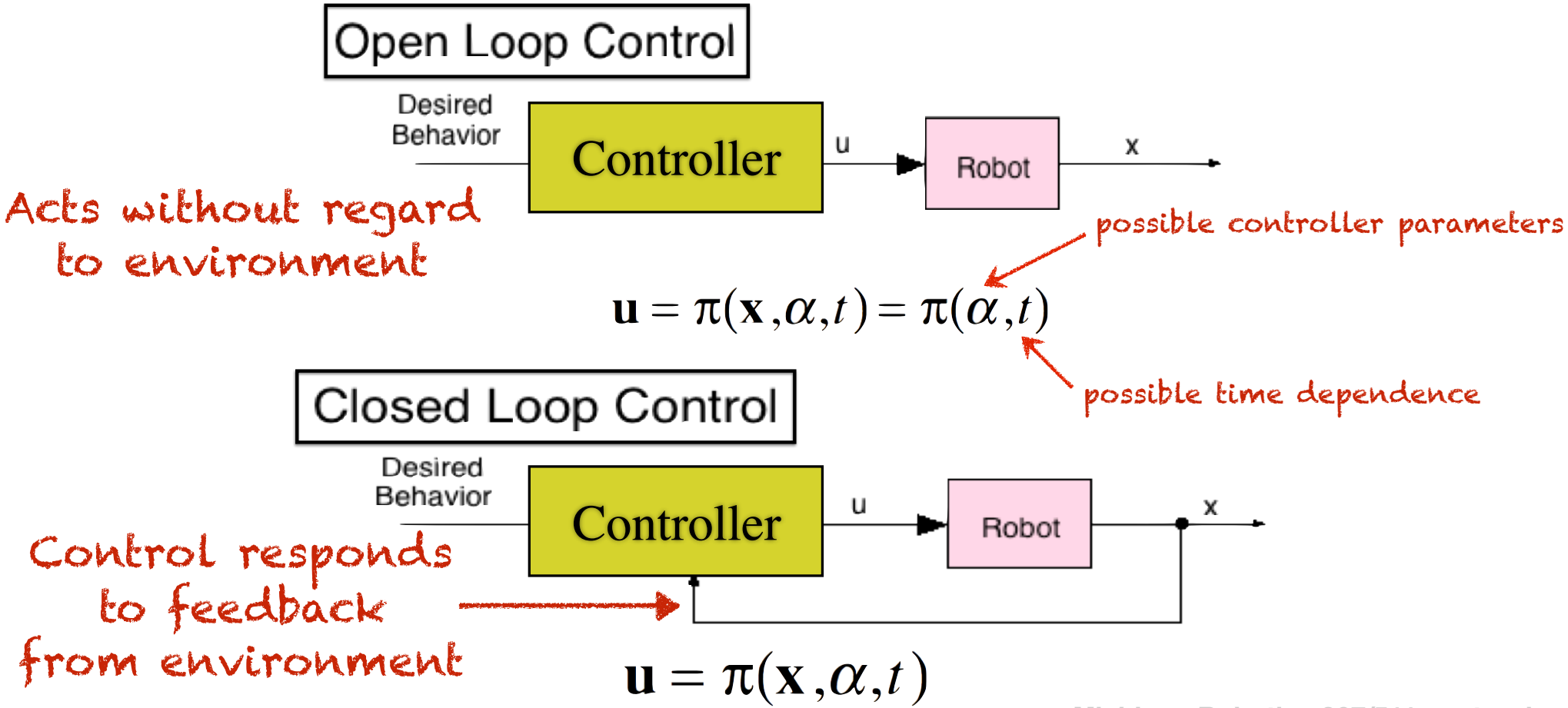
possible time dependence

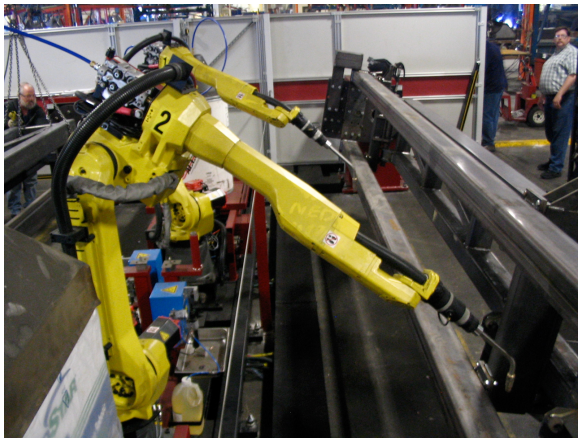


Control responds to feedback from environment

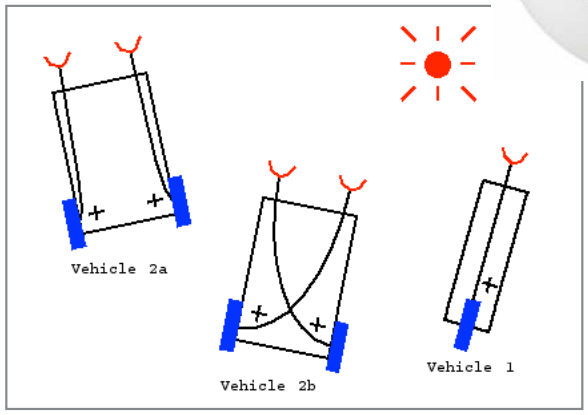
$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t)$$

When to use open loop vs closed loop?



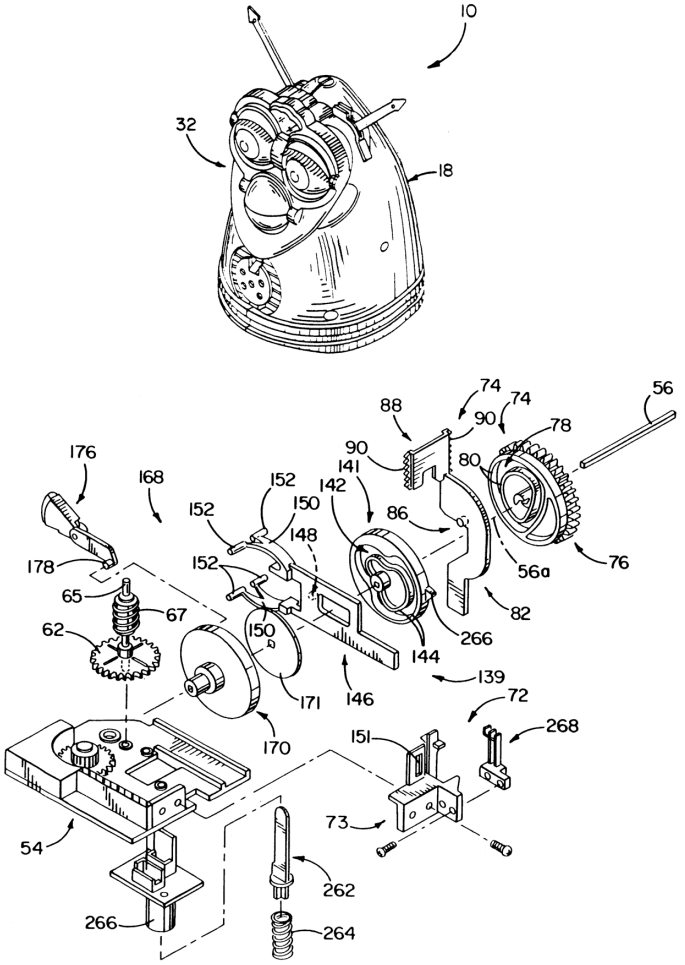


Which uses open loop?



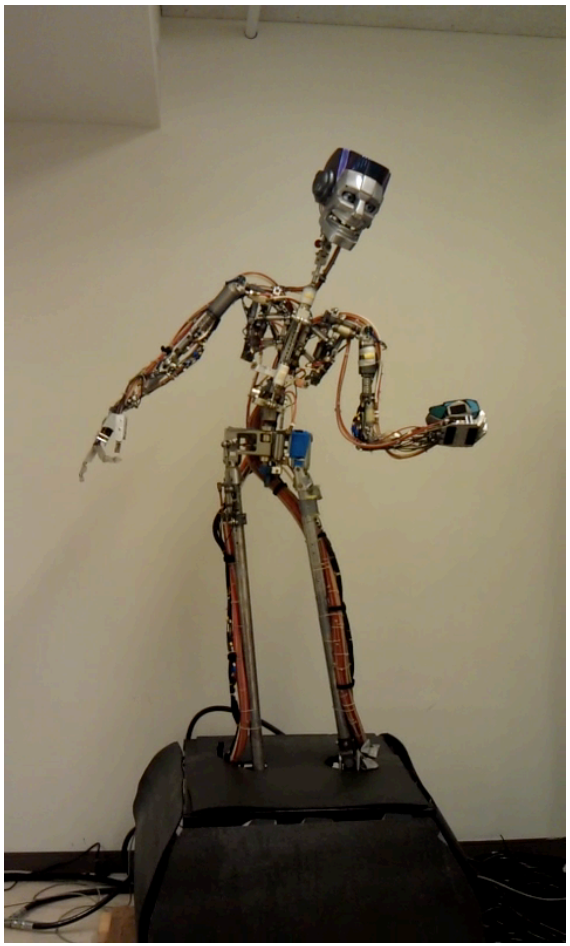

www.youtube.com/user/silvolf

Open loop: Furby



Hasbro

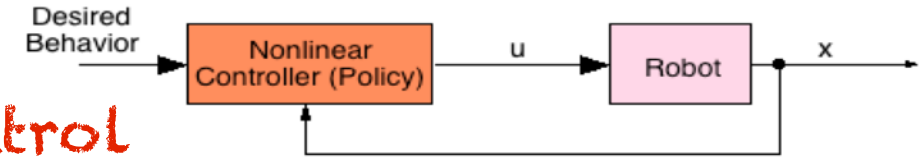
Disney



Types of Feedback Control

Types of Feedback Control

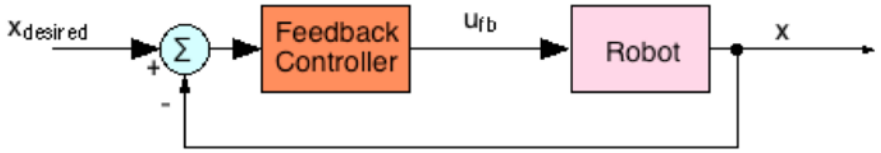
Feedback Control



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

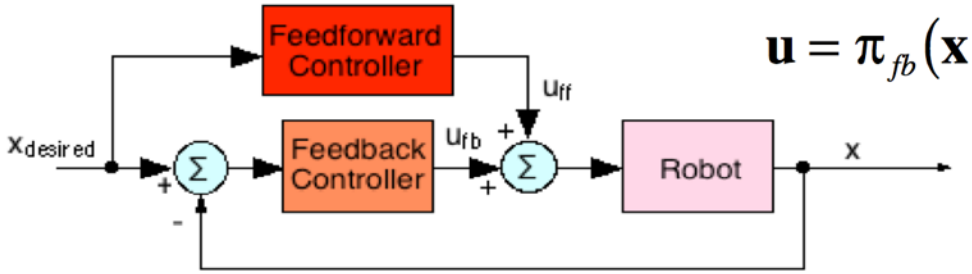
PID Control

Negative Feedback Control



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

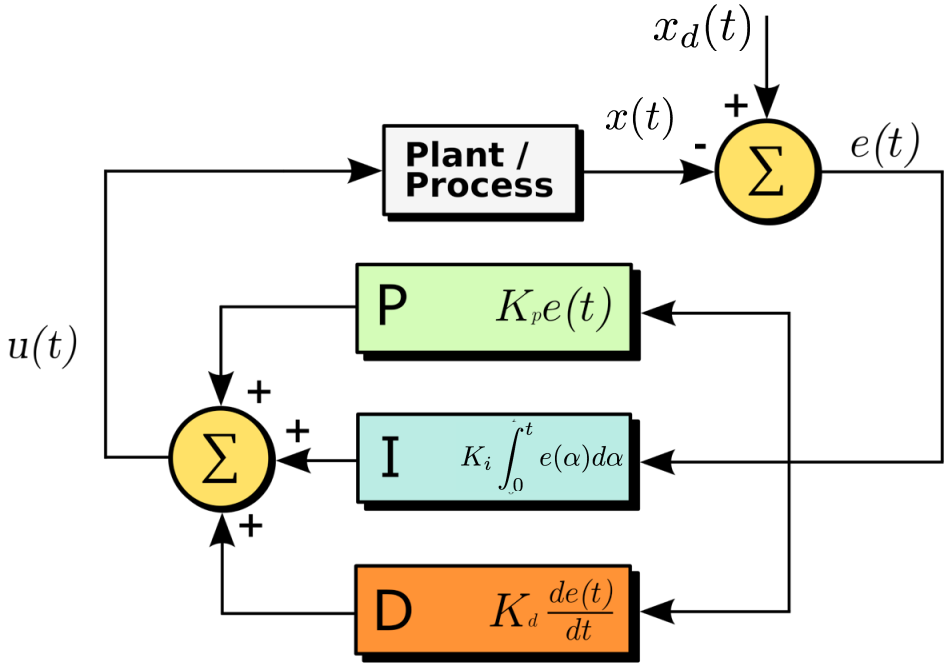
Neg. Feedback & Feedforward Control



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

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

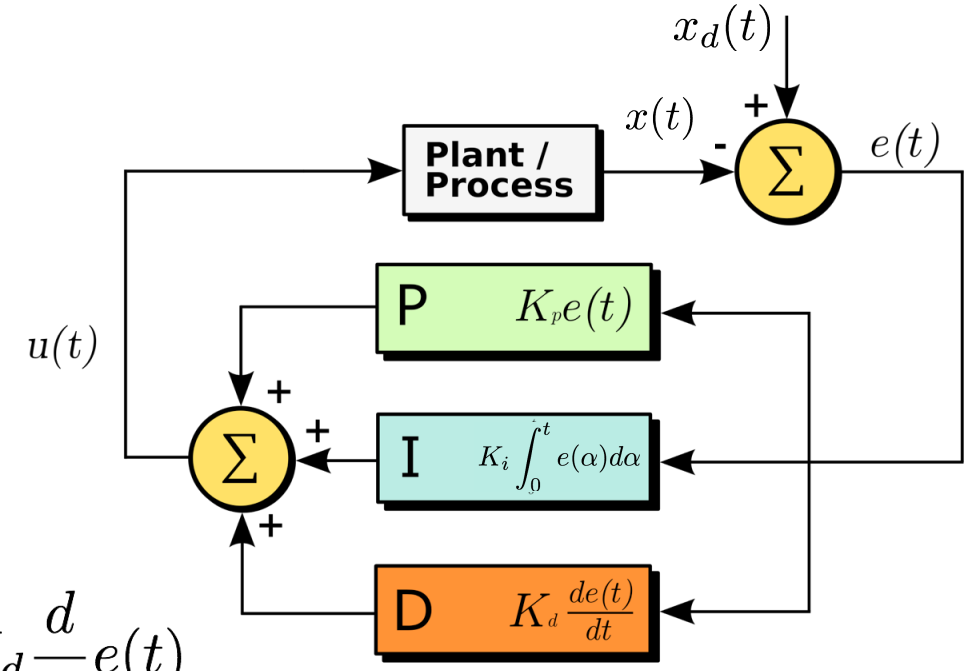


PID Control

for pendulum (x is θ)

Error signal:

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



Control signal:

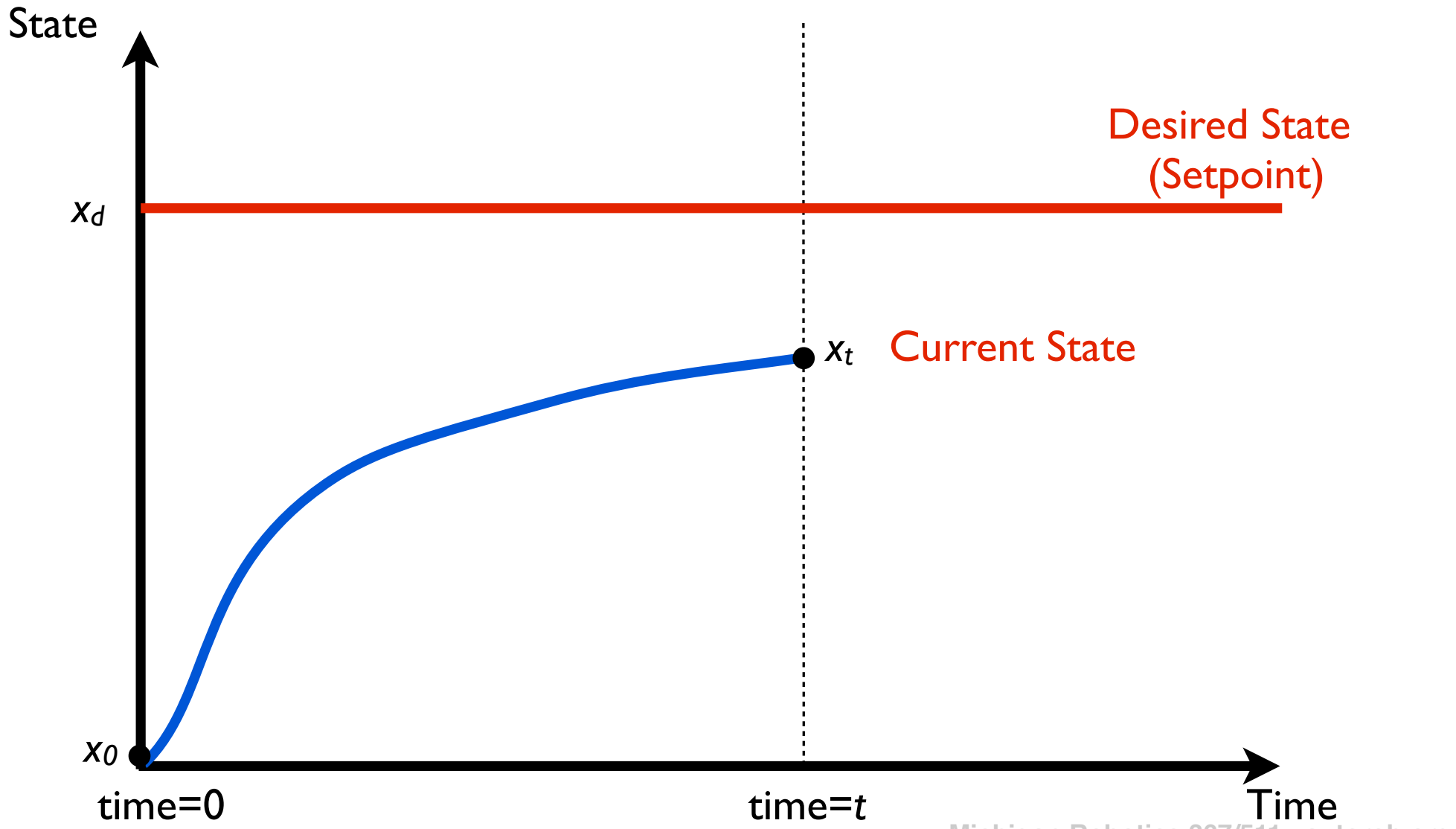
$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha)d\alpha + K_d \frac{d}{dt} e(t)$$

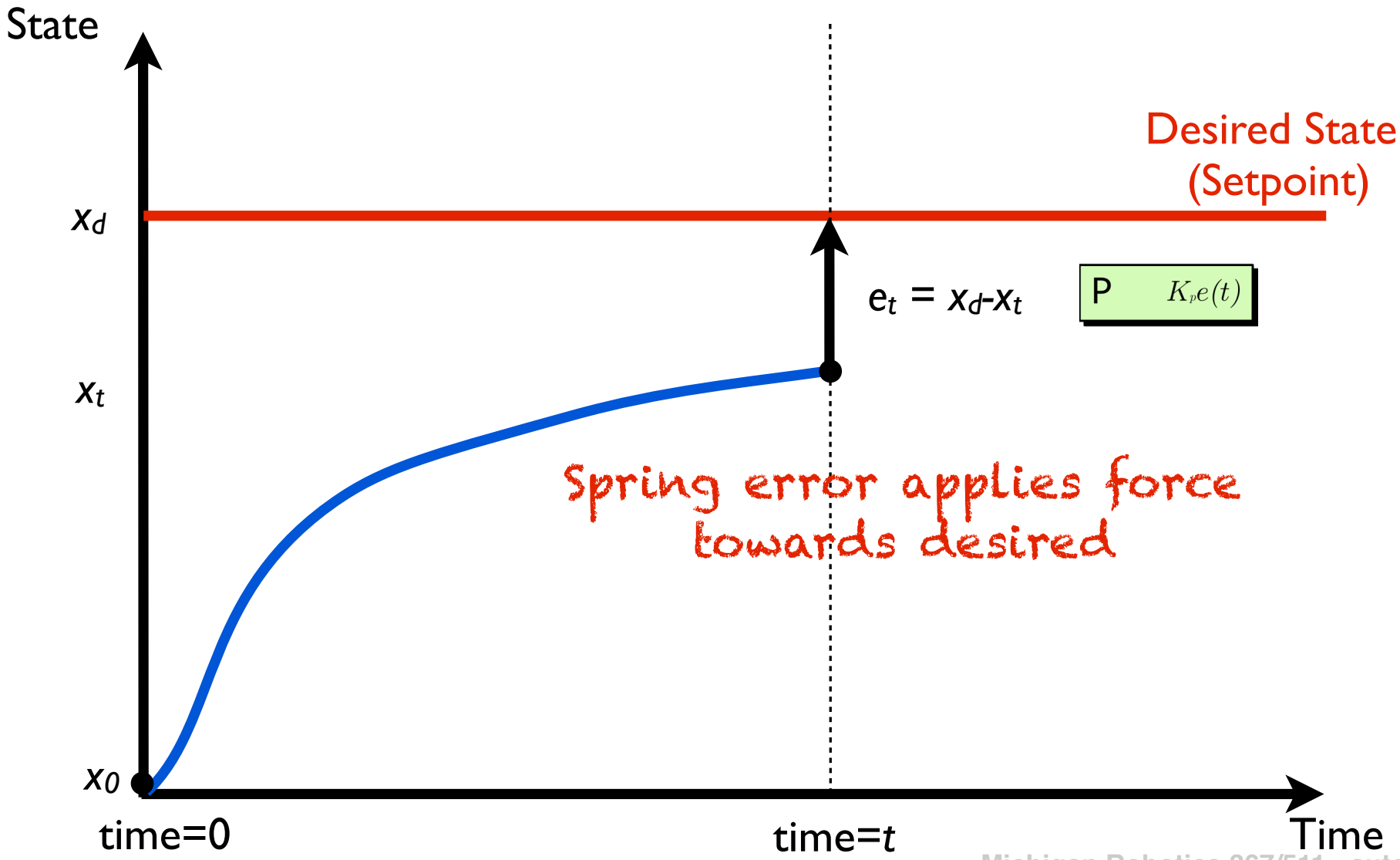
P $K_p e(t)$ — Current

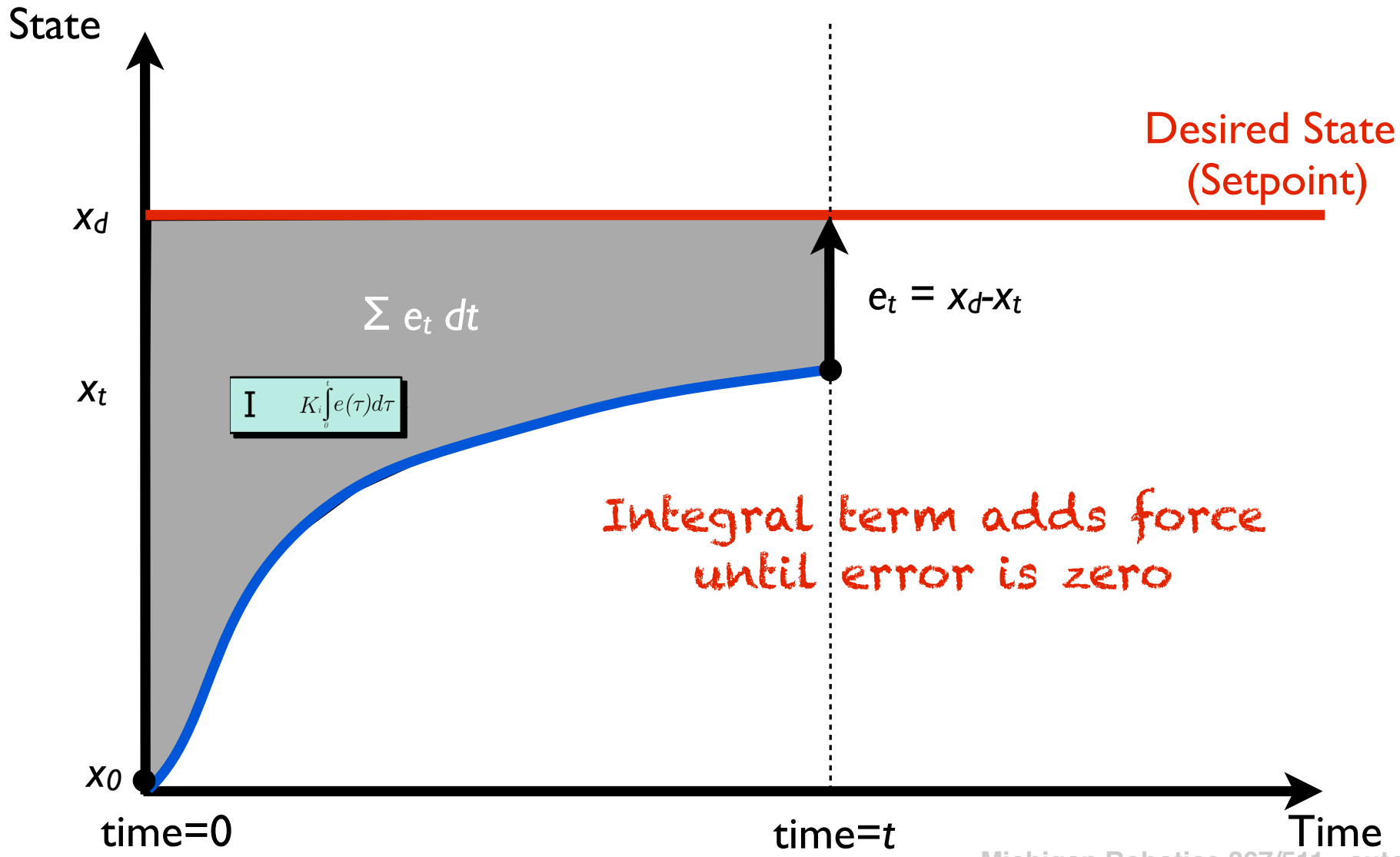
I $K_i \int_0^t e(\alpha)d\alpha$ — Past

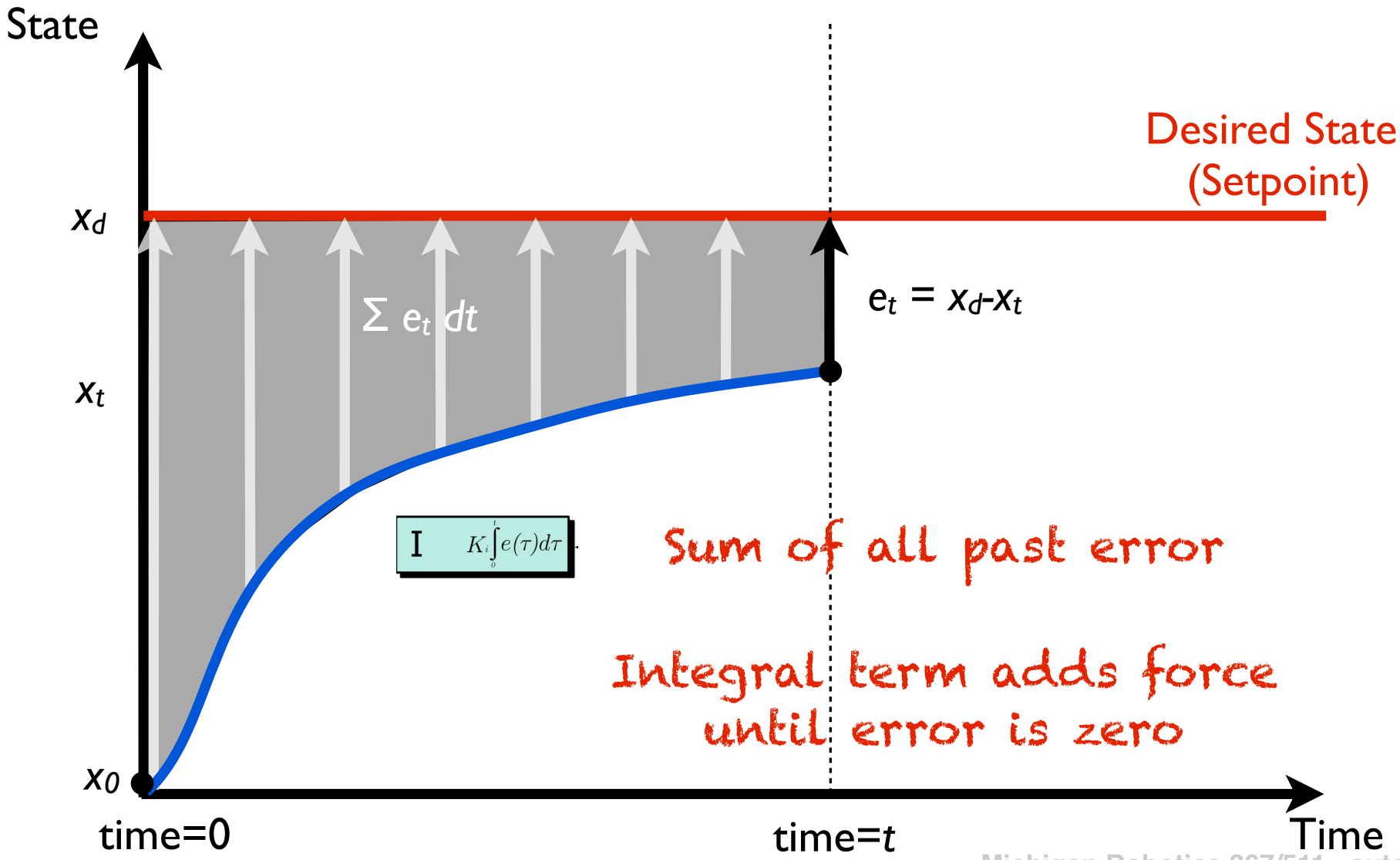
D $K_d \frac{de(t)}{dt}$ — Future

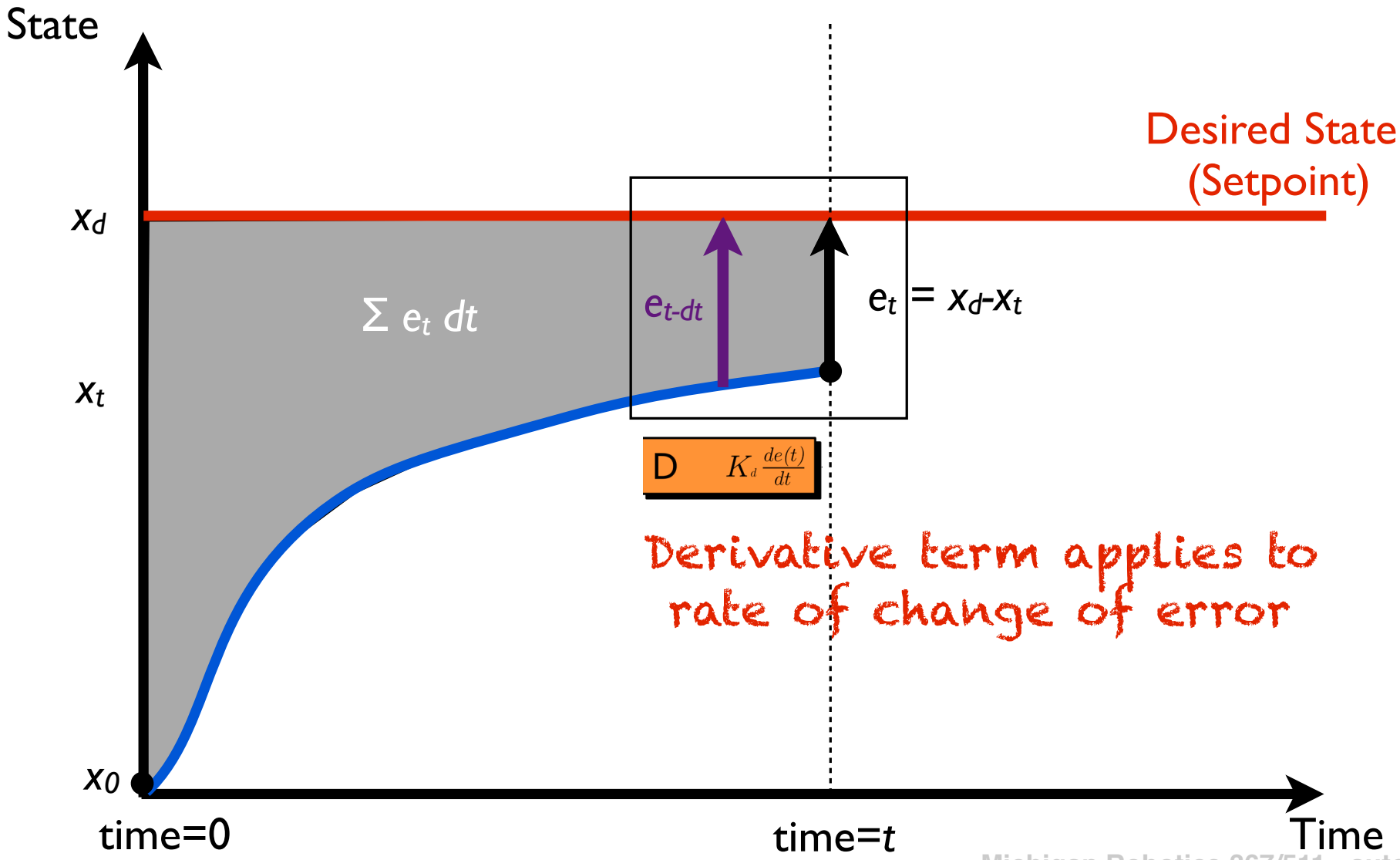
Consider PID wrt. state over time

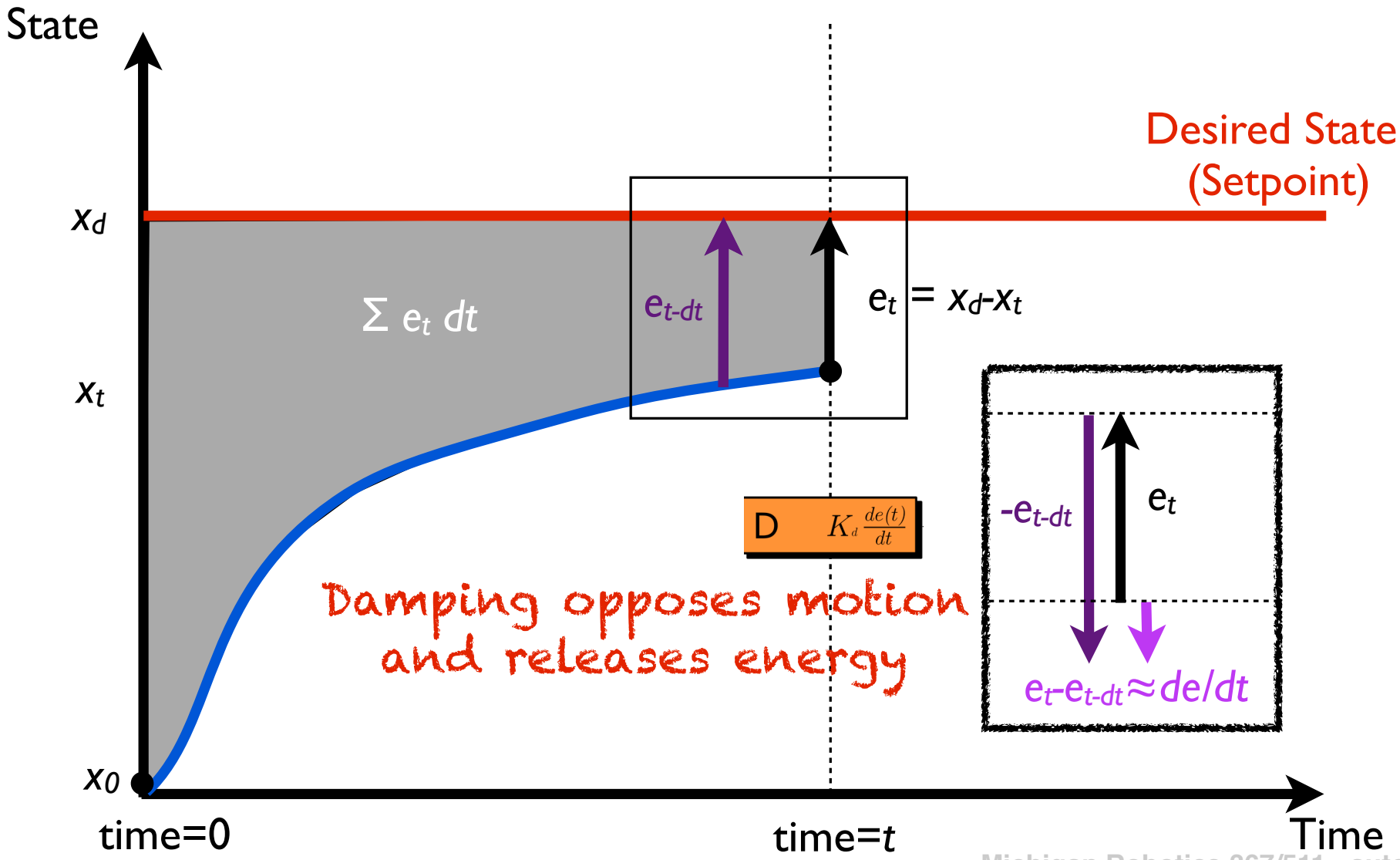

State
x_d
Desired State
(Setpoint)
x_t
Current State
x_0
time=0
time=t
Time

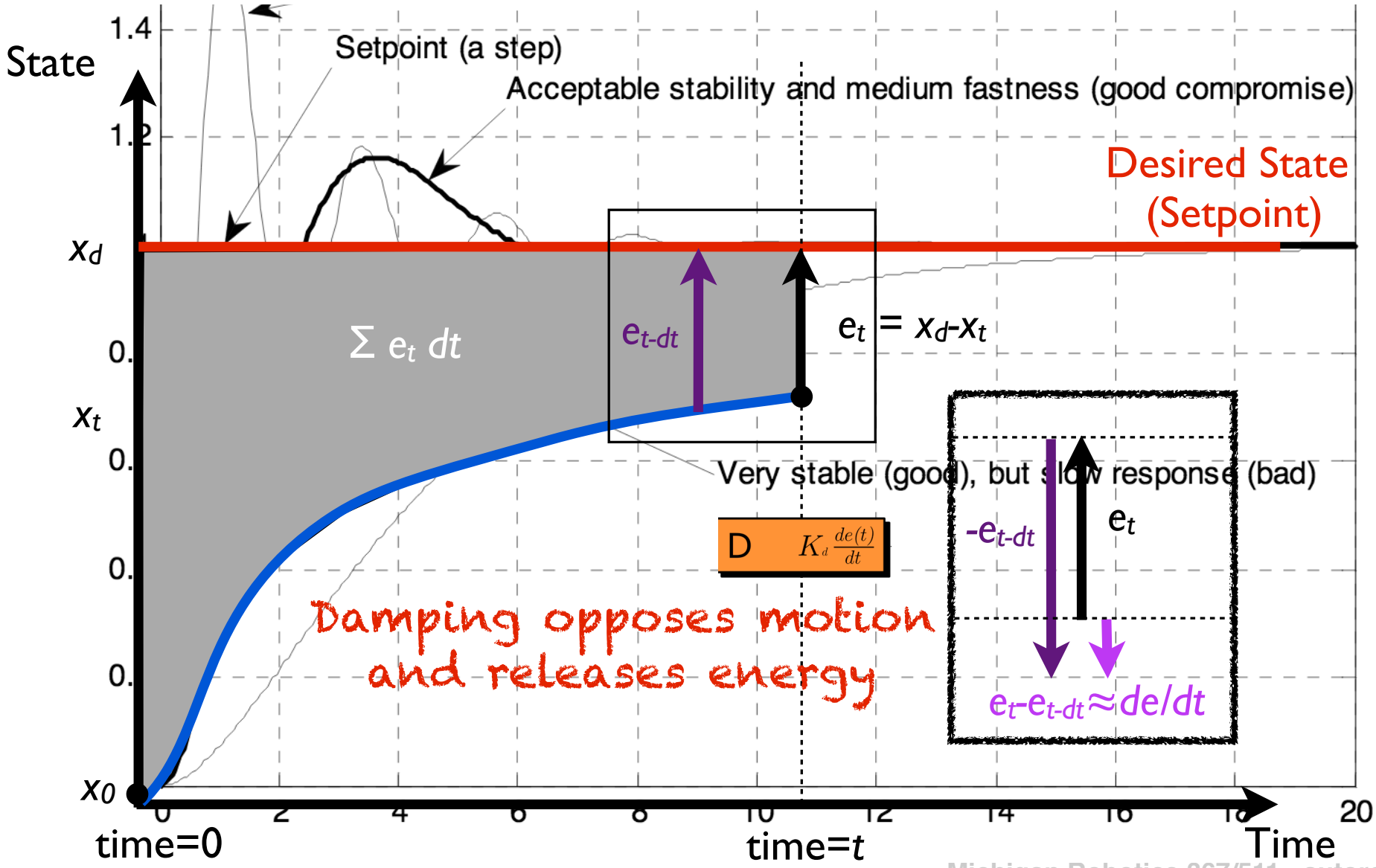

State
Desired State
(Setpoint)
x_d
$e_t = x_d - x_t$
P $K_p e(t)$
x_t
Spring error applies force
towards desired
x_0
time=0
time=t
Time


State
x_d
x_t
x_0
Desired State
(Setpoint)
$\Sigma\ e_t\ dt$
$e_t = x_d - x_t$
I $K_i \int_0^t e(\tau) d\tau$
Integral term adds force
until error is zero
time=0
time=t
Time

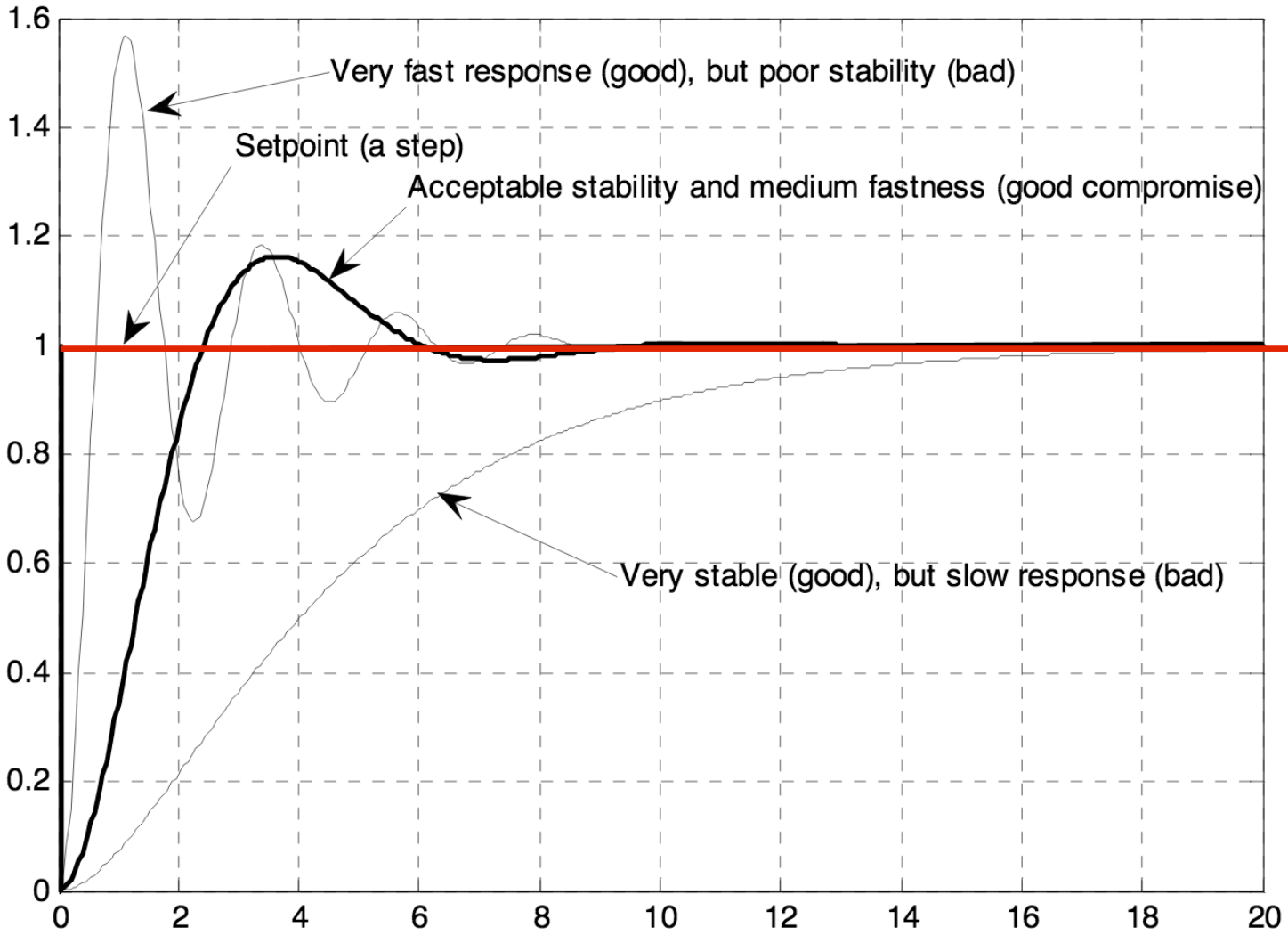

State
Desired State
(Setpoint)
x_d
x_t
x_0
$\Sigma\ e_t\ dt$
$e_t = x_d - x_t$
$I \quad K_i \int_0^t e(\tau) d\tau$
Sum of all past error
Integral term adds force
until error is zero
time=0
time=t
Time


State
Desired State
(Setpoint)
x_d
x_t
x_0
$\Sigma\ e_t\ dt$
e_{t-dt}
$e_t = x_d - x_t$
D $K_d \frac{de(t)}{dt}$
Derivative term applies to rate of change of error
time=0
time=t
Time


State
x_d
x_t
x_0
Desired State
(Setpoint)
$\Sigma\ e_t\ dt$
e_{t-dt}
$e_t = x_d - x_t$
D $K_d \frac{de(t)}{dt}$
Damping opposes motion
and releases energy
$-e_{t-dt}$
e_t
$e_t - e_{t-dt} \approx de/dt$
time=0
time=t
Time


State
1.4
1.2
0.
0.
0.
0.
0.
Setpoint (a step)
Acceptable stability and medium fastness (good compromise)
Desired State
(Setpoint)
x_d
x_t
x_0
$\Sigma\, e_t\, dt$
e_{t-dt}
$e_t = x_d - x_t$
Very stable (good), but slow response (bad)
D $K_d \frac{de(t)}{dt}$
Damping opposes motion
and releases energy
$-e_{t-dt}$
e_t
$e_t - e_{t-dt} \approx de/dt$
0
2
4
6
8
10
12
14
16
18
20
time=0
time=t
Time

PID Converenge



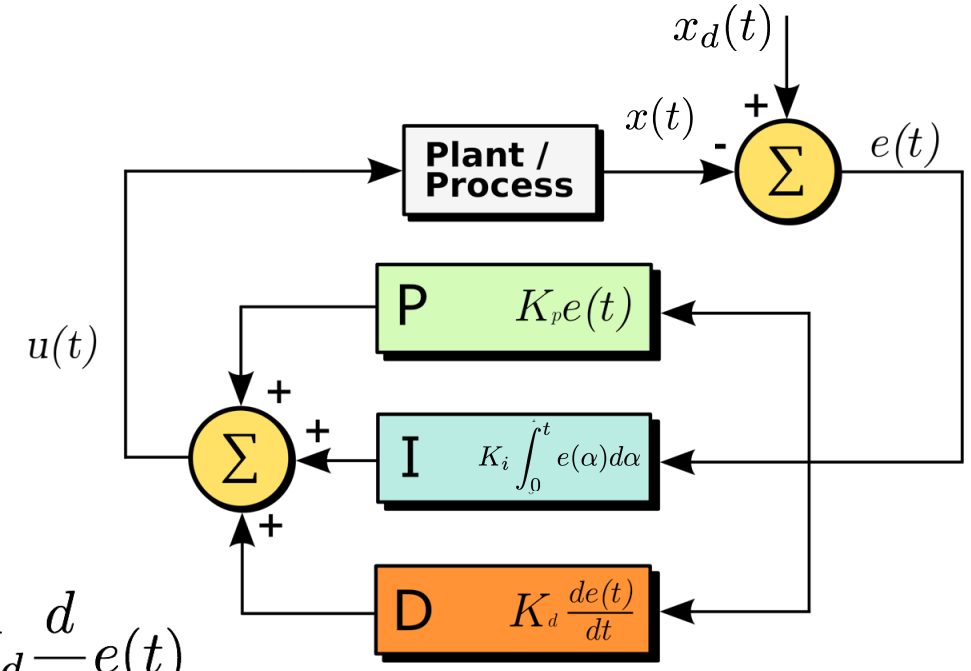
http://home.hit.no/~hansha/documents/control/theory/tuning_pid_controller.pdf

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 $K_p e(t)$ — Current

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

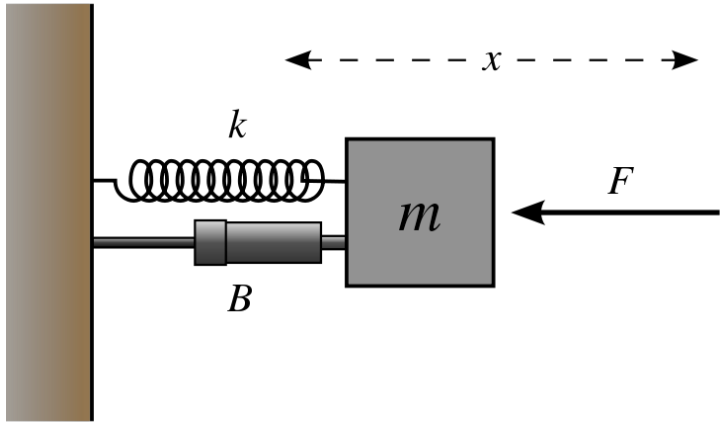
D $K_d \frac{de(t)}{dt}$ — Future

Hooke's Law

P $K_p e(t)$

- Describes motion of mass spring damper system as

$$F = -kx$$

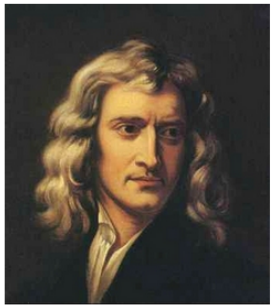


Robert Hooke
(1635-1703)

The Rivalry Between Isaac Newton and Robert Hooke

Updated on October 20, 2015

Melvin Porter more
Contact Author



Isaac Newton | Source

Source

HOME » EDITORIAL » ROBERT HOOKE AND THE WRATH OF ISAAC NEWTON

Robert Hooke and the Wrath of Isaac Newton

Posted By: Daryl.Worthington on: August 22, 2016 In: Editorial No Comments Print Email

His pioneering work with microscopes led to his famous engraving of a fly's eye. Just as remarkably, his early studies of petrified wood and other fossils made him one of the first to realise they were remains of once living things – a fact so well known now it seems obvious, but revolutionary at the time. How did a man involved in so much innovation, slip into obscurity? It is not as if Hooke was unappreciated in his own time. His book, Micrographia, was a bestseller, inspiring interest in the use of microscopes. He was a longtime president of the Royal Society, showing the regard in which his scientific work was held. However, he also had a powerful rival: Sir Isaac Newton. The two clashed bitterly in attempts to forge reputations as the greatest scientific minds of their age. In life this battle may have been a close run thing, but through history Newton became the undisputed winner. The first signs of conflict between these two massive egos came in 1672, when Newton submitted his first paper to the Royal Society. In the work, Newton claimed light was a particle, contradicting

THE ROYAL SOCIETY The Repository

Hooke, Newton, and the 'missing' portrait

3 December 2010 by Felicity Henderson

Portraits have a peculiar fascination for people. As Lisa Jardine pointed out recently, historical figures come to life so much more vividly when a portrait is available. This is true for historians almost as much as anyone else. Therefore the thought that there might be an undiscovered or lost portrait of a famous and controversial figure like Robert Hooke is extremely tantalising. It also grips the public imagination – several visitors to the Royal Society's 350th anniversary exhibition over the summer commented, 'they say Newton destroyed a portrait of Hooke'. Indeed ,'they' do say this. The final scene in the 2009 Royal Shakespeare Company production 'The Tragedy of Thomas Hobbes' showed Newton slashing a portrait, a reference that shows how familiar this story has become.

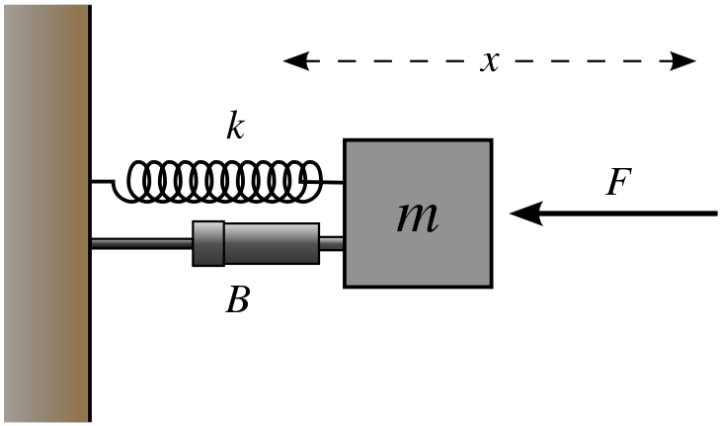
So the question is, did Newton do it? He seems to have had both motive and opportunity. His relations with Hooke had turned sour in 1686 following controversy over Hooke's contributions to Newton's theory of gravity. After Hooke's death in 1703 Newton was elected President of the Royal Society ('they' also say he waited until Hooke had died before becoming more active in the Society). Newton oversaw the Society's move to a new premises in Crane Court, and it is assumed that the portrait went missing during this move. Finally, Newton was a ruthless and overbearing character who held grudges – or so they say.

But before we pronounce Newton guilty of destroying Royal Society property, we need to consider the fundamental question of whether a portrait of Hooke existed in the first place

Hooke's Law

P $K_p e(t)$

- Describes motion of mass spring damper system as



$$F = -kx$$

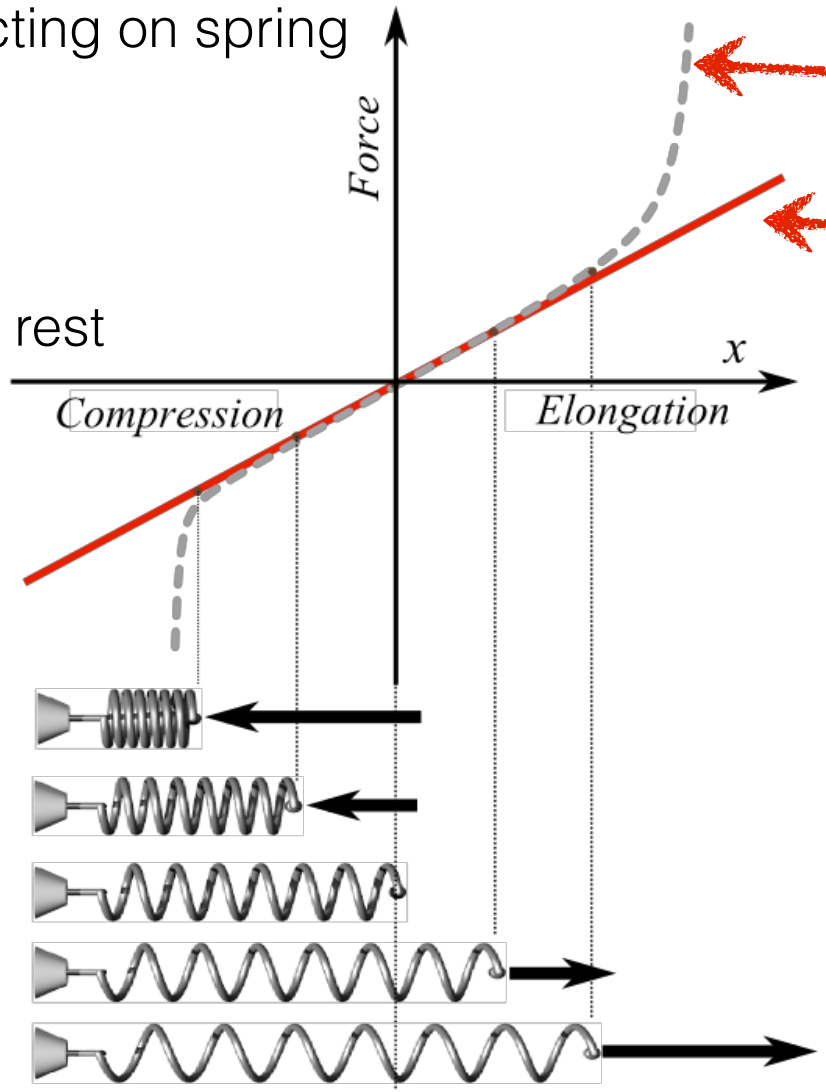
force moving spring towards rest

spring stiffness

distance from rest displacement

Vertical: Force acting on spring

Horizontal: Position from rest



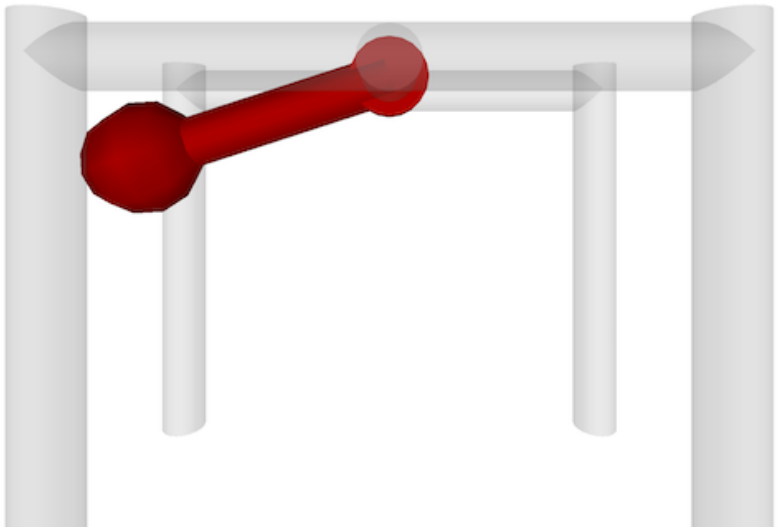
actual force relation

linear approximation (-kx)

$K_p e(t)$


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

Let's see what happens

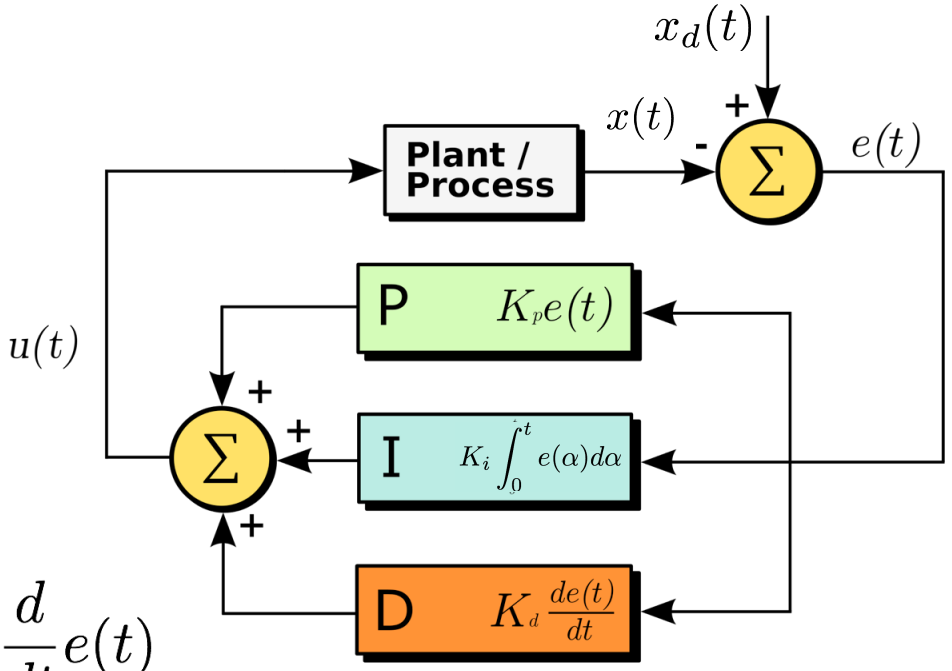


overstiff
understiff

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

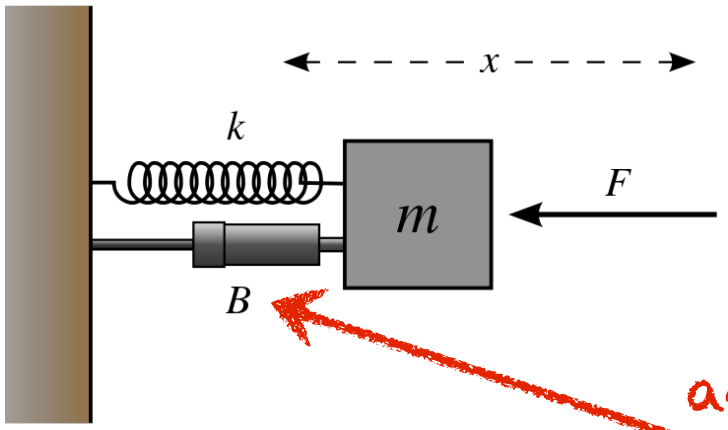
D $K_d \frac{de(t)}{dt}$ — Future

Spring and Damper

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

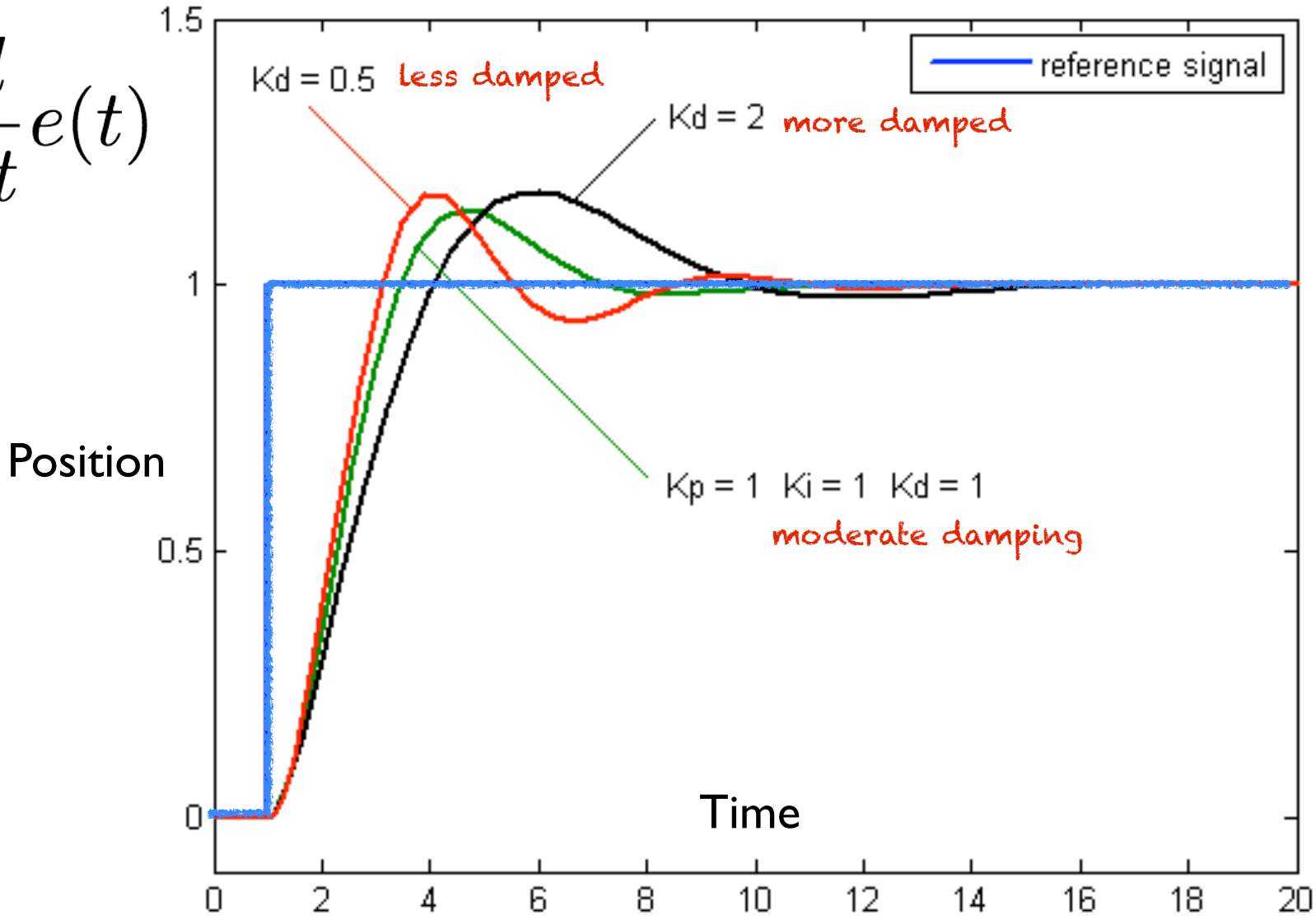
$$F = -kx + -b\dot{x}$$

assuming constant set point, velocity is derivative of error

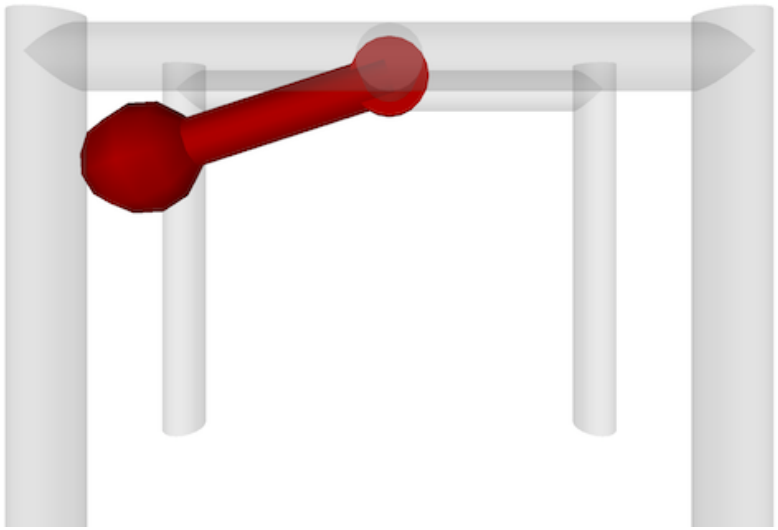


add damper to release energy

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



Let's see what happens

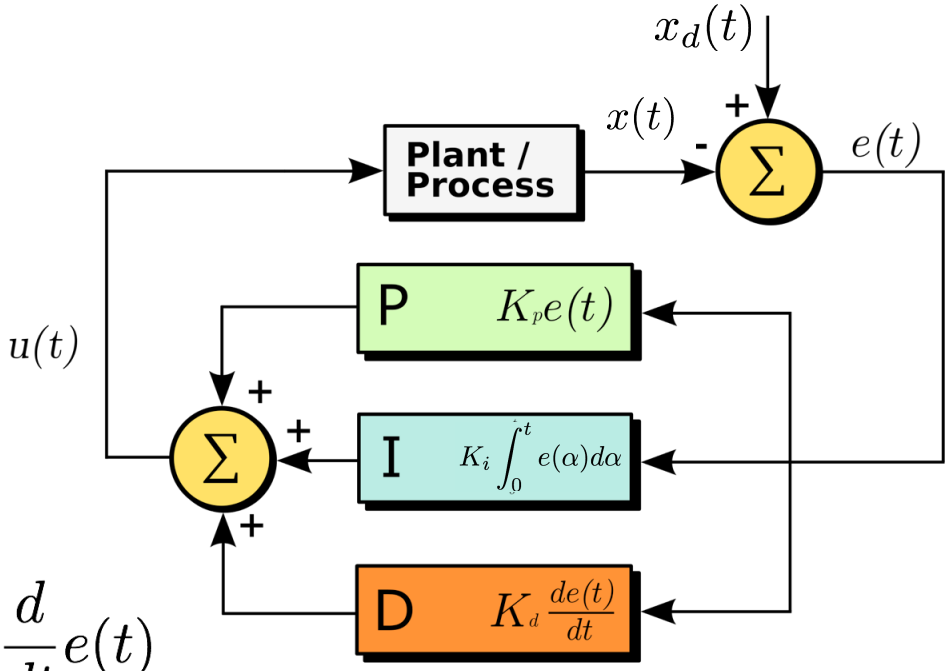


steadystate

PID Control

Error signal:

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



Control signal:

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

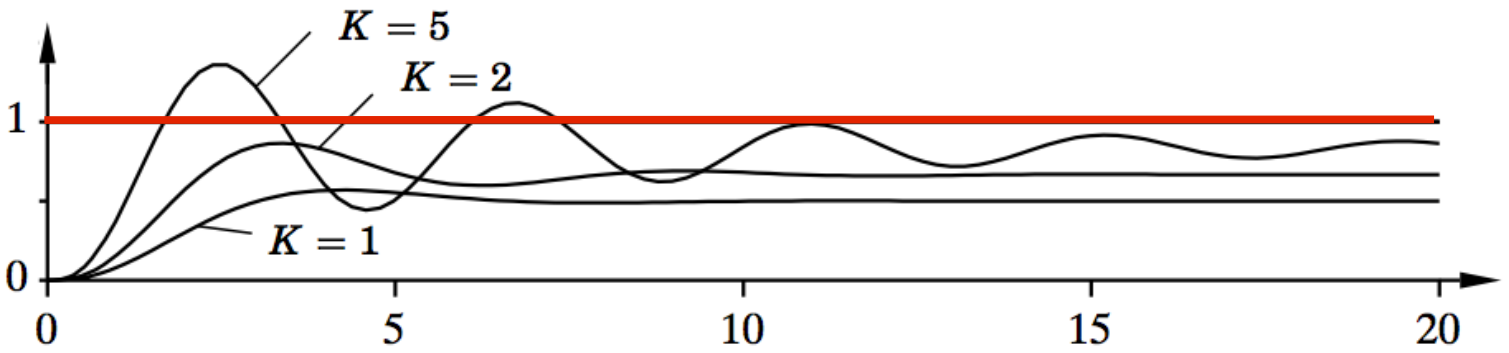
P $K_p e(t)$ — Current

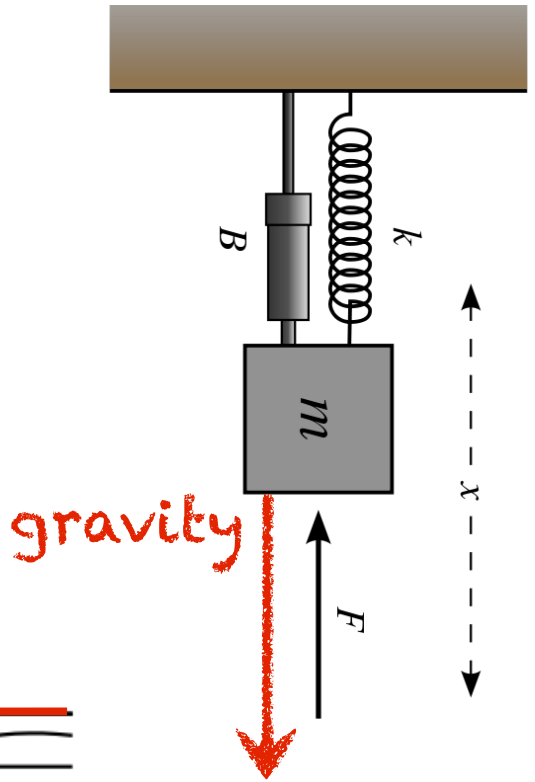
I $K_i \int_0^t e(\alpha)d\alpha$ — Past

D $K_d \frac{de(t)}{dt}$ — Future

Steady state error

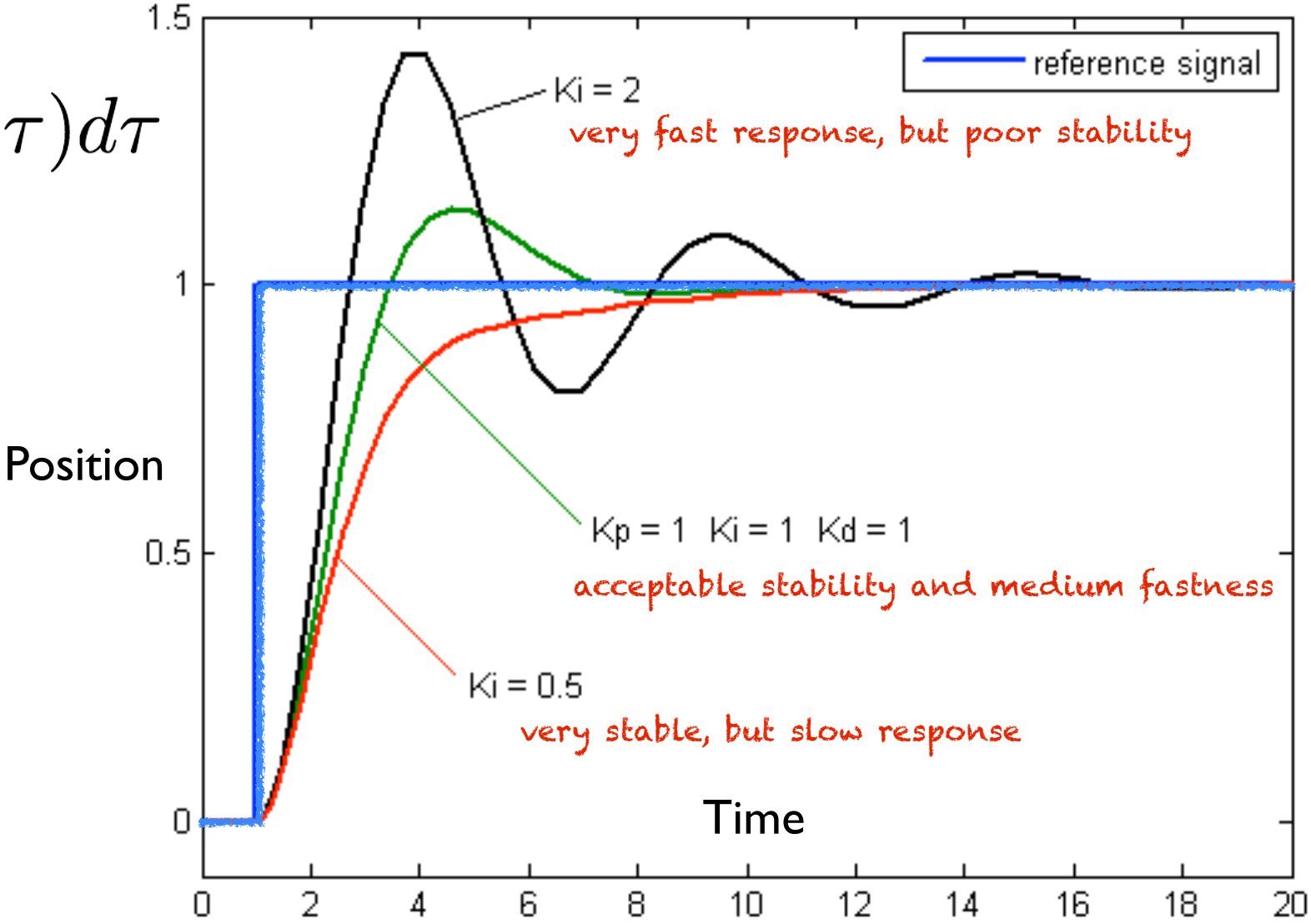
- Steady state error occurs when the system rests at equilibrium before reaching desired state
- Cause could be an significant external force, weak motor, low proportional gain, etc.
- PID integral term compensates by accumulating and acting against error toward convergence



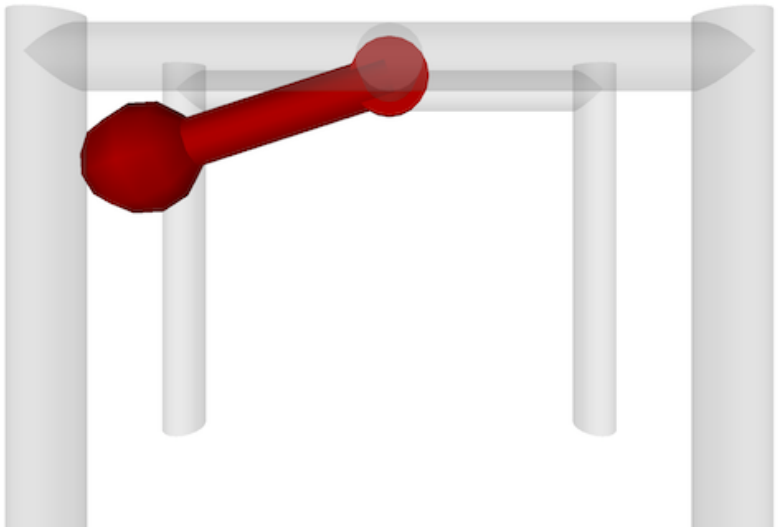


$$K_i \int_0^t e(\tau)d\tau$$

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.

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

P $K_p e(t)$

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

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

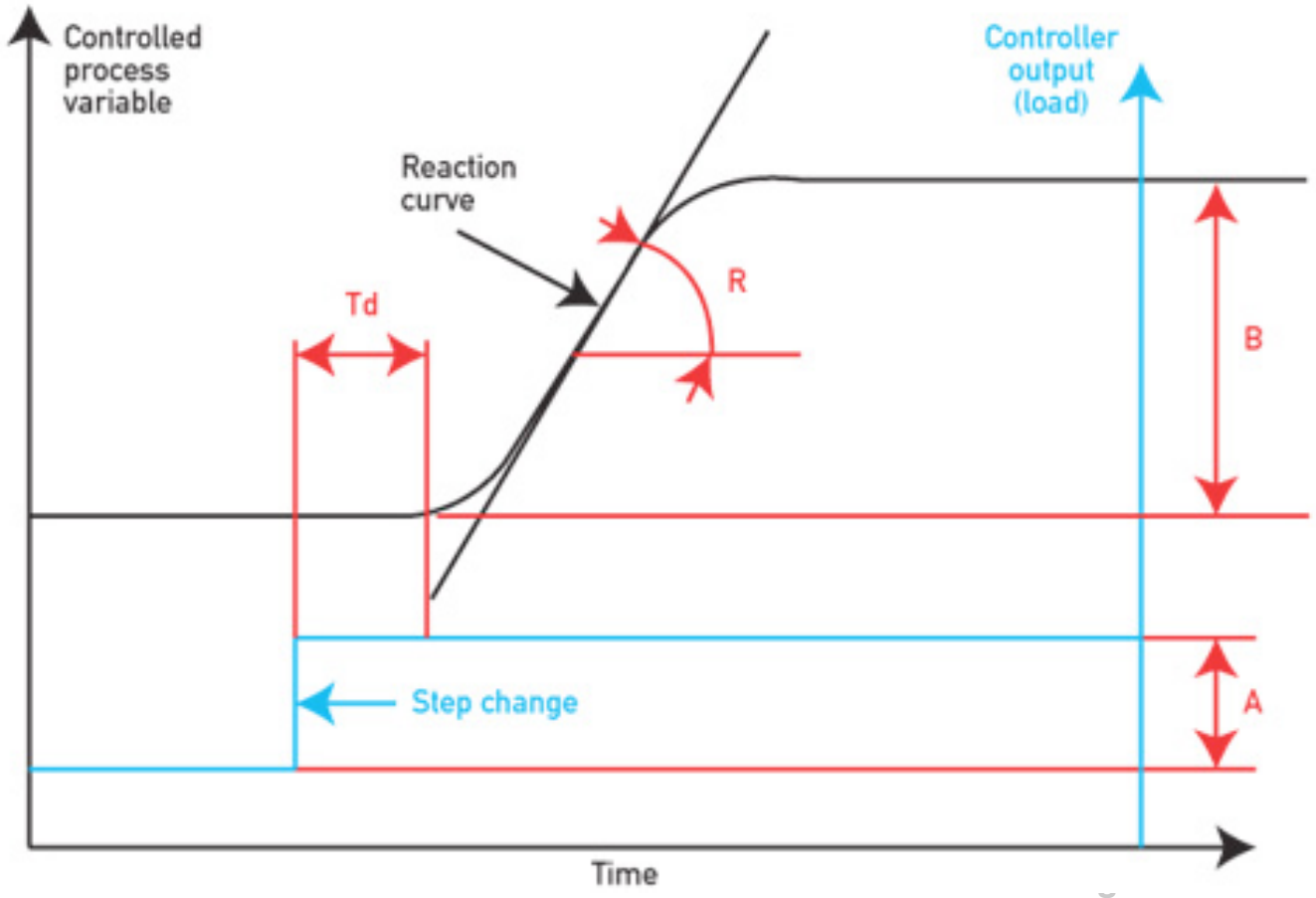
PID controller

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

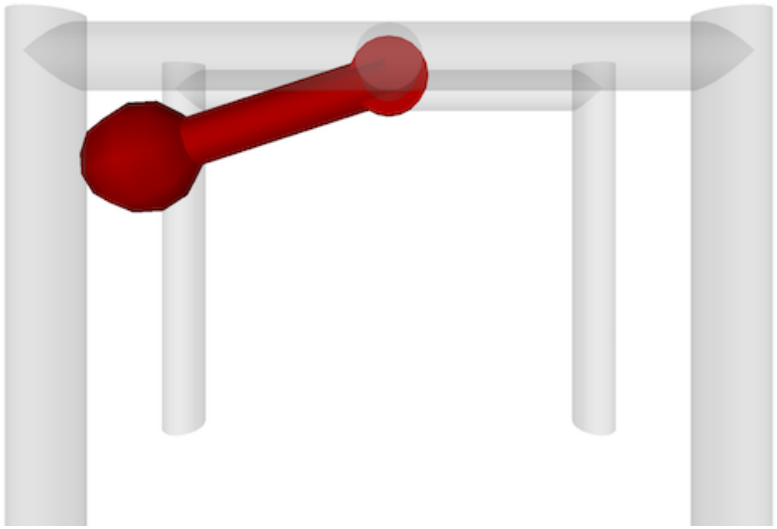
My approach to PID tuning

(take it or leave it)

- Start 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 stabilizes motion, 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)

Putting it all together (Pendularm)



Project 2: Pendularm

THIS LECTURE

Feedback Control

PID Control of 1 DoF robot

LECTURE 4

Dynamical Simulation

Equations of motion: Simple pendulum

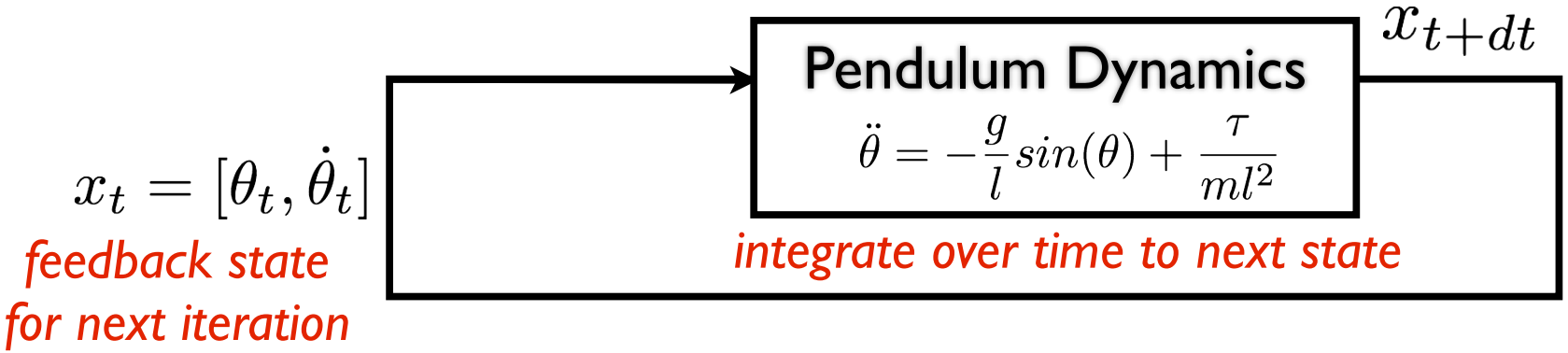
Integrators: Euler, Verlet, Velocity Verlet, RK4

Putting it all together (Pendularm)

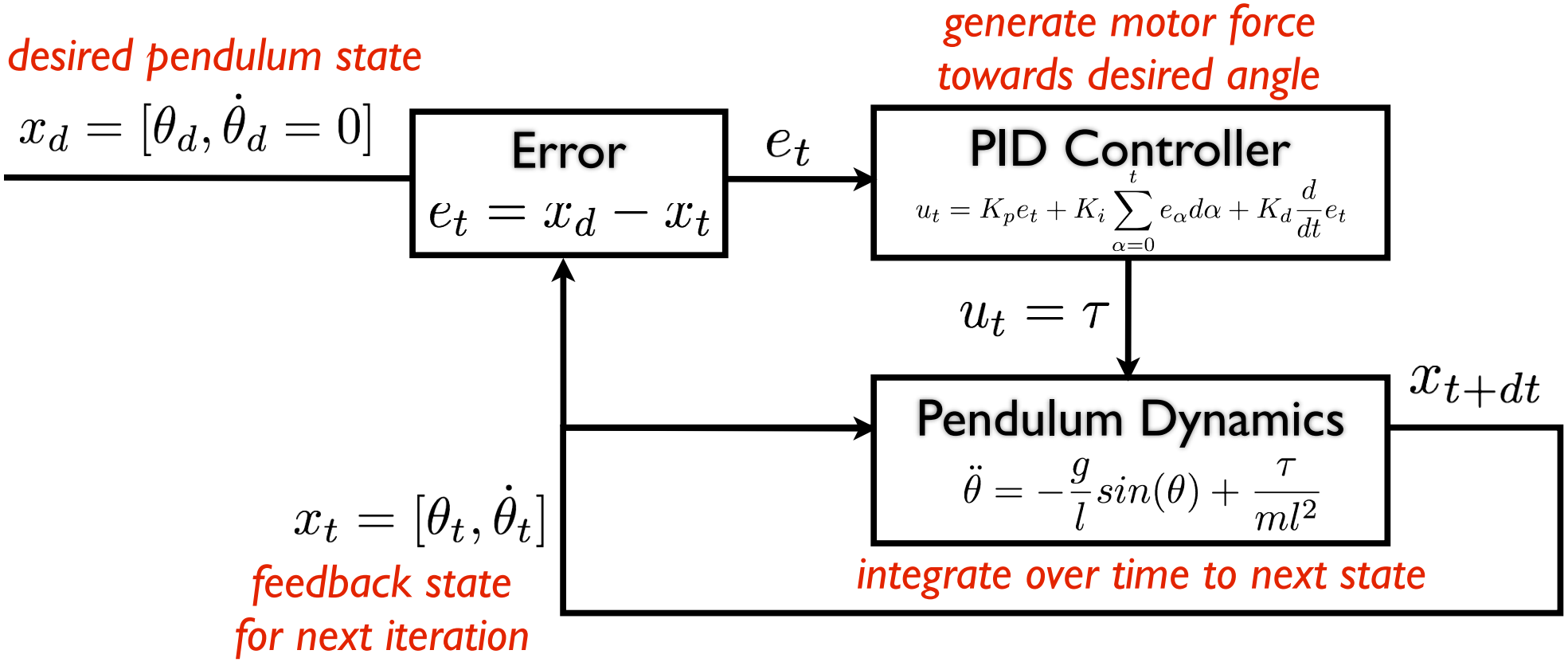
THIS LECTURE

Feedback Control

PID Control of 1 DoF robot



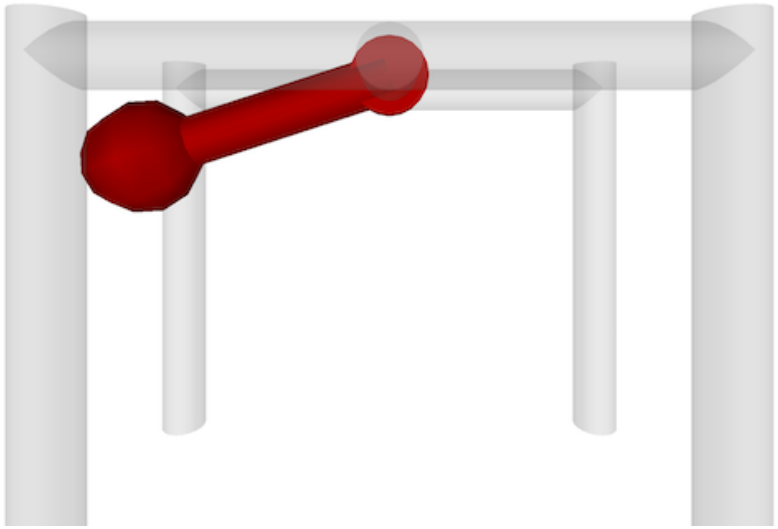
Putting it all together (Pendularm)



Warning:
Changing system -> changing gains

Let's see what happens

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



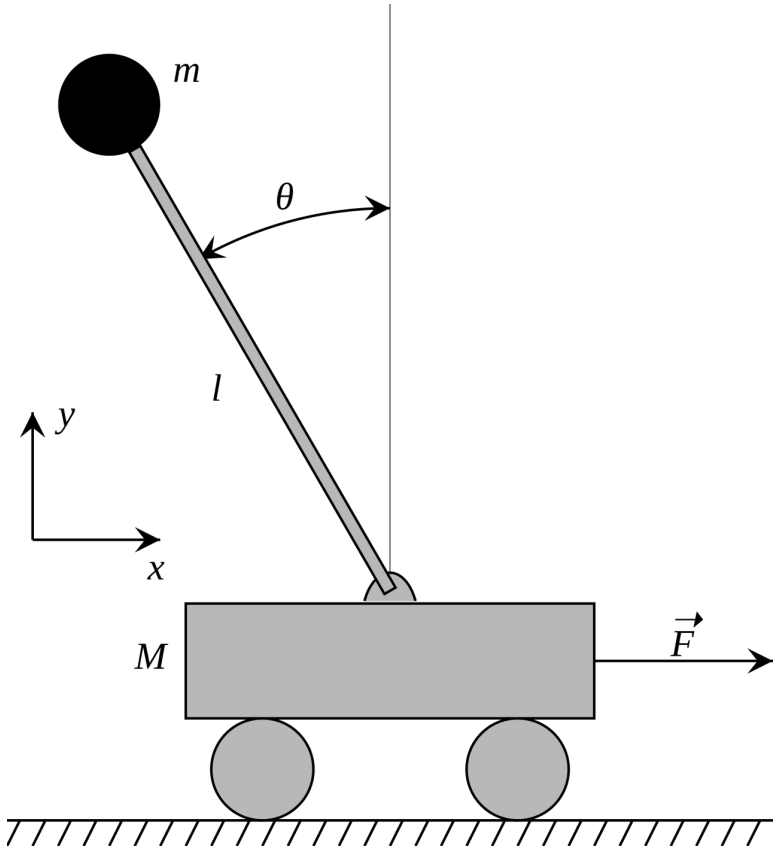
bigger
different

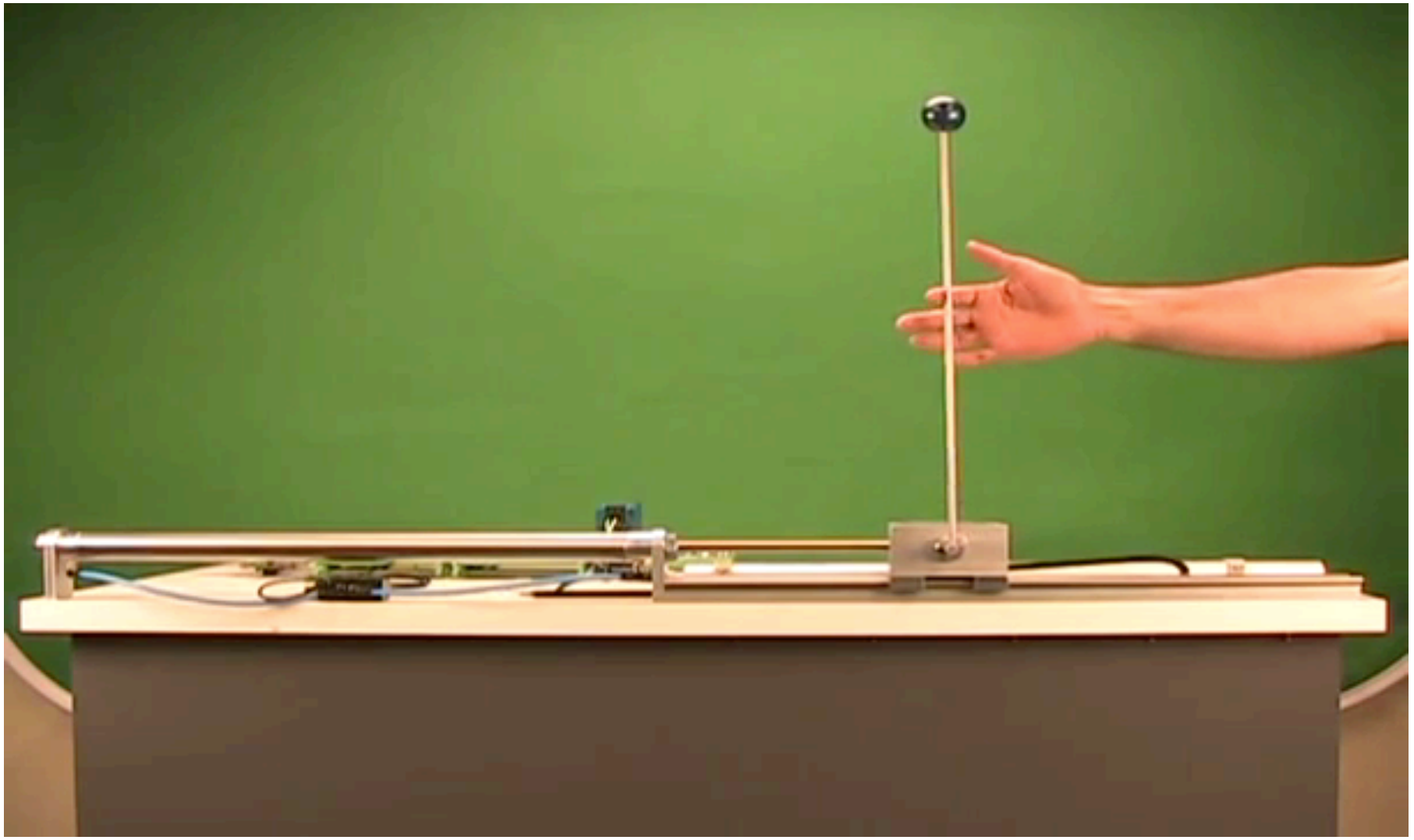
Other types of systems?

Cart Pole Balancer

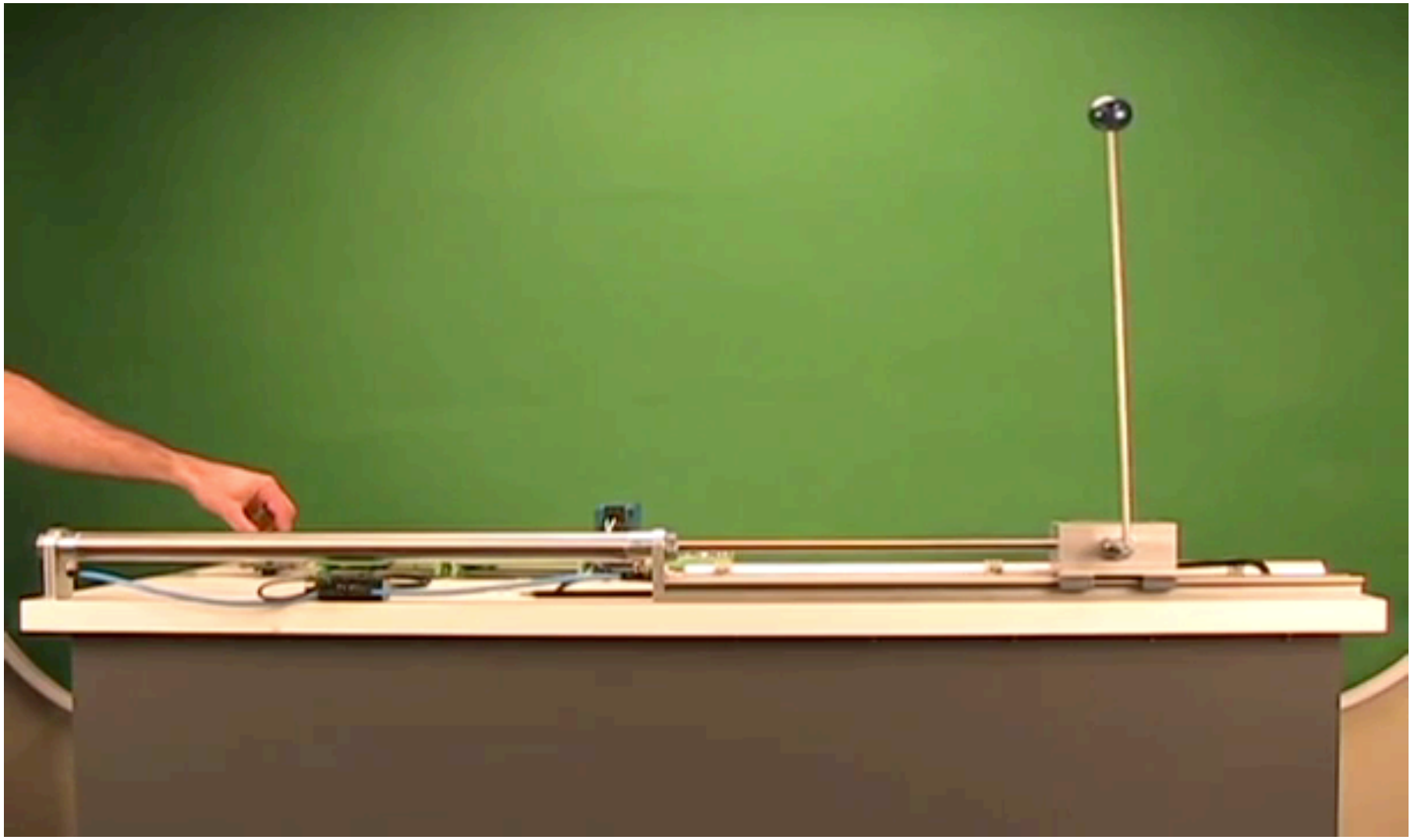
$$(M+m)\,\ddot{x} - m\ell\ddot{\theta}\cos\theta + m\ell\dot{\theta}^2\sin\theta = F$$

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

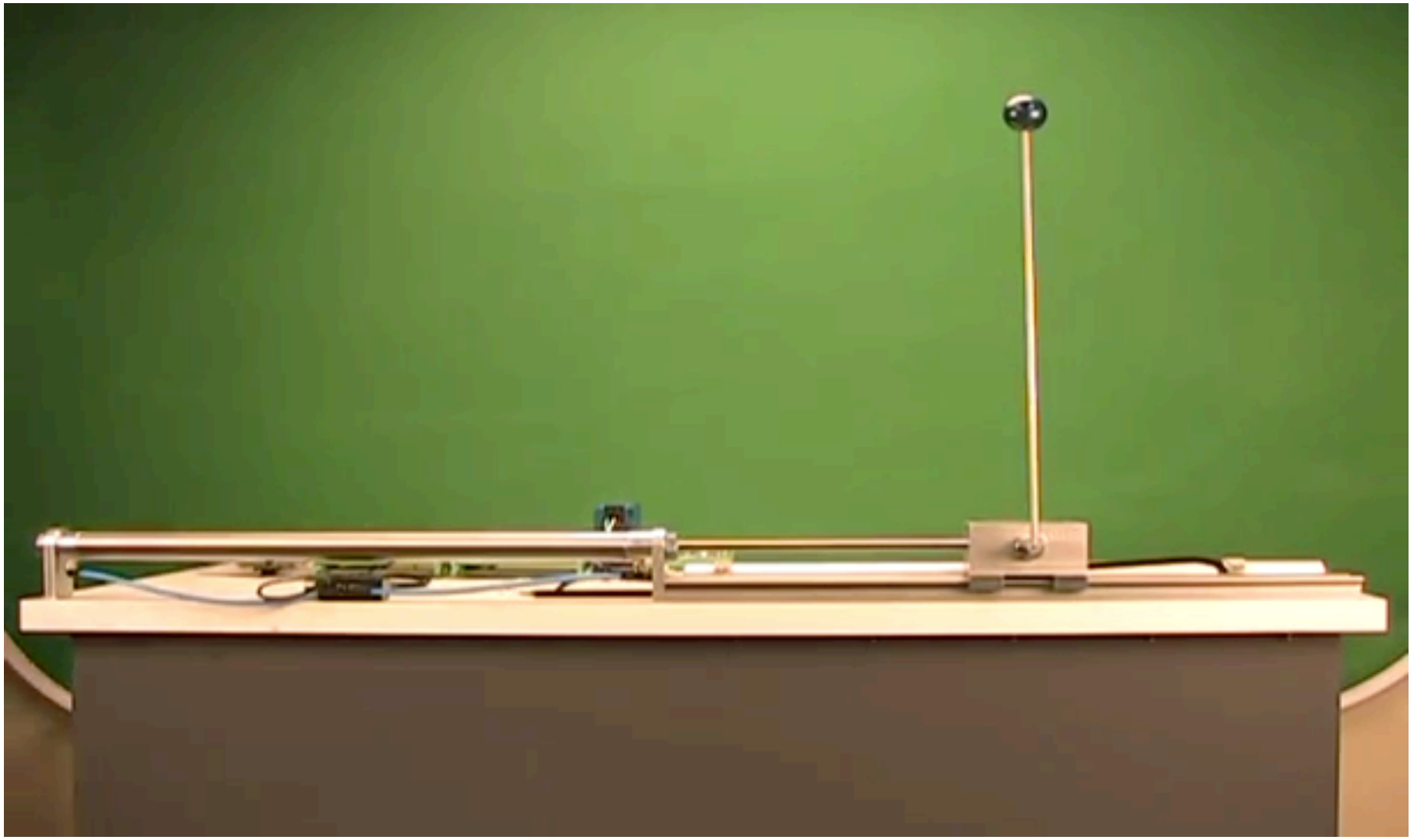




Enfield Technologies



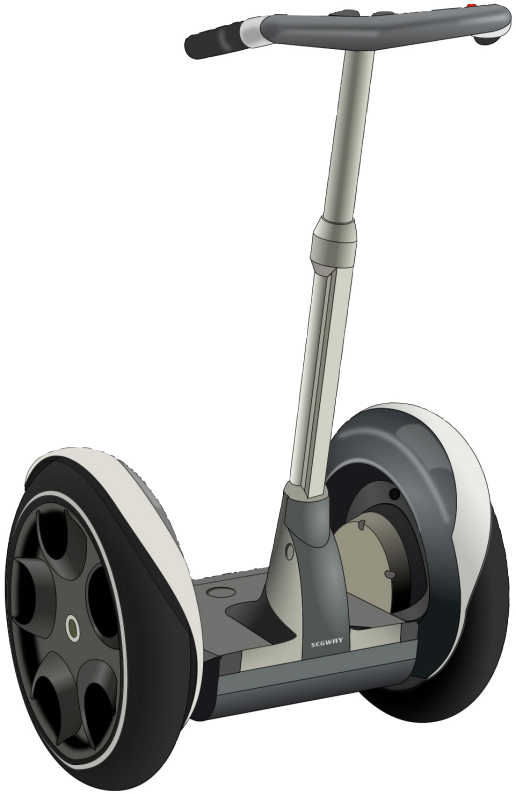
https://www.youtube.com/watch?v=a4c7AwHFkT8 Enfield Technologies



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

Enfield Technologies

Differential Drive Inverted Pendulum



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

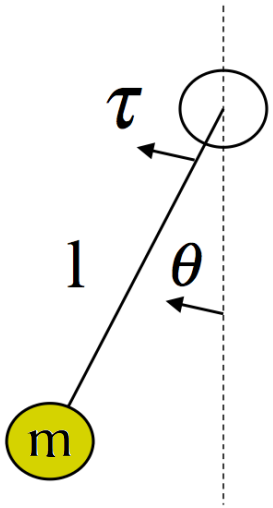
techbricks.nl

Robot links are rigid bodies

Robot links are rigid bodies

Pendulum as a particle

Mass assumed to be at a single point

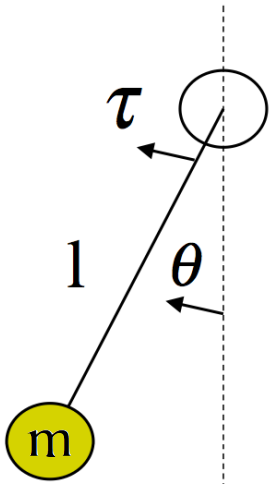


Only position of particle considered

Robot links are rigid bodies

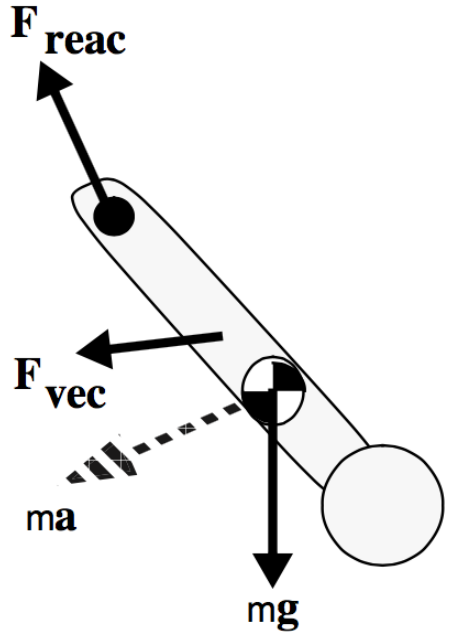
Pendulum as a particle

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 dynamics for translation and rotation of a 3D rigid body
- Newton's equation: $F = m\ddot{x}$
  - expresses the force acting at the body's center of mass (COM) 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 vectors for angular velocity (ω) and acceleration ($\dot{\omega}$), and 3-by-3 inertia matrix wrt. COM

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

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

Linear algebra refresher coming soon!

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

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

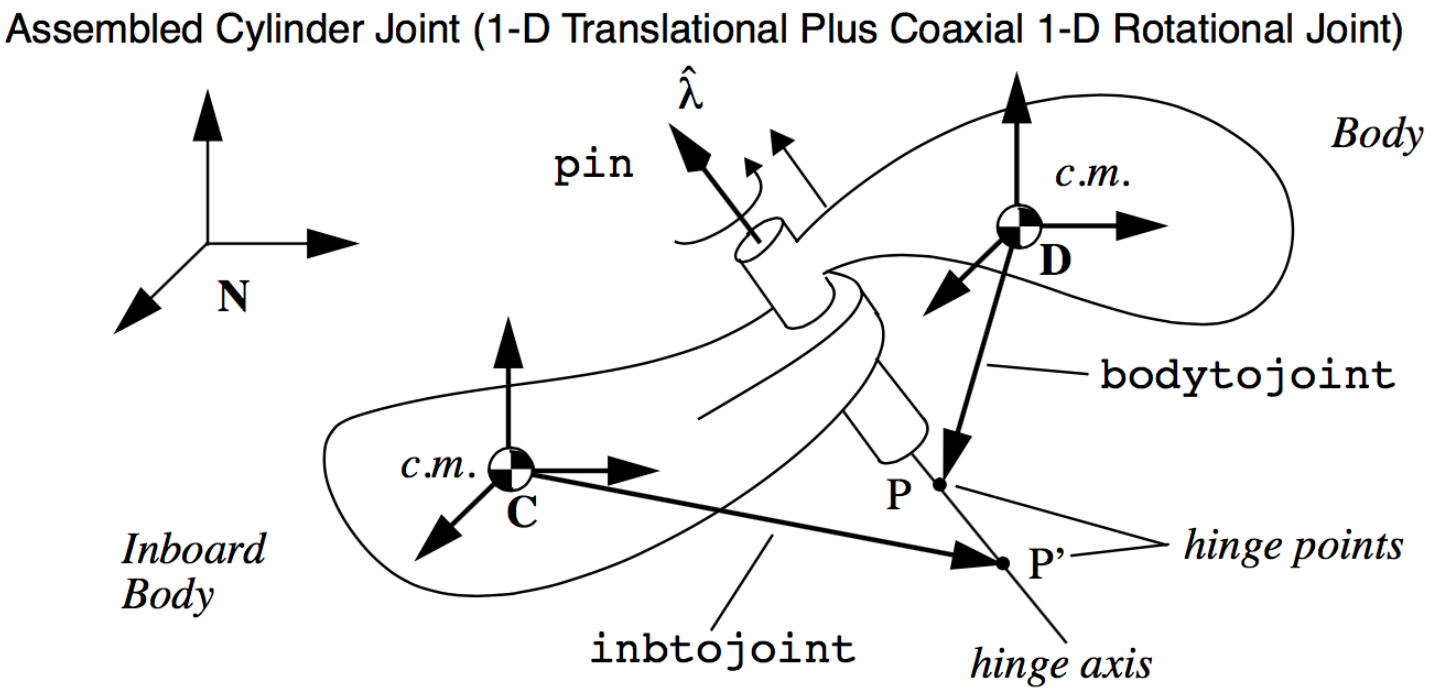
Cross product by skew-symmetric matrix

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

  - and with respect to an arbitrary frame of reference

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

Joints between rigid bodies



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) from last lecture
- **Newton-Euler recursive** forward-backward algorithm

Just a few options

  - 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

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.

Step 0: start from base link

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.

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

Step 1: 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.

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

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

Step 2: 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

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

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.

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

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

Step 3: compute acceleration at end of link

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

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.

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.

Step 4: compute acceleration at center of mass

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

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

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

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

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.

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

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

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

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

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.

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

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

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.

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

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

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

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

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.

Step 6: start from endeffector link

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.

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

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

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

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

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.

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

Step 7: compute force exerted by link$_{i-1}$ on link$_i$

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.

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

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

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

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

2. Start with the terminal conditions

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

and use (9.148) and (9.149) to compute f_i and τ_i for i decreasing fro

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

Step 8: compute torque exerted by link$_{i-1}$ on link$_i$

$$\tau_i = R^{i+1}_i \tau_{i+1} - f_i \times r_{i,ci} + (R^{i+1}_i f_{i+1}) \times r_{i+1,ci} + \alpha_i + \omega_i \times (I_i \omega_i)$$

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.

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

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

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

$$(\omega_i \times r_{i,i+1})$$

2. Start wit

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

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

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

$$\tau_i = R^{i+1}_i \tau_{i+1} - f_i \times r_{i,ci} + (R^{i+1}_i f_{i+1}) \times r_{i+1,ci} + \alpha_i + \omega_i \times (I_i \omega_i)$$

Final note:
Tangent into controls

PD as a Linear System

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

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

$[x_1\ x_2]^T$ 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;

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

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

Stability Analysis

Within a state space representation, dynamics expressed through:

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

SYSTEM MATRIX

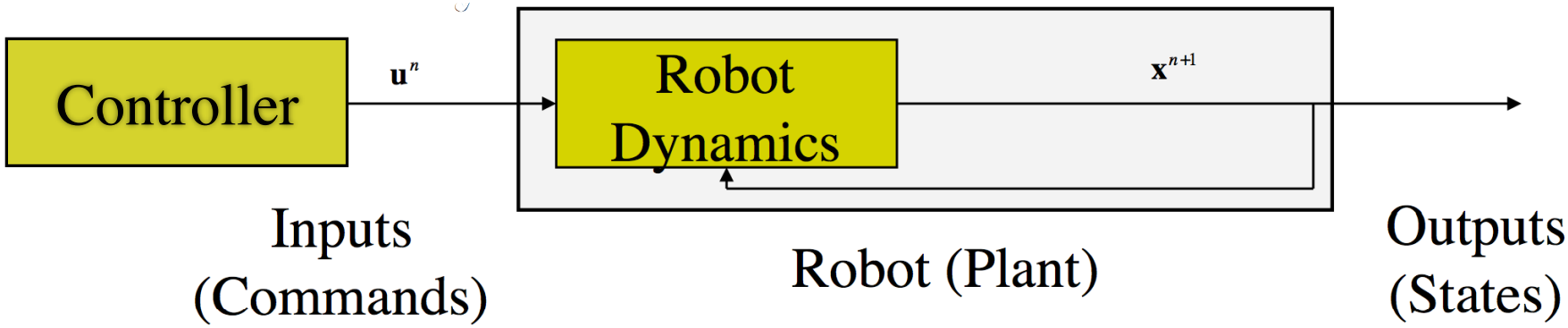
CONTROL MATRIX

Such representations enable analysis of the stability of the system, such as through the eigenvalues of the system matrix

https://en.wikibooks.org/wiki/Control_Systems/Eigenvalues_and_Eigenvectors

PID does not consider system dynamics and assumes independent joint control

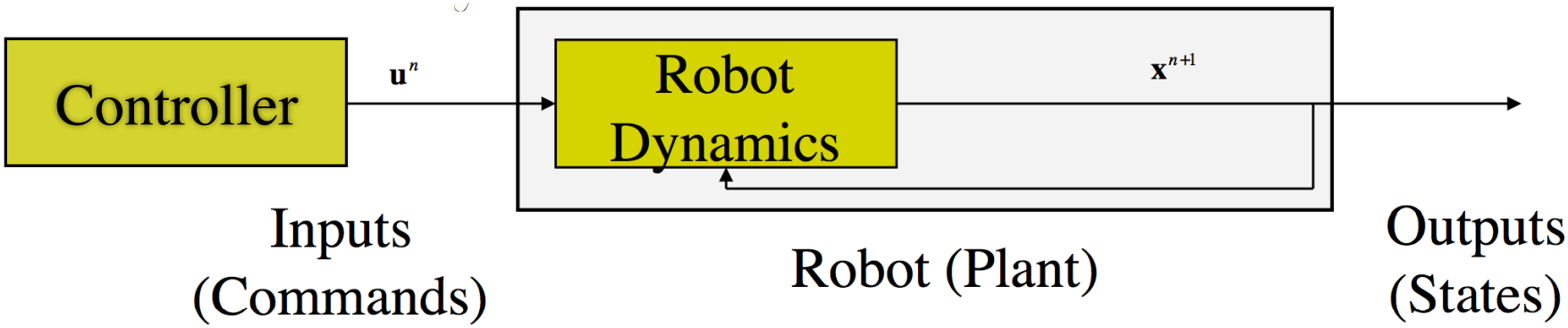
$$u = \pi(t, x, x_d) \qquad \dot{x} = f(t, x, u)$$



How can dynamics be accounted for in control?

forward model

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

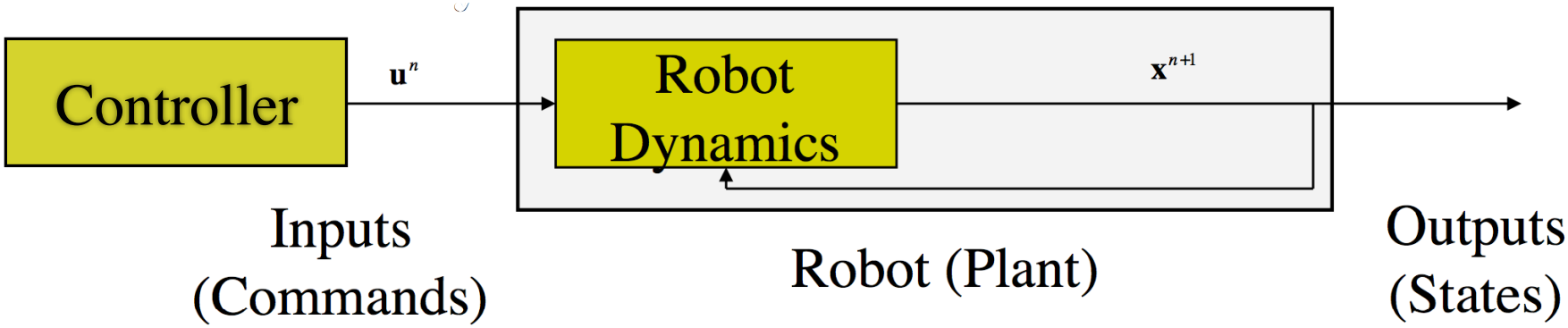


inverse model, if $\dot{x} = f(t, x, \pi(t, x, x_d))$

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

forward model

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



cost to desired state

Optimal Control

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

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

Forms of optimal control

Model predictive control

From Wikipedia, the free encyclopedia

Model predictive control (**MPC**) is an advanced method of process control that is used to control a process while satisfying a set of constraints. It has been in use in the process industries in chemical plants and oil refineries since the 1980s. In recent years it has also been used in power system balancing models[1] and in power electronics[2]. Model predictive controllers rely on dynamic models of the process, most often linear empirical models obtained by system identification. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot and then optimizing again, repeatedly, thus differing from Linear-Quadratic Regulator (LQR). Also MPC has the ability to anticipate future events and can take control actions accordingly. PID controllers do not have this predictive ability. MPC is nearly universally implemented as a digital control, although there is research into achieving faster response times with specially designed analog circuitry.[3]

Generalized

1 Overvie
1.1 T
1.2 P
2 Nonlinea
3 Explicit
4 Robust
5 Comme
6 MPC vs

Principles of MPC [edit]

Model Predictive Control (MPC) is a multivariable control algorithm that uses:

- an internal dynamic model of the process
- a cost function *J* over the receding horizon
- an optimization algorithm minimizing the cost function *J* using the control input *u*

An example of a quadratic cost function for optimization is given by:

$$J = \sum_{i=1}^{N} w_{x_i} (r_i - x_i)^2 + \sum_{i=1}^{N} w_{u_i} \Delta u_i{}^2$$

without violating constraints (low/high limits) with

x_i: i^{th} controlled variable (e.g. measured temperature)
r_i: i^{th} reference variable (e.g. required temperature)
u_i: i^{th} manipulated variable (e.g. control valve)
w_{x_i}: weighting coefficient reflecting the relative importance of x_i
w_{u_i}: weighting coefficient penalizing relative big changes in u_i

etc.

Linear–quadratic regulator

From Wikipedia, the free encyclopedia
(Redirected from Linear-quadratic regulator)

The theory of optimal control is concerned with operating a dynamic system at minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic function is called the LQ problem. One of the main results in the theory is that the solution is provided by the **linear–quadratic regulator** (**LQR**), a feedback controller whose equations are given below. The LQR is an important part of the solution to the LQG (linear–quadratic–Gaussian) problem. Like the LQR problem itself, the LQG problem is one of the most fundamental problems in control theory.

Contents [hide]
1 General description
2 F
3 In
4 F
5 In
6 R
7 E

Ge
The s
found
(engi

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 state 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

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

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., $\mathbf{A}, \mathbf{B}, \mathbf{Q}$ $\mathbf{R}$) 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*). T 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

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

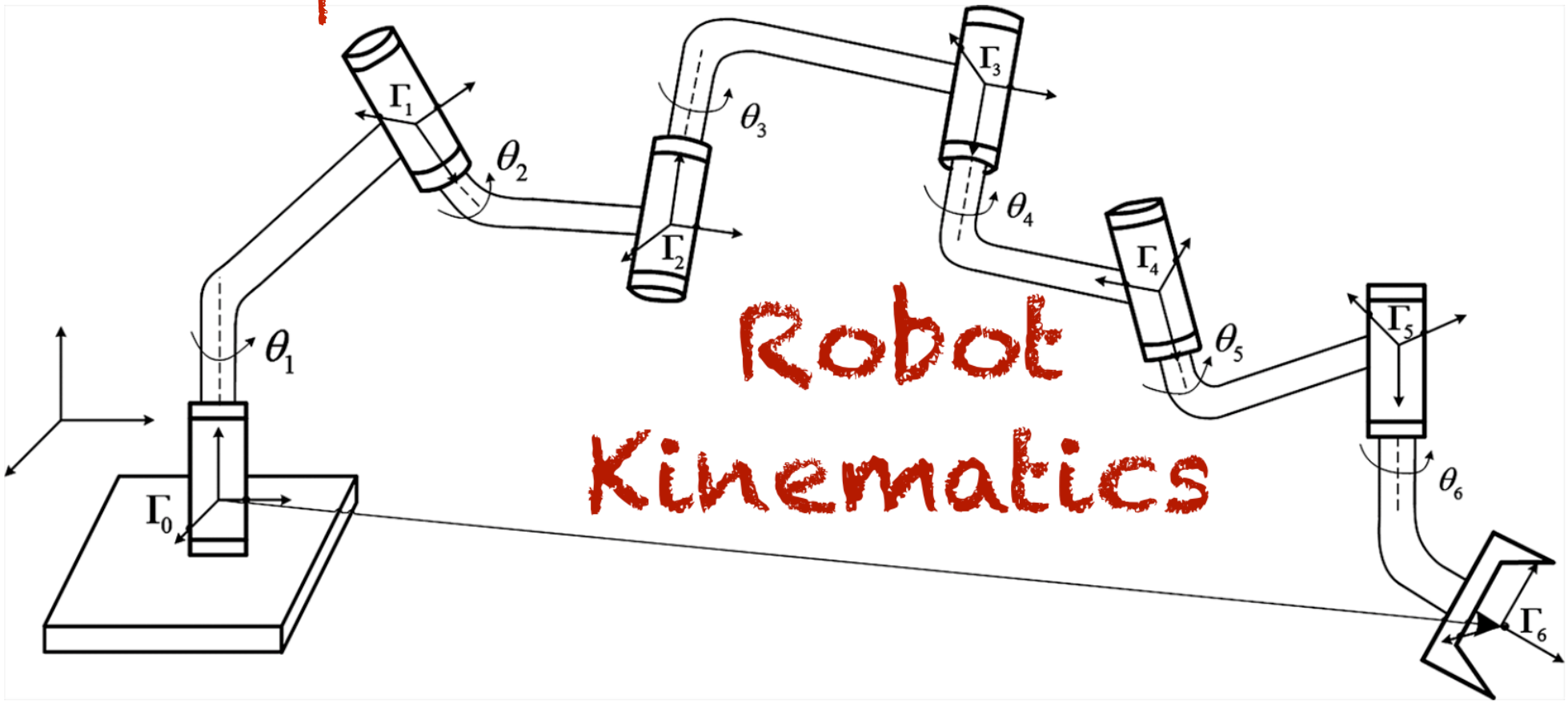
and the initial condition

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

Do not forget to review:

Coordinate Systems and Linear Algebra

Next topic:


Robot
Kinematics
Γ_0
Γ_1
Γ_2
Γ_3
Γ_4
Γ_5
Γ_6
θ_1
θ_2
θ_3
θ_4
θ_5
θ_6


invisible segway