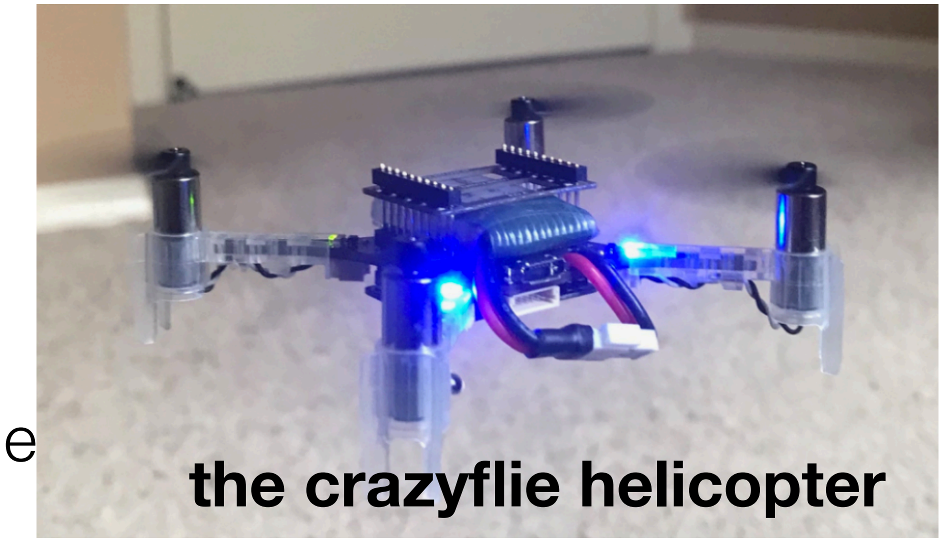
Flight control of hovering aircraft

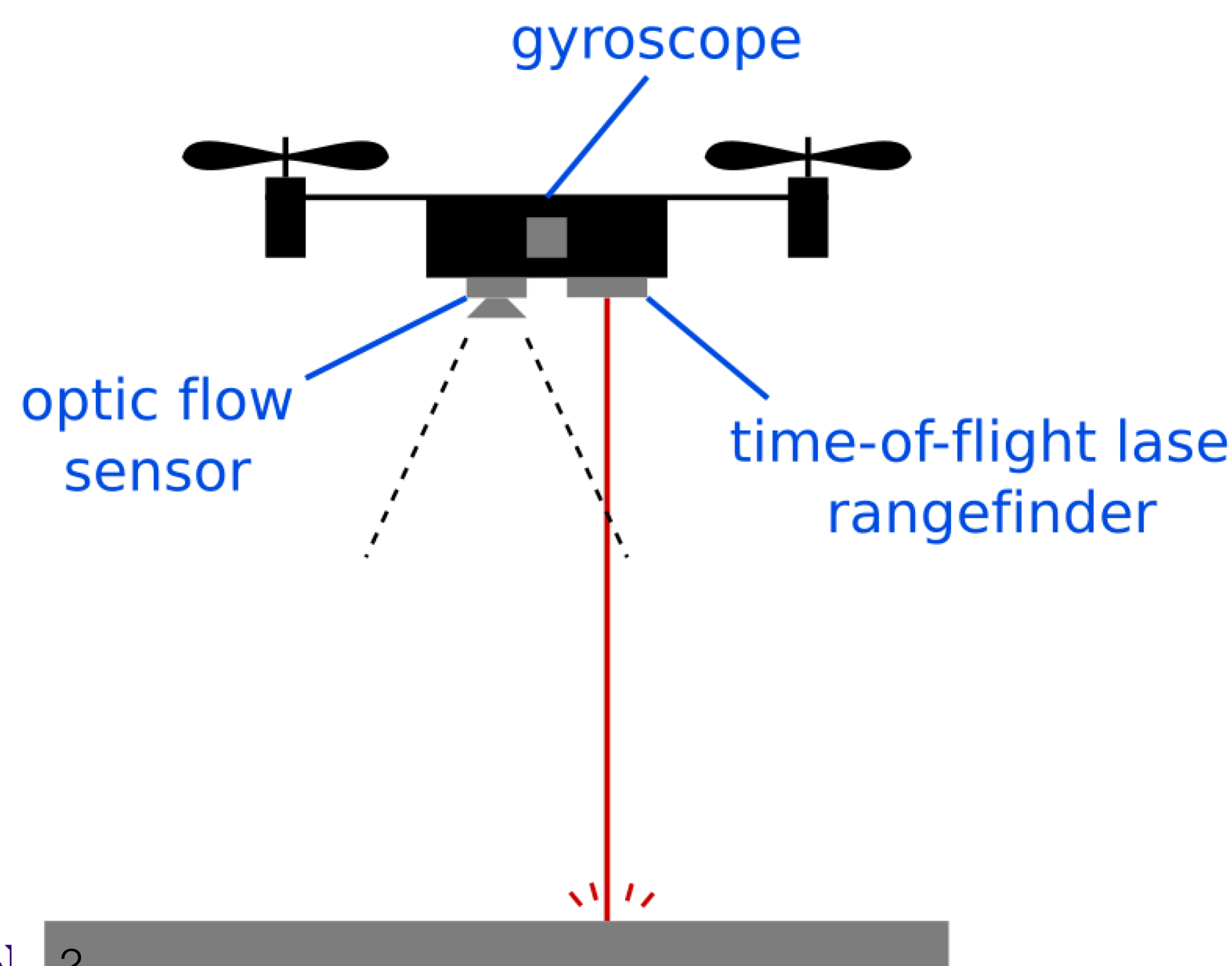
Prof. Sawyer B. Fuller

ME 586: Biology-inspired robotics

Project-based portion of this course

- you will work with the crazyflie helicopter as part of two homework problem sets
    - objectives: learn basics of robotics and drone control
- optionally, you may use this helicopter as part of your term project
- Crazyflie specs:
    - ~30 g, ~4 minute flight time
    - communicates in real-time over bluetooth to laptop
    - sensor suite gives information needed to stabilize and control flight
    - open-source control software

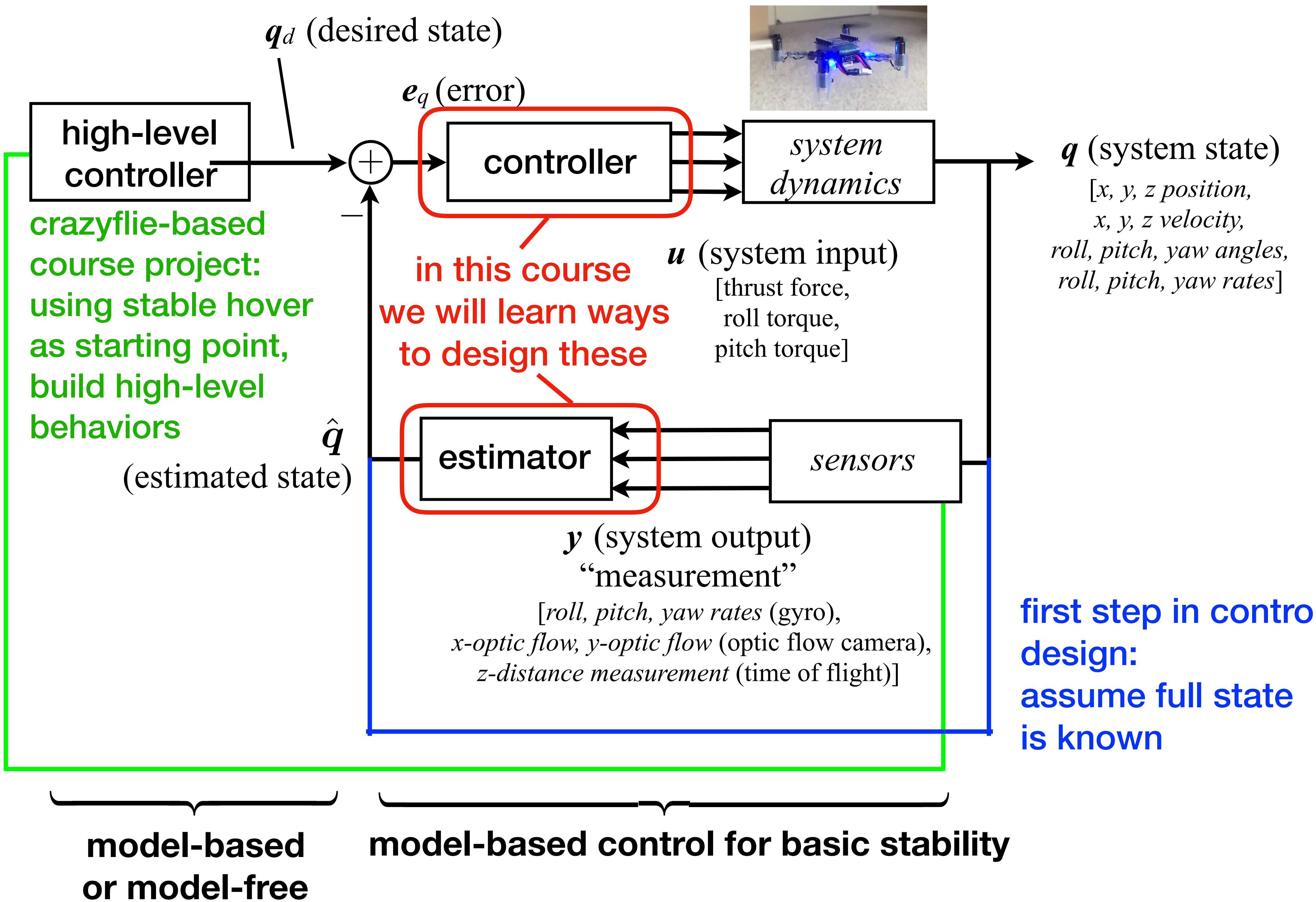




example crazyflie project: odor source localization

Odor
Localization

The control task we will cover



basics: actuation for hovering

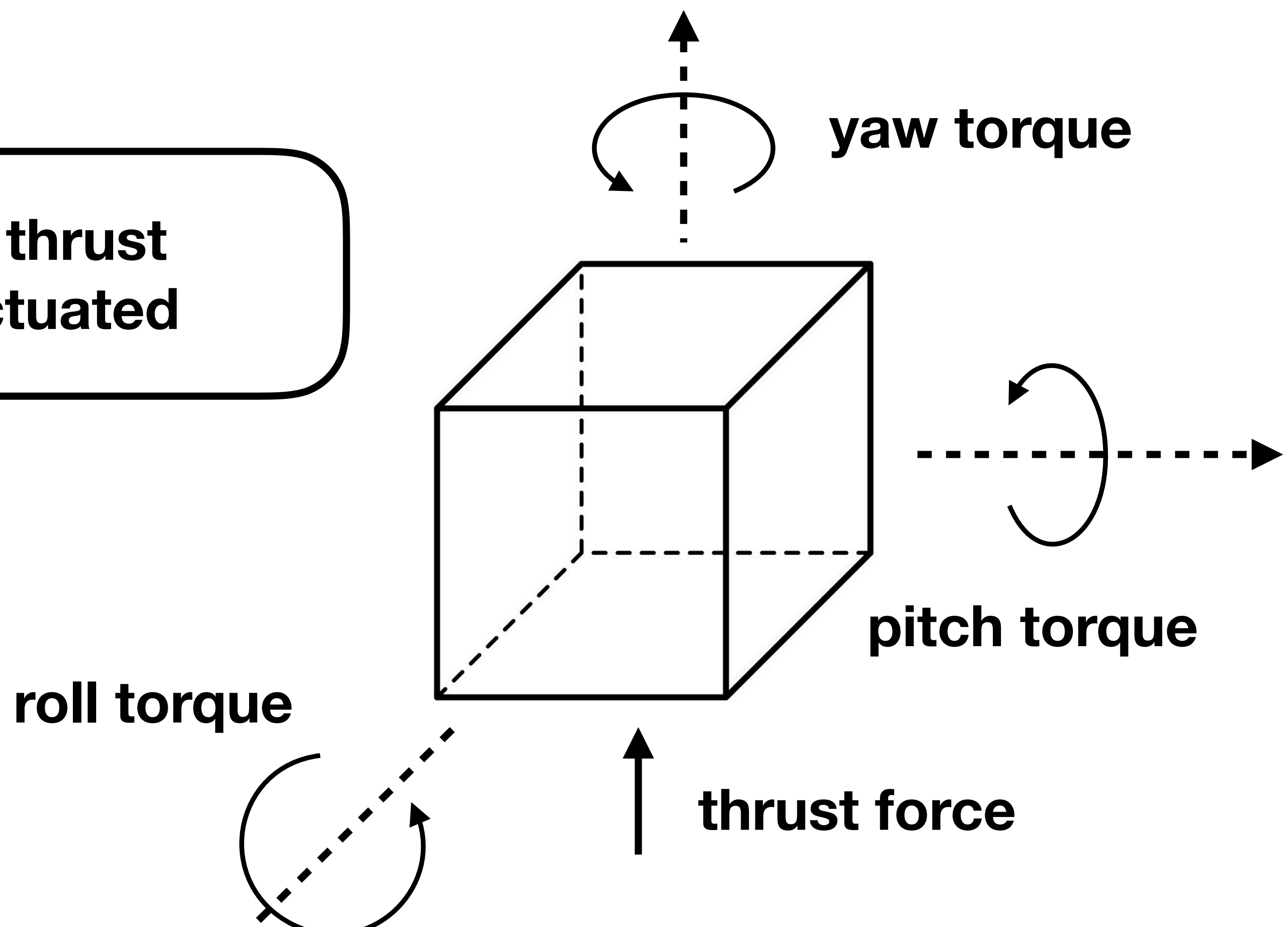
honeybee

single-rotor helicopter

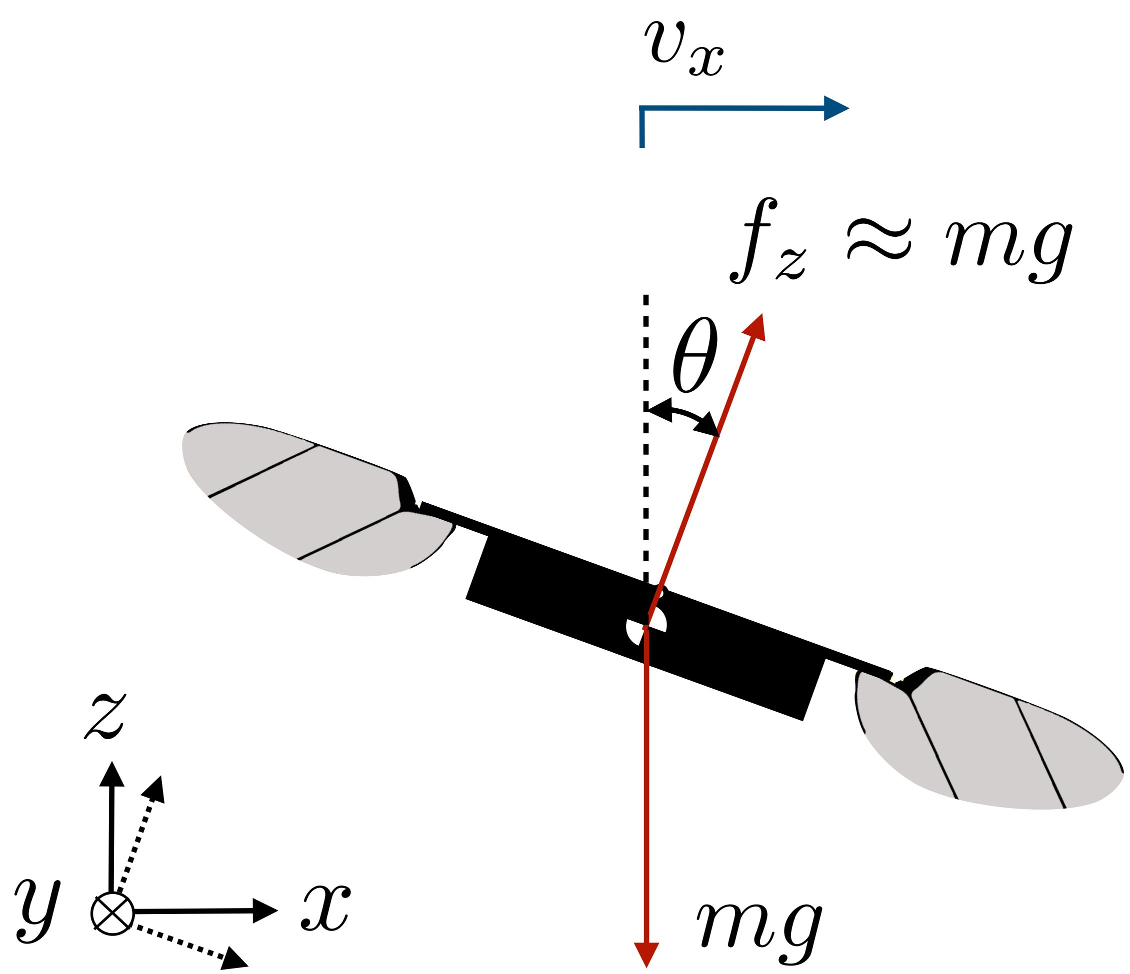
robot flies e.g. UW Robofly

four-rotor aircraft "quad-rotors"

typically, lateral thrust is not directly actuated



lateral actuation by tilting



lateral acceleration

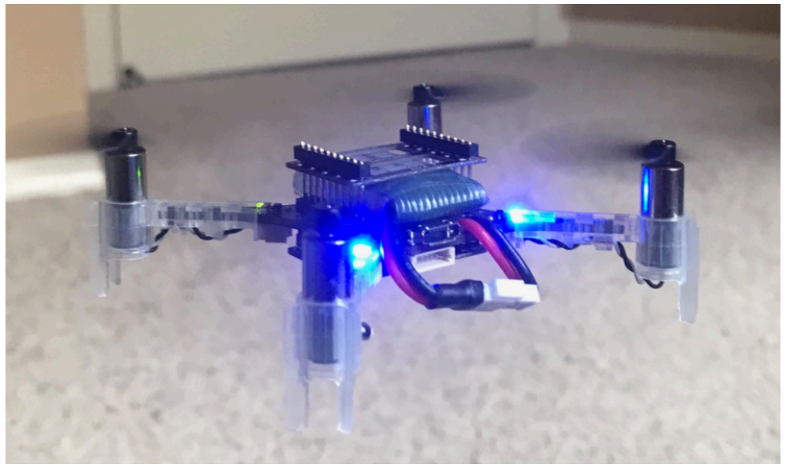
$$\dot{v}_x = \frac{1}{m} mg \sin\theta = g \sin\theta$$

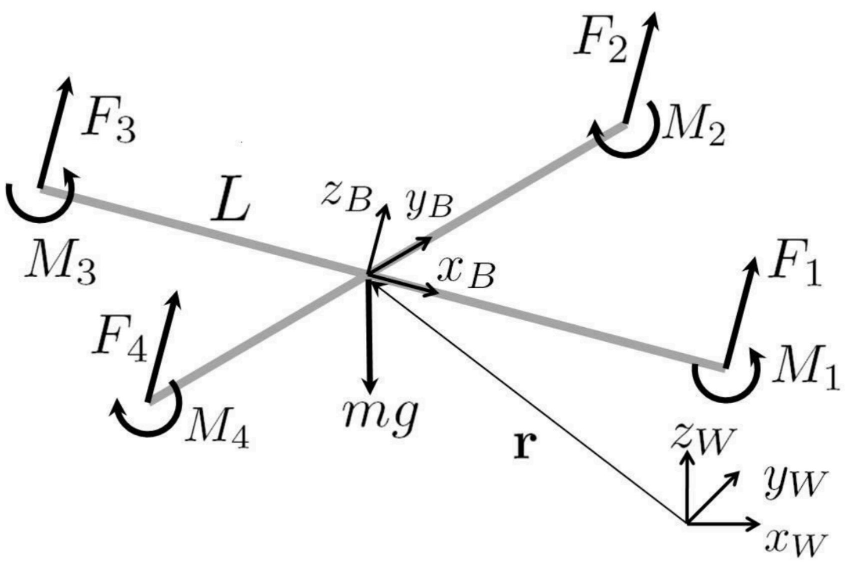
$$\approx g\theta \quad \textbf{for small } \theta$$

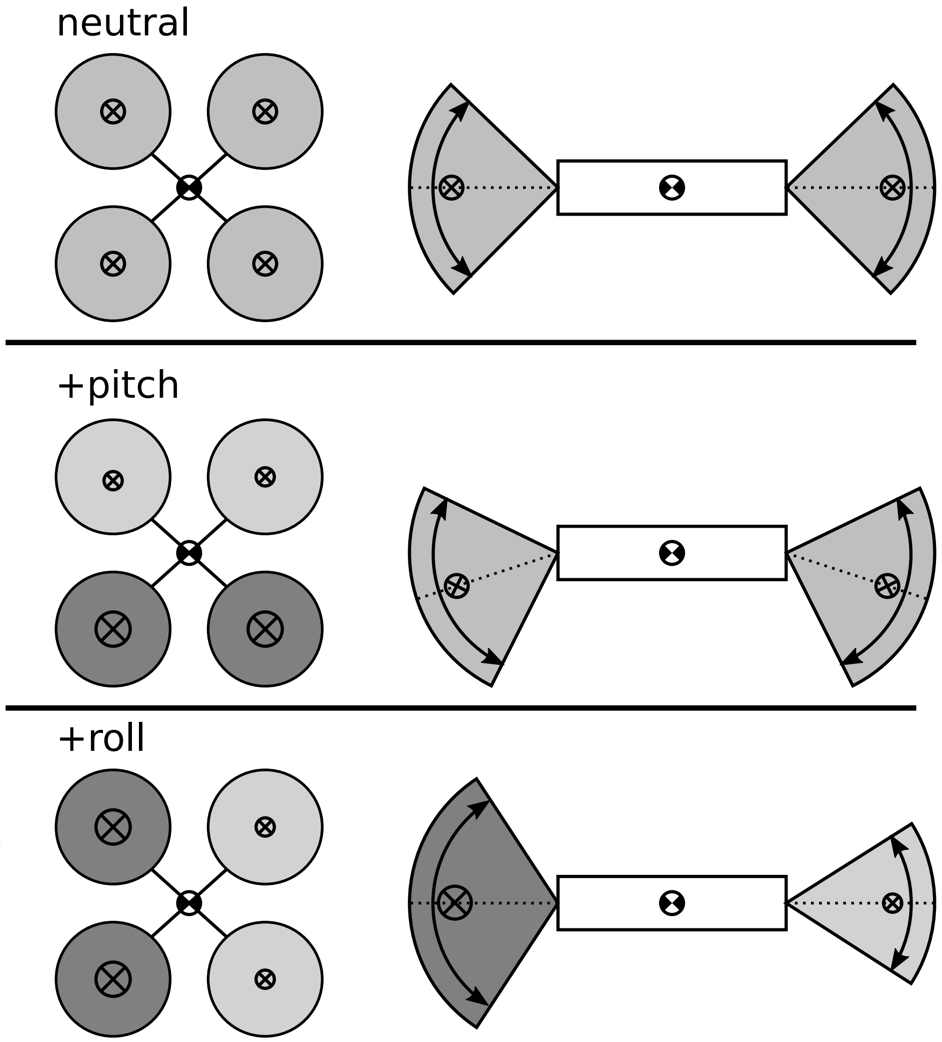
- “helicopter-like” lateral control

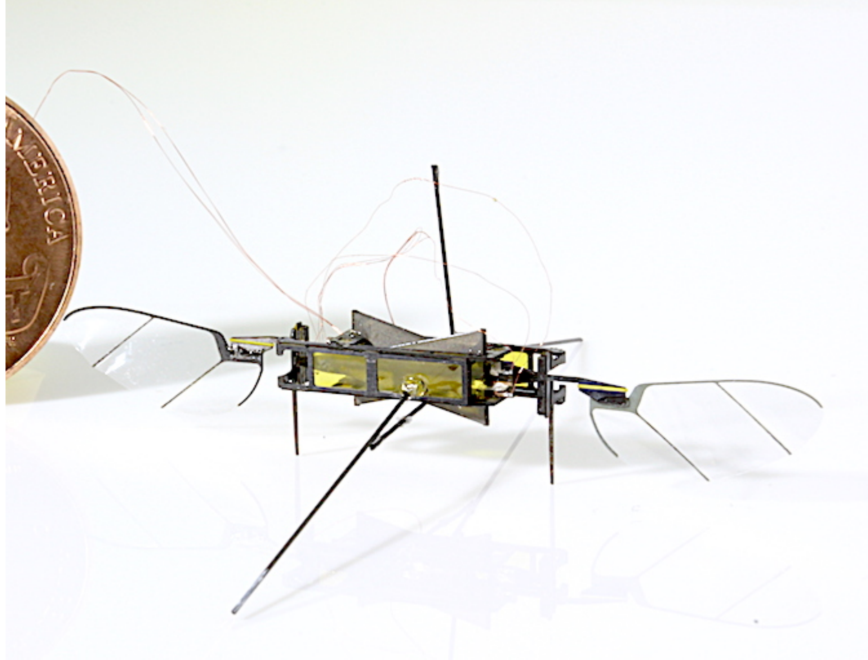
quad-rotor actuation

actuation with two wings





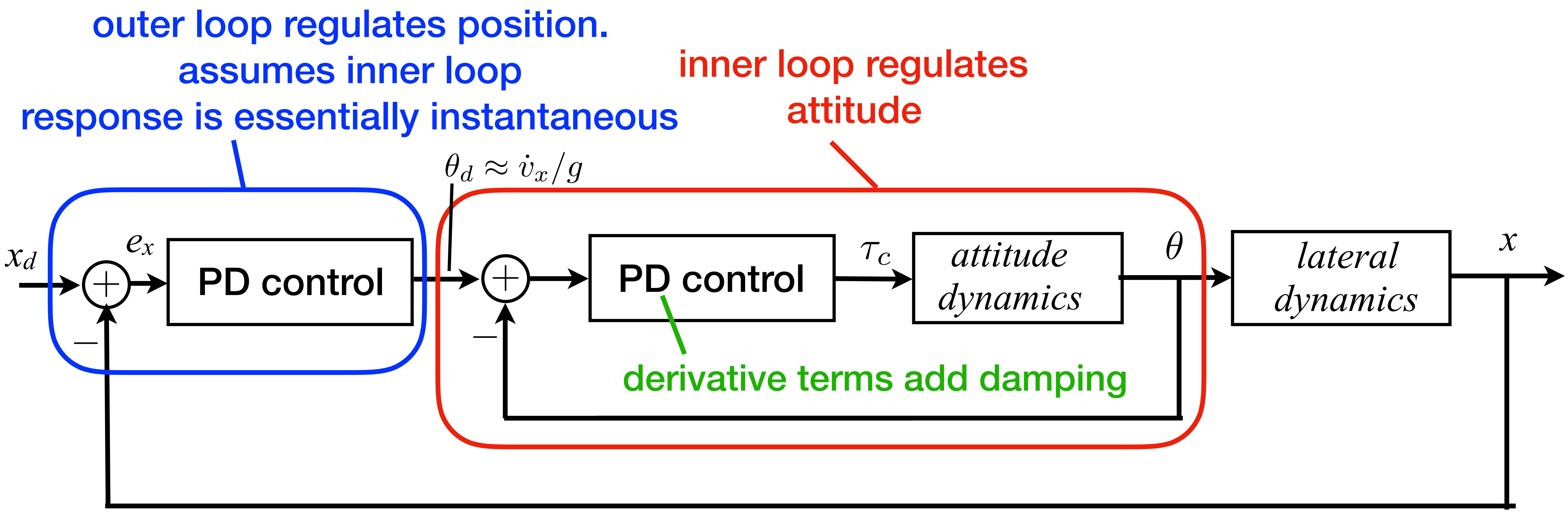




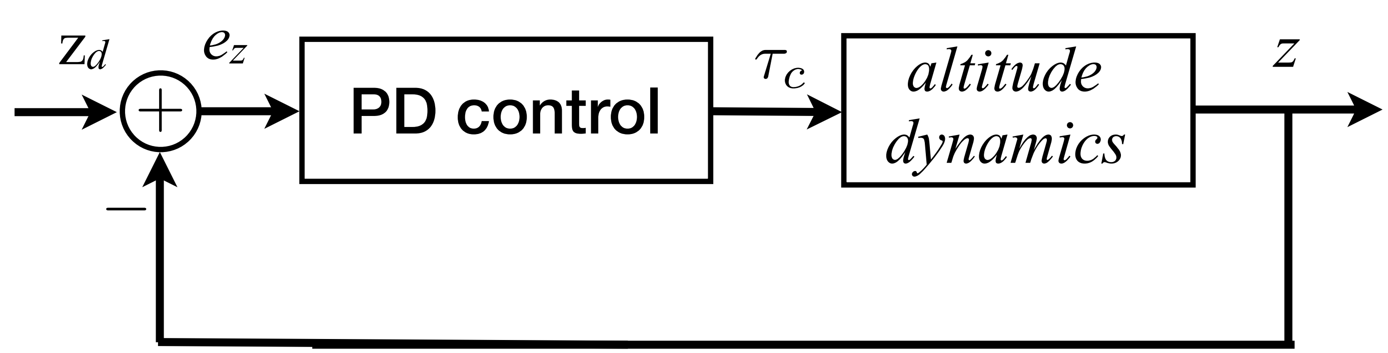
- two rotors spin one direction and two in the other direction

- vary angle and amplitude of flapping wings

insight into flight control: One approach is nested loops (problem set 2)



- plus a separate, independent altitude controller:



more systematic approach: start with Newton-Euler equations of Motion

$$\Sigma \boldsymbol{f} = m\dot{\boldsymbol{v}}$$

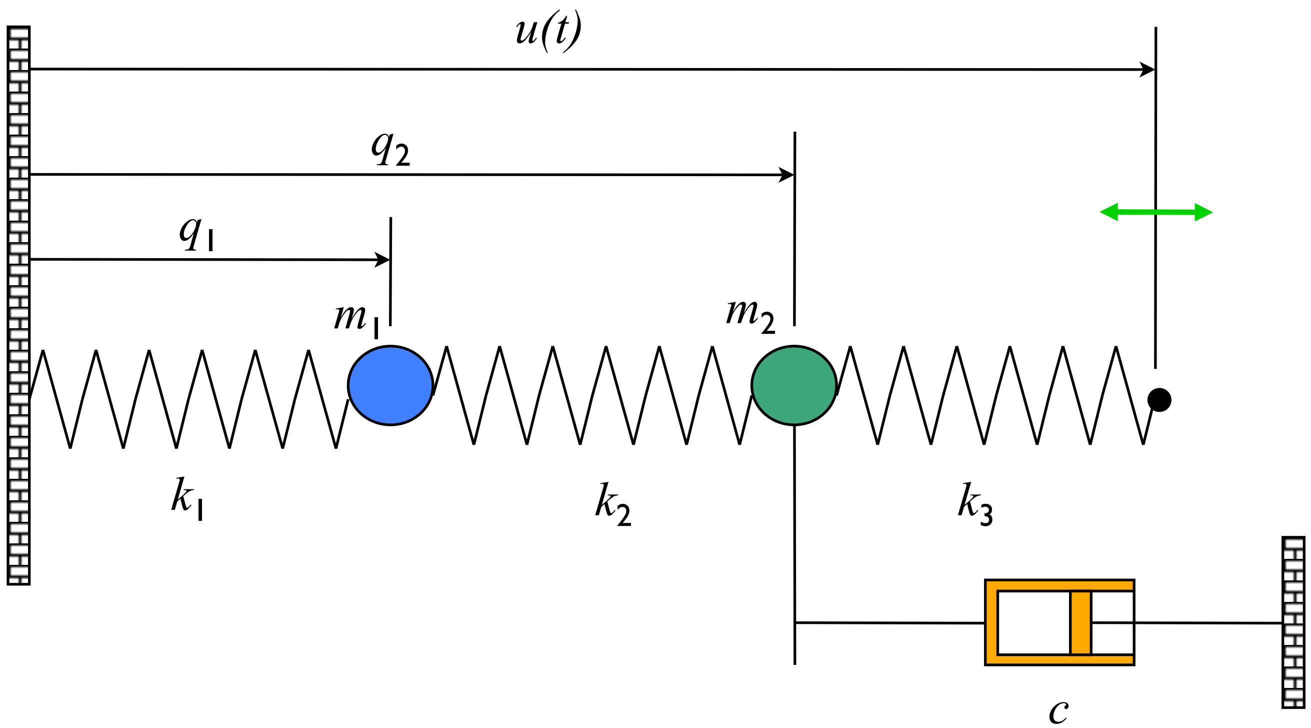
$$\Sigma \boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}$$

$\boldsymbol{f}, \boldsymbol{\tau}$	**force and torque**
$\boldsymbol{v}, \boldsymbol{\omega}$	**linear, angular velocity**
$\mathbf{J}$	**moment of inertia matrix**

- this is a *nonlinear system.*
- we will control it with *linear feedback controller*
- will return to this in more detail next week

Controlling nonlinear systems using linear state-space control

State-space model example: a Spring Mass System



Model: rigid body physics

- Sum of forces = mass * acceleration
- Hooke's law: $F = k(x - x_{\text{rest}})$
- Viscous friction: $F = c\,v$

$$\boxed{\begin{aligned} m_1\ddot{q}_1 &= k_2(q_2 - q_1) - k_1 q_1 \\ m_2\ddot{q}_2 &= k_3(u - q_2) - k_2(q_2 - q_1) - c\dot{q}_2 \end{aligned}}$$

Converting models to state space form

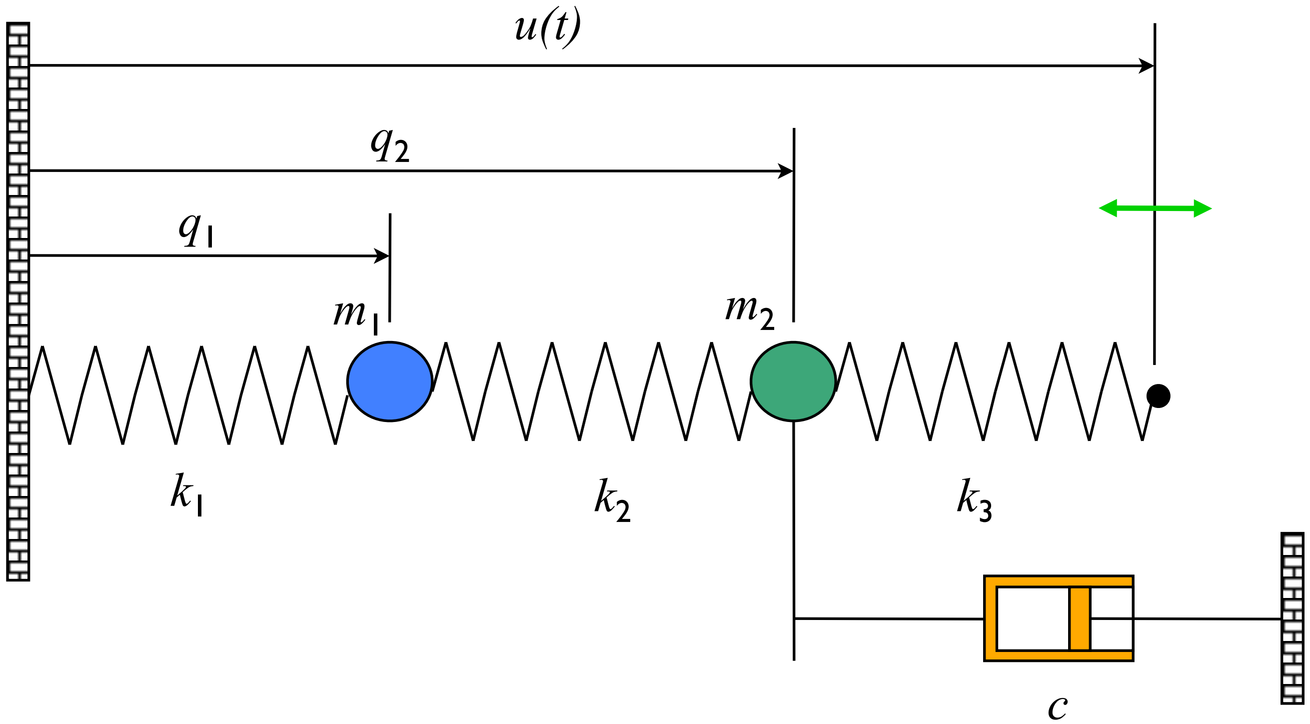
- Construct a *vector* of the variables that are required to specify the evolution of the system
- Write dynamics as a *system* of first order differential equations:

$$\frac{d}{dt}\begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \frac{k_2}{m}(q_2 - q_1) - \frac{k_1}{m} q_1 \\ \frac{k_3}{m}(u - q_2) - \frac{k_2}{m}(q_2 - q_1) - \frac{c}{m}\dot{q} \end{bmatrix}$$

$$y = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

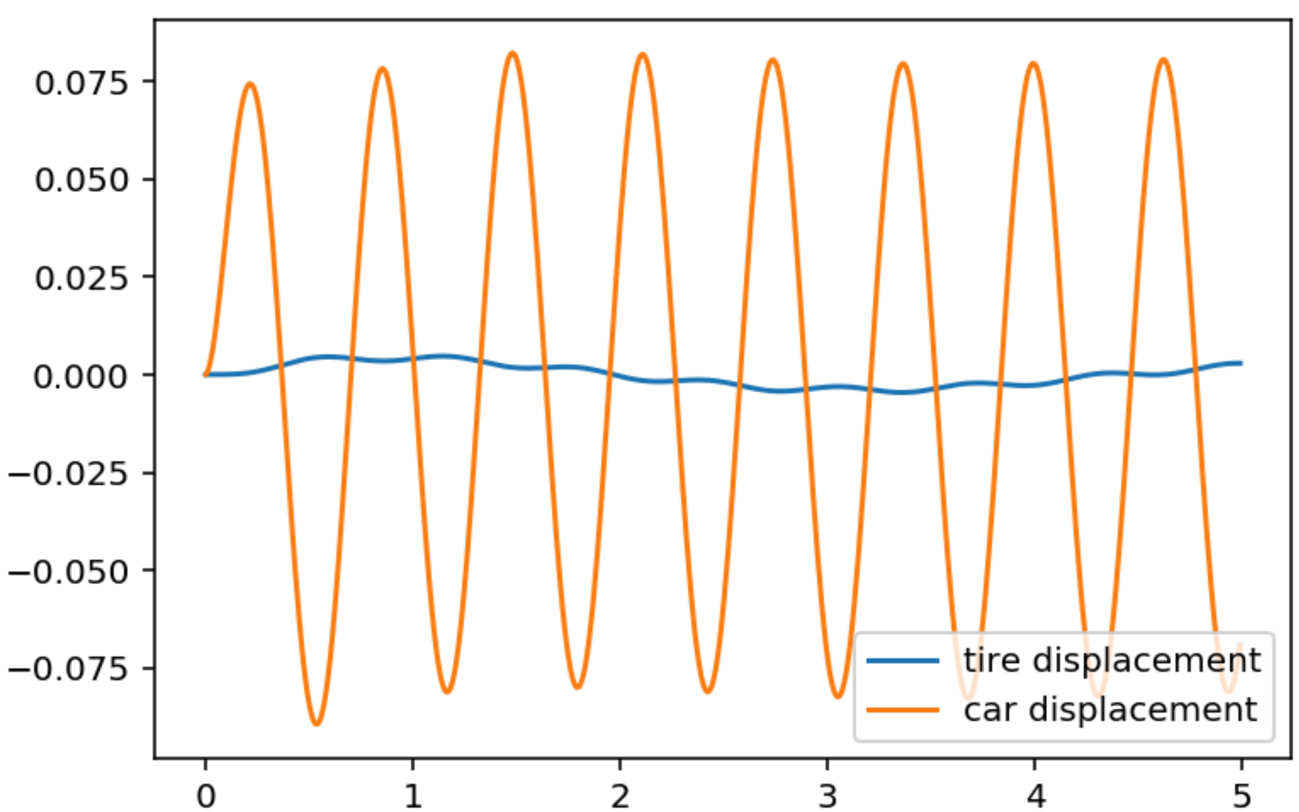
"State space form"

Simulating a state-space system



basic task: repeatedly calculate state update:

$$\boldsymbol{q}_{t+\Delta T} = \boldsymbol{q}_t + \Delta T \dot{\boldsymbol{q}}_t$$



Python simulation

```
import numpy as np
import matplotlib.pyplot as plt

k1 = k2 = k3 = m1 = c = 1
m2 = 0.1
dt = 0.01

time = np.arange(0, 5, dt)
y_data = np.zeros((len(time), 4))
y = np.array((0, 0, 0, 0))
```

initial condition

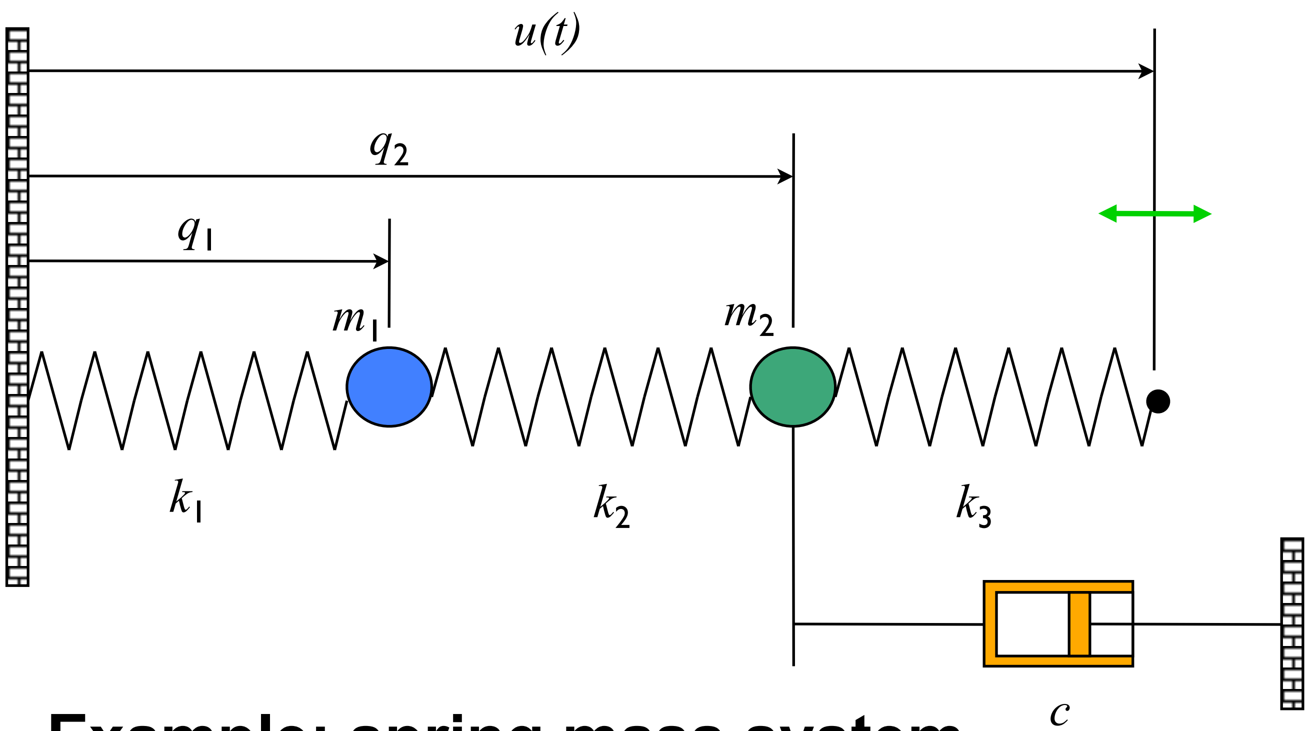
```
def dydt(y, u):
    return np.array((
        y[2],
        y[3],
        -(k1+k2)/m1*y[0] + k2/m1*y[1],
        k2/m2*y[0] - (k2+k3)/
            m2*y[1] - c/m2*y[3] + k3/m2*u))
```

dynamics function “f”

```
for idx, t in enumerate(time):
    u = np.cos(10*t)
    y = y + dt * dydt(y, u)
    y_data[idx,:] = y
plt.plot(time, y_data[:,0:2])
plt.legend(('tire displacement',
            'car displacement'))
```

update step

Modeling Terminology



***State* captures effects of the past**

- independent physical quantities that determines future evolution (absent external excitation)

***Inputs* describe external excitation**

- Inputs are *extrinsic* to the system dynamics (externally specified)

***Dynamics* describes state evolution**

- update rule for system state
- function of current state and any external inputs

***Outputs* describe measured quantities**

- Outputs are function of state and inputs ⇒ not independent variables
- Outputs are often *subset* of state

Example: spring mass system

- State: position and velocities of each mass: $q_1, q_2, \dot{q}_1, \dot{q}_2$
- Input: position of spring at right end of chain: $u(t)$
- Dynamics: basic mechanics
- Output: measured positions of the masses: q_1, q_2

Example: quad-rotor aircraft

- State: position and velocity of CM
- Input: speeds of the four motors
- Dynamics: Newton-Euler equations

general form of differential equations

State space form

$$\frac{dx}{dt} = f(x,u) \qquad y = h(x,u)$$

General form

$$\frac{dx}{dt} = Ax + Bu \qquad y = Cx + Du$$

Linear system

$$x \in \mathbb{R}^n, u \in \mathbb{R}^p \qquad y \in \mathbb{R}^q$$

- x = state; nth order
- u = input; will usually set p = 1
- y = output; will usually set q = 1

Higher order, linear ODE

$$\frac{d^n q}{dt^n} + a_1 \frac{d^{n-1} q}{dt^{n-1}} + \cdots + a_n q = u$$

$$y = b_1 \frac{d^{n-1} q}{dt^{n-1}} + \cdots + b_{n-1}\dot{q} + b_n q$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d^{n-1}q/dt^{n-1} \\ d^{n-2}q/dt^{n-2} \\ \vdots \\ dq/dt \\ q \end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} x$$

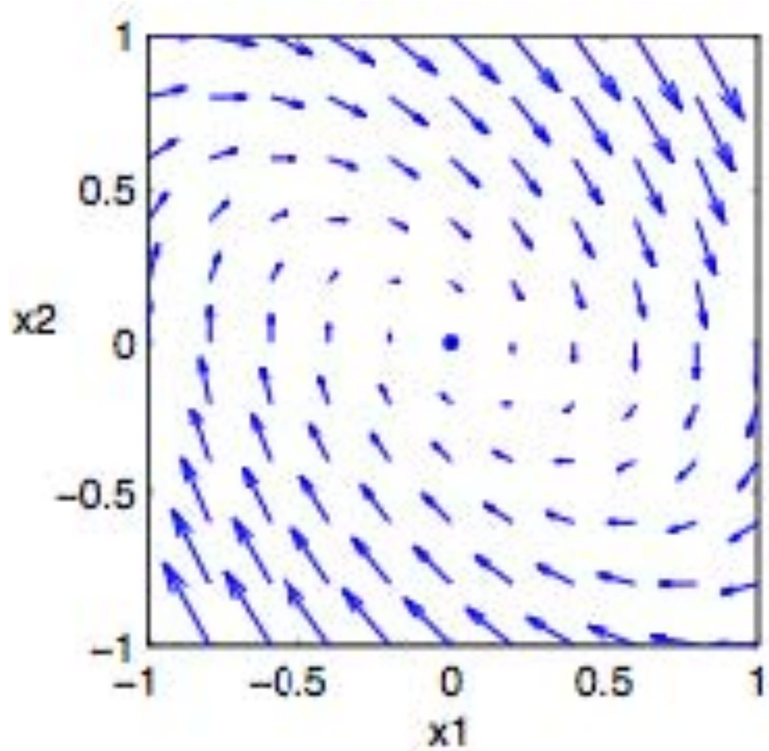
dynamic behavior can visualized for 2D systems using “phase portraits”

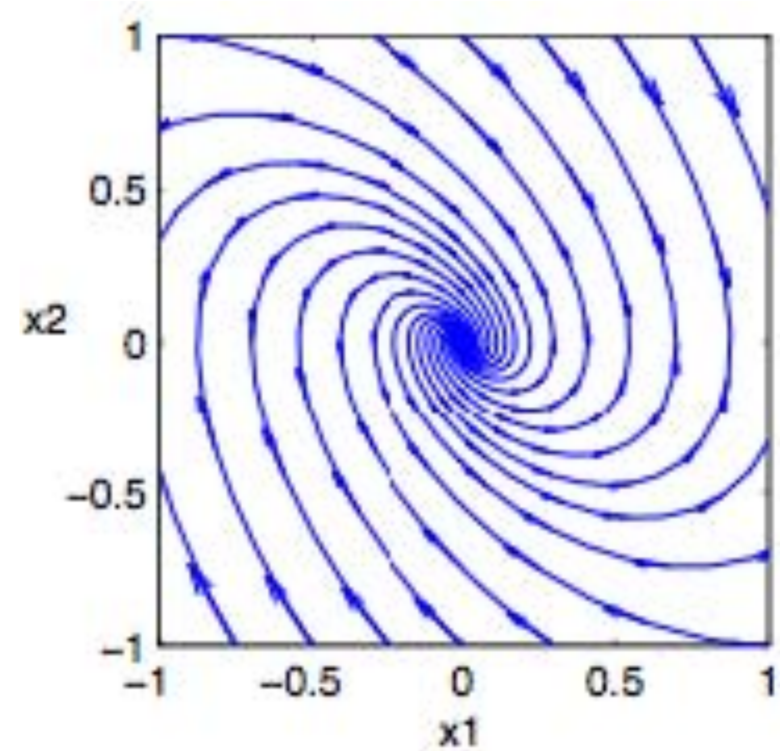
Phase plane plots show 2D dynamics as *vector fields* & *stream functions*

- $\dot{x} = f(x, u(x)) = F(x)$
- Plot $F(x)$ as a vector on the plane; stream lines follow the flow of the arrows

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_1 - x_2 \end{bmatrix}$$

python matplotlib function: ‘streamplot’



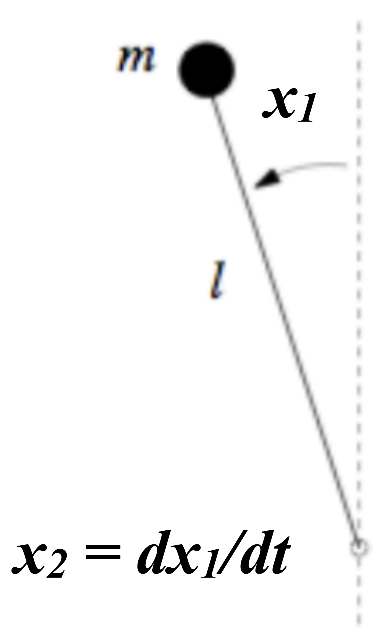


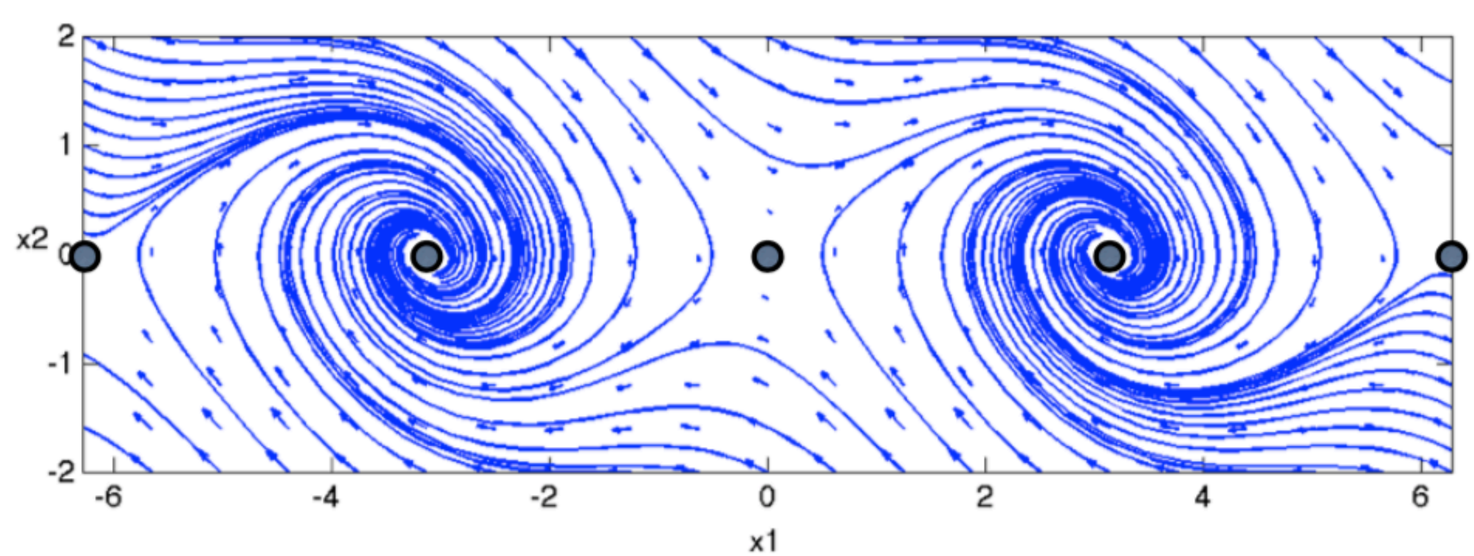
equilibrium points

Equilibrium points represent stationary conditions for the dynamics

The *equilibria* of the system $\dot{x} = f(x)$ are the points x_e such that $f(x_e) = 0$.

$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ \sin x_1 - \gamma x_2 \end{bmatrix} \qquad \Rightarrow \qquad x_e = \begin{bmatrix} \pm n\pi \\ 0 \end{bmatrix}$$





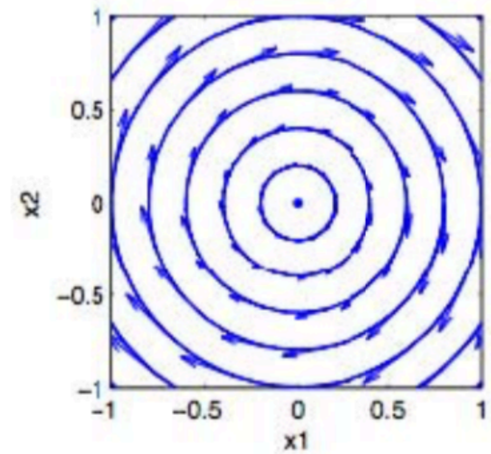
stability of equilibrium points

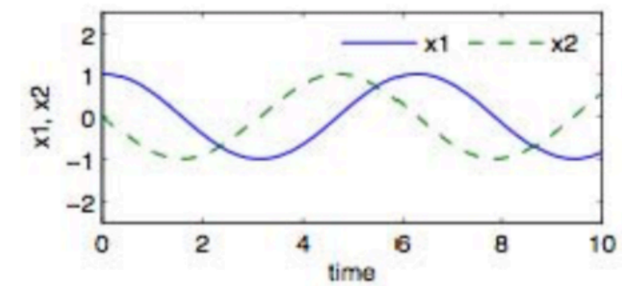
An equilibrium point is:

Stable if initial conditions that start near the equilibrium point, stay near

- Also called “stable in the sense of Lyapunov
- For all $\varepsilon > 0$, there exists $\delta\ s.t.$

$$\|x(0) - x_e\| < \delta \implies \|x(t) - x_e\| < \epsilon$$

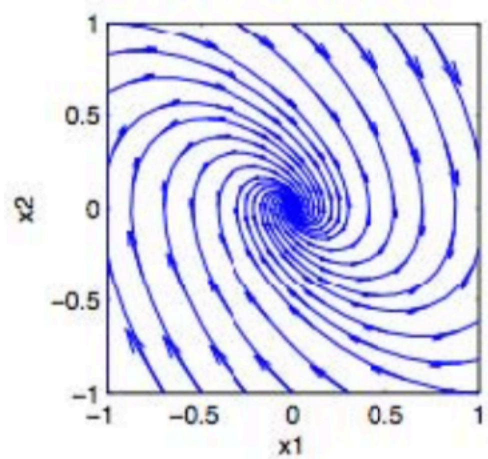


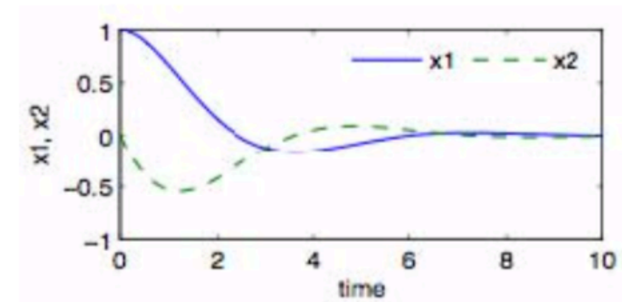


“stable” but not asymptotically stable

Asymptotically stable if all nearby initial conditions converge to the equilibrium point

- Stable + converging



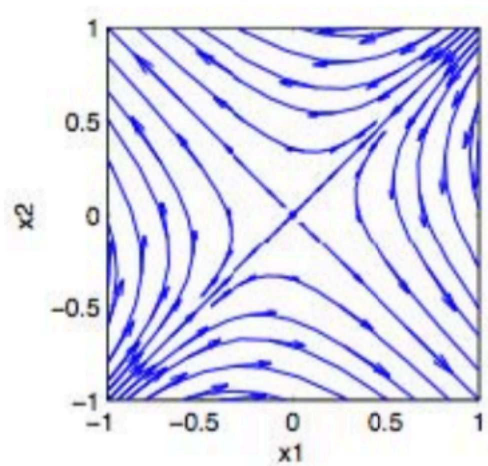


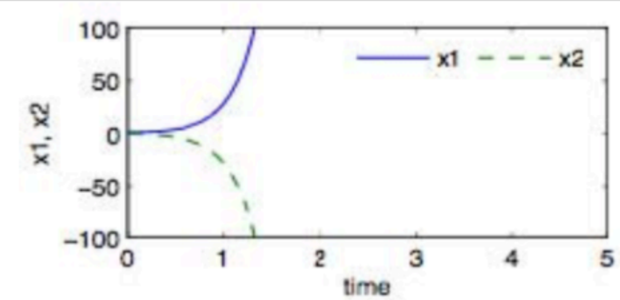
$$\lim_{t \to \infty} x(t) = x_e \quad \forall \|x(0) - x_e\| < \epsilon$$

asymptotically stable

Unstable if some initial conditions diverge from the equilibrium point

- May still be some initial conditions that converge



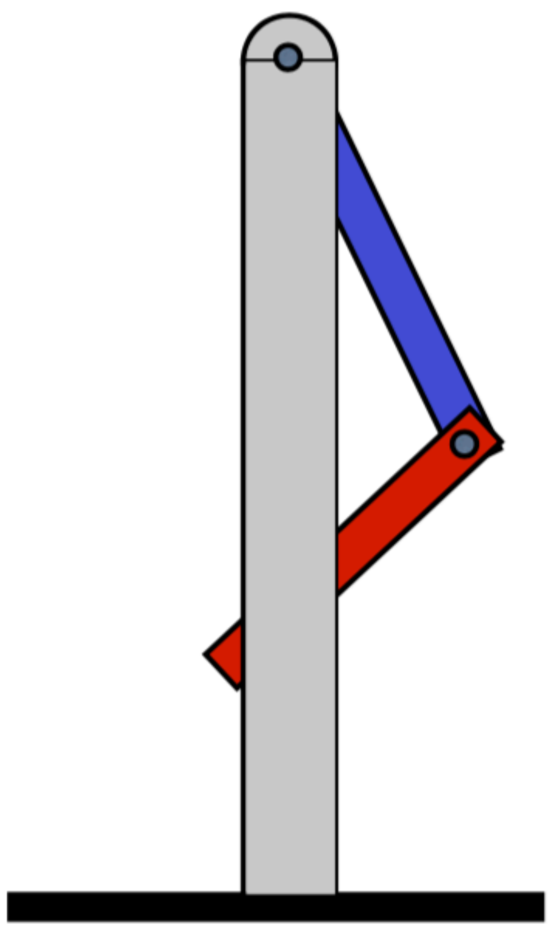


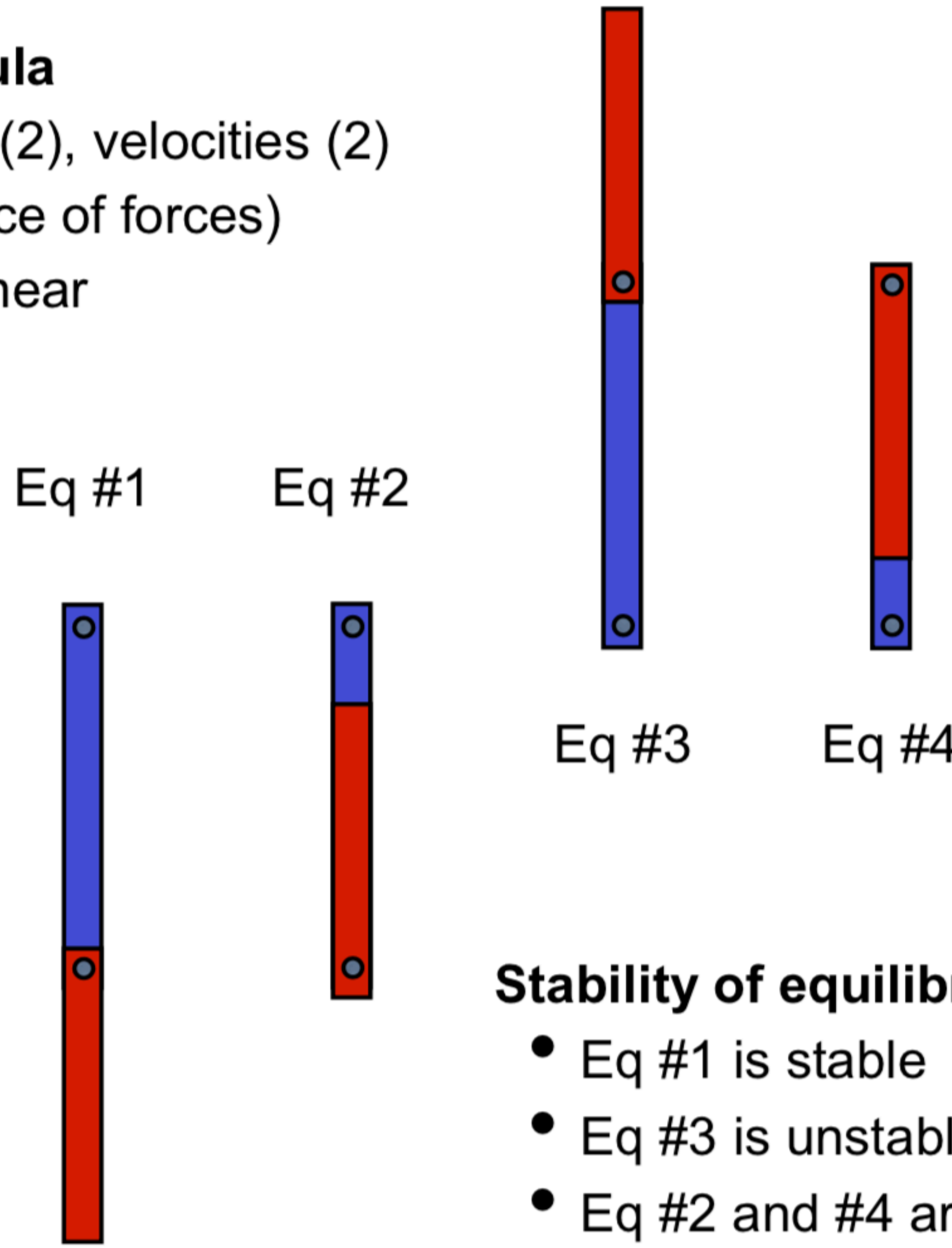
unstable

Example #1: Double Inverted Pendulum

Two series coupled pendula

- States: pendulum angles (2), velocities (2)
- Dynamics: *F* = *ma* (balance of forces)
- Dynamics are very nonlinear





Stability of equilibria

- Eq #1 is stable
- Eq #3 is unstable
- Eq #2 and #4 are unstable, but with some stable "modes"

Local Stability of Nonlinear Systems

Asymptotic stability of the linearization implies *local* asymptotic stability of equilibrium point

- Linearization around equilibrium point captures "tangent" dynamics

$$\dot{x} = \cancel{F(x_e)}^{\,0} + \left.\frac{\partial F}{\partial x}\right|_{x_e} (x - x_e) + \text{higher order terms} \xrightarrow{\text{approx}} \begin{array}{l} z = x - x_e \\ \dot{z} = Az \end{array}$$

- linearization is *stable* $\Longrightarrow$ nonlinear system *locally stable*
- linearization is *unstable* $\Longrightarrow$ nonlinear system *locally unstable*

- "degenerate case": if linearization is *stable* but not *asymptotically stable* $\Longrightarrow$ cannot tell whether nonlinear system is stable or not!

$$\dot{x} = \pm x^3 \xrightarrow{\textit{linearize}} \dot{x} = 0$$

- linearization is stable (but not asy stable)
- nonlinear system can be asy stable or unstable

Local linear approximation is valuable for control design:

- if dynamics are well-approximated by linearization near an equilibrium point, controller can *ensure* stability there (!)
- controller task: make the linearization stable

Linearization about an equilibrium point

$$\dot{x} = f(x,u) \qquad\qquad \dot{z} = Az + Bv$$
$$y = h(x,u) \quad \longrightarrow \quad w = Cz + Dv$$

to "linearize" around $x = x_e$:

1. find x_e, u_e such that $f = 0$

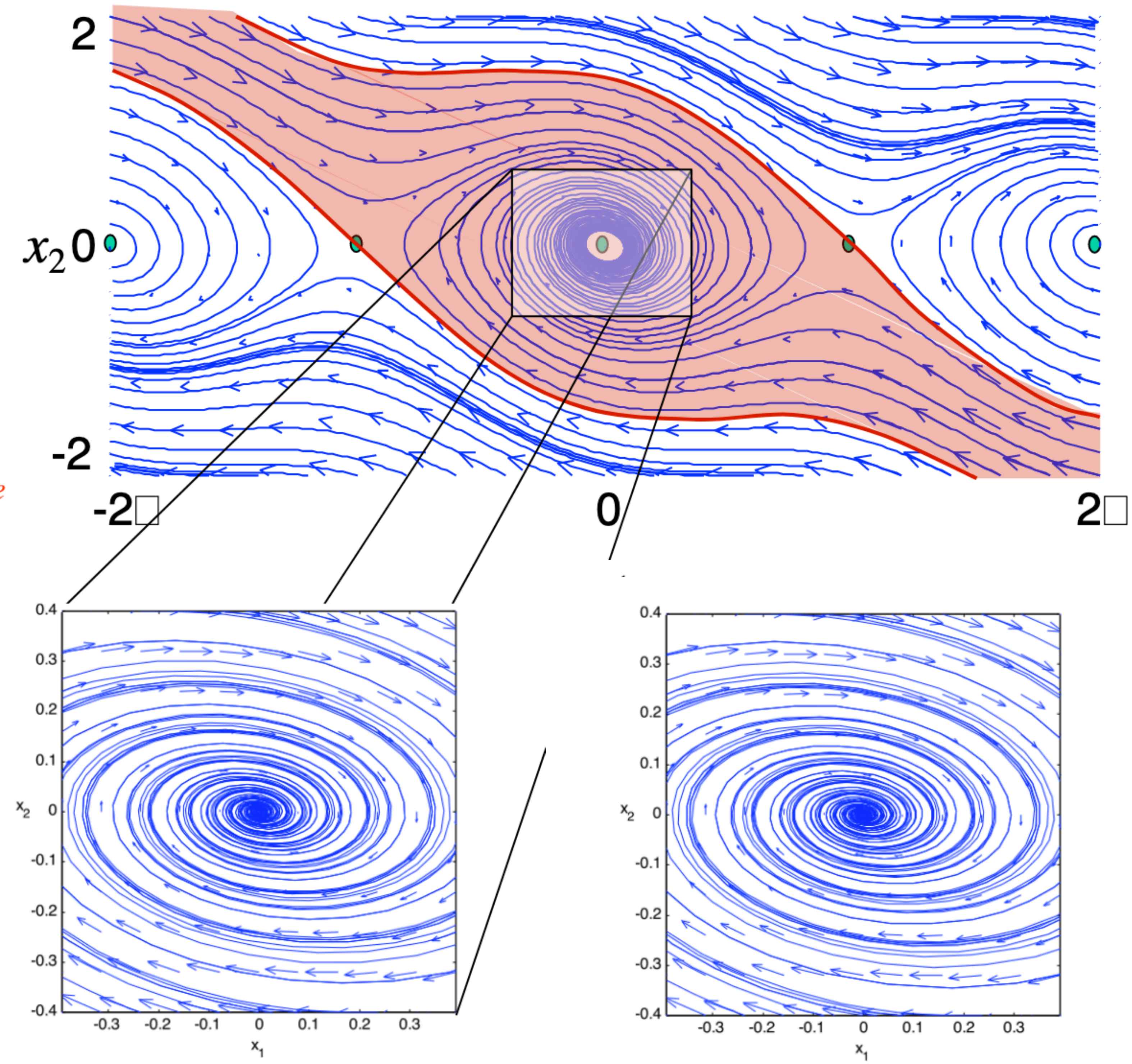
2. define $y_e = h(x_e, u_e)$

$z = x - x_e \qquad v = u - u_e \qquad w = y - y_e$

3. then

$$A = \left.\frac{\partial f}{\partial x}\right|_{(x_e,u_e)} \qquad B = \left.\frac{\partial f}{\partial u}\right|_{(x_e,u_e)}$$

$$C = \left.\frac{\partial h}{\partial x}\right|_{(x_e,u_e)} \qquad D = \left.\frac{\partial h}{\partial u}\right|_{(x_e,u_e)}$$



Full nonlinear model   Linear model (honest!)

Remarks

- In examples, this is often equivalent to small angle approximations, etc
- Only works *near* to equilibrium point
- use linearization to design controller

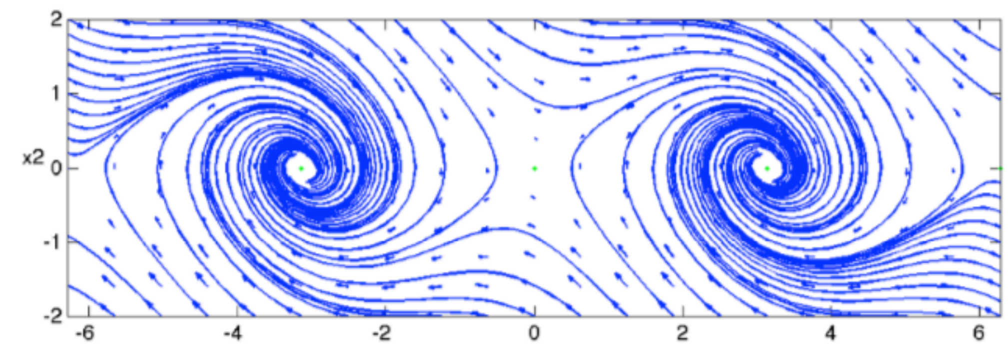
big idea: if combined linearized system + controller is stable
$\Longrightarrow$ full nonlinear system is stable nearby

Jacobian linearization matrix

$$A = \left.\frac{\partial f}{\partial x}\right|_{(x_e, u_e)} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}\right|_{(\boldsymbol{x_e}, \boldsymbol{u_e})}$$

Example: Stability Analysis of Inverted Pendulum

System dynamics



$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ \sin x_1 - \gamma x_2 \end{bmatrix},$$

Upward equilibrium:

$$\theta = x_1 \ll 1 \quad \Longrightarrow \quad \sin x_1 \approx x_1$$

$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ x_1 - \gamma x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -\gamma \end{bmatrix} x$$

- Eigenvalues: $-\frac{1}{2}\gamma \pm \frac{1}{2}\sqrt{4+\gamma^2}$ **for $\gamma = 0.1$, $\lambda \cong (0.95, -1.05) \Rightarrow$ unstable**

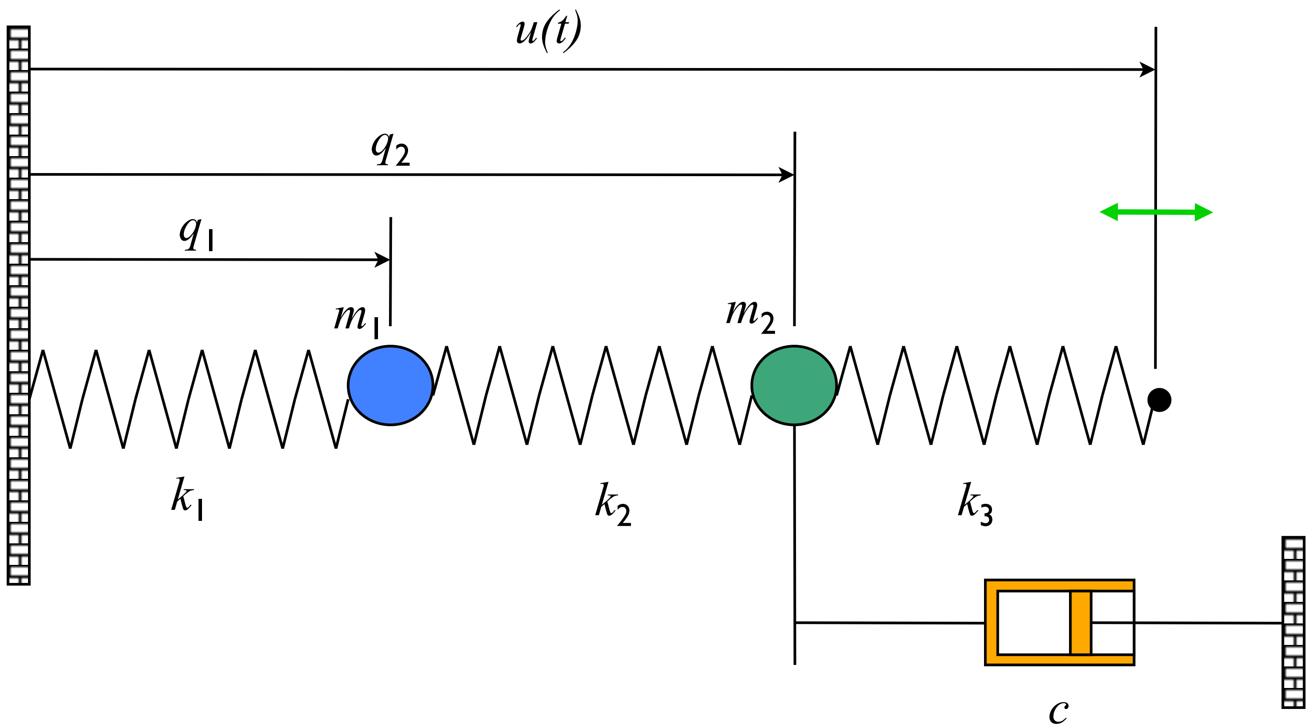
Downward equilibrium:

- Linearize around $x_1 = \pi + z_1$: $\sin(\pi + z_1) = -\sin z_1 \approx -z_1$
- Eigenvalues:

$$\begin{aligned} z_1 &= x_1 - \pi \\ z_2 &= x_2 \end{aligned} \quad \longrightarrow \quad \frac{dz}{dt} = \begin{bmatrix} z_2 \\ -z_1 - \gamma\, z_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -\gamma \end{bmatrix} z$$

$$-\frac{1}{2}\gamma \pm \frac{1}{2}\sqrt{-4+\gamma^2}$$ **for $\gamma = 0.1$, $\lambda \cong (-0.05+i, -0.05-i) \Rightarrow$ stable**

example 2: matrix representation of a linear system



Model: rigid body physics

- Sum of forces = mass * acceleration
- Hooke's law: $F = k(x - x_{\text{rest}})$
- Viscous friction: $F = c\,v$

$$\boxed{\begin{aligned} m_1\ddot{q}_1 &= k_2(q_2 - q_1) - k_1 q_1 \\ m_2\ddot{q}_2 &= k_3(u - q_2) - k_2(q_2 - q_1) - c\dot{q}_2 \end{aligned}}$$

$$\frac{d}{dt}\begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \frac{k_2}{m}(q_2 - q_1) - \frac{k_1}{m} q_1 \\ \frac{k_3}{m}(u - q_2) - \frac{k_2}{m}(q_2 - q_1) - \frac{c}{m}\dot{q} \end{bmatrix}$$

$$y = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

"State space form"

Matrix representation:

$$\dot{x} = Ax + Bu$$

$$\dot{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{k_1 + k_2}{m} & \frac{k_2}{m} & 0 & 0 \\ -\frac{k_2}{m} & -\frac{k_2 + k_3}{m} & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k_3}{m} \end{bmatrix} u$$

$$y = [1 \quad 1 \quad 0 \quad 0]x = Cx$$

State Space Control Design Concepts

System description: single input, single output system (MIMO also OK)

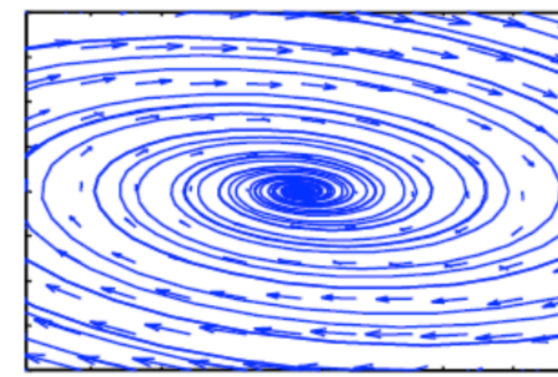
$$\dot{x} = f(x,u) \qquad x \in \mathbb{R}^n,\ x(0) \text{ given}$$
$$y = h(x) \qquad u \in \mathbb{R},\ y \in \mathbb{R}$$

Stability: stabilize the system around an equilibrium point

- Given equilibrium point $x_e \in \mathbb{R}^n$, find control "law" $u = \alpha(x)$ such that

$$\lim_{t\to\infty} x(t) = x_e \text{ for all } x(0) \in \mathbb{R}^n$$

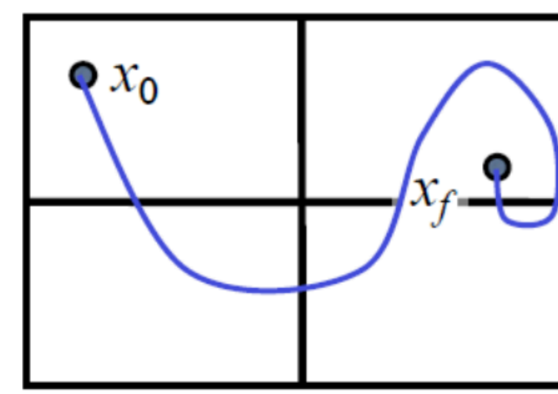
- Often choose x_e so that $y_e = h(x_e)$ has desired value r (constant)



Reachability: steer the system between two points

- Given $x_o, x_f \in \mathbb{R}^n$, find an input $u(t)$ such that

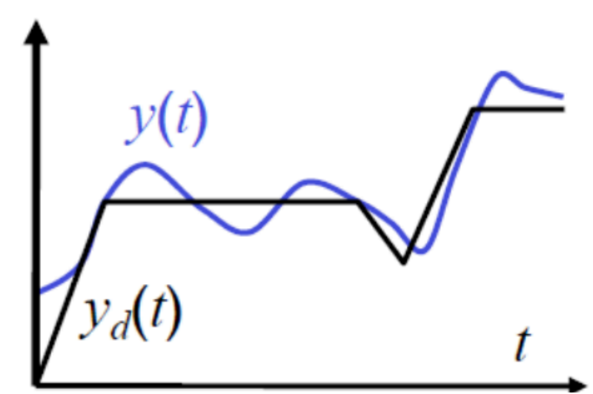
$$\dot{x} = f(x, u(t)) \text{ takes } x(t_0) = x_0 \to x(T) = x_f$$



Tracking: track a given output trajectory

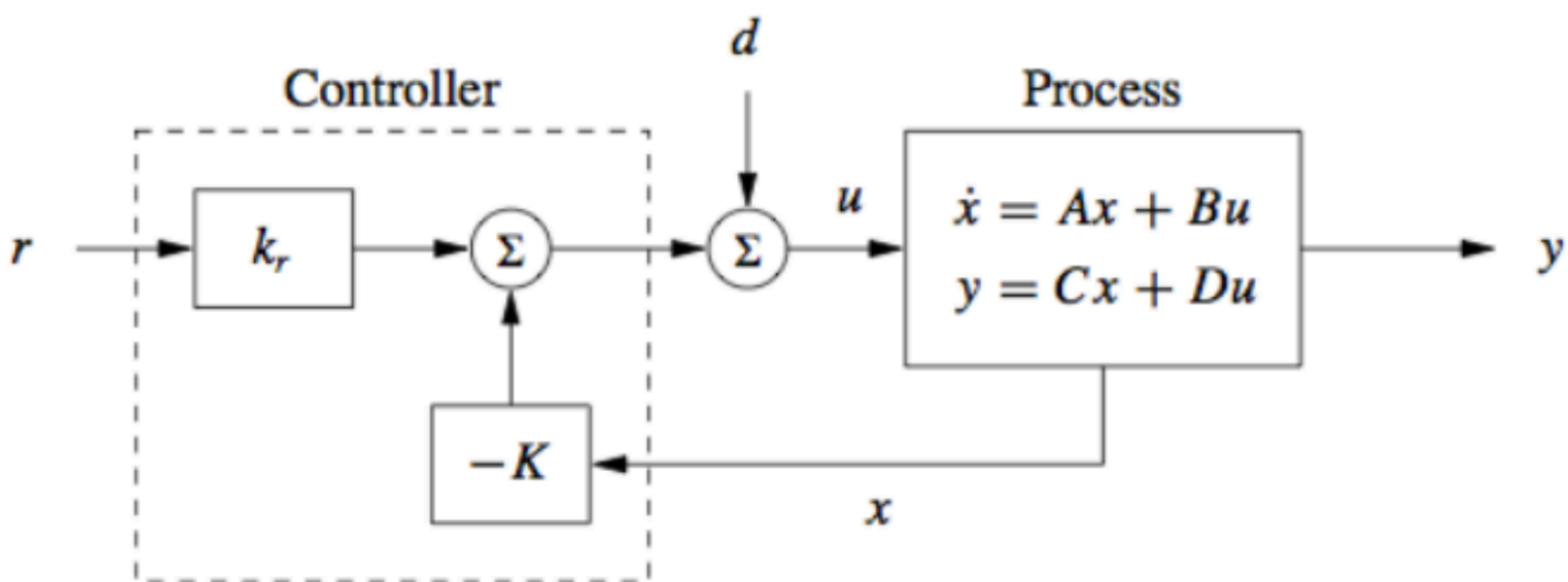
- Given $r = y_d(t)$, find $u = \alpha(x,t)$ such that

$$\lim_{t\to\infty} \left(y(t) - y_d(t)\right) = 0 \text{ for all } x(0) \in \mathbb{R}^n$$



State Feedback: $u(t) = -Kx(t)$

- Can place poles arbitrarily if system is reachable
- Can relate poles to performance criteria, such as overshoot.
- Can add "dynamic compensator", such as integral feedback, which overcomes modeling errors or uncertainty.
- *But,* states cannot always be measured, as needed for feedback.



Tests for Reachability

$$\dot{x} = Ax + Bu \qquad x \in \mathbb{R}^n,\ x(0) \text{ given}$$
$$y = Cx \qquad u \in \mathbb{R},\ y \in \mathbb{R}$$

$$x(T) = e^{AT}x_0 + \int_{\tau=0}^{T} e^{A(T-\tau)}Bu(\tau)d\tau$$

Thm A linear system is reachable if and only if the $n \times n$ *reachability matrix*

$$\begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

is full rank.

Note: also called "controllability" matrix

Remarks

- *Very* simple test to apply. In MATLAB, use ctrb(A,B) and check rank w/ det()
- If this test is satisfied, we say "the pair (A,B) is reachable"
- Some insight into the proof can be seen by expanding the matrix exponential

$$\begin{aligned} e^{A(T-\tau)}B &= \left(I + A(T-\tau) + \frac{1}{2}A^2(T-\tau)^2 + \cdots + \frac{1}{(n-1)!}A^{n-1}(T-\tau)^{n-1} + \cdots\right)B \\ &= B + AB(T-\tau) + \frac{1}{2}A^2B(T-\tau)^2 + \cdots + \frac{1}{(n-1)!}A^{n-1}B(T-\tau)^{n-1} + \cdots \end{aligned}$$

State space controller design for linear systems

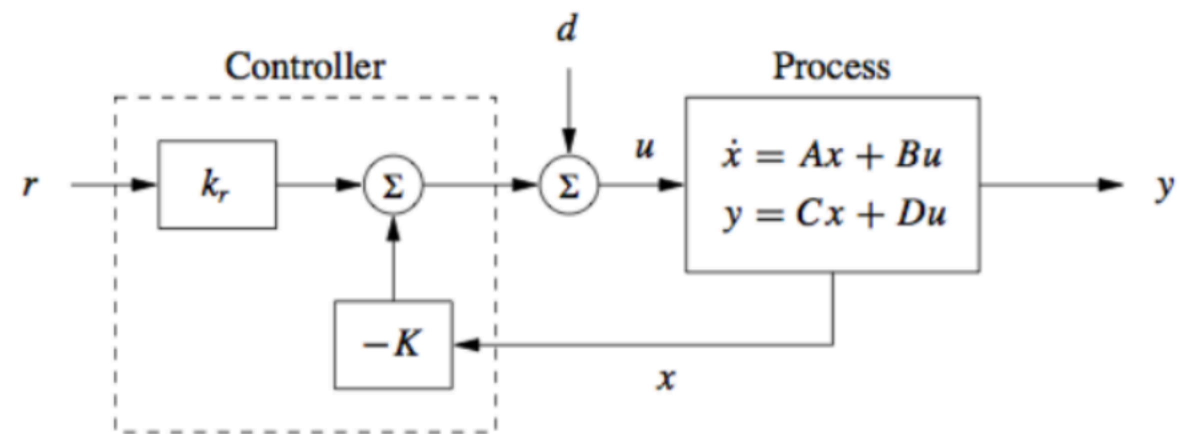
$$\dot{x} = Ax + Bu \qquad x \in \mathbb{R}^n,\ x(0) \text{ given}$$
$$y = Cx \qquad u \in \mathbb{R},\ y \in \mathbb{R}$$

$$x(T) = e^{AT}x_0 + \int_{\tau=0}^{T} e^{A(T-\tau)}Bu(\tau)d\tau$$

Goal: find a linear control law $u = -Kx + k_r r$ such that the closed loop system

$$\dot{x} = Ax + Bu = (A - BK)x + Bk_r r$$

is stable at equilibrium point x_e with $y_e = r$.



Remarks

- If $r = 0$, control law simplifies to $u = -Kx$ and system becomes $\dot{x} = (A - BK)x$
- Stability based on eigenvalues ⇒ use K to make eigenvalues of $(A - BK)$ stable
- Can also link eigenvalues to *performance* (eg, initial condition response)
- Question: when can we place the eigenvalues anyplace that we want?

Theorem The eigenvalues of $(A - BK)$ can be set to arbitrary values if and only if the pair (A, B) is reachable.

MATLAB/Python: K = place(A, B, eigs)

Python users: use python-control toolbox
(available at python-control.org)