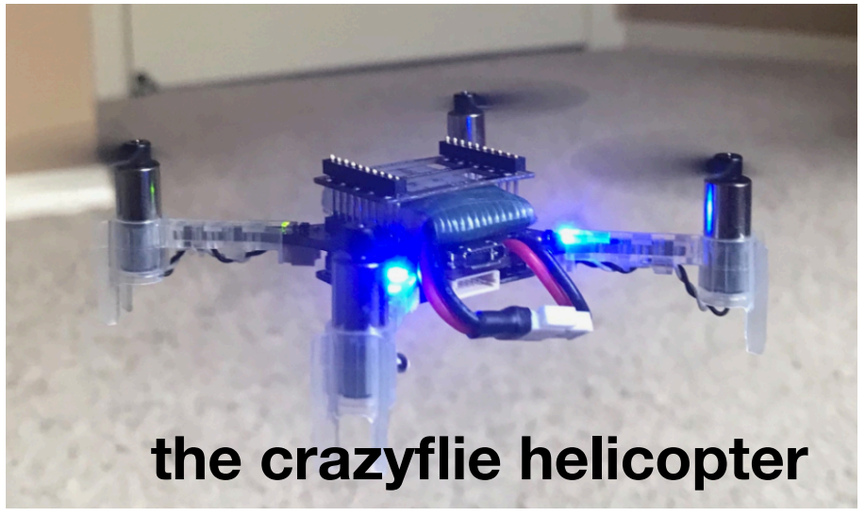
control and dynamics

Prof. Sawyer B. Fuller

ME 586: Biology-inspired robotics

Motivating extended example: small helicopters & insects



- objectives: learn some basics of dynamics and control
- Option platform for simulation project
  - Example: flying insect
    - ~1 g to 1 mg
    - Hours-long flight time
  - Example: Crazyflie helicopter:
    - ~30 g, ~4 minute flight time
    - communicates in real-time over bluetooth to laptop
    - sensor suite gives information needed to stabilize and control flight

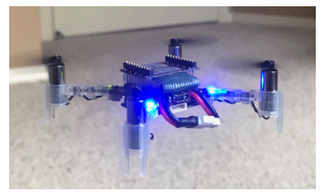
crazyflie in operation performing odor source localization

Odor Localization

Anderson, Sullivan, Horiuchi, Fuller, & Daniel,
Bioinspiration & Biomimetics
2020

UNIVERS

The control task we will cover



q_d (desired state)

e_q (error)

high-level controller

controller

system dynamics

q (system state)
[*x, y, z position,*
x, y, z velocity,
roll, pitch, yaw angles,
roll, pitch, yaw rates]

possible course projects: using stable hover as starting point, build high-level behaviors

in this course we will learn ways to design these

u (system input)
[thrust force,
roll torque,
pitch torque]

$\hat{q}$
(estimated state)

estimator

sensors

y (system output)
"measurement"
[*roll, pitch, yaw rates* (gyro),
x-optic flow, y-optic flow (optic flow camera),
z-distance measurement (time of flight)]

first step in control design: assume full state is known

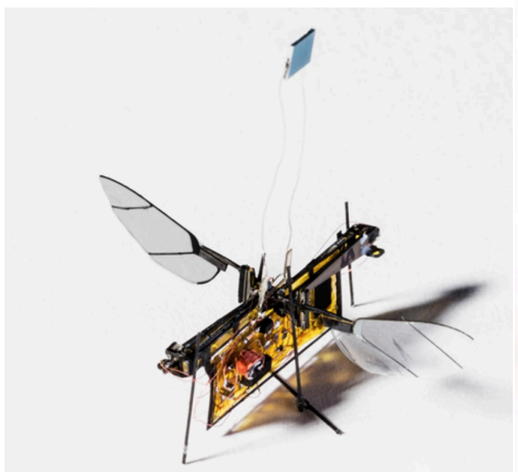
model-based or model-free

model-based control for basic stability

basics: actuation for hovering

honeybee

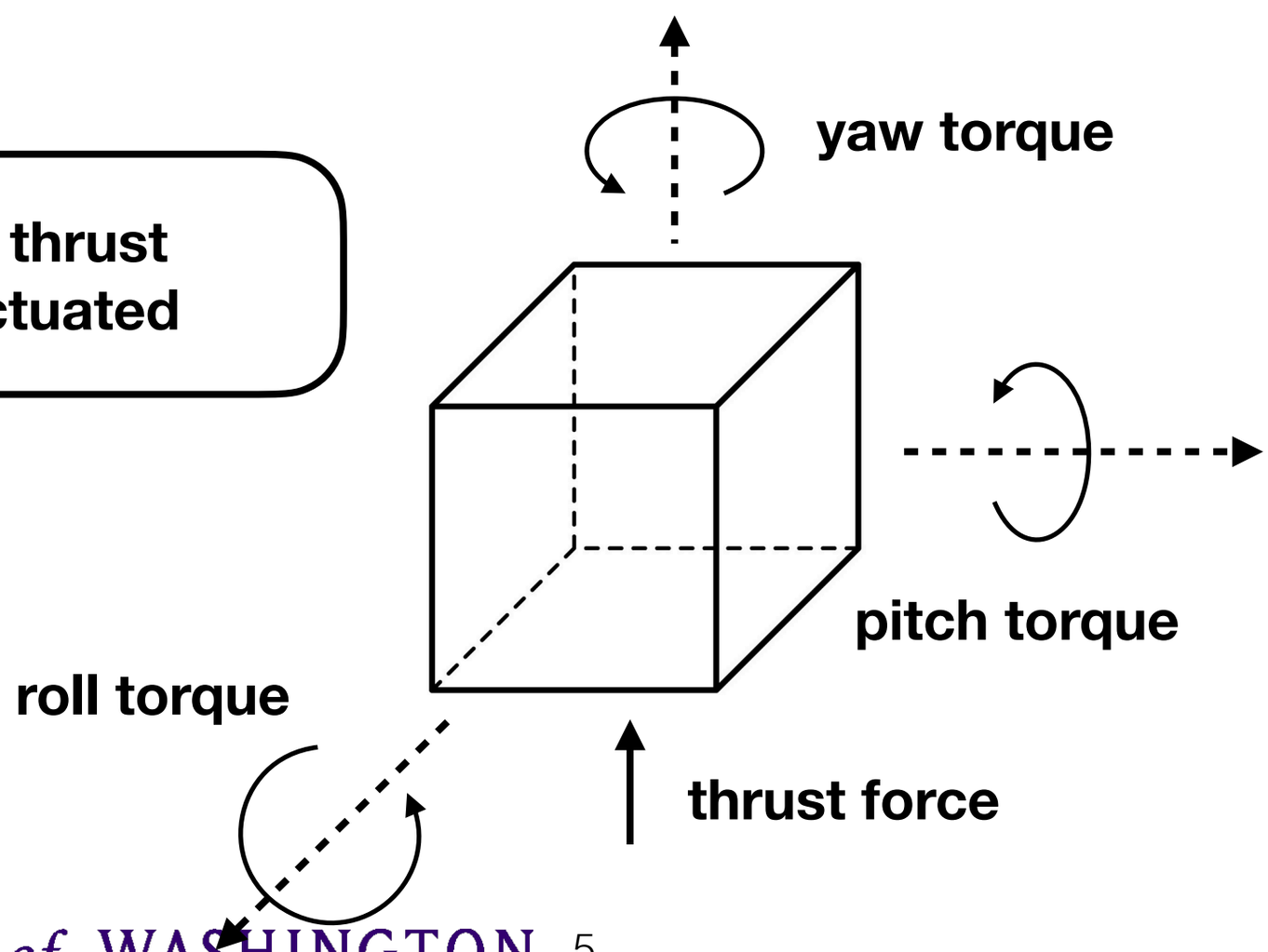
single-rotor helicopter



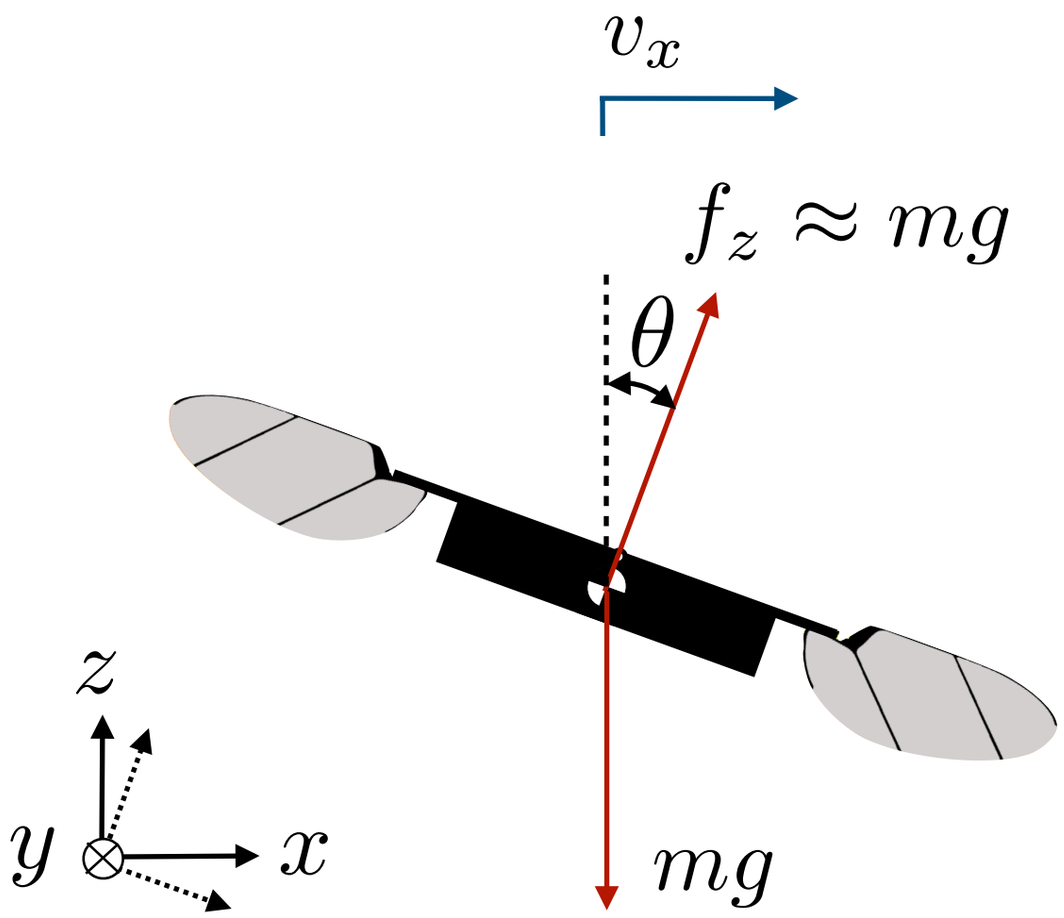
robot flies e.g. UW Robofly

four-rotor aircraft "quad-rotors"

typically, lateral thrust is not directly actuated



lateral actuation by tilting

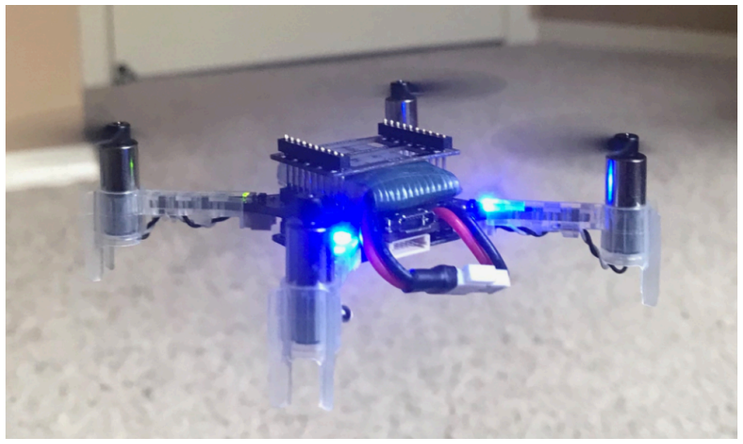


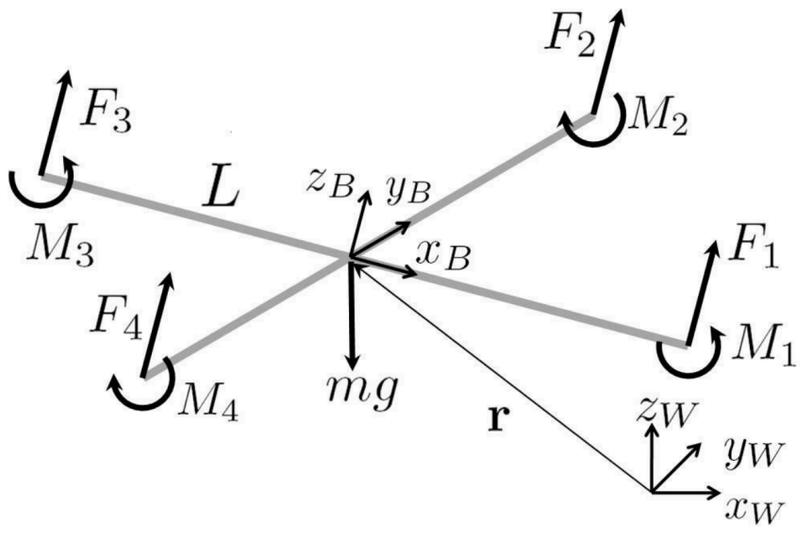
lateral acceleration

$$\dot{v}_x = \frac{1}{m} mg \sin\theta = g \sin\theta$$
$$\approx g\theta \quad \textbf{for small } \theta$$

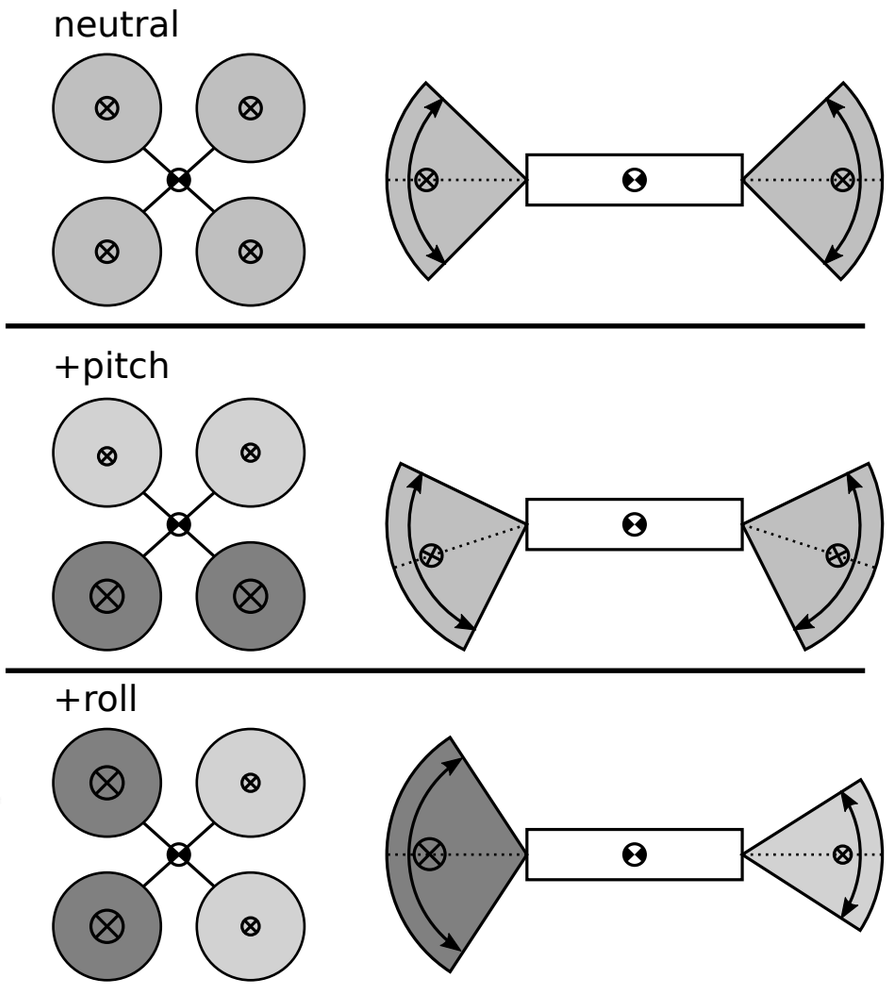
- “helicopter-like” lateral control

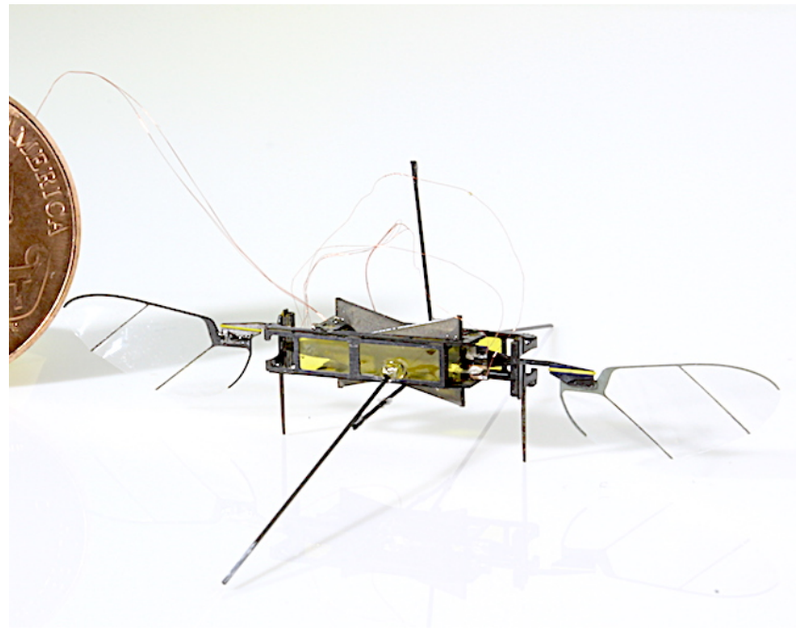
quad-rotor actuation





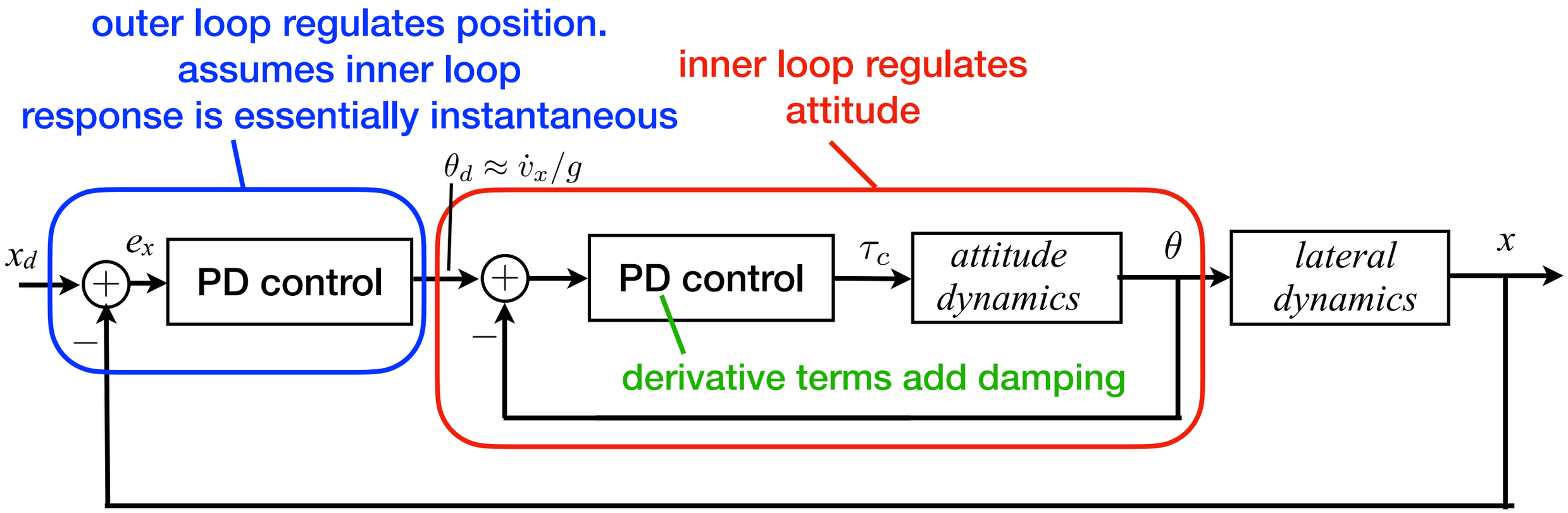
actuation with two wings



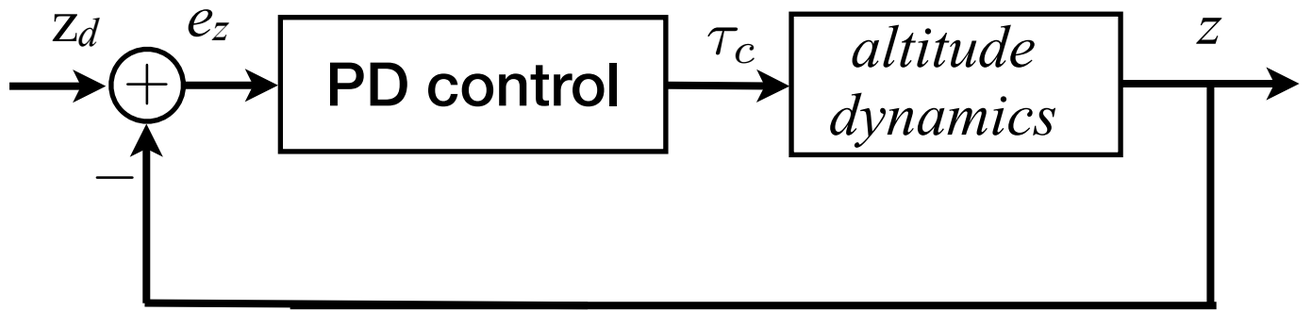


- two rotors spin one direction and two in the other direction
- vary angle and amplitude of flapping wings

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

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

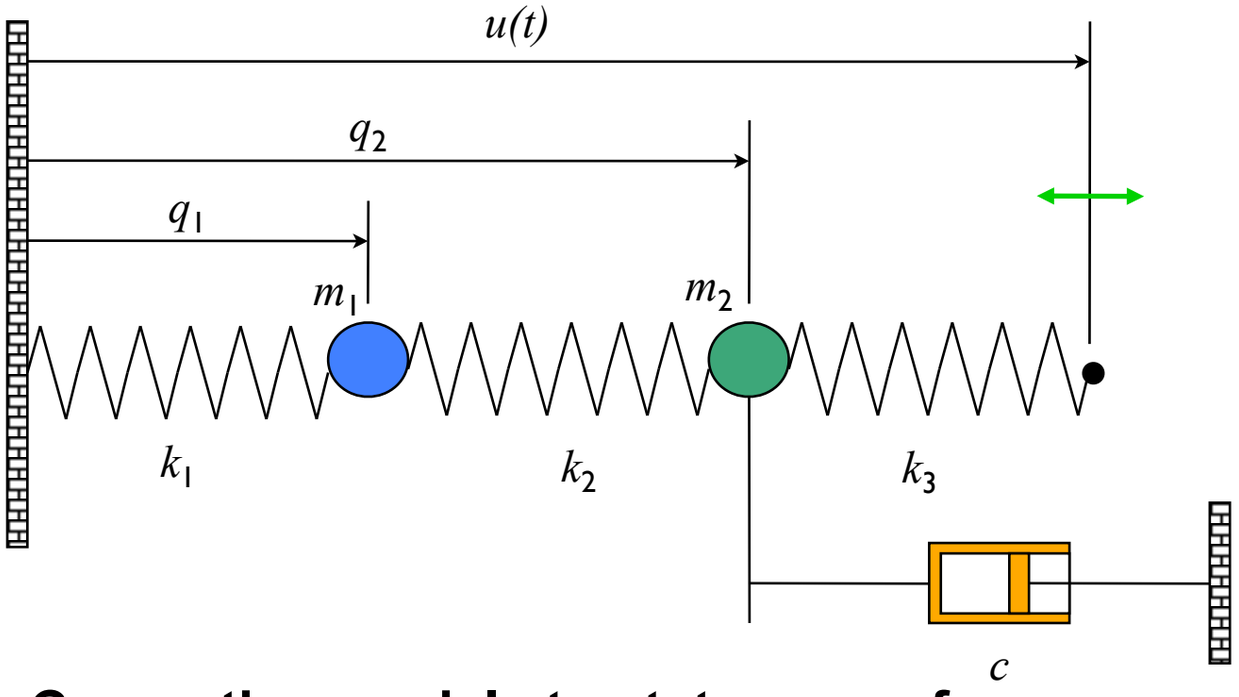
$$\Sigma \boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}$$

$\boldsymbol{f}, \boldsymbol{\tau}$	**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 these equations 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: $F = c\,v$

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

Converting models to state space form

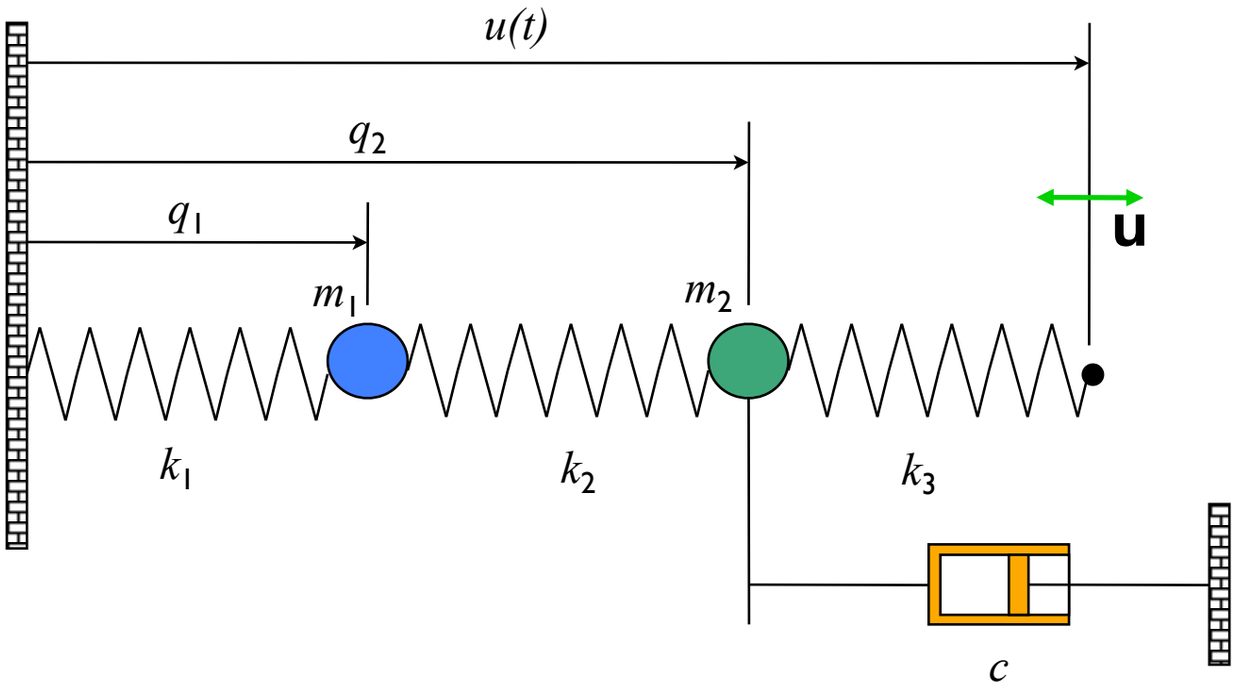
- Construct a *vector* of the variables that are required to specify the evolution of the system
- Write dynamics as a *system* of first order differential equations:

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

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

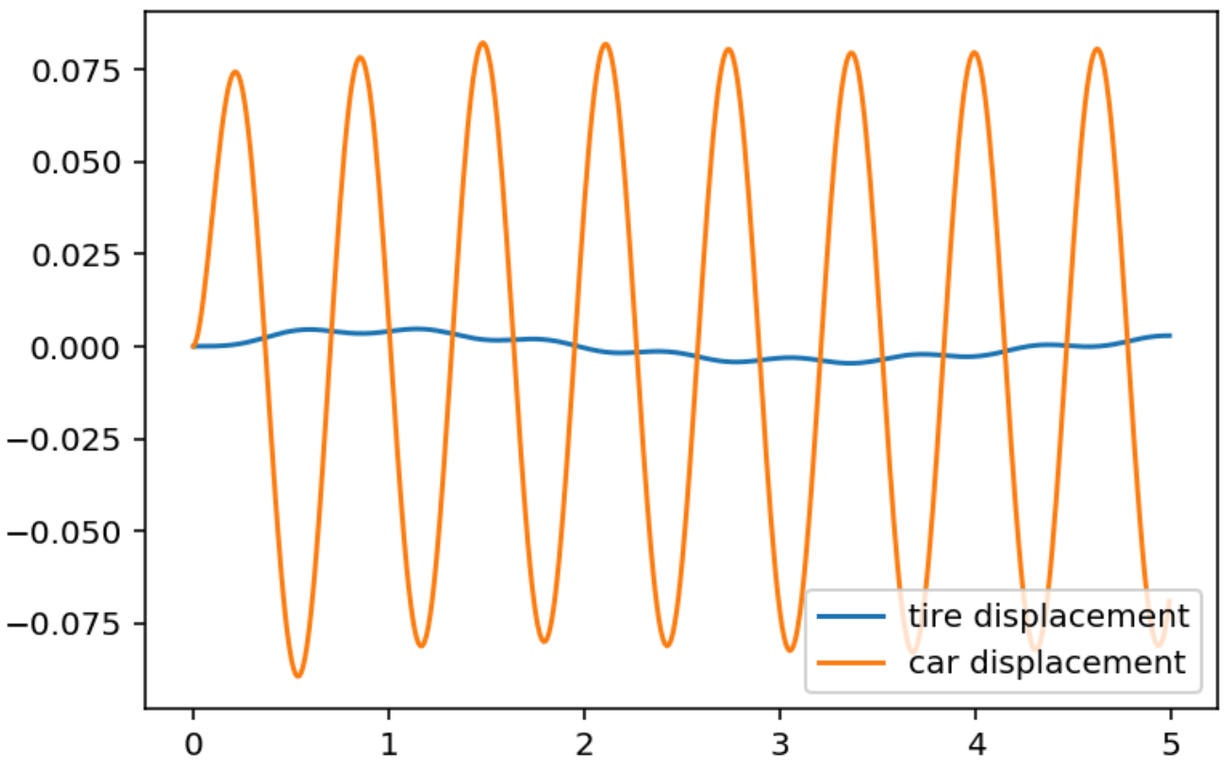
"State space form"

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))
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))

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'))
```

initial condition

dynamics function "f"

update step