ME 328: Medical Robotics
Winter 2019

Lecture 5:
Robot dynamics and simulation

Allison Okamura
Stanford University

Robot dynamics

equations of motion

describe the relationship between forces/torques and motion (in joint space or workspace variables)

two possible goals:

1. Given motion variables (e.g. $\vec{\theta}, \dot{\vec{\theta}}, \ddot{\vec{\theta}}$ or $\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}$), what joint torques ($\vec{\tau}$) or end-effector forces ($\vec{f}$) would have been the cause? *(this is inverse dynamics)*

2. Given joint torques ($\vec{\tau}$) or end-effector forces ($\vec{f}$), what motions (e.g. $\vec{\theta}, \dot{\vec{\theta}}, \ddot{\vec{\theta}}$ or $\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}$) would result? *(this is forward dynamics)*

developing equations of motion using Lagrange's equation

The Lagrangian is $L = T - V$

where T is the kinetic energy of the system and V is the potential energy of the system

Lagrange's equation is $\boxed{\dfrac{d}{dt}\left(\dfrac{\partial L}{\partial \dot{q}_j}\right) - \dfrac{\partial L}{\partial q_j} = Q_j}$

where $j = 1, 2, \ldots, n$, and $\dot{q}_j = \dfrac{\partial q_j}{\partial t}$ is the generalized velocity and Q_j is the nonconservative generalized force corresponding to the generalized coordinate q_j

what are generalized coordinates?

- equations of motion can be formalized in a number of different coordinate systems

- n independent coordinates are necessary to describe the motion of a system having n degrees of freedom

- any set of n independent coordinates is called generalized coordinates: $q_1, q_2, \ldots q_n$

- these coordinates may be lengths, angles, etc.

generalized forces

- When external forces act on the system, the configuration changes: generalized coordinates q_j change by δq_j , $j = 1, 2, \ldots, n$

- If U_j is the work done in changing q_j by δq_j , the corresponding generalized force is $Q_j = \dfrac{U_j}{\delta q_j}$, where $j = 1, 2, \ldots, n$

- Q_j is a force/moment and q_j is a linear/angular displacement. This can include dissipation (damping).

where does Lagrange's equation come from?

Hamilton's principle of least action: a system moves from $q(t_1)$ to $q(t_2)$ in such a way that the following integral takes on the *least* possible value.
$S = \int_{t_1}^{t_2} L(q, \dot{q}, t) dt$

The calculus of variations is used to obtain Lagrange's equations of motion. We're concerned with minimizing
$\int_{t_1}^{t_2} f\left(y(t), \dot{y}(t); t\right) dt$

The minimization leads to the equation
$\frac{\partial f}{\partial y} - \frac{d}{dt}\frac{\partial f}{\partial \dot{y}} = 0$

using Lagrange's equation to derive equations of motion

$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_j}) - \frac{\partial L}{\partial q_j} = Q_j$, where $j = 1, 2, \ldots, n$

$L = T - V$

Substitute:
$\frac{\partial L}{\partial \dot{q}_j} = \frac{\partial T}{\partial \dot{q}_j} - \frac{\partial V}{\partial \dot{q}_j}$
Last term is zero because P.E. is not dependent on velocities $\rightarrow \frac{\partial L}{\partial \dot{q}_j} = \frac{\partial T}{\partial \dot{q}_j}$
$\frac{\partial L}{\partial q_j} = \frac{\partial T}{\partial q_j} - \frac{\partial V}{\partial q_j}$

$\frac{d}{dt}(\frac{\partial T}{\partial \dot{q}_j}) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = Q_j$, where $j = 1, 2, \ldots, n$

Q_j is a nonconservative generalized force corresponding to coordinate q_j, e.g. damping

For a conservative system,
$\frac{d}{dt}(\frac{\partial T}{\partial \dot{q}_j}) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = 0$, where $j = 1, 2, \ldots, n$

adding dissipation

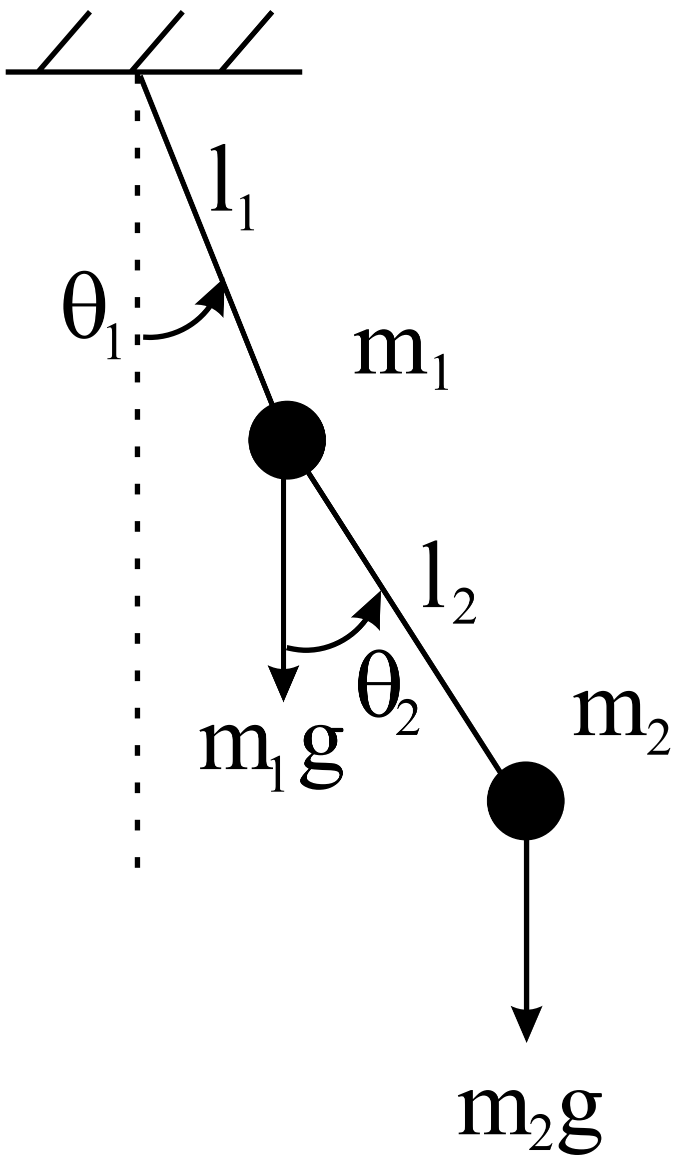
since the left side of Lagrange's equation only includes terms for potential and kinetic energy, any dissipative terms (e.g., damping) must be added on the right hand side (and Q_j are now only the input forces/torques)

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = Q_j - \frac{\partial R}{\partial \dot{q}_j}$$

where $R = \frac{1}{2}\sum_j b_j \dot{q}_j^2$

is a simplified form of Rayleigh's dissipation function

example: double pendulum (review on your own)



Velocity of m_1: $v_1 = l_1\dot{\theta}_1$

Velocity of m_2: $v_2 = (v_{2x}^2 + v_{2y}^2)$

$$v_{2x} = l_1\dot{\theta}_1\cos(\theta_1) + l_2\dot{\theta}_2\cos(\theta_2)$$

$$v_{2y} = l_1\dot{\theta}_1\sin(\theta_1) + l_2\dot{\theta}_2\sin(\theta_2)$$

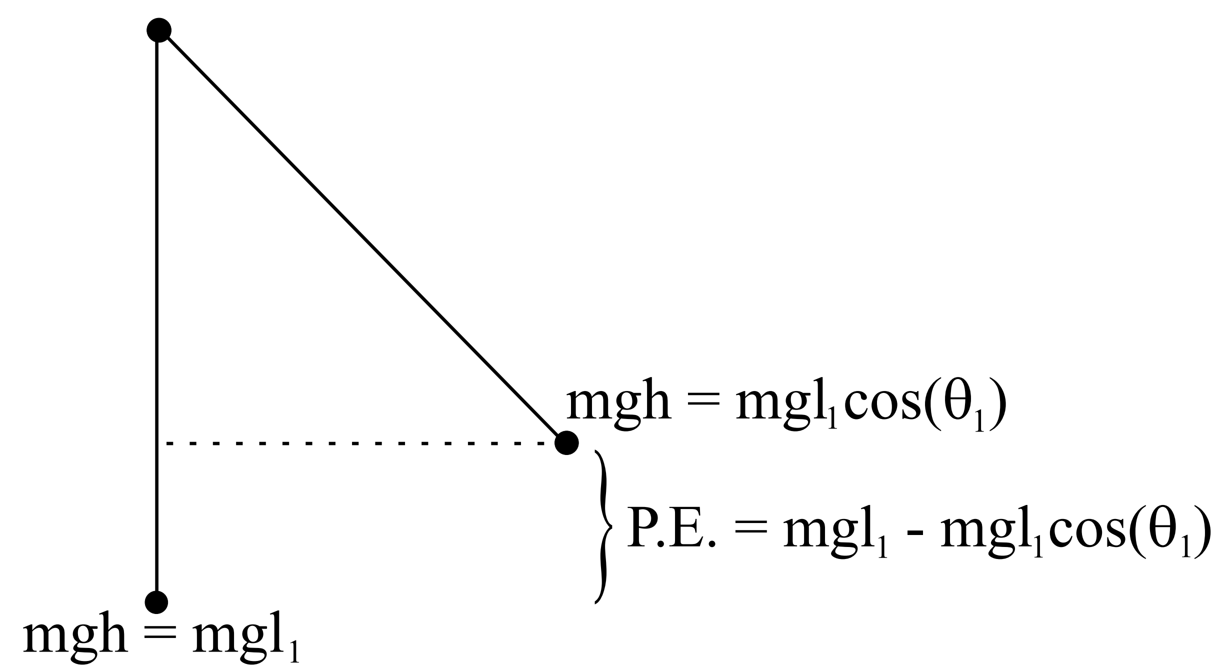
example: double pendulum

Kinetic energy:

$$T = \frac{1}{2}m_1(l_1\dot{\theta}_1)^2+\frac{1}{2}m_2((l_1\dot{\theta}_1\cos(\theta_1)+l_2\dot{\theta}_2\cos(\theta_2))^2+(l_1\dot{\theta}_1\sin(\theta_1)+l_2\dot{\theta}_2\sin(\theta_2))^2)$$

Potential energy

$$V = m_1 g l_1(1-\cos(\theta_1)) + m_2 g(l_1(1-\cos(\theta_1)) + l_2(1-\cos(\theta_2)))$$



The Lagrangian is:

$$L = T - V$$

$$L = \frac{1}{2}(m_1+m_2)l_1^2\dot{\theta}_1^2+\frac{1}{2}m_2 l_2^2\dot{\theta}_2^2+m_2 l_1 l_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)+(m_1+m_2)g l_1\cos(\theta_1)+ m_2 g l_2\cos(\theta_2)$$

example: double pendulum

For θ_1:
$\frac{\partial L}{\partial \dot{\theta}_1} = m_1 l_1^2 \dot{\theta}_1 + m_2 l_1^2 \dot{\theta}_1 + m_2 l_1 l_2 \dot{\theta}_2 \cos(\theta_1 - \theta_2)$

$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) = (m_1 + m_2) l_1^2 \ddot{\theta}_1 + m_2 l_1 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) - m_2 l_1 l_2 \dot{\theta}_2 \sin(\theta_1 - \theta_2)(\dot{\theta}_1 - \dot{\theta}_2)$

$\frac{\partial L}{\partial \theta_1} = -l_1 g (m_1 + m_2) \sin(\theta_1) - m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2)$

Thus, the differential equation for θ_1 becomes:
$(m_1 + m_2) l_1^2 \ddot{\theta}_1 + m_2 l_1 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + m_2 l_1 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + l_1 g (m_1 + m_2) \sin(\theta_1) = 0$

Divide through by l_1 and this simplifies to:
$(m_1 + m_2) l_1 \ddot{\theta}_1 + m_2 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + m_2 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + g (m_1 + m_2) \sin(\theta_1) = 0$

example: double pendulum

Similarly, for θ_2:

$$\frac{\partial L}{\partial \dot{\theta}_2} = m_2 l_2^2 \dot{\theta}_2 + m_2 l_1 l_2 \dot{\theta}_1 \cos(\theta_1 - \theta_2)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) = m_2 l_2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2 l_1 l_2 \dot{\theta}_1 \sin(\theta_1 - \theta_2)(\dot{\theta}_1 - \dot{\theta}_2)$$

$$\frac{\partial L}{\partial \theta_2} = m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) - l_2 m_2 g \sin\theta_2$$

Thus, the differential equation for θ_2 becomes:

$$m_2 l_2^2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2 l_1 l_2 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + l_2 m_2 g \sin\theta_2 = 0$$

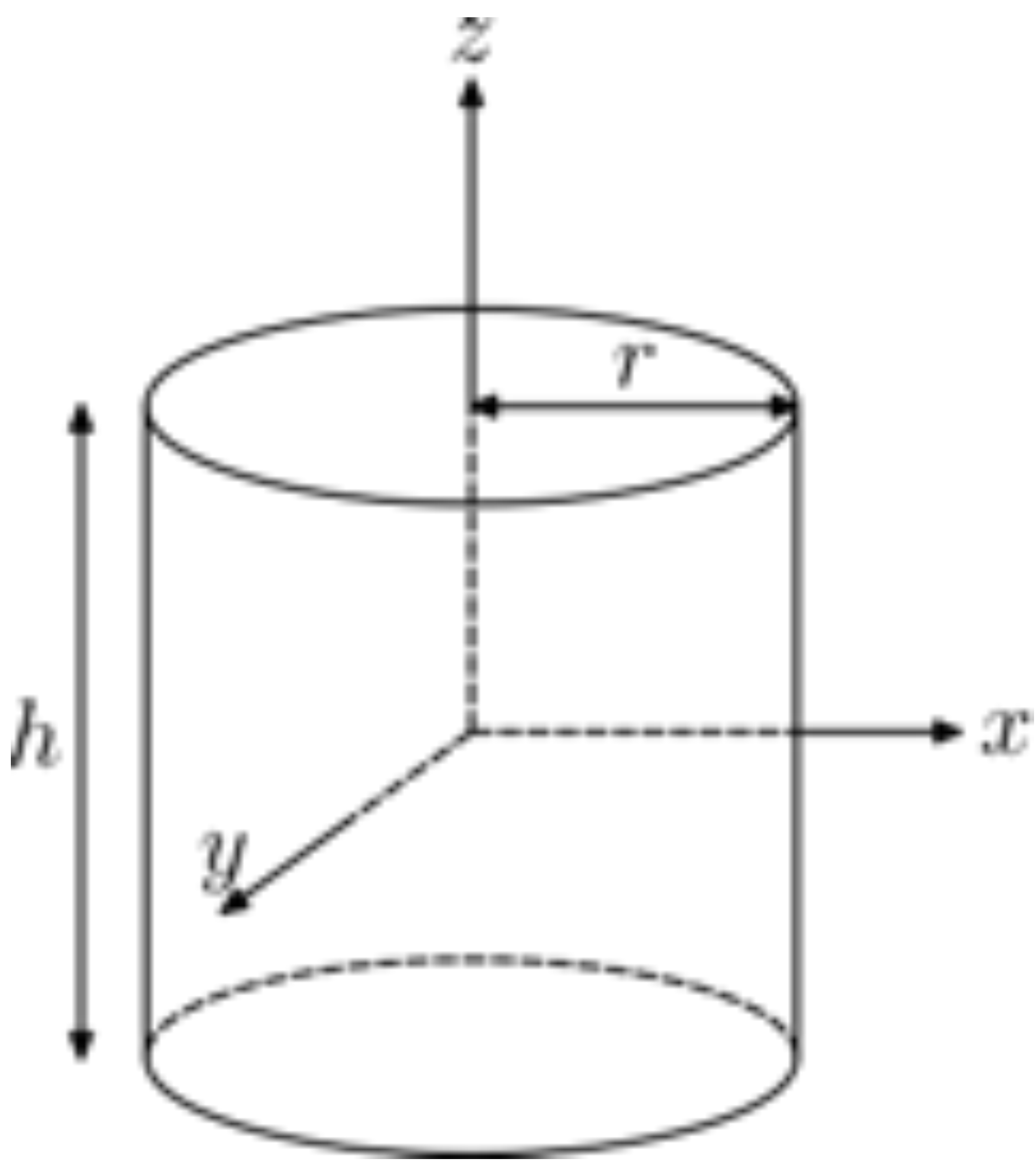
Divide through by l_2 and this simplifies to:

$$m_2 l_2 \ddot{\theta}_2 + m_2 l_1 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + m_2 g \sin\theta_2 = 0$$

So, we have developed a very complicated set of coupled equations of motion from a very simple system!

but robots are not point masses...

links could be represented by solid cylinders



moments of inertia about center:

$$I_z = \frac{mr^2}{2}$$

$$I_x = I_y = \frac{1}{12}m\left(3r^2 + h^2\right)$$

moment of inertia tensor:

$$I = \begin{bmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$$

http://en.wikipedia.org/wiki/List_of_moments_of_inertia

what do you do if the rotation is not about the center?

If the new axis of rotation is parallel to the original axis of rotation, use the parallel axis theorem:

$$I_{new} = I_{cm} + mR^2$$

I_{cm} moment of inertia of the object about an axis passing through its center of mass

m object's mass

R perpendicular distance between the two axes

and what if the new axis of rotation is not parallel to the original?

the moment of inertia of a continuous solid body rotating about a known axis is calculated by

$$I = \int_V \rho(\vec{r}) d(\vec{r}) \mathrm{d}V(\vec{r})$$

$\vec{r}$ radius vector (from origin to volume element of interest)

$\rho(\vec{r})$ object's mass density at $\vec{r}$

$d(\vec{r})$ shortest distance between a point at $\vec{r}$ and the axis of rotation

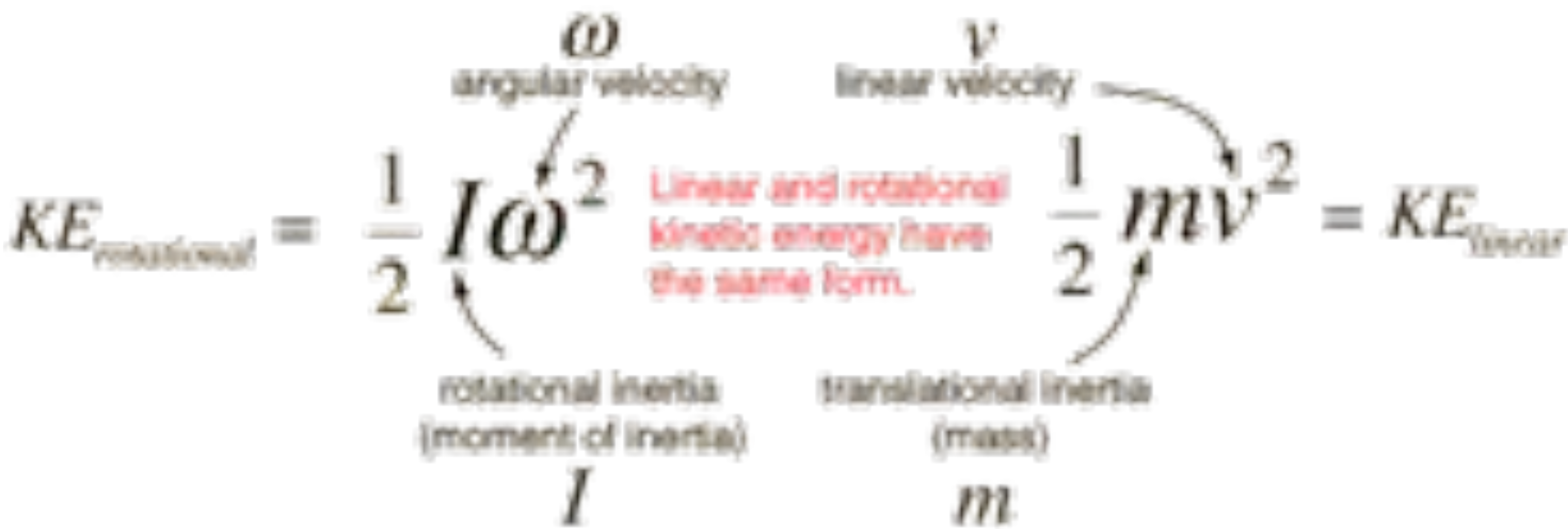
integrated over the volume V of the body

... but to reduce the complexity of this week's assignment, we will approximate the moment of inertia by assuming that we do have parallel axes

considering kinetic energy of solids

sum translational and rotational components

$$T = \frac{1}{2}mv^2 + \frac{1}{2}I\omega^2$$



http://hyperphysics.phy-astr.gsu.edu/hbase/rke.html

for robot dynamics background

class: CS223A / ME320: Introduction to Robotics

dynamics and robotics textbooks such as
John J. Craig
Introduction to Robotics: Mechanics and Control

… and many others

In Assignment 3, we will give you the dynamic equations,
but it helps to understand where they come from!

questions

- how do you think an RCM robot compares to a typical serial chain manipulator in terms of its dynamics?
- does the da Vinci have haptic feedback? (and how do the system dynamics affect this capability?)

Dynamic simulation

controller on one end, system dynamics on the other

a controller computes the desired force

e.g. $f = k_p * (x - x_d)$

desired force (in computer)

endpoint force/torque

in Assignment 3, you will simulate the effects of system dynamics

this force and externally applied loads result in robot motion

e.g., solve for x in $f = m\ddot{x} + b\dot{x}$

simulating equations of motion

goal: given an equation of motion and applied forces, what will the resulting robot motion be?

The dynamics equations we have are coupled, non-linear ODEs. They are hard (likely impossible) to solve analytically.

Instead, we solve them using numerical integration.

you can do a simple calculation yourself, i.e. integration using the trapezoidal rule, or use a handy and more accurate/robust Matlab function (Simulink is also an option)

There are a series of native Matlab functions that solve ODEs given initial conditions. Different functions are appropriate for different problem types ("stiffness", corresponding to large differences in scales) and and order of accuracy (low to high).

Solver	Problem Type	Order of Accuracy	When to Use
`ode45`	Nonstiff	Medium	Most of the time. This should be the first solver you try.
`ode23`	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
`ode113`	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
`ode15s`	Stiff	Low to medium	If `ode45` is slow because the problem is stiff.
`ode23s`	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
`ode23t`	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
`ode23tb`	Stiff	Low	If using crude error tolerances to solve stiff systems.

`ode45` is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver - in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, y(tn-1). In general, `ode45` is the best function to apply as a "first try" for most problems.

The basic syntax for these solvers is:

```
[T,Y] = solver(odefun,tspan,y0)
```

where:

`solver` is one of `ode45`, `ode23`, etc.
`odefun` is a function handle that evaluates the right side of the differential equations
`tspan` is a vector specifying the interval of integration, `[t0,tf]`
`y0` is a vector of initial conditions

Example: First order system

Let's say that we want to solve the function

$$\dot{y} + 2y = 0$$

for the initial condition $y(0) = 10$.

First, you need to create a function of the form $\dot{y} = f(t, y)$:

```
function ydot = my_function(t,y)

ydot = -2*y;  % end of subprogram
```

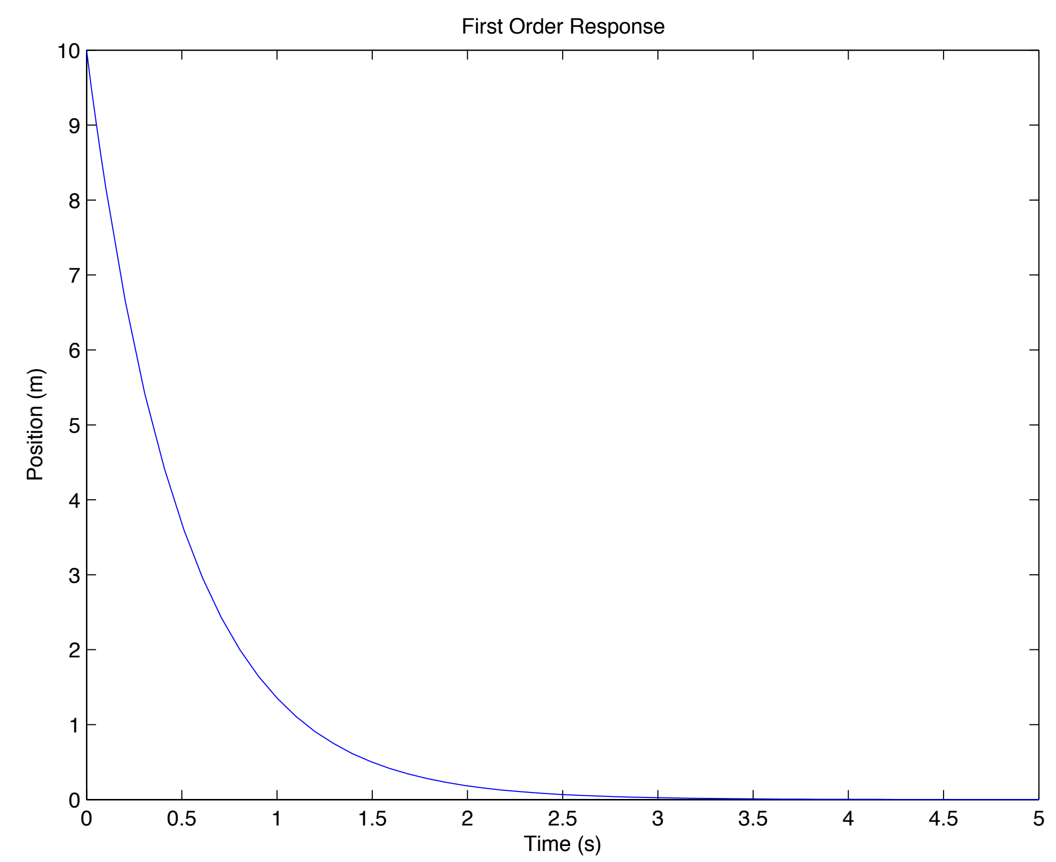
And the code to apply initial condition and solve the system is:

```
tspan = [0,5];                    %  time duration for calculation
y0 = 10;      % initial condition

% Note that the equation of motion is in a
% subprogram named my_func

[t,y] = ode45('my_func',tspan,y0);

% plot the response
figure
plot(t,y)
title('First Order Response')
xlabel('Time (s)')
ylabel('Position (m)')
```



trajectory generation

reference: Chapter 7 of Introduction to Robotics by J. J. Craig (any edition)

discussion

why would you want to generate a trajectory?

would you want to do trajectory generation in Cartesian space or joint space?

teleoperation → autonomy

user provides the desired position (trajectory) at each control loop

surgical planning specifies the start/end points and desired via points

for autonomous robots, we typically specify the trajectory based on start point, end point, via points, motion (e.g., velocity) constraints, and/or time constraints

discussion

what properties might you want in a trajectory?

Trajectory generation goal

move a manipulator from an initial pose to a final pose in a *smooth* manner

what does smooth mean?

at least C1 continuous position profile

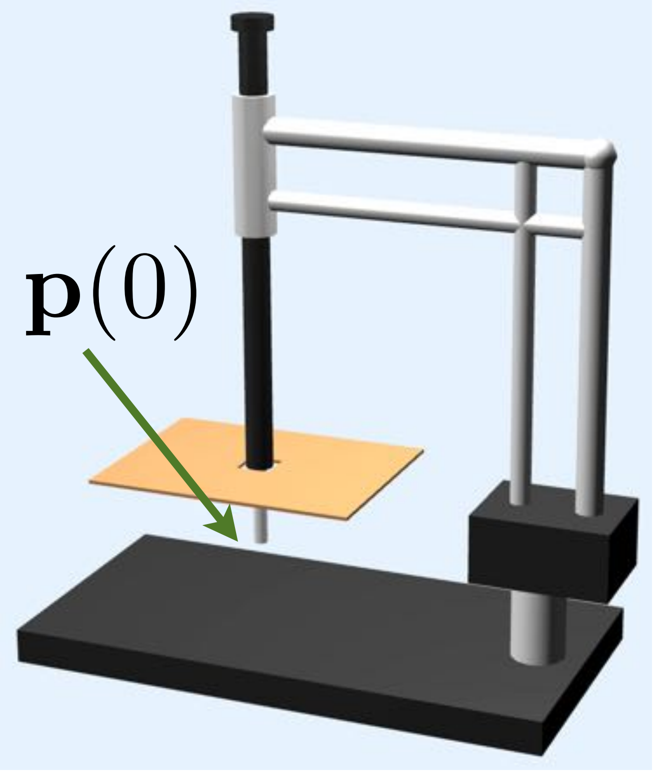
at least C0 continuous velocity profile

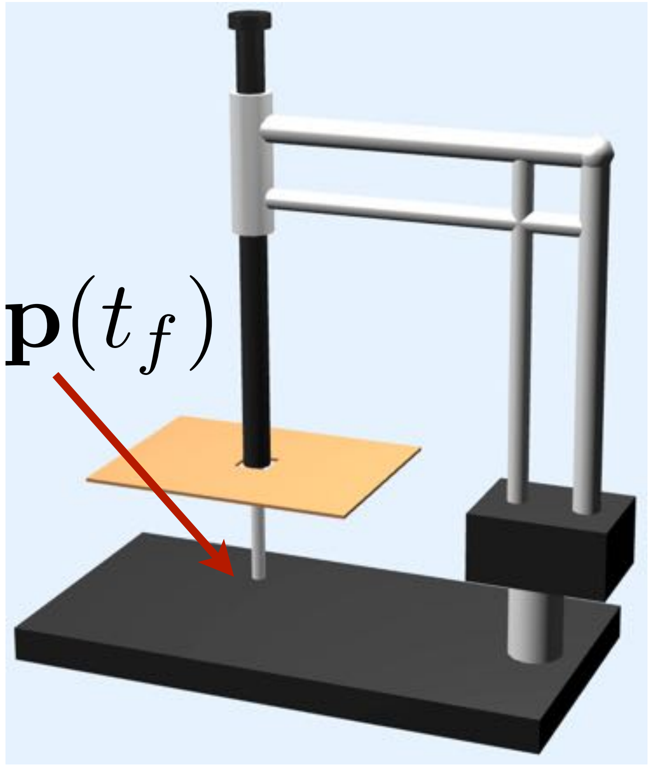
possibly continuous acceleration profile

point-to-point trajectory generation

consider the problem of moving a robot end-effector from its initial 3D pose to a final 3D pose in a certain amount of time

$$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$$





robot graphic courtesy Fidel Hernandez

a smooth path

cubic polynomial: $$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3$$

$$x(t) = a_{x0} + a_{x1} t + a_{x2} t^2 + a_{x3} t^3$$

or $$y(t) = a_{y0} + a_{y1} t + a_{y2} t^2 + a_{y3} t^3$$

$$z(t) = a_{z0} + a_{z1} t + a_{z2} t^2 + a_{z3} t^3$$

derivatives: $$\dot{\mathbf{p}}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = \mathbf{a}_1 + 2\mathbf{a}_2 t + 3\mathbf{a}_3 t^2$$

$$\ddot{\mathbf{p}}(t) = \begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \\ \ddot{z}(t) \end{bmatrix} = 2\mathbf{a}_2 + 6\mathbf{a}_3 t$$

constraints

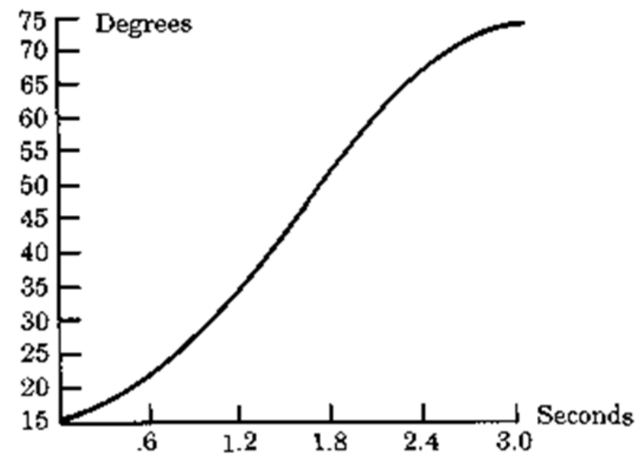
position and velocity at initial and final times:

$$\mathbf{p}(0) = \mathbf{p}_0$$
$$\mathbf{p}(t_f) = \mathbf{p}_f$$
$$\dot{\mathbf{p}}(0) = 0$$
$$\dot{\mathbf{p}}(t_f) = 0$$

with four equations and four unknowns, you can solve for the coefficients $\mathbf{a}$ in terms of $\mathbf{p}_0$ and $\mathbf{p}_f$ and the time t_f

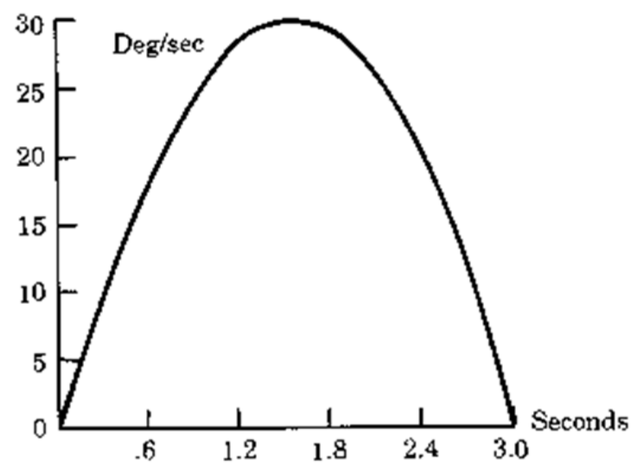
you can pick the time t_f based on how quickly you want to move between to the two points, or based on a maximum velocity constraint

example trajectory



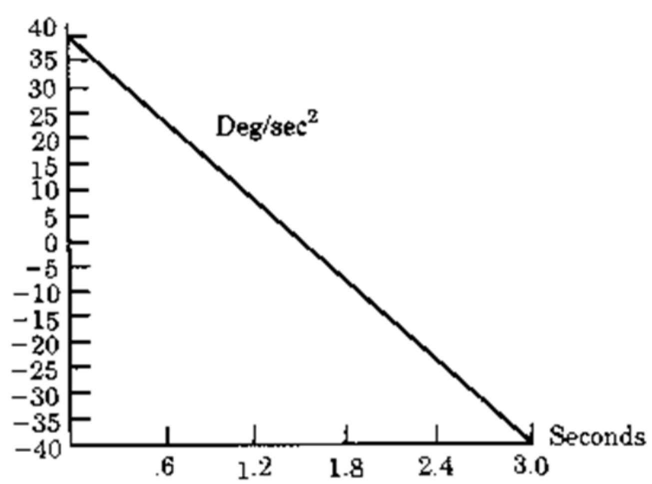
Position

position
cubic polynomial
C1 continuous



Velocity

velocity
quadratic polynomial
C0 continuous



Acceleration

acceleration
linear polynomial
not continuous (at t = 0)

now what?

now that you have created a trajectory
(i.e., you know the desired position
of the robot at each time step),
you can control the robot to follow
this trajectory

previously, we generated this trajectory
using the “master” of the teleoperator

discussion

why is a smooth trajectory important?

why might you want a *smoother* trajectory?

how could you compute this?

what do the produced trajectories look like spatially? what if you design in joint space instead of Cartesian space?

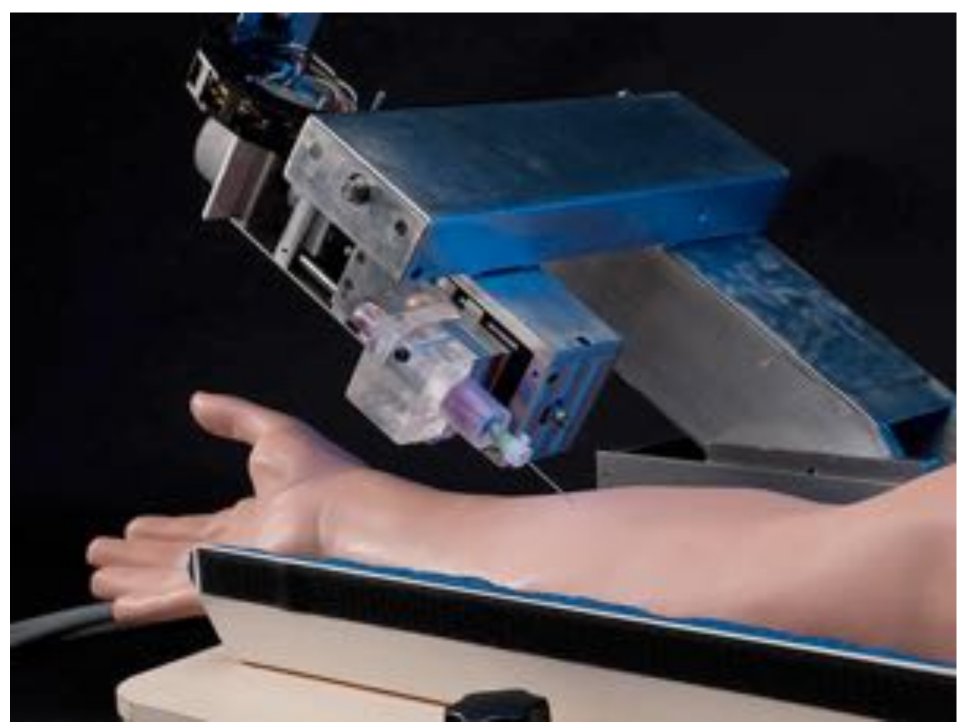
how might *via points* be useful?

how would you generate trajectories for them?

discussion

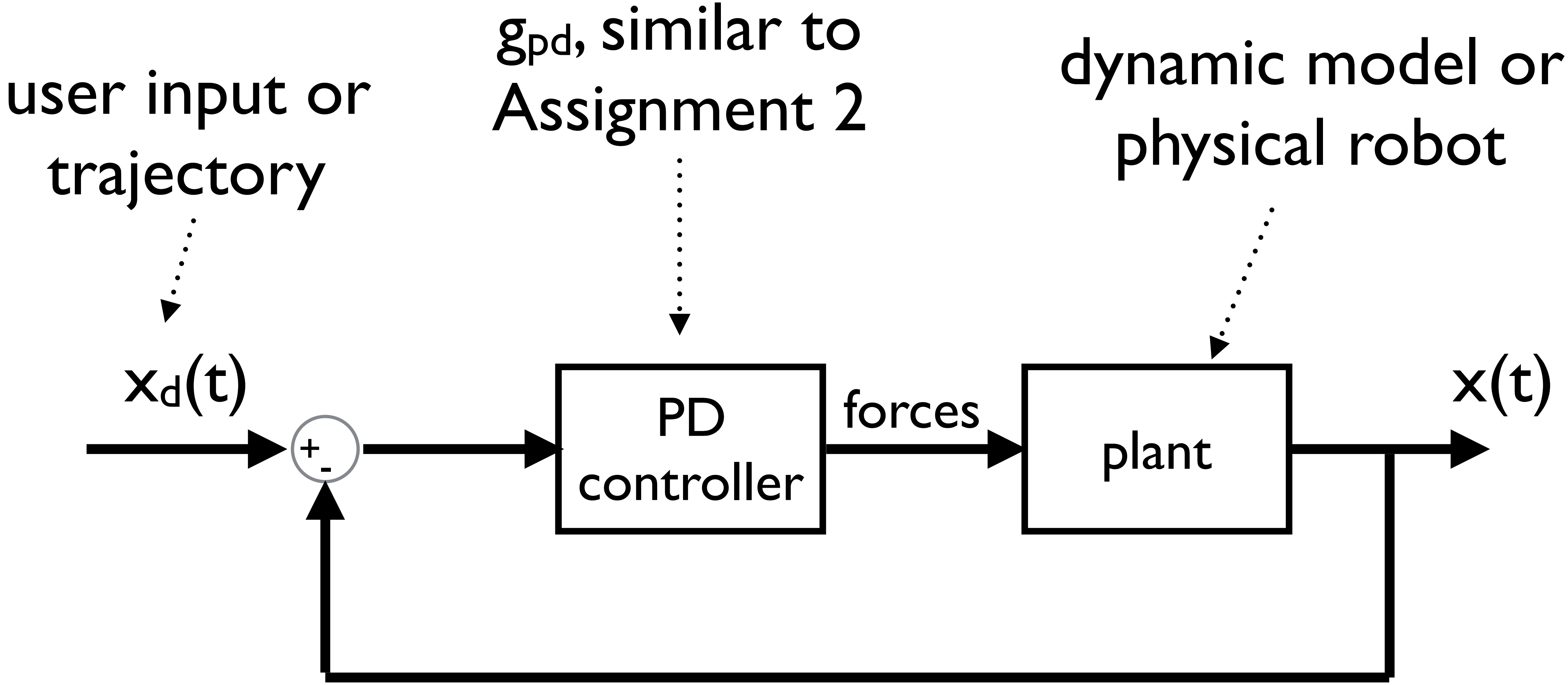
how would you compute t_f based on a defined maximum velocity?

what kind of trajectory would you want for a robot that inserts a needle into solid tissue?



bloodbot (Imperial College of London)

Overview



$$x'(t) = f(x(t), g_{pd}(x(t), x_d(t)))$$

the ODE we solve numerically

Assignment 3

Problem 0: Commentary on seminar

Problem 1: Read papers, answer questions

Problem 2: Modeling and simulation of medical robot dynamics

Problem 3: Effects of dynamics on robot control

To be posted tomorrow, due Thursday, Jan. 31 at 4 pm

FRIDAY'S SEMINAR IS AT **8:30 am**! (in 320-105)