Module 3 PD Control for Second-Order-Systems

(Review) Newton's Laws; Damped and Undamped
Video • 10 min

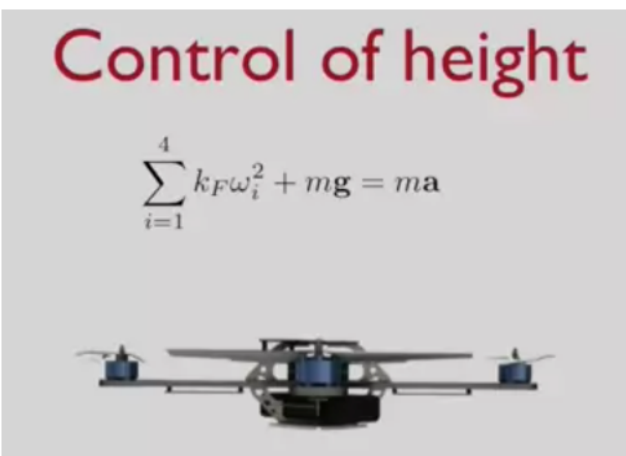
(Review) PD Control for a Point Particle in Space
Video • 5 min

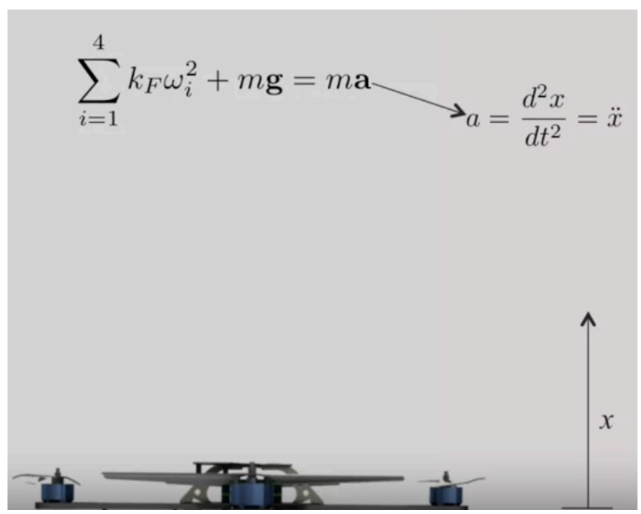
A2.1 PD Control for Second-Order Systems
Video • 6 min

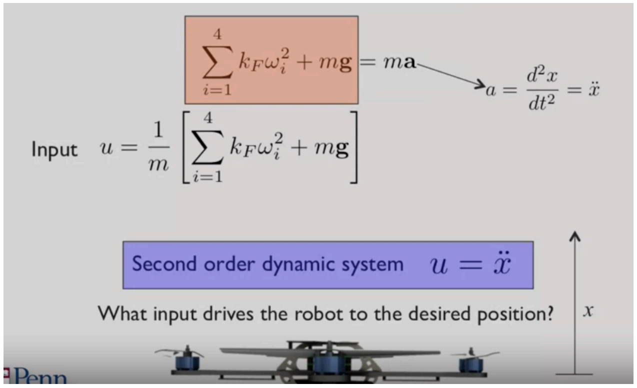
(Review) Infinitesimal Kinematics; RR Arm
Video • 3 min

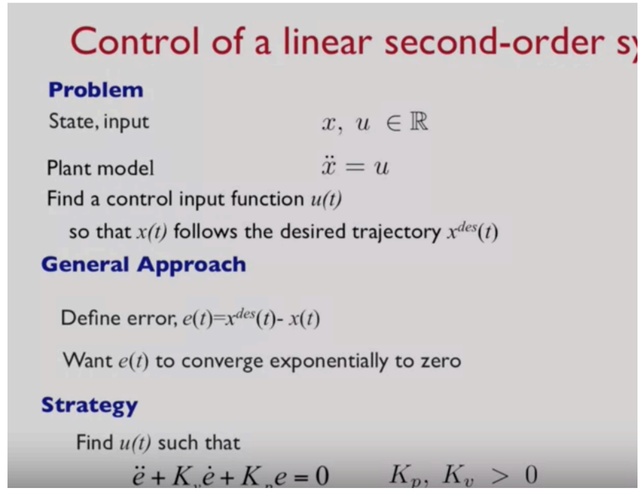
A2.2 PD Tracking
Due, Oct 25, 11:59 PM PDT Quiz • 2 questions • Grade: --

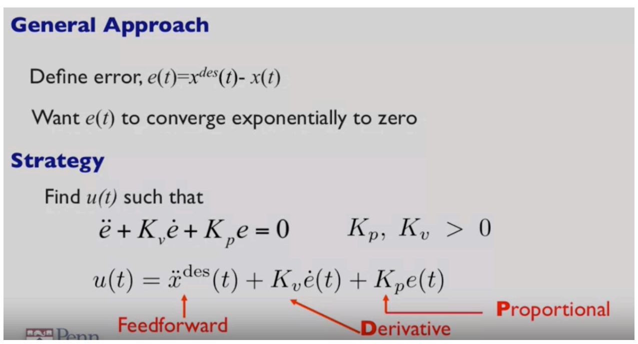
(Review) PD Control for a Point Particle in Space



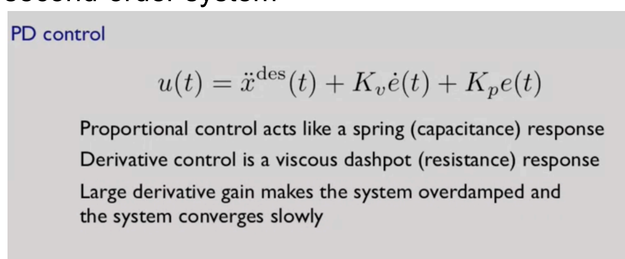








Control for Trajectory tracking in a simple second-order system

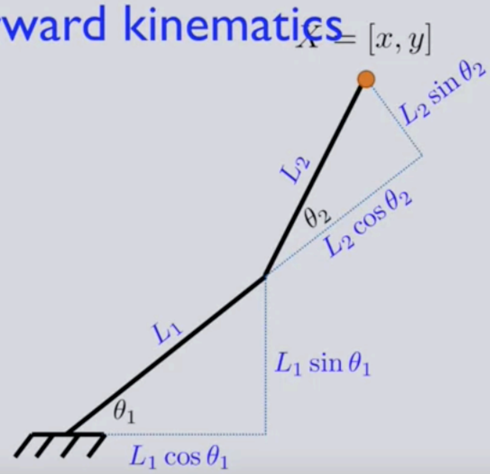


[Infinitesimal Kinematics; RR Arm]()

Forward and inverse kinematics

- Interested in the relationship between the position of the end effector, X , and the joint angles, θ
- Forward kinematics: joint angles known, determine position of end-effector
- Typically joint angles are known as they are instrumented in the machines, desired position of the end-effector can be computed
- Inverse kinematics: position of the end effector known, determine joint angles

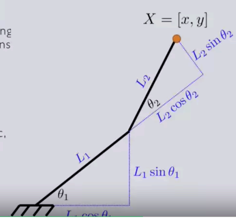
Forward kinematics



Inverse kinematics

- Generally this results in solving system of non-linear equations
- Solve using trigonometric identities
- Simpler cases can be done symbolically in MATLAB
- Make desired values symbolic, then use solve command on system of equations

$$\theta = f^{-1} X$$



Infinitesimal kinematics

- Jacobian (matrix of partial derivatives)
- Velocity implications

$$J = \frac{\partial X}{\partial \theta}\frac{\partial t}{\partial t} = \frac{\partial X}{\partial t}\frac{\partial t}{\partial \theta}$$

$$\frac{\partial X}{\partial t} = J\frac{\partial \theta}{\partial t}$$

Infinitesimal kinematics

- Jacobian (matrix of partial derivatives)
- Velocity implications
- Torque and Force implications
- Effective mechanical advantage

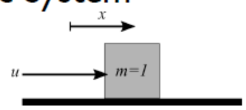
$$\tau = J^T F$$

$$F = J^{-T} \tau$$

PD control for second order systems

MIP track, week 2

Double integrator: a Simple Dynamic System

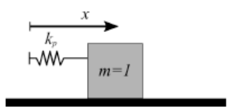


- Newton's law: $\ddot{x} = u$
- Think about *u* as "input", *x* as "output"

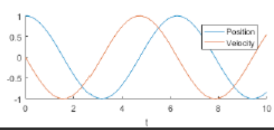
- Assume we want to send $x \rightarrow 0$

Virtual Spring for Control

- Idea: attach a spring between the block and the desired position
- Hooke's law $u_p(x) := k_p(x_{\text{des}} - x)$

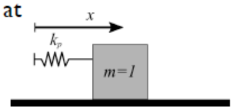


- Try using MATLAB to integrate $\ddot{x} = u_p(x)$
- Change stiffness to approach goal faster
- Overshoot



Dissipation

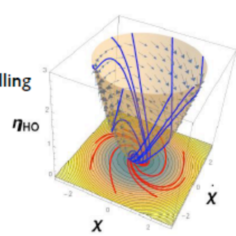
- Need to remove energy to stop at goal



- Viscous friction $u_d(x, \dot{x}) := -k_d\dot{x}$
- Add to previous input $\ddot{x} = u_p(x) + u_d(x, \dot{x})$
- Try to tune gains to approach without overshoot

Artificial energy landscape (total energy is a Lyapunov function)

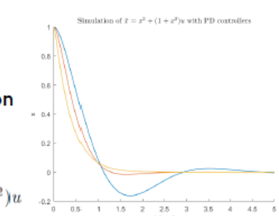
- Can *prove* that the PD controller is stabilizing
- Consider "total energy" $\eta(x, \dot{x}) = \frac{1}{2}k_p(x - x_{\text{des}})^2 + \frac{1}{2}\dot{x}^2$
- Take the time derivative, substituting $\ddot{x}$
- Total energy *monotonically decreases* (rolling downhill)
- Have your MATLAB simulation plot $\eta(x(t), \dot{x}(t))$
- PD controller creates this artificial hill



Where is PD control applicable?

- Nonlinear plant: $\ddot{x} = f(x, \dot{x}) + g(x, \dot{x})u$
- g invertible: set $u := g(x, \dot{x})^{-1}(-f(x, \dot{x}) + v)$
- Back to $\ddot{x} = v$
- "Feedback linearization" / "inverse dynamics"
- Implementation difficulties
- Recall hyperbolic approximation (Mobility 1.2)
- PD on nonlinear plants: e.g.

$$\ddot{x} = f(x, \dot{x}) + g(x, \dot{x})u := x^3 + (1 + x^2)u$$

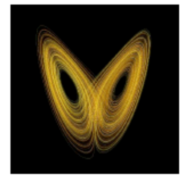


Module 2 ODE

Using MATLAB for Dynamic Simulations

MIP track, week 1

Dynamical Systems as ODE's

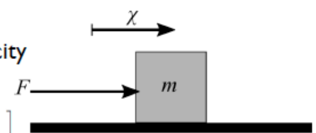


http://math.case.edu/files/2014/01/image3031.jpg

$$F\left(x, y, y', \cdots y^{(n-1)}\right) = y^{(n)}$$

- Order of an ODE is the highest order derivative that appears
- Mechanical systems are usually second order
- Recall Newton's second law $m\ddot{\chi} = F$

State-space for Second Order Systems



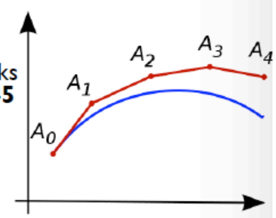
- Inertia means need velocity
- Define vector equation

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} := \begin{bmatrix} \chi \\ \dot{\chi} \end{bmatrix} \implies \dot{x} = \begin{bmatrix} x_2 \\ F/m \end{bmatrix}$$

- x is called the state
- First order
- MATLAB can integrate

Numerical ODE Integration

- Consider $\dot{x} = \alpha$
- Could use fixed timestep, dt
- Set $x(t_{k+1}) = x(t_k) + \alpha \cdot dt$
- If *α(t)* is not fixed, the results will be inaccurate
- MATLAB estimates how the right side is changing and picks the best timestep dt in **ode45**
- Only works when right hand side is *smooth*



Example: Harmonic Oscillator

- Consider $\ddot{x} = -x$
- Initial condition $x(0) = 1, \dot{x}(0) = 0$

```
function ode_example()
 X0 = [1,0];
 tspan=[0,10];
 [t, X] = ode45(@shosc, tspan, X0);
 clf
 hold all
 plot(t, X(:,1))
 plot(t, X(:,2))
 hold off
end

function Xd = shosc(t, X)
 x = X(1);
 xd = X(2);
 Xd = [xd; -x];
end
```