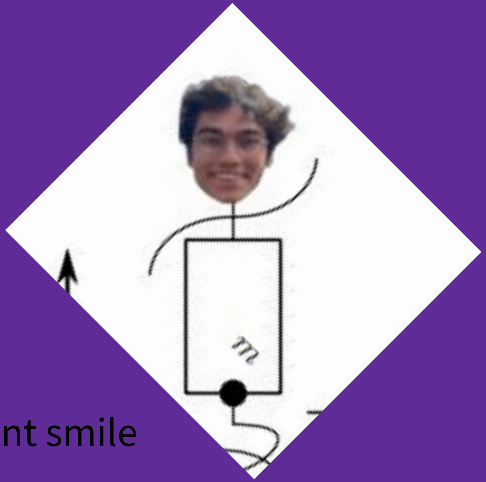
EECS/MechE/BioE C106A: Midterm 2 Review Session

The return of Prof. Tarun Amarnath!



Look at Sunay's brilliant smile

All the Past Content…

Rigid Body Transformations

- Length and orientation preserving
- Represent a movement or a change in coordinate frame
- Rotations, translations, or both (screw motion)

$$g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

Homogeneous Transformation Matrices

- Compact representation
- Both rotation and translation included
- Can stack and invert

$$g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \qquad g^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix}$$

Exponential Coordinates

- **Goal:** Create rotation and homogeneous transformation matrices as a *function of time*
- Comes from solving a differential equation
- We only need information about **how** the object moves (time is a parameter that's plugged in)

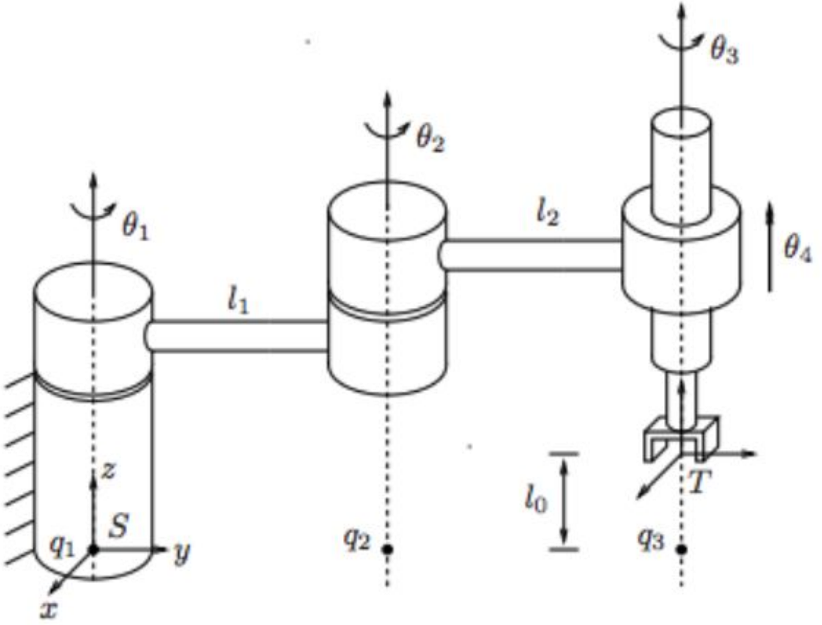
Twists
Axis of rotation

$g(t) = e^{\hat{\xi} t}$ $\qquad$ $R(t) = e^{\hat{\omega} t}$

Forward Kinematics

- **Goal:** Find the location of the tool after a multi-joint robot arm has moved around
- Compose exp. coords

$$g_{st}(\theta_1 \dots \theta_n) = e^{\hat{\xi}_1 \theta_1} \cdots e^{\hat{\xi}_n \theta_n} \cdot g_{st}(0)$$



Inverse Kinematics

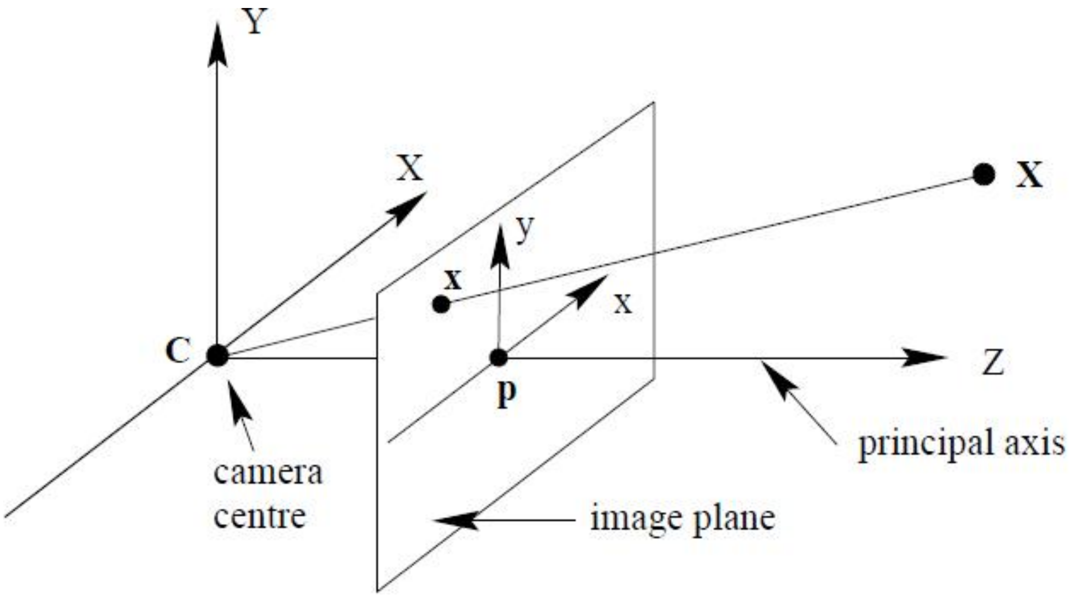
→ Given some final position $g_{st}(t)$

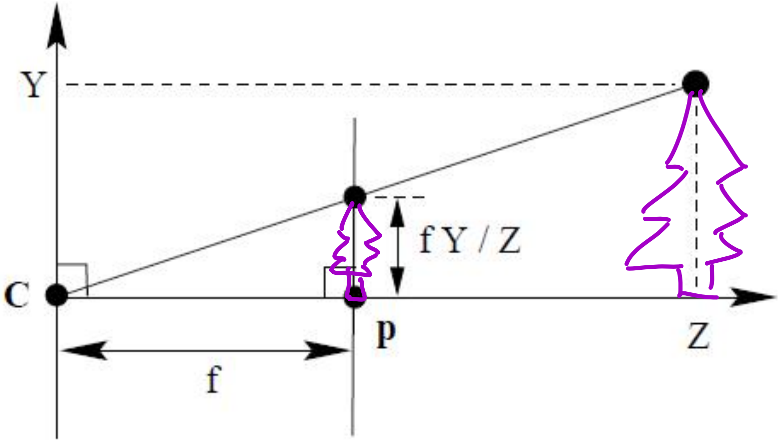
→ Find θ

- How do we move our robot's joints to reach a desired configuration?
- Use Paden-Kahan subproblems along with tricks (reduce problem down to simpler parts)

Computer Vision

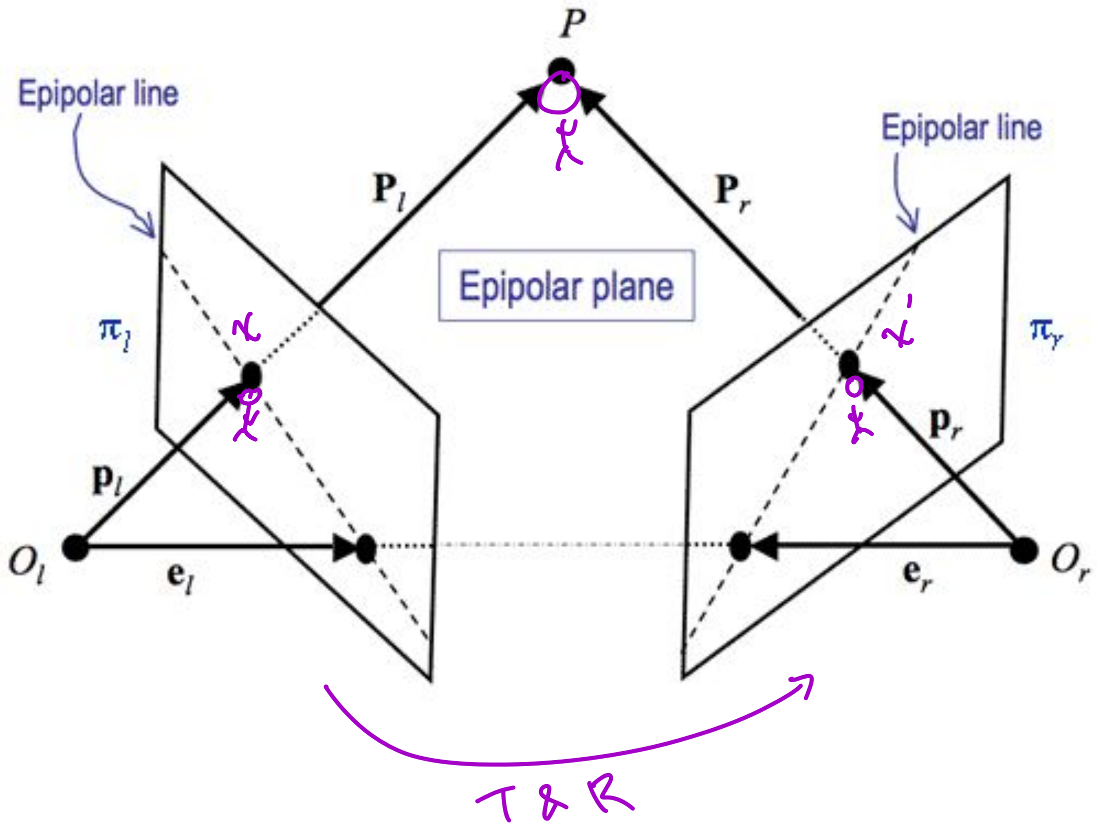
Pinhole Camera Model





$$(X, Y, Z)^T \longrightarrow (\frac{fX}{Z}, \frac{fY}{Z})^T \qquad K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

Two-View Geometry



$$(x')^T E x = 0$$

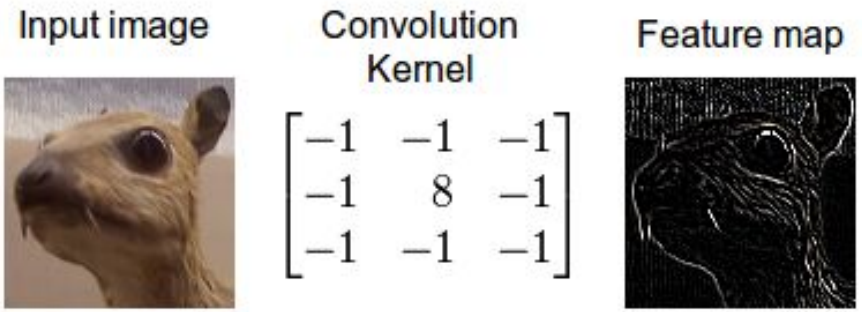
point location in img 2 in img coords

$$= \hat{T} R$$

- can determine depth info

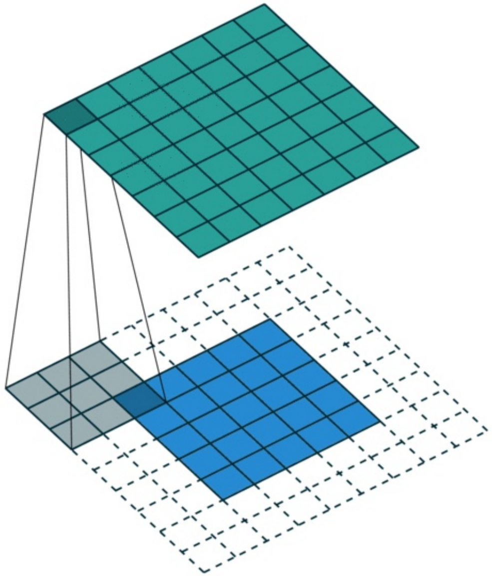
Convolutions

- Slide a kernel over some image
- Understand some information about the picture



Original image

Gaussian Blur filter applied



Homography

- Apply some kind of affine transformation to an image
- Change perspectives - for example, can straighten a picture
- Need at least 4 pairs of points to do this

Original sign

Sign points

Straightened sign

Velocities

What do we mean by them?

- Velocity in general is the rate of change with respect to some reference frame
- With robots, use a **stationary frame**
- Calculate the velocity of some point attached to the end effector wrt to the base

Some Important Considerations

- Spatial & body velocities - **just a coordinate shift**, tells us which coordinate system to use

- Spatial and body velocities are **twists** $\rightarrow V^s = \begin{bmatrix} v \\ \omega \end{bmatrix}$
- Generic expressions for any point
- Can apply them to a specific point to determine that point's velocity

$$v_q = \hat{V}^s_{ab} \cdot p_a$$

Spatial Velocity

- Express our point in the **spatial frame**

$$q_a(t) = g_{ab} \cdot q_b$$

$\frac{d}{dt}$

$$\dot{q}_a(t) = \dot{g}_{ab} \cdot q_b$$

switch frames

$$v_{q_a}(t) = \underbrace{\dot{g}_{ab} \cdot g_{ab}^{-1}}_{\hat{V}^s_{ab}} \cdot q_a$$

$$\widehat{V}^s_{ab} := \dot{g}_{ab} g_{ab}^{-1} = \begin{bmatrix} \dot{R}_{ab} R_{ab}^T & -\dot{R}_{ab} R_{ab}^T p_{ab} + \dot{p}_{ab} \\ 0 & 0 \end{bmatrix} \qquad V^s_{ab} = \begin{bmatrix} v^s_{ab} \\ \omega^s_{ab} \end{bmatrix} = \begin{bmatrix} -\dot{R}_{ab} R_{ab}^T p_{ab} + \dot{p}_{ab} \\ (\dot{R}_{ab} R_{ab}^T)^\vee \end{bmatrix}$$

Body Velocity

- Point is expressed in terms of the **body frame**

$$v_{q_b}(t) = \underbrace{g_{ab}^{-1}(t) \cdot \dot{g}_{ab}(t)}_{\hat{V}_{ab}^{b}} \cdot q_b$$

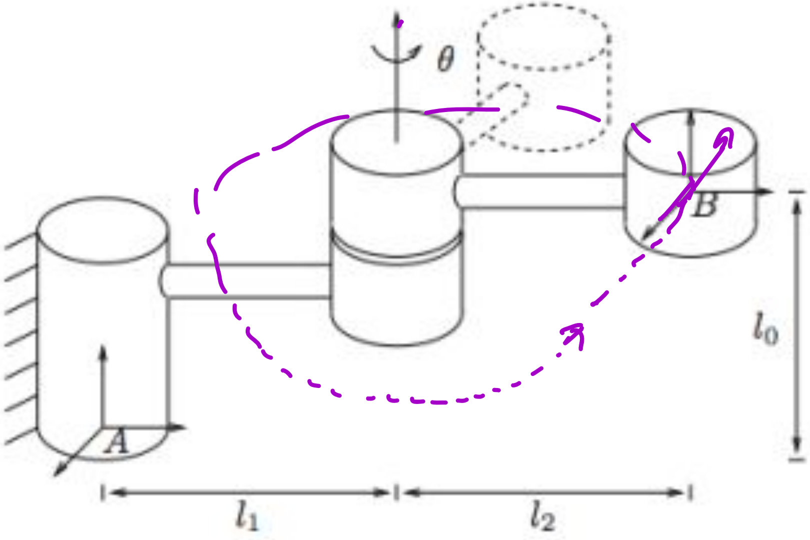
$$\widehat{V}_{ab}^{b} := g_{ab}^{-1}(t)\dot{g}_{ab} = \begin{bmatrix} R_{ab}^T \dot{R}_{ab} & R_{ab}^T \dot{p}_{ab} \\ 0 & 0 \end{bmatrix} \qquad V_{ab}^{b} = \begin{bmatrix} v_{ab}^{b} \\ \omega_{ab}^{b} \end{bmatrix} = \begin{bmatrix} R_{ab}^T \dot{p}_{ab} \\ (R_{ab}^T \dot{R}_{ab})^{\vee} \end{bmatrix}$$

Interpreting Velocities as Twists

- Can break them apart into *v* and *w* components
- Calculate each one separately

Quantity	Interpretation
ω^s_{ab}	Angular velocity of B wrt frame A, viewed from A.
v^s_{ab}	Velocity of a (possible imaginary) point attached to B traveling through the origin of A wrt A, viewed from A.
ω^b_{ab}	Angular velocity of B wrt frame A, viewed from B.
v^b_{ab}	Velocity of origin of B wrt frame A, viewed from B.

Review Example



$v = -\omega \times q$

↳ using velocities instead of direction

$$\mathcal{V}^s = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} l_1 \dot{\theta} \\ 0 \\ 0 \\ 0 \\ 0 \\ \dot{\theta} \end{bmatrix}$$

linear speed of origin attached to body frame

angular speed

$$\mathcal{V}^b = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -l_2 \dot{\theta} \\ 0 \\ 0 \\ 0 \\ 0 \\ \dot{\theta} \end{bmatrix}$$

linear speed of body (in body frame)

angular speed

$v_q = [\mathcal{V}] \cdot q$

Adjoints

What are they?

- Like a g matrix for twists!
- Change coordinate frames if we have a twist
- Because velocities are also twists, we can use adjoints to switch between spatial and body velocities

$$\widehat{\xi'} = g\widehat{\xi}g^{-1}$$

$$V_{ab}^{s} = Ad_{g_{ab}} \cdot V_{ab}^{b}$$

$$\xi' = Ad_g \xi$$

Formulas

$$\underbrace{\begin{bmatrix} R_{ab} & \widehat{p}_{ab} R_{ab} \\ 0 & R_{ab} \end{bmatrix}}_{:= Ad_{g_{ab}}}$$

$$V_{ac}^{s} = V_{ab}^{s} + Ad_{g_{ab}} V_{bc}^{s}$$

$$V_{ac}^{b} = Ad_{g_{bc}^{-1}} V_{ab}^{b} + V_{bc}^{b}$$

$$Ad_{g_{ab}}^{-1} = \begin{bmatrix} R_{ab}^{T} & -R_{ab}^{T}\widehat{p} \\ 0 & R_{ab}^{T} \end{bmatrix}$$

$Ad_{g_{ab}} \cdot Ad_{g_{bc}}$

$= Ad_{(g_{ab} g_{bc})}$

Jacobians and Singularities

Motivation

- We want to get the velocity of our **end effector**
- However, our **sensors** give us the **velocities of our links**
- Jacobian allows us to go from **link velocities → end effector velocity**

$$V_{st}^{s} = J_{st}^{s}(\theta)\dot{\theta}$$

Spatial Jacobian

- Gets us to the spatial velocity
- Columns of the Jacobian:
  - Twists of each of the links of the robot
  - In their *current* positions (i.e. not at 0 position, unlike FK)
  - Expressed in spatial coordinates
- Column represents derivative of end effector position wrt each of the links

Formulas

$$J^s_{st}(\theta) = \left[\left(\frac{\partial g_{st}}{\partial \theta_1}\right)^\vee \quad \cdots \quad \left(\frac{\partial g_{st}}{\partial \theta_n}\right)^\vee \right]$$

Partial derivatives

$$= \left[\xi_1 \quad \xi_2' \quad \cdots \quad \xi_n'\right]$$

Twists

$$\xi_i' = Ad_{\left(e^{\widehat{\xi}_1 \theta_1} \ldots e^{\widehat{\xi}_{i-1} \theta_{i-1}}\right)} \xi_i$$

$$v_{q_s} = \widehat{V}^s_{st} q_s = (J^s_{st}(\theta)\dot{\theta})^\wedge q_s$$

Body Jacobian

- Analogous to spatial Jacobian
- Gets us the body velocity, instead of the spatial velocity
- Each of the twists are represented in the body frame instead

$$J_{st}^b(\theta) = \begin{bmatrix} \xi_1^\dagger & \xi_2^\dagger & \ldots & \xi_n^\dagger \end{bmatrix}$$

$$\xi_i^\dagger = Ad^{-1}_{\left(e^{\widehat{\xi}_{i+1}\theta_{i+1}} \ldots e^{\widehat{\xi}_n\theta_n} g_{st}(0)\right)} \xi_i$$

$$v_{q_b} = \widehat{V}_{st}^b q_b = (J_{st}^b(\theta)\dot{\theta})^\wedge q_b$$

Conversion

- Jacobians are composed of twists
- Can use the adjoint to move between them!
  - Adjoint is invertible, can go the other way as well

$$J_{st}^{s}(\theta) = Ad_{g_{st}(\theta)} J_{st}^{b}(\theta)$$

Finding the Jacobian

- Can find the twists making up the columns directly by finding and applying adjoint transformation

- Alternatively, we can calculate the new positions of each of the v and w components that make up the twists

Singularities

$$V_{st}^{s} = \underbrace{J_{st}^{s}(\theta)\dot{\theta}}$$

- Jacobian drops in rank
- We can't reach all of the velocities that we *should* be able to no matter what we set each of our link velocities to
- This is a **singular configuration**
- Would prefer to avoid being in it or near it
  - Can't achieve instantaneous motion in certain directions
  - Could require significant amounts of force in certain directions around that area

Dynamics

Forces!

- In real life, we’re trying to control our robot by applying some force to its joints
- Need to get the **dynamics** of our system
- The forces in each direction so that we know exactly what to apply to achieve our trajectory

Use Energy!

- Forces can be difficult
  - When there are multiple reference frames, particularly rotating ones, in play
  - End up with many complicated terms
  - Sometimes have several “imaginary” forces to balance equations’
- Energy is nice!
  - Scalars
  - Only depends on current state of the object
  - Invariant to coordinate frame - choose any one

Method

1. Choose state $\rightarrow q$
2. Kinetic energy
3. Potential energy
4. Lagrangian
5. Equations of motion (convert to forces)
6. Separate into matrices

State

- Depends on the problem at hand
- Choose minimal representation needed *or* the representation that makes it easiest to determine what forces to apply
- Usually p, theta, or something similar

Kinetic Energy

- Translational

$$\frac{1}{2} m \cdot v^2 \quad v \to \dot{q}$$

- Rotational

$$\frac{1}{2} \cdot I \cdot \dot{\theta}^2$$

6D twist

$$\rightarrow T = \frac{1}{2} {V_i^b}^T M_i \cdot V_i^b$$

Identity

$$M_i = \begin{bmatrix} m \cdot I_3 & 0 \\ 0 & I \end{bmatrix}$$

Inertia matrix → 3x3

Potential Energy

- Gravitational

$$mg\underbrace{h}$$

height from our 0 position

- Spring

$$\frac{1}{2}kx^2$$

Lagrangian

* No matrices or vectors

→ scalar

$$L = T - V = \sum T_i - \sum V_i$$

Equations of Motion

$$\Upsilon = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q}$$

End up with vectors

- Vector w/ the same dimension as state
- Amt. of force applied on each state variable

Separation

Constant forces

$$\Upsilon = \underbrace{M(q)}_{\text{Mass/inertia matrix}}\ddot{q} + \underbrace{C(q,\dot{q})\dot{q}}_{\text{Coriolis}} + G(q)$$

Imaginary forces from spinning reference frames

Control

Trajectories

- Define how we want our robot to move
- Precomputed

$q, \dot{q}$

Realistic Motion

- Apply some control input (u) to follow trajectory
  - Feedforward control
- Friction, inefficiencies, and other real world issues create problems
- Adjust input to fix errors
  - Feedback

Systems

- Equations used to represent relationships between state variables
- Also incorporate control input
- Generated with knowledge of dynamics

$$\dot{x} = f(x) + g(x) \cdot u$$

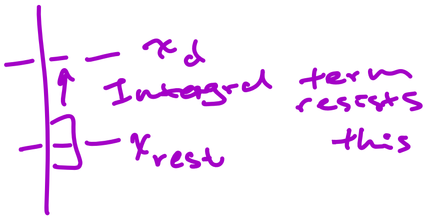
PID Control

- Used to error correct and can follow trajectory to some small extent
- Model-free control - only need to know error, not system equations

$$u(t) = \underbrace{K_p e(t)}_{\text{Proportional}} + \underbrace{K_i \int_0^t e(\tau) d\tau}_{\text{Integral}} + \underbrace{K_d \dot{e}(t)}_{\text{Derivative}}$$

The Terms

- Proportional
  - Workhorse
  - Applies input that pulls state towards desired trajectory
- Derivative
  - Dampens proportional response
  - Prevents oscillation and overcorrection
  - Allows for convergence
- Integral
  - Corrects steady-state error because of constant forces like g



↳ supplies force to stay at 0 error

Model-Based Control

- Uses system dynamics
- Much better inputs to control state
  - PIDs might estimate error with position
  - But input might be acceleration - not ideal
- Feedforward control - determine beforehand what the input should be

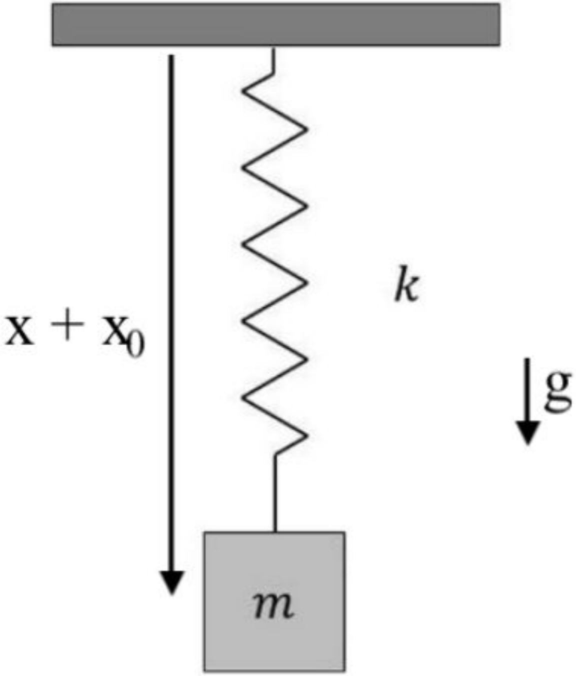
Feedback

$u = u_{ff} + u_{fb}$

Feedforward

PIDs, some error correction

An Example



$$m \cdot \ddot{x} = m \cdot g - kx + F_{input}$$

↓

Solve for input force achieve $\ddot{x}_d$

↓

Add a feedback term using PID control

$$m\ddot{x}_d = m \cdot g - kx + F_{input}$$

$$m\ddot{x}_d - mg + kx = F_{input}$$

$$u = F_{input} + K_p \cdot e + K_d \dot{e} + K_i \int e \, dt$$

Feedback Linearization

→ Feedforward control even though it's called feedback

- Setup input in such a way that we can directly plug in our trajectory as a control input

$$\dot{x} = f(x) + g(x) \cdot u$$

$$u = -g^{-1}(x) \cdot f(x) + g^{-1}(x) \cdot \dot{x}_d$$

↳ Input is a function of desired trajectory

$$\dot{x} = \dot{x}_d$$

Lab

GOAL: Prep for job interviews!
Work in industry!

- Make sure you're familiar with the basic setup operations!
- Sourcing, making, etc.
- Nodes, topics, publishers, subscribers
- Creating packages, running programs
- Types of communication protocols — server / client
- ROS parameter server
- Bashrc
- Work done in labs (planning, tracking, mapping, etc.)