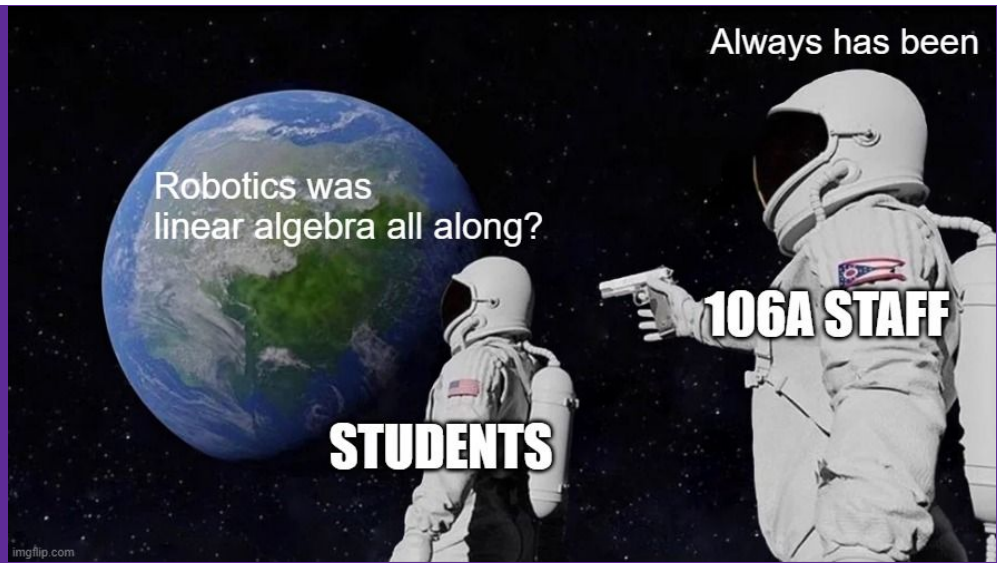
EECS/MechE/BioE C106A: Midterm 1 Review Session

Presented by Tarun Amarnath



Rigid Body Transformations

Rigid Body Transformations

- The forms of movement we discuss in this class
- 2 important qualities:
  - Length preserving

$$\|g(p) - g(q)\| = \|p - q\|$$

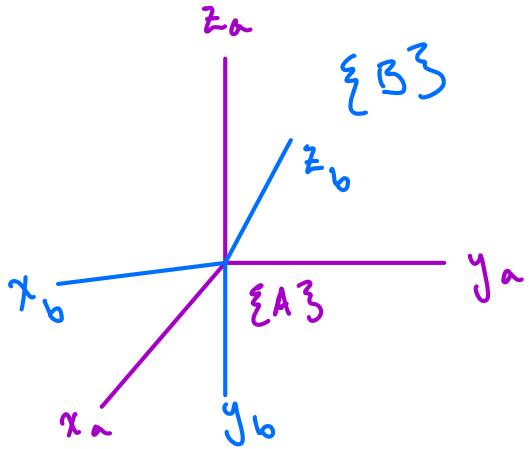
  - Orientation preserving

$$g(v \times w) = g(v) \times g(w)$$

Rotations

1st Rigid Body Transform: Rotations

- Let's say we have a world frame {A}
- There's a body attached to it {B} with its own orientation
- Rotation matrix represents the axes of {B} in terms of the axes of {A}



$$q_A = R_{ab} \cdot q_B$$

- Express point B in the A coordinate frame
- Transform point B by R

Rotations about World Axes

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} = e^{\hat{\boldsymbol{x}}\theta}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} = e^{\hat{\boldsymbol{y}}\theta}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = e^{\hat{\boldsymbol{z}}\theta}$$

Rodrigues' Formula

- Rotation about some **generic axis** *w* (not necessarily a world axis) by

$$R(w, \theta) = e^{\hat{w}\theta} = I + \hat{w}\sin(\theta) + \hat{w}^2(1 - \cos\theta)$$

- Skew-symmetric matrix:

$$\hat{w} = \begin{bmatrix} 0 & -w_2 & w_1 \\ w_2 & 0 & -w_0 \\ -w_1 & w_0 & 0 \end{bmatrix} \quad \in \mathfrak{so}(3)$$

Intrinsic (Euler) vs. Extrinsic (RPY) Rotations

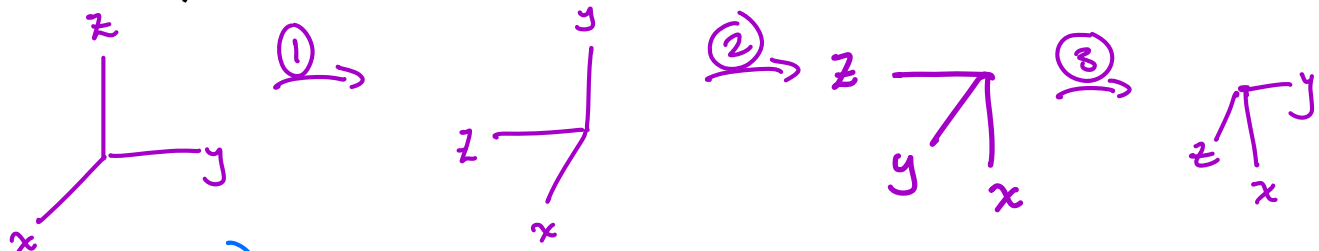
- Intrinsic - rotate based on axes of **body** frame
  - Write this out from right to left
- Extrinsic - rotate based on **world** axes
  - Write this out from left to right

- **Equivalent**, depending on how you read a composition

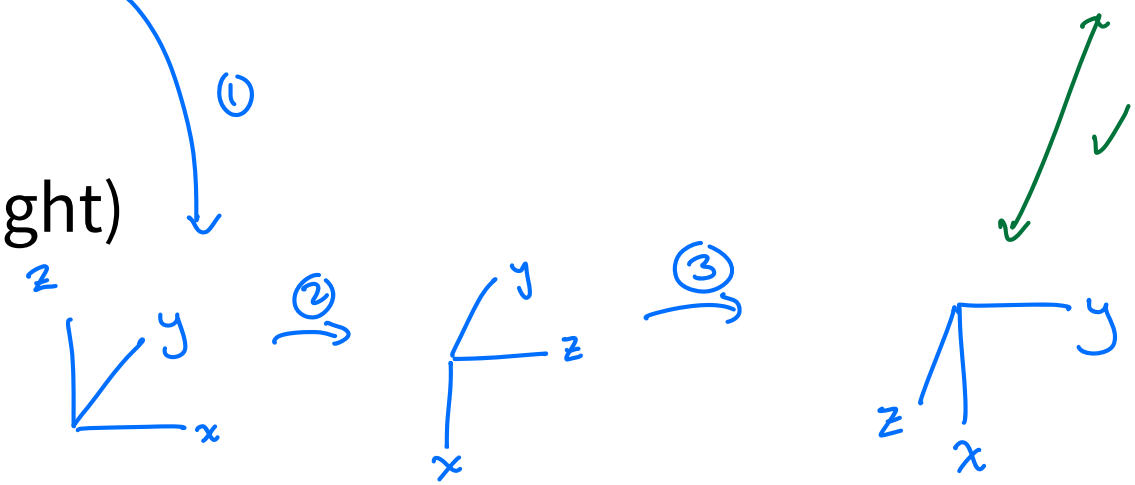
$$R_z(\pi/2) \cdot R_y(\pi/2) \cdot R_x(\pi/2)$$

Example

- RPY Rotation (right to left)
  - ① x-axis by 90°
  - ② y-axis by 90°
  - ③ z-axis by 90°



- Euler Rotation (left to right)
  - z-axis by 90°
  - *New* y-axis by 90°
  - *New* x-axis by 90°



Special Orthogonal Matrices

- 3D rotation matrices fall into the SO(3) **group**

Definition:

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times 3} \,\middle|\, R^T R = I, \det R = 1 \right\}$$

and

$$SO(n) = \left\{ R \in \mathbb{R}^{n\times n} \,\middle|\, R^T R = I, \det R = 1 \right\}$$

- Orthogonal matrices that follow the right-hand rule

$g_1, g_2 \in G$

Groups

- Multiplication maps into itself

$g_1 \cdot g_2 \in G$

- Identity operative

Unique $e \in G$: $g \cdot e = e \cdot g = g \quad * \; g \in G$

- Inverse

Unique inverse: $g \cdot g^{-1} = g^{-1} \cdot g = e$

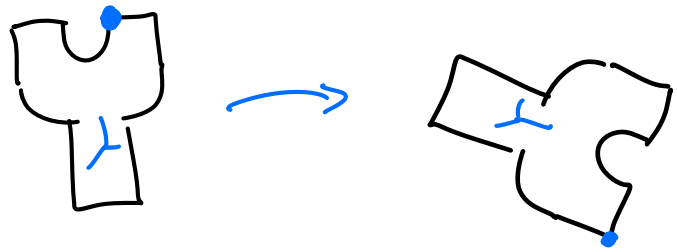
- Associativity

$g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3$

Rotation *and* Translation

General Rigid Body Transformations

- **Goal:** express rotation and translation
- (points translate, vectors simply rotate)



$$g = \underset{\text{rotate}}{R}x + \underset{\text{translate}}{P}$$

- Homogeneous coordinates:

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix} \qquad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix}$$

Homogeneous Transformation Matrices

- Compact representation
- Both rotation and translation included

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

$$g \cdot q = R \cdot q + p$$

↳ Transform a point

- Can stack and invert

$$g_{AC} = g_{AB} \cdot g_{BC}$$

$$g_{AC} = g_{CA}^{-1}$$

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

SE(3) is a group

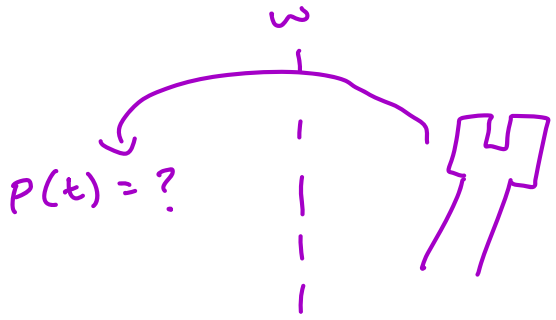
$$g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in SE(3)$$

$$SE(3) = \left\{ \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \;\middle|\; p \in \mathbb{R}^3,\ R \in SO(3) \right\}$$

Exponential Coordinates

Exponential Coordinates

- **Goal:** Create rotation and homogeneous transformation matrices as a *function of time*



- Comes from solving a differential equation → $\dot{p}(t) = - \cdot p(t)$
- We only need information about **how** the object moves (time is a parameter that's plugged in)

Exp. Coordinates for Rotation Matrices

- **Axis of rotation** creates a *rotation matrix*

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

* Same as Rodrigues Formula

Exp. Coords: (ω, θ)

Exp. Coords for General Rigid Body Motion

- **Twists** generate *homogeneous transformation matrices*

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix} \qquad g = e^{\hat{\xi}\theta} \qquad p(t) = e^{\hat{\xi}t} p(0)$$

- Pure rotation:

$$\xi = \begin{bmatrix} -\omega \times q \\ \omega \end{bmatrix}$$

- Pure translation:

$$\xi = \begin{bmatrix} v \\ 0 \end{bmatrix}$$

- Screw Motion:

$$\xi = \begin{bmatrix} -\omega \times q + h\omega \\ \omega \end{bmatrix}$$

Exp. Coords: (ξ, θ)

Reference Formulas

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}$$

- Rotation and Translation

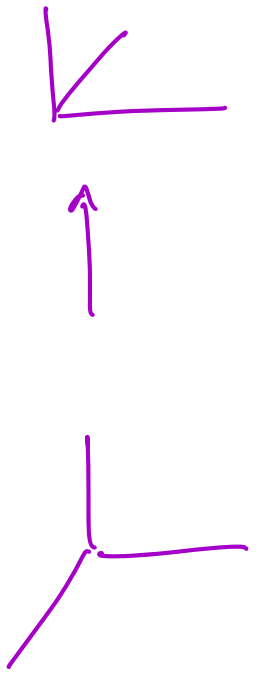
$$e^{\hat{\boldsymbol{\xi}}\theta} = \begin{bmatrix} e^{\hat{\boldsymbol{\omega}}\theta} & \left(\mathbb{I}_3 - e^{\hat{\boldsymbol{\omega}}\theta}\right)(\boldsymbol{\omega} \times \boldsymbol{v}) + \boldsymbol{\omega}\boldsymbol{\omega}^T \boldsymbol{v}\theta \\ \mathbf{0} & 1 \end{bmatrix}$$

- Pure Translation

$$e^{\hat{\boldsymbol{\xi}}\theta} = \begin{bmatrix} \mathbb{I}_3 & \boldsymbol{v}\theta \\ \mathbf{0} & 1 \end{bmatrix}$$

More on Screw Motion

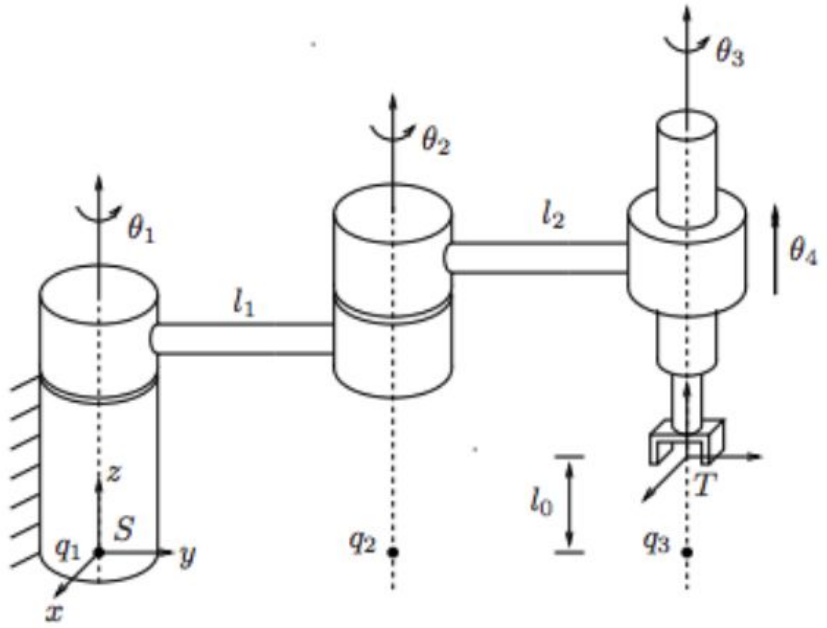
- Any rotation + translation can be expressed as a screw about a single axis
- Axis *w*
- Magnitude of rotation
- Pitch *h* (ratio of translation : rotation)
  - h = 0: pure rotation
  - h = infinite: pure translation



Forward Kinematics

Multi-Link Arms

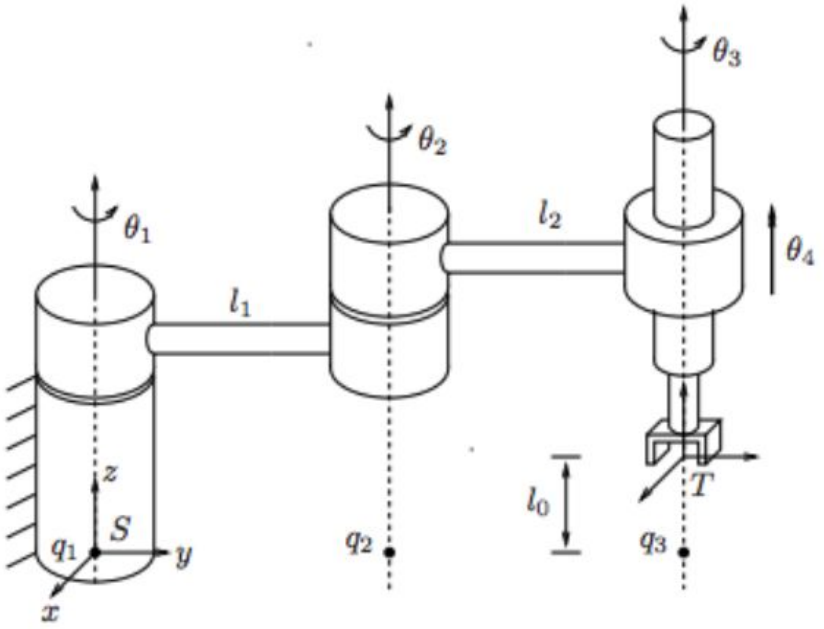
- **Goal:** Find the location of the tool after a multi-joint robot arm has moved around



Composition of Twists

1. Find twists of each joint in the reference configuration
   - All joint angles are 0
   - All vectors expressed in the *world frame*
2. Find starting position of tool
3. Put it together in exponential coordinates

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



$$\xi_1 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^T$$

$$\xi_2 = [l_1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^T$$

$$\xi_3 = [l_1+l_2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^T$$

$$\xi_4 = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]^T$$

$$g_{ST}(0) = \begin{bmatrix} I & \begin{pmatrix} 0 \\ l_1+l_2 \\ 0 \end{pmatrix} \\ 0 & 1 \end{bmatrix}$$

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

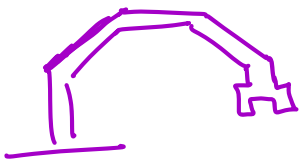
Notes

- **Order matters**: joints on the right cannot affect the position of joints on the left
- Forward kinematics (calculated using the world frame) will deliver the **same result** as composing homogeneous transformation matrices (see Discussion 3, Problem 3)

Inverse Kinematics

The Goal

- How do we move our robot's joints to reach a desired configuration?
- Given:
  - Where we want to go
  - Details about the robot
    - Twists
    - Starting Configuration



- Desired: $\theta_1 \cdots \theta_n$

Padan-Kahan Subproblems

- We have proven the solutions to some basic inverse kinematics problems
- Can we reduce our much more complicated robotics problem down to these?

Subproblem 1

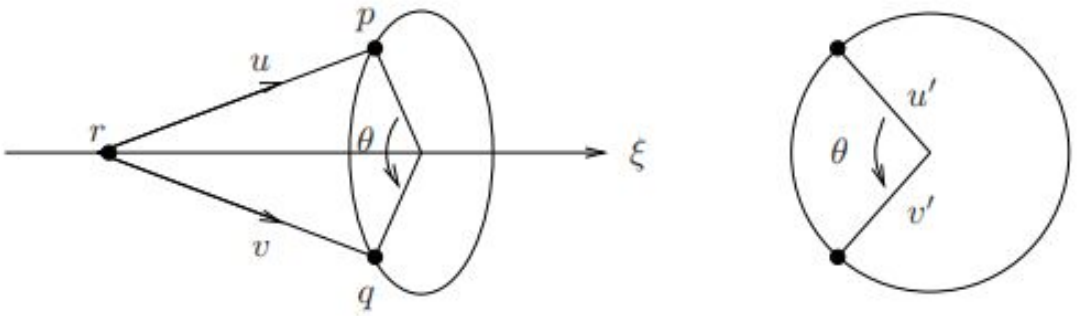
- Rotation about some fixed axis
  - We specify (p, q)
  - Find theta
- <= 1 solution

$e^{\hat{\xi}\theta} p = q$

≤ 1 soln

$\theta = \text{atan2}(w^T(u' \times v'), u' \cdot v')$

(won't need to ever compute this)

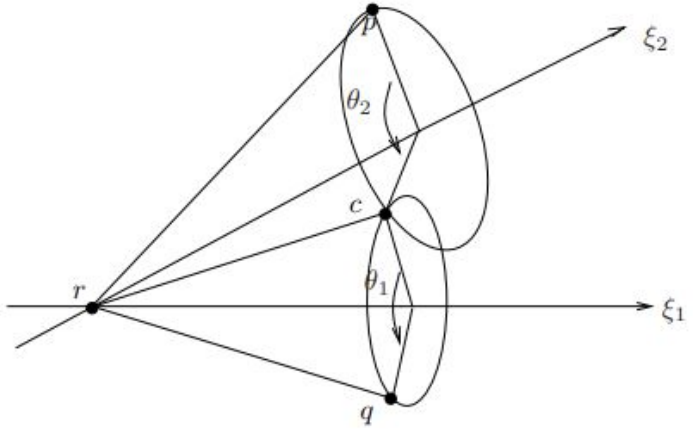


Subproblem 2

- Rotate about 2 intersecting axes
  - We specify (p, q)
  - Find theta
- <= 2 solutions

$$e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} p = q$$

≤ 2 sols

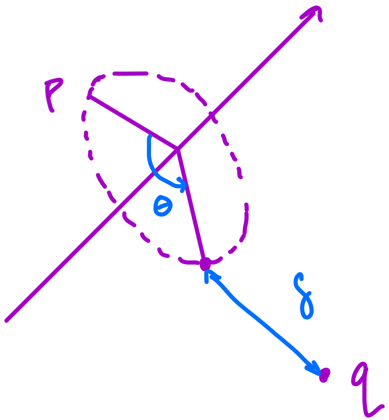


Subproblem 3

- Move to a specified distance from another point
  - Specify (p, q, delta)
  - Find theta
- <= 2 solutions

$$\| e^{\hat{\xi}\theta} p - q \| = \rho$$

$\leq$ 2 solutions



Extrapolating

- We know we can solve these simple IK problems to find theta
- But what about the complex robot arm?

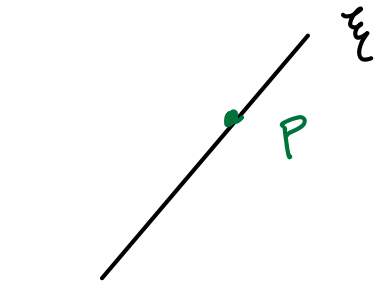
$$e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_6\theta_6} g_{st}(0) = g$$

- Solution: repeatedly apply subproblems using convenient points
- Some tricks to help us out

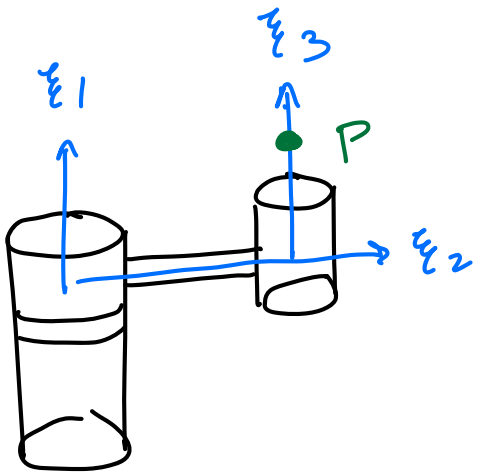
Rearrange

$$e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_n\theta_n} = g_d g_{st}^{-1}(0) := g$$

Trick #1: Eliminate on the RHS



$$e^{\hat{\xi}\theta} p = p$$



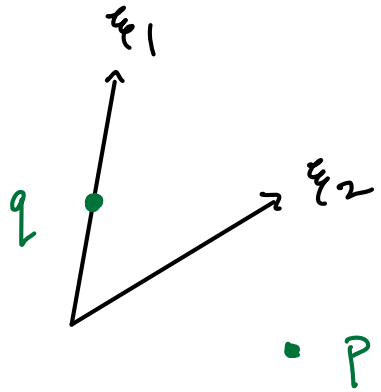
$$e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} e^{\hat{\xi}_3\theta_3} p = g p$$

- Pick a point p on ξ_3

$$\rightarrow \quad e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} p = g p$$

$\rightarrow$ Subproblem 2

Trick #2: Eliminate on the LHS Using Norms



$$e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} = g$$

- q on axis of joint 1
- p not on axis of joint 2

$$\| e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} p - q \| = \| gp - q \|$$

$$\| e^{\hat{\xi}_1\theta_1} (e^{\hat{\xi}_2\theta_2} p - q) \| = \| gp - q \|$$

Subproblem 3 $\longrightarrow$ $\| e^{\hat{\xi}_2\theta_2} p - q \| = \| gp - q \|$

Trick #3: Prismatic Joints

Get into form of S.P. 3:

$$\| e^{\hat{\xi}_3 \theta_3} p - q \| = \delta$$

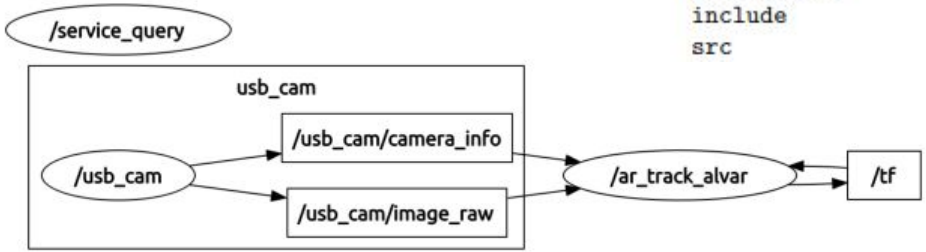
$$\delta = \underbrace{l_1}_{\text{original distance}} + \underbrace{\theta}_{\text{movement}}$$

Lab Content

ROS Overview

- Meta-operating system for a robot
- Hardware abstraction and message passing between nodes
- Things to know:
  - Computation graph
  - How the file system looks
  - Basic commands (ex. catkin_make)

```
ros_workspaces
  lab1
    build
    devel
      setup.bash
    src
      CMakeLists.txt
      foo
        CMakeLists.txt
        package.xml
      bar
        CMakeLists.txt
        package.xml
        include
        src
```



Publishers and Subscribers

- One way to pass content
- Publishers post messages, subscribers read them
- Things to know:
  - ROS message definitions
  - How to view various information (rostopic __)
  - Python code
    - You won't have to write your own, but fill-in-the-blank is definitely possible

```
<< data_type1 >>  << name_1 >>
<< data_type2 >>  << name_2 >>
<< data_type3 >>  << name_3 >>
...
```

Services, Requests, and Clients

- Like synchronous function calls
  - Client requests, waits for response
  - Server fulfills request and responds
  - Client processes response
- Execute in sequence instead of parallel subscriber callbacks
- Things to know
  - General Python file structure
  - How to run Python files (rosrun pkg file)
  - Make requests from command line (rosservice call …)


SNL/UNIVERSAL TELEVISION
ANY QUESTIONS?

Problem 7. When all else fails (15 points)

Let $u \in \mathbb{R}^3$ be a unit vector, and let $R = I + 2\hat{u}^2$.

(a) (3 points) Show that $R^T R = I$.

Hint: Recall equation (2.13) from the textbook: $\hat{u}^3 = -\hat{u}$.

(b) (3 points) Show that $\det R = 1$, and hence conclude that R is a rotation matrix.

Hint: The function $u \mapsto \det(I + 2\hat{u}^2)$ *is continuous, thanks to the continuity of the determinant. However, from part (a) it follows that* $\det(I + 2\hat{u}^2) \in \{+1, -1\}$. *Is it possible for a continuous function to take on exactly two discrete values? Can you conclude from this that* $\det(I + 2\hat{u}^2)$ *is actually a constant, for any* u*?*

(c) (5 points) Find the exponential coordinates for R i.e. find a unit vector ω and a scalar $\theta \in [0, 2\pi)$ such that $R = e^{\hat{\omega}\theta}$.

Hint: What does R *look like when we switch to a new reference frame where* u *is the* x *axis?*

Hint: Recall that the determinant of a transformation is invariant under a change of basis.

(d) (2 points) Verify that when Rodrigues' formula is applied to your answer in the previous part, you get the original matrix R.

(e) (2 point) What would go wrong if you tried to use Proposition 2.9 from the textbook to compute the exponential coordinates of a rotation matrix of this form?

Problem 3. Turtle Wrapper Node (15 points)

In Lab 1, we asked you to write a publisher node that would send a `geometry_msgs/Twist` over the `/turtle1/cmd_vel` topic in order to control your simulated turtle. You may recall only having to set two values of the twist: the linear x velocity, and the angular z velocity. This made sense at the time because our robot was entirely simulated in a 2D environment, reflecting the fact that it was a *unicycle modeled* robot; at any time, we may model the turtle's velocity relative to its own reference frame as

$$\vec{V} = \begin{bmatrix} v \\ \omega \end{bmatrix}.$$

where v is the linear x velocity and ω is the angular velocity. For a unicycle modeled robot, we always assume that the linear y velocity is 0. By controlling the turtle through directly manipulating a 6D Twist, you were breaking the abstraction between the perceived model of the robot and the commands the simulator needed to receive in order to control the turtle! To remedy this, you are now tasked with writing a "wrapper" node: you will construct a node that will listen for linear velocity and angular velocity commands published over a topic of your choice, and will publish that information to `/<turtle_name>/cmd_vel`, where `<turtle_name>` denotes the name of the turtle you want to control. Assume that your wrapper node has access to the desired turtle name through a command line argument. Assume this node will be written as a `.py` file placed in the appropriate location of a package named `midterm_1`.

(a) (3 points) What is the name of your node, what topic(s) do you want it to subscribe to, and what topic(s) do you want it to publish to? Remember that you want to be able to run multiple instances of your node if someone wants to use your wrapper node for multiple turtles.

(b) (2 points) You will be designing a new message type for the topic you choose to subscribe to. Define your `.msg` file. Make sure to indicate the name of the file somewhere.

(c) (5 points) Assume someone wants to control the turtle named "jturtle". A node named `user_control` is running that will send data of the appropriate message type to the topic your wrapper node subscribes to. Assuming that your wrapper node, turtlesim, and a `rostopic echo` node listening to the output of `user_control` for debugging purposes are all running. Draw the an approximate RQT graph that fits this scenario.

(d) (5 points) Time to code up your node! Fill in the appropriate blanks:

```
#!/bin/env python

import rospy
import sys
from geometry_msgs.msg import Twist
from ---------- import ----------

class TurtleWrapper:
```

```
    def __init__(self, turtle_name):
        rospy.Subscriber(__________, __________, receive_command)
        self.pub = rospy.Publisher(__________, __________, queue_size=10)

    def receive_command(self, cmd_vel_2D):
        cmd_vel = Twist()
        cmd_vel.linear.x = __________
        cmd_vel.angular.z = __________
        self.pub.publish(cmd_vel)

if __name__ == '__main__':
    rospy.init_node(_________, anonymous=True)
    turtle_name = sys.argv[1]
    wrapper = TurtleWrapper(turtle_name)
    rospy.spin()
```