CS 309: Autonomous Intelligent Robotics
FRI I

Lecture 19:
Alvar, TF, and Eigen

Instructor: Justin Hart

http://justinhart.net/teaching/2019_spring_cs309/

Checking in

- Everyone needs to do HW 4 by Friday!
- Everyone is on a team as far as I can tell
- HW 5 will be a **challenge** start **early**
  - Will be due April 16

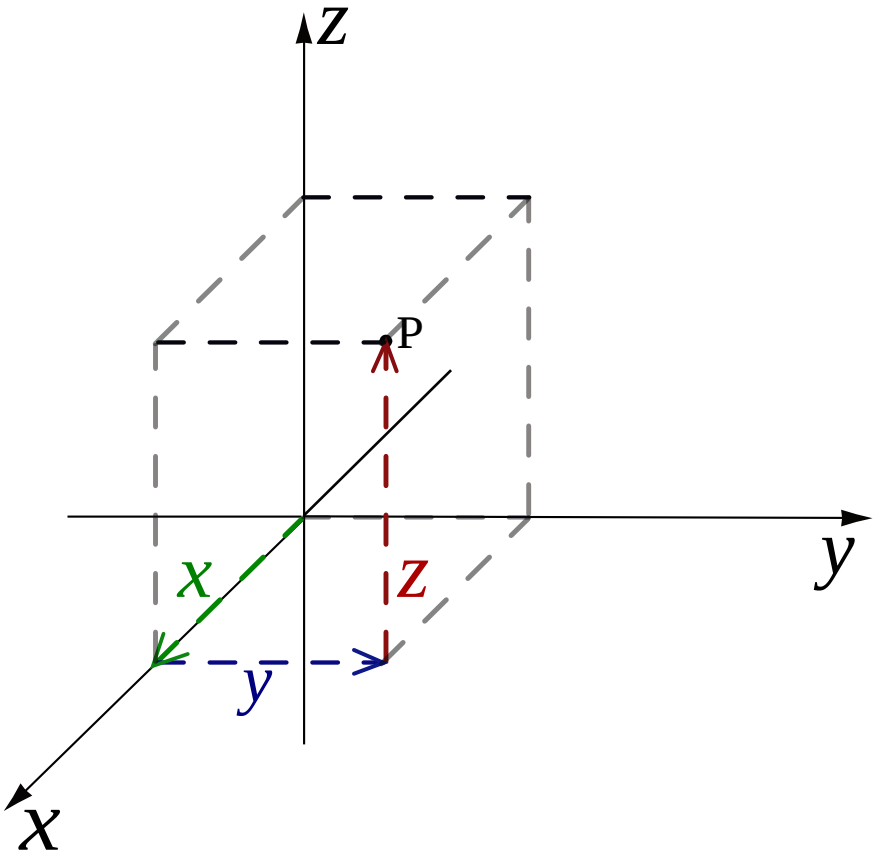
A couple of quick notes

- Robotics Study
  - If you're free, we appreciate the help. See the Canvas announcement.
- RoboCup@Home
  - We're still getting ramped up and you're welcome to participate!
- Unique ID for Fall 2019
  - **PLEASE** double-check this, I don't have the official number yet, but I think it will be: 50585
  - When you sign up, make **SURE** that you are signed up for the correct class (instructor: Hart)

ar_track_alvar

- ROS package for using AR Tags!
- Needs to be configured for your camera
  - This happens in a .launch file
- Generate AR Tags
- Print them
- Attach them to a rigid surface
- Your system can now track an AR Tag
- Example

3D Coordinate Frame



Getting serious about transforms

- In this example
  - AlvarMarker
    - Class I wrote to simplify your homework
  - PoseRecipient
    - Abstract class that you will use
- Example usage
  - Pick apart AlvarMarker
    - Nodehandle, subscriber
    - Note that Alvar is giving us poses, not transforms
    - TransformListener can transform a pose for us, which I put into the correct frame for you
    - PoseRecipient handles these poses
      - That's **your** job on the homework
  - ROSINFOPoseRecipient

Your homework will involve

- Recreating the functionality I’ve shown in this demo
- You will implement
  - OffsetPR
  - TFBroadcastPR
  - FaceOppositePR
- You will also make the robot follow the tag

Wrangling the mathematics

- Here, I’m going to provide a step-by-step guide regarding what you need to implement.
- First, let’s introduce the Eigen matrix math library, to simplify this.

Vectors

- Vectors have
  - Direction
  - Magnitude
- They all start at the origin
  - (0,0,0)
- You can think of a vector as an arrow pointing out of the origin.

Typically...

- X - <1,0,0>
- Y - <0,1,0>
- Z - <0,0,1>

3D Points

- Matrix math builds on the concept of vectors
- We’re not going to make you learn much, and all of the concepts will be in this presentation.
- We could represent a point P as
  - A Cartesian coordinate (x,y,z)
  - A vector <x,y,z>
- But we’re going to represent it as a matrix.

Matrices

- Matrices have rows and columns
- The identity matrix is all zeros, with 1's going down the diagonal.
- In case you're curious
  - You can think of each row of a matrix as being it's x, y, and z vectors.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotating Points

- Make point P a 3x1 matrix
- Make rotation R a 3x3 matrix
- Multiple R * P
  - The result is your rotated point

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$ Rotation around x axis

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$ Rotation around y axis

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$ Rotation around z axis

Translating Points

- Make point P a 3x1 matrix
- Make translation T a 3x1 matrix
- Add T + P
  - The result is your translated point

The robot’s orientation

- I get a TF message for base_link
- I want to know which direction the robot is pointing in
- I can answer this by
  - Putting the orientation quaternion into a rotation matrix R
  - Putting the z vector <0,0,1> into a matrix P
  - Multiplying R*P
    - The result is the vector in the direction the robot is facing.
  - This would also work for an AR Tag
    - **THIS IS HUGE HINT FOR YOUR HOMEWORK**

The Eigen Library

- Eigen is a matrix math library
- It provides all sorts of things
- For us it provides
  - Matrix types
  - Quaternions and Rotation Matrices
  - Matrix multiplication and addition

MatrixXd

- Eigen provides a very general way to describe many matrices
- MatrixXd means
  - The type is a Matrix
    - There are others, such as quaternions and vectors
  - X means that it can have an arbitrary number of rows and colums
  - d means that data is stored as doubles

Lets make a point!

```
Eigen::MatrixXd p(3,1);
p(0,0) = x;
p(0,1) = y;
p(0,2) = z;
```

Lets make a translation!

```
Eigen::MatrixXd t(3,1);
t(0,0) = tX;
t(0,1) = tY;
t(0,2) = tZ;
```

Lets try a rotation!

```
Eigen::Quaterniond r(w,x,y,z);
Eigen::MatrixXd rotatedP = r.matrix() * p;
```

- ROS expresses Poses and Transforms as quaternions, with x,y,z,w
  - **THIS IS ALSO A HUGE HINT FOR YOUR HOMEWORK.**

Lets try a rotation!

```
Eigen::Quaterniond r(w,x,y,z);
Eigen::MatrixXd rotatedP = r.matrix() * p;
```

- ROS expresses Poses and Transforms as quaternions, with x,y,z,w
  - **THIS IS ALSO A HUGE HINT FOR YOUR HOMEWORK.**

There are other ways to rotate

- Eigen also provides AngleAxis

```
Eigen::MatrixXd yAxis(3,1);
yAxis(0,0) = 0;
yAxis(1,0) = 1;
yAxis(2,0) = 0;
Eigen::AngleAxisd rY(angleInRadians, yAxis);
Eigen::MatrixXd rotatedP = rY.toRotationMatrix() * P;
```

Quaternions & AngleAxis

- If you want to rotate a quaternion (to compose 2 rotations together, it is simple)

  Eigen::Quaterniond r...
  Eigen::AngleAxisd rY…
  Eigen::Quaterniond rComposed = rY * r

- Note that order MATTERS
  - rY * r is different from r * Ry

- **THIS IS A BIG HINT FOR YOUR HOMEWORK!**

Order Matters
Remember the TF Tree

- Matrices representing transforms go from left to right, and from proximal to distal
  - That means that the leaf of your tree is all the way on the right, and the root is all the way to the left

    OR
  - It means that the transform that you're going FROM (the tip of the finger) is on the right, and the one you're going to (your eye, your shoulder) is on the left

Rotation and Translation

- Generally, translation will go to the right of rotation
- WHY?
  - Think about the tip of your finger, and your wrist rotates
    - If your finger is offset along the z-axis of your hand, then rotating your hand moves the tip of your finger.
    - Now rotate your elbow.
      - Relative to your elbow, getting to your finger tip is translate to tip from wrist, then rotate wrist, then translate back to elbow, then rotate elbow

So, now what?

- Well, starting your homework involves rendering tfs in rviz.
- You can send them to rviz using TransformBroadcaster as earlier in the slides.
- TransformBroadcaster sends StampedTransforms
- You pull data from the incoming Pose into Eigen to work out your math
- You pack the output Transform that goes into the StampedTransform using the results from Eigen
- Then you send!

Next time

- Next time, we will discuss turning this into navigation goals for the robot
- Combining the results from the first 2 parts of the homework with this will allow you to drive the robot
- So get started early!