CS 309: Autonomous Intelligent Robotics
FRI I

Lecture 19:
Forming Project Groups
Coordinate Frames Recap
TF, Alvar, & Eigen

Instructor: Justin Hart

http://justinhart.net/teaching/2018_spring_cs309/

A couple of quick notes

- Homework 4
  - Due Today
  - You need to do this **in front of a mentor**
- Robotics Study
  - If you’re free, we appreciate the help. See the Canvas announcement.

Looking forward

- Homework 5
  - Goes out tomorrow
  - Due April 12
- Homework 6
  - Also due April 12
  - Final project prospectus!

Today

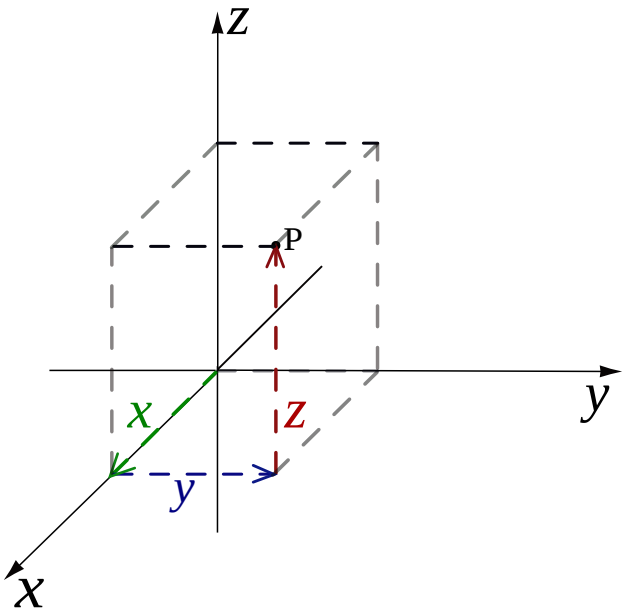
- Forming project groups – 15 mins
- Recap coordinate frames & TF
- More TF functionality
- Pose & Transform types
- Alvar
- Eigen

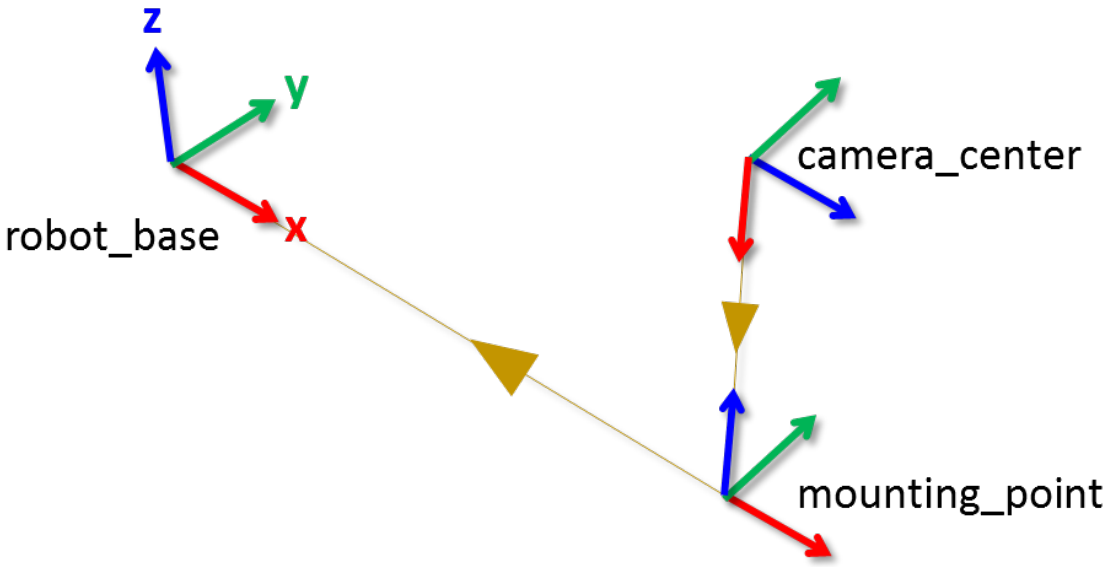
Project Group Formation

- This group will
  - Do homework 5 together
  - Do homework 6 together
  - Do the final project together
- Take 15 minutes to choose a team
  - Come up to me and I will put you together as a team in Canvas

Coordinate Frames

- Used to discuss poses
  - A position
  - An orientation





Coordinate Frames

- Two coordinate frames are related by a transformation
  - Rotation
  - Translation
- These are internally stored hierarchically in ROS
  - But we use TF to simplify this, and just refer to them by name

TF Classes & Functions

- tf::TransformListener listener
  - Special class that listens on behalf of the TF library so it can compute transforms between frames.
- tf::StampedTransform transform;
  - A spacial transformation
- Listener.lookupTransform("odom", "base_link", ros::Time::now(), transform);
  - Looks up the transformation “base_link” into “odom”

One more..

- listener.waitForTransform("odom", "base_link", ros::Time(0), ros::Duration(4));
  - Wait for the transform to be available
  - TF may not have heard enough data to make this transform work, and it will throw an error in that event.

You can also send frames

- tf::TransformBroadcaster br
- br.sendTransform(
  tf::StampedTransform(
  transform, ros::Time::now(),
  fromFrame, toFrame));

- StampedTransform - simply adds a timestamp to the transform
- ros::Time::now() - makes that timestamp now
- fromFrame, toFrame – The names the the frames.

Poses & Transformation

- Pose
  - The position and orientation of an object in space
  - Generally expressed as a position and a rotation matrix into the correct pose
- Transformation
  - The relationship between two poses
  - Generally expressed as a translation (a position) and a rotation
- ROS has types for both!
  - But they contain basically the same data

Poses & Transformation

- geometry_msgs::Pose pose
  - pose.position
    - pose.position.x (y,z)
  - pose.orientation
    - pose.orientation.x (y,z,w)
  - Orientation is expressed as a **quaternion**
    - We have software that handles it.

- tf::Transform transform
  - getOrigin()
    - getOrigin().x() (y(), z())
  - tf::Vector3 origin(x,y,z)
  - setOrigin(origin)
  - getRotation()
  - tf::Quaternion q(x,y,z,w)
  - setRotation(q)
  - The tf::Quaternion class also supports
    - Euler angles
    - Axis & angle

Let’s revisit Alvar

- Quick example
- To make your homework easier, I am providing some working classes
  - Your job will be to fill in the details.
  - I am going to provide you with a sketch of those details.
  - Your group will work out the rest.

Let’s revisit Alvar

- To review in this example
  - AlvarMarker
  - PoseRecipient
- For the first part of your homework, you will replicate
  - OffsetPR
  - TFBroadcastPR
- So we won’t be reviewing those

Your homework will involve

- Recreating the functionality I’ve shown in this demo
- Making the robot follow the AR tag.
- You will implement
  - OffsetPR
  - TFBroadcastPR
  - FaceOppositePR
  - NavGoalPR

Let’s revisit Alvar

- I will provide you with the AlvarMarker class
- In your homework, you will implement several pieces of functionality that build on each other
  - Transforming the orientation pulled out of Alvar
  - Rendering that tf in rviz
  - Turning that tf so it faces the opposite direction
  - Making the real robot move based on that pose
- I will also provide little functions and classes that fix things so everything lines up properly for navigation

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 HOEWORK!**

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!