CS 309: Autonomous Intelligent Robotics
FRI I

Lecture 20:
ROS Navigation Goals
Practical tips for HW 5

Instructor: Justin Hart

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

About the homework

- No, the package I will post does not compile
  - It is a starting point. You need to implement the relevant classes.
  - The code that I deleted is the solution.
    - Now you provide that code.
- We'll go more into the homework later in the class today, so let's hold questions for now.

A couple of quick notes

- Homework 5
  - Due April 16
  - Start early, it is hard and involves the robots
  - You can do it with your final project groups
- Homework 6
  - Also due April 16, we'll fudge until April 19, but plan on April 16, because you'll want time to work.
  - 1.5 page description of your final project plan
    - Discuss with me in advance
    - Your grade on this is based on how complete we think your plan is, and how good it is, not your ability to dump 1 page of text.

Undergraduate Research Forum!

- Friday, April 12
- 1:00pm – 6:00pm
- We will have a poster!
- I am looking for students who would like to present this poster.
- I will be judging the competition.
- This is a good opportunity to practice presenting in a professional setting, and should be easy, laid-back, and fun.
- If you’re interested, contact me.

Poster Presentation

- Do URF, and earn 0.5 pts of extra credit on your final grade.
- You don’t need to be there the whole time, but we need people the whole time.

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)

ROS Navigation Goals

- Tell the robot to move to a pose
- This lecture is based in part on
  - http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals

MoveBaseClient

- MoveBaseClient ac("move_base", true);

- Uses ROS ActionLib
  - ActionLib is like a ROS Service Call, with additional info
  - We skipped this at the start of the class in favor of more time on Topics and Services
  - MoveBaseClient USES ActionLib, but doesn’t require in-depth knowledge of ActionLib

MoveBaseClient

- while(!ac.waitForServer(ros::Duration(5.0)))
  - This line of code attempts to connect to the action server
  - It says, “While I’m not connected, keep trying to connect.”

MoveBaseGoal

- Used to send a pose to the robot for one of its frames to enter into using navigation
  - We will use “base_link”
    - Where the robot is on the floor for navigation
- This pose is ***relative*** to its ***current*** position
  - Remember how all poses are relative to another pose?

MoveBaseGoal

- move_base_msgs::MoveBaseGoal goal;
  - Message format for navigation goals
- ac.sendGoal(goal);
  - Sends the motion goal to the robot
- ac.waitForResult();
  - Waits for the result of the command
    - Could be success or failure

ac.getState()

- ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED
  - Tests whether the command was successful

MoveBaseGoal

- geometry_msgs/Pose pose
  - geometry_msgs/Point position
    - float64 x
    - float64 y
    - float64 z
  - geometry_msgs/Quaternion orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w

In the code

- geometry_msgs/PoseStamped target_pose
  - std_msgs/Header header
    - uint32 seq – We don’t use this
      - But you could think of sending a long sequence of messages and simply counting up: 1, 2, 3, 4
      - Then you would know which message was which in sequence
    - time stamp = ros::Time::now();
      - The timestamp associated with the message being sent.
    - string frame_id = "base_link";
      - The frame of the pose

In the code

- geometry_msgs/Pose pose
  - geometry_msgs/Point position
    - float64 x = 1.0
      - Move one meter forward
    - float64 y
      - Left/right
    - float64 z
      - This would go through the floor
  - geometry_msgs/Quaternion orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w = 1.0
      - This basically just means “facing forward”

The rest of the code

- geometry_msgs/Pose pose
  - geometry_msgs/Point position
    - float64 x = 1.0
      - Move one meter forward
    - float64 y
      - Left/right
    - float64 z
      - This would go through the floor
  - geometry_msgs/Quaternion orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w = 1.0
      - This basically just means “facing forward”

HW5 – Break it down into pieces

- This is a warm-up for your final project
  - On your final project, you really will be much more on your own than even this!!
- You are highly unlikely to solve your problem if you try to solve it all at once
- Take two basic passes at this
  - The big picture
    - How does everything fit together?
  - The details
    - How do the pieces all work?

HW5 – Break it down into pieces

- Once you have taken those two passes, work each detail out individually and know it works.
- ROS is built around this style of design
  - One thing controls the robot base.
  - One thing gives it navigation goals.
  - One thing tracks the markers.
  - Nothing tracks the markers and moves the base.
    - Instead, things might tell you how to pose the base relative to the markers.

HW5 - rosbag

- rosbag doesn’t work well with TF
  - There are hacks around this, but they are a bit more complicated than we’ve done before
  - So, share the Kinect while you work, don’t take it out of the lab.
  - We’ll try to get more Kinects in there as HW5 ramps up

HW5 - freenect

- You need to get Kinect point cloud data into the system
- Freenect does this!
  - roslaunch freenect_launch freenect.launch

HW5 – Visualize!

- rosrun rviz rviz

- rviz allows you to visualize ROS information
- Examples are in previous lecture notes

HW5 – Visualize!

- You will want
  - Point cloud data
    - /camera/depth_registered/points
    - This is the color point cloud from the Kinect
  - Where is it!?!
    - Go to “Global Options” in the menu on the left.
    - “Fixed Frame”
      - Drag down to “camera_link”
    - Familiarize yourself with clicking and dragging around the rviz interface

HW5 - ar_track_alvar

- This can be a bit tough to get working
- Use the launch file that I have included in the hw5_pkg
  - roslaunch hw5_pkg kinect_alvar.launch

HW5 – AR Tags

- Print out more AR Tags
  - Google ar_track_alvar
  - In the tutorial is the procedure for generating a marker.
  - Generate only Marker 0, it will make your life easier
  - Stick it to a rigid surface
    - The markers are assumed to be PLANAR
  - Share the markers!
    - **BUT DON’T SHARE YOUR CODE!!**

HW5 – Visualize your transforms!

- Now you want to see the camera's position and the marker
- Click "Add" in rviz
  - "By display type"
    - "rviz"
      - "TF"
- This will add TF frames to your rendering
  - Expand "TF" in the left column and show only those frames you want
  - I would pick "camera_link" and "ar_marker_0"
    - "ar_marker_0" will only show up if you are running the Alvar launch file.

HW5 – Get the software to compile

- Put classes together for the missing cpp files in the hw5_pkg
  - They MUST inherit from PoseRecipient
    - If you get something that indicates that the access isn't correct, maybe you need to inherit **PUBLICLY**
  - You can figure out what's missing by reading the CMakeLists.txt
  - To start with, simply put **EMPTY** classes that compile, **THEN** implement

HW5 – PoseRecipient

- To simplify things, I’ve provided a PoseRecipient abstract class
- Inherit from this to implement your classes for this assignment
- Quite simply, PoseRecipient classes should process poses provided through the abstract method
  - void receivePose(geometry_msgs::Pose &pose);

HW5 – TFBroadcastPR

- I would suggest getting this to work at this point in the process
- The purpose of TFBroadcastPR is to **broadcast** TF frames using the tf::TransformBroadcaster class
  - Notes on how to do this are in the previous lecture's slides!

HW5 – TFBroadcastPR

- If implemented correctly, you should see your broadcast TF frame in rviz.
  - Both in the **class** and in the program's **main**
  - Are you passing data through an empty OffsetPR?

HW5 – OffsetPR

- This class should work out the math to offset your pose from the marker.
  - This will place your broadcast pose 1 meter in front of the marker.

HW5 – OffsetPR – Getting Started

- Offset PR will **transform** the input pose from **receivePose()** and pass it along to another **PoseReceipient**
- A good starting point for OffsetPR is simply passing the pose through.
  - Where does the other pose recipient come from?
  - How do we pass that pose to the next class?

HW5 – Thinking about the math

- Now you need to work through the math.
- Last time we discussed the Eigen library in enough depth to build HW5
- As a warm up, try these things
  - Pull the numbers from the Pose's orientation into an Eigen::Quaterniond, then put them into the output pose
  - Pull the numbers from the Pose's position and put them into a MatrixXd(3,1), then put into the output pose
  - This won't change the result, but hopefully the warmup will help you to understand Eigen well enough for the rest.

HW5 – Thinking about the math

- OffsetPR should put output a Pose 1 meter in front of the marker.
- Translation is simply addition in our coordinate system.
- So, you should be able to take the position from the pose, add it to the offset, and arrive at the correct orientation, right?

HW5 – Thinking about the math

- Well, wrong.
  - Adding those two coordinates will put the output marker into **an** orientation 1 meter away from the marker, but in some other frame!
  - You want to be 1 meter **in front of the marker.**
  - What is wrong with your output frame?

HW5 – Thinking about the math

- The marker should be 1 meter away, rotate with respect to the frame
  - Easy enough!
  - You just formed a rotation from your quaternion.
  - Take your offset point, rotate it, render it.

HW5 – Thinking about the math

- **THIS IS MADDENING! WHAT'S WRONG!?!**
- Recall that a pose is expressed as a rotation and a translation.
- So the pose of our AR Tag, that we want to be relative to, is composed of a rotation and a translation.
- We've tried rotating the marker and translating it.
- Go solve the rest.

HW5 – Note on the orientation.

- Note that we haven’t been changing the orientation of the output pose.
- **HOWEVER**
  - We want the output pose in the same orientation as the original pose.
  - This should be easy enough.

HW5 – Now flip the orientation

- The next step in the homework is to flip the orientation of the output TF such that it is FACING the marker.
- Go back to last lecture’s notes and think about how to **rotate** that orientation.

HW5 – Run it on the robot

- You should now take your **current** code and try it on the robot.
- It doesn’t work :-(
- You will need to modify the .launch file included to use the Kinect sensor on the robot
  - This only involves changing the names of the topics referenced in the launch file.