Lab 9: Mapping

Learning Outcomes

- produce a pointcloud
- visualize pointclouds in rviz

Section 1: Requirements and Design

In this lab, you will build a map by creating a point cloud using data from the sonar sensors. You can conduct this lab in any environment with some obstacles (boxes) in it. You will manually drive the robot while collecting point cloud data.

Section 2: Installing the Lab

To perform this lab, you will need to get the lab9_mapping template into your catkin workspace. First ensure that your Jet has internet access by connecting it using WiFi or ethernet. Next ssh into Jet and enter the following command:

```
wget http://instructor-url/lab9_mapping/lab9_mapping-code.zip
```

Where the url should be replaced by the URL provided by your instructor. Now unzip the lab:

```
unzip lab9_mapping-code.zip -d ~/catkin_ws/src/jetlabs/lab9_mapping
```

Delete the zip file:

```
rm lab9_mapping-code.zip
```

To build the code, use the following command when you are in `~/catkin_ws/`:

```
catkin_make --pkg lab9_mapping && source devel/setup.sh
```

To launch the mapping node run

```
roslaunch lab9_mapping lab9.launch
```

Section 3: Producing PointClouds in ROS

You can read about the documentation for the PointCloud message type in ROS here: http://docs.ros.org/kinetic/api/sensor_ms
The important part of the PointCloud message is the array of `geometry_msgs/Point32`. Each point contains an x, y, and z location. The coordinates of each point are relative to the robot, not to the origin. In addition to the array of points, a PointCloud message also contains an array of `sensor_msgs/ChannelFloat32`. This array is the same size as the array of points, but it contains additional information like the color of the point or the intensity of the point.

In this lab, you will produce a point from each sonar sensor that has a valid (nonzero) reading; therefore the array of points have between 0 and 3 elements. To make this happen, you will need to subscribe to all three sonar sensor topics ("arduino/sonar_#"). "/arduino/sonar_3" is the left, "/arduino/sonar_2" is the center, and "/arduino/sonar_1" is the right. In the callback for each of these topics, save the sonar reading in a variable.

Implement a main loop that will populate a message PointCloud message and publish it. The units of your points should be meters (although the sonar values are in centimeters). The x and y coordinates will need to be calculated differently for the left, center, and right sonar sensors.

It is important that you do not output a point for a sonar sensor that is zero because zero means that ping was never received and therefore no object was detected. You can set the size of the points array by calling `resize(new_size);` where new_size is between 0 and 3. The PointCloud message header's frame_id should be "base_link".

You can run your node with `roslaunch lab9_mapping lab9.launch`. You can inspect the outputted points with `rostopic echo point_cloud` where "point_cloud" is replaced with the name of your PointCloud message. Try driving your robot around with the keyboard_teleop_twist_joy node.

Section 4: Visualizing PointClouds

PointClouds are most interesting when you can visualize them in Rviz. Open Rviz with `rosrun rviz rviz` on your desktop.

Add your PointCloud topic to the visualization and ensure that in the Global Options menu, Fixed Frame is set to "/map". You will need to increase the size of the points in the PointCloud settings to 0.1 or higher. It can be useful to add the current Odometry to the visualization to see where the robot is located on the grid.

By default only the most recent points are shown, but this prevents you from seeing the entire environment map. The decay parameter can be used to set how long the points are kept in Rviz. Set the decay paramter to 600 and drive the robot around for a while. How accurately does the point cloud match the real conditions? Are the errors due to the odometry or to the sonar sensors?

Section 5: Challenge

PointClouds are great for visualizing the robot's environment, but they are not easy to incorporate into planning systems that need a costmap. The costmap that we used in module 6 was an image with light-colored pixels corresponding to safe regions and dark-colored pixels corresponding to obstacles. Can you convert the PointCloud into a 2s costmap? To do this, you will need to infer the location of obstacles from the points. A basic way to approach this problem is to construct a large opencv Matrix that is initially all white. Then for each point, place a black circle on the matrix - the circle should be large enough that it will overlap with other nearby points. You will need to experiment with the size of the points, and you may need to perform additional operations to perfect the map. Once you are satisfied with the map, try to use it for planning.

This work is licensed by Cal Poly San Luis Obispo and NVIDIA (2016) under a Creative Commons Attribution-NonCommercial 4.0 License.