Warmup Project

Computational Introduction to Robotics

Loren Lyttle

9/27/20

Project Outline

The goal of this project was to become more familiar with ROS (Robot Operating System) by programming a Neato robot for a few different behaviors.

Behaviors:

Keyboard Operation:

Goal: Control the Neato via keyboard input.

Challenge: When controlling robots manually, it is always difficult to find a non-blocking method to read the controller. While the tty module provided a solution to this, it also meant that key releases were not registered. This made it harder to create intuitive and natural feeling keyboard control.

Approach: Because only one key was registered at a time, I decided to simply assign the WASD keys to the different directions. I toyed with the idea of defining velocity as a variable number, and it worked in creating gradual turns to the left and right. However, it failed when the Neato had to switch between going forwards and backwards. The delay between the keyboard and when the Neato moved in gazebo was too large to easily control the velocity with a variable. I settled on defining ‘forwards’ and ‘backwards’ as constants. To address the key release problem, the ‘E’ key stops the Neato and leaves it waiting for the next command.

Driving in a Square:

Goal: Autonomously drive in a 1 meter by 1 meter square.

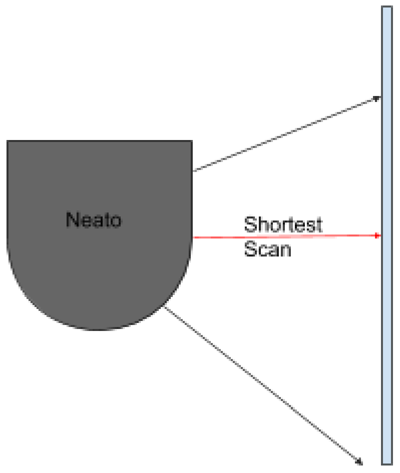
Challenge: I decided to use the Odometer to control the Neato in this instance. For driving in a square, this meant I had to choose 4 points in the global frame that made a square, then transform them to the Neato’s reference frame so it could find the vector to each point. I used a 3x3 transformation matrix to accomplish this task.

Approach: It was easy to get confused when deciding how to organize my code for this problem. I decided that, overall, I would like to enter the 4 points into my code, and have the Neato sequentially drive to each one. Considering that it would be easily adaptable to other behaviors as well, I created a Point Follower node that would drive the Neato to any global point with the input (x, y). Inside this node, the global point (x,y) would be converted to the Neato’s reference frame, then a vector could be made from $(0, 0)_{Neato}$ to the point $(x, y)_{Neato}$. This vector was used to derive the desired angle, and distance to set the Neato to. With this done, a separate node was used to simply input a new point every 6 seconds for each corner of the square.

Wall Following:

Goal: Drive parallel to a wall.

Challenge: While driving parallel to a wall by itself is fairly simple, the challenge for this behavior was when the Neato encountered the outside of a corner. The Neato does not inherently know that the next corner will be a left or right turn.



Approach: Out of the 360 distance scans for each degree around the neato, I decided that as long as the scan pointing at 270 degrees was the shortest, the Neato had to be traveling parallel to the Wall. Once it was known that the wall would always be right of the Neato, it simply turned right when unsure. If the Neato ran into the inside of a corner, it would drive straight until the wall ahead of it was closer than the wall next to it. A few if statements kept the neato between 0.5 and 1 meter away from the wall.

Person Following:

Goal: Follow an object in gazebo similar to a dog following a person.

Challenge: Initially, I had wanted the Neato to be able to track a person among other non-moving objects for this behavior. Doing this meant determining which objects were moving versus stationary in a global frame of reference. However, the LIDAR scans were too inconsistent, and Gazebo didn't register moving objects until after they were in their new position.

Approach: Despite assuming only one person would be present, the node for this behavior is setup to recognize how many objects are within the scanning range. The Neato cycles through its 360 degrees of LIDAR scans looking for non infinite values. If some are found, they are classified as a new object. Each time the Neato comes across an infinite value next to a finite number, it recognizes that it signals the end or beginning of a new object in the world. For each object, the Neato stores the average angle with respect to the neato that the object is located at. It then drives in that direction until its distance is less than 1 meter.

Obstacle Avoidance:

Goal: Reach a specified point while avoiding objects that are in the way

Approach: This was relatively similar to the Person Following task. However, instead of counting objects, the Neato counted the gaps between them. If one gap meant that the area in front of the Neato was clear, 2 separate gaps meant that an object was within scanning range. An arbiter was used to determine if there was a gap directly in front of the neato. If not, it aimed for the center of the gap that caused the least diversion from it's target course.

Finite State Controller:

Goal: Combine multiple previous behaviors to work together as one.

Approach: For this task, I decided to combine the wall follower behavior and the person follower behavior. Due to time restrictions, this behavior is just the amalgam of said behaviors. An arbiter was created to decide between the two based on the objects in view of the Neato. Normally, the Neato follows a wall, However, if it detects an object on it's left side, it breaks off the wall to go follow it.

Code Structure

For reusability, each of my nodes almost fully consists of a class that calls on methods. Each class consists of an init and run method as the basic skeleton. In addition, by the later behaviors, I almost always had an arbiter-like method that would make decisions and delegate to other functions.

Overall Challenges

Over the course of the project, my main limitation was insufficient knowledge of python. Almost every new method required countless google searches for applicable functions, correct syntax, and troubleshooting error messages. As this class and its projects continue, I hope to see my self reliance increase.

Improvements

One of the most prominent problems I encountered was with the Point Follower Node. Although it is useful because it defines the Neato in a global frame of reference, error comes from using the simulated wheel encoders. Even in the real world, wheel encoders can fail for many reasons (terrain, traction, gear slip, etc.) I believe this can be minimized by better utilizing variable speed control. My Point Follower Node uses set variables for the rotation and velocity, given the correct conditions. Jolting the wheels leads to a greater chance of slipping and messing up the global positioning. Gradually ramping the wheels to each speed would be more accurate.

For the Obstacle Avoider Node, the Neato chose which alternate path to take based on how far it deviated from its current trajectory. In its current state, the neato would drive towards such a gap even if it could not fit. However, this node also records the size of each gap. An improvement would be to take this gap size into account when deciding the next best path of travel.

For the Finite State Controller, there are a few notable improvements I would make. First would be a search feature. If no objects or walls are detected, the Neato should enter a search behavior and seek out more objects. Second, differentiating between the wall and a person can be handled with more finesse. My Finite state controller now assumes that anything on the right of the Neato is a wall, and anything on the left is a person. In addition, when switching to the Person Following Behavior, it often treats the wall like a person and just continues following it. Like mentioned in the Person Following behavior, choosing to follow a moving object can make it much easier.

Key Takeaways:

Make sectionalized and reusable nodes, classes, and functions.

> One of the most useful things I created in this project was the Point Follower Node. It was easy to modify, and usable in a few of my behaviors. I believe can also be used and referenced to in many future projects.

Keep track of data types.

> Both in python and ROS syntax, it is important to remember which datatypes you use. Often, certain operations require specific data types, and finding that bug in a lot of code can be tedious.

Ask for help.

> A huge resource for this project was my suitemates. Quickly asking them for help was both useful and boosted morale. However in the future, attending office hours and asking the teachers may be even more helpful.

Don’t be intimidated.

> As an add on to the previous takeaway, while it was hard to ask questions because I felt like I would expose myself for not knowing something basic. This was wrong for two reasons.
> First: It set me back farther taking the time to do research myself.
> Second, and more importantly: Everyone is learning, and we go to Olin! Everyone is glad to help.