CompRobo Warmup Project

Carlo Colizzi, Ben Grant

September 2022

1 Teleop

For this first part of the project, we focused on building a teleoperation keyboard control for the Neato, similar to what is contained in the *teleop_twist_key* package. This allows the robot to move in all directions at the user's command.

We decided to use the WASD keys to move the Neato around, specifically:

- ”W” moves the robot forward
- ”A” rotates the robot left (or counterclockwise)
- ”S” moves the robot backward
- ”D” rotates the robot right (or clockwise)
- Any other key will stop the robot from moving

We found out quickly that the robot would not stop when we lifted our fingers from one of the WASD keys, and therefore had to implement a new way of stopping the Neato.

2 Drive in a Square

For the second part of the project, we used a simple while loop and a series of ”sleep” commands to make the robot turn and go forwards. Our code creates a publisher and then follows the following flow:

1. Go forwards - publish command
2. Sleep 3s (time taken to go forwards 1m)
3. Turn left - publish command
4. Sleep 2.1s (time needed to turn 90º)
5. Repeat

After the first implementation of our code, we noticed that the Neato often was still moving forwards when it started to turn, and vice versa. Therefore, we decided to add a "stop" command, along with a 0.5s sleep, after steps 2 and 4. This allowed the Neato to come to a complete stop before taking on the next command, ensuring more consistent behavior.

3 Wall Following

For the third part of the project, we created a script that made the Neato follow a wall. We started by assuming that the Neato would always be following a wall to its right side, moving forward at a constant linear velocity, and steering to keep a constant distance of 1m from the wall.

3.1 Distance to Wall

The first step of our implementation was to find a way of sensing how close the Neato was to the wall. After trying a few different methods, we ended up simply finding the minimum laser scan value of the right-hand half of the Neato's laser scan array (elements 180 to 359). This value can be interpreted as the point-to-line distance from the center of the Neato's LIDAR sensor to a wall on the Neato's right side.

3.2 PID Steering

By using an imported library (https://pypi.org/project/simple-pid/), we created a PID controller with a set point equal to 1.0 (the distance in meters that we wanted the Neato to follow the wall at). Each time our ROS node received a new laser scan message, we used the PID controller to calculate an angular velocity magnitude to steer the Neato towards following the wall at a constant distance. The PID controller basically functions as a proportional controller that steers the Neato towards or away from the wall proportional to its distance error from 1m to the wall. However, our PID implementation is not as liable to oscillation as a simple proportional controller would be.

3.3 Results

We tested our wall following algorithm in the simulated maze environment where it has demonstrated the ability to follow both inside and outside corners, even steering the Neato 180 degrees around protruding walls.

4 Person Following

4.1 Object Detection

The first part of our person following algorithm was the object detector. We used a naive method of object detection given that we were testing in an empty world within the Neato simulator where each laser scan angles from the Neato where nothing was detected had values equal to inf. By filtering the scans for non-infinite laser scan points and finding the centroid of these points. We then calculated the object's location in global coordinates and stored it to a state member.

4.2 Object Following

This portion of our follower algorithm heavily relied on the idealized odometry localization from the simulated Neato environment. We calculated the Neato's desired heading that pointed from the Neato's global position to the global position of the detected object. We commanded the Neato to drive this heading by using a PID controller. Each loop iteration, we changed the PID's setpoint to the desired heading and fed it the Neato's current heading as the process variable, sending the PID's output to the Neato as an angular velocity command. Additionally, we implemented a second PID controller to command the Neato's linear velocity using a constant setpoint, and a process variable of the Neato's current distance to the detected object.

4.3 Results

As long as our object detection was actively sensing the location of an object, we were able to command the Neato to drive while steering to the object, slowing down and ultimately stopping right next to the object.

5 Obstacle Avoidance

5.1 Object Detection

In this script we used an approach similar to the one used in the ”person following” part of the project.

5.2 Odometry

In addition to the laser scan, our script also had a second subscriber for the odometry. This was used to find the Neato's position at at any point in time. We had an arbitrary target in global coordinates, and thus we were able to calculate the vector to get to this target. We then used the odometry to find the robot's position, and subtracted one from the other to obtain a vector for the direction we wanted to travel in.

5.3 Avoiding the Obstacles

We used the information gathered using the laser scan to pinpoint the location of the object in local coordinates, creating a vector pointing away from the obstacle and towards the Neato. We then assigned a weight to this vector, inversely proportional to the distance to the obstacle, so that an obstacle gained ”importance” as it got closer. We then added this vector to the other two previously obtained, thus resulting in a vector that suggested the ”best” path to reach the target point without hitting any obstacles. We used a PID controller (section 3.2) to control the steering angle, allowing the Neato to steer without abrupt turns.

6 Finite-state Control

6.1 Desired Behaviors

For our finite state controlled Neato routine, we wanted to combine our solutions for object following and wall following. Our goal was to run this routine in the simulated Neato environment with a cylindrical object placed on the plane. We wanted to drive our Neato up to the cylinder, turn the Neato to align its right side to the wall of the cylinder, and drive the Neato around and around the outside of the cylinder.

Behavior	**Description**
Follow	Follow the detected object
Align	Align heading to be parallel to object wall
Orbit	Drive circles around object
Stop	Stop moving if no object detected

Table 1: State descriptions

6.2 Switching Between Behaviors

Our finite state machine switches between distinct behaviors based on whether the Neato's LIDAR can sense an object and if so, how close the Neato is to the object. If the object detection does not pick up anything, we stop the Neato's drive motors. If an object is detected but is further than 1.1m away from the Neato, we activate the object following behavior, and if the object is less than 1.1m away from the Neato, we first turn the Neato 90° to the left and then enter the wall following behavior.

Current State	**Input**	**Next State**	**Output**
Follow	object seen	Follow	move towards object
	object reached	Orbit	turn 90° left
	object lost	Stop	stop motors
Orbit	object seen	Follow	move towards object
	object reached	Orbit	wall follow on object
	object lost	Stop	stop motors
Stop	object seen	Follow	move towards object
	object lost	Stop	none

Table 2: State-transition table