Computational Robotics: Warmup Project Writeup

Mia Chevere and Airel Chen

9-25-2024

Robot Teleop

Premise: This node collects user input to determine the robot's direction of movement. It has three preset angle and velocity states of movement each corresponding to the A, W, or D keys. The A key will direct the robot's movement forward and to the left, the W key will only direct the robot forward and the D key directs the robot forward and to the right. When no keys are being pressed the robot will continue in the direction that was last pressed.

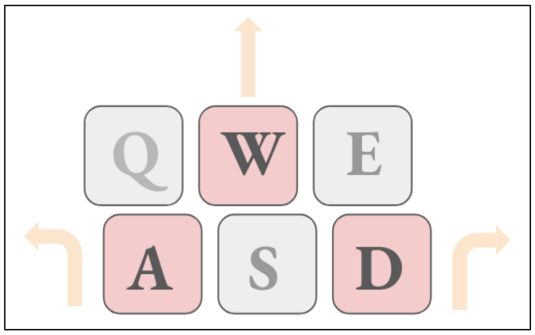

Fig. Keyboard layout with relevant keys

Results: This node functioned as intended! When deciding on what key inputs to take we wanted to simplify the program by not including a general start moving key or a general stop moving key and instead having the direction, and movement both be controlled by the same minimum number of keys. However, after implementing this we realized that the only way to stop the robot was to kill the program. Next time we will be sure to include a way to stop the robot with a key press.

Drive in square

Premise: This node use simple for loop for moving 1m forward then turning 90 degrees to the left, four times to complete a square.

Results: This node functioned as intended! However we did not incorporate the bump message to stop the neato when obstacles are encountered. We also didn't have time to implement other algorithms to account for the fact that friction might make the square inaccurate if the code is purely based on timing to determine when to turn.

Wall following

Premise: This node uses the neato's lidar scans to identify wall like objects and adjusts its linear and angular velocity to maintain a set distance from these objects. Once the program is initialized the neato moves forward at 0.1 m/s until it identifies a wall in one of two angular ranges 220-260 and 280-320. When it identifies a viable distance in one of these ranges it uses basic trigonometry to calculate the slope and intercept of the wall. Both can then be used to calculate the angle and distance between the neato and the wall. The neato can then calculate the difference between the desired distance from the wall and the current distance to the wall. It computed angular velocity using proportional control while keeping linear velocity at a constant value executes this as a twist. The effect should be that the robot turns as it approaches a wall to maintain the desired distance.

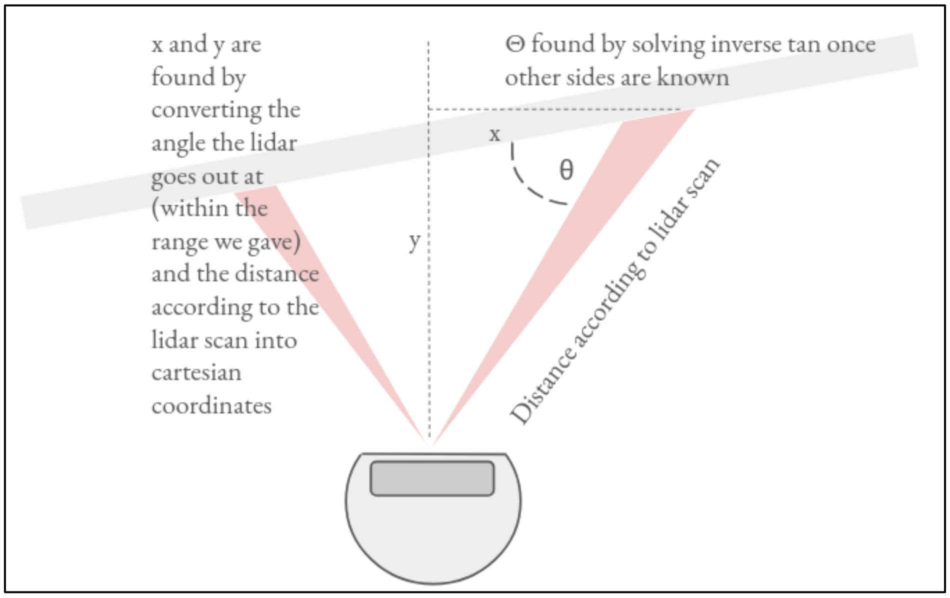
x and y are found by converting the angle the lidar goes out at (within the range we gave) and the distance according to the lidar scan into cartesian coordinates
Θ found by solving inverse tan once other sides are known
x
θ
y
Distance according to lidar scan

Fig. Visual representation of relationship between distance variables in code

and explanation of mathematical operations

Results: Unfortunately, this node did not execute the desired behavior reliably during testing. Depending on the neato's orientation to the wall it would occasionally turn at the wall but more often turn much earlier. While debugging we wrote print statements into the code that demonstrated that the walls were being correctly identified and the linear and angular velocity were both being changed accordingly via a twist msg. From this we noticed that the angular velocity was changing very slightly. In order to correct this, we increased the kp_distance value. In future tests the robot stopped running into walls because it didn't turn fast enough but instead, we got much more erratic turning, and it had a hard time staying focused on one wall. This behavior can be seen in the wall_follower bag file included in the warmup project. We suspect that overall, the constants require further tuning. Specifically, the distance from the wall might be too dramatic for the neato to correct to by the time that it has "seen" the wall.

Person Following

Premise: This node has the neato scan for leg like objects, orient itself to them and drive in their direction. Upon implementation the neato begins to use lidar to scan for sudden changes in an objects distance to it. Once an object enters its field of view causing this change it verifies that the object is about the appropriate size of a leg. This step is taken to avoid falsely identifying walls or miscellaneous small objects as legs. If the object is identified as a leg it uses trigonometry similar to that of the wall follower to estimate the position of the legs relative to the neato. It then uses proportional control to adjust the neato's linear and angular velocity such that it moves towards the legs. Given that this was a slightly more complicated system we included markers in the code for the position of the neato and the legs. The neato was represented as a sphere where as the legs were represented as a square the idea was that we could run Rviz2 as we ran our code and see what the neato was identifying as legs and also where it thought it was for debugging purposes.

Results: After extensive debugging we were able to tune our node such that it regularly performed the desired behavior. During the first test, the neato ran away from legs it identified. After running various print statements it became clear that it was an issue with the constants. Specifically the distance it was supposed to be keeping from the kegs was too great. After this modification it followed people but in a very limited capacity. Other modifications we made to improve our node included widening its field of vision, decreasing its linear velocity and shortening its field of vision so it didn't latch onto far away objects. After these modifications it followed people quite reliably as shown in the person_follower bag file and also the person follower video. One thing we did stay stuck on was that we were never able to get Rviz2 working with the markers. We wrote print statements to ensure that we published the markers without issue, and we were able to add them from the topics section in Rviz2 but no visible sphere nor square was ever generated on the plane. Even though this node performed as we wanted it to, if we had time to revisit it, we would do further debugging on our markers.

Object avoidance

Premise: This node has the neato scan for objects using its lidar sensor and move forward while avoiding the obstacles. collects lidar scan data from the neato and then filter out the obstacles that are either too far away (>= 1 meter away) or is already behind/beside the neato. Once all the obstacles have been identified, have the neato prioritize the obstacle closest to it and turn until the obstacle is no longer in the view of the neato, then the neato moves forward.

Results:

The node worked as intended! We have observed that the neato gets stuck when 2 objects (one on the left and one on the right) are at the same distance. We can solve that by having variable turning angle based on how close the object is to the neato or implement additional measures to prevent this from happening. Ex. Have the neato go straight under certain s

Finite-state

Premise: This node intends to combine the teleop node and the obstacle avoidance node. It sets the teleop node as its default behavior until an obstacle is detected. Then the obstacle avoidance node overrides the teleop node and the neato follows the obstacle avoidance node as its behavior until the obstacle is clear.

Results:

The teleop node worked as intended, however we encountered some problem when writing the conditions to switch to obstacle avoidance node. We tried to send the lidar scan data from the finite state node to the obstacle avoidance node for processing and making decisions but by using print statements under if statement as our debugging method, we discovered that the if statement was never executed therefor the obstacle avoidance node was never activated. Next time we will try importing code from both nodes and work with that instead of trying to have both nodes running in the background.

Take aways

Over this project, we were exposed to new challenges and as a result, we learned how to use new tools to overcome them. The most immediate examples are the neato simulation tools. Throughout the project we were able to debug code that we felt was too unpredictable to run through a neato with gazebo or see if the neato’s lidar sensor was even picking up on anything with Rviz2. As we continue through this course, writing more complicated code, it will be important to have a wide variety of debugging tools.

As useful as we expect the neato simulation tools to be in the future we are excited to have learned about technical concepts such as proportional control and odometry. Having to implement these concepts helped us get a better understanding of what information the neato is actually collecting and how we can leverage that data to inform it’s interactions with the world around it. This generalized understanding of the neatos sensors will help us scope projects throughout the class and come up with relevant solutions to challenges we’re presented with.