Lab 2: LIDAR Alarm for Turtlebot

Sasha Belotserkovskaya, avb27@case.edu; Kristina Collins, kvc2@case.edu; Shawn Qian, xxq82@case.edu; Connor Wolfe, cbw36@case.edu

I. Introduction

This assignment presents an example of obstacle avoidance using sensor detection on the Turtlebot platform. It serves to demonstrate how to effectively filter sensor input to avoid obstacles.

II. Testing

A. Demonstration Steps

First, connect to Turtlebot using the instructions in the lab assignment. The Turtlebot acts as the ROS master. Recommended to use Google's DNS for stability. Also, note that connecting to the Turtlebot in one terminal does NOT connect you to it in other terminals. To wit, for deeplearning02:

- **host deeplearning02w 8.8.8.8** Take note of the robot's IP, using Google's DNS.
- **ifconfig** Take note of the computer's IP.
- **ping deeplearning02w 8.8.8.8** Check to ensure that data can be exchanged.
- **source /opt/ros/indigo/setup.bash**
- **export ROS_MASTER_URI = http://deeplearning02w.eecs.cwru.edu:11311** Set the Turtlebot as ROS master.
- **export ROS_IP=XXX.XX.XXX.XX** Use the computer's IP here.
- **source /devel/setup.bash** Switch the source to match the location of the written nodes.s

It's wise to verify connection by checking rostopics with the command **rostopic list**. Next, run the program:

- **roscd**
- **rosrun bacon1 bacon_lidar**
- **rosrun bacon1 bacon_reactive**

The Turtlebot should start to move forward. When it sees an obstacle, it will stop and rotate counterclockwise in order to avoid it, then continue forward when it believes the obstacle is past. For detailed debugging, it may be wise to reduce the speed and yaw rate in code. The angle and distance of a perceived obstacle will be printed in the terminal.

B. Tips and Notes

It's wise to keep an eye on the Turtlebot and make sure it doesn't bump into anything. The instructions above and in the README recommend using the Google DNS and addressing the turtlebots by IP rather than hostname. We demonstrated our code on deeplearning02. Finally, note that connecting to the Turtlebot in one terminal does *not* create a connection in other terminal windows. For this reason, it's wise to use bash commands (CTRL+Z, B+G, F+G) to run both nodes simultaneously in the same terminal window. (This makes processes a little more difficult to identify and kill when necessary, but still saves time).

III. Code Changes

A. LIDAR Alarm

The lidar alarm code for PS2 subscribes to the rostopic **/laser/scan**, which is specific to STDR. In the Turtlebot implementation, it was necessary to subscribe instead to the rostopic **/scan**.

The PS2 lidar alarm code used for this assignment was based on Kristina's code, which drew on points from -90 degrees to +90 degrees (with 0 defined as the direction directly in front of the robot). This does not directly translate to the Turtlebot, which, unlike the robots in STDR, has a lidar range of less than 180 degrees. The arc of the "pizza slice" of points scanned was reduced from π radians to $\pi/8$ radians after larger arcs (e.g., $\pi/4$) resulted in false alarms.

Other possible points of alteration include the velocity and yaw rates for the robot, as well as the minimum safe distance in front.

B. Reactive Commander

The reactive_commander node was essentially the same as the one provided in the stdr_command package, with one key difference: our version publishes to /cmd_vel instead of /robot0/cmd_vel. This change, and other changes from STDR to Turtlebot-friendly message (including the rostopic change noted above), could potentially be taken care of by a general Turtlebot republishing node.

IV. Testing Results

As shown in our video, we ran a qualitative test of the code, which consisted of the robot chasing us around in a pleasing, calisthenic and undignified manner. The robot was able to successfully identify obstacles (us, mostly) and rotate counterclockwise in order to avoid them. In addition to its prescribed safe distance, the lidar appeared to have a minimum range as well, so it would only respond to obstacles between approximately .25m and 1m away. Standing immediately in front of the robot, therefore, would not usually set off the lidar alarm and stop it from moving forward. This behavior is best characterized by running the lidar alarm without the reactive commander node, moving in front of the lidar sensor while the robot stands in place, and watching the output of the terminal. The video can be accessed with the following link:

https://youtu.be/fnngkxnb74k

Our code may be accessed on Github from the following link:

https://github.com/shacks2017/teambacon

V. Conclusion

This exercise provided an example of feedback control, adapting simulation-oriented code to a physical robot, and running more than one ROS node simultaneously. The obstacle avoidance methods developed here may be of use in future projects concerning locomotion and path planning in dynamic environments.