Team Omicron

Midterm Project: Navigation Enhancements

Maxwell Bauer | Matthew Tong | Arpit Sarawgi | Vamsy Viswanath Putta

Mobot_Path_Execution_Omicron

Reference Code

- Code from Professor Newman's Repo was used as reference
    - https://github.com/wsnewman/learning_ros/tree/master/Part_4
- For clarity we named our Assignment nodes to be consistent with the HW pdf
    - Our Package Variation = Equivalent From Reference Code
    - Bold is given instructions and non-bold is our additions

Nodes

1. current_state_publisher = odom_tf
    - **This node will later combine absolute pose information (e.g. from GPS or LIDAR/map-based localization) with high-speed Odom information. For this assignment, this node should merely subscribe to the Odom topic and republish Odom on the topic "current_state".**

Left basically the same as Essentially we did no more than the assignment asked and created a new topic called current state that publishes the odom topic. The odom topic was left alone and published the same information.

2. **lidar_alarm**

Basically it just has a TRUE/FALSE lidar_alarm and when the lidar_alarm senses something within a given range (that we arbitrarily chose during testing with Gazebo) it reads as TRUE. It also uses ROS_INFO to let us know in the terminal if there is an object ahead that the lidar senses. In our code, we defined the scan area to be a rectangular area in front of Jinx. From testing on the physical Jinx, we were able to tweak the length and width of the rectangle so that Jinx would navigate to the 2 tables without hitting obstacles while at the same time not trigger a false alarm by detecting obstacles too far away and preventing Jinx from going to a safe waypoint.

3. **modal_trajectory_controller**

As mentioned above this function 'Steers' the robot. Additional functionality was added in the midterm assignment but for this one is simply creates a 're-publisher' that republishes the des_state information to control the mobot

4. **des_state_publisher_service = des_pub_state**

When invoked these 'cases' cause the robot to perform the desired pre-planned action for control. In essence the 'desired state' becomes what action you want the mobot to perform

5. **traj_builder = traj_builder**

Creates a series of desired points for the robot to travel to using the above controller options. We build the ‘trajectory’ of the robot that we want it to move along.

Discussion

During this project, we struggled in the beginning with ROS/repository formatting:

- Had difficulties in making mobot_urdf and given files for mobot alongside our altered files to try
    - catkin_make --help gave us some useful options in fixing these difficulties
    - Particularly, the 'make only packages with dependencies' option for our packages and Prof Newman's Part 2 mobot_urdf package
- In our updated 'midterm' assignment we also added in a launch file to run of all of the nodes except the navigation_coodinator for convenience in testing

Later on, most of our frustrations were with the physical equipment. After getting our simulation to work successfully, our initial expectations were that moving to the physical Jinx would be more or less seamless. Unfortunately, we ran into issues with AMCL/LiDar.

During our testing on Jinx, we observed many inconsistencies and unexpected movements compared to our simulations. To be specific, random sudden speed changes and differences in code defined waypoints vs. physical waypoints. We later noticed the pattern that whenever there was an unexpected behavior, the terminal running AMCL would display an error “couldn’t determine robot’s pose with laser scan” and the RVIZ pose array would be very scattered indicating that the robot did not know its reference/transpose to the map. At times, we would obtain coordinates physically by pushing the robot to the desired location, recording the coordinates, coding them into our Mobot navigation code, testing in Gazebo, running on Jinx. It seemed that running

the code, moving Jinx back to the origin, and running the same code again could yield different results. Sometimes Jinx would successfully navigate to the first table perfectly fine, backup fine, then fail to get to the second table, sometimes it would not even make it to the first table. To correct this, we would reobtain all the waypoints and try again and again until we were finally about to get Jinx navigate to the first table, backup, navigate to the second table, and back to the origin (shown in video).