Team Omicron

Midterm Project: Navigation Enhancements

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

Midterm Mobot Mapping

Extended Work From PS4 Mobot Path Execution Assignment

- Problem Set 4 Assignment for Mobile Robotics
    - https://github.com/mab405/Mobot_Path_Execution_Omicron.git

Reference Code

- Code from Professor Newman's Repo was used as reference
    - NOETIC CODE
        - https://github.com/wsnewman/learning_ros_noetic.git
    - MELODIC CODE
        - https://github.com/wsnewman/learning_ros.git

Nodes

1. current_state_publisher

Refined from PS4 assignment with unnecessary information code and topics removed from cpp file. Same functionality as last time but with a more efficient file to re-subscribe to odom topic and publish info to current_state.

2. lidar_alarm_mobot

As before it returns information regarding if an obstacle is within some specified distance (TRUE/FALSE). Also has functionality to check that our range is within lidar parameters. Returns additional information (vectors of ranges to a designated point) regarding information within the lidar 'box' range that we are sensing. This then gives us the right, left and middle of the box. We also publish this information to lidar_alarm for use in other nodes. 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 1st table 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 ‘cases’ were invoked within the file as it is run from des_state_publisher_service that switches the mobot to a sort of movement mode. Forward, Backward, Spin Right, Spin Left, Brake, Stop, etc. We can then combine these movement ‘modes/cases’ to control the mobot. These modes invoke movement using desired state and rotation values from elsewhere in the file.

4. des_state_publisher_service

As mentioned above a series of Cases were created for use in controlling the robot that can be invoked in different files to switch how the robot moves and control it. It uses the pieces from traj_builder, where the desired velocity trajectory plan is obtained from, based on the case. It subscribes to lidar alarm and pauses the robot, when the alarm is triggered.

5. navigation_coordinator

It defines the coordinates where the robot has to move. It has pieces to publish the points to des_state_publisher_service and receive response from it, to determine if the movement was a success or not. It also has the ability to retry if failed. It also has a backup piece which can be called from this node itself.

6. 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. In this section we also added in code to allow our Cases to function. In particular since PS4 we added in code to allow the new ‘Backup’ Case to function. Breaking Traj is already completed in the PS4 code, which has been reused.

Discussion

- As seen in our video we were having some accuracy issues with going between the second set of tables
    - We believe that given some more testing we could likely improve this further, but with limited time and class-wide issues with the mobot we felt our video more than sufficiently demonstrates the primary aspects of this assignment
    - Our code runs perfectly as expected within Gazebo, and our real-life issues had more to do with things running strangely across the class than

any particular issue with our code or understanding of Jinx

- It was surprising how well our simulations translated to Jinx in the real world
    - It was additionally surprising how easy it was to utilize Jinx in running our code after learning how to do so for the first time
- We had some issues with AMCL and in particular noted some strange things. In particular we found our robot would occasionally have ‘hiccups’ in its measurements of where it THOUGHT it was for, as best we and the Professors could tell, for no particular reason.
    - This meant we never FULLY trusted the measurements we were receiving and fundamentally altered what we could consider to be a ‘precise’ measurement from the computer

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).