ROS Navigation
Comparative Report
Manu Parashar

1. Mapping

The first step for Navigating a robot in an environment is to create a map for it. I used the slam_gmapping node from the gmapping package. The launch file for this with all the parameters is in my_summit_mapping package.

I used the keyboard_teleop.launch file provided in summit_xl_gazebo to manually control the summit_xl robot and create a map of the environment.

The map pgm and yaml files are in my_summit_mapping/maps directory.

2. AMCL

The next step in creating the navigation stack is to localize the robot in the environment. For this I created the my_summit_localization package. The AMCL needs to know the map of the surroundings that we created in the previous step. This is done by launching the map_server node with the path to the map's yaml file sent as arguments.
The following lines added in the my_summit_localization.launch file does the same.

```
  <node pkg="map_server" type="map_server" name="map_server"
output="screen"
 args="$(find my_summit_mapping)/maps/project_map.yaml"/>
```

Then I added the code for the amcl node along with the parameters to the launch file.

The next step the project asked to do was to create the spots.yaml file needed later while interacting with navigation stack. This file is in the spots directory in the my_summit_localization package. I used the keyboard_teleop.launch to move the robot to the desired locations and manually create this file by spectating the /amcl_pose topic.

However, as required for the project I did create the save_spots.py file which contains a service which writes the current pose of the robot to spots.txt file.

This service can be launched with rosrun and the command to call the service is:

```
rosservice call /record_spot "label: fetch_room"
```

3. Path Planning

The last piece of puzzle for the Navigation Stack is path planning. I created the my_summit_path_planning package which has the my_path_planning.launch file. I added the code to launch the amcl launch file inside.

The path planning is automatically done using the move_base node, we just have to provide it the appropriate parameters. All the parameter yaml files are in the config directory inside the my_summit_path_planning package.

I use the Navfn as the global planner which uses the Dijkstra algorithm to find the shortest path to the goal, and the dwa_local_planner for the local planner which uses dynamic window approach, the local map created from the sensor data and the static global map provided in the /map topic to navigate the robot while avoiding unseen obstacles.

4. Interacting with the Navigation Stack

Lastly in this project we created a service that when called with a label from the spots.yaml file in the spots directory of my_summit_localization package will move the summit_xl robot to that pose.

The code for the service to be called is in get_coordinates_service_server.py which when called with the label gets the goal pose and saves it in rosparam and initializes a SendGoal object defined in the send_coordinates_action_client.py file which is a action client for the /move_base action server.

This client gets the goal pose from rosparam and sends the goal to the server and then the Navigation stack completes the job.

The message used by this service server is located in the my_summit_localization/srv folder

The launch file includes along with the node to initialize this service the code to launch the launch file from my_summit_path_planning package.

5. Parameters for Navigation

The most challenging part of this project was to set the right parameters.
To build the navigation stack for the summit_xl robot we had to use the

/hokuyo_base/scan instead of /scan
/summit_xl_control/cmd_vel instead of /cmd_vel
/summit_xl_a_odom instead of /odom
/summit_xl_a_base_link instead of /base_link

We also had to add the following lines in the amcl launch file to make the navigation stack work with summit_xl:

<param name="odom_frame_id" value="summit_xl_a_odom" /> <param name="base_frame_id" value="summit_xl_a_base_footprint" />

During Navigation the robot seemed to be having trouble navigating into the doors of the rooms, I played around with a few parameters and eventually setting the inflation_radius to 0.6 in the costmap_common_params.yaml file fixed the issue.

Also once the goal was reached the robot sometimes kept rotating to and fro infinitely, this was solved by increasing the yaw_goal_tolerance and xy_goal_tolerance to 0.8

6. Suggestion

One suggestion to improve the course would be to dive a little deeper into how the global and local planner are implemented.