Packages in the Home Service Robot

The Home Service Robot uses various packages that cover localization, mapping and navigation. This document provides an overview of these packages.

1) My_robot
    a) A custom robot is built using Gazebo that is reused through various Robo-ND projects. The environment has a building inside which there is a two wheeled robot.
    b) The environment also hosts various features, which is later useful for sensors to help localize the robot in this environment. This is created through universal robot description files that host plugins and jointst created.
    c) The robot has an RGBD camera (openni_kinect) and Laser (Hokuyo).
    d) The package serves to interface with various other packages/nodes for example, keyboard teleoperation, localization like amcl/gmapping, rviz through various launch files.

2) AMCL
    a) The adaptive Monte Carlo localization is a probabilistic localization method. In this project, the AMCL takes map data as input (created by pgm_map_creator) and helps the my_robot localiza in this map. Given the map, control actions and sensor measurements, the robot pose is estimated in this package.
    b) Particles with estimation of robot pose are used to represent the robot. After a motion update (estimates pose belief) and sensor update (corrects pose belief) step, a resampling wheel based on weights of particles (proportional to posterior of motion and sensor update) is done which decides to keep only a predefined number above threshold weight.
    c) To mitigate particle deprivation, (if all particles converge to an erroneous state), a minimum number of particles need to represent every step of the iteration, by randomly adding extra particles based on threshold condition.
    d) Please note the position of sensors, for example a laser on the robot. This should not be in a position that causes obstruction for laser scan, from the robot body, as this will lead to improper localization.
    e) Parameters associated with the amcl node can be tuned, to achieve better results.
        i) Overall filter:
            (1) As amcl dynamically adjusts its particles for every iteration, it expects a range of the number of particles as an input. Often, this range is tuned based on your system specifications. A larger range, with a high maximum might be too computationally extensive for a low-end system.
            (2) Also an initial robot pose can be provided to this node..

(3) Amcll relies on incoming laser scans. Upon receiving a scan, it checks the values for update_min_a and update_min_d and compares to how far the robot has moved. Based on this comparison it decides whether or not to perform a filter update or to discard the scan data. Discarding data could result in poorer localization results, and too many frequent filter updates for a fast moving robot could also cause computational problems.

ii) Laser:

(1) There are two different types of models to consider under this - the likelihood_field and the beam. Each of these models defines how the laser rangefinder sensor estimates the obstacles in relation to the robot.

(2) Tuning of these parameters will have to be experimental. While tuning them, the laser scan information in RViz may be observed in order to try the laser scan to be aligned with the actual map, and how it gets updated as the robot moves. The better the estimation of where the obstacles are, the better the localization results.

iii) Odometry:

(1) The robot uses differential drive for mobility,so the diff-corrected type is used as odom-type. There are additional parameters that are specific to this type - the odom alphas. These parameters define how much noise is expected from the robot's movements/motions as it navigates inside the map.

3) Move_base

a) Using the move_base package one can define a navigation goal position for the robot in the map, and the robot will navigate to that goal position.

b) It utilizes a costmap - where each part of the map is divided into which area is occupied, like walls or obstacles, and which area is unoccupied. As the robot moves around, a local costmap, in relation to the global costmap, keeps getting updated allowing the package to define a continuous path for the robot to move along.

c) Based on specific conditions, like detecting a particular obstacle or if the robot is stuck, it will navigate the robot around the obstacle or rotate the robot till it finds a clear path ahead.

d) Similar to the amcl node, move_base node requires a set of parameters to move the robot in the world. The rosparam tag can be used to include config files to set multiple parameters. For in the config ‘base_local_planner_params.yaml’, pdist_scale indicates that the robot should give high priority to the global planner. gdist_scale indicates that the robot should give high priority to the local planner. occdist_scale indicates how far the robot should travel from the obstacles.

Setting a really low value indicates that the robot can travel very closely to the obstacle and may even collide.

4) SLAM (gmapping)
    a) The underlying difference between localization and SLAM is that localization estimates pose using known map, control actions and sensor measurements and in SLAM, the problem that is solved is simultaneously creating the map and localizing on it given the control actions and sensor measurement.
    b) SLAM is of two types by form: Online and Full slam. They are continuous and discrete by nature, respectively. Full slam will estimate robot pose and map at time t-1, given measurements and control for that time. At time t, it will try to estimate pose for the entire path thus estimating robot variables throughout robot travel time. In contrast, Online SLAM will estimate variables that occur only for time t.
    c) Grid based fast slam employs a particle filter like amcl and also an occupancy grid mapping to solve the SLAM problem. Occupancy grid mapping generates occupancy value (negative log odds) for the grid in the perception field.
    d) In this project, SLAM is tested using gmapping and a map is generated. Please note that keyboard teleoperation is used for this purpose by relying on the teleop_twist_keyboard package for robot movement actions in the environment.

5) Pick_objects
    a) The package serves as a provider of navigational goals to the move_base package aforementioned.
    b) This replaces manually commanding the robot to reach multiple goals with the 2D NAV Goal arrow in rviz.
    c) The ROS navigation stack then creates a path for the robot based on the uniform cost search algorithm (a variant of Dijkstra's algorithm).
    d) With the help of the relevant, official ROS tutorial, this node can be created.The code is modified to add multiple navigation goals.
    e) In this project, two goals are continuously provided to the robot- a pick up and a drop location, in the map of the environment. (Note that, goal locations can be sent as a rosparam using yaml config to the launch file.)

6) Add_markers
    a) The package serves to model virtual objects with markers in rviz.
    b) In this project, the virtual object is the one being picked and delivered by the robot, thus it should first appear in its pickup zone, and then in its drop off zone once the robot reaches it.
    c) With the help of the relevant, official ROS tutorial, this node can be created.The code is modified to publish a single type of virtual object.

d) In this project, the add_markers node subscribes to the odom topic from gazebo. Before the robot reaches pick up location as provided in pick_objects node, a marker (cube type) is continuously published at pick up location.
e) Once the robot reaches the pick up location, the marker disappears in a few seconds mimicking a pick up. A marker of the same virtual object type, reappears at drop location, when robot reaches drop location.
f) Another solution is to use ROS parameters, subscribe to the AMCL pose, or even to publish a new variable that indicates whether or not the robot is at the pickup or drop off zone.