Project: Were Am I

Abstract:

The localization project is to create a ROS package, develop a custom robot model for Gazebo, integrate Adaptive Monte Carlo Localization (AMCL) and Navigation packages, and tune parameters for localizing the robot in the provided map and moving it to the target.

Background:

Localization is important for Robot. Localization is the challenge of determining robot's pose in a mapped environment. There are four popular algorithms for localization.

- Extended Kalman Filter
- Markov Localization
- Grid Localization
- Monte Carlo Localization

Extended Kalman Filter (EKF) uses gaussian filters to estimate the state of robot in non-linear models. Markov localization is a Bayes filter localization which maintains a probability distribution over the set robot poses might be. Grid localization is a histogram filter. Monte Carlo (MCL) is also known as particle filter estimating robot's pose by particles.

Three types of challenges of location problems in a static environment.

- local localization (position tracking), known initial position, uncertainty of robot motion and sensor data
- Global localization, unknown initial position, must determine its pose relative to ground truth map
- Kidnapped robot problem, teleported to a different location, more challenge than global localization

EKF and MCL are the two most popular location algorithms. MCL has advantage over EKF in ease of implementation, over Gaussian distribution limitation, and memory and resolution control.

Adaptive Monte Carlo (AMCL) can adaptively adjust particles. AMCL even can address kidnapped localization problem besides local and global localization.

This project builds a simple robot based on the project sample and uses AMCL in ROS package.

It demonstrates satisfactory results by localizing the robot and moving it around obstacles to reach the target.

Introduction:

Steps to setup project:

1. Create ROS package

First, create a directory catkin_me for the project.

$ mkdir -p ~/catkin_me/src

$ cd ~/catkin_me/src

Second, create package udacity_bot under it.

$ catkin_create_pkg udacity_bot

$ cd udacity_bot

Then create directory for worlds and launch.

$ mkdir launch worlds

Copy world description from project instruction to udacity.world under the worlds directory. Udacity.world is an XML file contains ground plane, light source, and world camera for the Gazebo environment.

The launch directory contains launch files for the project. As you can see under ~/catkin_me/src/udacity_bot/launch directory, there are three launch files, amcl.launch, robot_description.launch, and udacity_world.launch. Udacity_world.launch will launch a robot defined by robot_description.launch and an empty world with jackal_race.world.

2. Develop a custom robot model for Gazebo

Udacity_bot.xacro defines the robot in ~/catkin_me/src/udacity_bot/urdf. There are links, footprint, chassis, left_wheel, right_wheel, camera, and hokuyo (laser range finder). Robot_footprint_joint connects chassis to robot_footprint as fixed. Joint left_wheel_hinge connects link left_wheel to chassis as continuous mode as wheel can rotate. Similarly, right_wheel_hinge connects to chassis as continuous mode.

Camera_joint connects camera to chassis as fixed. hokuyo_joint connects laser to chassis as fixed.

Inside definition of a link, it might contain inertial, collision, and visual definitions. For example, left_wheel has mass five units, and 3x3 rotational matrix only diagonal elements with one. In collision properties, the geometry is cylinder with radius 0.1 meter and length 0.05 meter. Material visual property specifies it with green color.

```
  <link name='left_wheel'>
    <inertial>
      <mass value="5.0"/>
      <origin xyz="0.0 0 0" rpy=" 0 1.5707 1.5707"/>
      <inertia
          ixx="0.1" ixy="0" ixz="0"
          iyy="0.1" iyz="0"
          izz="0.1" />
    </inertial>
    <collision name='collision'>
      <origin xyz="0 0 0" rpy=" 0 1.5707 1.5707"/>
      <geometry>
        <cylinder radius=".1" length="0.05"/>
      </geometry>
    </collision>
    <visual name='left_wheel_visual'>
      <origin xyz="0 0 0" rpy=" 0 1.5707 1.5707"/>
      <geometry>
        <cylinder radius=".1" length="0.05"/>
      </geometry>
      <material name="green">
          <color rgba="0 1.0 0 1.0" />
      </material>
    </visual>
```

</link>

For Hokuyo laser scanner, it has an image mesh from ”package://udacity_bot/meshes/hokuyo.dae”.

Under ~/catkin_me/src/udacity_bot/urdf/udacity_bot.gazebo, it defines Gazebo plugin. Libgazebo_ros_diff_drive.so is a C++ shared library.

```
<gazebo>
    <plugin name="differential_drive_controller"
filename="libgazebo_ros_diff_drive.so">
      <legacyMode>false</legacyMode>
      <alwaysOn>true</alwaysOn>
      <updateRate>10</updateRate>
      <leftJoint>left_wheel_hinge</leftJoint>
      <rightJoint>right_wheel_hinge</rightJoint>
      <wheelSeparation>0.4</wheelSeparation>
      <wheelDiameter>0.2</wheelDiameter>
      <torque>10</torque>
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <robotBaseFrame>robot_footprint</robotBaseFrame>
    </plugin>
  </gazebo>
```

URDF tutorial on the ROS wiki (5) illustrates how to create a robot from scratch. Objects and images of poll, grippers, and gripper fingers were obtained from the example and put in my robot as shown in Fig. 1.

Both structures of link and join are straight forward. Pole, gripper, and finger are all defined in links. Join connects a link to its parent by specifying the origin of the child in the parent parent frame. For example, in gripper_extension

<origin rpy="0 0 0" xyz="0 0 0.1”/>

specifies the origin of poll is in the <0, 0, 0.1> of its parent chassis.

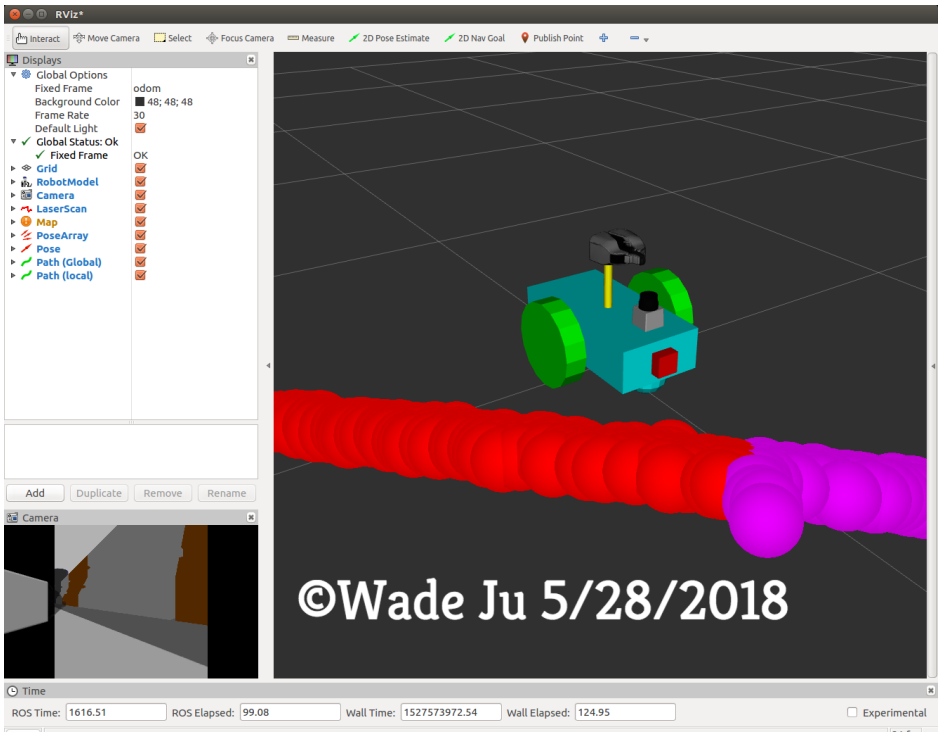

Fig. 1 Robot on RViz

3. AMCL and Navigation Packages

Then we need to setup map with AMCL for localization and Navigation package to move robot around obstacles to the target.

Copy jackal_race.pgm and jackal_race.yaml from https://github.com/udacity/RoboND-Localization-Project/tree/master/maps to ~/catkin_me/src/udacity_bot/urdf/maps. And set argument world_name in ~/catkin_me/src/udacity_bot/launch/udacity_world.launch to jackal_race.world.

AMCL dynamically adjusts the number of particles as the robot navigates around in a map. This adaptive process offers a significant computational advantage over MCL primarily in computation. Minimum particles are set to 10 and maximum particles are set to 80. Higher number particulars are better to work with unknown location like kidnapped or catastrophe.

Amcl.launch is created under ~/catkin_me/src/udacity_bot/launch. It contains map server, localization, laser, odom, and move base configurations. For move base configuration, there are four configuration files under directory ~/catkin_me/src/udacity_bot/config.

```
Let's launch it with the following command.

$ roslaunch udacity_bot udacity_world.launch

In a new terminal, launch AMCL model as well.
$ roslaunch udacity_bot amcl.launch

Then you can see Gazebo as Fig.2 and RViz as Fig. 3
```

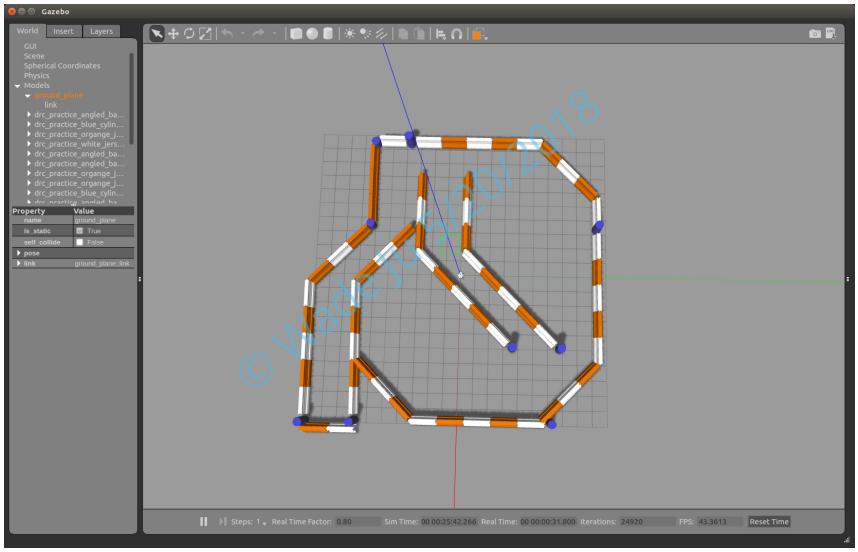

Fig. 2 Gazebo shows race track

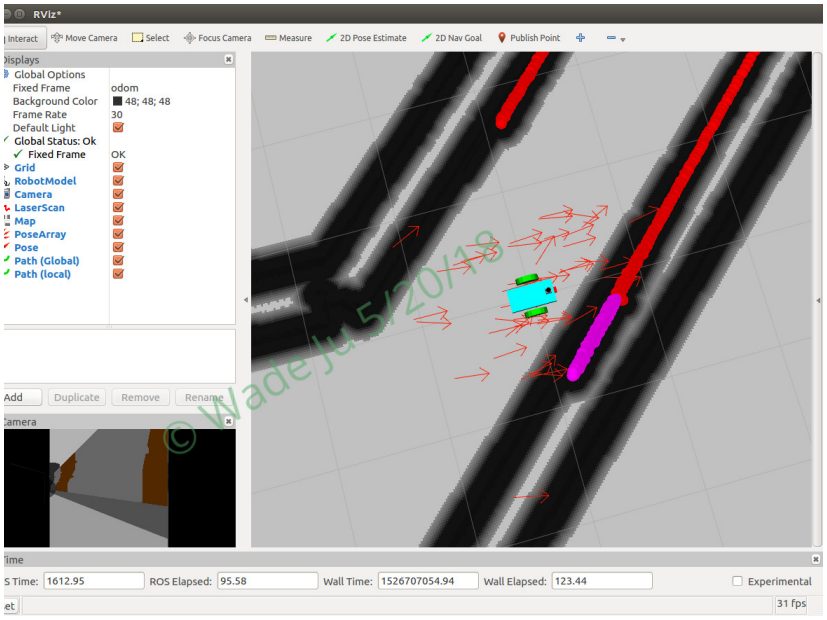

Fig. 3 RViz shows Robot in Race Track

Base_local_planner_params.yaml defines parameters for TrajectoryPlannerROS including (x,y, rotational) acceleration limitation, maximum and minimum (x,y, rotational) velocity, holonomic or non-holonomic robot, goal tolerance parameters, trajectory scoring parameters, forward simulation parameters, etc.

Holonomic_robot is set to true. A holonomic robot has the same number of controllable degrees of freedom as total degrees of freedom. Most default values are good to use. Max x velocity is set to 0.45 meters/sec instead of default 0.5. Smaller value makes it easier to control. Escape velocity is set to -0.2 meters/sec to give it a higher velocity to backup to escape.

Forward-simulate trajectories time (sim_time) is set to two seconds. Longer time period gives better results as long as computation powder supports. Two seconds performed better than the default one second in testing. Controller frequency is set to 10Hz. The testing computer can't keep up at the default 20Hz. Meter scoring is turned on. It uses a formula to calculate cost of trajectory in considerations of distance to local goal, to end point of planned trajectory, and to obstacle. Occdist_scale is raised from 0.01 to 0.1 that helps from avoiding obstacles. Heading_lookahead is raised from 0.325 meters to 0.5 meters to give a longer lookahead for trajectory. Again, this is bound to computation capability of CPU/GPU.

costmap_common_params.yaml defines common parameters for both global and local cos-maps for example obstacle_range for obstacle thresholds in the cos-map . Here are properties that set in costmap_common_params.yaml.

obstacle_range: 2.5
raytrace_range: 3.0
transform_tolerance: 0.3
robot_radius: 0.1

inflation_radius: 0.2
observation_sources: laser_scan_sensor
laser_scan_sensor: {sensor_frame: hokuyo, data_type: LaserScan, topic: /udacity_bot/laser/scan, marking: true, clearing: true}

Global_costmap_params and local_costmap_params define properties for global and local respectively. global_costmap_params sets global frame to map. local_costmap_params sets global_frame to odom. Both set update_frequency and publish_frequency to 10. It would be better to set them to 20Hz as default control frequency if hardware can keep up in computation. Height and width are set to 20 meters. bigger area has high impact on update and publish frequency. If commutation can't keep up to update and publish, system could be unstable.

Results: Navigation command sends commend to Robot to reach goal. By issuing the command on a new terminal,

$rosrun undacity_bot navigation

Robot has reached the goal as shown in Fig. 4 and Fig. 5.

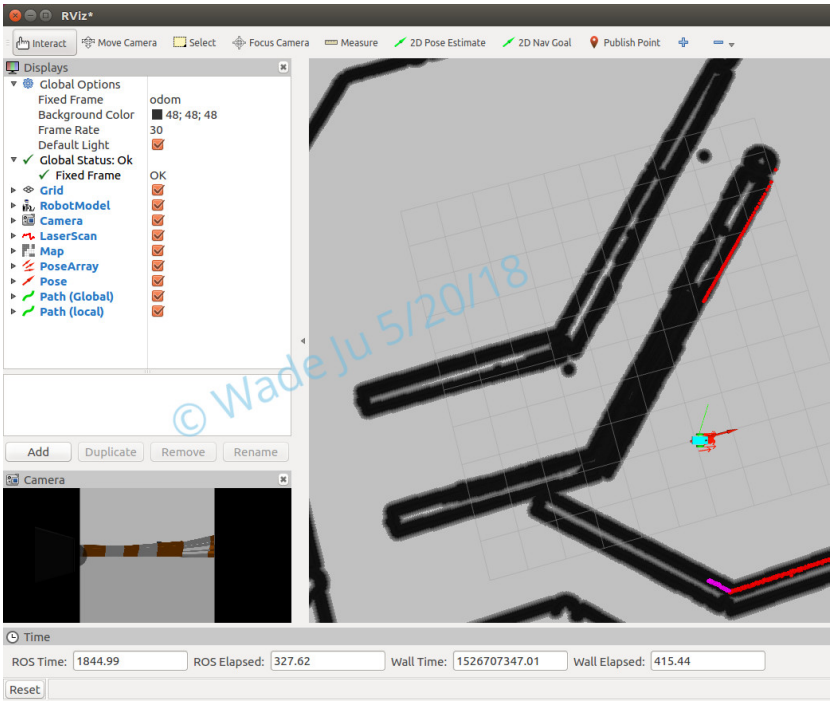

Fig. 4 Robot Reaches Goal (RViz)

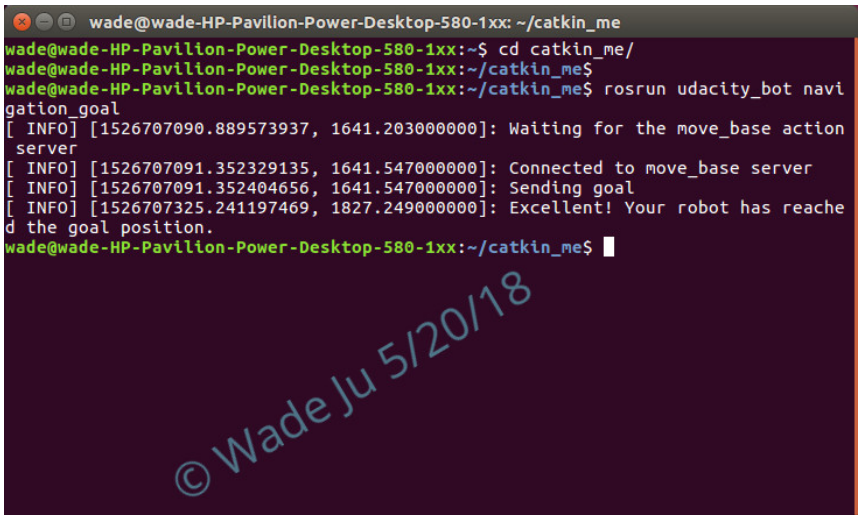

Fig. 5 Robot Reaches Goal (Terminal)

Discussion: To run AMCL and navigation stack take quite a lot computation power as well as rendering on Gazebo and RViz. When I was working on Ubuntu VM (VMWare) on Mac, I got bogus screen like Fig. 6. After installing Ubuntu 16 with ROS packages in a PC, things went back to normal.

AMCL(6) has computation advantage over MCL as it can adaptively adjust particles. Besides, it gives a visible feedback with small number of dense particles in a high certainty situation. For a kidnaped robot(7), AMCL might be able to get its location by adapting the number of particles. Again, here is another tradeoff in the maximum of particles of AMCL. Higher number of particles helps unknown situation like kidnapped or catastrophe in expense of computation.

Most of work of project is on ROS robot setup and parameter tuning. Parameter tuning is time consuming to tradeoff accuracy with computation power of the testing system. For example, scanning and publish frequency are set to 10Hz instead of 20 Hz. Longer look ahead distance can make trajectory and moving to avoid unnecessary errors. The size of local and global cost map has a significant impact on performance. A larger local cost map

makes a better coverage and accuracy; however, it costs more to publish topic. When publish can't keep up from the underline CPU, system could be unstable.

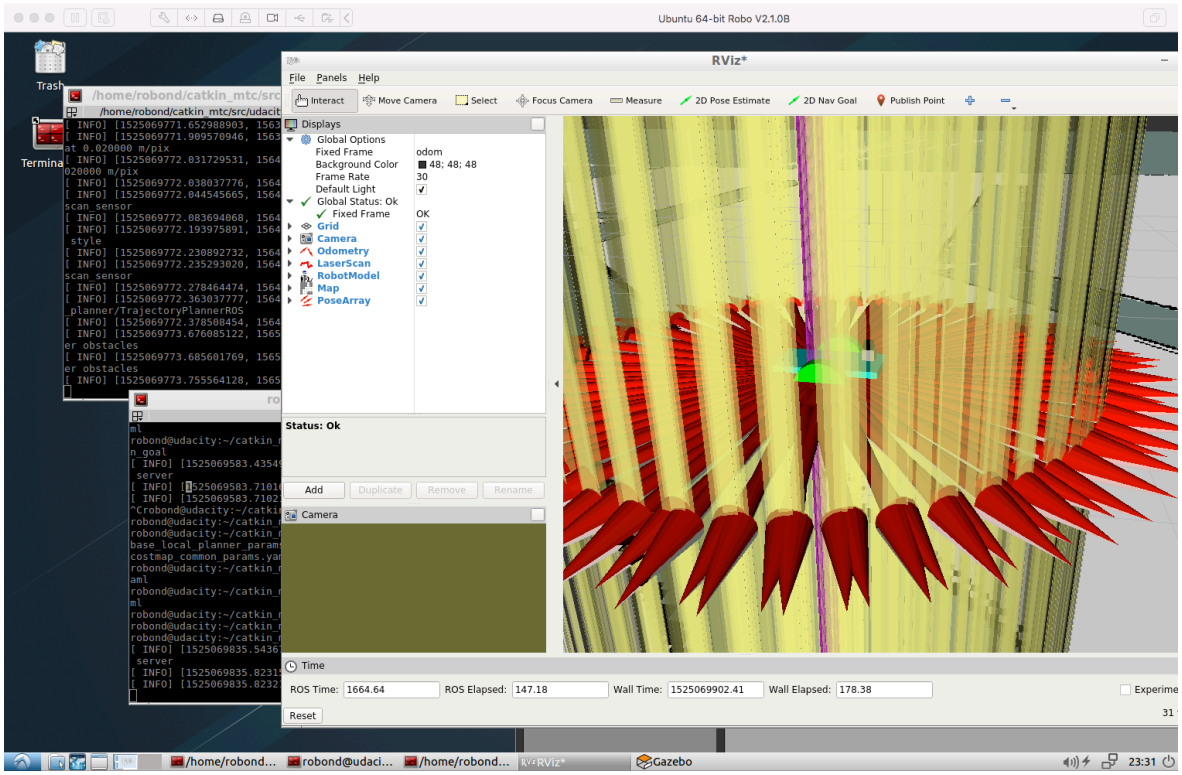

Fig. 6 RViz Rendering problem on VM

Conclusion / Future Work:

One problem of the project is robot usually starts moving in the wrong direction. It can go back and finally reaches the goal. However, it could be optimized for the shortest route and possibly

shortest time as well. In certain situation, shortest route and shortest time to reach goal might be different.

Turing parameters is a labor intensive job. The parameters that I got in the project are not optimal. Later on, we are going to learn reenforcement learning in the program. By applying reenforcement learning, it could be a better approach to get optimal parameters.

To run it on hardware is another task for future. With difference in environment and hardware, potentially some parameters need to be adjusted.

Only camera and laser range finder are used in the project for localization and navigation. Future work can add more sensors like IMU, RADAR, infra-red, etc. Computation capability is also an important factor. An AMD Ryzen 1700 CPU with 16 GB memory native Ubuntu 16.04 that I have can’t fully support the system with 20Hz in Gazebo and rViz.

References

1. http://osrf-distributions.s3.amazonaws.com/sdformat/api/dev.html
2. http://wiki.ros.org/navigation/Tutorials
3. http://wiki.ros.org/amcl
4. http://wiki.ros.org/navigation
5. http://wiki.ros.org/urdf/Tutorials
6. http://robots.stanford.edu/papers/fox.aaai99.pdf
7. http://robots.stanford.edu/papers/thrun.robust-mcl.pdf