RoboND-Localization-Project Where Am I?

Noriaki Ibe

Abstract—The second project in term 2 of the Robotics Software Engineer Nanodegree Program requires students to utilize ROS packages to accurately localize a mobile robot inside a provided map in the Gazebo and RViz simulation environments.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, deep learning.

1 Introduction

THE Localization is one of the main part in Robotics and those could be classed as fixed base such as Kuka equipments etc. and moving base such as iRobot etc. Of course, Auto-driving car shall be classed as the moving base too. The project has several aspects of robotics with a focus on ROS, including -

- Building a mobile robot for simulated tasks.
- Creating a ROS package that launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack.
- Exploring, adding, and tuning specific parameters corresponding to each package to achieve the best possible localization results.

Fig. 1. Fixed Base

Fig. 2. Moving base (c) 2018 iRobot Corporation

2 Background / Formulation

The project requires students to perform:

1) Follow the steps outlined in this Project Lesson, to create own ROS package, develop a mobile robot model for Gazebo, and integrate the AMCL and Navigation ROS packages for localizing the robot in the provided map.
2) Using the Parameter Tuning section as the basis, add and tune parameters for ROS packages to improve localization results in the map provided.
3) implement own robot model with significant changes to the robot's base and possibly sensor locations.
4) improve upon localization results as required for this new robot.

The steps outlined in this Project are follows and it's conducted in the Udacity provided Project Workspace:

2. Gazebo: Hello, world! This section includes creating a new empty package, a world file using the XML file format to describe all the elements that are being defined with respect to the Gazebo environment.in Gazebo, and a launch file in ROS allow us to execute more than one node simultaneously.

3. Robot Model: Basic Setup This section is creating a Robot URDF. The part for Robot base with costar wheels is provided and A robot actuation part such as SIDE wheels must be added. The params for SIDE wheels are follows:

- link name - "SIDE_wheel", where the SIDE is either left or right.

- geometry - ”cylinder” with radius 0.1 and length 0.05.
- Origin for each element - [0, 0, 0, 0, 1.5707, 1.5707]
- mass of each wheel - ”5”.
- inertia values as the ones for the chassis for simplicity:

```
ixx="0.1" ixy="0" ixz="0"
iyy="0.1" iyz="0"
izz="0.1"
```

- the code creating a joint between left wheel (the child link) and the robot chassis (the parent link).

```
<joint type="continuous"
          name="left_wheel_hinge">
  <origin xyz="0 0.15 0" rpy="0 0 0"/>
  <child link="left_wheel"/>
  <parent link="chassis"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
  <limit effort="10000" velocity
      ="1000"/>
  <dynamics damping="1.0" friction
      ="1.0"/>
</joint>
```

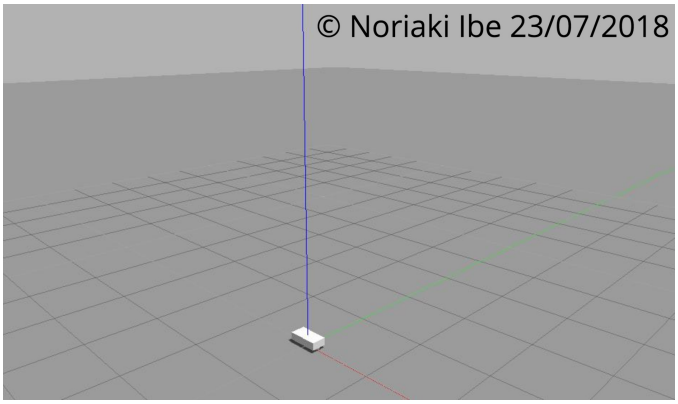

Fig. 3. Robot base with costar wheels

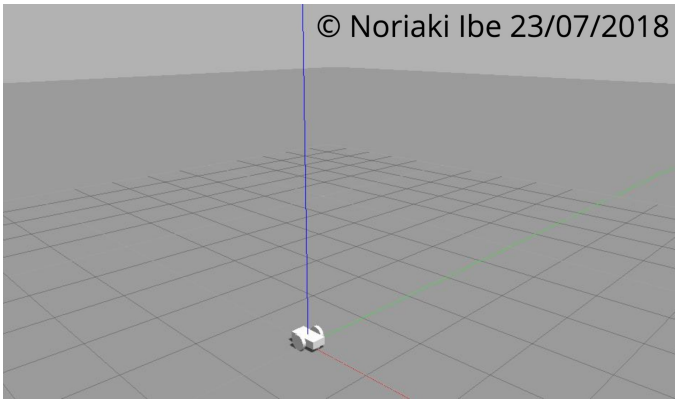

Fig. 4. Robot base with SIDE wheels

4. Robot Model: Basic Setup This section adds Robot Sensors such as Camera and Laser Rangefinder and params are given:

- link name - ”camera”
- link origin - ”[0, 0, 0, 0, 0, 0]”
- joint name - ”camera_joint”
- joint origin - ”[0.2, 0, 0, 0, 0, 0]”
- geometry - box with size ”0.05”
- mass - ”0.1”
- inertia - ixx=”1e-6” ixy=”0” ixz=”0” iyy=”1e-6” iyz=”0” izz=”1e-6”
- joint parent link - ”chassis”, and joint child link - ”camera”

- link name - ”hokuyo”
- link origin - ”[0, 0, 0, 0, 0, 0]”
- joint name - ”hokuyo_joint”
- joint origin - ”[.15, 0, .1, 0, 0, 0]”
- geometry - box with size ”0.1” for ¡collision¿, and a mesh file for ¡visual¿
- mass - ”0.1”
- inertia - ixx=”1e-6” ixy=”0” ixz=”0” iyy=”1e-6” iyz=”0” izz=”1e-6”

Additionally, gazebo file is given for Gazebo Plugins.

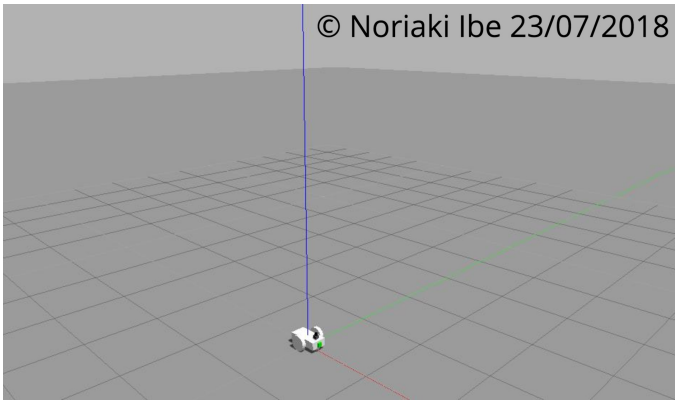

Fig. 5. Built Robot

5. RViz Integration This section guides how RViz can visualize any type of sensor data being published over a ROS topic like camera images, point clouds, Lidar data, etc, while Gazebo is a physics simulator, instructions to modify robot_description file and udacity_world.launch file are given. and steps to display is follows:

Select the RViz window, and on the left side, under Displays:

- Select odom for fixed frame
- Click the Add button and
  - add RobotModel
  - add Camera and select the Image topic that was defined in the camera gazebo plugin
  - add LaserScan and select the topic that was defined in the hokuyo gazebo plugin.

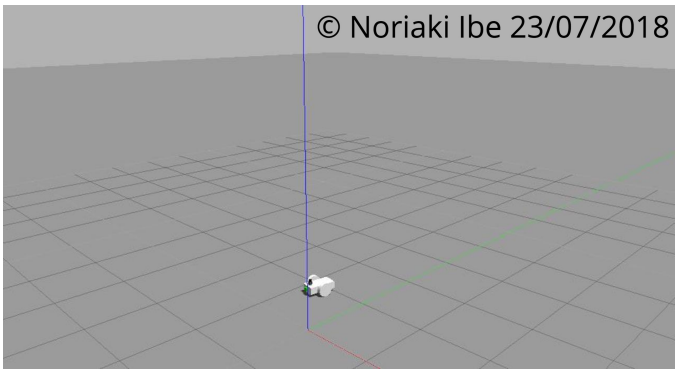

Fig. 6. Robot in Gazebo

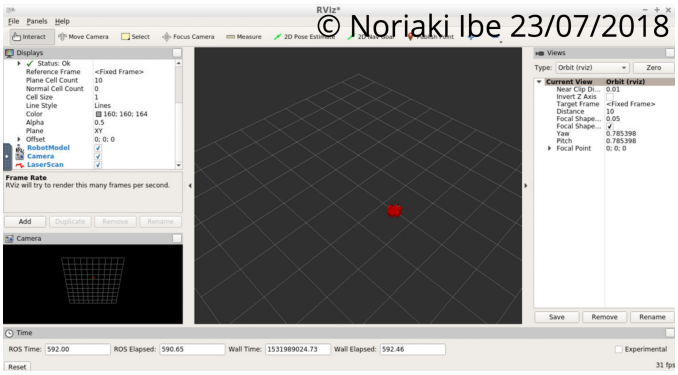

Fig. 7. Robot in RViz

6. Localization: Map This section guides how to add a Map by Clearpath Robotics for robot in a new environment. Two files jackal_race.pgm and jackal_race.yaml are given.

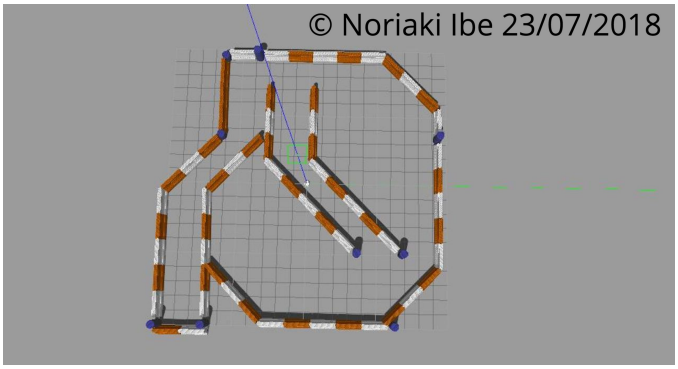

Fig. 8. Robot in Map

7. Localization: AMCL Package Adaptive Monte Carlo Localization (AMCL) dynamically adjusts the number of particles over a period of time, as the robot navigates around in a map. This adaptive process offers a significant computational advantage over MCL. The ROS amcl package implements this variant and you will integrate this package with robot to localize it inside the provided map. This section guides how to apply The ROS amcl, move_base package to define a goal position for robot in the map, and the robot will navigate to that goal position. Some configuration files for parameters or definitions pertaining to the costmaps as well as the local planner that creates a path and navigates the robot along that path are given.

- local_costmap_params.yaml
- global_costmap_params.yaml
- costmap_common_params.yaml
- base_local_planner_params.yaml

8. 9. Localization: Parameter Tuning - 1 , 2 Section 3 Parameter Tuning and Own Model explains about those topics.

10. Launching and Testing This section guides how to point the goal in RViz with using the 2D Nav Goal button in RViz toolbar. For Launching, a robot needs to navigate to a specific goal position while localizing itself along the way and C++ node navigating the robot to the goal position are given. Finally, CMakeLists.txt is given, then following steps should be taken to launch and run Ros.

```
### 1st Terminal
$ cd /home/workspace/catkin_ws
$ catkin_make
$ source devel/setup.bash
$ roslaunch udacity_bot udacity_world.launch

### 2nd Terminal
$ cd /home/workspace/catkin_ws
$ source devel/setup.bash
$ roslaunch udacity_bot amcl.launch

### 3rd Terminal
$ cd /home/workspace/catkin_ws
$ source devel/setup.bash
$ rosrun udacity_bot navigation_goal
```

The project has two files catkin_ws_udacity_bot.tar.gz and catkin_ws_v2.tar.gz. for Udacity_bot and own model. The one of those file shall be expanded under /home/workingspace, then the above shall be executed. (Don't forget 5.RViz Integration as the above.)

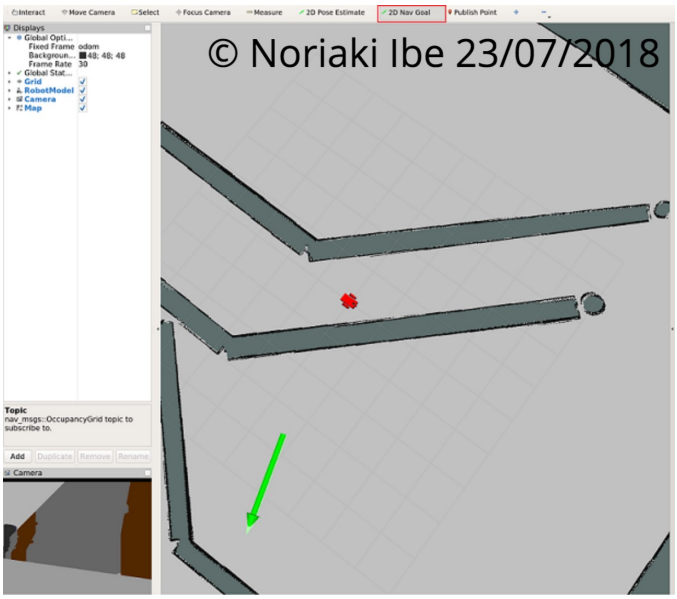

Fig. 9. The arrow, as indicated above, will represent the goal position, and the direction will represent the goal orientation for the robot.

3 Parameter Tuning and Own Model

Parameter Tuning The first task is to add and tune this parameter for both the amcl node in amcl.launch file and for the move_base node in the costmap_common_params.yaml file. Due to documentation ROS:AMCL,

```
~transform_tolerance (double, default: 0.1
    seconds)
Time with which to post-date the transform
    that is published, to indicate that this
     transform is valid into the future.
```

So this project tried from 0.1, 0.2 and 0.3 and 0.3 was fine.

Additionally, following params were set as follows:

- obstacle_range - For example, if set to 0.1, that implies that if the obstacle detected by a laser sensor is within 0.1 meters from the base of the robot, that obstacle will be added to the costmap.
-
- raytrace_range - This parameter is used to clear and update the free space in the costmap as the robot moves.
-
- inflation_radius - This parameter determines the minimum distance between the robot geometry and the obstacles.

Due to documentation ROS: costmap_2d,

```
~<name>/<source_name>/obstacle_range (double
    , default: 2.5)
The maximum range in meters at which to
    insert obstacles into the costmap using
    sensor data.
~<name>/<source_name>/raytrace_range (double
    , default: 3.0)
The maximum range in meters at which to
    raytrace out obstacles from the map
    using sensor data.
```

Additionally, Due to documentation ROS:xnavigation, inflation_radius was set around 0.5. So this project set those params as follows:

```
obstacle_range: 2.0
raytrace_range: 5.0
inflation_radius: 0.5
```

raytrace_range was set a bit bigger than default for covering wider by sensor data after several iterations.

Furthermore, Due to documentation ROS:base_local_planner and discussion Ros Answer about the controller frequency for move_base when using the navigation stack, base_local_planner_params.yaml was set as follows:

```
rajectoryPlannerROS:

  holonomic_robot: false
  sim_time: 1.0
  meter_scoring: false
  pdist_scale: 0.6
  yaw_goal_tolerance: 0.05
  xy_goal_tolerance: 0.05
  controller_frequency: 10
  acc_lim_x: 2.5
  acc_lim_y: 2.5
  acc_lim_theta: 3.2
  max_vel_x: 0.5
  min_vel_x: 0.1
  max_vel_theta: 1.0
  min_vel_theta: -1.0
  min_in_place_vel_theta: 0.4
```

Finally, update_frequeny and publish_frequency in local_costmap_params.yaml and global_costmap_params.yaml must be set. It's because a larger and more detailed map will result in a large global costmap, and would use more resources. It should be modified help free up some resources. Then, the project set it as 7 from 50 to reduce.

Regarding Robot Localization,

Due to documentation ROS:AMCL and lecture on Udacity following params are set on amcl.launch.

- Overall Filter
  - min_particles(int, default: 100) and max_particles(int, default: 5000) - As amcl dynamically adjusts its particles for every iteration and shall be reduced if it's too computationally extensive.
  - initial_pose - For the project, it should be set the position to [0, 0].
  - update_min* - amcl 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. (the project doesn't set it)
- Laser
  - 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 range finder sensor estimates the obstacles in relation to the robot.
  -
  - The likelihood_field model is usually more computationally efficient and reliable for an environment such as the one you are working with. So you can focus on parameters for that particular model such as the -
  -
  - laser_*_range
  - laser_max_beams
  - laser_z_hit and laser_z_rand
- Odometry
  - odom_model_type - Since the project are working with a differential drive mobile robot, its best to use the diff-corrected type. There are additional parameters that are specific to this type - the odom_alphas (1 through 4).

```
<!-- Localization: Parameter Tuning - 2 -->
    <param name="min_particles" value="10"/>
```

```
<param name="max_particles" value
    ="200"/>

<param name="initial_pose_x" value="0"/>
<param name="initial_pose_y" value="0"/>
<param name="initial_pose_a" value="0"/>

<param name="laser_model_type" value="
    likelihood_field"/>
<param name="laser_z_hit" value="0.99"/>
<param name="laser_z_rand" value
    ="0.01"/>

<param name="odom_alpha1" value
    ="0.010"/>
<param name="odom_alpha2" value
    ="0.010"/>
<param name="odom_alpha3" value
    ="0.010"/>
<param name="odom_alpha4" value
    ="0.010"/>
```

Own Model The project requires the students to build an own model. Basically, the project has modified only following files:

- udacity_bot.xacro
- udacity_bot.gazebo
- costmap_common_params.yaml
- base_local_planner_params.yaml

udacity_bot.xacro was modified especially,

- Chassis Link
- Add color
- Change sensors (camera and hokuyo) location (see origin of joints)

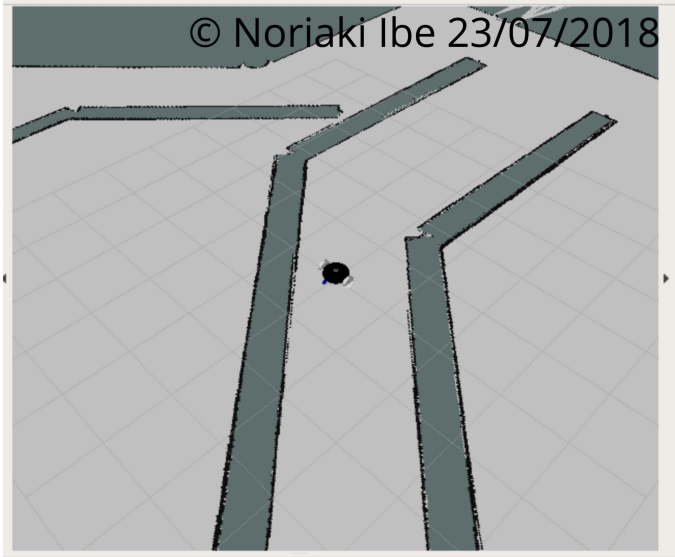

Fig. 10. Own Robot.

Before Parameter Tuning, the project just run Ros as same as the robot given from Udacity. Then, there was the stuck point as follows:

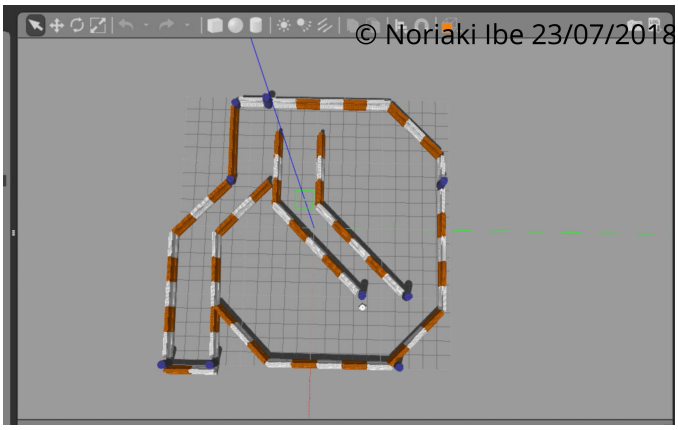

Fig. 11. Stuck Point

Especially, it seems that there is difficulty to pass the wall (obstacles). Then, the following params were made bigger a bit.

- obstacle_range
- raytrace_range
- inflation_radius

4 RESULTS

The both of the robot given by Udacity and the own robot could reach the goal anyhow. The following pictures are the result as step by step such as Start ¿ Set Goal ¿ Goal on Gazibo and RViz. In fact(please check the link of Youtube), it seems the Robot had been making a detour a bit to pass the obstacle as mentioned the last section. But, It shall be OK.

The Robot given by Udacity

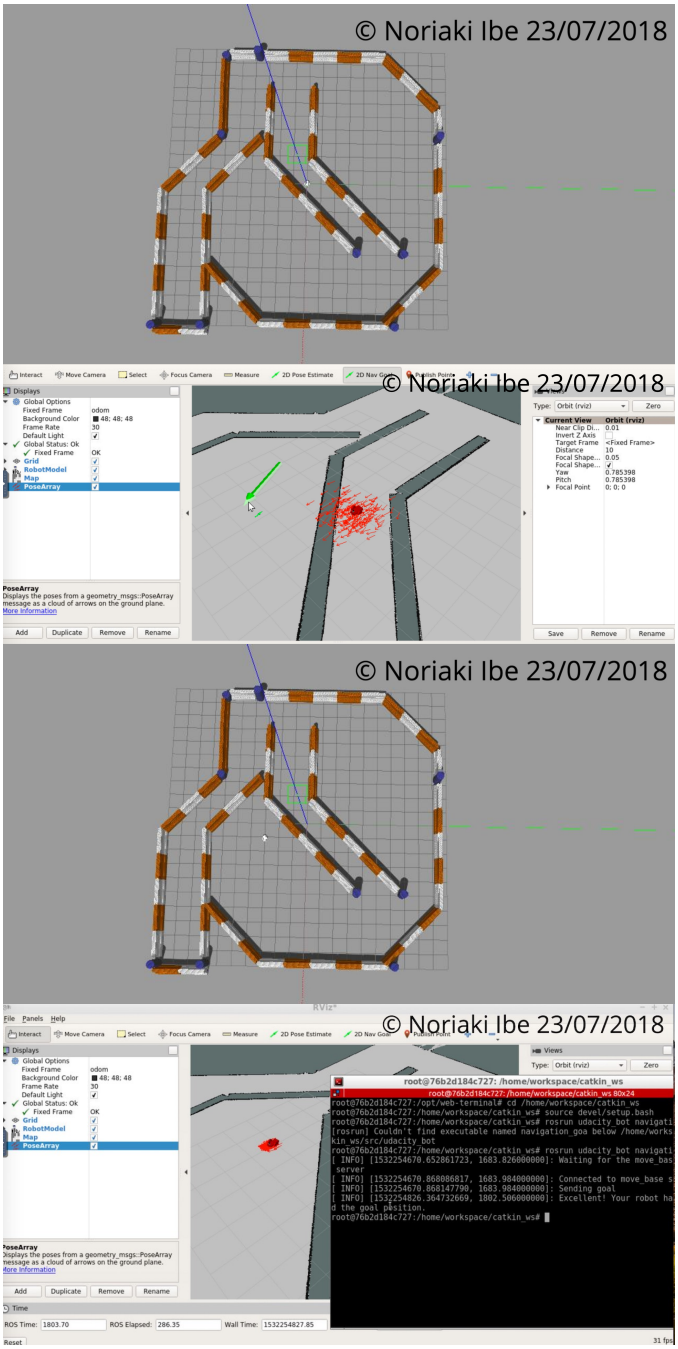

Fig. 12. Start¿Set Goal¿Goal on Gazibo and RViz

It's the linkof Youtube for the above.

The Own Robot

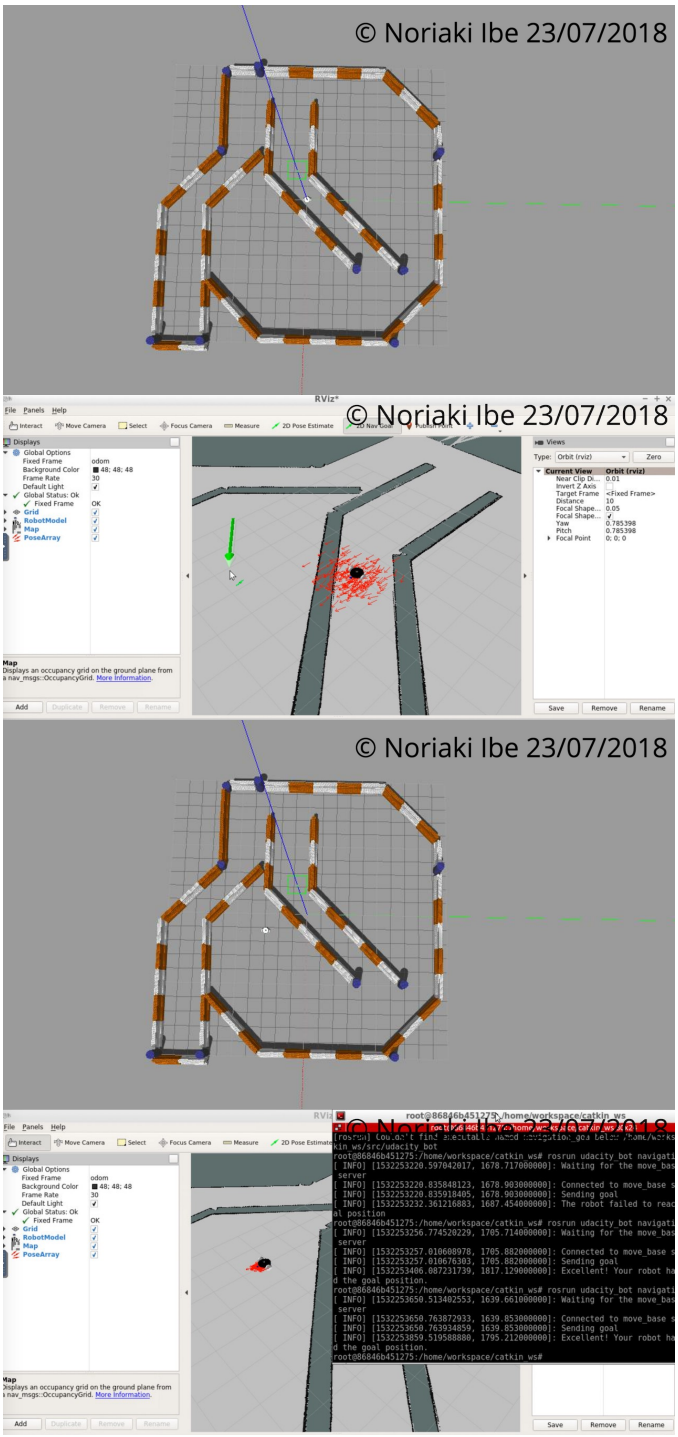

Fig. 13. Start¿Set Goal¿Goal on Gazibo and RViz

It's the link of Youtube for the above.

5 Discussion

As the robot was stuck, there are several difficulty to reach goal especially in passing obstacles. And it depends on cases whether the robot focus on the shortest path or the safety and accuracy in enough time. The project shows the importance of Parameter Tuning, it's because it must be done even the project changes base and location sensor a bit. Then, ROS has great documentation so that everyone should reference to those especially those documents has default number for each params. Those should be great helpful to build own robot.

6 Conclusion / Future work

The project could perform at least as the requirements stated on Background / Formulation. Especially, it was the great practice to used to the ROS environments. Of course, there are a lot of part which should be improved without adding any other function. It shall be possible to reach goal smoothly more. So, improving current model shall be done as Future work, but it shall be done to combine with the first project. The students has learn how to use Robotic Inference so that it shall be interesting to use input data from camera into Robotic Inference to detect something and making decision.