FEBRUARY 11, 2019

RoboND-Localization-Project

ABSTRACT

This project is to build robot model from scratch to localize robot inside provided map.

Robot model is built using ROS, simulated in gazebo & RViz. Monto-carlo localization algorithm is used to localize robot and obtain the map.

Camera, laser scanners and encoders was used to build monto-carlo algorithm.

Workspace used for this project is Ubuntu with ROS-kinetic installed.

INTRODUCTION

Although, some project are easy to be implement and tested without simulation.

There is on the other side projects that are complex or can cause high material & resources loss if not tested properly in simulation before implementing them in real life.

Gazebo & rViz are powerful tools that can be used for simulation in any robotics project due to the simplicity of using them with ROS, powerful tools & advantages they have.

This project focus on building the basic blocks in any robotic simulation project and implementing mont-carlo algorithm to work efficiently in your robot.

BACKGROUND / FORMULATION

This is high level knowledge of techniques that can be used in localization projects.

1- KALMAN-FILTER LOCALIZATION

"The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown." (G.Welch and G. Bishop, 2004)

ADVANTAGES

The Kalman filter is a very powerful tool when it comes to controlling noisy systems because:

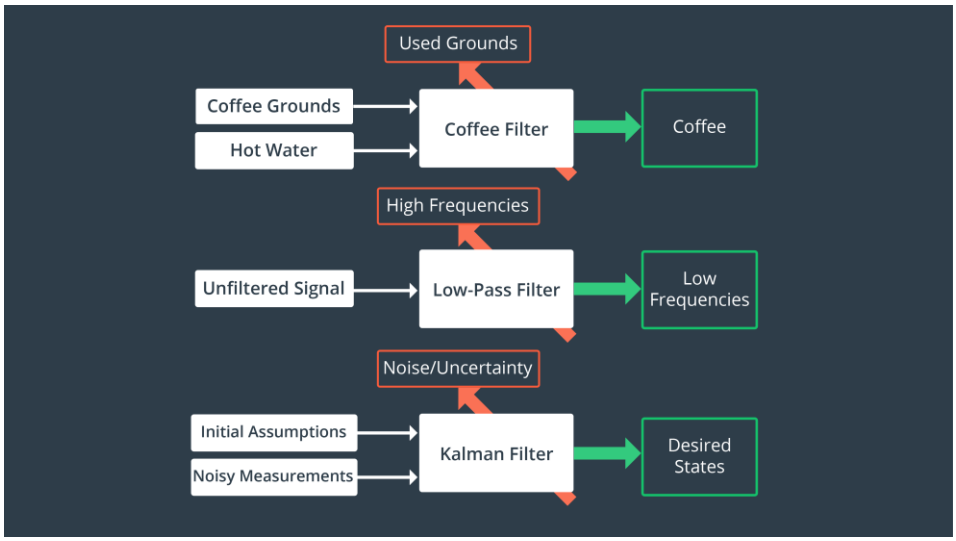
- It can develop a surprisingly accurate estimate of the true value of the variable being measured
- Requires less information
- Can produce accurate data after few measurements
- Can be used with sensor fusion to produce very accurate data

2- MONTO-CARLO LOCALIZATION

A Particle filter is another great choice for localization if a robot system doesn't fit nicely into a linear model, or a sensor uncertainty doesn't look very Gaussian. A Particle Filter make it possible to handle almost any kind of model, by discretizing the problem into individual "particles" – each one is basically one possible state of your model, and a collection of a sufficiently large number of particles lets you handle any kind of probability distribution, and any kind of evidence (sensor data).the estimation error in a particle filter does converge to zero as the number of particles (and hence the computational effort) approaches infinity. Particle filter is also called Monte Carlo Localization (MCL).

ADVANTAGES

- Can be used for non-linear systems
- Can deal filter can deal with non-Gaussian noise distribution
- Can often surprisingly make simple implementations for complex nonlinear systems



SETTING WORKSPACE

BUILDING NEW PACKAGE

Let's start by navigating to the `src` directory and creating a new empty package.

```
$ cd /home/workspace/catkin_ws/src/
$ catkin_create_pkg udacity_bot
```

CREATING NECESSARY DIRECTORIES

```
$ cd udacity_bot
$ mkdir launch
$ mkdir worlds
```

CREATING EMPTY WORLD

This results simple world, with no objects or models that will be launched later in Gazebo.

```
$ cd worlds
$ nano udacity.world
```

Add the following to `udacity.world`

```
<?xml version="1.0" ?>

<sdf version="1.4">

  <world name="default">

    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- Light source -->
    <include>
      <uri>model://sun</uri>
    </include>
```

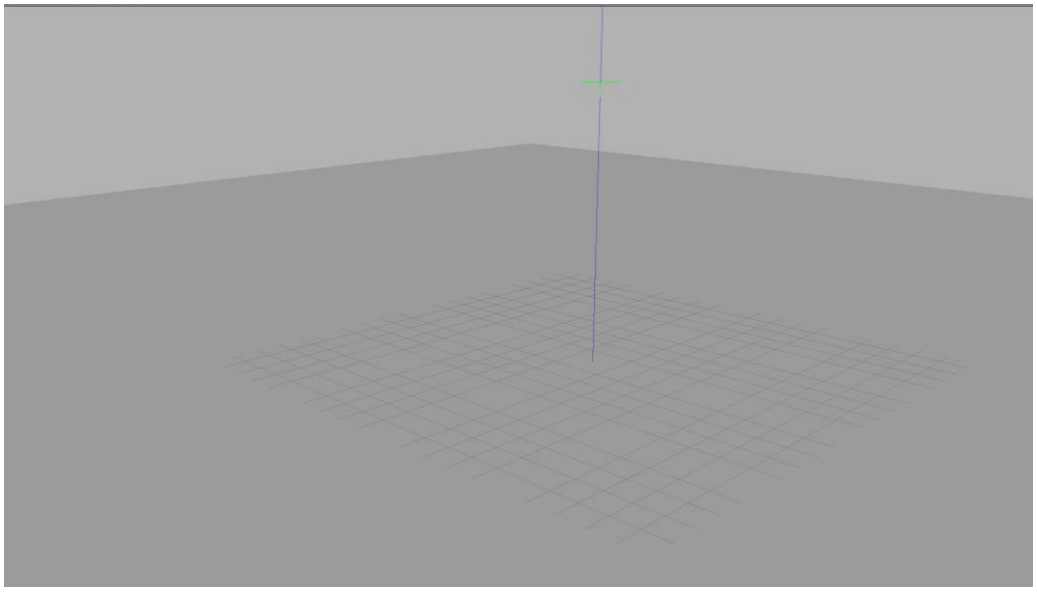
```
    <!-- World camera -->
    <gui fullscreen='0'>
      <camera name='world_camera'>
        <pose>4.927360 -4.376610 3.740080 0.000000 0.275643 2.356190</pose>
        <view_controller>orbit</view_controller>
      </camera>
    </gui>

  </world>
</sdf>
```

CREATING LAUNCH FILES

Launch file is made to launch the empty gazebo world and set necessary arguments

RESULT



BUILDING ROBOT MODEL

CREATE NECESSARY DIRECTORIES &FILES

```
$ cd /home/workspace/catkin_ws/src/udacity_bot/
$ mkdir urdf
$ cd urdf
$ nano udacity_bot.xacro
```

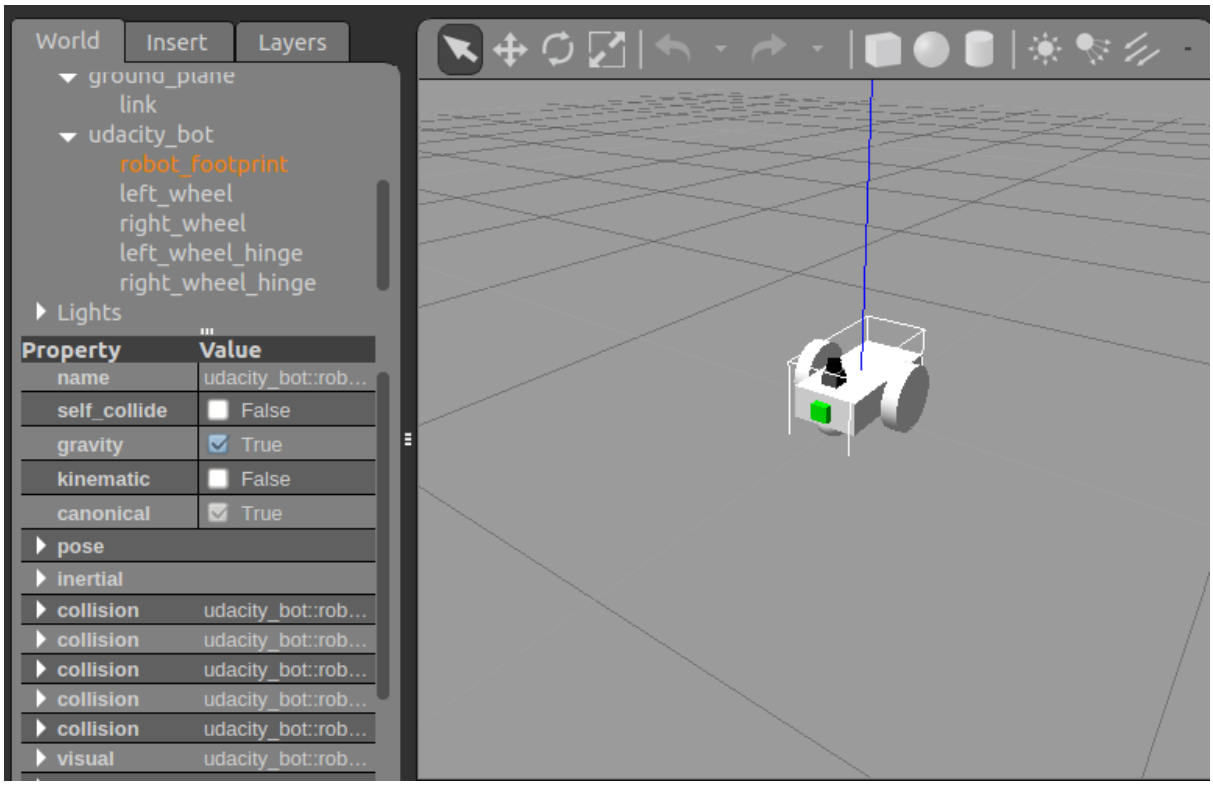
CHOOSING MODEL

Model used here consist of a cuboidal base with two caster wheels. The caster wheels will help stabilize this model. They aren't always required, but it can help with weight distribution, preventing your robot from tilting along the z-axis at times.

BUILDING CHASIS

Urdf is modified to build chasis model.

- Chassis link is created and collision, inertial and visual data is set.
- Front and back caster wheel visual and collision data is added
- Chassis size is box with “4 .2 .1" dimension

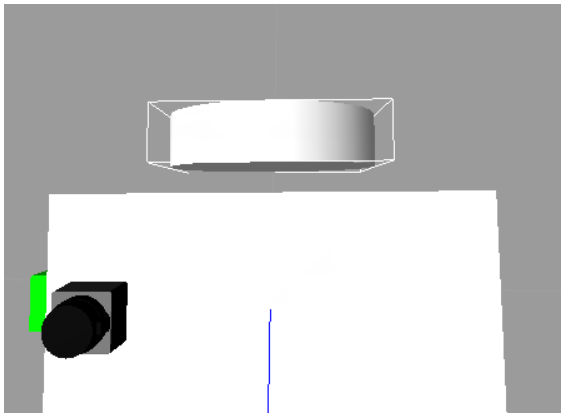


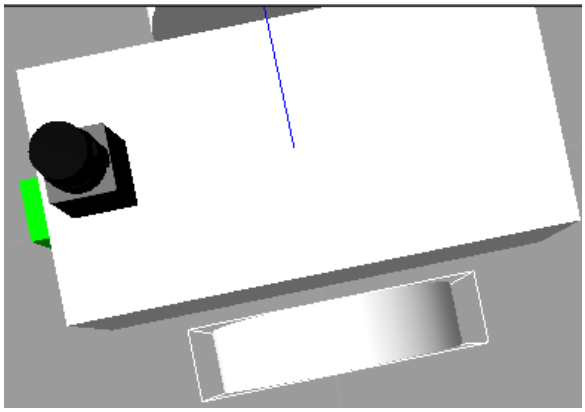
BUILDING DIFFERENTIAL WHEELS

- To create two differential wheels first two links are created

- Inertial, visual and collision properties and parameter is modified to create wheels properties

- Once you have your links defined, you need to define the corresponding joints. The added code will create a joint between your left wheel (the child link) and the robot chassis (the parent link).



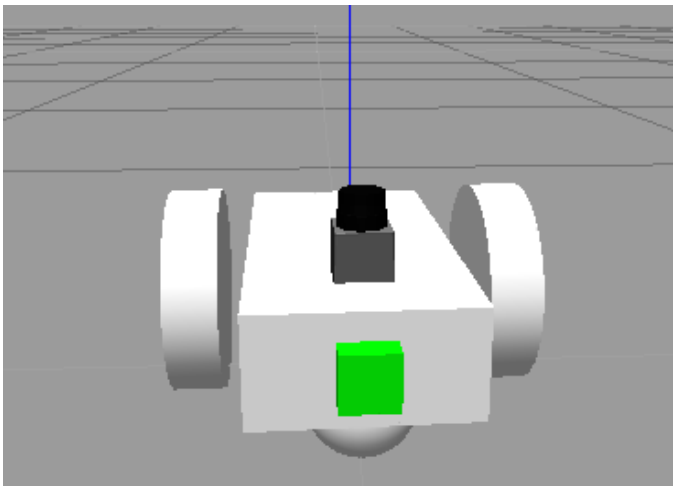


CREATING CAMERA LINK & JOINT

- creating new link called camera
- setting origin to [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"
- setting inertia

CREATING LASER LINK & JOINT

- `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"
- `joint type` = fixed



ADDING NECESSARY PLUGINS FOR

- camera
- hokuyo sensor
- controlling the wheels

SETTING RVIZ CONFIGURATION

MODIFY ROBOT DESCREPTION FILE

```
$ cd /home/workspace/catkin_ws/src/udacity_bot/launch/
$ nano robot_description.launch
```

Add the following after the first “param” definition.

```
<!-- Send fake joint values-->
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher">
    <param name="use_gui" value="false"/>
  </node>

<!-- Send robot states to tf -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="false" output="screen"/>
```

MODIFY THE UDACITY_WORLD LAUNCH FILE

Add the following at the end of the file. After the `urdf_spawner` node definition.

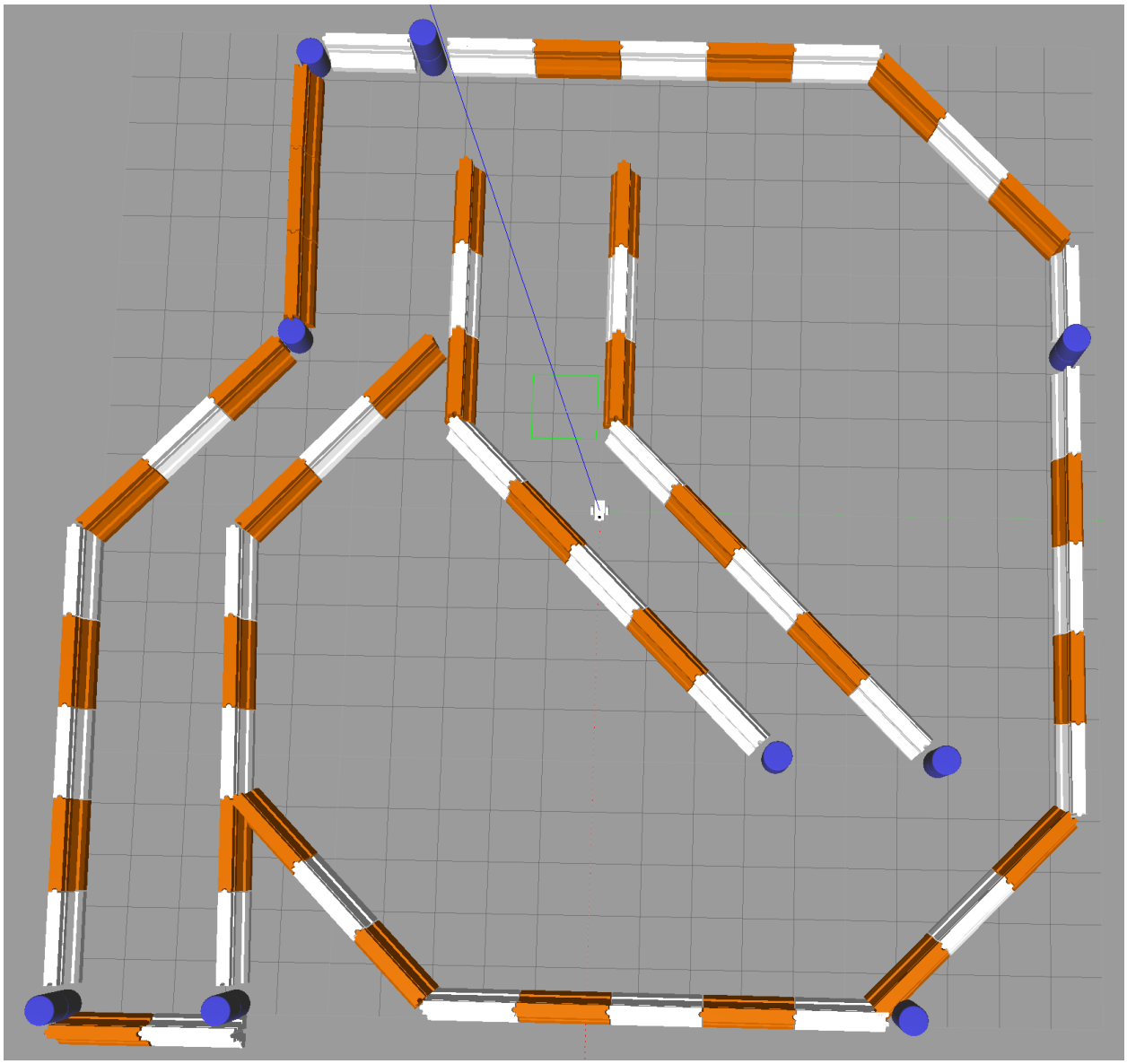
```
<!--launch rviz-->
<node name="rviz" pkg="rviz" type="rviz" respawn="false"/>
```

The above will create a node that launches the package `rviz`. Let's launch it.

RESULTS

ADD PROVIDED MAP

In this section, you will launch your robot in a new environment using a map created by Clearpath Robotics.



CREATING MAPS DIRECTORY

```
$ cd /home/workspace/catkin_ws/src/udacity_bot/
$ mkdir maps
$ cd maps
```

COPYING MAP FILES INTO MAPS FOLDER

MODIFY LAUNCH FILE

You will have to modify the udacity_world.launch file and update the path to this new map.

ADDING 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.

CREATE AMCL PACKAGE LAUNCH FILE

```
$ cd /home/workspace/catkin_ws/src/udacity_bot/launch/
$ nano amcl.launch
```

MODIFY AMCL PACKAGE

TUNNING PARAMETERS

There is some parameters that should be adjusted carefully to obtain efficient results

TRANSFORM TOLERANCE

Tuning the value for this parameter is usually dependent on your system and it causes errors that prevent robot form moving.

- Suitable value =0.2

1- UPDATE FREQUENCY & PUBLISH FREQUENCY

This parameter set to law value =10 Hz to suite system, but this decreases efficiency on the other hand.

2- 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. Tuning this parameter can help with discarding noise, falsely detecting obstacles, and even with computational costs.

- Suitable value=8

3- RAYTRACE RANGE

This parameter is used to clear and update the free space in the costmap as the robot moves.

- Suitable value =4.0

4- INFLATION RADIUS

This parameter determines the minimum distance between the robot geometry and the obstacles.

- Suitable value=0.4

5- OVERALL FILTER

- `min_particles` and `max_particles` - 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.
  - Min_partice suitable value=20
  - Max_particles suitable value =800

6- 7-INITIAL POSITION

For the project, you should set the position to [0, 0]. Feel free to play around with the mean yaw value.

7- UPDATE PARAMETERS

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. 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.

Update_min_a suitable value =0.1

Update_min_d suitable value=0.25

8- LASER TYPE

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.

The likelihood_field model is usually more computationally efficient and reliable for an environment such as the one you are working with.

Used type=" likelihood_field"

9- LASER MAX RANGE

Suitable value=20

10- LASER MAX BEAM

Suitable value=20

11- ODOM_MODEL_TYPE

Diff-corrected is chosen since we are working with a differential drive mobile robot

12- ODOM_ALPHAS

Left as they are

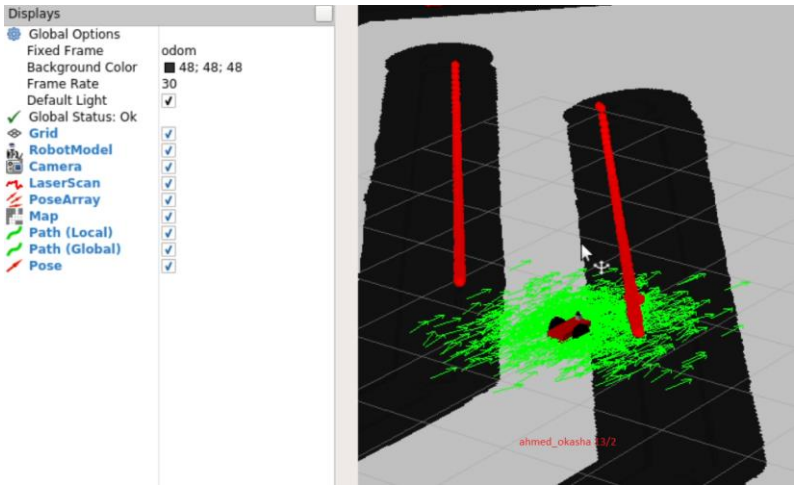
TESTING &RESULTS

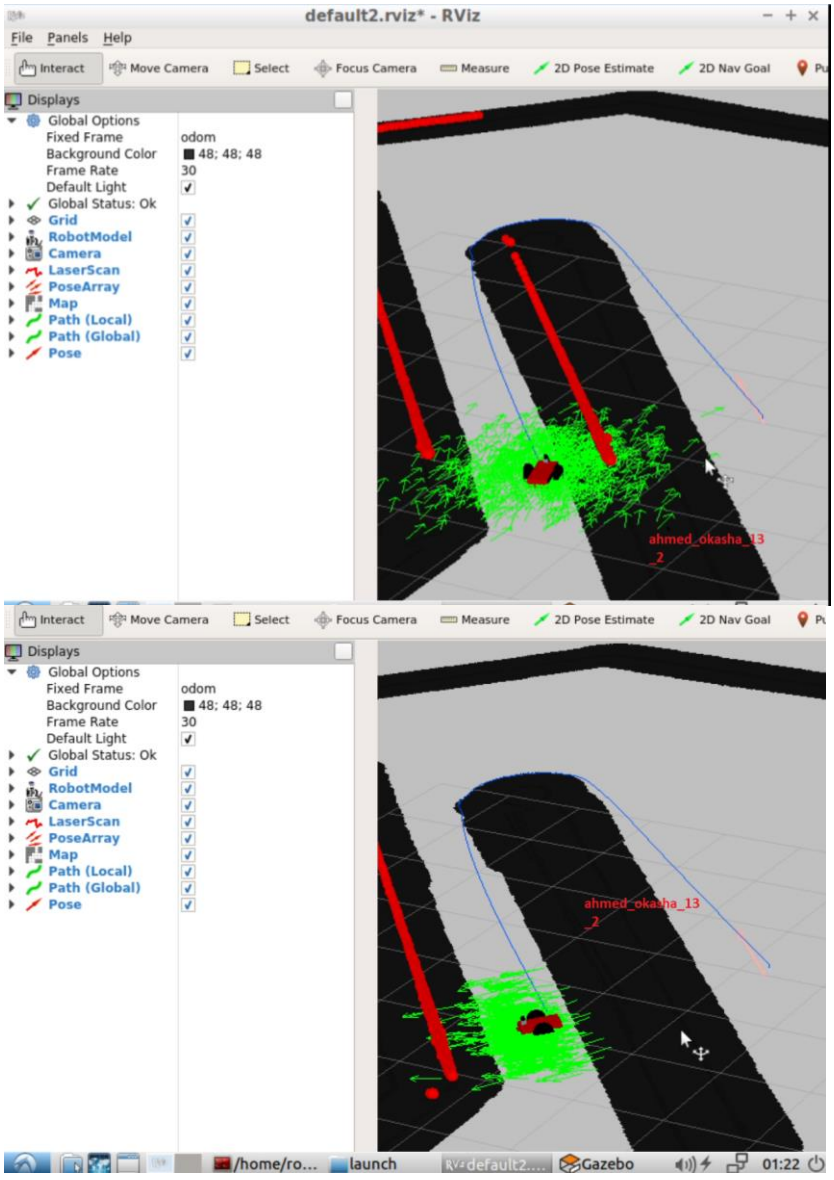
MOVING WITH MOVE_BASE

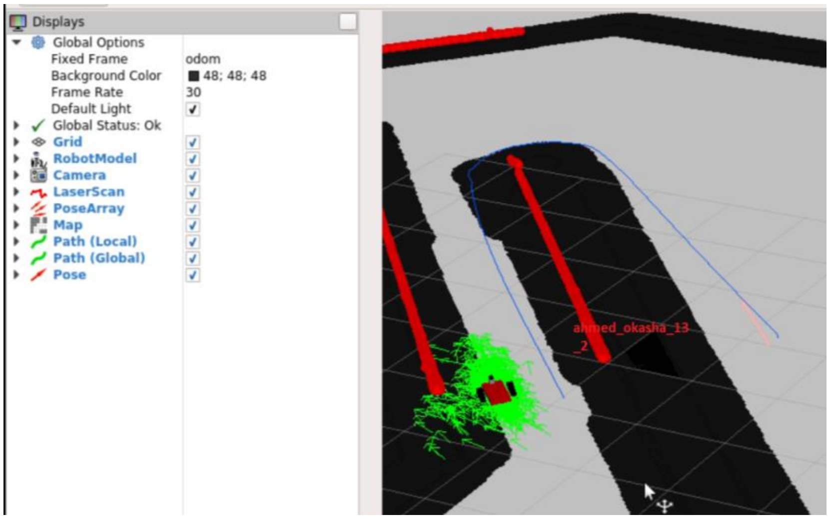
Using move_base package robot moves to destination successfully.

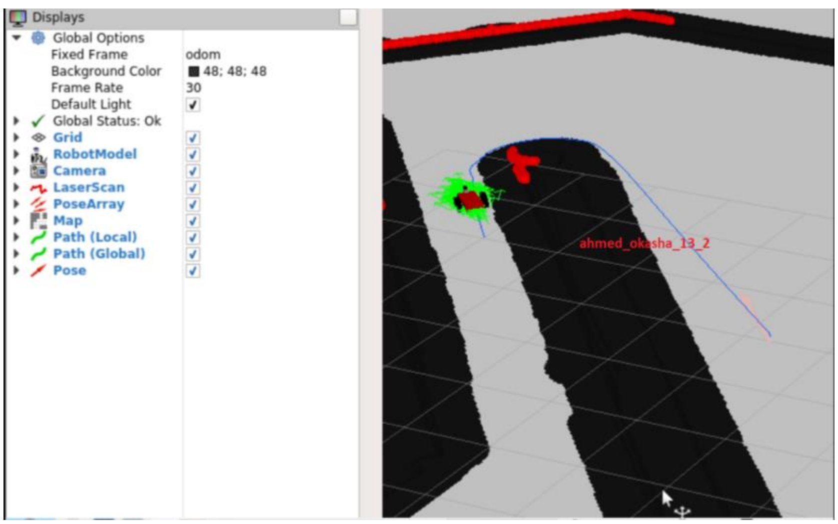
Observations:

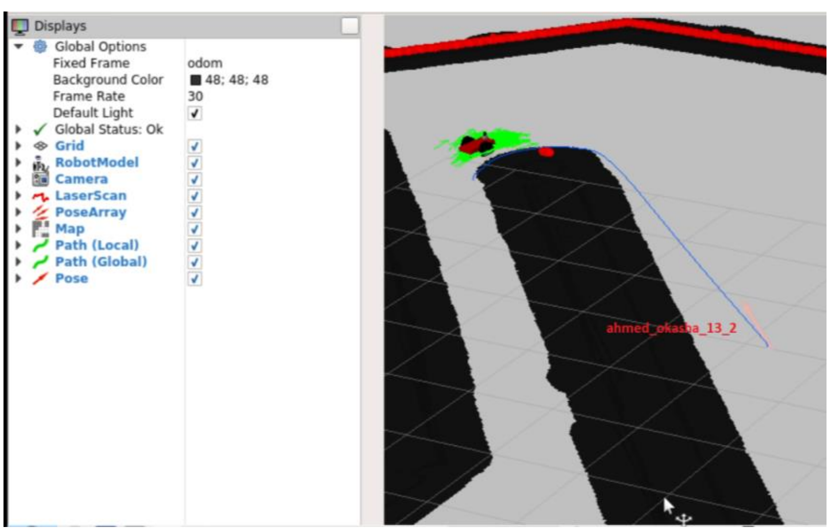
- Robot rotates sometime but it corrects its path direction after few time.
- Monto-carlo localization needs high process power
- This algorithm isn't good for wide maps

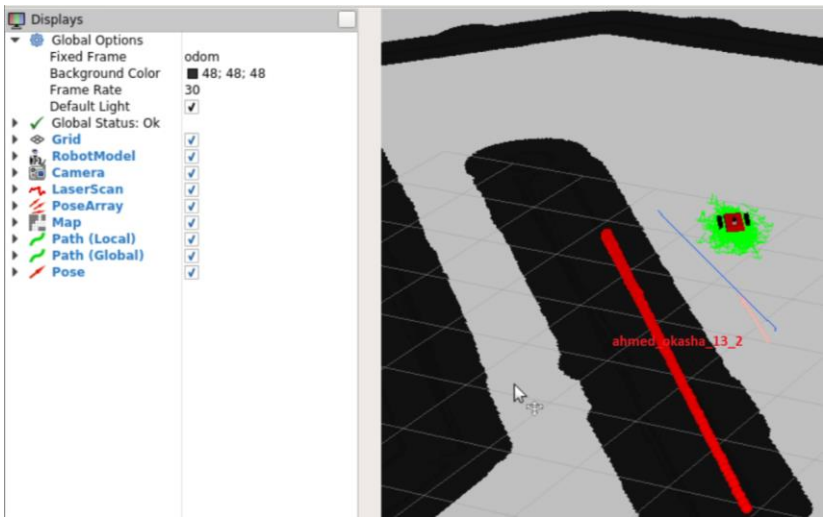

Displays
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Path (Local)
Path (Global)
Pose

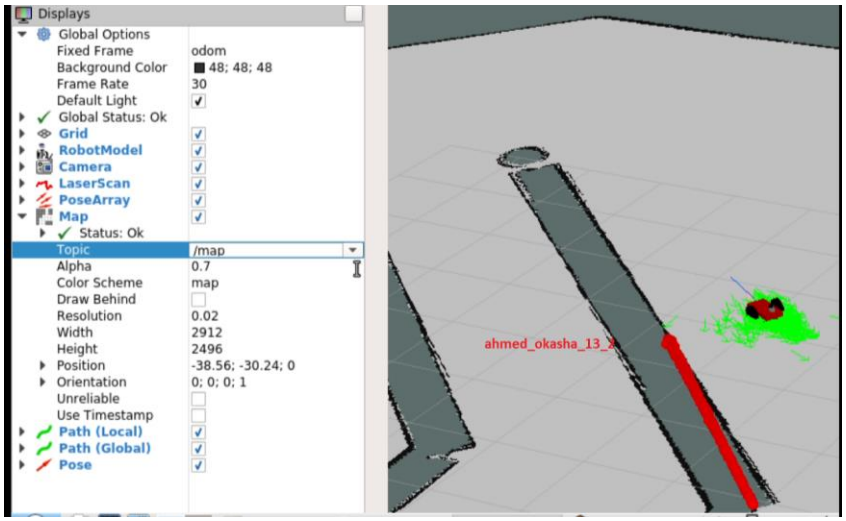

default2.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal
Displays
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Path (Local)
Path (Global)
Pose
ahmed_okasha_13_2
/home/ro...
launch
default2....
Gazebo
01:22

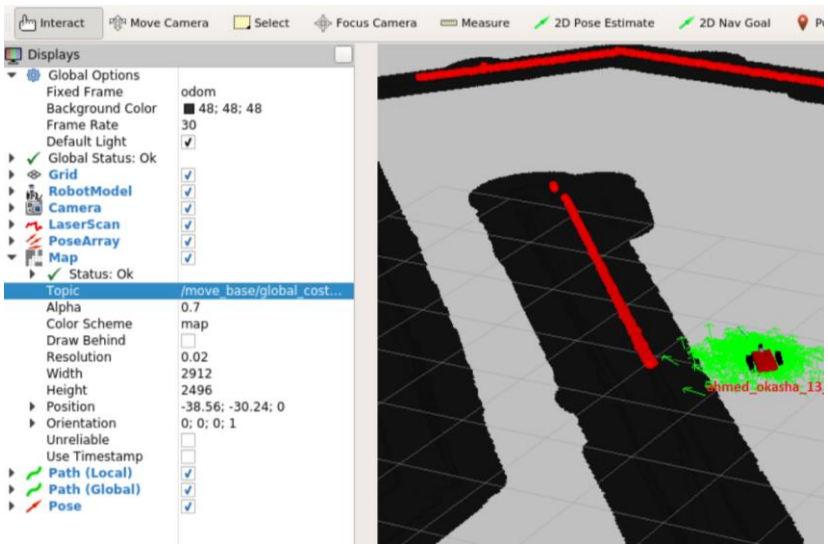

Displays
Global Options
Fixed Frame
odom
Background Color
48; 48; 48
Frame Rate
30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Path (Local)
Path (Global)
Pose


Displays
Global Options
Fixed Frame
odom
Background Color
48; 48; 48
Frame Rate
30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Path (Local)
Path (Global)
Pose


Displays
Global Options
Fixed Frame
odom
Background Color
48; 48; 48
Frame Rate
30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Path (Local)
Path (Global)
Pose


Displays
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Path (Local)
Path (Global)
Pose
ahmed_okasha_13_2


Displays
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Status: Ok
Topic /map
Alpha 0.7
Color Scheme map
Draw Behind
Resolution 0.02
Width 2912
Height 2496
Position -38.56; -30.24; 0
Orientation 0; 0; 0; 1
Unreliable
Use Timestamp
Path (Local)
Path (Global)
Pose
ahmed_okasha_13_2


Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Displays
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Grid
RobotModel
Camera
LaserScan
PoseArray
Map
Status: Ok
Topic /move_base/global_cost...
Alpha 0.7
Color Scheme map
Draw Behind
Resolution 0.02
Width 2912
Height 2496
Position -38.56; -30.24; 0
Orientation 0; 0; 0; 1
Unreliable
Use Timestamp
Path (Local)
Path (Global)
Pose

MOVING WITH NAVIGATION GOAL NODE

Observations:

- Robot keeps rotating without approaching to its goal

CONCLUSION / FUTURE WORK

CONCLUSION

- Monto- carlo can be is efficient with non-linear systems but requires high process power.
- By tuning monto-carlo parameters, stable localization system can be achieved.

FUTURE WORK

- Robot model can be upgraded to work with four wheels and decent design
- Several features can be added to robot such that it can perform some action depending on the position and maze solving option.