FEBRUARY 11, 2019

RoboND-Localization-Project

ABSTRACT

- In this project, robot model is built from scratch using ROS & Gazebo to simulate localization algorithms for robots.
- Monto-carlo, which is an efficient algorithm when coming to localization processes is used to generate a map for the field where robot operates.
- For localization process input data is taken from RGB camera, Laser range finder sensor and encoders.
- Input data is processed through AMCL package which is based on particle filters to generate field map and probabilities of robot location.
- Workspace needed to implement this project is Linux system that has ROS installed on it.
- Software used for simulation:
  - Gazebo ,
  - RViz

INTRODUCTION

When it comes to robotics area, localization isn't that travail task that can be handled perfectly.

There has been a lot of researches and development in localization area due to the great importance for stable localization algorithms in many robotic applications.

Nowadays, there is several algorithms that can work with high efficiency

In localization tasks. The most popular ones are:

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

Each of these algorithms of course has its Pros and Cons, so choosing the suitable one for your project mainly depends on the robot field and circumstances that should handle.

In this project, localization problem is handled using Monte-Carlo localization because of its efficiency in handling wide range of circumstances. Also, it is easy to implement, can handle non linearity issues and, it is efficient for local localization which is used in this project.

Another goal of this project is to have the basic knowledge to build any robotic model for simulation testing.

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:

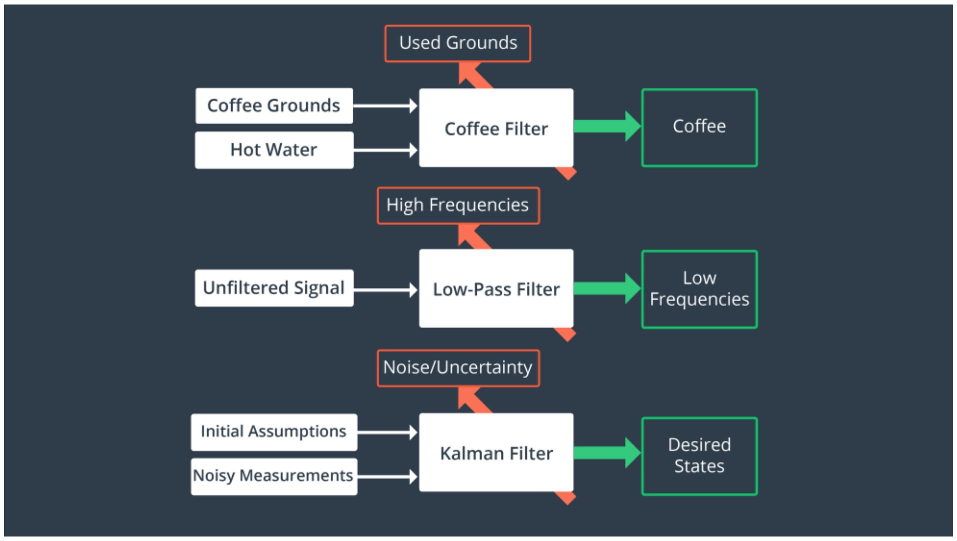
- It can develop a surprisingly accurate estimate of the true value of the variable being measured
- Requires less information
- Can produce accurate data after few measurements
- Can be used with sensor fusion to produce very accurate data

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
```

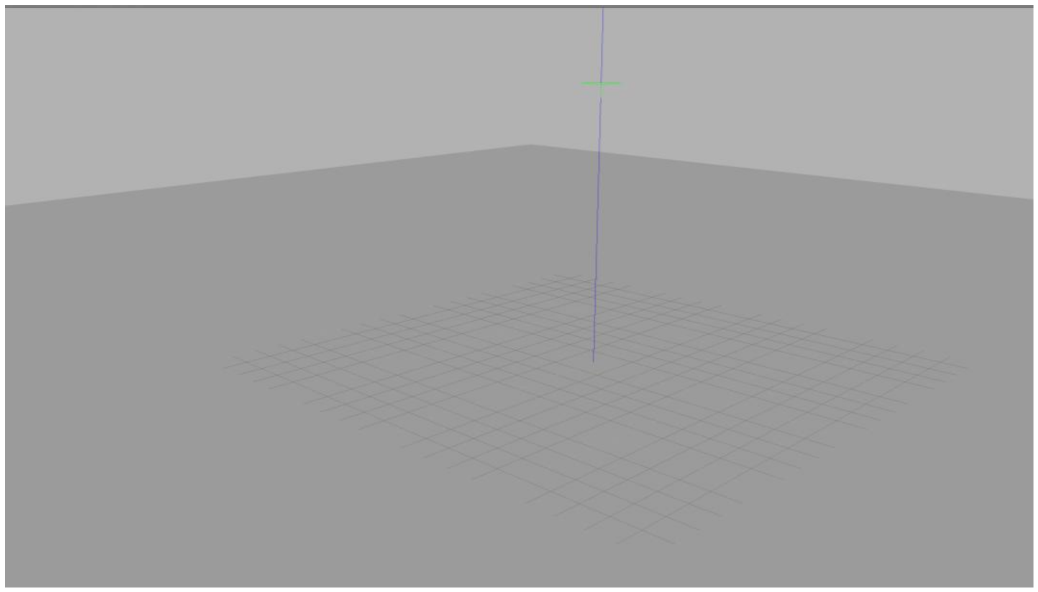
Add the following to `udacity.world`

```
<?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
```

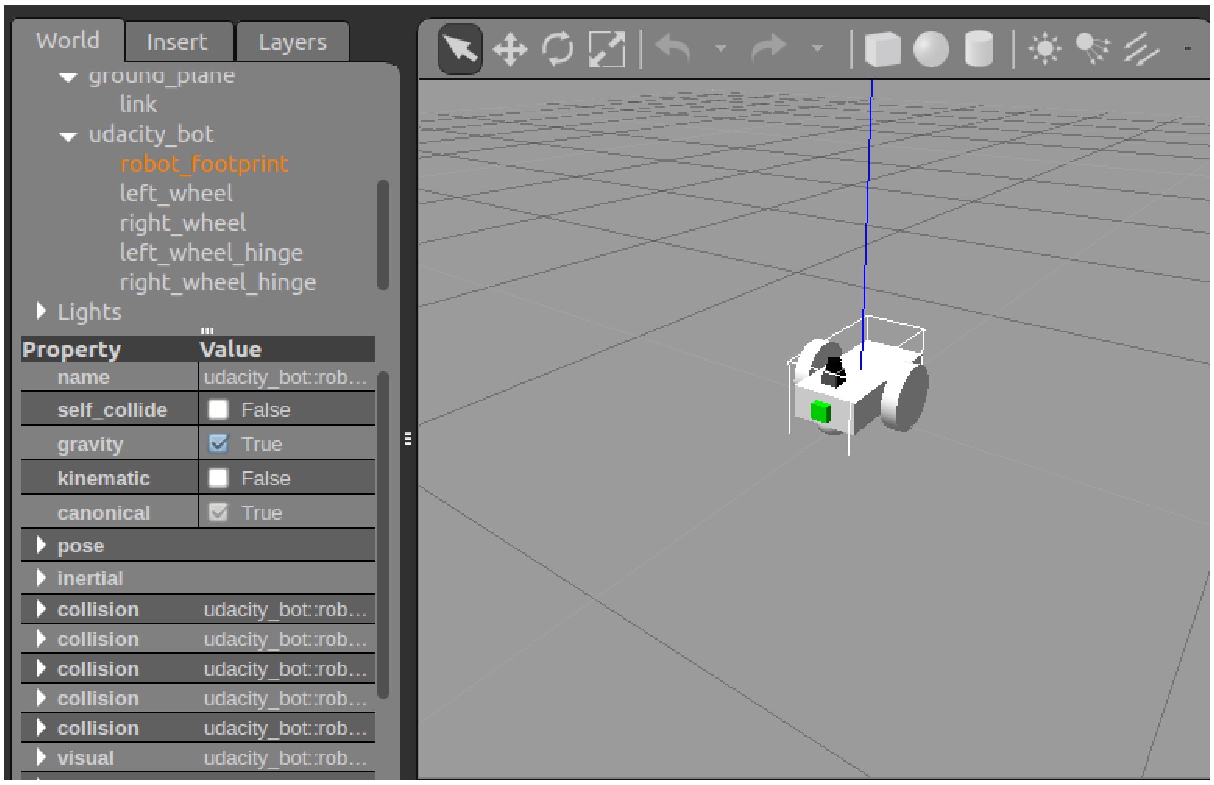
CHOOSING MODEL

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



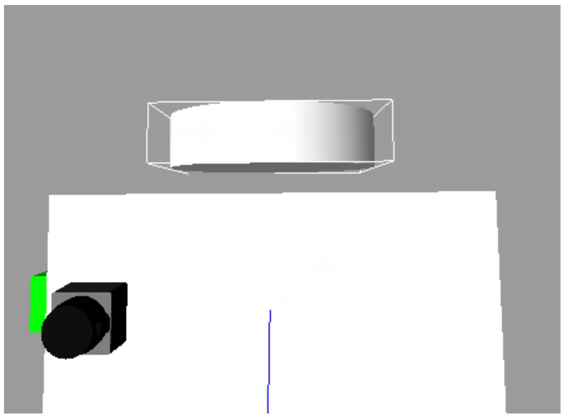
Property	Value
name	udacity_bot::rob...
self_collide	False
gravity	True
kinematic	False
canonical	True
pose	
inertial	
collision	udacity_bot::rob...
collision	udacity_bot::rob...
collision	udacity_bot::rob...
collision	udacity_bot::rob...
collision	udacity_bot::rob...
visual	udacity_bot::rob...

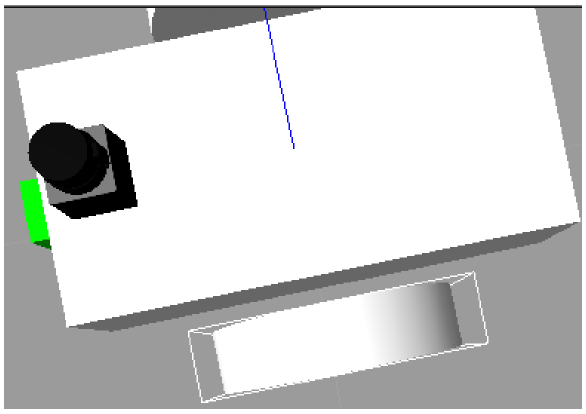
BUILDING DIFFERENTIAL WHEELS

- 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).



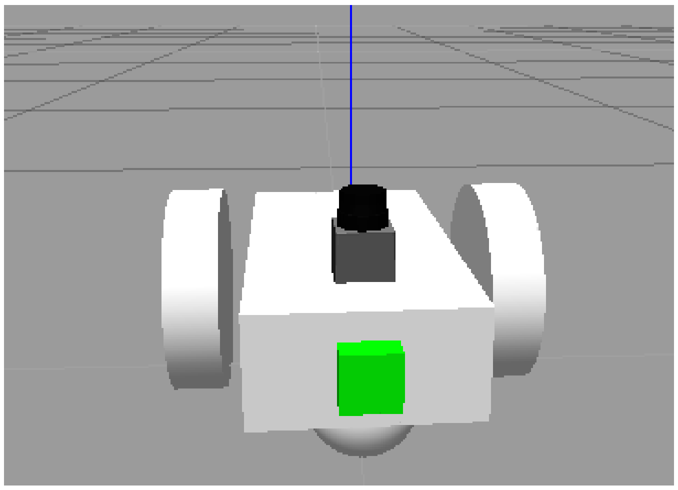


CREATING CAMERA LINK & JOINT

- 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.

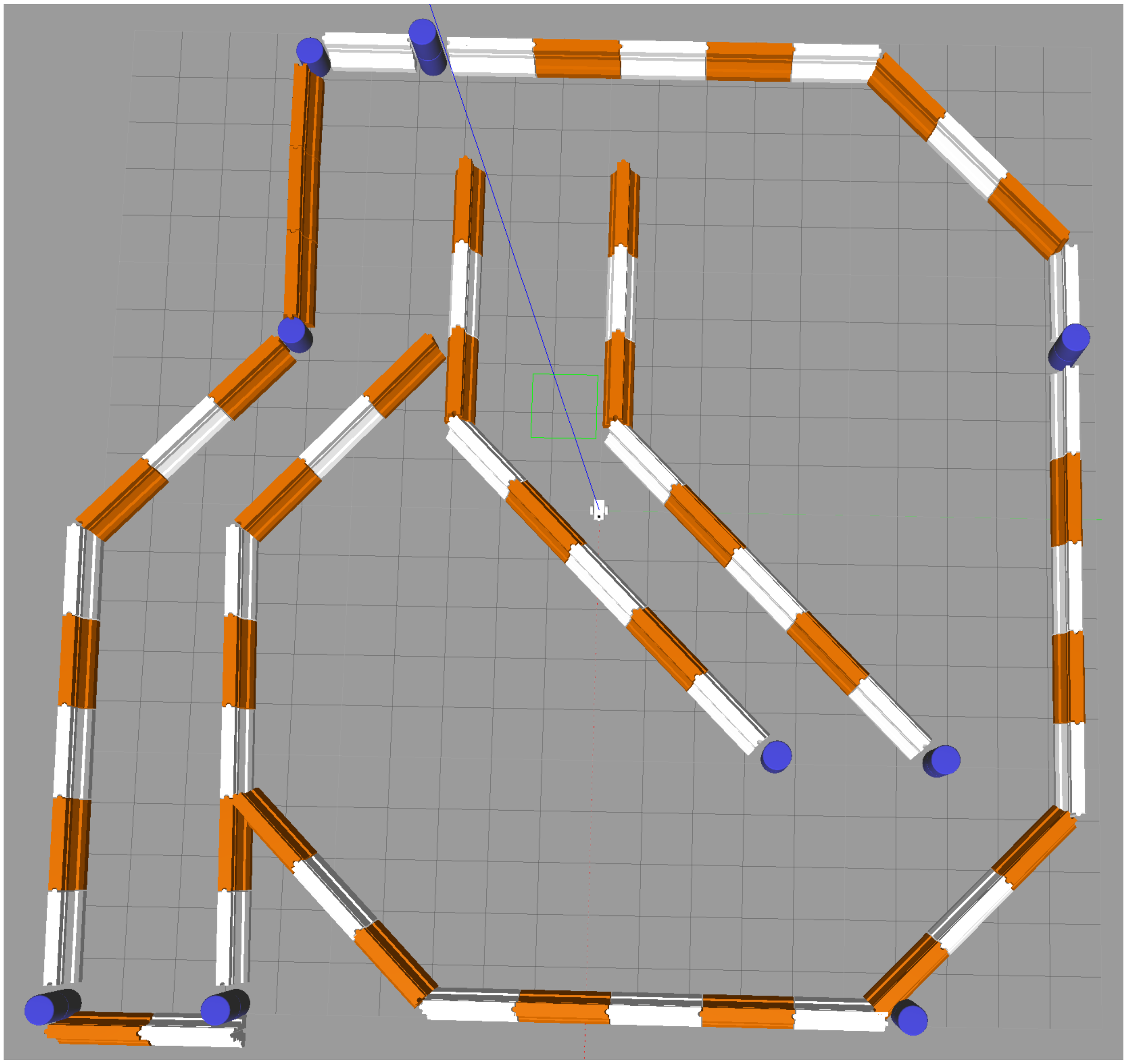
```
<!--launch rviz-->
<node name="rviz" pkg="rviz" type="rviz" respawn="false"/>
```

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.
- When set it to high values it produced high uncertainty in robot motion and causes robot to rotate specially in open areas.
- When set to low value it causes the robot to stuck in place and gives error message

- Suitable value =0.3

1- UPDATE FREQUENCY & PUBLISH FREQUENCY

- This parameter set to law value =10 Hz to suite system, but this decreases efficiency on the other hand.
- When set to high values it gives error message
- This parameter mainly depend on process power available
- Suitable observed value for them is to be the same

2- OBSTACLE RANGE

- For example, if set to 0.1, this refers 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.
- When set to small values robots hits obstacle and stuck ex:'0.1'.
- When set to very high value (val>10) it causes the robot to detects very far obstacles and this can cause missing some open paths.
- Suitable observed value=5 .

3- RAYTRACE RANGE

- This parameter is used to clear and update the free space in the costmap as the robot moves.
- High values cause a lot of uncertainty and rotation (Val>10).
- Small values gives stable motion and smooth movement (Val<1).
- Suitable observed value =0.9.
-

4- INFLATION RADIUS

- This parameter determines the minimum distance between the robot geometry and the obstacles.
- When set to high value it narrows free area to move in which can prevent robot from moving in narrow passages
- When set to small value it increases probability of hitting obstacles and stuck situations.
- Suitable observed value 0.35

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.
- Minimum particle should stay in low range (val<25).

- When max_particels is too high or too low robot confused and rotational movement increase.
- Min_partice suitable value=20.
- Max_particles suitable value =600.

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.
- Very small values increases process power
- When set to high values, time between each filter sample increase which results poor efficiency and rotation
- Update_min_a suitable value =0.05
- Update_min_d suitable value=0.125

8- RESAMPLE INTERVAL

- Number of filter updates required before resampling.
- When set to high number, it can decrease process power but this decreases efficiency on the other hand
- Suitable value which is default value =2.

9- 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"

10- LASER MAX RANGE

- Max range laser can measure
- In our map 20 m is more than enough
- Suitable value=20

11- LASER MAX BEAM

- How many evenly-spaced beams in each scan to be used when updating the filter.
- When set to very high value it increases zigzag and rotational movement.
- Robot response is fast when set to low number
- Both high and low value reach destination at the end
- Suitable value=10

12- Z_HIT

- Mixture weight for the z_hit part of the model.
- Increases particle closeness to each other
- When set to low value (val<0.5) robot become uncertain and model is lossy
- Better at high values(1>val>0.9)
- Suitable observed value =0.98

13- Z_RAND

- Mixture weight for the z_rand part of the model.
- Better movement achieved at high values(val>0.9)
- Robot stucks at low values
- Suitable observed value =.95

14- ODOM_MODEL_TYPE

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

15- ODOM_ALPHAS

- These parameters define how much noise is expected from the robot's
- Left as they are because default is already=.05

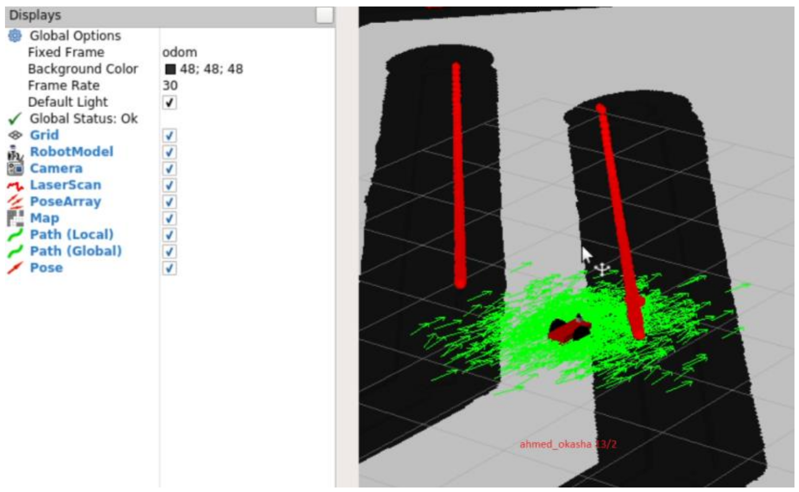
TESTING &RESULTS

MOVING WITH MOVE_BASE

Using move_base package robot moves to destination successfully.

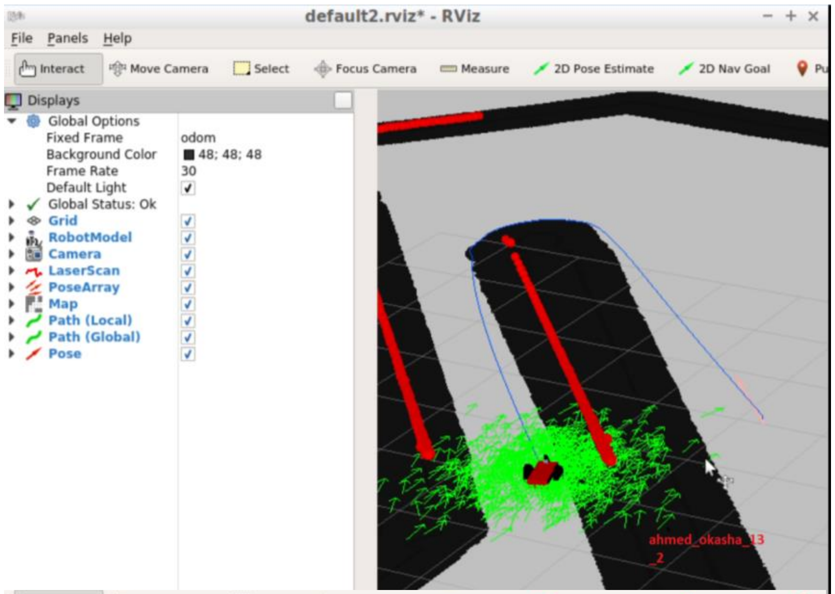
Observations:

- 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

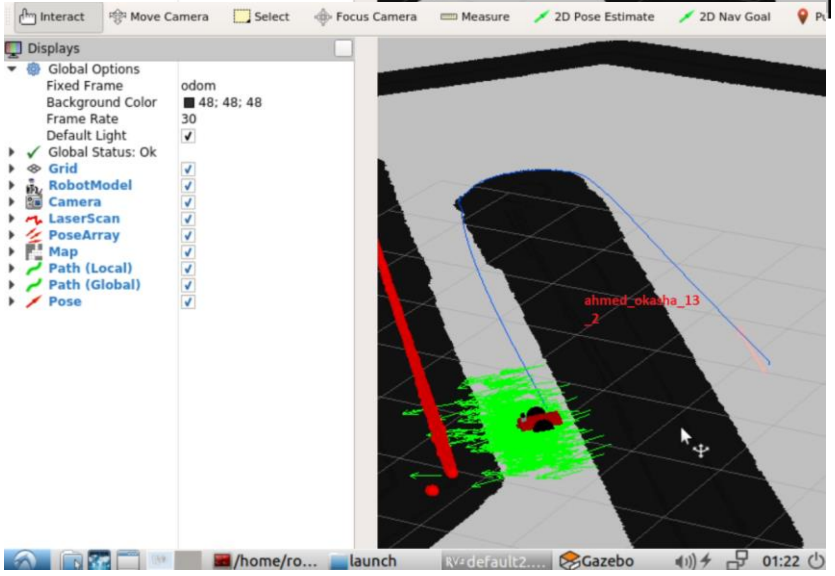


Displays

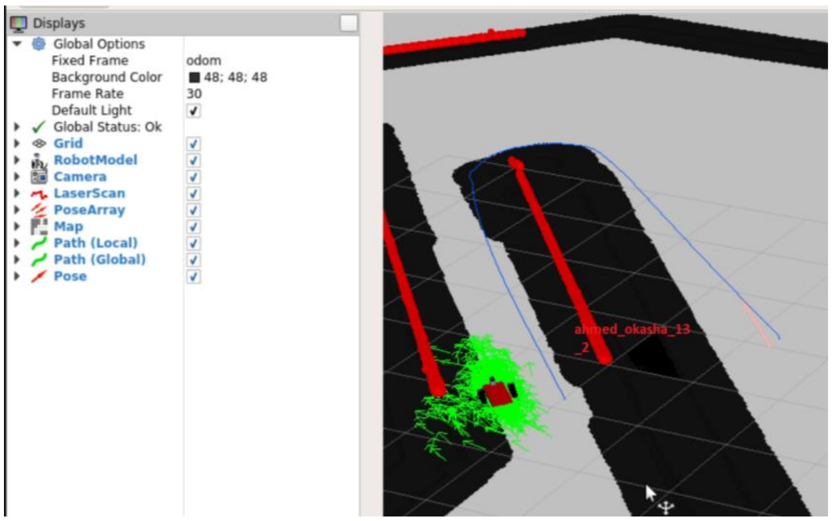
Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
RobotModel	✓
Camera	✓
LaserScan	✓
PoseArray	✓
Map	✓
Path (Local)	✓
Path (Global)	✓
Pose	✓



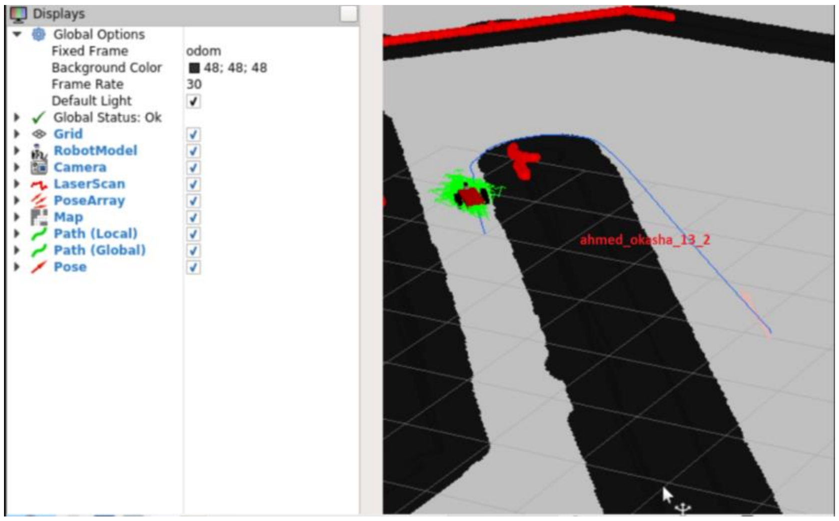
Displays
- Global Options
  - Fixed Frame: odom
  - Background Color: ■ 48; 48; 48
  - Frame Rate: 30
  - Default Light: ✓
- ✓ Global Status: Ok
- Grid ✓
- RobotModel ✓
- Camera ✓
- LaserScan ✓
- PoseArray ✓
- Map ✓
- Path (Local) ✓
- Path (Global) ✓
- Pose ✓

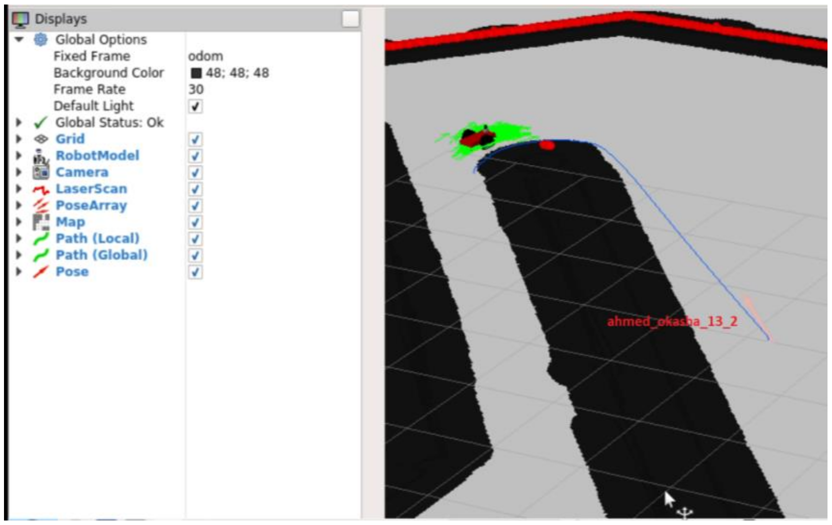


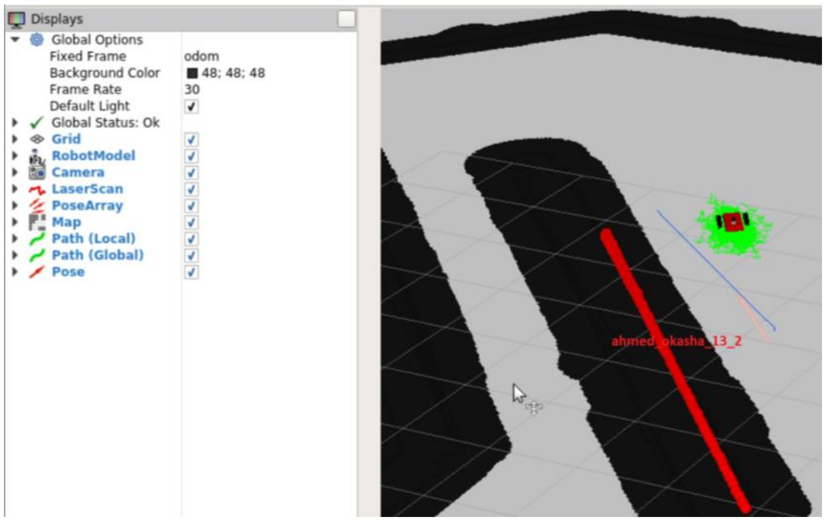
Displays
- Global Options
  - Fixed Frame: odom
  - Background Color: ■ 48; 48; 48
  - Frame Rate: 30
  - Default Light: ✓
- ✓ Global Status: Ok
- Grid ✓
- RobotModel ✓
- Camera ✓
- LaserScan ✓
- PoseArray ✓
- Map ✓
- Path (Local) ✓
- Path (Global) ✓
- Pose ✓

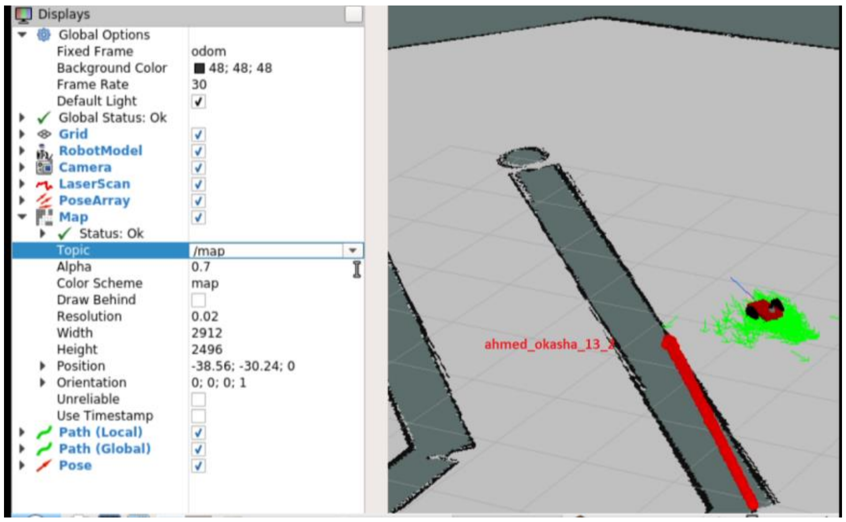


Displays
- Global Options
  - Fixed Frame: odom
  - Background Color: ■ 48; 48; 48
  - Frame Rate: 30
  - Default Light: ✓
- ✓ Global Status: Ok
- Grid ✓
- RobotModel ✓
- Camera ✓
- LaserScan ✓
- PoseArray ✓
- Map ✓
- Path (Local) ✓
- Path (Global) ✓
- Pose ✓

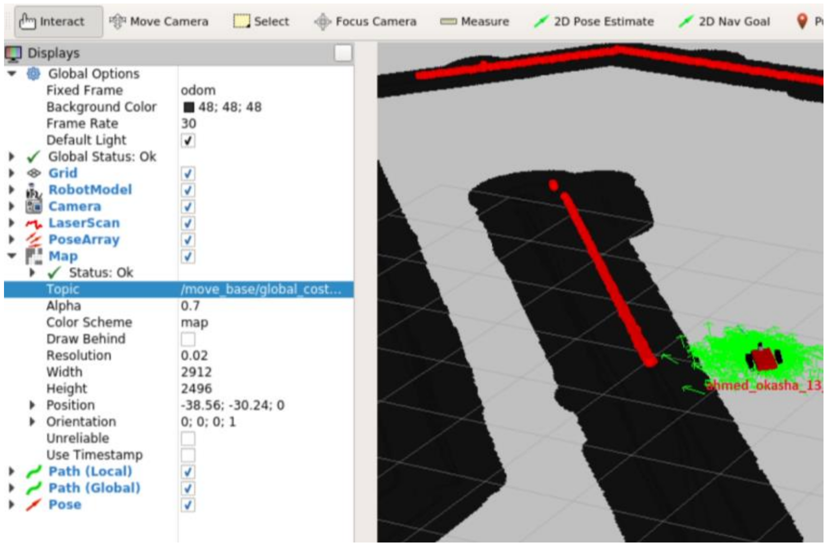

```
Displays
▼ Global Options
    Fixed Frame          odom
    Background Color     ■ 48; 48; 48
    Frame Rate           30
    Default Light        ✓
▸ ✓ Global Status: Ok
▸ Grid                   ✓
▸ RobotModel             ✓
▸ Camera                 ✓
▸ LaserScan              ✓
▸ PoseArray      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


```
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                    ✓
  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                   ✓
```



```
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:

- Still didn't work after optimizing parameter
- I think this problem because of move base package because if this path is split to several points robot achieve its destination with high certainty

CONCLUSION / FUTURE WORK/DISCUSSION

KIDNAPPED ROBOT ISSUE

- Robot can handle kidnapped case because Monto Carlo works for both global and local localization.
- To handle this issues Robot need to change particle number again and it will start to recognize its position with the upcoming iterations
- Also robot should be in the same map or it can't recognize its place.
- We can create function that increases particle number when there is huge uncertainty .but this only works after creating area map at the first time.

INDUSTRIAL APPLICATIONS

This algorithm can be used in:

- Car factories where robotic parts perform operations in exact location.
- Factories where robots have to move material from place to another.
- Creating assistant robots in houses.

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.

DIFFCULTIES OF MONTO CARLO IN WIDE MAPS (IN MY OPENION)

- It will need high process power
- Also wide maps can have similar places that perhaps can confuse the robot
- Can cause the system to have high amount of delay