Where Am I?

Gaurav Saxena

Abstract—The purpose of this project is to learn about localization utilizing ROS packages to accurately localize a mobile robot inside a provided map in the Gazebo and RViz simulation environments. We will be using the Monte Carlo Algorithm for solving the localization problem. Using a 2 wheeler robot with differential steering system. Will be leveraging the navigation stack provided by ROS for localization. We will be fine tuning the con-figs for cost map and base local planner parameters so that robot can accurately localize itself and navigate to designated location.

Index Terms—Robot, ROS, Monte Carlo Algorithm

1 Introduction

LOCLIZATION is the challenge of determining robot's pose in a mapped environment. The need of having localization capability is getting more important day by day. As we want robots to be more autonomous in nature. We can leverage different algorithm to solve localization problem. Some of most commonly used algorithms are Kalman Filter, Extended Kalman Filter and Monte Carlo localization. In this project we will be using the Monte Carlo localization to solve this problem.

2 Background / Formulation

As discussed above there are different algorithms to solve localization. Out of all others Extended Kalman Filter and Monte Carlo localization are most prominent one. So we will be discussing about these two.

2.1 Kalman Filters

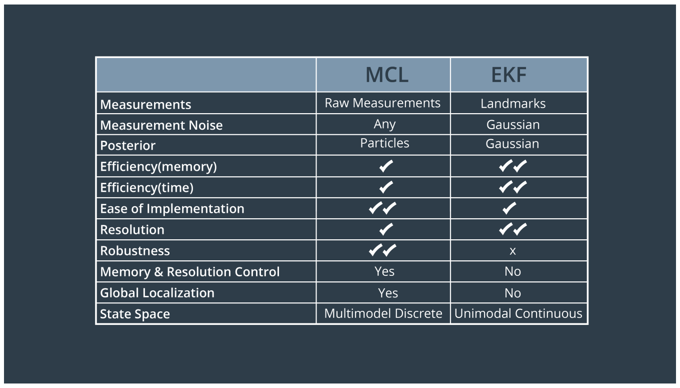
Kalman Filter is an estimation algorithm that is very prominent in controls. It's used to estimate the value of a variable in real time as the data is being collected. The reason that the kalman filter is so noteworthy is because it can take data with a lot of uncertainty or noise in the measurements and provide a very accurate estimate of the real value. Also we can use sensor fusion technique to use multiple input source like range finder wheel encoder to guess the location of the robot. We need to take the assumption for kalman Filter that "Motion and measurement models are linear" and "State space can be represented by uni-model Gaussian distribution". To over come this limitation we use Extended Kalman Filter. The Extended Kalman Filter can be summarized as "Extended Kalman Filter linearizes the nonlinear function f(x) over a small section and calls it F. This linearization, F, is then used to update the state's variance.".

2.2 Particle Filters

Particle filter localization, is an algorithm for robots to localize using a particle filter. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, i.e., a hypothesis of where the robot is. The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning the robot has no information about where it is and assumes it is equally likely to be at any point in space.[4] Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are resampled based on recursive Bayesian estimation, i.e., how well the actual sensed data correlate with the predicted state. Ultimately, the particles should converge towards the actual position of the robot.

2.3 Comparison / Contrast

When compared MCL vs EKF, MCL provides ease of implementation, MCL represents non-Gaussian distributions but EKF represents only Gaussian distributions also using MCL we can control memory and resolution by changing the number particles distributed uniformly and randomly throughout the map. More differences are shown below in the image.



	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	✓	✓✓
Efficiency(time)	✓	✓✓
Ease of Implementation	✓✓	✓
Resolution	✓	✓✓
Robustness	✓✓	x
Memory & Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodal Continuous

Fig. 1. MCL vs EKF.

3 Results

Both robots reached the goal under 60 seconds after tweaking the parameters. Particles were uniformly spread at the start of the simulation and later narrow down to a small group as shown in the below images while going to the goal. In case of my bot the mass was increased to 20 and a fin was added with bumper due to which the camera sensor was moved forward. But most of the details was similar. So the result was also similar.

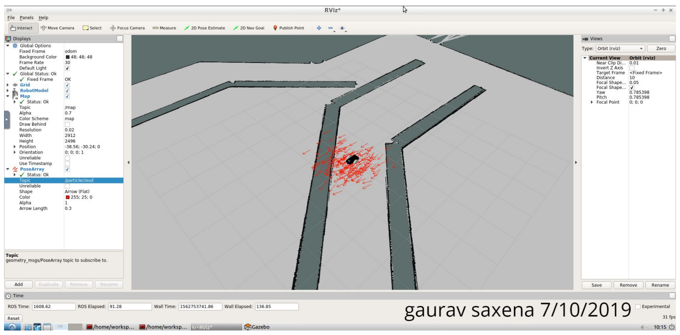


Fig. 2. udacity bot at start

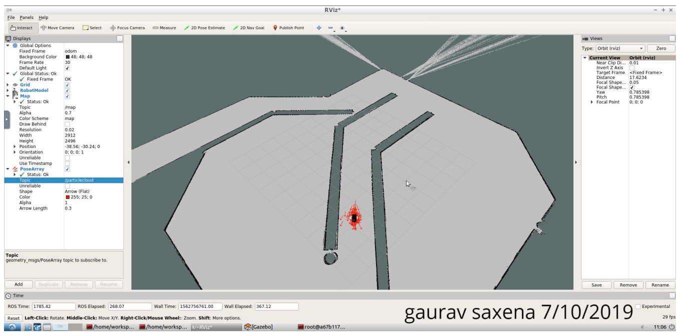


Fig. 3. udacity bot moving out of the pathway

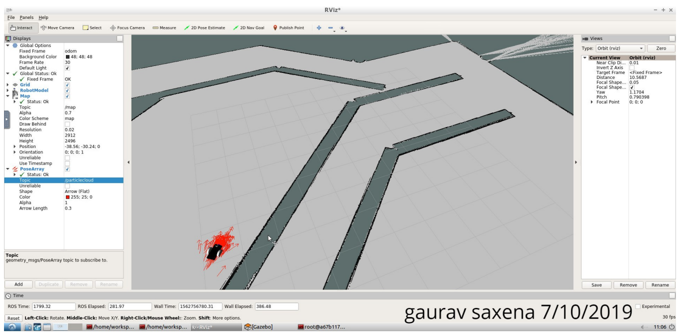


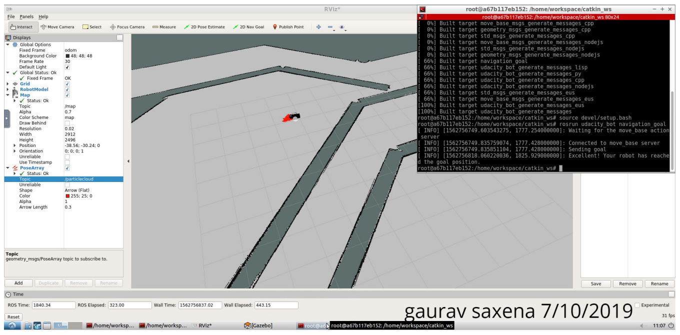
Fig. 4. udacity bot moving toward the goal

4 Parameter Tuning

The parameter tuning was the most time consuming part of the project. As we were given pointers to ROS documentation for understanding each parameter meaning and its impact. Then we introduce each parameter and see the impact of that parameter and later fine tune each parameter to achieve desired result.

We modified the parameters present in different yaml files to fine tune the costmap parameters.

Also we added and modified the parameters in amcl.launch file



```
root@a67b117eb152: /home/workspace/catkin_ws
root@a67b117eb152: /home/workspace/catkin_ws 80x24
[  0%] Built target move_base_msgs_generate_messages_cpp
[  0%] Built target geometry_msgs_generate_messages_cpp
[  0%] Built target std_msgs_generate_messages_cpp
[  0%] Built target move_base_msgs_generate_messages_nodejs
[  0%] Built target std_msgs_generate_messages_nodejs
[  0%] Built target geometry_msgs_generate_messages_nodejs
[ 66%] Built target navigation_goal
[ 66%] Built target udacity_bot_generate_messages_lisp
[ 66%] Built target udacity_bot_generate_messages_py
[ 66%] Built target udacity_bot_generate_messages_cpp
[ 66%] Built target udacity_bot_generate_messages_nodejs
[ 66%] Built target std_msgs_generate_messages_eus
[ 66%] Built target move_base_msgs_generate_messages_eus
[100%] Built target udacity_bot_generate_messages_eus
[100%] Built target udacity_bot_generate_messages
root@a67b117eb152:/home/workspace/catkin_ws# source devel/setup.bash
root@a67b117eb152:/home/workspace/catkin_ws# rosrun udacity_bot navigation_goal
[ INFO] [1562756749.603543275, 1777.254000000]: Waiting for the move_base action
 server
[ INFO] [1562756749.835759074, 1777.428000000]: Connected to move_base server
[ INFO] [1562756749.835851104, 1777.428000000]: Sending goal
[ INFO] [1562756818.060220036, 1825.929000000]: Excellent! Your robot has reache
d the goal position.
root@a67b117eb152:/home/workspace/catkin_ws#
```

Fig. 5. udacity bot goal reached.

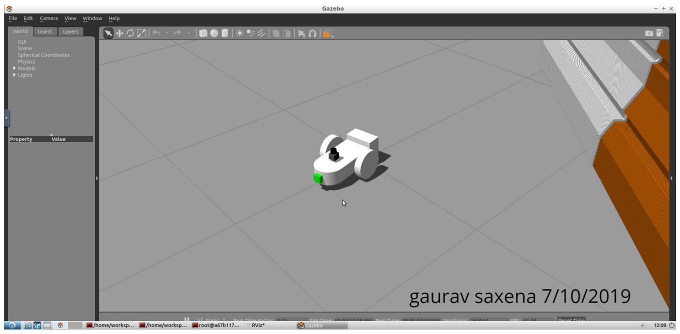


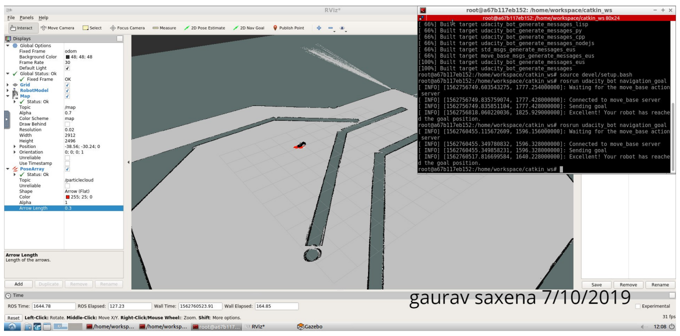
Fig. 6. my bot.

5 Discussion

As seen in the project we need to understand the algorithm and model off the robot and how it effects the behaviour of the robot and fine tune the parameters to achieve desired behaviour. As I learned in this project that localization is a complex problem. And the complexity increases with the introduction of multiple sensory inputs which to solve that we need to use the sensor fusion. With current implementation both the robots reach the target within 60 seconds. But both the robot take more time in orienting it self once reached on the target. To solve this need to increase the refresh frequency.

6 Conclusion / Future work

Recently Udacity has introduce Sensor Fusion Engineer Nanodegree which i am thinking of taking after this course.



```
root@a67b117eb152: /home/workspace/catkin_ws
root@a67b117eb152: /home/workspace/catkin_ws 80x24
[ 66%] Built target udacity_bot_generate_messages_lisp
[ 66%] Built target udacity_bot_generate_messages_py
[ 66%] Built target udacity_bot_generate_messages_cpp
[ 66%] Built target udacity_bot_generate_messages_nodejs
[ 66%] Built target std_msgs_generate_messages_eus
[ 66%] Built target move_base_msgs_generate_messages_eus
[100%] Built target udacity_bot_generate_messages_eus
[100%] Built target udacity_bot_generate_messages
root@a67b117eb152:/home/workspace/catkin_ws# source devel/setup.bash
root@a67b117eb152:/home/workspace/catkin_ws# rosrun udacity_bot navigation_goal
[ INFO] [1562756749.603543275, 1777.254000000]: Waiting for the move_base action
 server
[ INFO] [1562756749.835759074, 1777.428000000]: Connected to move_base server
[ INFO] [1562756749.835851104, 1777.428000000]: Sending goal
[ INFO] [1562756818.060220036, 1825.929000000]: Excellent! Your robot has reache
d the goal position.
root@a67b117eb152:/home/workspace/catkin_ws# rosrun udacity_bot navigation_goal
[ INFO] [1562760455.115672609, 1596.156000000]: Waiting for the move_base action
 server
[ INFO] [1562760455.349780832, 1596.328000000]: Connected to move_base server
[ INFO] [1562760455.349858231, 1596.328000000]: Sending goal
[ INFO] [1562760517.816699584, 1640.228000000]: Excellent! Your robot has reache
d the goal position.
root@a67b117eb152:/home/workspace/catkin_ws#
```

Fig. 7. my bot goal reached.

TABLE 1
local_costmap_params.yaml

global_frame	odom
robot_base_frame	robot_footprint
update_frequency	20.0
publish_frequency	5.0
width	5.0
height	5.0
resolution	0.05
static_map	false
rolling_window	true

TABLE 2
global_costmap_params.yaml

global_frame	map
robot_base_frame	robot_footprint
update_frequency	20.0
publish_frequency	5.0
width	20.0
height	20.0
resolution	0.05
static_map	true
rolling_window	false

By taking that course I will be able to have a greater understanding of the localization domain. I also looked at the other projects submitted by diffident students and learned there approach to solve this problem. Need to learn more about localization problem and different solution by digging on the internet. I am also thinking about assembling my own robot platform to implement the localization solution.

REFERENCES

TABLE 3
costmap_common_params.yaml

map_type	costmap
obstacle_range	1.2
raytrace_range	1.2
transform_tolerance	0.4
inflation_radius	0.5
observation_sources	laser_scan_sensor

TABLE 4
amcl.launch

transform_tolerance	0.3
min_particles	50
max_particles	200
update_min_a	0.1
update_min_d	0.25
laser_model_type	likelihood_field
laser_max_beams	20
laser_z_rand	0.05
laser_z_hit	0.95
odom_alpha1	0.05
odom_alpha2	0.05
odom_alpha3	0.05
odom_alpha4	0.105