ROS AMCL Robot Navigation

Ussama Naal

Abstract—In this work we investigate the use an MCL variant algorithm known as AMCL. We do this within a simulated environment using ROS and Gazebo. We setup a two wheeled robot with a map that has a given track. The robot is then equipped with a rangefinder and an odometer to track the robot motion. We then feed the readings from these sensors to AMCL ros package and let it control the robot. With a bit of tuning for the parameters of AMCL and other ros packages the the robot can successfully navigate to the target goal.

Index Terms—Robot navigation, ROS, particle filter, MCL, AMCL.

1 Introduction

To carry out a successful robot navigation within a given world or map, the robot must be able to locate itself within that map. For this, robots are typically equipped with multiple sensors such as cameras, laser scanners, odometers, etc. These sensors help the robot identify its surrounding area as it moves through out the map. For example, laser scanners give the robot a measurement of how far obstacles like walls are from the robot. An odometer tracks the path that robot is moving along from the initial pose.

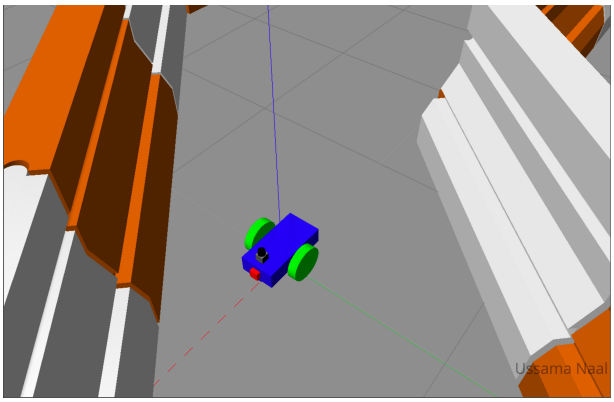

Fig. 1. Image of our test robot.

2 Background / Formulation

As mentioned, the sensors provide the robot with measurements that can help the robot estimate its position within the map. However, the measurements supplied by the sensors may not be fully accurate due to noise. Kalman filters can reduce the impact of the noise on the measurements if the noise has a Gaussian distribution and the system is linear. Extended Kalman Filters can work for system that are non-linear in nature but still requires the noise to follow a Gaussian distribution. However, when this is not the case a better alternative would be to employ MCL or Particles Filter. Particles filter work for non-linear systems and doesn't require that noise coming from sensor measurements to have a Gaussian distribution.

The following listing shows the basic steps in the MCL algorithm:

Algorithm 1: MCL Algorithm

Function MCL (X_{t-1}, u_t, z_t) **:**
  $\bar{X}_t = X_t = \phi$;
  for *m=1 to M* **do**
    $X_t^{[m]} = motion\_update(u_t, x_{t-1}^{[m]})$;
    $w_t^{[m]} = sensor\_update(z_t, x_t^{[m]})$;
    $\bar{X}_t = \bar{X}_t + < x_t^{[m]}, w_t^{[m]} >$;
  end
  for *m=1 To M* **do**
    Draw $X_t^{[m]}$ from $\bar{X}_t$ with probability $\propto w_t^{[m]}$;
    $X_t = X_t^{[m]}$;
  end
return X_t

3 Model Configuration

Our robot is composed of a main chassis and two large side wheels. To help keep the robot stable we add two caster wheels at the front and the back of the robot as shown below:

We also equip the robot with a camera at the front of the robot which allows us the see what the robot might be facing while it moves. We place a Hokuyo rangefinder on the top of the robot at the front. The rangefinder measures how far obstacles are from the robot.

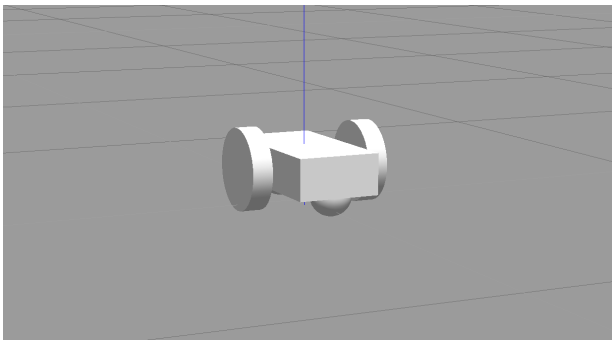

Fig. 2. Robot Setup.

Then we enable AMCL ROS package which will guide the robot to a target goal. The AMCL package implements an adaptive version of the MCL algorithm, where the algorithm changes the number of samples used in real-time [1]. Once the robot localizes itself, lower number of samples is required to do the tracking. The advantages of this is a better of efficiency for the robot, and also allows the robot stay responsive with measurements that come from sensors at a higher rate [2].

The following table lists parameters to make the simulation functioning within the system used to run the simulation:

TABLE 1
Simulation Update Parameters

Parameter	Value
transform_tolerance	0.2
update_frequency (Local)	14 Hz
publish_frequency (Local)	14 Hz
update_frequency (Global)	14 Hz
publish_frequency (Global)	14 Hz
controller_update	14 Hz

Setting these values is important to have the simulation proceed with no issues or warnings reported on the console.

Next we configure another set of parameters for the costmap as shown in Table 2.

TABLE 2
Costmap Parameters

Parameter	Value
inflation_radius	0.7
inflation_radius	0.5
raytrace_range	4.0

These parameters tune the costmap such that it prevents the robot from getting stock against walls or other obstacles on the map.

Finally, we tweak some of the parameters for the AMCL package [3] as shown in Table 3.

TABLE 3
AMCL Parameters

Parameter	Value
min_particles	100
max_particles	300
initial_pose_x	0.0
initial_pose_y	0.0
initial_pose_a	-0.02
update_min_d	0.2
update_min_a	0.52
laser_mode_type	likelihood_field
laser_x_range	-1.0
laser_y_range	-1.0
laser_max_beams	30
laser_z_hit	0.95
laser_z_rand	0.05
odom_model_type	diff_corrected
odom_alpha1	0.2
odom_alpha2	0.2
odom_alpha3	0.2
odom_alpha4	0.2

Default value for the max_particles is 5000. We limit this to only 300 such that the system doesn't require a huge amount of computation and can still run smoothly within the simulated environment. Ultimately more particles should give better results but at the expense of increased execution time for each iteration of the algorithm.

4 Results

With these parameters set we launch the simulation and then run the navigation_goal executable to configure the goal at run time. Once navigation_goal is run the robot starts to move around a bit then within 30-60 seconds the robot is able to reach the target goal. The following figure shows how the robot successfully navigates to the goal with the right pose:

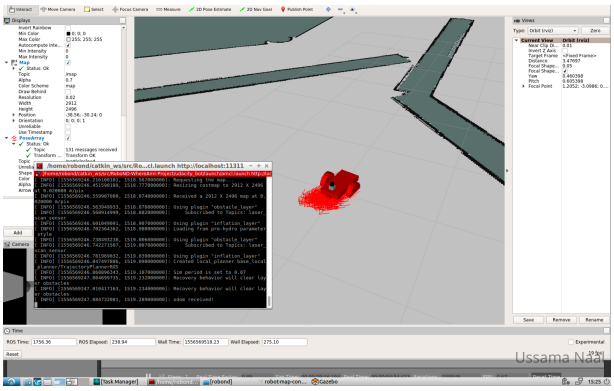

Fig. 3. Robot at target location.

5 Discussion

The robot successfully manages to navigate to the target goal, however, there are still some occasions where the robot enters a stale state. That means there a possible room for improvement and parameter tuning to avoid going into such state.

Default MCL algorithm does not deal with kidnapped robot state. i.e. if the robot was moved to another spot on the map without being notified with change. The AMCL variant, however, can recover from this by continuously spawning few random samples from uniform distributed over the map.

Both MCL and AMCL can be used in scenarios where a given robot needs to localize and track its movement within a known map. Example of such uses are the Knightscope [4] autonomous security guard, where the robot constantly moving around to secure the perimeter of the protected site. Other examples are delivery robots with office or hotel spaces, vacuum cleaning robots, etc.

6 Conclusion / Future work

Further tune listed parameters and investigate the remaining list of parameters for the move_base and AMCL packages.

Configure and run the system on actual robot.

Addition of other sensors such as RGB-D camera or a LIDAR would give better scan of the surrounding area of the robot and theoretically should improve localization accuracy. However, this would come at the cost of increased processing time, due to larger incoming data from the sensors. As mentioned in [2] the responsiveness of the system is also important to successfully perform the localization. Because of that it would necessary to upgrade the system with better hardware for data processing like relying on GPUs. Luckily many hardware vendors are providing good options for GPUs specifically made for on-board processing for the robot such as NVIDIA's Jetston X2 [5].

References

[1] R. K. Base, "Adaptive Monte Carlo Localization." http://roboticsknowledgebase.com/wiki/state-estimation/adaptive-monte-carlo-localization/, 2018. [Online; accessed 02-May-2019].

[2] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *IN PROC. OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI*, pp. 343–349, 1999.

[3] B. P. Gerkey, "ROS AMCL." http://wiki.ros.org/amcl#Parameters/, 2018. [Online; accessed 30-April-2019].

[4] knightscope, "Knightscope." https://www.knightscope.com/, 2019. [Online; accessed 02-May-2019].

[5] nvidia, "NVIDIA EMBEDDED COMPUTING." https://developer.nvidia.com/embedded-computing/, 2018. [Online; accessed 02-May-2019].