Localization with Monte-Carlo Method based on ROS Package

Ming Lin

Abstract—Monte-Carlo based localization is a widely use algorithm in robot application. The main application in robotics is called particle filter. Particle filter is a kind of filter algorithm which is totally different with filter like KF, EKF, UKF family. Particle filer doesn't use moment of noise like mean value and variance value to filtering the noise. Particle filter use many particles to approximate many kinds of noise distributions. So particle filter can extract any noise distribution and filtering any noise distribution with enough particles. But as the quantity of particles grows, the more computation power it uses. So it is a trade off of computation efficiency and precision of estimation. In this paper, a ROS package which is named 'AMCL' with adaptive particle filter is used to localize the robot in simulation environment. The simulation results in GAZEBO and RVIZ shown the effectiveness of AMCL package.

Index Terms—Robot, ROS, Udacity, LaTeX, Localization, AMCL.

1 Introduction

The particle filter is a algorithm based on monte-carlo approach method. It uses many particles to approximate the noise distribution and filtering the noise. The main advantage of monte-carlo based algorithm is it can localize the robot in both local and global localization. AMCL package is short of Adaptive Monte Carlo localization. This package uses adaptive quantity of particles to trade off the computation efficiency and precision of estimation. In this paper, in order to use the AMCL package to localize the robot, the ROS system is used. For verifying the performance the simulation tool GAZEBO and RVIZ are used. For moving the robot, the navigation stack and cost2dmap packages in RVIZ are used.

2 Background

Localization problem is a hard problem in real world. Robot needs to estimate it's position and transform the sensors data in to local map. After generates the local map, if robot is set to moving to a goal position, then it needs path planning module to make local and global plan. In this paper, environment is assumed that the global map is given. So robot needs to navigate to the goal position with on-board camera sensor and LIDAR sensor. Adaptive monte carlo localization method is chosen for it advantages. AMCL has advantages in global localization. It also has computational efficiency and precision when quantity of particles are adaptive to the localization problem. [1] The whole process are shown as below

1) Define the robot model in gazebo in URDF format. Construct the links and joints of robot.
2) Define the robot sensors
3) Configure the global cost map and the local cost map.
4) Configure the amcl.launch file to get good localization performance.
5) Configure the interface of each files. Such as remap topic.
6) Construct the simulation environment
7) Run the navigation-goal node.
8) Iterate 3-7 steps and tuning the parameters. (Udacity-bot and truck-bot were made for simulation)

2.1 Kalman Filters

Kalman filter is a very famous algorithm to estimate robot's pose and orientation problem. At the first of version of kalman filer, the main assumption of real world is the noise is always gaussian. It means KF just assume the whole noise in real world is gaussian noise. And it also need to be linear. Then the second version of kalman filter for processing the non-linear problem was invented which is called Extended kalman filter. EKF can linearize the nonlinear function with Taylor series expansion. But it still can't cover the nonlinear noise distribution. So the UKF was maded. UKF uses unscented transformation to cover the nonliear distribution of noise. All of these KF based filters are based on parameters to linearize the noise distribution. All of these still assume noise is gaussian. So these are still too weak in non-gaussian distribution.

2.2 Particle Filters

Particle filter is a nonparameter filter method which can approximate any noise distribution. No matter the noise is nonliear or linear, gaussian or nongaussian. UKF uses unscented transform approach is similar to particle filter. Particle filter doesn't use specify parameters to define the noise distribution. It uses a lot of particles to approximate the noise distribution. Each particle represents a distribution of noise and each of particle gets a weight in whole particles. After each measurement comes in, algorithm updates each particle's weight and resampling the particles. The mean of new weighed particles becoming the final belief in this time step.

Algorithm MCL(X_{t-1}, u_t, z_t):

$\bar{X}_t = X_t = \emptyset$

for $m = 1$ to M:

$x_t^{[m]}$ = motion_update($u_t, x_{t-1}^{[m]}$)

$w_t^{[m]}$ = sensor_update($z_t, x_t^{[m]}$)

$\bar{X}_t = \bar{X}_t + \left\langle x_t^{[m]}, w_t^{[m]} \right\rangle$

endfor

for $m = 1$ to M:

draw $x_t^{[m]}$ from $\bar{X}_t$ with probability $\propto w_t^{[m]}$

$X_t = X_t + x_t^{[m]}$

endfor

return X_t

Fig. 1. Particle fitler pseudo code

2.3 Comparison of KF and PF

KF and PF are all bayesian based filter. They are based on the same theory but using different kind of parameters to approach the posterior belief of robot. The comparison table is shown as below. The most important keypoint is MCL based particle filter can approximate any noise distribution. [1]

	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. 2. KF and PF comparison

3 Simulations

In order to figure out the robot pose in a given map, ROS based simulation was implemented. The simulation platform is consist of GAZEBO, RVIZ, ROS. Not only the udacity bot was generated, but also author specified robot was generated. Both of them were simulated in above platform.

3.1 Achievements

With the simulation results, it can be confirmed that both of udacity robot and author -defined robot reached final goal position.

3.2 Benchmark Model

3.2.1 Model design

The udacity bot is consist of 2 wheels and a chassis. The sensor system is consist of a LIDAR and a camera. Robot model is defined in Gazebo and RVIZ with URDF file which is a description of a robot in HTML format. The robot model is defined based on links and joints. Udacity basic robot has 4 links and 4 joints. Camera sensor and LIDAR sensor based on the GAZEBO plugin are used for publishing sensor raw data. [2]

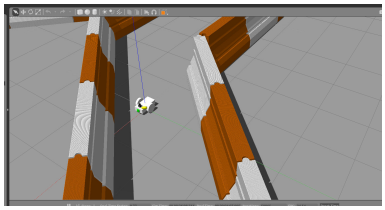

Fig. 3. Udacity robot in GAZEBO

3.2.2 Packages Used

The packages used in this project are shown as below.

1) Navigation stack. Navigation stack package based on 2D navigation goal, static map, particle cloud, robot footprint, obstacles, inflated obstacles, unknown space, global plan, local plan, planner plan and current goal modules to realize the navigation mission.
2) Costmap 2d. Costmap 2d package provides an implementation of a 2D costmap that takes in sensor data from the world, builds a 2D or 3D occupancy grid of the data, and inflates costs in a 2D costmap based on occupancy grid and a user specified inlation radius.
3) AMCL package. AMCL package is used for localize the robot position. AMCL is adaptive monte carlo localization. The word 'adaptive' means algorithm can adjust the quantity of particles for trade off the computation efficiency and localization accuracy.
4) GAZEBO, RVIZ plugin. These packages are used for transmit the sensors data.
5) pre-made map. It is also called global map.

3.2.3 Parameters

The key parameters are set as below. The detail parameters can be found in github. [3] The local cost map and global cost map layer is for navigation package. [4]. obstacle range, rayrace range and inflation radius are set in here.

TABLE 1
Local costmap Parameter Table

update frequency	10.0
publish frequency	10.0
width, height	5.0
resolution	0.05
static map	false
rolling window	true

TABLE 2
Global costmap Parameter Table

update frequency	10.0
publish frequency	10.0
width, height	40.0
resolution	0.05
static map	true
rolling window	false

3.2.4 results

The results are shown as below. Finally the udacity bot reached the goal position by using the navigation stack, costmap 2d and amcl package.

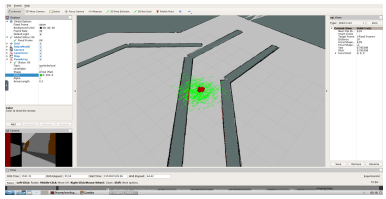

Fig. 4. Udacity robot in rviz with sensor measurement

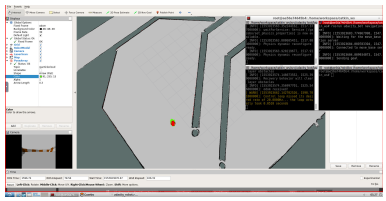

Fig. 5. Udacity robot in rviz while turning

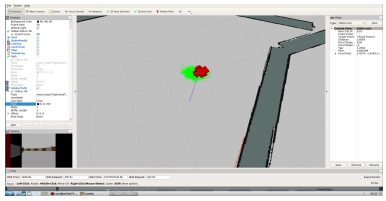

Fig. 6. Udacity robot reaches goal position

The pose array showed that the robot converge to it's position in few steps and ran with a smooth path. It showed that all parameters were tuned appropriately.

3.3 Personal Model

3.3.1 Model design

The re-designed model is shown as below. The sensor position doesn't change. The length of robot and the height of robot is larger. So the robot's turning radius becomes longer. It needs more space to turning.

3.3.2 Packages Used

The key parameters are set as below. The detail parameters can be found in github. [3] The local cost map and global cost map layer is for navigation package. [4]. obstacle range,

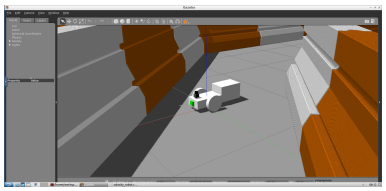

Fig. 7. re-designed robot truck

raytrace range and inflation radius are set in here. There are nothing changes from udacity bot except added some new links and joints.

3.3.3 Parameters

There is no changes about used packages. It is same with udacity bot.

1) Navigation stack. Navigation stack package based on 2D navigation goal, static map, particle cloud, robot footprint, obstacles, inflated obstacles, unknown space, global plan, local plan, planner plan and current goal modules to realize the navigation mission.
2) Costmap 2d. Costmap 2d package provides an implementation of a 2D costmap that takes in sensor data from the world, builds a 2D or 3D occupancy grid of the data, and inflates costs in a 2D costmap based on occupancy grid and a user specified inlation radius.
3) AMCL package. AMCL package is used for localize the robot position. AMCL is adaptive monte carlo localization. The word 'adaptive' means algorithm can adjust the quantity of particles for trade off the computation efficiency and localization accuracy.
4) GAZEBO, RVIZ plugin. These packages are used for transmit the sensors data.
5) pre-made map. It is also called global map.

3.3.4 Results

The results are shown as below. Finally the udacity bot reached the goal position by using the navigation stack, costmap 2d and amcl package. There are also no differences with udacity bot.

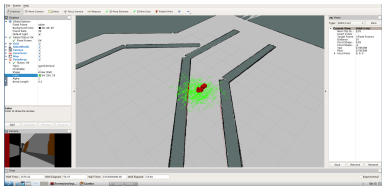

Fig. 8. Re-designed robot in rviz with sensor measurement

The pose array showed that the robot converge to it's position in few steps and ran with a smooth path. It showed that all parameters were tuned appropriately. However, it can be found that robot truck's turning rate is pretty long. The algorithm seems like based on costmap, calculates the path which is far away from obstacle.

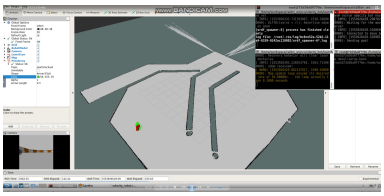

Fig. 9. Re-designed robot in rviz while turning

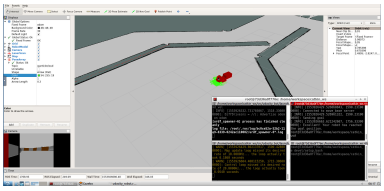

Fig. 10. Re-designed robot reaches goal position

3.4 Localization Results Technical Comparison

Both of the robots reach goal position with provided c++ navigation goal file and get a good localization accuracy. However, robot truck took more time than udacity bot. The reason is robot truck gets longer than udacity bot and it needs larger turning radius. Another reason is, truck robot is heavier than the udacity bot. So with the same parameters setting, truck runs slowly and with a larger turning rate.

4 Conclusion / Future Work

In this paper, many packages are used for handling where is the robot problem. Not only localization , the navigation stack package are used for path planning. With the amcl package help, the localization performance is good enough for the robot reaching goal position. The costmp 2d provides a local path planning and global path planning solution for the robot navigation problem. By tuning the parameters for navigation stack and amcl package, robot performance meets the requirement. In the future, since the world is 3D world, so 3D localization and 3D navigation related algorithms should be researched.

4.1 Hardware Deployment

1) What would need to be done? If the packages implemented in embedded hardware, then the packages input interface should be modified appropriately. Especially the speedometer should as accurate as it can be. Then the sensor's drivers also should be installed appropriately.
2) Computation time/resource considerations? With current computation requirement, raspberry pi 3 B is good enough for low speed localization and navigation.

References

[1] S. Thrun., *Probabilistic robotics*.
[2] "https://www.udacity.com,"
[3] "https://github.com/fred159/udacity robotics term2 p2 where am i,"
[4] "https://wiki.ros.org/navigation/tutorials,"