Robot Localization

Jackal Cao(Xinlei Cao)

Abstract—Localization is a complex problem for robots to be able to accurately find themselves, especially when the robot is constantly moving. This paper describes the way to achieve robot localization in simulation environment of ROS system. It is mean that the robot can effectively avoid obstacles and correct driving to an aim by good localization and route planning. This paper introduces the difference between Kalman filter and particle filter which are two main models for mobile robot localization, and also explains the choose. The style of robot in this paper is designed by the author. This article also describes the parameters and configuration in the project, as well as the space for improvement in future work.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

Robot navigation means the ability of robots to determine their own position in the frame of reference and then to plan a path towards some goal location. In order to navigate in the environment, the robot or any other mobility device requires representation, i.e. a map of the environment and the ability to interpret that representation. There are three question waiting for the robot in navigation: ”Where am I?”, ”Where should I go?” and ”How can I get there?”. The first question is described as localization and there are two kinds of methods to solve this problem. The relative localization method as like Visual Odometry. And the absolute localization method as GPS or probabilistic method. GPS is usually used in navigation apps of cars and phones. It’s convenient to deploy but not so accurate and need external signal. So GPS is not suitable for indoor environment. Probabilistic method like Monte Carlo Localization and Kalman Filters use sensor data to estimate a robot’s pose. This paper use probabilistic method to solve the problem ”Where am I?”

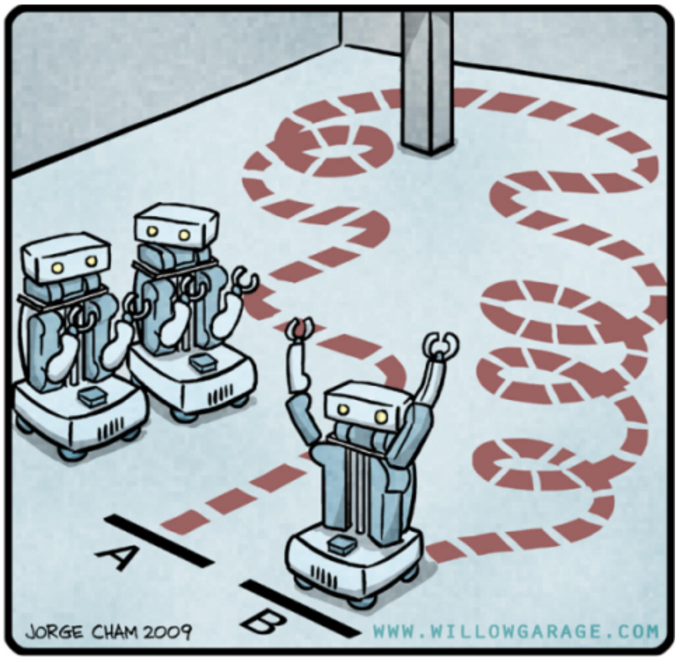


Fig. 1. Robot Navigation[1].

2 Background

Without localization, robot may be never arrive the correct place. If the robot thinks it is in the kitchen but in fact it is in the outside street, the robot will lose it’s way and never come back. When a robot has been set in a known map(global cost map), robot can try to use sensor like laser to perceive the surrounding environment and generate a dynamic map(local cost map). For constantly updating these maps in the ROS system and make sure they are always adapted, robots can use probabilistic algorithms to estimate their position. Two probabilistic algorithms are mostly used, one is the Kalman Filters and the other is Particle Filters with Monte Carlo Localization method. The project choise the Particle Filters. Here is the comparison of them.

2.1 Kalman Filters

Kalman Filter(KF) is an algorithm that uses the linear system state equation to estimate the state of the system through the input and output of the system. Since the observed data includes the effects of noise and interference in the system, the optimal estimate can also be seen as a filtering process. The KF in robot localization is a cycle of state prediction and measurement update by caculate a Gaussian distribution. Then estimate the location of robot. However, the Kalman Filter can only use in linear system but the real world is nonlinear. The KF is limited to update the linear mean and linear variance which are component of the Gaussian distribution. To solve this problem, a nonlinear function can update the mean and the linear approximation can be obtained by Taylor Series to update the variance in nonlinear system. This is called Extended Kalman Filter(EKF).

2.2 Particle Filters

Monte Carlo Localization(MCL) is also known as particle filter localization which is an algorithm for robots to localize using a particle filter. The MCL is usually accompanied by a group of particles. Each particle can be seen as a hypothesis that the robot's localization at this position. All particles work based on the Bayes rule.

2.3 Comparison / Contrast

This project is modeled in 2D environment of simulation, so the noise will not a simple Gaussian distribution. MCL can deal with any kind of noise but EFK can only deal with Gaussian. Second, MCL is easier to implementation than EKF. The third reason to choise MCL is the number of particles which effect memory and resource can be controled. The last advantage of MCL is Global Localization. For robot localization, MCL is almost better than EKF.

3 Simulations

The simulation of this project is ROS system(include basic component such as Gazebo, RViz). Cmakelist file should be modified for the project, include need message types, set the C++11 compile option, link the navigation.cpp and so on.

3.1 Achievements

This project need two robot models, one is limited named "udacity-bot", the other is created by author named "my-bot". Robot need more packages to sence and move. Tha MCL has different kinds of parameters to tune. Paremeters include laser, odom, and so on. All these are discuss below.

3.2 Udacity Robot

3.2.1 Model design

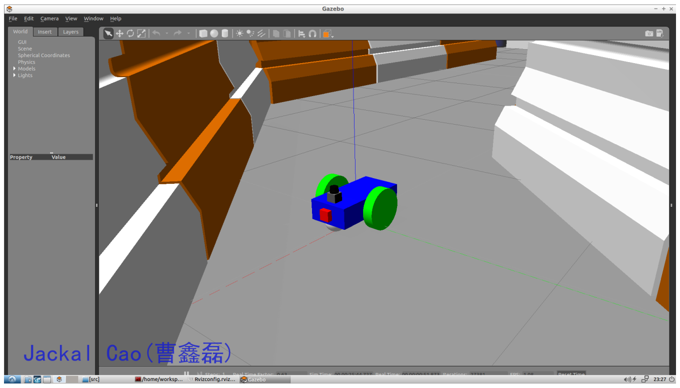


Fig. 2. Udacity Bot.

TABLE 1
Udacity Robot Model

component	shape	long-size	wide-size	high-size	radius
chassis	box	0.4	0.2	0.1	No
two casters	sphere	No	No	No	0.05
two wheels	cylinder	0.05	No	No	0.1
camera	box	0.05	0.05	0.05	No
hokuyo	box	0.1	0.1	0.1	No

3.2.2 Packages Used

The ROS system use messages to achieve all functions. the packages are used in this project as below.

- ros-navigation: Robot can caculate path with this package.
- ros-map-server: Maps can be create in the simulation by this package.
- ros-move-base: Public move command and subscribe messages from ROS
- ros-amcl: The package of MCL.

3.2.3 Parameters

Follow the class and modify parameters below.
AMCL Parameters:

- transform_tolerance: Set to 0.2
- initial_pose_(x,y,a): Set x and y to default value of 0.0, and the pose a can decide whether the costmap and the real map are aligned in the initialization, which makes good effect in the first movement. See more in the image below.

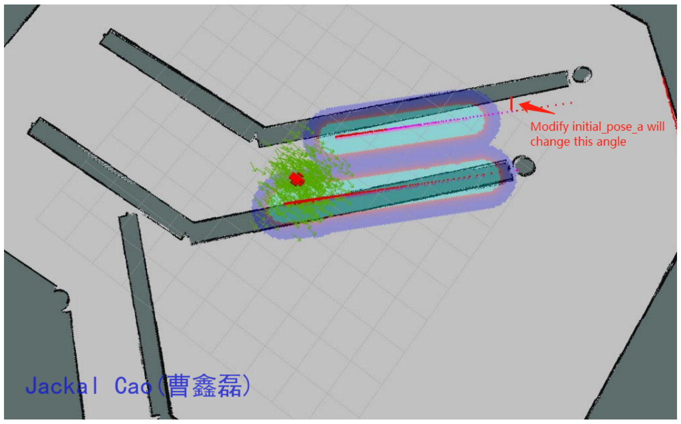


Fig. 3. Parameter Initial Pose

- particles numbers: Set min_particles to 100 and max_particles to 500, this paramater can be set between 100 to 1000, over 1000 seems no more improve.
- The next four odom_alpha parameters which decide particles distribution are very important, modify there parameters will make robot prefer to go straight or turn more.
- odom_alpha1: Set to 0.008
- odom_alpha2: Set to 0.2
- odom_alpha3: Set to 0.3
- odom_alpha4: Set to 0.005
- the effect of these for odom parameters like this:

Move_base Parameters:

- controller_frequency: Set to 0.6, help to avoid warning.
- meter_scoring: Set to true, help to avoid warning.
- obstacle_range: Set to 5.0, this parameters decide the range of robot perception for obstacles.
- raytrace_range: Set to 7.0
- inflation_radius: Set to 0.7, this parameters decide how far the global navigate path away from obstacles.

- update_frequency(two costmap): Set to 4.0, take no warning.
- publish_frequency(two costmap): Set to 4.0, take no warning.
- width and height of local_costmap: Set to 5.0
- pdist: Set to 1.6
- gdist: Set to 0.8

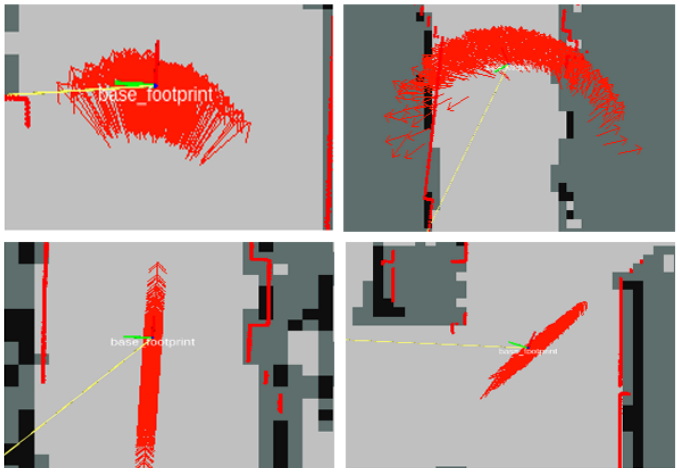


Fig. 4. Odom Parameter[2].

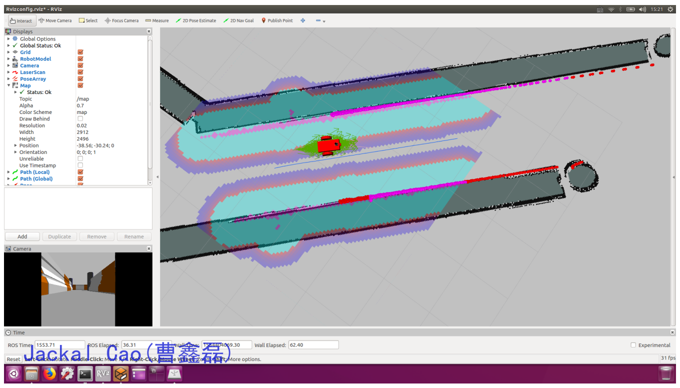


Fig. 5. Particles of Udacity robot

Width and height of local_costmap are most important parameters to modify. As the default value of 20x20, the robot would check the back way first then find nothing. Finally block or turn aroud again to find the goal. This frequently happens when using ”navigation.cpp”. Thanks to @Fernando in Slack. Let robot know just enough information will make it more smart. Parameters of pdist and gdist control the weight of two costmap when caculating. So make pdist a little high is better for robot to keep itself more close to the global path(the blue line in RViz)

3.3 My Robot

What My Robot looks like is the image below.

3.3.1 *Model design*

3.3.2 *Packages Used*

Packages are same as the Benchmark Model

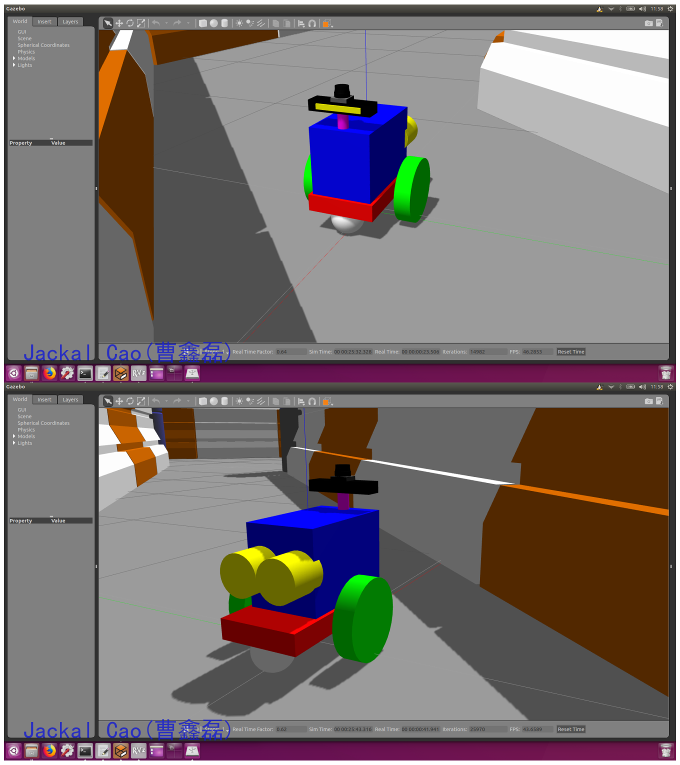


Fig. 6. My Robot(Front and back)

TABLE 2
My Robot Model

component	shape	long-size	wide-size	high-size	radius
chassis	box	0.4	0.2	0.05	No
two casters	sphere	No	No	No	0.05
body	box	0.3	0.18	0.2	No
two turbos	cylinder	0.1	No	No	0.05
neck	cylinder	0.12	No	No	0.015
camera_box	box	0.05	0.18	0.03	No
two wheels	cylinder	0.05	No	No	0.1
camera	box	0.01	0.13	0.015	No
hokuyo	box	0.1	0.1	0.1	No

3.3.3 *Parameters*

- particles numbers: Set min_particles as 50 and max_particles as 250.
- the odom_alpha parameters make the particles more concentrated
- odom_alpha1: Set to 0.005
- odom_alpha2: Set to 0.01
- odom_alpha3: Set to 0.01
- odom_alpha4: Set to 0.005
- Other parameters are same as Udacity robot.

4 RESULTS

As the result, all of two robots can arrive the goal, and seems they are full of power. No run around, no stuck.

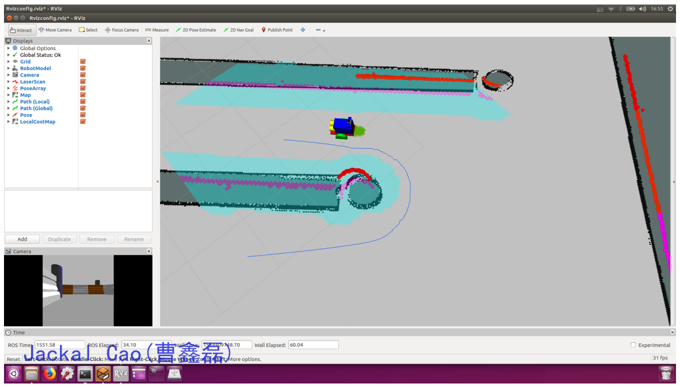
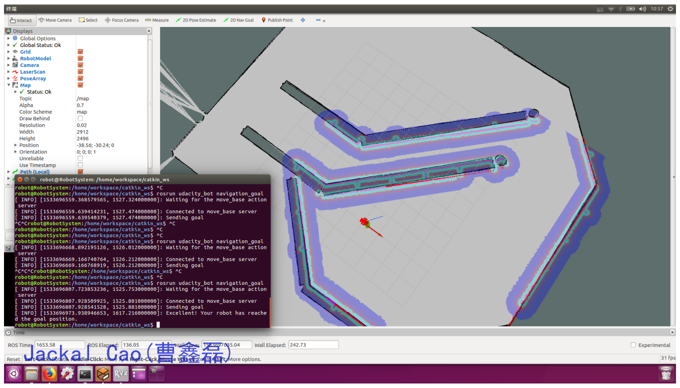


Fig. 7. Particles of My Robot

4.1 Localization Results

4.1.1 *Udacity robot*

After 30 times of test. The Udacity robot can arrive the goal as the image below:



```
robot@RobotSystem:/home/workspace/catkin_ws$ ^C
robot@RobotSystem:/home/workspace/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1533696559.368579565, 1527.324000000]: Waiting for the move_base action
 server
[ INFO] [1533696559.639414231, 1527.474000000]: Connected to move_base server
[ INFO] [1533696559.639540379, 1527.474000000]: Sending goal
^C^Crobot@RobotSystem:/home/workspace/catkin_ws$ ^C
robot@RobotSystem:/home/workspace/catkin_ws$ ^C
robot@RobotSystem:/home/workspace/catkin_ws$ ^C
robot@RobotSystem:/home/workspace/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1533696668.892195126, 1526.012000000]: Waiting for the move_base action
 server
[ INFO] [1533696669.166740764, 1526.212000000]: Connected to move_base server
[ INFO] [1533696669.166768919, 1526.212000000]: Sending goal
^C^C^Crobot@RobotSystem:/home/workspace/catkin_ws$ ^C
robot@RobotSystem:/home/workspace/catkin_ws$ ^C
robot@RobotSystem:/home/workspace/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1533696807.723853236, 1525.753000000]: Waiting for the move_base action
 server
[ INFO] [1533696807.928509925, 1525.881000000]: Connected to move_base server
[ INFO] [1533696807.928541528, 1525.881000000]: Sending goal
[ INFO] [1533696973.938946653, 1617.216000000]: Excellent! Your robot has reache
d the goal position.
robot@RobotSystem:/home/workspace/catkin_ws$
```

Fig. 8. Success of Udacity robot

4.1.2 *My Robot*

The image of My Robot arriving the goal as below:

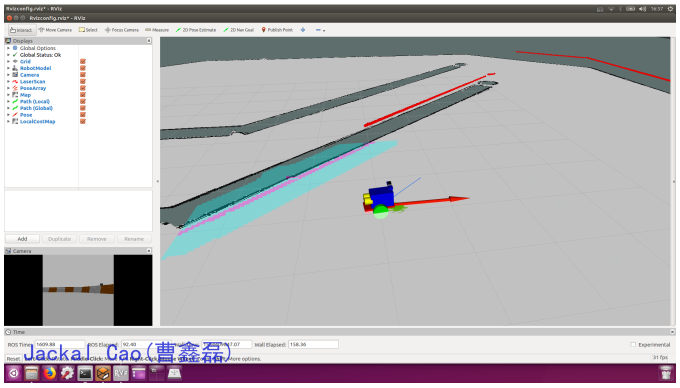


Fig. 9. Success of My Robot

4.2 Technical Comparison

Before modify the width and height of local_costmap. Robots always like going to off-path. Turn around and come back again. But this takes 10 minutes to 1 hour. After that, both udacity-bot and my-bot can arrive the goal in 2 minutes. Robots can almost along the global path(the blue line in RViz!) so the default parameters gdist_scale and pdist_scale seems great.

5 Discussion

Importance of parameters: As this project, the MCL is used in the robot localization task, and parameters that takes most effects to the result are size of local cost map and odom_alpha. Importance of map design: If the map has multiple paths, long dead ends and other unfavorable factors for navigation, it can seriously affect the local path caculating of robot. Importance of robot model design: The robot model should avoid unbalanced design to keep the center of gravity.

5.1 Topics

- Which robot performed better? The Udacity robot is better than My Robot.
- Why it performed better? (opinion) The robot model is so easy and tiny for localization. Every components which add for new robot may take new problem. As like balance, velocity and turning.
- How would you approach the ’Kidnapped Robot’ problem? Robot may be lost direction in the ’Kidnapped Robot’ test. The global costmap and local costmap may be not aligned. When running this project in Gazebo and RViz, move robot to anywhere in Gazebo. Then put a goal, and robot will find hard to arrive this goal. Modify the parameter ”static_map” to false and ”rolling_window” to true in global costmap config file will make it better. How ever, AMCL can not deal with ’Kidnapped Robot’ along well.
- What types of scenario could localization be performed? First is to check if two costmap are all correct before moving, and if wall in sensor but not in costmap. Second is to check if this place has been seen before. These two kinds of scenarios will make it better.
- Where would you use MCL/AMCL in an industry domain? MCL/AMCL is valuable for auto-drive, express delivery and warehouse management when people use robot to finish these works.

6 Conclusion / Future work

This project is successful for the purpose to study how robot can make localization by itself. With this recap, the robot can not do well in the initial state, say the parameters as like the number if particles or the frequency for map update. To many particales need lots of calculation, hence the robot usually waiting for the result in the half way. But less particales will make the opposite result - run around. So the main job is to find the balance among caculation resource, complex robot design and suitable parameters. There are also something to try in the future work is valuable.

6.1 future work

- Try different base size and base shape and more complex design.
- Try other type of actuator as like four wheels or tracked wheels.
- Babys alway like climb anywhere but they don't know how to protect themselves. If the robot can localization one baby and use gesture recognition to sensor the danger and alarm their parents in time, that's so amazing.

7 REFERENCES

[1] ROS.org, "navigation":http://wiki.ros.org/navigation/
[2] Blog.csdn.net, "ROS Navigation amcl introduction": https://blog.csdn.net/qq_29796781/article/details/80001355