Where Am I

Sim Kae Wanq

Abstract—Robot localization is the process of determining where a mobile robot is located with respect to its environment. Adaptive Monte Carlo Localization (AMCL) approach was introduced in this project. The goal of this project is to configure a number of ROS packages that used in conjunction to accurately localize and navigate mobile robot to goal in a provided map. The final result showed that mobile robot could reach goal although it is not best optimal configuration.

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

1 Introduction

This project consists of creating a complete ROS package that includes Gazebo world, a robot model in URDF, and the ROS navigation stack. The robot has to localize itself in the virtual world, avoid obstacle and finally move to a desired destination and pose.

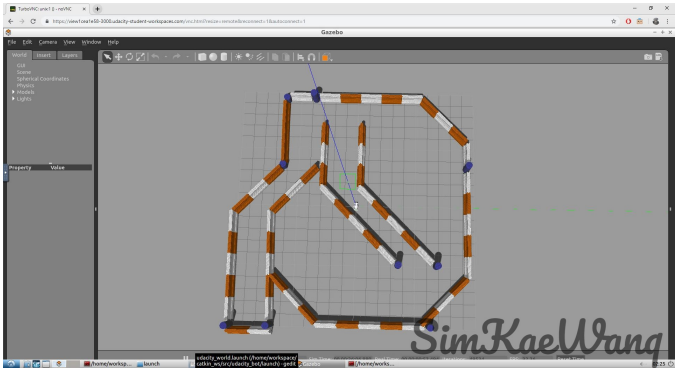

Fig. 1. the virtual world

2 Background

Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robot's own location is an essential precursor to making decisions about future actions. [1]

For a robot to localize itself on a map, it needs to filter noisy sensor data using probabilistic methods. The problem includes:

1) Position tracking: robot's initial position is known, algorithm keeps track of the robot's position and movement.
2) Global localization: robot's initial position is unknown, and to be determined when moves. This problem combines the uncertainties from measurements and actions in a cycles to achieve a precise estimate of location.
3) Kidnapped robot: to recover from abrupt changes in position. It's difficult for Bayesian algorithms to recover from since they preserve an internal belief that interfere with the new robot state

Two localization methods were learned: Kalman Filters and Particle Filters. Both probabilistic methods can be successfully applied to solved first two problems mentioned above in localization process.

2.1 Kalman filter

The Kalman filter (KF) estimates the value of a variable by updating its estimate as measurement data is collected filtering out the noise. KF models the state uncertainty using Gaussians and it is capable of making accurate estimates with just a few data points.

KF starts with an initial state estimate then performs the following cycle: measurement update produced by sensor measurements followed by a state prediction from control actions. [2]

2.1.1 Multidimensional Kalman Filter

Most of the real world problem happens in multi dimensions, i.e. a drone's position is defined by x, y, z coordinates and roll, pitch, yaw angles. In N-dimensional, the state is a vector with N state variable. The Multidimensional Kalman Filter (MKF) algorithm consists of calculating the Kalman Gain K, that determines how much weight should be placed on state prediction, and how much on the measurement update. It's an averaging factor that changes depending on the uncertainty of measurement update P and state update x.

$$K = P'H^T S^{-1}$$

$$x = x' + Ky$$

$$P = (I - KH)P'$$

2.1.2 Extended Kalman Filter

KF assumes that motion and measurements models are linear and can be represented by unimodal Gasussian distribution. However, non-linear actions will result in non-Gaussian posterior distributions that cannot be properly modeled by a closed form equation. Thus, Extended Kalman Filter (EKF) approximates motion and measurements to linear functions locally (i.e. by using the first two terms of a Taylor series)

to compute a best Gaussian approximation of the posterior covariance. This trick allows EKF to be applied efficiently to a non-linear problems.

2.2 Particle filters

Monte Carlo localization algorithm similar to Kalman Filters estimates the posterior distribution of a robots position and orientation based on sensory information but instead of using Gaussians it uses particles to model state.

The algorithm is similar to KF where motion and sensor updates are calculated in a loop but MCL adds one additional step: a particle resampling process where particles with large importance weights (computed during sensor updates) survive while particles with low weights are ignored.

In the MCL example below all particles are uniformly distributed initially. In the following update steps the particles that better match the predicted state survive resulting in a concentration of particles around the robot estimated location.

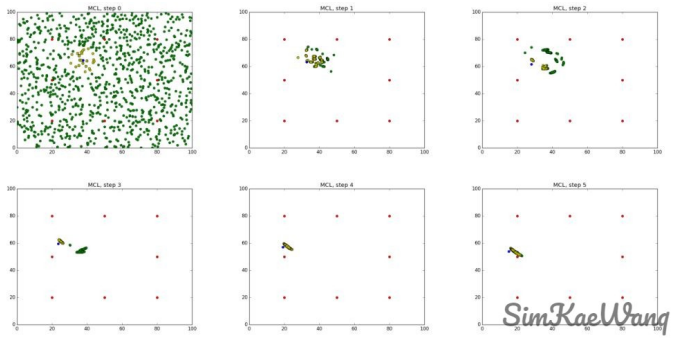

Fig. 2. MCL steps visualization

2.3 Comparison / Contrast

In this project, MCL was selected because of ease of implementation. Second, MCL represents non-Gaussian distributions, and can approximate any other practical important distribution. MCL able to model a much greater variety of environments compared to EKF. Besides, in MCL, computational memory and resolution of solution can be configured by changing the particles distributed uniformly and randomly throughout the map. Full comparison table between EKF and MCL can be found at appendix, table 1.

3 Simulations

This section introduces the simulation of localization and navigation.

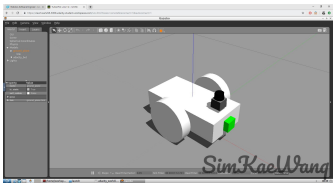

Fig. 3. standard robot model

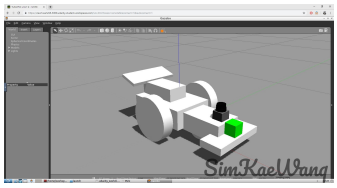

Fig. 4. implemented robot model

3.1 Robot model

Upon setting up the virtual world and launch files, a standard robot with a base, 2 actuators, a camera and a laser sensor, as shown in Fig 3 was model. On top of the standard robot model, the robot was re-design with new look as shown in Fig 4. This robot has a differential actuation and 2 wheels in order to comply with the ROS navigation stack hardware requirements.

3.2 Configuration Tuning

In this project, 2D navigation stack was used, which takes information from odometry, sensor streams, and a goal pose and output safe velocity commands that are sent to mobile robot. [3] However, the default setting of each parameters was not optimized to have a very great performance. Thus, in following section, ration of each parameter tuning of ROS packages were discussed below

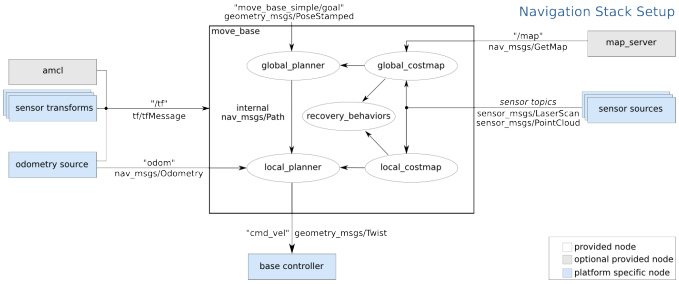

Fig. 5. Overview of navigation stack [4]

3.2.1 *base_local_planner_params.yaml*

In base_local_planner_params.yaml file, `controller_frequency` was added and reduced to avoid control loop delay warning.

3.2.2 *costmap_common_params.yaml*

In costmap_common_params.yaml file, first to increase the `transform_tolerance` to 0.3, to eliminate the transform timeout warning.

The distance range that the costmap should be updated based off the laser reading to either add or remove obstacles, `obstacle_range` and `raytrace_range` were set to 3.0 and 2.5 accordingly.

To avoid having the robot bumping on the walls, `inflation_radius` was increased to 0.6. This parameter defines a padding added to obstacles. The navigation planner takes the padding into account when calculating global path.

3.2.3 *global_costmap_params.yaml*

In global_costmap_params.yaml, to avoid the map update loop warning, `update_frequency` and `publish_frequency` were decreased to 0.8. This warning occurred because maps not getting update fast enough on a slow system.

3.2.4 local_costmap_params.yaml

Similar to global_costmap_params.yaml, to avoid the map update loop warning, `update_frequency` and `publish_frequency` were decreased to 0.8. Besides, the `width` and `height` of local costmap were reduced to 5.0 from 20.0. The local costmap size was by far the most important parameters to get right to successfully navigate around the corner at the end of the corridor. This is because the goal creates a huge influence over the local costmap. This resulted in the robot getting pulled away from the calculated global path. Reducing the size of the local costmap to approximate to the corridor width solved this problem.

3.2.5 amcl.launch

There were several parameters that could be tuned to improve the performance of AMCL localization package, and can be categorized into three: overall filter, odemetry, and laser. In this project, only parameters under overall filter category were changed.

In overall filter category, number of particle was decrease and limit to `min_particles` more than 5 and `max_particles` less than 100. It's greatly reduced compared to default setting, due to the limitation of system performance. When the low performance system couldn't support huge number of particle and calculation, the result of AMCL get worse. Besides, the default values of `update_min_a` and `update_min_d` were too large for this slow moving robot. By decrease these two parameter values to 0.005 and 0.01, state updates received more frequent, resulting in a reduction of the location uncertainty.

4 Results

Two robot were tested in this project- standard mobile robot with default setting and custom mobile robot with tuned parameters. Both of them have similar mass, size and actuators.As result of running ros node `navigation_goal`, both robots were able to reach the goal. However, standard robot with default setting spend much longer time in reaching goal, and may failed.

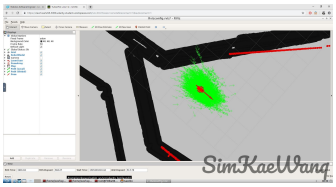

Fig. 6. standard robot reach goal position with pose array shown

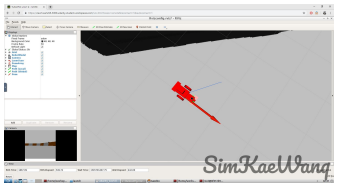

Fig. 7. custom robot reach goal position with pose array shown

4.1 Localization Results

As shown in Fig 6, 7, 8, 9, pose array of custom robot is much lesser but more concentrated compared to pose array of standard robot. The orientation of each particles were more accurate compared to standard robot's.

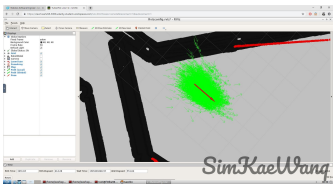

Fig. 8. standard robot pose array at goal

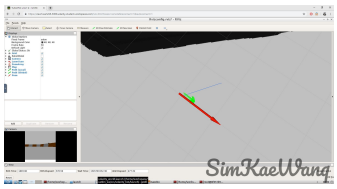

Fig. 9. custom robot pose array at goal

5 Discussion

As shown in the result above, custom robot with tuned parameter performed much better than standard robot. It performed better in term of particles accuracy, duration to reach the goal.

Particles accuracy was greatly improved when the number of particle was reduced to 5100 unit. Reduction of number of particle also cut down the system calculation time in each cycle, as the system performance was quite limited.

Duration to reach the goal as a result of accurate localization and small local costmap size. With default setting, the standard robot tried to run away from the global path and attempt to head straight to the goal, and bounce off the obstacle. It reached to goal eventually after many round of try and error with bouncing the wall.

It is important to point out that the robot would fail to localize when moved to a different locations. AMCL is a Bayesian algorithms that holds an internal belief of the world that is difficult to change in case of abrupt changes in environment. One way to go about would be to reset the AMCL internal state, and make it back to uniform distribution of particles, when robot was placed a new location. This would reduce the kidnapped robot global localization problem mentioned above.

6 Future Work

AMCL algorithm used in this project would work well in industry domains that use mobile robots in environments that have walls or fences as obstacles, but parts of environment couldn't be too similar. In future work, improvement of the navigation planner will make robots more efficient in getting shorter path and commercially viable.

Below listed several works that can be done for improvement:

- GPU with greater performance to reduce the load of data processing.
- Test and solve kidnapped robot problem
- Deploy the algorithm in actual robot and environment to verify the performance.

References

[1] S. H. G. Dissanayake, "Robot localization: An introduction," tech. rep., 2016.
[2] "Kalman filters."
[3] "navigation."
[4] "Setup and configuration of the navigation stack on a robot."

7 Appendix

TABLE 1
Comparison between EKF and MCL

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