Where Am I? Writeup

Shingo Mori

Abstract—A ROS package is created and mobile robots performing 2D navigation in a gazebo environment is implemented. ROS packages included in Navigation Stack are used and their parameters are tuned to improve the localization results. Two models, Benchmark Model and Personal Model, are created. Parameters are configured for each model, and both robots are able to reach the goal with reasonable localization accuracy.

Index Terms—Udacity, ROS, SLAM, Navigation Stack, AMCL.

1 Introduction

Localization is the most basic perceptual problem in robotics. Almost all the mobile robot require knowledge of its location. Since the pose of a robot usually can not be measured directly, it must be inferred from input data. To deal with various uncertainties that come from the real world, an probabilistic approach [1] is widely applied.

There are three types of localization problems: Position tracking, global localization, and kidnapped robot problem. Position tracking, also called as local problem, is the easiest of all. The initial pose of a robot is known, so the localization is done by filtering out the measurement noise as the robot moves around. Methods for position tracking often assume pose uncertainty is small. In global localization, the initial robot pose is unknown. The pose uncertainty is not bounded to local regions near the robot, which makes the problem more difficult than position tracking. Kidnapped robot problem, a variant of global localization, is even more difficult. A robot gets teleported to another location during the operation. The robot has to properly abandon the wrong belief of its pose and recover from the failure.

Another thing that has critical impact on localization problem is the environment. An environment where the robot is the only object that moves is called a static environment. On the contrary, an environment containing other objects that change their pose over time, it is called a dynamic environment. Localization in a dynamic environments is obviously more difficult than localization in a static one.

In this project, a ROS package is created to solve global localization problem in a static environment. A mobile robot spawns in a gazebo environment as shown in Fig. 1 and attempts to move to its final pose. The goal of this project is to utilize ROS packages to accurately localize the robot and successfully navigate it to the goal position.

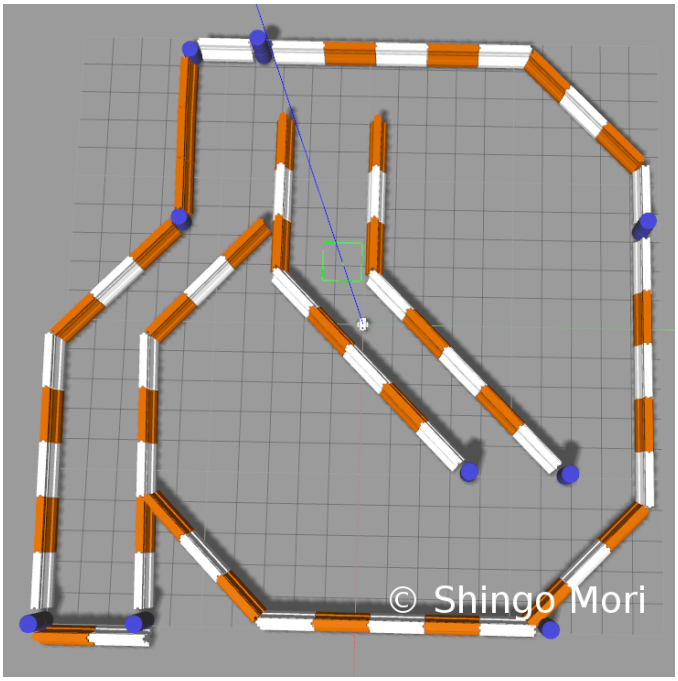


Fig. 1: 2D navigation environment.

2 Background

There are several approaches to perform localization. Two major methods, the Kalman Filter and the Particle Filter, are covered in the classroom [2]. Each method is a derivative of the Bayes Filter, an algorithm that recursively updates its belief distribution using incoming sensor and control data.

2.1 Kalman Filters

Implementations of the Kalman Filter represent their beliefs by Gaussian distribution. Hence the algorithm assumes the unimodal distribution around the true pose of the robot. Because of its computational efficiency, Kalman Filters are applied to many tracking problems in robotics, where the posterior is often focused around the true pose with a limited amount of uncertainty.

2.2 Particle Filters

Particle Filters are implementations of the Non-parametric Filter, where the posterior belief is represented by a finite number of hypothetical values. In Particle Filters, those values are represented by random samples of robot state. They do not assume mathematical form of posterior. Hence their state space is multimodal and distinct.

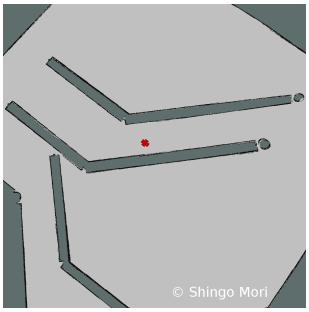
2.3 Comparison / Contrast

Particle Filters have various advantages over Kalman Filters. First, Particle Filters solve global localization problem better than Kalman Filters, and some implementations can even solve kidnapped robot problems. This is because of the multi-modal and distinct characteristics of Particle Filters. Second, Particle Filters can represent measurement noises as non-Gaussian distribution. This is useful when measurement noises often do not follow the Gaussian distribution, which is often the case in the real world. Third, the computational usage of Particle Filters can be controlled by configuring the number of particles. This is critical in robotics where computational resource is often limited. For the above reasons, an implementation of the Particle Filter is used in this project to solve global localization problem.

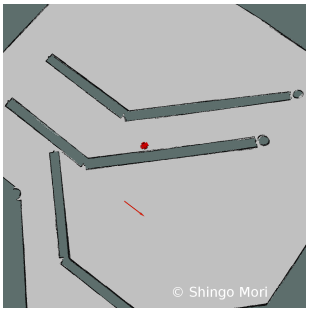
3 Simulations

A ROS package is created and configured to perform 2D global localization and navigation in a Gazebo environment. Two different robot models, a benchmark model and a personal model are specified in the Unified Robot Description Format (URDF). ROS packages provided in Navigation Stack [3] are used and their parameters are configured. RViz is used to visualize ROS topics.

All the simulation is conducted using ROS Kinetic on Ubuntu 16.04 LTS system with a Core i7-7700 CPU, two 8 GB DDR4 memories, and a NVIDIA GeForce GTX1080Ti GPU.



(a) Start pose. The belief is initially spread around the robot.



(b) Goal pose. This is specified in the provided .cpp file by Udacity.

Fig. 2: Start and goal poses.

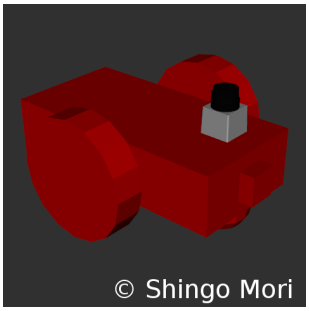
3.1 Achievements

2D global localization and navigation are achieved on both the benchmark model and the personal model. Fig. 2a shows the initial robot pose and Fig. 2b shows the goal pose of the robot. Green arrows represent particles created by the amcl package.

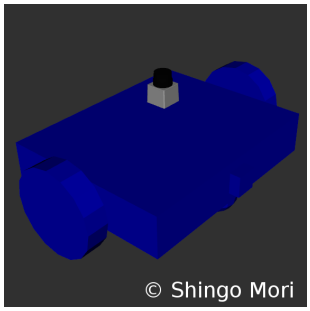
3.2 Benchmark Model

3.2.1 Model design

The benchmark model has a rectangular chassis (size=0.4m, 0.2m, 0.1m) with two spherical caster on the bottom and two wheels on the side. A camera and a laser range finder are attached to sense the environment. Inertial and collision parameters are specified to each links. An image of the model is shown in Fig. 3a.



(a) Benchmark Model



(b) Personal Model

Fig. 3: Robot models.

3.2.2 Packages Used

A ROS package is created and other packages such as amcl [4], base_local_planner [5], and costmap_2d [6] included in the Navigation Stack are utilized. The package uses the global map created and provided by Udacity.

The created package subscribes to sensor messages and publishes velocity commands.

3.2.3 Parameters

Parameters included in the Navigation Stack are configured. Only the parameters that have impact on the performances of the localization and the navigation are described in this section.

Parameters for the amcl package is shown in TABLE 1. min_particles, max_particles and transform_tolerance are configured based on the system resource. laser_min_range and laser_max_range are set according to the specification of the sensor. Because odom_alpha1-4 determine the uncertainty of the motion model, they have major impact on the localization accuracy. Since the controller noise of the robot is small, setting them to smaller values leads to a better result.

TABLE 1: Benchmark Model: amcl parameters

name	value	default
min_particles	100	100
max_particles	1000	5000
transform_tolerance	0.5	0.1
laser_min_range	0.1	-1.0
laser_max_range	30.0	-1.0
odom_alpha_1	0.001	0.2
odom_alpha_2	0.001	0.2
odom_alpha_3	0.001	0.2
odom_alpha_4	0.001	0.2

Parameters for the base_local_planner package is shown in TABLE 2. holonomic_robot is set to false because a differential robot is non-holonomic in a 2D environment. sim_time and pdist_scale are increased to prevent poor local path-planning which results in the robot running into a infinite loop movement.

Parameters for the costmap_2d package is shown in TABLE 3. There are two types of costmap, the global costmap and the local costmap. Common parameters are applied to both costmaps, while other parameters are configured

TABLE 2: Benchmark Model: base_local_planner parameters

name	value	default
holonomic_robot	false	true
xy_goal_tolerance	0.4	0.10
yaw_goal_tolerance	0.2	0.05
sim_time	5.0	1.0
pdist_scale	0.7	0.6

for each. transform_toloerance, update_frequency and publish_frequency are configured based on the system resource. robot_radius and inflation_radius are decreased to give more space for the robot to move around. obstacle_range and raytrace_range are increased to improve the efficiency of creating the local and global costmaps.

Width and height for the global costmap are set to 40 to cover the entire environment. static_map is set to true since the global costmap uses the provided map to create its costmap.

Setting width and height to 20 for local costmap is sufficient for the local path-planning. static_map is set to false to create and maintain its own map.

TABLE 3: Benchmark Model: costmap_2d parameters

name	value	default
common		
transform_tolerance	0.5	0.2
robot_radius	0.4	0.46
inflation_radius	0.4	0.55
obstacle_range	0.3	0.25
raytrace_range	0.4	0.3
global costmap		
update_frequency	10.0	5.0
publish_frequency	5.0	5.0
width	40	10
height	40	10
static_map	true	true
rolling_window	false	false
local costmap		
update_frequency	10.0	5.0
publish_frequency	5.0	5.0
width	20	10
height	20	10
static_map	false	true
rolling_window	true	false

3.3 Personal Model

3.3.1 Model design

Basic structure of the personal model is similar to the benchmark model, but with larger chassis (size=0.4m, 0.5m, 0.1m) and different placement of the laser range finder. An image of the model is shown in Fig. 3b.

3.3.2 Packages Used

Packages used for the personal models are the same as those used for the benchmark model.

3.3.3 Parameters

Most parameters for the personal model are the same as those for the benchmark model. Only the parameters configured specifically for the personal model are described in this section.

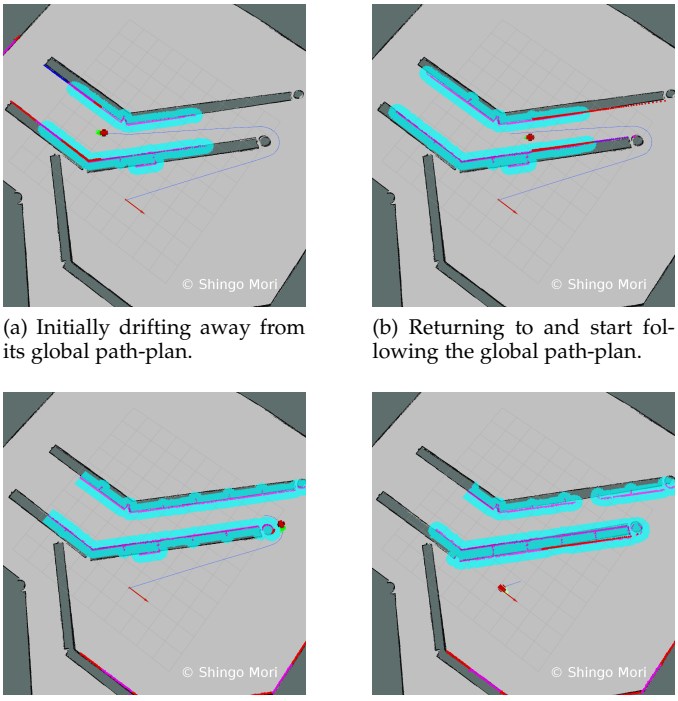
Parameters for the costmap_2d package is shown in TABLE 4. robot_radius is set according to the size of the robot. inflation_radius is set to maintain the same obstacle-free space as the one for the benchmark model.

TABLE 4: Personal Model: costmap_2d parameters

name	value	default
common		
robot_radius	0.5	0.46
inflation_radius	0.3	0.55

4 Results

Both the benchmark model and the personal model reaches the goal with sufficient localization accuracy. Although the robot drifts away from the global path when it starts its first movement (Fig. 4a), it comes back to the correct path after moving around a several seconds (Fig. 4b). After retuning to its correct path, the robot moves smoothly to the goal position without being stuck (Fig. 4c and Fig. 4d). Duration of the time for the robot to reach the goal is around around 90 seconds with the benchmark model, and around 100 seconds with the personal model.



(a) Initially drifting away from its global path-plan.

(b) Returning to and start following the global path-plan.

(c) Passing the corner without being stuck.

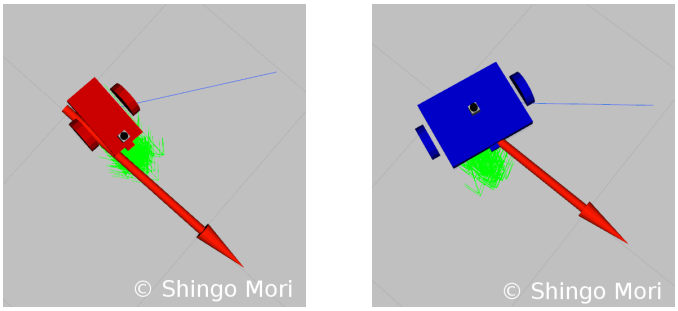
(d) Reaching the goal pose.

Fig. 4: Resulting movements.

4.1 Localization Results

4.1.1 Benchmark Model

As shown in Fig. 5a, the benchmark model is localized accurately at the goal position. It took around 15 seconds for the particle filter to converge.



(a) Benchmark Model (b) Personal Model

Fig. 5: Localization Results.

4.1.2 Personal Model

As Fig. 5b shows, the localization result of personal model is quite similar to the one from the benchmark model. It took around 15 seconds for the particle filter to converge.

4.2 Technical Comparison

The personal model has wider chassis than the benchmark model. This caused it harder for the personal model to curve. However, after configuring the inflation_radius, and the personal model is able to move around as freely as the benchmark model. Although the layout and the sensor position of the personal model is changed from the benchmark model, the noise parameters of the sensor model and the motion model remains the same. Hence, the localization accuracy stays almost the same despite the changes made on the layout.

5 Discussion

The performance of the benchmark model and the personal model are almost the same as mentioned in the section 4.2. This is because changing the layout did not affect the controller noise and the measurement noise of the robot.

Dealing with the parameters for the base_local_planner was the hardest. As described in the section 4, the robot initially drifts away from the global path. Since the weight of local path-planning is high, the robot seems to 'prefer' the local path over the global path if there is a conflict between the two. This causes the robot drifting away. However, simply increasing the weight of the global path-planning is not a straight forward solution; It reduces the flexibility of the overall path-planning and results in getting easily stuck in walls and corners. pdist_scale and sim_time must be carefully configured to get a good balance.

Particle filters converges quite easily by setting the small values to odom_alpha1-4. However, this is because the odometry noise and the measurement noise are small in this project. Noise parameters should be configured more carefully if the environment is more dynamic or noises are more significant.

Although the algorithm implemented in the amcl package is capable of solving the kidnapped problem, this feature is disabled by default. recovery_alpha_slow and recovery_alpha_fast must be set and configured properly for the algorithm to solve the problem.

This package can be used in an industry domain since the localization accuracy and the frequency (over 5Hz) is sufficiently high. However, some parameters should be configured in accordance to the system requirement.

6 Conclusion / Future work

Two models, Benchmark Model and Personal Model, performing 2D global localization and navigation are implemented in a ROS package. Both models generally follow their global path and reach the goal without being stuck in objects like walls and corners. The localization accuracy is high while the processing time remains reasonable. Adaptive characteristic of the amcl algorithm enables users to deploy the package on embedded hardwares just by configuring a few parameters.

For future work, the package is deployed on an actual device and tested outside of the simulated environment. Some parameters should be configured to meet the system requirement and deal with the complexity of the real world.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
[2] "Robotics software engineer nanodegree." Udacity, Last accessed 18 May 2018.
[3] "Navigation stack." ROS.org, Last accessed 13 May 2018.
[4] "amcl package." ROS.org, Last accessed 13 May 2018.
[5] "base_local_planner package." ROS.org, Last accessed 13 May 2018.
[6] "costmap_2d package." ROS.org, Last accessed 13 May 2018.