Report: Where am I project

Xuan He

Abstract—In this project, two robot models were built using Unified Robot Description Format (URDF). Each robot model has basic components, sensors (cameras and laser rangefinder) and actuators. In simulation environment, Gazebo plugins were incorporated to mimic the behaviors of camera sensor, the hokuyo sensor and the differential drive controller. Additionally, ROS navigation packages was utilized to localize the robot in a provided map for position reaching task. For navigation, standard ROS Navigation Stack such as AMCL (adaptive monte carlo localization) is used and its parameters are finely tunned to direct the robot towards goal in a provided static map.

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

1 Introduction

Most robotics software development are involved with system and hardware level implementations. In order to iteratively improve robot before manufacturing, engineers typically resort to simulation software which can ease the process of designing components, selecting sensors, changing materials and tuning parameters. These parameters influence dynamics configuration of a robot which subsequently influence the controlling program, planing algorithms and other tasks depending on them.For example wheel diameters and the distances need to be fed to the odometry algorithm and a differential driver. Cameras intrinsic and extrinsic parameters need to be known before we can use it for a perception layer.
A very important information of a robot is to know is its location in a surrounding space so it can reason about neighboring features and build plans for the movement goals. Two common types of localization are recognized: local and global. Local localization is determining robot pose and orientation based on odometry and perception sensors data before any matching to the map. Global localization, on the other hand combines output from local localization and matches it to the known map so to place the robot in exact position and orientation on the map. In other word, it is describing the translation and rotation from $odom$ frame to the map frame. In this project, we are building the basic URDF model of a robot, which represents the full links geometry descriptions along with inertial, visual and connection parameters. And full capabilities from ROS Navigation Stack are added to make robot planning and navigating on a provided static map.

2 Background

In robotics navigation and mapping, localization is the process of keeping track of the locations of robots within the known environment. Without localization, robots would have no idea its locations and it cannot make informed decision on how to plan its actions.
Popular localization algorithms for localization include different variations of Kalman filters and particle filters. Kalman filters are considered a sufficiently accurate algorithm that works great for small incremental errors that arise due to the sensors errors and motion models discrepancies (noise representation, wheel slippage etc) like odometry estimation and local robot pose. However, Gaussian assumption employed by Kalman filters is restrictive for global localization problem.
Monte Carlo Localization algorithms based on sampled particles and thus can represent any distribution function, that is not limited by a Gaussian model and is not linear. MCL is also much simpler in calculation and implementation compare to Kalman filters algorithms. MCL is also a good candidate when one can want to control memory and resolution control of a filter which can necessary for the small embedded computers with limited resources.

2.1 Kalman Filters

Kalman filter is an estimation filter to estimate a value of a variable in real times as the data is being collected such as the pose of robots. This filter is very prominent in control systems due to its accuracy and computational efficiency. It is good at taking in noisy measurements in real time and providing accurate predictions of the actual variables such as position. Kalman filter algorithm is based on three assumptions:

- State transition probability must be linear with added Gaussian noise.
- The measurement probability must also be linear with added Gaussian noise State space can be represented by a unimodal Gaussian distribution
- The initial belief, represented by the mean and covariance, must be normally distributed.

These assumptions limit the applications of Kalman filters because most real world is much more complicated. Most mobile robots execute nonlinear motions, and this poses a problem for Kalman Filter.
The variant Extended Kalman Filter (EKF) addresses one limitation by linearizing general non-linear motion or measurement with multiple dimension using multi-dimensional Taylor Series. But it's computationally heavy.

2.2 Particle Filters

Particle filters use particles to represent the state of the robot. Particle filters operate by uniformly distributing particles throughout the entire state space and then removing those particles that least likely represent the current state of the robot. The particle filter uses Monte Carlo simulation on an even distribution of particles to determine the most likely position value. Monte Carlo Localization is not subject to the limiting assumptions of Kalman Filters. The basic MCL algorithm are listed as follows:

- Particles are drawn randomly and uniformly over the entire environment
- Measurements are taken and an importance weight is assigned to each particle
- Motion is effected and a new particle set with uniform weights and the particles are re-sampled
- Measurement assigns non-uniform weights to each particle
- Motion is effected and a new re-sampling step is about to start

In this project, Adaptive Monte Carlo Localization is used. AMCL dynamically adjusts the number of particles over a period of time, as the robot navigates the map.

TABLE 1
Comparison of EKF and MCL

	MCL	EKF
Measurement	Landmark	Raw Measurement
Noise	Gaussian	Any
Posterior	Gaussian	Any
Memory	More efficient	-
Speed	Faster	-
Ease of Implementation	-	Easier
Resolution	Better	-
Robustness	-	more robust
Global Localization	No	Yes

2.3 Comparison / Contrast

EKF is great for position tracking problems and work well if the position uncertainty is small. If the EKF is implemented for the right system , it is more time and memory efficient compared with MCL algorithm. However, EKF performs really poorly in general global localization problems. Global localization is where the MCL is more robust than the EKF. By also adding random particles in the particle set, it also solves the kidnapped robot problem. The accuracy and computational costs for the MCL can be tweaked through setting the size of particles. EKF, on the other hand, has great accuracy but heavy in resources due to the matrix computations in the algorithm. The table 1 summarizes the comparison of the two implementations [1].

3 Results

In this project, two robot models are built and simulated for navigation. Both robots reached the goal. For both cases, robots particles were uniformly spread at the start of the simulation and eventually narrowed down to a small group of particles as the robots proceeded to the goal. At the goal, the particles had a tight group pointing towards the orientation of the goal.

Fig. 1. Classroom Benchmark Robot navigation result state.

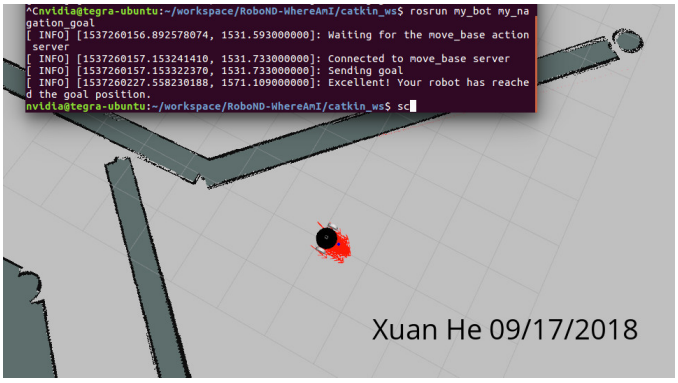

Fig. 2. Student Robot navigation result state.

It took 73 seconds for classroom bot to finish and 78 seconds for student bot while the parameters varied between them. The tuning of parameters is covered in the next section. To notice, classroom bot has narrow turn at the lane exit while student bot is turning more smoothly. While at start, it took longer for student bot to be localized, yet classroom bot was localized quickly.

4 Model Configurations

In a simulation environment such as Gazebo and Rviz, the algorithms are put in use for two robot models: (1) *classroom udacity_bot*, (2) *Student my_bot*.

Model design

The robot model is designed using URDF files which defines the shape of the robot. Classroom bot has a square chassis with two wheels and one caster. Thus, control base is a standard differential drive. Standard package of sensors was chosen.

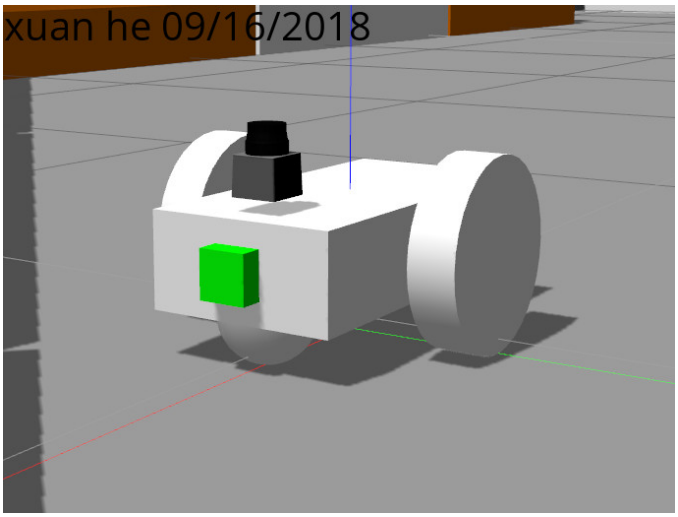

Fig. 3. Classroom Robot.

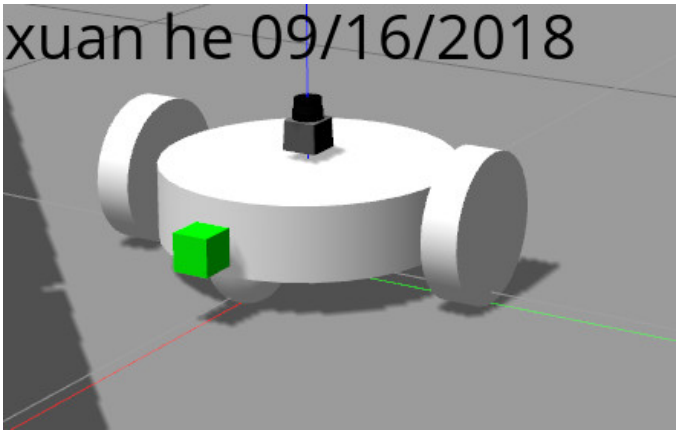

Fig. 4. Student Robot.

- lase scanner (Hokuyou), placed on the had in order to have a free space in 180 degrees in front of a robot.
- camera, placed in front to see what's ahead.

Student bot has a round chassis with two wheels and one caster. The laser scanner is placed at the center of the chassis, so when the bot turns, it is still at the center. Camera is also placed in the front.

Packages Used The project uses several packages, and each package may utilize more package. The main component used is listed below.

- amcl (Adaptive Monte Carlo Localization)
- move_base
- map_server

4.1 Classroom Udacity Robot

4.1.1 AMCL Parameters

The parameter choices used for robots were based on the ROS Navigation Tuning Guide. The initial pose parameters for x, y and yaw were all set to 0. Adaptive Monte Carlo localization algorithm uses variable number of particles depending on certainty which helps free computational resources. Since the given task is relatively simple, from experiments, the minimum number of particles as 100 works well and maximum number of particles is set to 1000. If the given task is very complicated, it is necessary to increase the parameters to compensate.
With the $odom\_model\_type = diff - corrected$ standard values for noise in odometry rotation and translation estimates is not working well and should be decreased. The parameters tuned are the odometry alpha parameters 1 to 4 as defined in ROS. These parameters specify the noise level in the $sample\_motion\_model\_odometryalgorithm$ [1]. Odometry alphas 1 to 4 are the only parameters tuned since the robot uses a differential drive system. All four alpha parameters are set to 0.002 to produce a tight bound on the location. Keeping decreasing the value may produce a tighter bound occasionally but less robust to noisy environment. Hence, 0.002 is chosen in the tradeoff of robustness and accuracy.
In addition, the longevity of transform (transform_tolerance) was set up to 0.3 seconds which was derived experimentally from the time of processing on test machine. This parameter helps select the durability of the transforms(s) being published for localization reasons and should be account for any lag in the system being used. Following the tuning guide, final value of the transform tolerance is 0.3 to meet the system needs.

1) initial_pos_x, initial_pos_y, initial_pos_a = [0, 0, 0]
2) min_particles, max_particles = [100, 1000]
3) odom_alpha_1 = 0.002
4) odom_alpha_2 = 0.002
5) odom_alpha_3 = 0.002
6) odom_alpha_4 = 0.002
7) transform_tolerance = 0.3

4.2 Move Base and Cost Map Configuration

Move base configuration includes the common parameters, local and global costmap parameters. For common parameters, values were selected that works well for two robots. For example, $footprint$ of custom robot is a bit smaller than benchmark robot but a bit higher values work great for both models. $transform\_tolerance$ was set up to 0.3 in accordance to the AMCL configuration. $robot\_radius$ was set to 0.25 to be closer to the robot shape. Furthermore, the update and publish frequency and the map size for both local and global costmaps were reduced drastically to release computational resources. The resolution was also increased from its default 0.05 to 0.1, this was good enough value to reduce computational load while keeping the preferred accuracy.

To avoid hitting the barriers in the simulation, the i$nflation_radius$ was set to 0.50 in costmap common params. This ensures that the robot will keep a distance of half a meter away from the barrier to prevent the robot colliding with the barrier.

1) transform_tolerance = 0.3
2) robot_radius = 0.25
3) inflation_radius = 0.5
4) update_frequency = 20.0 Hz
5) publish_frequency = 5.0 Hz

6) width = 5.0
7) height = 5.0
8) resolution =0.1

4.3 Student Bot

4.3.1 AMCL Parameters

The parameters for student bot are based on classroom bot. However, motion dynamics changed as the shape of the bot differed. The bot typically got stuck when it tried to turn at the lane exit. The odometry alpha parameters were increased from 0.002 to 0.01. With extra noise, the particles have wider spread but the bot doesn't get stuck easily. Additionally, the max number of particles were decreased to 300 to reduce computational cost.

1) initial_pos_x, initial_pos_y, initial_pos_a = [0, 0, 0]
2) min_particles, max_particles = [100, 300]
3) odom_alpha_1 = 0.01
4) odom_alpha_2 = 0.01
5) odom_alpha_3 = 0.01
6) odom_alpha_4 = 0.01
7) transform_tolerance = 0.3

4.3.2 Move base and costmap Parameters

The move base and costmap parameters are based on classroom bot configurations with some tweaking. First, robot radius and inflation radius changes due to the different shape of the bots. Robot radius is 0.3 nand inflation radius is 0.6 respectively to avoid getting into walls. Additionally, the parameters of TrajectoryPlannerROS for base_local_planner are tuned. From experiments, these parameters don't have much influence on the navigation behaviors of the bots since the environment is simple. By changing max velocity to 1.0 from 0.5, the bot can reach a higher speed and thus finish the task quicker. The parameters that are changed from default are listed below. Additionally, the pdist parameter in base local planner was also set to 0.5 so that the robot will follow the global path, but it will give the robot enough freedom to recover from any mistakes.

1) transform_tolerance = 0.3
2) robot_radius = 0.3
3) inflation_radius = 0.6
4) update_frequency = 20.0 Hz
5) publish_frequency = 10.0 Hz
6) width = 5.0
7) height = 5.0
8) resolution =0.1
9) meter_scoring true
10) pdist_scale: 0.5
11) gdist_scale: 1.0
12) max_vel_x: 1.0
13) min_vel_x: 0.1
14) max_vel_theta: 2.0
15) acc_lim_theta: 5.0
16) acc_lim_x: 5.0
17) acc_lim_y: 5.0
18) controller_frequency: 15.0

5 Discussion

For AMCL configuration, *odom_alpha* params was crucial in order to achieve a Gaussian estimation of the robot pose, without them the deviation is too high. As for the *move_base* parameter configuration, it is much more difficult because parameters are scattered among different sub-packages. Fortunately, the navigation task is simple and with only a few parameters tuned, the bot can finish the task. The rest of the tuning is find optimum between the trade off of the localization accuracy and performance speed. For example, by increasing resolution parameter in costmap, information is lost but with a gain of faster computation. Similarly, decreasing the number of particles can increase each cycle of the AMCL computation. But if the robots are very large or move very fast, more particles are needed to be more representative. Tuning parameters can be very daunting when there are so many parameters are laid out across packages. The strategy is nothing but to tune one node at a time instead of simultaneously changing several parameters. To compare the result of classroom robot and student robot, the models are similar with chassis shape changes and size changes. The performance in terms of time is similar too: classroom bot finishes at 73 seconds and student bot finishes at 78 seconds. In this project, the AMCL did not do well solving the kidnapped robot problem. When Gazebo resets the robot position to initial state. The AMCL algorithm was not able to locate the robot. This is due to the small values of odom alphas. There is not enough deviation for localized group of particles to be spread out again.
For real world applications, the MCL algorithm will be well suited in industries such as distribution centers or warehouses where there are not a lot of randomness where number of particles can be controlled and particle filter can be powerful. These have flat surfaces and are easily mapped to robot to use for navigation and localization purposes. The shelves also act as barriers to help the robot navigate through the given path.

6 Conclusion / Future work

Two robots designed with URDF in this project successfully completed navigation task with ROS navigation packages after parameters are tuned. The AMCL algorithm was used where particles represented robot states. The dynamics of the group of particles were updated step by step and resampled for localizing the robots. The accuracy went up as the robot was moving and taking measurements and after a while the group of particles closely represented the pose of the robots. In this project, it was observed that accuracy and processing time have an inverse relationship. Two robots have similar shapes but one small changes could induce a lot of parameters retuned.

Possible improvement would be to experiment more with parameters in TrajectoryPlannerROS, for example, increasing the velocity parameters of the robots for speed-up. If the robot is implemented in real hardware, the measurement noise level will be higher and the motion model has to be more robust for it to work. More particles are needed to localize the robot which requires more computational resources like a powerful CPU. Additionally, parameters

needed to be tuned after hardware is complete since in real environment, friction and obstacles needs to be taken in account.

REFERENCES

[1] S. Thrun, W. Burgard, D. Fox, and R. Arkin, *Probabilistic Robotics*. Intelligent robotics and autonomous agents, MIT Press, 2005.