Udacity RoboND Term-2 Where Am I Project Report

Emre Özdemir

Abstract—In this project, localization and navigation performance of two different differential drive robot is investigated. First, theoretical background of Extended Kalman Filter and Adaptive Monte Carlo Localization is introduced. Then, difference between Particle Filters and Kalman Filters are summarized. In the simulation section, hyperparameter space for AMCL and move_base package is introduced and tuning strategy is explained. Then, simulation results of both models are given and performance is compared. According to the results personal robot ”ozdemre_bot” performed better in terms of navigation. As a last step, hardware implementation and future studies are given.

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

1 Introduction

ROBOT is any sort of machine that can perceive it's environment, make a decision and act. Being able to answer the question ”Where am I?” is part of a perception problem, where robot needs to figure out where it is in the world and what should be it's action. Many methodologies are exist for Robot localization, such as Extended Kalman Filter, Monte Carlo Localization, Markov Localization, Grid Localization and so on. In this project, first theoretical background of Kalman and Particle filters are introduced. Then, Adaptive Monte Carlo Localization (AMCL) algorithm is tested on two custom built differential drive robots (udacity_bot and ozdemre_bot) for performance evaluation in ROS-Gazebo environment. Lastly, pros and cons for localization algorithms and conclusion is given.

Fig. 1. Question of the project.

2 Background

Localization problem tries to determine a robot's pose in a known environment using sensor measurements. Main challenges of this process usually seen in real world applications where noisy sensor measurements exist (encoder noise due to wheel slip, non-reflective surfaces that disrupt Lidar measurements ..etc). Localization algorithms such as Ektended Kalman Filter (EKF) and Adaptive Monte Carlo Localization (AMCL) aims to overcome this problem and accurately determine the robot's pose in a ”known environment” or map. For this study AMCL method is chosen since it is more robust compared to EKF method. AMCL can be tuned depending on environment and robot itself for better memory usage and performance. And lastly Kalman Filter is capable of only dealing with Gaussian noise however, AMCL uses particle filer which enables it to deal with any sort of measurement noise.

2.1 Kalman Filters

In very general terms, the Kalman filter estimates the value of a variable by updating its estimate as measurement data is collected while filtering out the noise. Kalman Filter models the state uncertainty using Gaussians and it is capable of making accurate estimates with just a few data points which makes it very efficient and fast way of reducing uncertainity.

For the scope of localization problem, 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.

Kalman Filter assumes that motion and measurement models are linear and that the state space can be represented by a unimodal Gaussian distribution. Most mobile robots will execute non linear motion like following a curve. Non linear actions will result in non-Gaussian posterior distributions that cannot be properly modeled by a closed form equation.

This is where EKF comes into play. 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.

In addition to localization problem, Kalman Filer can be used filter out the noise coming from data itself which makes it a very useful tool for any sort of statistical calculation.

2.2 Particle Filters

Monte Carlo localization algorithm, similar to Kalman Filter method, estimates the posterior distribution of a robot's 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. This is intoduced as "Resampling Wheel" in the classroom material.

"Adaptive" Monte Carlo Localization is just one step further from MCL. As the name suggest AMCL provides tuning option for paremeters such as, maximum minimum number of particles. This is important tuning parameter for obtaining good memory allocation/localization performance trade off.

2.3 Comparison / Contrast

As explained previous sections there are number of advantages and disadvantages while choosing Kalman or Particle filter. These are summarized in the able given below. It should be noted that none of the algorithms introduced so far has ability to solve "kidnapped robot" problem

TABLE 1
Table

Property	EKF	AMCL
Measurement Noise	Gaussian Distribution	Any
Memory Efficiency	Great	Requires Tuning
Time Efficiency	Great	Requires Tuning
Resolution	Great	Requires Tuning
Global Localization	Not Possible	Possible
Robustness	Average	Great
IMU integration	Possible	Not Possible

3 Simulations

Simulations are performed for two different mobile robots called "udacity_bot" and "ozdemre_bot". Environment is chosen as famously known jackal_race.world. Technical specifications of robots are given in the following sections. AMCL package is used for localization and move_base package is used for navigation. List of AMCL and move_base parameters, their explanations and tuning strategies are given below. Parameter values given below are used for benchmark model "udacity_bot". Even though most of the parameters same for both robots, some of them are changed for personal model "ozdemre_bot". Detailed explanation about differences given in the section 3.3.3.

AMCL parameters are defined in the amcl.launch file in the ROS package

- **min_particles and max_particles: 10, 50**: Maximum an minimum allowed number of particles. For the purpose of better localization this can be increased however, computational cost will be higher. From my experiments I realized that this value provides sufficient performance while computational efficient.
- **update_min_a and update_min_d: 0.005, 0.01**: Translational and rotational movement required before performing a filter update. For slow moving robots it is good idea to lower these value.
- **laser_max_beams: 20**: How many evenly-spaced beams in each scan to be used when updating the filter
- **laser_z_rand: 0.05**: Mixture weight for the z_rand part of the model. Left it as default value.
- **laser_z_hit: 0.95**: Mixture weight for the z_hit part of the model. Left it as default value.
- **laser_model_type: likelihood_field**: Which model to use, either beam, likelihood_field, or likelihood_field_prob. The likelihood_field model is usually more computationally efficient and reliable for this environment.
- **odom_alpha*: 0.02**: Specifies the expected noise in odometry's translation and rotation (1 to 5). As simulation environment does not produce any noise for the odometry, we will set it to a small value o inform that we do not expect any noise.

Move_base parameters are defined in the .yaml files in config folder.

- **obstacle_range: 4.0**: For example, if set to 0.1, that implies that if the obstacle detected by a laser sensor is within 0.1 meters from the base of the robot, that obstacle will be added to the costmap. Tuning this parameter can help with discarding noise, falsely detecting obstacles, and even with computational costs.
- **raytrace_range: 4.0**: This parameter is used to clear and update the free space in the costmap as the robot moves.
- **transform_tolerance: 0.2**: Specifies the delay in transform (tf) data that is tolerable in seconds. Trial and error method is used to ensure no warnings on the terminal.
- **inflation_radius: 0.6**: This parameter determines the minimum distance between the robot geometry and the obstacles.An appropriate value for this parameter can ensure that the robot smoothly navigates through the map, without bumping into the walls and getting stuck, and can even pass through any narrow pathways.
- **pdist_scale: 1.0**: This is the weight for how much the local planner should stay close to the global path. A high value will make the local planner prefer trajectories on global path.
- **gdist_scale: 0.4**: This is the weight for how much the robot should attempt to reach the local goal, with

whatever path. Experiments show that increasing this parameter enables the robot to be less attached to the global path. I lowered he default value a bit to make it more attached to the global plan.

- **occdist_scale: 0.01**: This is the weight for how much the robot should attempt to avoid obstacles. A high value for this parameter results in indecisive robot that stuck in place.
- **max_vel_x and min_vel_x: 0.5, -0.5** Used for limiting robots linear velocity. I tried to use lower values such that robot does not perform high speed manoeuvres that lowers the localization performance.
- **max_vel_theta and min_vel_theta: 0.5, -0.5** Used for limiting robots angular velocity. I used lower values to avoid quick in place rotations (for example at the goal location) that lowers localization performance.
- **update_frequency: 10.0**: Higher is better however, trial and error method is used to ensure no warnings on the terminal.
- **publish_frequency: 5.0**: Higher is better however, trial and error method is used to ensure no warnings on the terminal.
- **controller_frequency: 5.0**: Higher is better however, trial and error method is used to ensure no warnings on the terminal.
- **xy_goal_tolerance: 0.05**: Used for checking whether or not robot is at the goal place. Not necessary for this application since provided script already checks it. But I added anyway.
- **yaw_goal_tolerance: 0.01**:Used for checking whether or not robot is at the goal place. Not necessary for this application since provided script already checks it. But I added anyway.
- **publish_cost_grid_pc: true**: I enabled this to observe the cost grip map in the Rviz as point cloud.

3.1 Achievements

With the aforementioned parameters both mobile robots managed to reach the published destination by navigation_goal node. In the following sections, details of the robots and results are given.

3.2 Benchmark Model

3.2.1 *Model design*

Udacity_bot model parameters are based on classroom material. Chassis size is 0.4x0.2x0.1 meters (length x width x height). Model is supported with 2 caster wheels modelled as spherical geometries with no friction. Excitation is provided by two wheels on the both sides with 0.1m radius and 0.05m length. Robot excitation is differential drive where steering is provided by angular velocity difference of excitation wheels. Robot is equipped with Hokuyo LiDAR and depth camera at the front. Mobile robot physical parameters (size, mass, friction ..etc) are defined in the udacity_bot.xacro file. Gazebo interaction, such as differential drive and sensor plugins are defined in the udacity_bot.gazebo file.

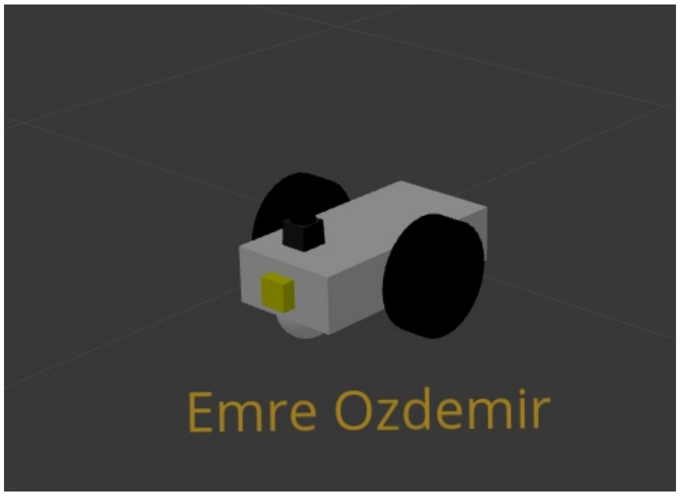

Fig. 2. Udacity Bot.

3.2.2 *Packages Used*

For driving the robot two main packages are used.

- **udacity_bot:**Package that contains, robot desctiption, AMCL package, Move_base package, navigation_goal script and necessary launch files. In order to launch the environment following launch files must be run respectively:
  1) roslaunch udacity_bot udacity_world.launch
  2) roslaunch udacity_bot amcl.launch
  3) rosrun udacity_bot navigation_goal_u
- **turtlebot_teleop:** Keyboard tele-operation package that is used for testing the model manually in simulation environment. Downloaded from turtlebot3 ROS-Github repo and used directly.

3.2.3 *Parameters*

For udacity_bot aforementioned (in the beginning of section 3) values are used with given parameters for AMCL and move_base.

3.3 Personal Model

3.3.1 *Model design*

At the beginning first attempt was building a simple differential drive robot model with 2 caster wheel at the front similar to Trailer type AGV's for personal model. This model includes a chassis two excitation wheels at the back and 2 caster wheels with rotation enabled on both z and y axis. However, even though it worked well in terms of kinematics, it is seen that unrealistic parameter selection (caster wheel friction, damping, mass ..etc) started to become an issue during dynamic simulation. Therefore a simpler model is preferred as given below, but AGV model is still in the urdf folder of the repo for additional studies. Second attempt was building a model that looks a bit similar to simple autonomous car trucks. It has a similar chassis but wider wheelbase (0.4x0.2x0.1 meters chassis, 0.3 m wheelbase) compared to "udacity bot" and has a spoiler at the back for better traction. Two caster wheels for stabilizing the car is located at the front and back side of the chassis with spherical geometries. Wheel diameters are left as it is to provide sufficient traction force to move and rotate the body. Camera and Lidar is placed in front of the truck. Differential drive plugin is also updated with new wheelbase value.

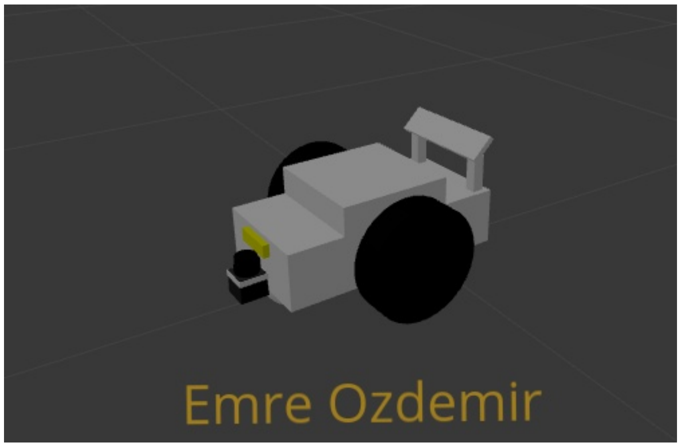

Fig. 3. ozdemre_Bot.

3.3.2 *Packages Used*

For driving he robot two main packages exist.

- **ozdemre_bot:**Package that contains, robot desctiption, AMCL package, Move_base package, navigation_goal script and necessary launch files. In order to launch the environment following launch files must be run respectively:
    1) roslaunch ozdemre_bot udacity_world.launch
    2) roslaunch ozdemre_bot amcl.launch
    3) rosrun ozdemre_bot navigation_goal
- **turtlebot_teleop:** Keyboard tele-operation package that is used for testing the model manually in simulation environment. Downloaded from turtlebot3 ROS-Github repo and used directly.

3.3.3 *Parameters*

For "ozdemre bot" same AMCL parameters have been used with the benchmark model. However for the local and global planner parameter for move base has been changed. Maximum an minimum linear and angular velocity constraints are added into config file to avoid quick rotations which has a bad effect on localization performance. Following parameters are also changed for better navigation performance. Rest of the parameters are left as it is since overall performance was satisfactory.

- **pdist_scale: 1.4**: Previous value is increased for fine tuning trajectory selection. This has en effect of robot preferring global trajectory rather than local one, which decreases unnecessary oscillation and creates smoother path.
- **gdist_scale: 0.4**: Previous value is decreased for he same reason above.
- **occdist_scale: 0.05**: For some cases, robot sddenly stopped around first corner. Increasing this value increases maximum obstacle cost along the trajectory, which eliminated the problem of robot suddenly stopping.

4 Results

Present an unbiased view of your robot's performance and justify your stance with facts. Do the localization results look reasonable? What is the duration for the particle filters to converge? How long does it take for the robot to reach the goal? Does it follow a smooth path to the goal? Does it have unexpected behavior in the process?
For demonstrating your results, it is incredibly useful to have some watermarked charts, tables, and/or graphs for the reader to review. This makes ingesting the information quicker and easier.

Overall both robots achieved the goal around 70 seconds of travel time without getting stuck. AMCL worked well by shrinking the pose array within 1-2 seconds. Parameters like inflation_radius and obstacle_range has a significant impact on building the costmaps. Badly tuned values might cause robot to stuck near the walls or sharp turns. Here are the final pictures of two robots reaching their destination.

4.1 Localization Results

4.1.1 *Benchmark*

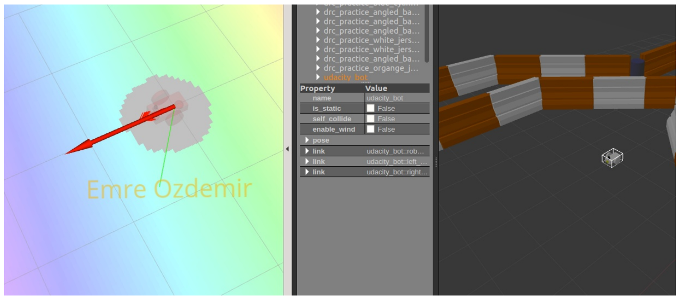

Fig. 4. Move Base Target Reached in 3 minutes for Udacity_bot.

4.1.2 *Student*

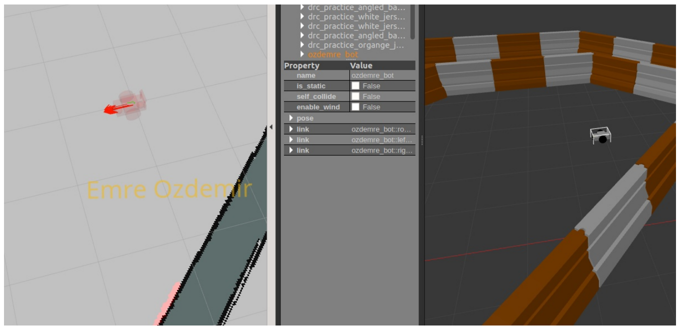

Fig. 5. Move Base Target Reached in 72 seconds for ozdemre_bot.

4.2 Technical Comparison

In overall, personal robot "ozdemre_bot" had a better performance in terms of navigation. This is simply due to spending more time on tuning the local and global planner parameters. Benchmark robot get confused around first corner where it should make a smooth turn by following global path. Instead robot rotated in place couple of times to catch the local plan which caused robot reaching it's target around 3 minutes even though linear speeds are similar on straights for both robot. Localization performance for both robots are the same as Pose Arrays are shrunk quickly.

5 Discussion

According to the simulation results personal design robot performance was slightly better. This is due to more tuning in the hyper parameter space. One another possible cause might be reference robot simulation model. In the classroom material, differential drive plugin for udacity_bot is generated with "wheel separation" parameter set to 0.4. However, from the robot description file and double checking with Rviz, this value should be 0.3. This might be considered as possible disturbance on the system control which causes worse performance for udacity_bot.

5.1 Topics

Overall, following conclusions can be derived:

- Personal Robot has a better performance in terms of navigation. For the localization no major change is observed.
- Reason why ozdemre_bot has a better performance is simple spending more time on parameter tuning.
- 'Kidnapped Robot' is not something that can be solved by EKF or AMCL.
- AMCL can be used for any sort of localization task (AGV, delivery robot, service robot) with a know environment (map).
- Autonomous Grounded Vehicles (AGV) is an area I am currently working where localization is performed in a known environment (production line). This is a project that I get a chance to apply what I have learned from this project.

6 Conclusion / Future work

This section is intended to summarize your report. Your summary should include a recap of the results, did this project achieve what you attempted, how would you deploy it on hardware and how could this project be applied to commercial products? For Future Work, address areas of work that you may not have addressed in your report as possible next steps. This could be due to time constraints, lack of currently developed methods / technology, and areas of application outside of your current implementation. Again, avoid the use of the first-person.

In this project, localization and navigation performance of two different differential drive robot is investigated. Firs theoretical background for AMCL and EKF is investigated. Then hyper-parameter space for AMCL and move_base package is introduced. It is seen that ROS AMCL and move_base package has good performance and extensive capability for parameter tuning. For the benchmark robot mostly default values for the parameters are used. As a result benchmark robot manage to reach it's goal without getting stuck. Secondly, personal robot is investigated with spending more time on tuning. By changing distance goal and obstacle bias values cost function is optimized for performing smoother trajectory without getting stuck. As a result, personal robot performed better in Clearpath Robotics jackal_race simulation environment.

Apart from simulation world, this project has an extensive application area in the real world. AGVs, UGVs delivery robots has a similar operation conditions that has been investigated here. SoOme of the mobile robots are based on AMCL and move_base localization, however, some uses only AMCL to estimate it's pose in a known map and perform path tracking algorithms (pure pursuit, stanley ..etc) to navigate around. Real world applications definitely require more tuning due to nature of sensor noise and potentially require EKF implementation with IMU sensors.

6.1 Modifications for Improvement

Examples:

- Sensor Amount: 2 LiDAR covering 360 degree of view can potentially improve localization. Most Lidar's do not have 360 degree capability, however using 2 of them (front and back with 180deg PoV each, or opposite corners with 270deg PoV each) with laserscan_multimerger ROS package would increase localization performance.

6.2 Hardware Deployment

1) In a real world implementation, wheel odometry and Lidar measurements are noisy. This requires more tuning or potentially usage of ekf (with/without IMU) to avoid real world implementation problems.
2) Since AMCL and move_base is computationally intense processes, this requires a PC with a sufficient RAM and CPU capability. For industrial applications, Industrial PC's with sufficient RAM (¿8GB) or Nvidia Jetson family is a convenient choice.

References