Where Am I Project Writeup

Marko Sarkanj

Abstract—The focus of this project is localizing the robot in known environment and reaching the desired location. This has been achieved with the help of ROS, Gazebo simulator, RViz and Monte Carlo Localization algorithm (MCL). The simple robot has been created with the help of Xacro file and imported into Gazebo simulator. The robot has odometry sensor, camera and Hokuyo(laser) sensor. Only odometry and Hokuyo sensors have been used to perform MCL. The robot was able to localize itself in known environment. Path planning performs well but not perfect. Robot can easily reach the coordinates assigned by Udacity, but there are some edge cases where robot is not able to reach the desired location on the map.

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

1 Introduction

As part of the Udacity's Robotics Engineer Nanodegree program one of the assignments was to locate the robot in the computer simulated environment with the help of Monte Carlo Localization. Monte Carlo Localization focuses on the very precise localization of already known (mapped) environment by detecting landmarks and finding the most likely current position of the robot. This kind of localization is important because it can provide very accurate robot positions and orientations, in contrast to GPS localization, where most of the devices have measurement error around 10 meters.

2 Background

To perform the localization task two approaches have been considered, Kalman Filters and Particle Filters. The comparison between them has been performed in the next three subsections.

2.1 Kalman Filters

The Kalman Filter is an estimation algorithm that is very prominent in controls. It is able to estimate the value of a variable in a real time and it is very computationally efficient. It can be used not only for estimating position but for the estimation of any other sensor reading like temperature or velocity. The algorithm can be used for sensor fusion as well. Kalman filter works by predicting value and by updating the prediction based on a sensor reading. If variance of sensor readings is high, there is more weight added to prediction. From the other side if the prediction is unreliable, sensor reading will affect the predicted value more. This is regulated by Kalman gain that is recalculated on every iteration. The main disadvantage of Kalman filter is that it assumes linear motion and measurement models with state space represented by an unimodal Gaussian distribution. Those preconditions are rarely fulfilled in practice and this is why the Extended Kalman filter is predominately used. Extended Kalman filter performs co-variance update by performing linear approximation of function f(x) with the help of Taylor series. This enables Kalman filter to perform predictions based on non-linear motion and measurement models.

2.2 Particle Filters (MCL)

Particle filter is localization technique that uses some number of particles that are spread across the map. On 2D map each particle has its own x position, y position, orientation and weight. Based on some robot measurement, for example laser, particles are compared to robot's current position on each iteration. If the position of the particle on the map corresponds to the robot's current position/detected landmarks, particle gets higher weight. Particles with higher weights are more likely to be chosen for the next iteration, which results in all particles merging near the robot's current position.

2.3 Comparison / Contrast

The main advantage of the Kalman filter is the computing and memory efficiency. From the other side particle filter is easier to implement, more robust and it can work with any type of measurement noise without need for workarounds or transformations. As previously described, Kalman filter can work only with measurement noise presented as a Gaussian distribution. Another advantage of particle filter over Kalman filter is that it can localize globally on the map, not just locally. Because of the already stated reasons, particle filter has been chosen for this project.

3 Simulations

In this section it will be described how the parameters were tuned and how two robots behaved in the Gazebo simulator. Two robots were used in this project, first is the Udacity's benchmark robot and the second is the custom robot that is modified version of the Udacity's benchmark robot.

3.1 Benchmark Model

3.1.1 Model design

The benchmark model consists of two actuators(wheels), two casters, chassis, camera and Hokuyo sensor(laser sensor). Chassis has the shape of a box with the Hokuyo sensor on the top and the camera on the front side.

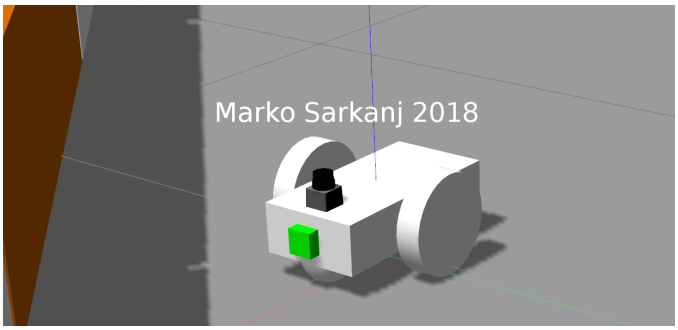

Fig. 1. Design of the Udacity's benchmark robot

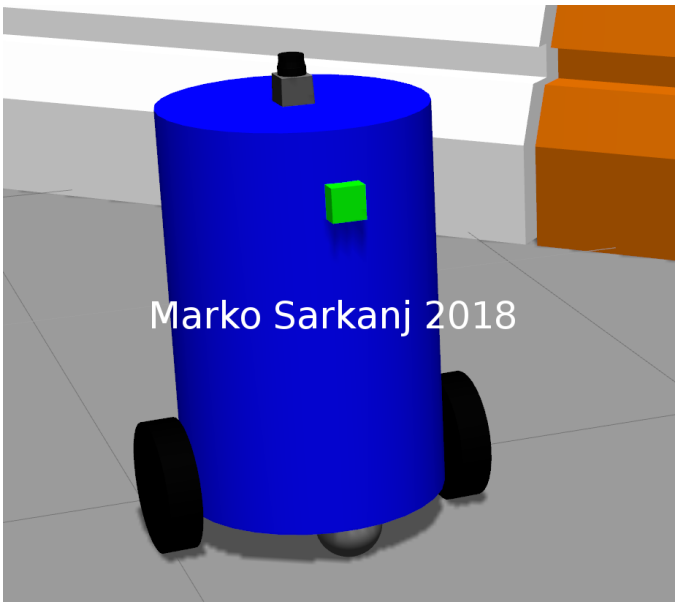

Fig. 2. Design of the custom robot

3.1.2 Packages Used

Packages used for the robot simulation were differential drive controller, camera and Hokuyo controller. Differential drive controller was used to simulate the robot's movement by turning left and right wheels. Camera was used to take pictures of the environment and Hokuyo sensor was used for the laser scan of the environment. Hokuyo controller publishes scan topic, camera image_raw and camera_info topics. Differential drive controller publishes odom topic and receives cmd_vel topic.

Package used for the localization was AMCL - Adaptive Monte Carlo Localization package. This package receives the scan topic from Hokuyo controller, the odom topic from differential drive controller and the map topic from the map server.

Move_base package was used for path planning and robot movement. It receives an estimation from the AMCL package on a robot's position and combines it with the map topic to perform global and local planning. Then it publishes cmd_vel topic to differential drive controller according to a planned path.

3.1.3 Parameters

Move_base package is configured with four Yaml files: costmap_common_params.yaml, local_costmap_params.yaml, global_costmap_params.yaml and base_local_planner.yaml.

In costmap_common_params.yaml file parameters that configure global and local costmaps have been defined. Those parameters are:

- obstacle_range - maximum range in meters at which obstacles are added to costmap.
- raytrace_range - maximum range in meters at witch raytrace detects obstacles.
- transform_tolerance - sets maximum allowed latency of transformations between different coordinate frames.
- footprint - defines a robot's dimensions
- inflation_radius - defines how close a robot can approach obstacles without incurring costs on a cost map.

The obstacle range was set to 5 meters and the raytrace range to 6 meters. Those values were determined by experimentation and they have shown sufficient for this robot as it moves at slow speeds. Transform tolerance of 0.5 seconds has been set after the robot failed to move with the transform tolerance set to 0.0. Footprint has been set based on the robot's dimensions to [[-0.1, 0.2], [0.1, 0.2], [0.1, -0.2], [-0.1, -0.2]]. Inflation radius has been set to 2 meters as this value prevented the robot from hitting into walls with the optimal planning still possible.

local_costmap_params.yaml and global_costmap_params.yaml files contain the same parameter types that configure local and global path planning respectively:

- global_frame - frame in which robot will operate. Odom (Odometry) is for a local costmap and map for a global costmap.
- robot_base_frame - base frame of a robot (footprint).
- update_frequency - frequency how frequently planner updates.
- publish_frequency - frequency how frequent planner publishes new path.
- with, height, resolution - with, height and resolution of a costmap.
- static_map - true for a global costmap where the a doesn't change, false for a local costmap.
- rolling_window - whether or not to use rolling window version of a costmap. This parameter is always false in the case of static map. In the case of a local costmap this parameter has been set to true.

For the local planning odom(odometry) has been defined as global_frame, as the local planner plans trajectory from the robot's perspective. For the global planning map has been defined as global_frame. Update and publish frequencies have been set to 10 Hz for both planners. This value was determined by experimentation as it performed best on the testing configuration. The local planner has with and height of the map set to 20x20. The global planner has with and height of the map set to 100x100(the whole map area).

base_local_planner.yaml file contains following trajectory planner related parameter types:

- max_vel_x - maximal forward velocity of a robot in meters per second.
- min_vel_x - minimal forward velocity of a robot in meters per second.
- max_vel_theta - maximal rotational velocity in radians per second.
- min_in_place_vel_theta - minimal rotational velocity when a robot turns in place in radians per second.

Maximal forward velocity has been set to 0.45 meters per second and minimal to 0.1 meters per second. Maximal rotational velocity has been set to 1 radians per second when robot moves. Minimal rotational velocity of 0.4 radians per second has been set for the robot when it turns in place. This was determined by experimentation, as those velocity parameters performed best given the robot's sensors, actuators and computing capabilities of the testing configuration.

AMCL parameters are configured directly in the amcl.launch launcher file:

- max_particles - number of particles used for Monte Carlo localization.
- odom_alpha15 - expected noise in odometry's rotation and translation.

Number of particles has been set to the maximum number of 5000 as the testing configuration was performant with this configuration as well. Odom alpha 1 to 5 parameters have been set based on the recommendation from the following forum post: https://answers.ros.org/question/227811/tuning-amcls-diff-corrected-and-omni-corrected-odom-models/ .

3.2 Personal Model

3.2.1 Model design

Personal model is modified version of the benchmark model. Chassis has the cylindrical shape, the robot is significantly higher, camera has been placed near top of the custom robot and Hokuyo sensor has been placed in the middle of the robot's top side.

3.2.2 Packages Used and Parameters

Personal model was able to operate based on the same parameters and with the same packages as the benchmark model, as described in the section "Technical Comparison".

4 Results

4.1 Localization Results

Both robots, benchmark and personal model were able to reach the target location set by Udacity.

4.1.1 Benchmark model

Benchmark model is able to locate in 15 seconds. First the robot starts going in the wrong direction but after short period of time it is able to locate and start going into the right direction. After that it locates almost perfectly until the goal has been reached, as shown in figures 3 and 4. Robot has generally smooth path to the goal but it gets stuck for a moment during the U-turn at the end of the left wall.

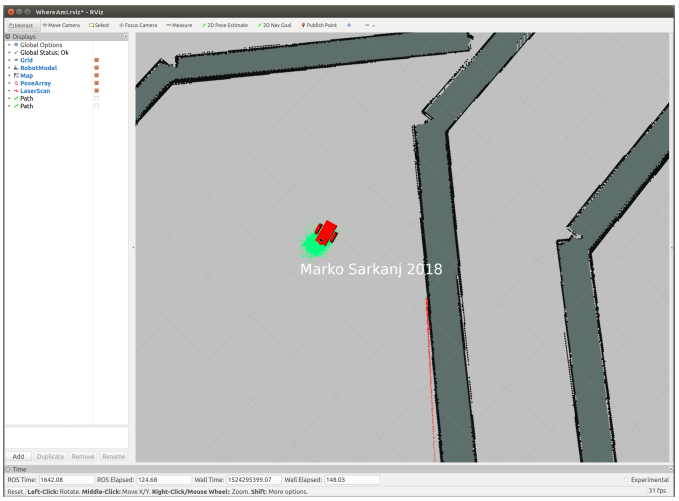

Fig. 3. Benchmark model at the goal position

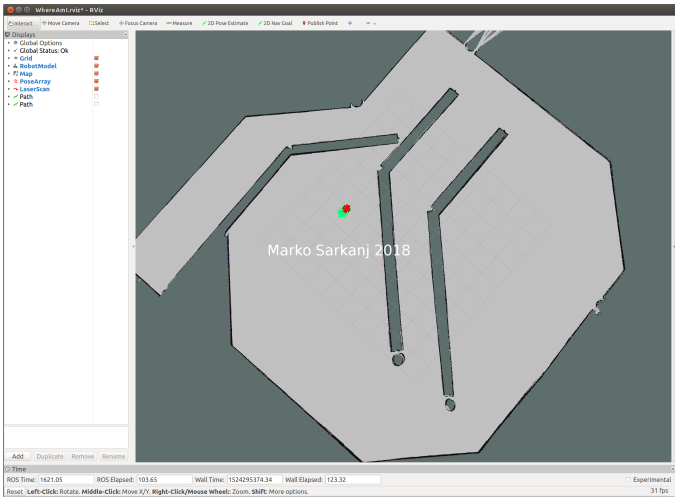

Fig. 4. Benchmark model at the goal position

4.1.2 Personal model

Personal model takes around 20 seconds to locate and starts going into the right direction. It is able to reach the goal but it never locates perfectly as shown in figures 5 and 6. The path taken is almost identical to the benchmark robot, personal robot takes few seconds more to locate and it gets stuck for a moment at the U-turn as well.

4.2 Technical Comparison

The main differences between the benchmark model and the personal model are the chassis shape and the Hokuyo sensor position. Because of the Hokuyo sensor positioned in the middle of the personal model, laser readings are slightly worse. This results in slightly worse Monte Carlo Localization. The custom robot is still able to reach the goal location.

5 Discussion

In this project both robots performed well, they have reached the desired goal. The benchmark robot performed slightly better. The reason for that was the Hokuyo sensor

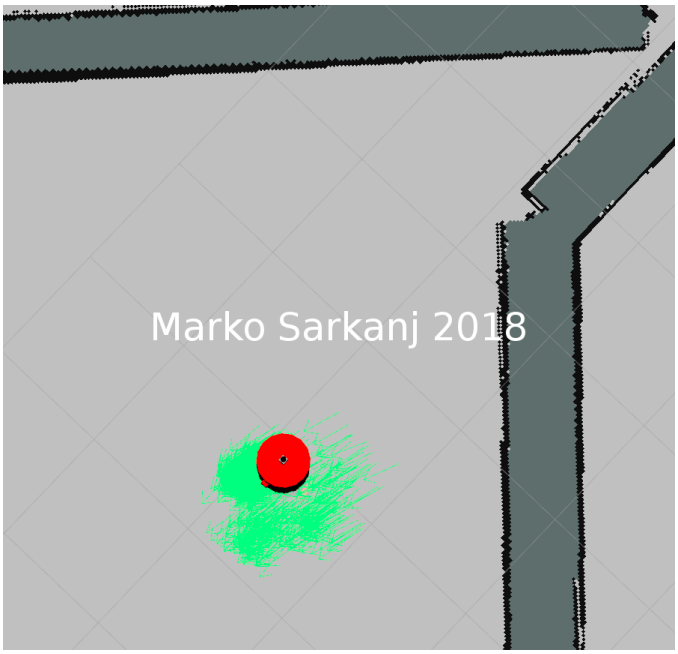

Fig. 5. Custom model at the goal position

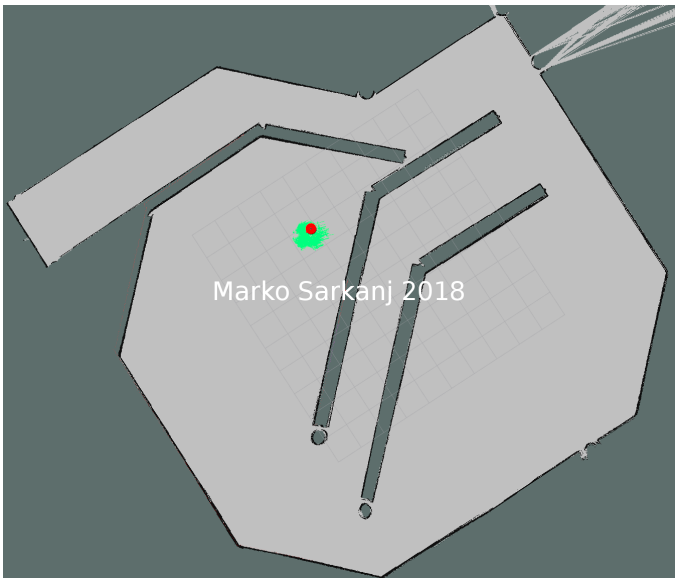

Fig. 6. Custom model at the goal position

located on the front side of the robot. This enabled better object detection and the benchmark model was able to locate better because of that. During the tests some edge cases have been detected, where robots are not able to reach the goal position. This happens because of the too much weight given to the local planner. During experiments where more weight was given to the global planner, robot was not able to perform the initial localization as good as with the current settings. Current settings were kept to optimize the robot's navigation for the task assigned for the project, for some other tasks parameters should be reconfigured. Kidnapped Robot problem would be impossible to solve with the current configuration because the robot is only able to operate in known environment. For the Kidnapped Robot problem some other algorithm should be used, SLAM for example. In an industry domain MCL can be used in known environments with landmarks that stay the same all the time to enable orientation, for example on the factory floor or in the warehouse.

6 Conclusion / Future work

As already stated both robots performed fairly well, they have reached the desired goal. They are still not reliable enough for the real world use because there are still some edge cases where robots get lost, and that needs to be improved for the real world use.

6.1 Modifications for Improvement

Robot could be improved by adding one more additional Hokuyo sensor, so that there is one sensor on both ends of the robot. That would make them more reliable and enable MCL algorithm to work with more information, which should improve localization in turn. Some cost improvements could be made as well, the robot cold use RGBD camera instead of the expensive Hokuyo sensors for the localization as cheaper alternative. On the other side if reliability is essential some even more expensive sensor could be implemented. The range of the current Hokuyo sensors is just 5-6 meters, which can be very limiting on large surfaces. Radar could be added as well to support the localization, as there are some situations where the Hokuyo sensor doesn't perform well, for example under heavy rain or fog.

6.2 Hardware Deployment

This code could be easily deployed on Nvidia Jetson TX 2. New robot should have Hokuyo sensor, actuators and camera. Area where robot would move should be mapped as well. As the whole project was done in ROS, there shouldn't be much issues transferring it into the real world. Topics published by Gazebo should be replaced with topics published by robot's sensors. After some inevitable initial tuning of the parameters robot should be able to operate in the real world. Number of particles used by MCL should probably be lowered to work on the mobile hardware with the battery consumption as low as possible.