Where Am I?

Vladimir Kobal

Abstract—Progress in the field of localization algorithms allows building more useful autonomous assistants. Nowadays more and more mobile robots are used not only in industry but household applications too. As self-moving device development becomes a widespread task, corresponding tools become more mature. The goal of this project is to study the main features of the most popular localization algorithms and to develop a robot model while verifying its functionality in a simulated environment. Thanks to ROS it is not a difficult task.

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

1 Introduction

While mobile robots can be preprogrammed for certain movements in a specially prepared environment it is greatly advantageous when a robot has the possibility to estimate its pose and use this information while performing its task. It vastly increases the number of applications where robots can be used. The problem of finding out the position and orientation of a robot in a known environment (when the map is given) is often called the localization problem. There are a lot of methods for solving the localization problem, but this paper considers Extended Kalman Filter (EKF) and Monte Carlo Localization (MCL) algorithms. The project's goal is to build a simple model of a self-localizing mobile robot using ROS and test it in the Gazebo environment.

For accomplishing the task a few resources were used for additional information. Especially valuable ones were [1], [2], [3] and [4]

2 Background

Although there are many localization methods, EKF and MCL are the most popular. Each of them has its advantages and drawbacks.

2.1 Kalman Filters

Kalman Filters are widely used in different fields of application. They are fast, lightweight and offer highly accurate estimation. In the basis of Kalman Filters lies the assumption that measured data and predicted state are uncertain and this uncertainty is represented by the Gaussian distribution. Two steps are executed consequently in a cycle: measurement update and state prediction. The former determines the new mean as the weighted sum of prior belief (the initial state is given for the first step) and measurement and calculates the new variance as for more confident distribution. The latter is as simple as adding means and variances of updated measurement and state change. For multidimensional Kalman Filters covariance is introduced and linear algebra is used for calculations at every step.

Kalman Filters are limited to manipulate with linear functions. The state can be represented by Gaussian distribution only. It is impossible to solve the global localization problem with Kalman Filters. To cope with one of these drawbacks EKF was designed. It allows using nonlinear motion and measurement functions through linearization by using Taylor series.

2.2 Particle Filters

MCL uses Bayes filtering at the core of the estimation process. Particles represent possible states and at the first step are distributed randomly and uniformly across the mapped space. In the main cycle of the algorithm, particles are shifted towards new predicted state according to the movement of the robot and resampled based on the recursive Bayesian estimation of measured data. At motion and sensor update steps noise has to be taken into account.

MCL can deal with non-linear functions and non-Gaussian noise. It solves the global localization problem, and it is quite easy to implement. Discrete nature of state space imposes a limitation to the resolution of the algorithm.

2.3 Comparison / Contrast

MCL is less computational effective than EKF, but on the other hand, it allows to use raw measurement data and is easy to implement, especially in ROS environment. Achieving the project's goals does not assume high demands for resolution and computational efficiency, that's why MCL is preferred over EKF and will be used for completing the task.

3 Simulations

3.1 Benchmark Model

3.1.1 Model design

The supplied model is symmetric along both axes: longitudinal and lateral. It simplifies usage of differential driver controller. Mass is spread evenly across the front and rear parts of the chassis except for lightweight sensors in the front. A camera is positioned on the frontal plane of the chassis which guarantees an unobstructed view. Total mass of the robot is 25.2 kg, dimensions — 0.45 x 0.35 x 0.25 m.

3.1.2 Packages Used

For localization `amcl` package is used. It utilizes the adaptive Monte Carlo localization algorithm. Measurements are taken from `udacity_bot/laser/scan` topic which is simulated by `gazebo_ros_head_hokuyo_controller` plugin. Navigation is implemented with `move_base` package. Odometry data is published, and driving commands are executed by means of `differential_drive_controller` gazebo plugin.

3.1.3 Parameters

In the `amcl` node parameters a few changes were made (Table 1). `transform_tolerance` was increased to give the algorithm the time to calculate every step without dropping out many iterations. As far as global localization is not performed in the project and the map is relatively small and simple, it turned out that quite a small number of particles did the work. Further increase of particle number did not provide any noticeable improvement. For boosting the precision and the speed of convergence, update thresholds were set to relatively small values. Chosen laser range keeps relevant and cuts off redundant data. Other parameters control the laser scanner precision.

TABLE 1
AMCL parameters

transform_tolerance	0.15
min_particles	50
max_particles	200
update_min_d	0.05
update_min_a	0.1
laser_min_range	0.3
laser_max_range	15.0
laser_max_beams	70
laser_z_hit	0.99
laser_z_rand	0.01

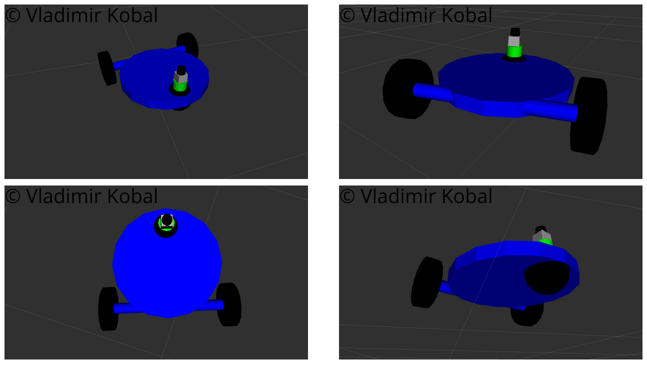
The `move_base` node had problems with the calculation speed as well, so many parameters for controller, update and publish frequencies were substantially decreased. Local map size was shrunk to 2x2 m otherwise the robot's behavior was unpredictable as algorithm periodically preferred local path to the global one for approaching the goal position, which resulted in cycle movements. The most important parameters for bump-free movement are `robot_radius`, `inflation_radius` and `cost_scaling_factor`. They were set to 0.3, 1.0 and 3.0 respectively. The chosen values of the two latter parameters stimulate the robot to move quite far from walls. At the same time as the path cost decline gradually movements are smooth.

3.2 Personal Model

3.2.1 Model design

For the custom model an asymmetric front-rear design was chosen (Table 2). Chassis is based on a flat disk. Rear axle holds 2 driven wheels. Coordinates of the chassis were shifted to keep the axle at x=0. Front caster friction was decreased down to 0.0. The custom model is slightly larger and heavier than the supplied one. Total mass of the robot is 35.2 kg, dimensions — 0.55 x 0.45 x 0.27 m. Applied torque was increased up to 20 Nm. Sensors are stacked up on the front end of the chassis.

TABLE 2
Custom robot



3.2.2 Packages Used

The 4-wheel design was considered at first. Skid Steering Drive Controller plugin could be used in that case. As far as the custom model still uses the differential 2 wheel drive scheme, there was nothing to change in packages, plugins and data flow.

3.2.3 Parameters

The only parameter that had to be changed was the robot radius. Tuning of other parameters didn't play much on the robot's performance. Neither robot's size and weight, nor its asymmetric design affected its behavior.

4 Results

Both supplied, and custom robots successfully accomplish the task (Fig. 1, 2). The robot reaches the goal position in a minute, and particles estimate its pose with enough precision (Fig. 3, 4). The trajectory is smooth, and the robot does not bump into the walls. It takes about 10 seconds for particles to converge when the robot moves.

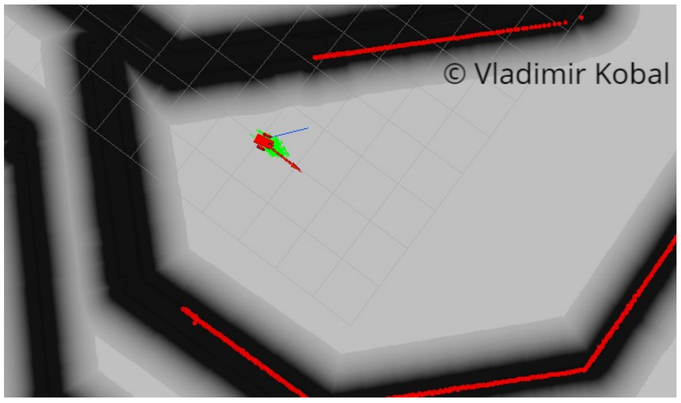


Fig. 1. Supplied model in the goal position

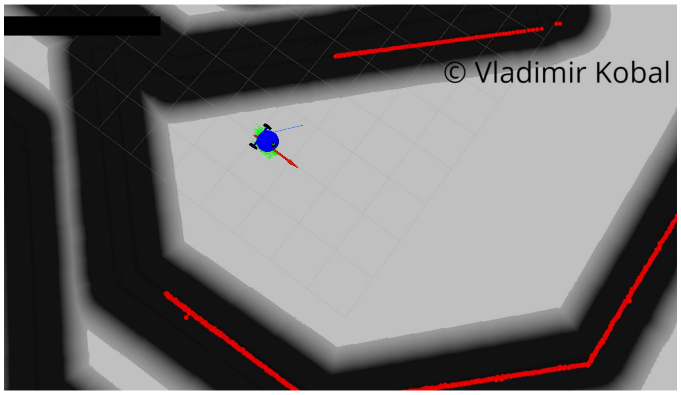


Fig. 2. Custom model in the goal position

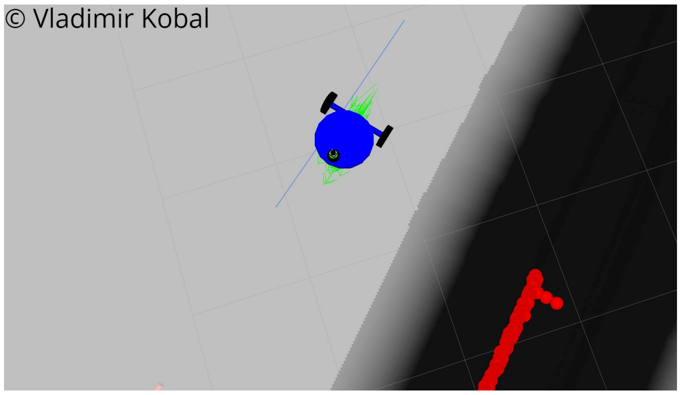


Fig. 3. Pose array while robot moves

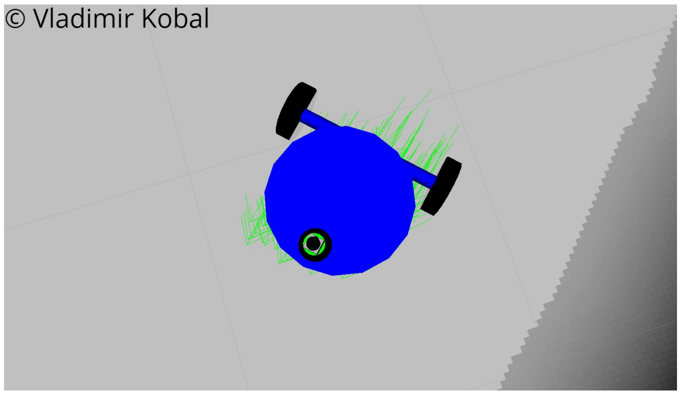


Fig. 4. Pose array when robot reached the goal position

5 Discussion

Performance of the custom robot is similar to the supplied one. Both models are fully functional. Achieved results prove that the AMCL algorithm can be quickly and easily implemented in ROS environment.

To extend the usability of the algorithm in the real world applications kidnapped robot problem can be solved with minor modification to AMCL. For example, it can be implemented by constant monitoring of measurement data and running global localization process in case of unexpected significant changes. However, more work might be required for developing a precise and stable solution. AMCL based models could be used in the development of mobile robots like automatic warehouse transporters, vacuum cleaners, lawnmowers and many other types of self-moving devices.

6 Conclusion / Future work

The project goals were achieved relatively easy thanks to the instruments that ROS offers. Gazebo simulation environment and such a powerful tool as RViz provide a considerable aid in robot development. For deploying this project on hardware, actual resource availability has to be taken into consideration. Usually, computational resources on mobile platforms are strongly limited because of space and power shortage. Pricing can also impose a limitation. So a compromise has to be found between processing time and accuracy for an actual implementation.

The model can be improved in different ways. Additional sensors can be installed to achieve better accuracy. For example, a depth camera could provide additional spatial data. It can also be fruitful to collect data not only in front of the robot but around it. A bigger base could carry more payload and fit more sensors. On the other hand, a smaller base will lead to the better maneuverability of the robot.

References

[1] Kaiyu Zheng. ROS Navigation Tuning Guide. 2016.
[2] ROS Documentation.
[3] ROS Answers.
[4] Robotics StackExchange.