Project “Where Am I”

Dmitry Gavrilenko

**Abstract** – This paper describes the process of a two-wheeled robot model design in Robot Operating System (ROS). The model physics is simulated in Gazebo environment. Its position is localized in a predefined map with a particle filter, implemented by amcl ROS package. A differential\_drive\_controller, following local paths generated by base\_local\_planner/TrajectoryPlannerROS package, controls its motion. Local paths are evaluated to be as close as possible to global paths of navfn/NavfnROS package.  
  
**Index** – ROS, particle filter, navigation, amcl, path planning, localization

# Introduction

Robotic Operating System (ROS) is a framework intended to facilitate robot design, development and debugging. It consists of a physical simulator, hardware drivers, communication protocols and a set of standard algorithms for sensor fusion, computer vision, inverse kinematics, localization and path planning.

The robotic software in ROS is split into multiple processes, called nodes. Nodes exchange messages by subscribing to or publishing topics. Each topic is a communication channel with a predefined message format.

In order to create a mobile robot model in a ROS physical simulator, the developer needs to set up a number of configuration files, defining the model geometry, its physical properties and parameters of nodes, responsible for localization, control and path planning.

# Background

Localization is the process of determining the position and orientation of a robot with respect to the map. Since the real world and robot motion are noisy, the localization process is inherently probabilistic.

There are two popular probabilistic algorithms for localization: Kalman Filter and Particle Filter. Both are based on Bayesian Inference theory and Markov Assumption (see also [1], [2] and [3]).

Bayesian Inference approach treats probability as the amount of uncertainty about the state of the world. Its foundational formula is:

Where *P(o)* is the probability of some observation (e.g. sensor values), *P(s)* is the probability of a state (e.g. position of a robot), *P(s|o)* is the probability of a state given sensor values, and *P(o|s)* is the probability of sensor values given a state.

The formula (1) explains how to predict the probability of a state, given sensor measurements, the previous state and some initial statistics about what states produce what measurements. In case of robot localization, the probabilities are usually continuous values. The formula (1) turns into multiplications of multi-dimensional integrals, which can be approximated by Gaussian distributions (Kalman Filter) or set of randomly generated particles (Particle Filter).

Markov Assumption assumes that the probability of a state *si* depends only on the previous state *si-1* and the current observations *oi*. All the previous history of states and observations is thus ignored. This removes a lot of redundant calculations and enables development of efficient algorithms that work well in practice.

Kalman Filter is one of such algorithms, with the following expectations from states and observations:

* States and observations must be continuous probabilistic values
* Each new state linearly depends on the previous state and current observations; or these dependencies can be linearly approximated
* State and observation distributions are Gaussian or can be approximated by Gaussians

Gaussian distributions are *unimodal*. Unimodality assumes there is only one most probable value, which is located in the center of the distribution.

Unimodal property of Gaussian Filter limits the applicability of the classical implementation of the Kalman Filter: it can be used only for object tracking (inferring position of a robot from the previously known position and observations), but not for localization (inferring position of a robot only from observations, where multiple potential positions, corresponding to the same observations, exist).

Kalman Filter has *O(n3 + m2)* time efficiency, where *n* is the state size, and *m* is the observation size.

Particle Filter, in turn, has the following expectations:

* States and observations must be continuous values
* Each new state may non-linearly depend on the previous state and current observations
* States and observation distributions do not have to be Gaussians. They may also be multimodal. That is, they may have multiple local maximums

Multimodality implies Particle Filter may be applied to a robot localization task, in which the position of a robot within a map is initially unknown, initial observations correspond to multiple positions or orientations in different corners of the map.

With some extensions, Particle Filter may also be used to solve Kidnapping problem, when position of a robot was known, but then it was teleported to a new place so that its position can no longer be inferred from the previous one and should be determined only from observations from inception.

Properties, described so far, hint that Particle Filter outpaces Kalman Filter. However, Particle Filter has one significant disadvantage: its time and space complexity are of *O(pn)* where p is the number of particles along one state dimension, and n is the size of a state.

Since Particle Filter complexity exponentially grows with the state size, in practice, it cannot be applied to states, which dimensionality exceeds 4.

The robot, described in this paper, has 3 degrees of freedom: X and Y coordinate along the ground plane, and orientation yaw. Thus, it can be efficiently localized with Particle Filter, implemented in amcl ROS package.

# Results

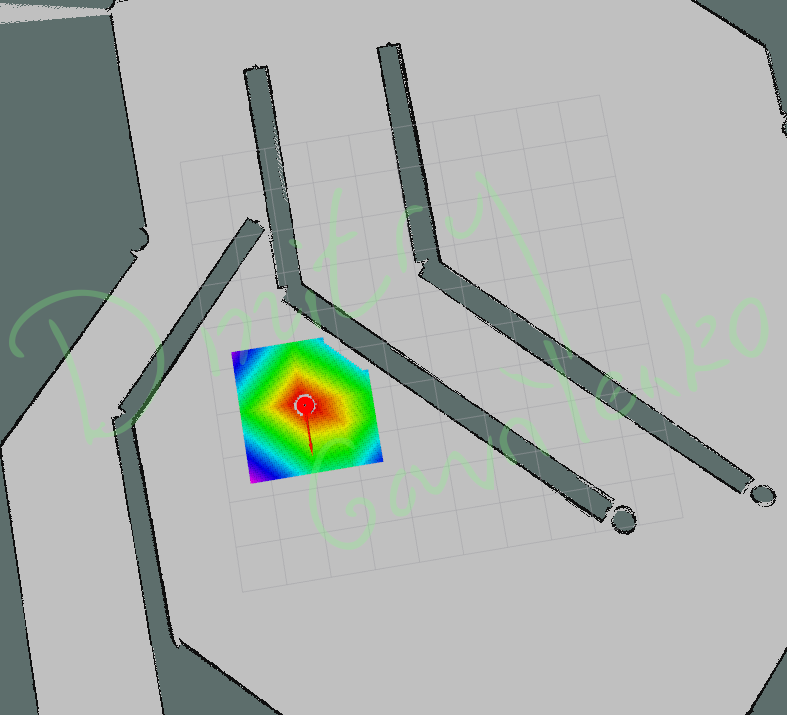


Fig. 1. my\_bot reaching navigation\_goal

Fig. 1 shows the robot, designed from inception within the scope of “Where Am I” project, which has reached the goal, sent by navigation\_goal node.

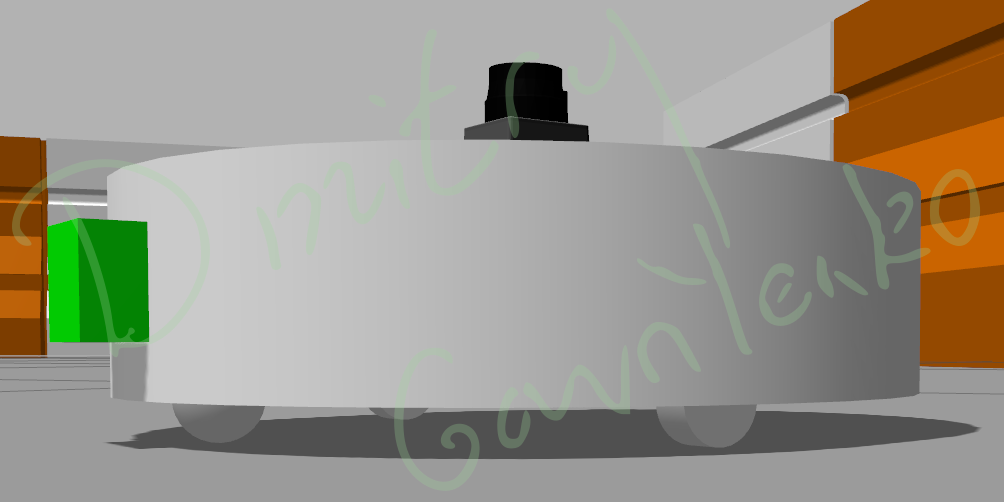


Fig. 2. my\_bot side view

The robot model, shown in Fig. 2, reminds a vacuum cleaner robot with the LIDAR sensor installed in the top center to return 360 degree symmetric scan of the map. The camera is installed in the forward side of the cylindrical shape of the robot in order to avoid occlusion of LIDAR sensor beams.

The forward camera also increases the stability of the robot, shifting its center of mass forward so that only three points of contact with the ground are enough: the caster and two wheels.

The robot wheels are located to the left and to the right from the origin of the robot to enable pure rotational motion around its center. TrajectoryPlannerROS path planner from base\_local\_planner expects that the robot can rotate around the origin of its local reference frame with zero linear velocity. An attempt to shift the wheels to the rear side of the robot failed: in this case, the robot is unable to rotate around its origin without any linear motion.

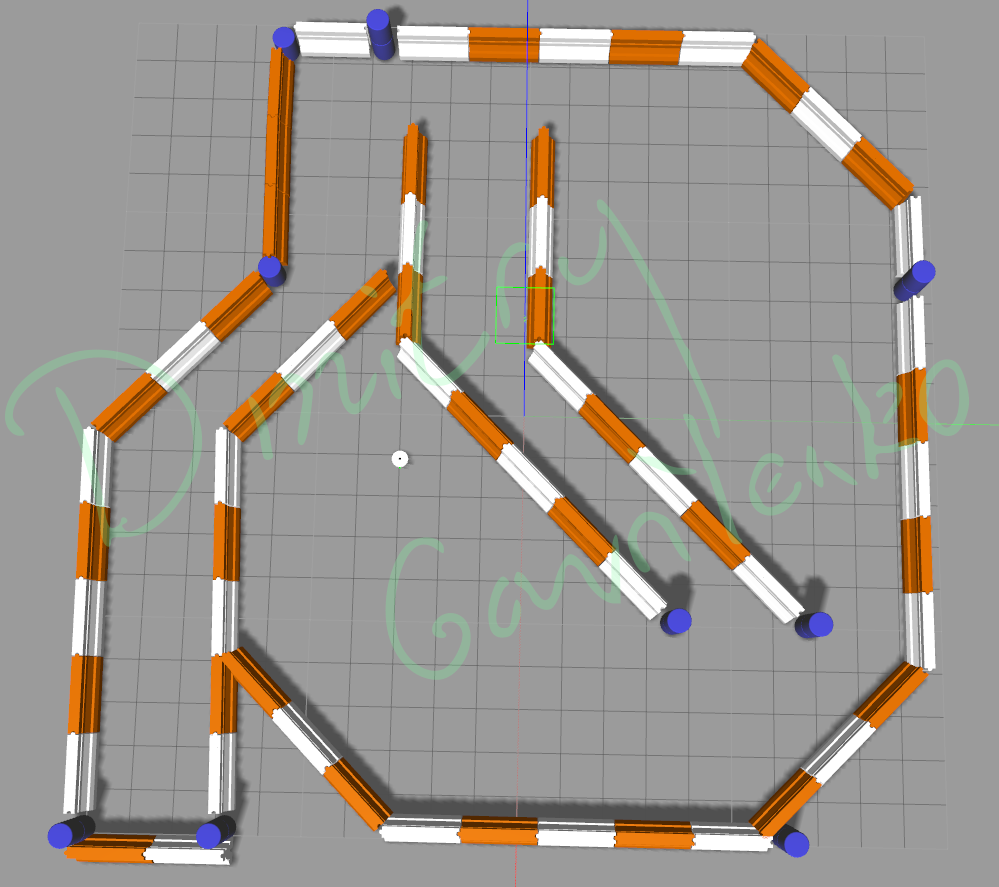


Fig. 3. my\_bot top view in Gazebo

my\_bot moves much faster than udacity\_bot due to decreased weight and friction coefficients. Cylindrical shape simplifies local path planning task, because the orientation of the robot does not affect whether the map region is passable or not.

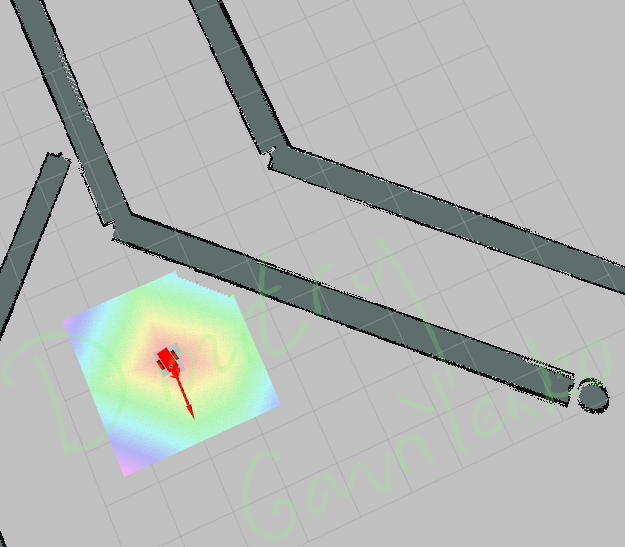


Fig. 4. udacity\_bot reaching navigation\_goal

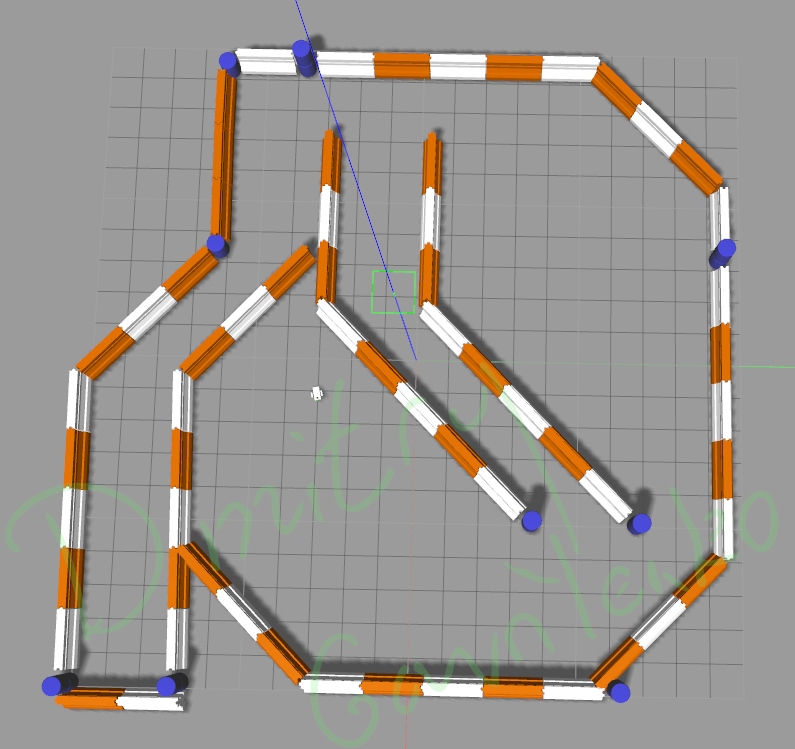


Fig. 5. udacity\_bot top view in Gazebo

udacity\_bot was also set up to work well in practice. Rviz screenshot, corresponding to the navigation\_goal pose, is shown in Fig. 4. Gazebo screenshot, corresponding to the same moment, is in Fig. 5.

# Model Configuration

Model configuration of my\_bot consists of the following files:

./urdf/my\_bot.xacro format is explained in [4] and ROS documentation. It defines the model visual and collision geometry in <links> as well as relationships between different parts of the robot in the form of <joints>. <inertial> values were taken from [12]. Initial tests with small <inertial> values, corresponding to the mass and the shape of the object, produced numerical errors in Gazebo simulator. Small damping and friction coefficients for wheels, as well as zero friction values for the caster helped to eliminate numerical stability issues.

./urdf/my\_bot.gazebo defines additional properties of sensors and controllers installed in the robot. Two actuators attached to left and right wheels control the robot motion. <wheelSeparation> define the distance between the left and right wheels, and <wheelDiameter> define the diameter of the wheels, defined in my\_bot.xacro file. To prevent numerical stability issues with Gazebo, and allow fast motion, mu1 and mu2 friction coefficients are set zero for caster object, which is rigidly fixed to the chassis body: ROS does not have a specific joint type for the caster, therefore it is simulated with the fixed joint and zero friction.

./config/my\_base\_local\_planner\_params.yaml defines the properties of the local path planner TrajectoryPlannerROS. controller\_frequency has been decreased to 10 times per second as a good balance between performance and accuracy of the controller.

./config/my\_costmap\_common\_params.yaml

./config/my\_global\_costmap\_params.yaml

./config/my\_local\_costmap\_params.yaml

# Discussion

TODO

# Future Work

TODO

# References

1. Sebastian Thrun et al, *Probabilistic Robotics*, 2006
2. Roger R Labbe Jr, *Kalman and Bayesian Filters in Python*, 2018, <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>
3. Cameron Davidson-Pilon, *Bayesian Methods for Hackers*, 2016, <https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers>
4. Carol Fairchild et al, *ROS Robotics by Example, Second Edition*, 2017
5. Kaiyu Zheng, *ROS Navigation Tuning Guide*, 2016
6. <http://wiki.ros.org/amcl>
7. <http://wiki.ros.org/navigation>
8. Richard Wang, *[Tutorial] Building a Simulated Model for Gazebo and ROS from Scratch,* 2016,   
   <https://www.youtube.com/watch?v=8ckSl4MbZLg>
9. <https://medium.com/@fernandojaruchenunes/udacity-robotics-nd-project-6-where-am-i-8cd657063585>
10. Pieter Abbeel et al, *Discriminative Training of Kalman Filters*, 2005
11. <https://scikit-optimize.github.io>
12. <https://en.wikipedia.org/wiki/List_of_moments_of_inertia>