Localization Project: Where Am I

Jianfeng Yang

Abstract—In this project, attempts were made to accurately localize two mobile robots inside a provided map in the Gazebo and RViz simulation environments, utilizing ROS packages. Two robots were built: the first one with the provided specification, as a practice; the second one with different shape and placements of the sensors. Several aspects of robotics were learned and discussed, with a focus on ROS. A ROS package was created, which launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack. A significant number of specific parameters corresponding to each package were explored and tuned to achieve the best possible localization results. The final model robot can successfully navigate to its given goal position while simultaneously localizing itself.

Index Terms—Robots, Localization, Navigation, ROS, AMCL, Mobile Robots, Gazebo, RViz, IEEEtran, Udacity.

1 Introduction

Localization is the term used to describe the techniques in determining the pose of a robot in a mapped environment. It goes without saying that for autonomous robots, to know where exactly they are is one of the prerequisites for navigation, and thus is extremely important. The technique is to use a probabilistic algorithm to filter noisy sensor measurement and extract useful information, including position and orientation. Together with mapping, localization has become a vital part of the modern technologies called SLAM. For the past few decades, SLAM has made revolutionary progress and enabled drones, self-driving cars, and other autonomous robots. [1]

1.1 Algorithms

Currently, there are four algorithms used in localization: Extended Kalman Filter, Markov Localization, Grid Localization (also known as Histogram Filter), and Monte Carlo Localization, also known as Particle Filter. Two of them are briefly discussed and compared here namely, the Extended Kalman Filter, and the Monte Carlo Localization.

1.2 Categories of Problems

The localization problems can be categorized into three types. First one is the position tracking, also known as Local Localizing, in which the initial position of the robot is known, and the primary task is to estimate the robot's pose dynamically as the robot moves. Secondly, there is Global Localizing, in which the initial pose is unknown, and it is up to the robot to determining it against a ground truth map. The third and also the most difficult one is called the kidnapped robot problem, which is similar to the global localizing problem, but involving the possible event that the robot might be "kidnapped" and transported to a new location that is unknown to it. Though very uncommon, the kidnapped robot problem indeed represents the worst scenario in localization and provides valuable experience on how to recover from mistakes and errors in localizing, and hence worth investigating.

Another way to categories localization problems is based on the environments the robot is in, namely, static environment and dynamic environment. However, this project focuses solely on static ones, in which the environment will always match the ground truth map and stay unchanged.

2 Background

2.1 Kalman Filter

The Kalman Filter is an estimation algorithm that is extensively used in control engineering, data science, economics, computer vision, and other fields that require real-time estimation of value while it is being sampled. What makes it outstanding is its ability to extract accurate information from a noisy signal with high uncertainty. The processing speed of the Kalman filter is also its advantage, without the need for a significant amount of data to perform the estimation.

2.1.1 How it works

Kalman Filter works as a two-step iterative process: at time zero, an initial guess of the states is made. With this guess as a basis, a state prediction is performed, using the current state information to predict future states. After that, a measurement update takes place, using the measurements from the sensors to update states. This process repeats infinitely while the data keeps coming in.

2.1.2 Common Types

The Kalman Filter is usually categorized into three different types, based mainly on the systems they are applied to:

- Kalman Filter, or the "standard" Kalman Filter, is used on linear systems;
- Extended Kalman Filter, or EKF, is used on nonlinear systems. It is the more applicable Kalman Filter in robotics since the environments robots are dealing with are often nonlinear;
- Unscented Kalman filter, or UKF, is used on highly nonlinear systems.

2.1.3 Extended Kalman Filter

For the "standard" or linear Kalman Filter, one of its notable limitations is that it only works if two assumptions are satisfied: first, the motion and the measurements can be modeled linearly; second, state space can be illustrated by a unimodal Gaussian distribution. These are particular requirements for which most of the systems in the real world would not qualify. Extended Kalman Filter is thus introduced to address this problem, by incorporating linear approximation in the computation for the motion or measurement function, usually with a Taylor Series.

2.2 Particle Filter

Particle Filter, also referred to as the Monte Carlo Localization(MCL), is a probabilistic localization algorithm that can solve the local and global localization problems "in a highly robust and efficient way". It uses virtual particles to represent the pose of the robot, which iteratively approximate and converge to the actual location. [2]

2.3 Comparison with EKF

Comparing with EKF, MCL possesses many advantages:

- MCL can accommodate almost any distribution of the measurement noises and posteriors, while EKF only works with Gaussian;
- MCL is much easier to implement than EKF;
- MCL is more robust than EKF;
- MCL can solve the global localization problem while EKF is not;

However, nothing is perfect, and the same goes for MCL. A couple of limitations of MCL include:

- More resource demanding on the hardware, and often more time consuming compared to EKF;
- Usually has a lower resolution than EKF.

Considering the ease of implementation of MCL and its broad acceptance, this project uses only MCL to achieve localization.

3 Simulations

In this project, two mobile robots were built from scratch. They are then accurately localized inside a provided map in the Gazebo and RViz simulation environments, utilizing various ROS packages.

3.1 Achievements

Both models have successfully reached goal position. Compared with the benchmark model, the personal one has encountered more obstacles that required the change of design, and also consumed much more time on the parameter tuning.

3.2 Benchmark Model

For the first part of the project, a robot is constructed by the given specifications, namely, the Udacity bot.

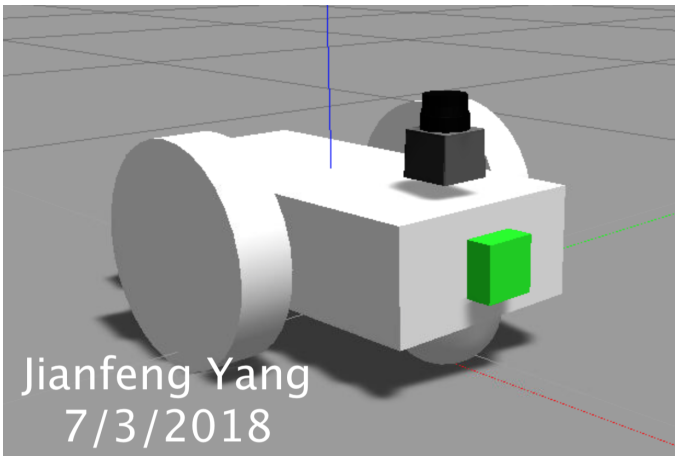

Fig. 1. Udacity Bot

3.2.1 Model design

It has a box-shaped body, with two wheels on the side, and two casters to help it move smoothly. It also has a hokuyo laser scanner and a camera to help it localize and navigate along the way.

3.2.2 Packages Used

The packages used are the same for both parts of the project.

- AMCL: localizing the robot;
- Map server: serve the map files for the project;
- Move base: navigates the robot;
- Joint State Publisher: send joint values, in this project, it is just a dummy;
- Robot State Publisher: send robot state to the TF;
- URDF Spawner: generate a robot inside gazebo using the pre-defined urdf file;
- RViz: launch RViz for the simulation;

3.2.3 Topics

There are lots of topics being published and received, and some of the noticeable ones are:

- Common topics:
  `../parameter_descriptions`
  `../parameter_updates`
  These topics register and updates the parameters for each of the packages.
- AMCL topics:
  `/amcl_pose`

TABLE 1
Bench Mark Model Design

Parts	Shape	Size	Placement
Chassis	Box	0.4 x 0.2 x 0.1	Center
Front Caster	Sphere	r = 0.05	Front bottom
Back Caster	Sphere	r = 0.05	Back bottom
Left Wheel	Cylinder	r = 0.1, l = 0.05	Left Side
Right Wheel	Cylinder	r = 0.1, l = 0.05	Right Side
Camera	box	0.05 x 0.05 x 0.05	Front Center
Laser Scanner	N/A	N/A	Top Front

This topic publishes the pose information of the robot.

- Move base topics:

  `/move_base/NavfnROS/plan`
  `/move_base/TrajectoryPlannerROS/cost_cloud`
  `/move_base/TrajectoryPlannerROS/global_plan`
  `/move_base/TrajectoryPlannerROS/local_plan`

  These topics plans the route for the navigation.

  `/move_base/current_goal`

  This topic provides the current goal to be visualized in RViz.

  `/move_base/global_costmap/costmap`
  `/move_base/global_costmap/costmap_updates`
  `/move_base/local_costmap/costmap`
  `/move_base/local_costmap/costmap_updates`

  These topics publish the global and local cost maps.

- Other topics:

  `/cmd_vel`

  This topic receives velocity commands and controls the movements of the robot.

  `/particlecloud`

  This topic publish the distribution of the particles.

  `/udacity_bot/camera1/image_raw`
  `/udacity_bot/laser/scan`

  These topics publish the images from the camera and the reading from the laser scanner.

3.2.4 Parameters

There are mainly three places to add and tweak the parameters: in AMCL package, in the configuration file for the cost map, and in the Move_base package.

1) transform_tolerance
   This is the parameter to start with. After the model has been created and the environment established, the robot would not be moving and the terminal would be filled with errors and warnings. Adding this parameter will help, though the default value would usually be insufficient, and would require careful tweaking to remove warnings like below.

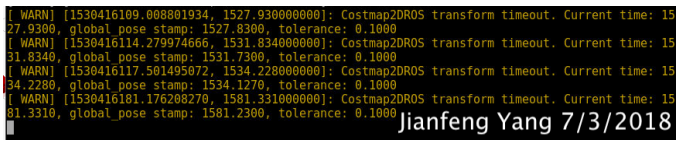

Fig. 2. Transform Time Out

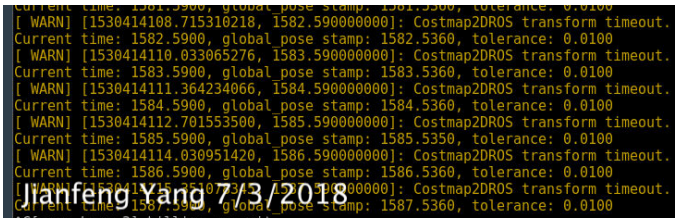

Fig. 3. Transform Tolerance Too Small

It is worth noting that this parameter exists in both the cost map configuration file and AMCL node, and needs to be set up in both places.

2) update_frequency and publish_frequency
   These two parameters are responsible for the following warnings:

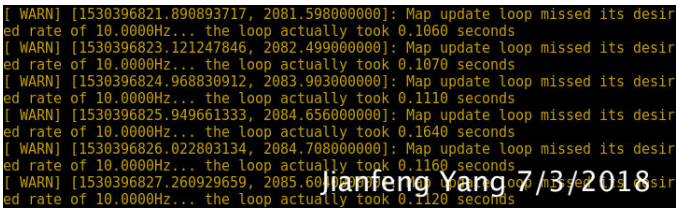

Fig. 4. Update Frequencies Too High

Though high update and publish frequencies are always preferred and desirable for the simulation, the limitation of the hardware usually requires these two parameters to be tuned down.

3) width, height, and resolution
   These three parameters are also responsible for lessening the load on the hardware. However, over tuning could severely impact the performance.

4) controller_frequency
   Sometimes the following warnings will show, especially when the robot is having a hard time finding its way through somewhere narrow or trying to escape from a tight spot. Tuning down this parameter usually address this problem. Caution though, that lower this parameter too much would sometimes cause the robot considerably longer to figure out the next move, and stay immobilized.

Fig. 5. Controller Frequency Too High

5) min_particles and max_particles
   These two parameters reside in the AMCL package and are solely responsible for the amount of particles generated. The provided values are way too large and caused the system to lag and it would take the robot much longer to localize and navigates to its goal.

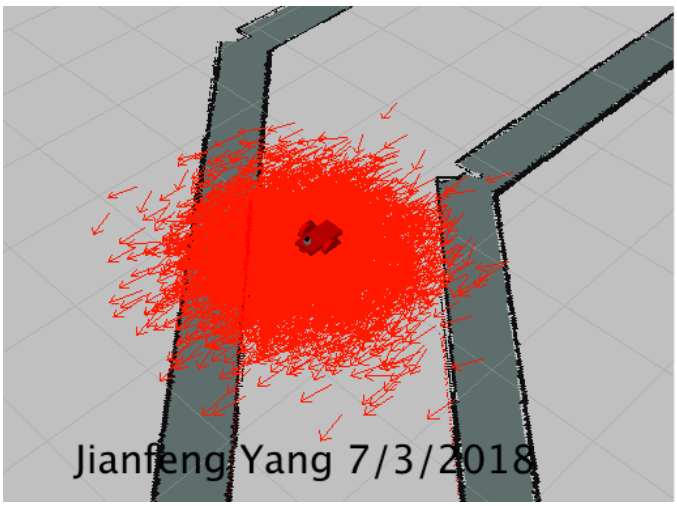

Fig. 6. Provided Amount of Particles: Too Large

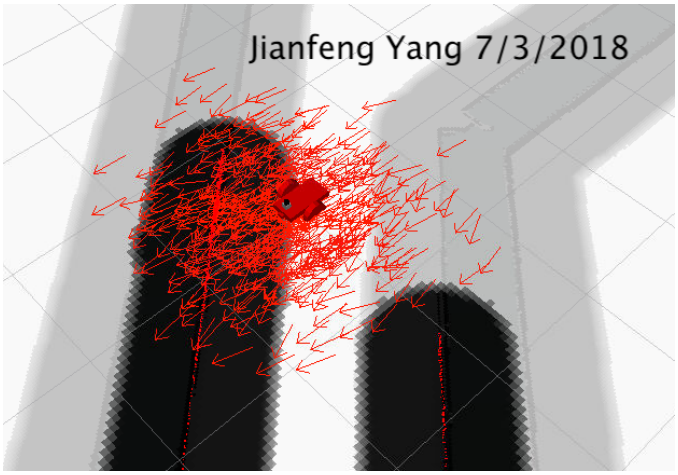

Fig. 7. Less Particles

6) obstacle_range, raytrace_range, and inflation_radius
These three parameters are genuinely the most difficult ones to tune and consumed most of the project time. Though if tuned correctly, the robot can usually reach the goal without touching other parameters.
Generally speaking, a smaller obstacle and raytrace range are preferred. A high value on these two ranges would cause the robot to be "too afraid" to move forward and keeps searching in a circle for wider open space. In a narrow space, this could mean that the robot can never find a "good enough" solution and keeps circling. However, if they are too small, the robot would become too aggressive and head straight for the obstacles, almost always leads to an unrecoverable stuck situation.

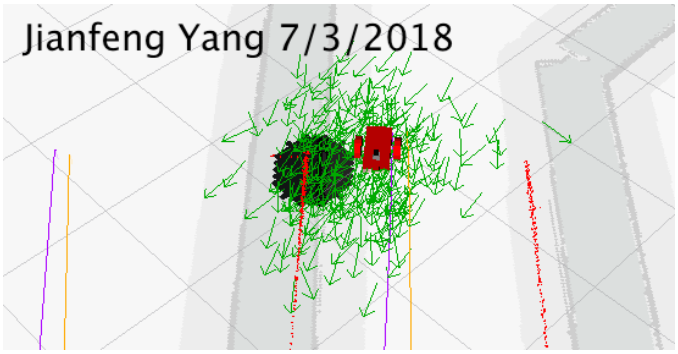

Fig. 8. Raytrace Range Too Small

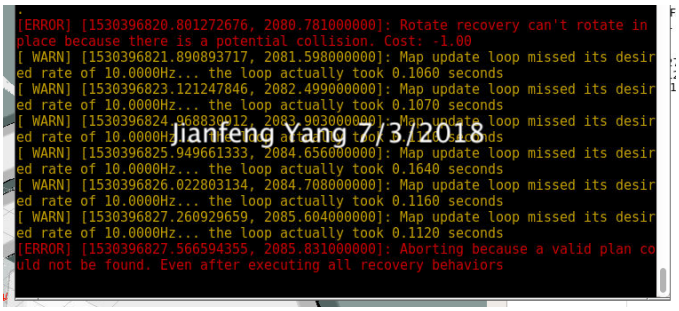

Fig. 9. Robot Stuck

On the other hand, the inflation radius is preferred to be large enough so that on the cost map, the obstacles appear slightly more massive than they are. However, if it is too large, there will not be enough room left for the robot to maneuver, especially when recovery is needed. Therefore, the robot will again keep circling, or move on a less optimal route.
A lot of other parameters listed on the ROS wiki page are tried, but all trials ended up better with default values.

A complete list of parameters and their values are listed below.

TABLE 2
Parameters

Parameters	Values
transform_tolerance	0.2
update_frequency	9.0
publish_frequency	9.0
width (local cost map)	15
height (local cost map)	15
width (global cost map)	30
height (global cost map)	30
resolution	0.05
controller_frequency	16
min_particles	20
max_particles	300
obstacle_range	1.5
raytrace_range	2.0
inflation_radius	0.3

3.3 Personal Model

For the second part of the project, a customized robot is built according to my own design. It is modeled after my vacuum robot, the Roborock, and has the same basic structure.

3.3.1 Model design

It has a circular body, with two main wheels hidden under it, and two casters too. Aside from the body shape, another difference from the Udacity bot is its scanner location: it was placed at the center of the robot's body, instead of the front. Its design was modified multiple times, which will be discussed in detail in the following sections.

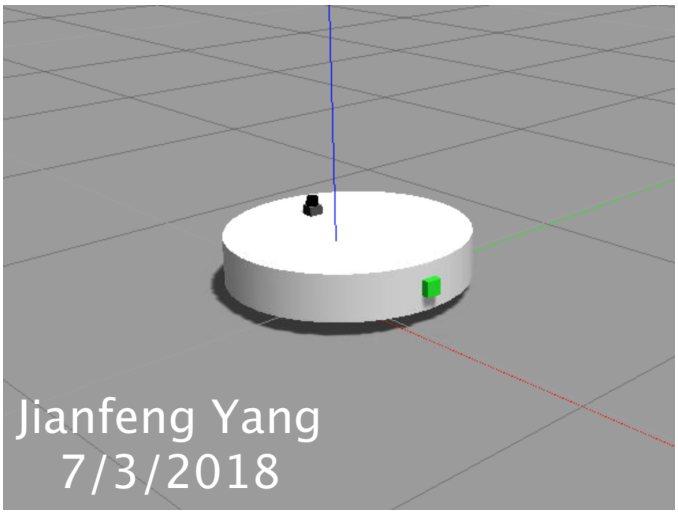

Fig. 10. RoboRock, version 1, Top view

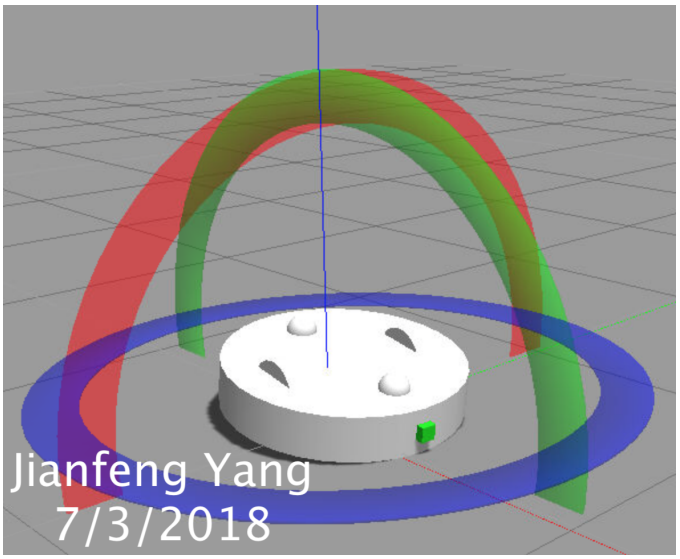

Fig. 11. RoboRock, version 1, Bottom view

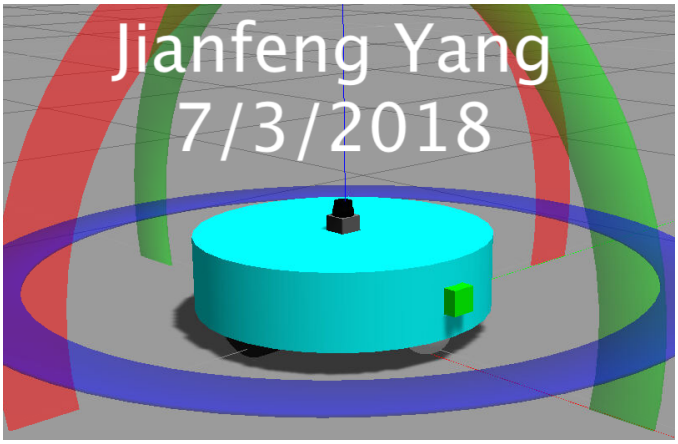

Fig. 12. RoboRock, version 2, Top view

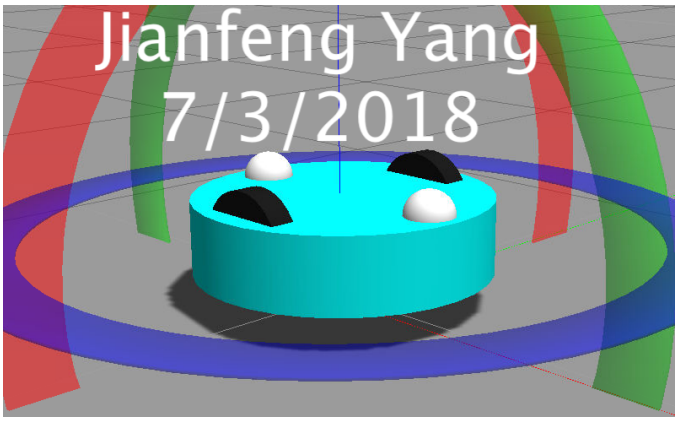

Fig. 13. RoboRock, version 2, Bottom view

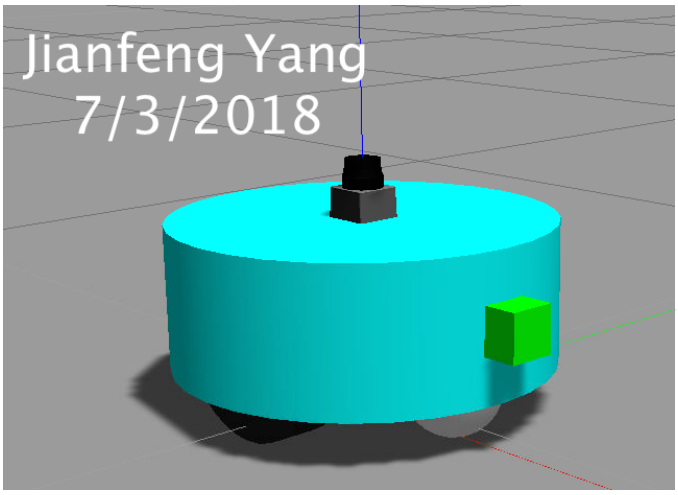

Fig. 14. RoboRock, version 3, Top view

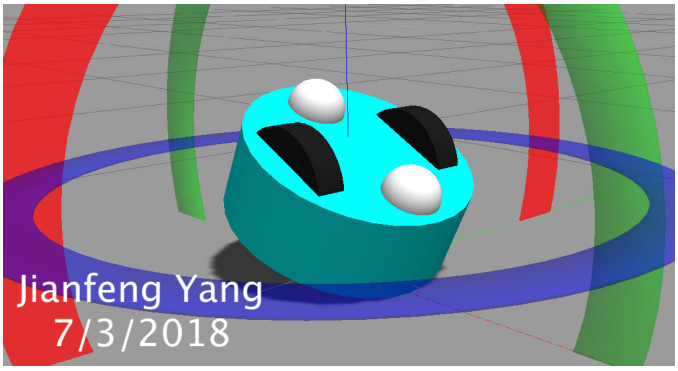

Fig. 15. RoboRock, version 3, Bottom view

TABLE 3
Personal Model Design

Parts	Shape	Size	Placement
Chassis	Cylinder	r = 0.2, l = 0.15	Center
Front Caster	Sphere	r = 0.05	Front bottom
Back Caster	Sphere	r = 0.05	Back bottom
Left Wheel	Cylinder	r = 0.1, l = 0.05	Left Side
Right Wheel	Cylinder	r = 0.1, l = 0.05	Right Side
Camera	box	0.05 x 0.05 x 0.05	Front Center
Laser Scanner	N/A	N/A	Top Center

3.3.2 Parameters

The set of parameters added in the configurations for this model is the same with the Udacity bot, only differs in values.

TABLE 4
Parameters

Parameters	Values
transform_tolerance	0.2
update_frequency	9.0
publish_frequency	9.0
width (local cost map)	15
height (local cost map)	15
width (global cost map)	30
height (global cost map)	30
resolution	0.05
controller_frequency	15
min_particles	20
max_particles	300
obstacle_range	2.0
raytrace_range	2.5
inflation_radius	0.56

Fig. 17. Terminal Shows Udacity Bot at Goal

3.3.3 Changes

As shown in the pictures, the design has undergone various changes. First of all, its body was getting smaller and smaller, which has made it a lot easier for the Roborock to turn and recover in a tight space. Apart from that, its height was slightly increased every time, to hide the wheels and casters inside entirely. Finally, the laser scanner was moved from the back of the body to the center, which has dramatically improved the result of the localization and navigation.

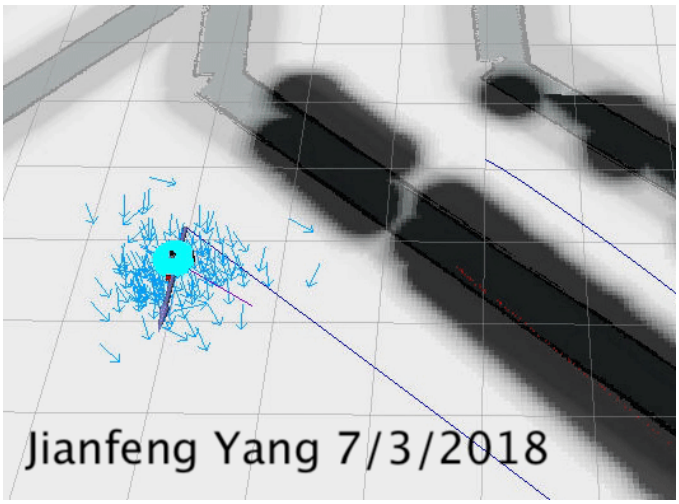

Fig. 18. Roborock at Goal

4 Results

4.1 Localization Results

Both models have successfully reached goal position, as shown below.

4.1.1 Benchmark

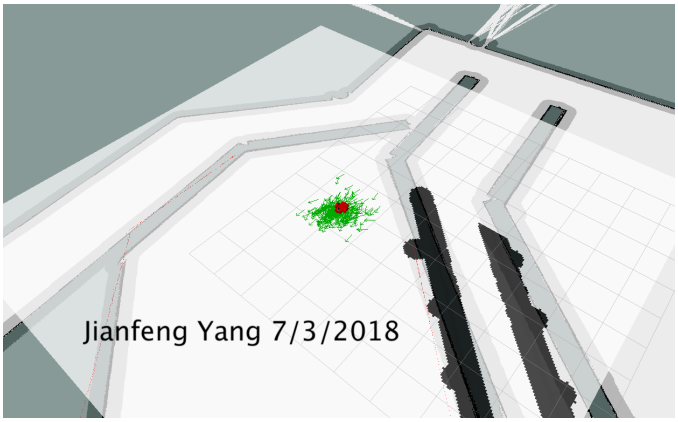

Fig. 16. Udacity Bot at Goal

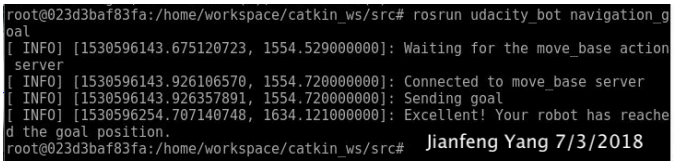

Fig. 19. Terminal Shows Roborock at Goal

4.1.2 Student

4.2 Technical Comparison

Judging by the final result, it can be inferred that the two models perform almost equally in reaching the goal, while Roborock localized slightly better. Having watched the entire navigation process of each robot, one would also argue that Roborock was better at finding a way onto the planned path initially, due to the circular shape and slightly smaller size. Overall, there were no apparent advantages.

5 Conclusion / Future work

In this project, two robot models were constructed using various ROS packages and simulated inside Gazebo and RViz. A simulation for localization and navigation was then performed, and both robots have reached the given goal position within an acceptable time.

5.1 Modifications for Improvement

Due to time constraints, there are still a lot more parameters left untouched.

- Base Dimension
  The body shape of Roborock changes from a box to a cylinder. There are far more different shapes that can potentially promote the result, for example, a hybrid shape of a box front with a circular tail can enable the robot to go nearer to the edge of the walls, improve its mobility.
- Sensor Location
  During the design process of Roborock, two different placements of the laser scanner have been tried, and the central placement has greatly improved the result. In the future, more placements can be investigated.
- Number of Beams
  At this moment, the number of beams is the default value. Increase the beam number to have a wider range of scanning would surely improve the result, but it also put weights on the computation.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, "Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, p. 13091332, 2016.

[2] S. Thrun, D. Fox, W. Burgard, and F. Dellaer, "Robust monte carlo localization for mobile robots," *Artificial Intelligenc*, Summer 2001.