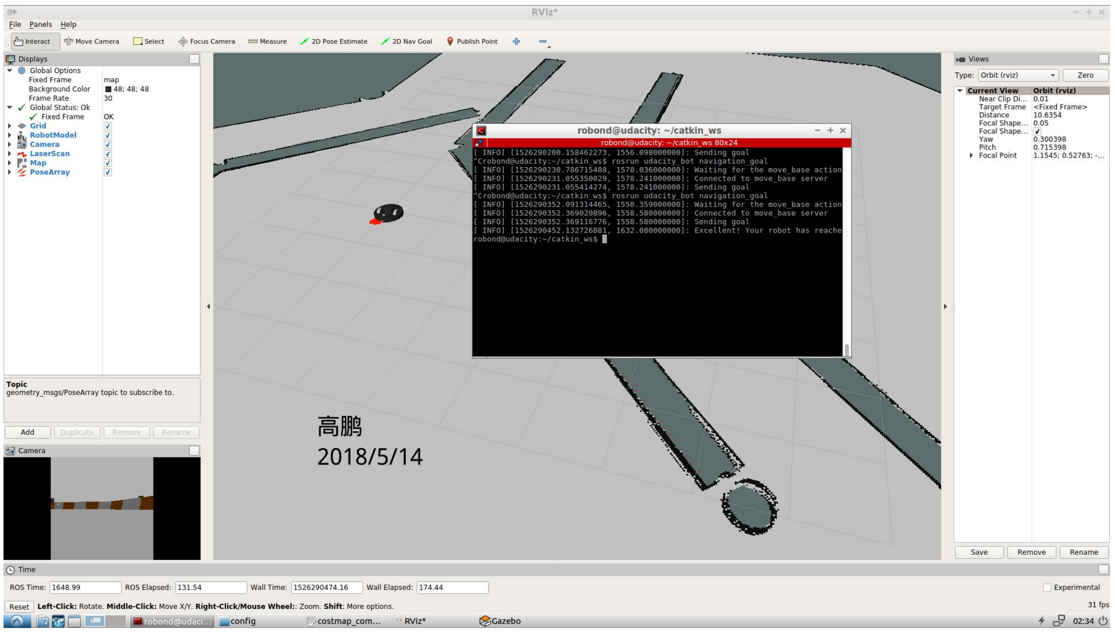
Udacity Robotics Software Engineer Project

Term2 Localization Lab

Abstract

In this project, based on mobile robots, self-designed mobile robots, and ROS packages, real-time robot positioning and map navigation are realized. First, given the initial position and posture of the mobile robot, aside from the location of the robot in the Gazebo and RViz simulation environments; The sensor and ACML (Adaptive Monte Carlo Localization) algorithm are used to estimate and predict the current position and attitude of the mobile robot. Finally, the autonomous navigation of the current map is further realized and the specified target position is reached. In this paper, the corresponding technology will be implemented. For further description and explanation, and provide corresponding results and comparisons.



```
robond@udacity: ~/catkin_ws
[ INFO] [1526290200.158462273, 1556.098000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1526290230.786715488, 1578.036000000]: Waiting for the move_base action
[ INFO] [1526290231.055350029, 1578.241000000]: Connected to move_base server
[ INFO] [1526290231.055414274, 1578.241000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1526290352.091314465, 1558.359000000]: Waiting for the move_base action
[ INFO] [1526290352.369020896, 1558.580000000]: Connected to move_base server
[ INFO] [1526290352.369116776, 1558.580000000]: Sending goal
[ INFO] [1526290452.132726881, 1632.080000000]: Excellent! Your robot has reache
robond@udacity:~/catkin_ws$
```

Figure 1. Result of self-designed mobile robot

Introduction

The project focuses on the following several aspects of robotics:

- In Gazebo and Rviz, mobile robots are created using urdf files to complete the corresponding virtual simulation tasks.
- Create ROS package to launch the mobile robot in urdf to the world of

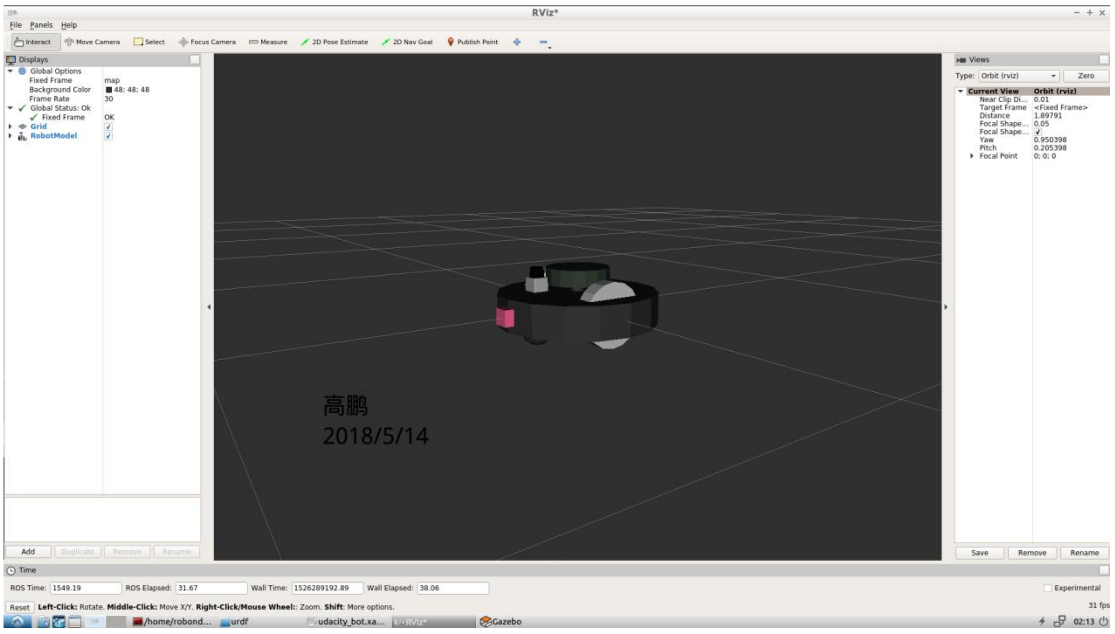
gazebo, and launch ACML and Navigation Stack to realize autonomous navigation of mobile robot in the map.

● Explore and fine tune the parameters related to autonomous navigation accuracy in ACML and config files so that mobile robots can navigate and position as accurately and quickly as possible on the map.

The project created two robots:

● The preset mobile robot model in the project for testing the environment and trying to adjust parameters.(Name:udacity_bot)

● The author designed the mobile robot model to further learn and understand the ACML algorithm's tuning process.(Name:self_designed _bot)


RViz*
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Views
Zero
Add Duplicate Remove Rename
Save Remove Rename
Time
Experimental
Reset
31 fps
/home/robond... urdf udacity_bot.xa... RViz* Gazebo 02:13
高鹏
2018/5/14

```
Global Options
    Fixed Frame          map
    Background Color     ■ 48; 48; 48
    Frame Rate           30
Global Status: Ok
    ✓ Fixed Frame        OK
Grid                     ✓
RobotModel               ✓
```

```
Type: Orbit (rviz)
Current View             Orbit (rviz)
    Near Clip Di...      0.01
    Target Frame         <Fixed Frame>
    Distance             1.89791
    Focal Shape...       0.05
    Focal Shape...       ✓
    Yaw                  0.950398
    Pitch                0.205398
    Focal Point          0; 0; 0
```

```
ROS Time: 1549.19   ROS Elapsed: 31.67   Wall Time: 1526289192.89   Wall Elapsed: 38.06
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
```

Figure 2. Self-designed mobile robot

Both mobile robots must follow the requirements of the C++ navigation target and rely on their own sensors (camera and laser) and ACML algorithm to reach the designated location as soon as possible.

Background

In real life, autonomous positioning and navigation of robots in a known environment are widely used. The basic method of autonomous positioning of mobile robots is to obtain effective information within the environment through the fusion of its own single sensor or multiple sensors. Based on the obtained effective information, a reasonable positioning algorithm is adopted to realize the estimation of its own position and posture and the refresh of measurements information.

In simple terms, the core idea of the positioning problem is to accurately infer the position and posture of the mobile robot in the map, and the navigation problem is to achieve further real-time positioning and decision-making based on the solution of the positioning problem. The accuracy of the robot's position and attitude inference is determined that positioning algorithm is good or bad in a practical problem.

From the surface, this problem is not very complicated. However, in real life, autonomous positioning and navigation algorithms need to be generalized in different environments. In fact, it is suitable for positioning and navigation algorithms in an environment. But It is often difficult to apply it properly to another scene. For example, there are many algorithms for outdoor wide scenes, but it is difficult for indoor scenes that are very cluttered.At the same time, the cumulative error of the sensor in different scenes also brings difficulty in screening sensors for positioning and navigation problems.

In summary, in the positioning and navigation problems, it is mainly necessary to accurately infer the robot's position and posture. Therefore, we must first find a reasonable way to quantitatively represent the robot's position and posture. For the position, you can use Cartesian coordinates and poles. The coordinate system is used to realize its precise representation. For the gesture, the robot's roll, pitch and yaw can be expressed by Euler's formula.

Hardware:

Positioning and navigation problems require heavy calculations, so using a high-performance computer will make experiments faster and smoother. As shown:

Processor: Intel i7-4700MQ CPU

RAM: 16GB

Operation System: Window 10 Pro

VMware Workstation 12 Pro

Processor: 4

Memory: 8GB

Hard Disk: 50 GB

Software:

1.Using an Udacity ROS (Kinetic) package to create a robot simulation environment on VMWare machine. This ROS includes Python (2.7), Gazebo (7.10.0) and RViz (1.12.15) packages.

2. Using URDF (Unified Robot Description Format) to create the robot model

which includes pose, inertial, collision and visual data. Two sensors - a camera and a laser rangefinder (Hokuyo)[1] was added in this URDF model.

3. A map created by Clearpath Robotics[2] was used for both robots in the project.

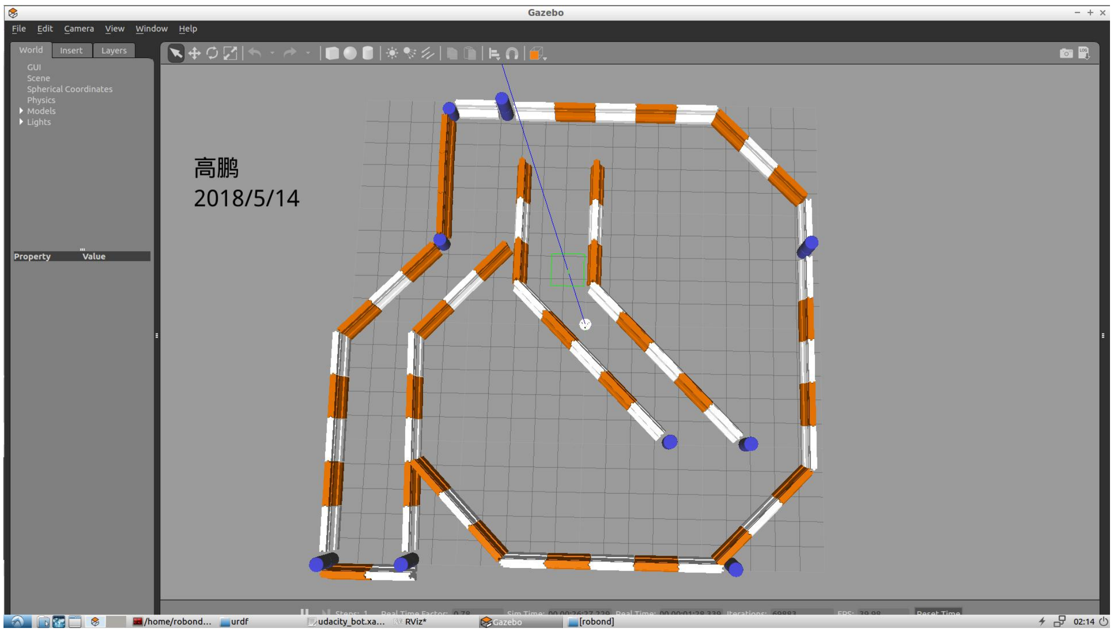


Figure 3. Map of localization and navigation

4. AMCL (Adaptive Monte Carlo Localization) algorithm was used to dynamically adjust the number of particles over a period of time.

5. A C++ code was used to send a target position to move_base action server.

Kalman Filters and Monte Carlo Localization

(1).Kalman Filters and EKF

The algorithm is divided into two steps: Prediction state and Measurements update. In the prediction step, the Kalman filter generates an estimate based on the current state variables and their uncertainty. Once the next measurement (including random noise) is observed, these estimates will be updated using the weighted average, and so on. The recursive algorithm will generate more weights to further improve the certainty of the estimate. It can be run in real time, using only the current input measurements and the previously calculated status and its uncertainty matrix; there is no additional past Information is required.

Although the Kalman filter does not require Gaussian distribution of the input data, the state estimation of the output of the Kalman filter becomes Gaussian.

The ordinary Kalman filter is only suitable for linear systems, but the practical problems are often nonlinear. In order to generalize the Kalman filter for nonlinear systems, an extended Kalman filter (EKF) is proposed. Gaussian distribution in EKF The expectation is transformed into a vector, and the variance is transformed into a covariance matrix[3].

(2).Monte Carlo Localization

Monte Carlo Localization is an algorithm for robots localization using particle positioning filter.

In a given environment map, the position and attitude are estimated based on the information acquired by the sensors during the movement of the robot. The core of the algorithm is to use a particle filter to represent all possible positions and poses of the mobile robot in real time at any moment. First, the space is uniformly distributed from the particles (this means that the robot has no information about its position and assumptions), where each particle represents the possibility of a position and attitude of the mobile robot at this moment. Whenever the robot moves, this algorithm will move particles to predict their new post-movement state. Whenever the mobile robot refreshes the sensor information, the algorithm will resample based on these particles. The Bayesian estimation is performed based on the recursive algorithm, which evaluates the degree of correlation between the actual sensed data and the prediction. Finally, all particles should converge on the actual robot’s position and posture[4].

(3).Compare Monte Carlo Localization vs. Extend Kalman Filters

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Guassian
Posterior	Particles	Guassian
Efficiency(memory)	OK	Good
Efficency(time)	OK	Good
Ease of implementation	Good	OK
Resolution	OK	Good
Robustness	Good	Poor
Memory & Resolution Control	Yes	No

Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodel Continuous

The Challenges

The positioning problem of mobile robots is the basis of automation and intelligence of robots. The main idea is to infer the position and attitude of mobile robots in real time based on sensor data and positioning algorithms. In order of increasing difficulty, there are the following three kinds of robot positioning problems.

(1).Local Localization

Assuming that the initial pose of the robot is known, then the position and attitude of the mobile robot are explored and inferred. This is considered to be a local problem because the inferred uncertainty is limited to the region close to the actual pose.

(2).Global Localization

Global localization does not assume the initial pose with respect to the Local Localization problem. However, it subsumes the local problem since it uses knowl- edge gained during the process to keep tracking the position. The goal of the MCL algorithm introduced in this assignment is to perform global localization.

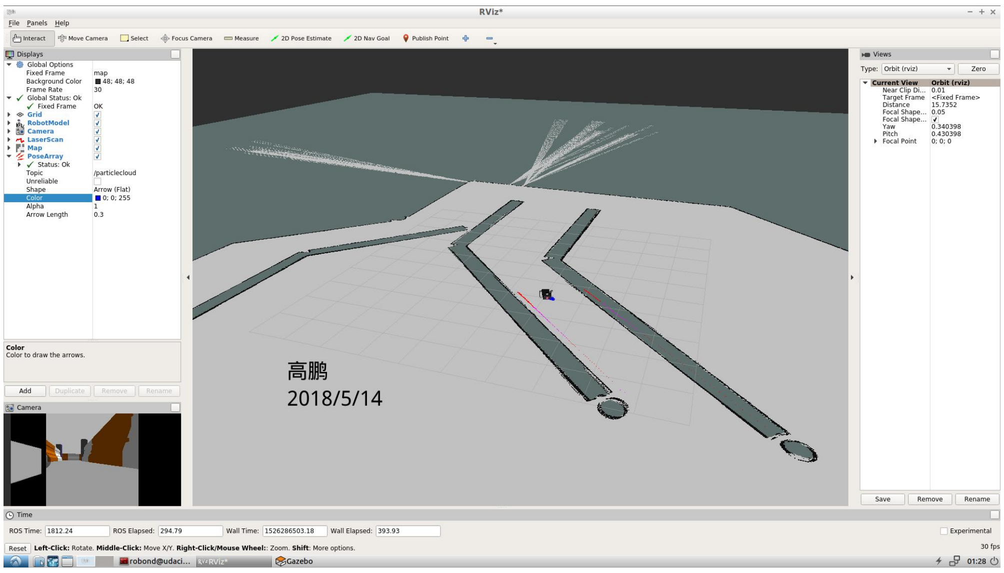
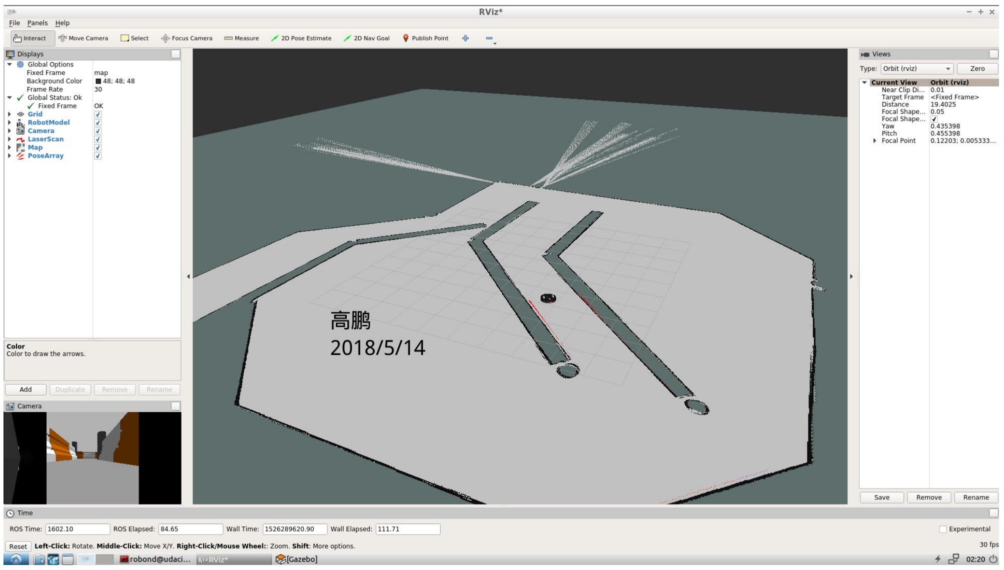
(3).Kidnapped Robot Problem

The kidnapped robot problem arises from the movement of a successfully localized robot to a different unknown position in the environment to see if it can globally localize. Thus it is more difficult than global localization problem since the robot has a strong but wrong belief in where it is. The original MCL algorithm does not have the ability to recover from kidnapping. Such failure is also often referred to as catastrophic failure.

Results

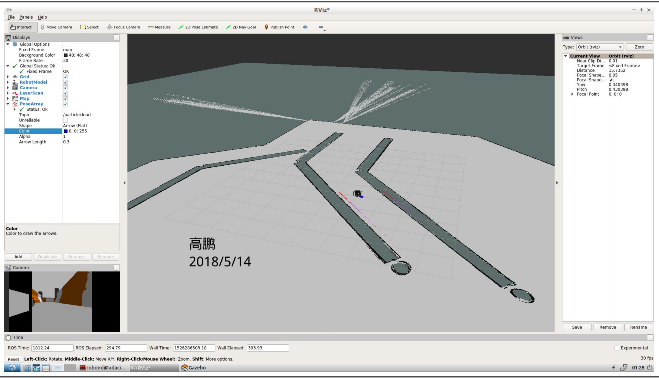
(1).Testing scenario:

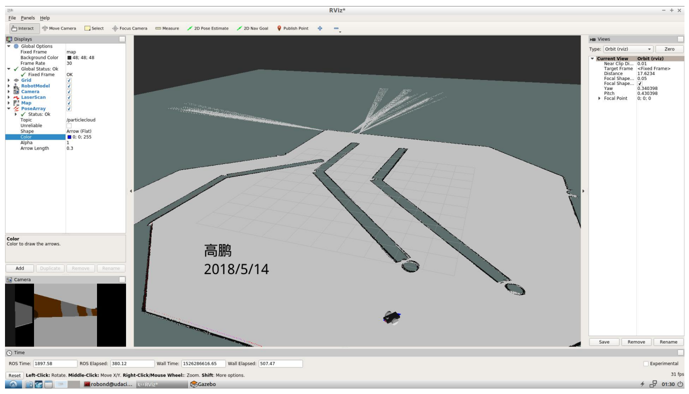
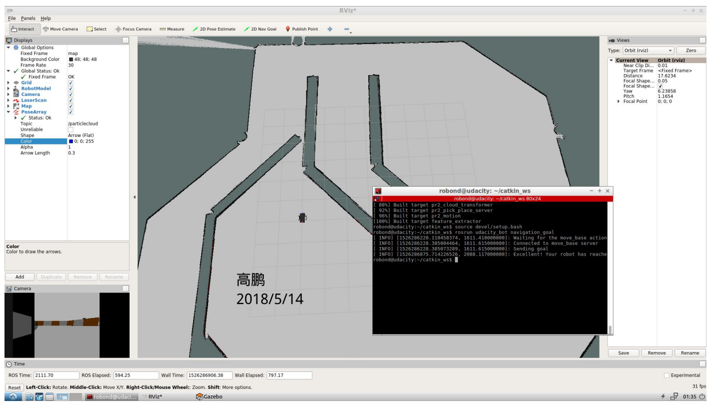
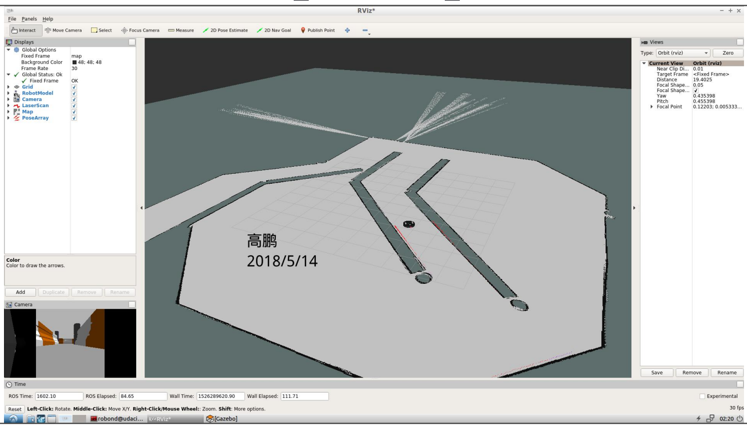
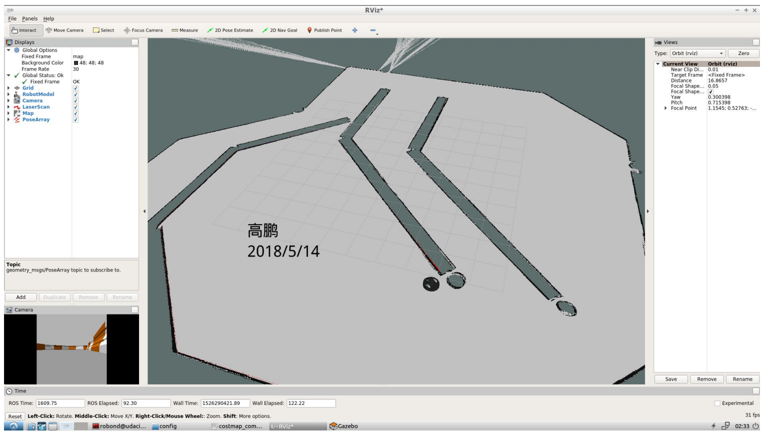
Both robots used the same map with same starting (0 0 -0.785) and target (0.995 -2.99 0) position.

udacity_bot

self_designed _bot


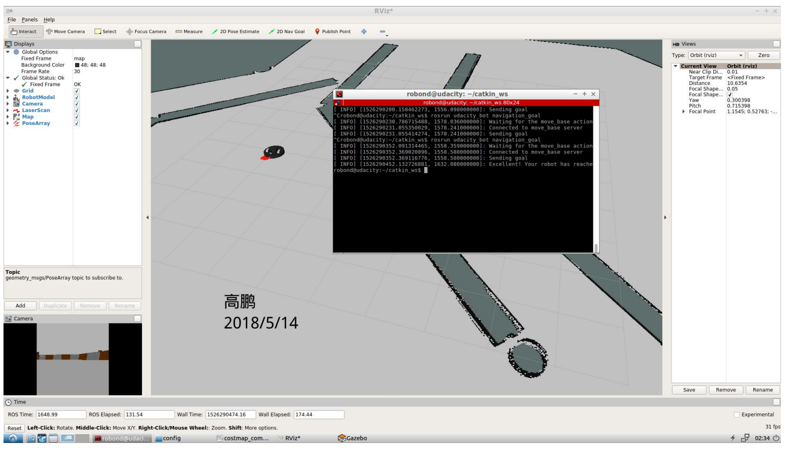
Testing results

Both robots navigated in map well and could arrive to the target position within reasonable time.

	udacity_bot
Go straight	

Make a turn	
Arrived target	
Average Time	7-8 minutes
	self_designed _bot
Go straight	
Make a turn	

```
robond@udacity: ~/catkin_ws
robond@udacity: ~/catkin_ws 80x24
[ 88%] Built target pr2_cloud_transformer
[ 92%] Built target pr2_pick_place_server
[ 96%] Built target pr2_motion
[100%] Built target feature_extractor
robond@udacity:~/catkin_ws$ source devel/setup.bash
robond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1526286228.110458374, 1611.410000000]: Waiting for the move_base action
[ INFO] [1526286228.385004464, 1611.615000000]: Connected to move_base server
[ INFO] [1526286228.385073289, 1611.615000000]: Sending goal
[ INFO] [1526286875.714226526, 2088.117000000]: Excellent! Your robot has reache
robond@udacity:~/catkin_ws$
```

Arrived target	
Average Time	3-4 minutes

```
robond@udacity: ~/catkin_ws
[ INFO] [1526290200.158462273, 1556.098000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1526290230.786715488, 1578.036000000]: Waiting for the move_base action
[ INFO] [1526290231.055350029, 1578.241000000]: Connected to move_base server
[ INFO] [1526290231.055414274, 1578.241000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1526290352.091314465, 1558.359000000]: Waiting for the move_base action
[ INFO] [1526290352.369020896, 1558.580000000]: Connected to move_base server
[ INFO] [1526290352.369116776, 1558.580000000]: Sending goal
[ INFO] [1526290452.132726881, 1632.080000000]: Excellent! Your robot has reache
robond@udacity:~/catkin_ws$
```

First of all, both mobile robots need to explore above the track during the process of starting navigation. After one round of circling, they will go straight down to the lower track and turn to reach the target location. It will obviously waste some time in the process of wrapping. Secondly, the design The parameter robot_radius is important for mobile robots to navigate and successfully reach the target location. Its value depends on the size of the mobile robot.

Model Configuration

These parameters were adjusted in the project to improve the robot performance:

● /amcl/laser_model_type:likelihood_field_prob=(string,default:"likelihood_field") Which model to use, either beam, likelihood_field, or likelihood_field_prob (same as likelihood_field but incorporates the beamskip feature, if enabled)[5]

Used likelihood_field_prob, make laser sensor has beamskip feature.

● /amcl/max_particles: 300 (int, default: 5000) Maximum allowed number of particles.

● /amcl/min_particles: 50 (int, default: 100) Minimum allowed number of particles[5].

Adjusted these two value lower to reduce CPU usage and improve performance.

● /amcl/resample_interval: 1.5 (int, default: 2) Number of filter updates required before resampling[5].

Set a lower value to Increase the number of resamples and to improve performance.

- /amcl/transform_tolerance: 3.5 (int, default: 2) Number of filter updates required before resampling[5].

  Set the value higher to improve localization accuracy.

- /move_base/TrajectoryPlannerROS/sim_time: 3.0 (double, default: 1.0) The amount of time to forward-simulate trajectories in seconds[5].

  Set higher value to speed up robot navigation.

- /move_base/TrajectoryPlannerROS/xy_goal_tolerance: 0.10 (double, default: 0.10) The tolerance in meters for the controller in the x & y distance when achieving a goal[5].

  Reduce the value to increase the challenge to achieve a goal

- /move_base/controller_frequency: 6.0 (double, default: 20.0) The frequency at which this controller will be called in Hz. Uses searchParam to read the parameter from parent namespaces if not set in the namespace of the controller. For use with move_base, this means that you only need to set its "controller_frequency" parameter and can safely leave this one unset[5].

Set the lower value to eliminate the warning message "Control loop missed its desired rate of 20.0000Hz". This parameter doesn't impact robot performance, but it will reduce these unnecessary warning messages on the screen and in the log file.

- /move_base/global_costmap/raytrace_range: 10.0

- /move_base/local_costmap/raytrace_range: 10.0 (double, default: 3.0) The maximum range in meters at which to raytrace out obstacles from the map using sensor data[5].

  Used higher value to increase sensor detecting obstacles distance

- /move_base/global_costmap/robot_radius:0.19

- /move_base/local_costmap/robot_radius: 0.19 (double, default: 0.46) The radius of the robot in meters, this parameter should only be set for circular robots, all others should use the "footprint" parameter[5].

Set a lower value to match the project robot size. This value depends on the specific size of the mobile robot.

- /move_base/global_costmap/transform_tolerance:0.5

- /move_base/local_costmap/transform_tolerance: 0.5 (double, default: 0.2) Specifies the delay in transform (tf) data that is tolerable in seconds. This parameter serves as a safeguard to losing a link in the tf tree while still allowing an amount of latency the user is comfortable with to exist in the system[5].

  Set a little higher value to increase the system tolerable.

Discussion

- According to the specific mobile robot parameters adjustment (ACML and config file) is a relatively difficult part of the project. Each parameter has a corresponding impact on the entire positioning algorithm, a reasonable adjustment of each parameter, to learn from each other to meet the requirements High-quality algorithms. Therefore, in the process of specific adjustment of parameters, the specific requirements of the problem and the interrelated effects of the parameters of each other must be considered.
- For kidnapped robot problem, MCL/ACML algorithm will not be able to achieve a good positioning and tracking requirements. Even if the sensor can obtain the corresponding position and attitude information, but in the new environment is also difficult to accurately infer the new position and posture.
- MCL/ACML algorithm can be used in fixed scenes such as factories and shops, but it is difficult to adapt to changing scenes.

Future Work

- The self-designed mobile robot performs better than the preset mobile robot in realizing the turning motion. However, in the process of adjusting the parameters, it may also happen that the mobile robot is stuck in the turning. This phenomenon requires further adjustment of the parameters to find the optimum.
- The fusion of more sensors on the mobile robot makes it possible to acquire more measurement information in the real-time positioning and tracking of the mobile robot, thereby more accurate positioning of the robot.
- Try to implement adjustment algorithms based on deep learning to further improve the efficiency and effectiveness of tuning parameters.

Reference

[1] Hokuyo, “Hokuyo laser scanner home page.” 2018.

[2] ClearPathRobotics,”Clearpath robotics home page.”2018.

[3] Wikipedia, “Kalman filter”.2018.

[4] Wikipedia, “Monte Carlo localization”.2018.

[5] wiki.ROS.ORG, “Documentation”.2018.