PROJECTWHERE AM I***

Abstract – In this paper, the topic of localization is discussed and applied to two robots in simulation using ROS, Gazebo, and RViz to successfully traverse through a given maze and reach the desired goal pose and orientation. The localization technique specifically used in the simulation is the Adaptive Monte Carlo Localization, or AMCL, a Particle filter-based algorithm, which is compared to another algorithm, the Kalman filter. The resulting performance of each robot is then compared in terms of efficiency, and the design of the created robot is discussed along as to why certain parameters were chosen to improve accuracy of localization for each robot.

------------------------------------ ✐✐✐ ------------------------------------

1. Introduction

Localization approximation of the current position of a robot given in robotics means determining good uncertainties of noisy sensors such as a camera or LIDAR (LightDetection and Ranging) and uncertainties due to imperfect actuators moving the robot.

Two robots were developed and tested in a simulation environment. The robots successfully navigated the maze using the Adaptive Monte Carlo Localization (AMCL) algorithm. The benchmark definition of one of the robots was given as part of the project, while the second was created independently.

The benchmark robot is called *'Udacity_Bot'* :

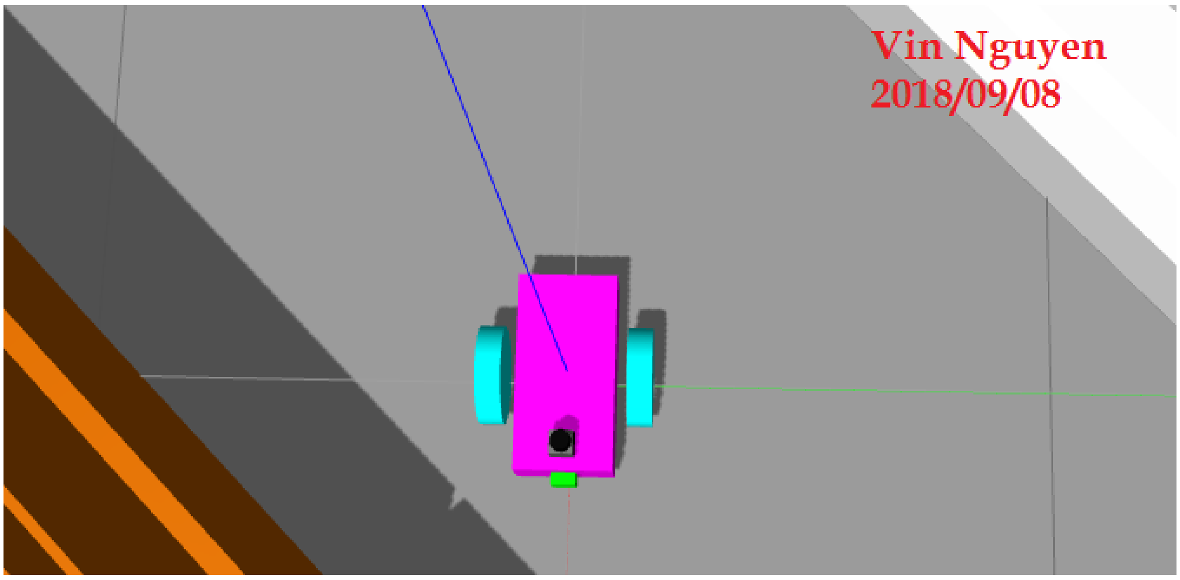


Figure 1: Udacity_Bot

And the second robot designed for this project is called *'My_Bot'* throughout this paper.

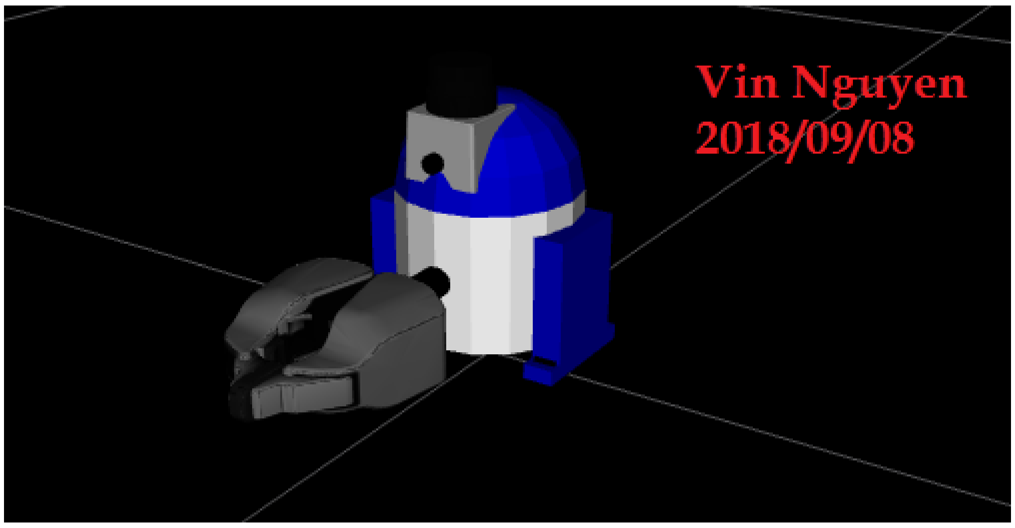


Figure 2: My_Bot

2. Background

Localization is a fundamental issue in developing mobile robots. In an imperfect world where sensors are inaccurate or noisy and actuators are imprecise, localization is central to determining a good estimate of the actual position of a robot. The two most common approaches to Localization are Kalman Filters and Monte Carlo Simulation.

2.1 Kalman Filter

The Kalman Filter is very prevalent in control systems. It is very good at taking in noisy measurements in real time and providing accurate predictions of actual variables suchas position. The Kalman Filter algorithm is based on two assumptions:

- Motion and measurement models are linear.
- State space can be represented by a unimodal Gaussian distribution.

These assumptions limit the applicability of the Kalman Filter as most real robot scenarios do not meet these assumptions.

The Extended Kalman filter (EKF) addresses these limiting assumptions by linearizing a nonlinear motion or measurement function with multiple dimensions using multidimensional Taylor series. While the EKF algorithm addresses the limitations of the Kalman Filter, the mathematics is relatively complex and the computation uses considerable CPU resources.

2.2 Particle Filter

Particle Filters operate by uniformly distributing particles throughout a map and then removing those particles that least likely represent the current position of the robot. The Monte Carlo Filter is a particle filter that uses Monte Carlo simulation on an even distribution of particles to determine the most likely position value. Computationally it is much more efficient the Kalman Filters. Monte Carlo localization is not subject to the limiting assumptions of Kalman Filters as outlined above. The basic MCL algorithm steps are as follows:

1) Particles are drawn randomly and uniformly over the entire space.

2) Measurements are taken and an importance weight is assigned to each particle.

3) Motion is effected and a new particle set with uniform weights and the particles are resampled.

4) Measurement assigns non-uniform weights to the particle set.

5) Motion is effected and a new resampling step is about to start. In this implementation we use Adaptive Monte Carlo Localization. AMCL dynamically

adjusts the number of particles over a period of time, as the robot navigates a map.

2.3 Comparison

Kalman Filters have limiting assumptions of a unimodalGaussian probability distribution and linear models of measurement and actuation. Extended Kalman filters relax these assumptions but at the cost of increased mathematical complexity and greater CPU resource requirements. Monte Carlo Localization does not have the limiting assumptions of Kalman filters and is very efficient to implement. As such Monte Carlo Localization is used in thisproject. The differences between the approaches can be summarized in a table:

AMCL AND EKF Comparison		
Measurements	Raw	Landmarks
Measurement Noise	Any	Gaussian
Posterior Belief	Particles	Gaussian
Memory Efficiency	OK	Good
Time Efficiency	OK	Good
Ease of Implementation	Good	OK
Resolution	OK	Good
Robustness	Good	Poor
Global Localization	Yes	No
State Space	Multimodal Discrete	Unimodal Continuous

Table 1: AMCL and EKF comparison

3. Simulation

The simulations show the actual performance of the robots. The simulations are done in an environment using Gazebo and RViz tools for visualization. The Navigation Stack as derived from the Willow garage NavStack [4] can be visualized as follows:

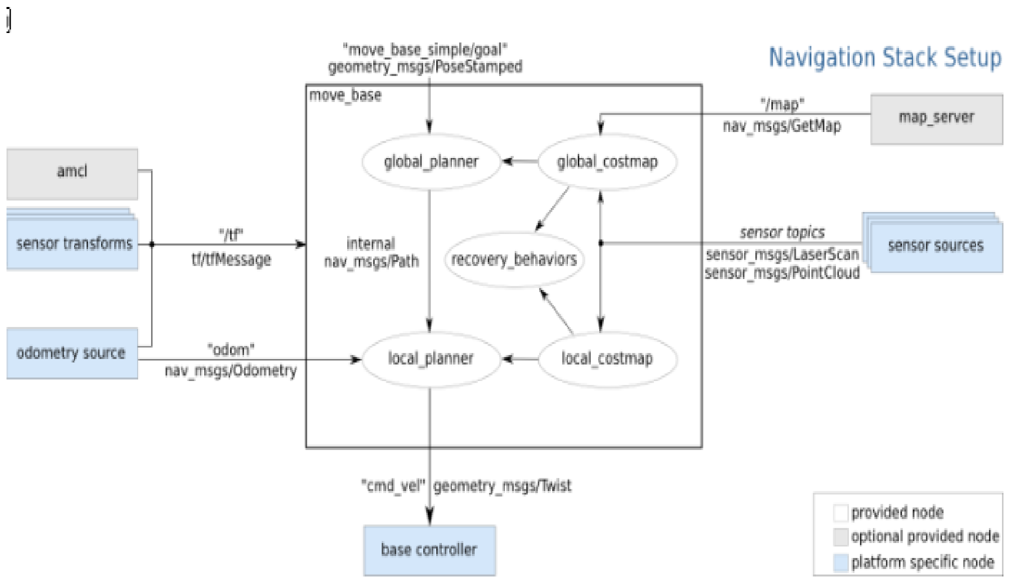


Figure 3: Navigation stack

3.1 Udacity Bot

At the start of the simulation the particles are broadly distributed, indicating great uncertainty in the robot position. At this point the sensors have not yet provided any information as to location. Figure 4 shows the broad distribution of particles (shown as green arrows). After starting the simulation, sensor measurements are taken and the localization of the robot improves. Figure 5 shows the narrowing distribution of particles as more sensor data is gathered.

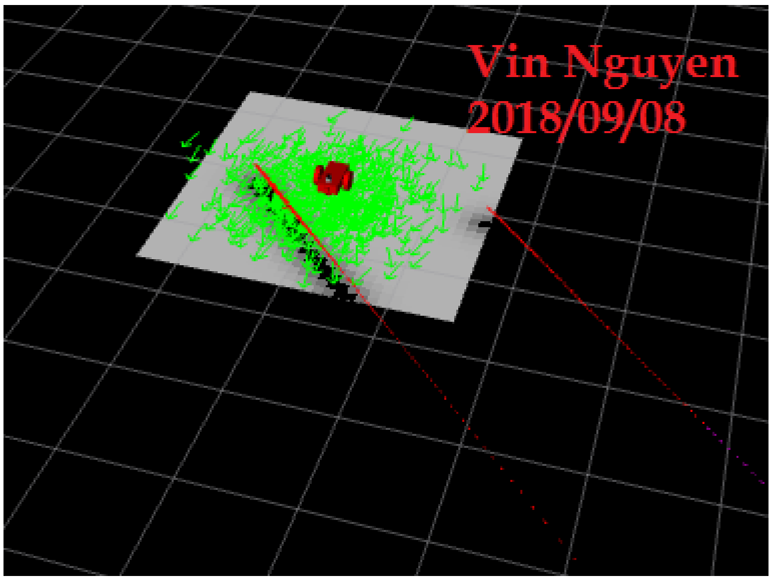


Figure 4: Udacity_Bot shown at the Start Position

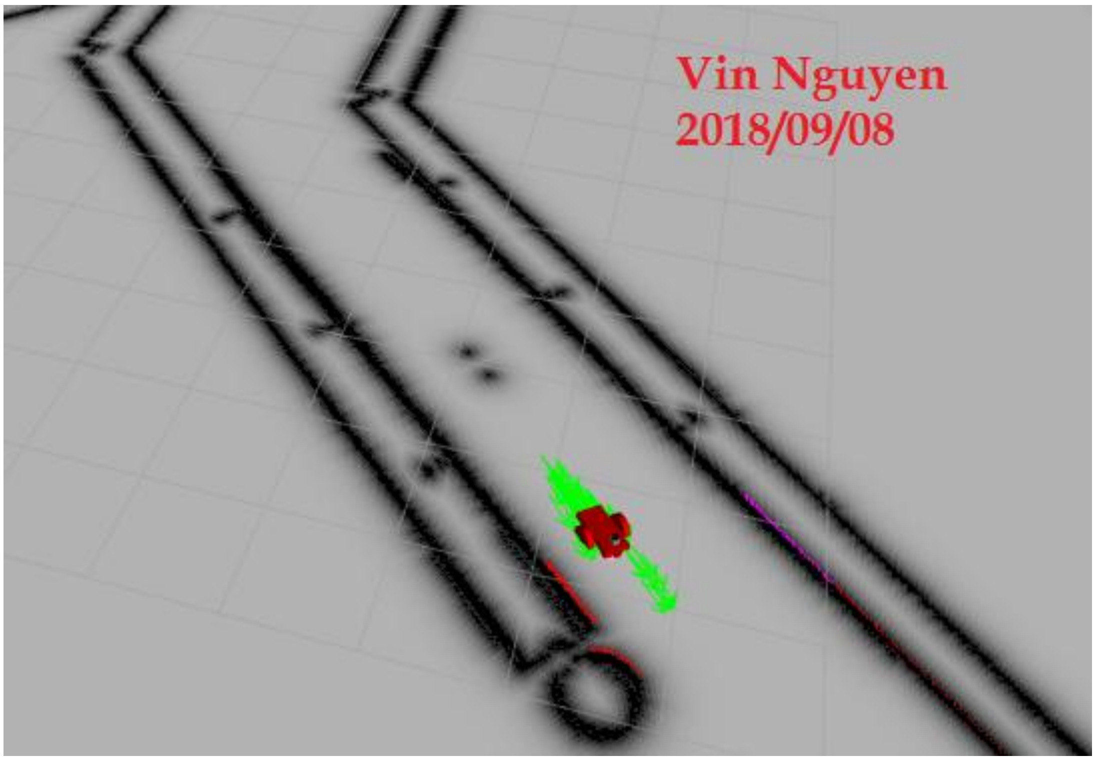


Figure 5: Udacity_Bot shown moving along the navigation path

3.2 My Bot

At the start of the simulation the particles are broadly distributed, indicating great uncertainty in the robot position. At this point the sensors have not yet provided any information as to location. Figure 6 shows the broad distribution of particles (shown as green arrows). After starting the simulation, sensor measurements are taken and the localization of the robot improves. Figure 7 shows the narrowing distribution of particles as more sensor data is gathered.

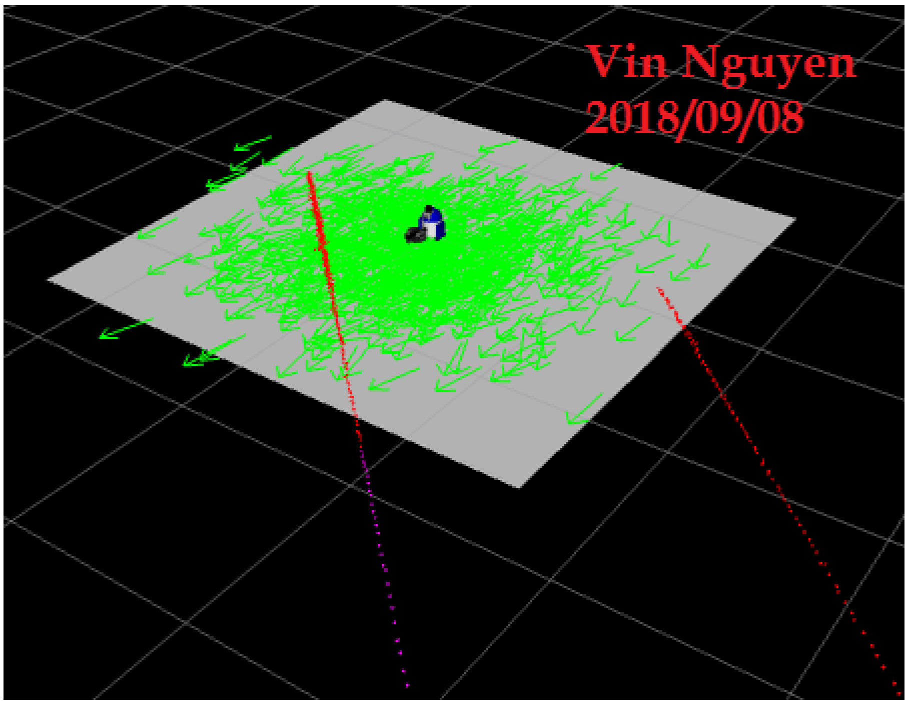


Figure 6: My_Bot shown at the start position

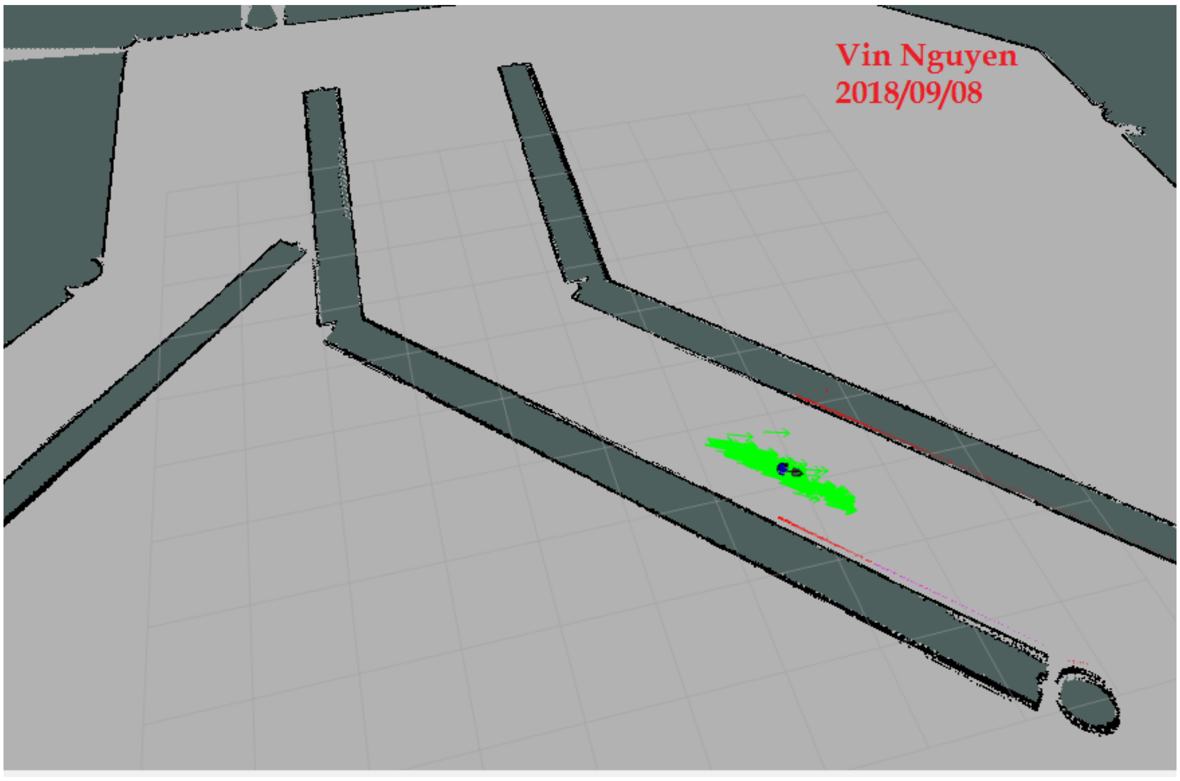
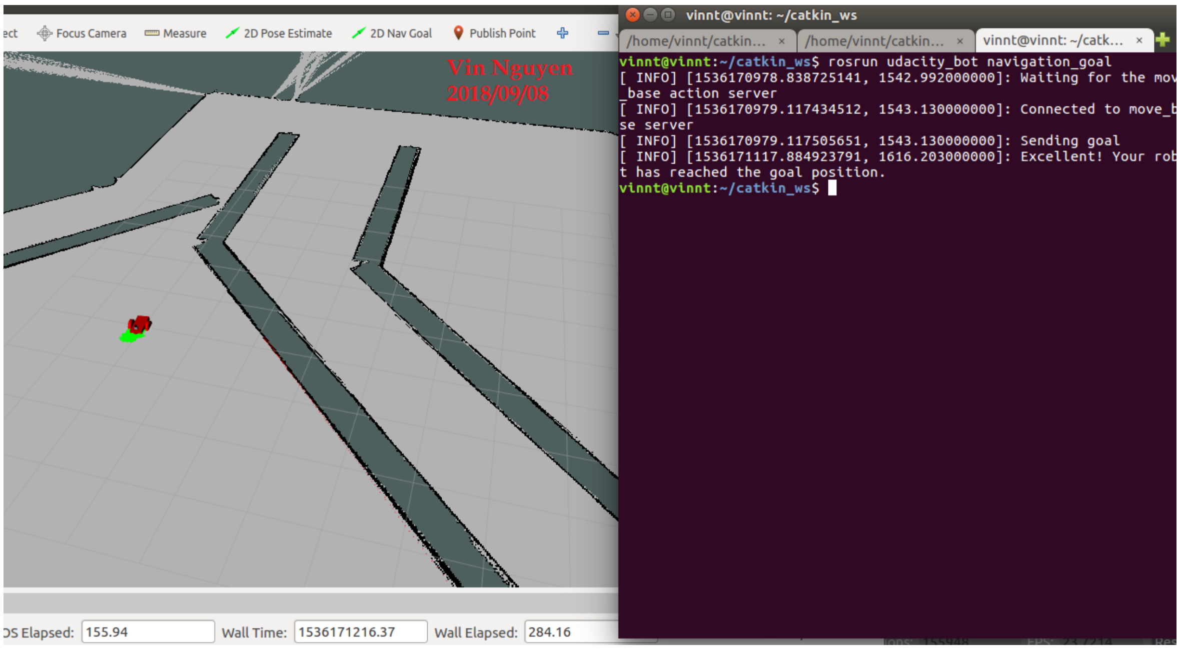


Figure 7: My_Bot Shown moving along the navigation path

4. Results

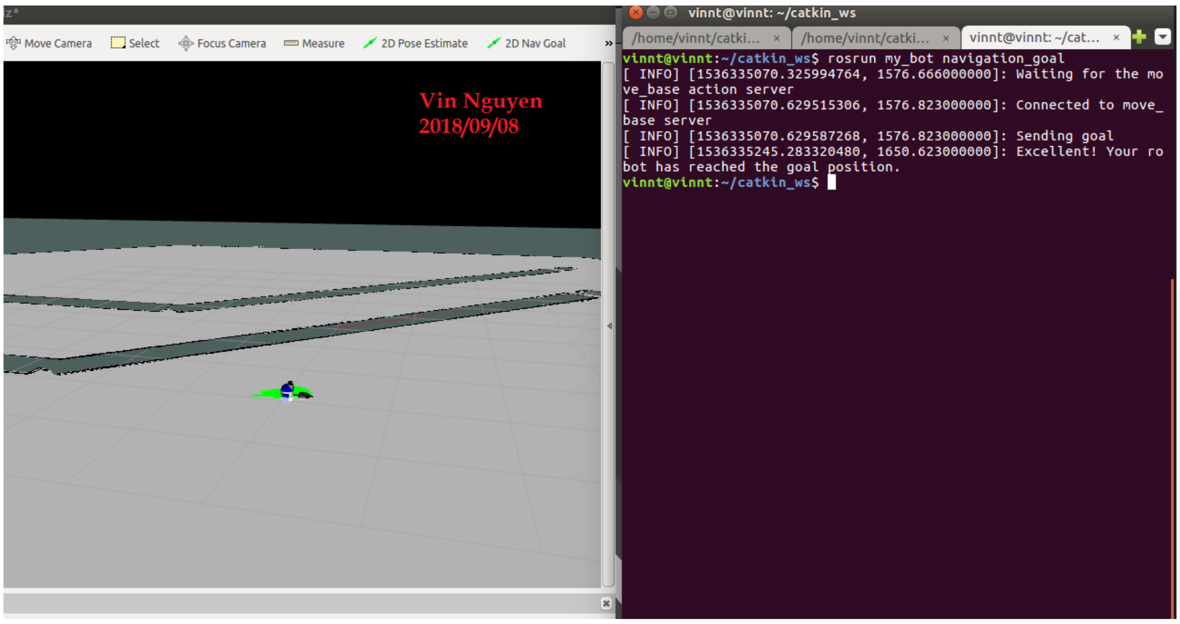
Both robots reached the goal after tweaking parameters. Both robots' particles were uniformly spread at the start of the simulation and eventually narrowed down to a small group of particles as the robots proceeded to the goal. At the goal, the robots' particles had a tight grouping pointing towards the orientation of the goal.

However, My_Bot moved more difficultly and speed slowly than Udacity_Bot. Because, My_Bot designed more complex. It has a gripper that is based on the PR2 gripper.



```
vinnt@vinnt:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1536170978.838725141, 1542.992000000]: Waiting for the mov
_base action server
[ INFO] [1536170979.117434512, 1543.130000000]: Connected to move_b
se server
[ INFO] [1536170979.117505651, 1543.130000000]: Sending goal
[ INFO] [1536171117.884923791, 1616.203000000]: Excellent! Your rob
t has reached the goal position.
vinnt@vinnt:~/catkin_ws$
```

Figure 8: Udacity_Bot shown at the goal position



```
vinnt@vinnt:~/catkin_ws$ rosrun my_bot navigation_goal
[ INFO] [1536335070.325994764, 1576.666000000]: Waiting for the mo
ve_base action server
[ INFO] [1536335070.629515306, 1576.823000000]: Connected to move_
base server
[ INFO] [1536335070.629587268, 1576.823000000]: Sending goal
[ INFO] [1536335245.283320480, 1650.623000000]: Excellent! Your ro
bot has reached the goal position.
vinnt@vinnt:~/catkin_ws$
```

Figure 9: My_Bot shown at the goal position

5. Model Configuration

5.1 Udacity_Bot

5.1.1 Design

The Robot's design considerations include the size of the robot and the layout of sensors as detailed below:

- ✓ **Maps**: The ClearPath [3] jackal race.yaml and jackal race.pgm packages were used to create the maps.
- ✓ **Meshes**: The Laser Scanner simulated is a Hokuyo scanner [5]. The hokuyo.dae mesh was used to render it.
- ✓ **Launch**: Three launch scripts are used:
  - **robot description.launch**: defines the joint state publisher which sends fake joint values, robot state publisher which sends robot states to tf, and robot description which sends the URDF to the parameter server.
  - **amcl.launch:** launches the map server, the odometry frame, the AMCL localization server, the move base server, and the trajectory planner server.
  - **udacity world.launch:** includes the robot description launch file, the gazebo jackal race world, the AMCL localization server, spawns the robot in gazebo world, and launches RViz.
- ✓ **Worlds:** Two worlds are defined:
  - **jackal race world:** defines the maze.
  - **udacity world:** defines the ground plane, light source and world camera.
- ✓ **URDF:** The URDF files defines the shape of the robot. Two files define the Gazebo view and the basic robot description:
  - **udacity bot.gazebo:** provides definitions of the differential drive controller, the camera and camera controller, the Hokuyo laser scanner and controller for Gazebo.
  - **udacity bot.xacro:** provides the robot shape description in macro format.

5.1.2 Parameters

The parameter choices used for the robots were based on the ROS Navigation Tuning Guide [7]. The first parameters that were tweaked were in the amcl.launch file. Here the initial pose parameters for x, y, and z were all set to 0, and the min and max particles parameters were set to a value of 5000, a number too large for the simulation environment on the simulation platform. It was modified down to a minimum of 100 and maximum of 500 particles. Not only reduced unnecessary particles in the simulation, but it also freed up computational resources. The next parameters modified, and the most important in terms of localization accuracy, were the odom_alpha parameters. These parameters specify the expected noise in odometry's rotation estimate. Odom_alphas 1-4 are the only parameters tuned since the robot uses a differential drive system. These were initially all set to their default values, 0.2, and after every iteration each odom_alpha parameter was incremented by 0.1. However, it was observed that as the odom_alphas increased, the particles seemed to spread more as the robot moved instead of coming closer together.

After noticing this behavior, the odom_alphas were decremented first by 0.1 from their default values. Several iterations later, the particles weren’t accurate enough. After consulting the tuning guide, it seemed that the step size of 0.1 used to increment or decrement was too large and the default value of 0.2. The values needed to be significantly smaller. This suggestion led to the final values of the odom_alphas 1-4 as follows:

- ❖ odom_alpha1 = 0.0005
- ❖ odom_alpha2 = 0.001
- ❖ odom_alpha3 = 0.0005
- ❖ odom_alpha4 = 0.0005

Setting the parameters to these resulted to tighter packed particles as the robot moved towards the goal. To reduce computational load, the transform_tolerance parameter was adjusted in both amcl.launch and costmap_common_params.yaml files. This parameter helps select the durability of the transform being published for localization reasons and should account for any lag in the system being used. Following the tuning guide, final value of the transform_tolerance led to 0.3 to meet the system needs. Furthermore, the update and publish frequency and the map size for both local and global costmaps were reduced drastically to release computational resources. The resolution was also increased from its default value 0.05 to 0.1, this was a good enough value to reduce computational load while keeping the preferred accuracy. To avoid hitting the barriers in the simulation, the inflation_radius parameter in costmap_common_params.yaml was set to 0.5. This ensures that the robot will keep a distance of half a meter away from the barrier to prevent the robot colliding with the barrier. Additionally, the pdist parameter in base local planner was also set to 3.5 so that the robot will follow the global path, but it'll give the robot enough freedom to recover from any mistakes.

5.2 My_bot

5.2.1 Design

The design of the robot was depend on web page [2]. This platform worked for what was needed so that it was possible to reposition the sensors to somewhere higher from the udacity_bot. To stabilize the robot, 4 wheels were used. All wheels are actuated to give the robot better handling when turning corners. Lastly, the inertial mass of the robot was much heavier compared to the udacity_bot due to being more upright and having a higher center of gravity. This will safeguard the robot from falling over when accelerating or decelerating.

✓ **Maps**: The ClearPath *jackal race.yaml* and *jackal race.pgm* packages were used to create the maps, identical to UdacityBot [3].

✓ **Meshes**: The Laser Scanner simulated is a Hokuyo scanner. The *hokuyo.dae* mesh was used to render it. In addition, MyBot has a gripper that is based on the PR2 gripper as defined by Willow Garage. The files *l finger.dae*, *l finger.tif*, *l fing tip.dae* and *l finger tip.tif* from Willow Garage [1] are used.

✓ **Launch**: The same three launch scripts are used as UdacityBot.

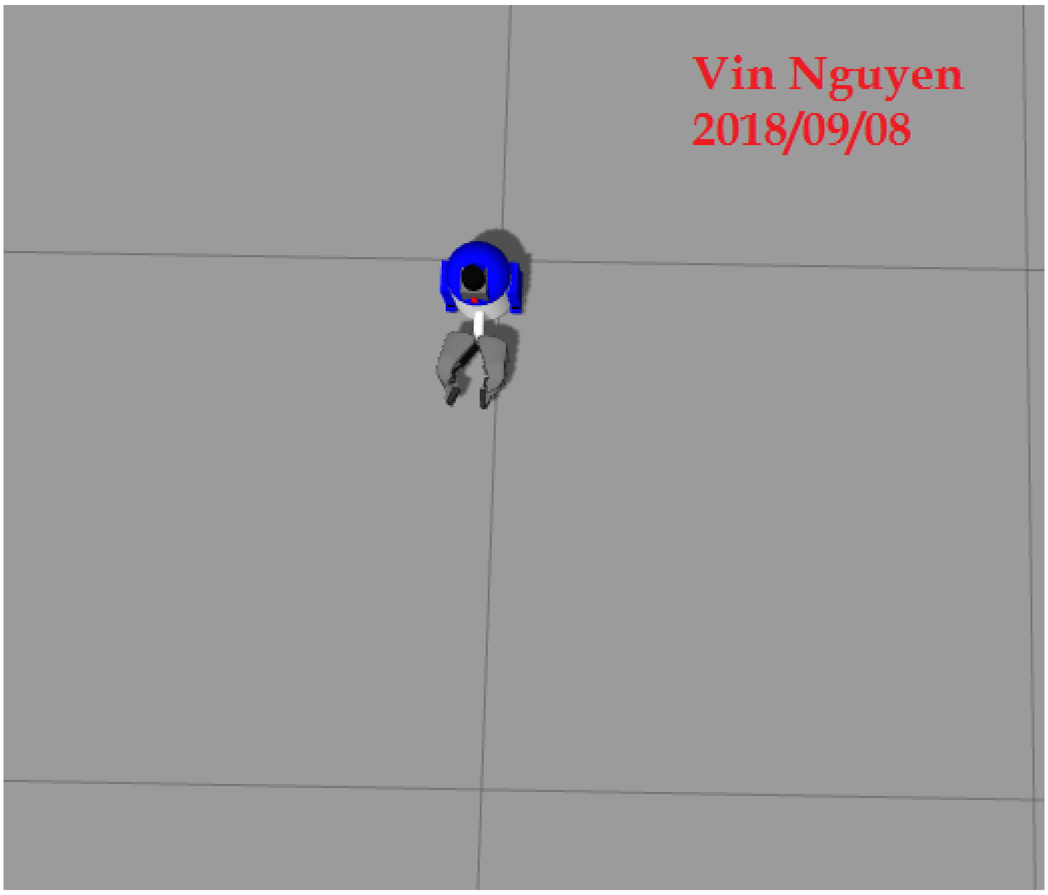


Figure 10 : My_Bot

5.2.2 Parameters

The parameter choices used for the My_Bot were based on the ROS Navigation Tuning Guide [7]. Almost, the same parameters of Udacity_Bot. Only the min and max particles parameters were set 100 and 800 to robot moves more well.

6. Discussion

Using the AMCL algorithm, both robots were able to accomplish the goal of this project successfully. The particles initially were spread out around the robots indicating uncertainty, but as the robot navigated through the maze, the algorithm became more confident in locating the robot's pose thus the particles began to group closer and closer. When the robot reached the goal, the particles were tightly grouped under the robots and on the goal's location, pointing in the goal's direction.

The Laser Scanner placement modification also had no impact on performance in the given environment as the scanner was still lower than the height of the barriers.

MCL/AMCL would work well in any industry domain where clear barriers guide the path of the robot. The ground also needs to be flat and clear of obstacles particularly in the case of DougBot where the Laser Scanner is placed on top of the tower.

7. Future work

There may be little to no adjustments needed to further enhance the accuracy of the localization software wise in the current system. In terms of hardware however, a more powerful CPU or an addition of a GPU would be helpful to ease the localization and navigation computations.

The next steps to this project is to implement the software on a physical robot using the Jetson TX2. This will be more involving since it now involves hardware, software, and physical properties, such as friction and gravity, which will all be taken into account in designing a new robot or modifying the existing design and tweaking configurations.

8. References

[1] WillowGarage, “Willow garage home page.” https://www.willowgarage.com, 2018.

[2] ROS.ORG, “Building a visual robot model with urdf from scratch.” http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch, 2018.

[3] ClearPathRobotics, “Clearpath robotics home page.” https://www.clearpathrobotics.com, 2018.

[4] WillowGarage, “Willow garage nav stack image.” http://wiki.ros.org/navigation/Tutorials/RobotSetup, 2018.

[5] Hokuyo, “Hokuyo laser scanner home page.”

https://www.hokuyo-aut.jp, 2018.

[6] ROS.ORG, “Ros navigation tuning guide.”

http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide, 2018.

[7] K. Zheng, “Ros navigation tuning guide.”

http://kaiyuzheng.me/documents/navguide.pdf, 2018.

[8] Wikipedia, “Holonomic constraint.”

https://en.wikipedia.org/wiki/Holonomic constraints, 2018.

[9] D. Teeple, “Github robo-nd localization project.”

https://github.com/douglasteeple/RobotLocalization, 2018