AMCL Project

Gene Foxwell; Udacity Robotics Software Engineer Project

ABSTRACT

In many practical scenarios, it is useful for a robot to know where it is currently located inside its environment. In this project, this problem was explored for two distinct simulated robots using an implementation of the Adaptive Monte-Carlo Localization algorithm (amcl) and the ROS navigation stack. Parameters where tuned for each robot to provide good localization results and each robot was then tested using a predefined goal point on the jackal-world map built by ClearPath Robotics. These Robots where then evaluated based on the quality of the localization results they had obtained by the time they had reached their goal.

INTRODUCTION

This project looks at addressing the problem of determining where robot's location is on a given map, and accurately tracking the robot's location as it moves around.

To explore this problem, we created two robots' using the sdf format provided by Gazebo. The first such robot, the so-called Udacity bot, was built from specifications provided by the Udacity course materials. Second was the student designed robot named "Marvin". This robot was designed to be smaller, faster, and more agile than its Udacity provided cousin. Figure 1 and Figure 2 show images of the Marvin and Udacity bot's respectively inside a blank Gazebo world.

Each robot was tested in the an Udacity provided Gazebo world – the "Jackal Race" world. A top down view of this world as seen in the Gazebo simulator can be seen in Figure 3 with the Marvin bot waiting at the starting location. Robots were required to transverse the world and arrive at the location highlighted in Figure 3 while maintaining a good approximation of their current location.

Localization of the robots was achieved using the amcl package obtained from ROS ecosystem. Each robot was tuned with its own set of parameters for the amcl package to allow it to localize itself effectively.

Navigation of the robots was done using the ROS navigation stack's "move_base" package. This package makes of both local and global "costmaps" to calculate paths for the robot to attempt to follow as well as publishing odometry information to the robots themselves. As with the amcl package, the navigation stack had to be fine tuned for each robot to get optimal results.

Once the parameters had been optimized, each robot's navigational and localization abilities was tested by having them navigate from the starting position to a new goal point provided by the Udacity's "navigation_goal" utility. This tool set a navigation goal to the move_base package and monitored the system until the robot reached the goal. Once the goal was reached, or it became clear the robot could never reach the goal, an appropriate message was outputted to the terminal and the utility self-terminated.

Performance of the robots was tracked using Rviz. Rviz was used to track the robot's current location and output the current set of particles generated in the form of a "RobotPose" array. An example of this Pose Array in Rviz is provided in Figure 4. Our performance goal was to have for the pose array at the end of the robots' path to be tightly packed around the robots final location, with all poses pointing in the same direction as the robot itself.

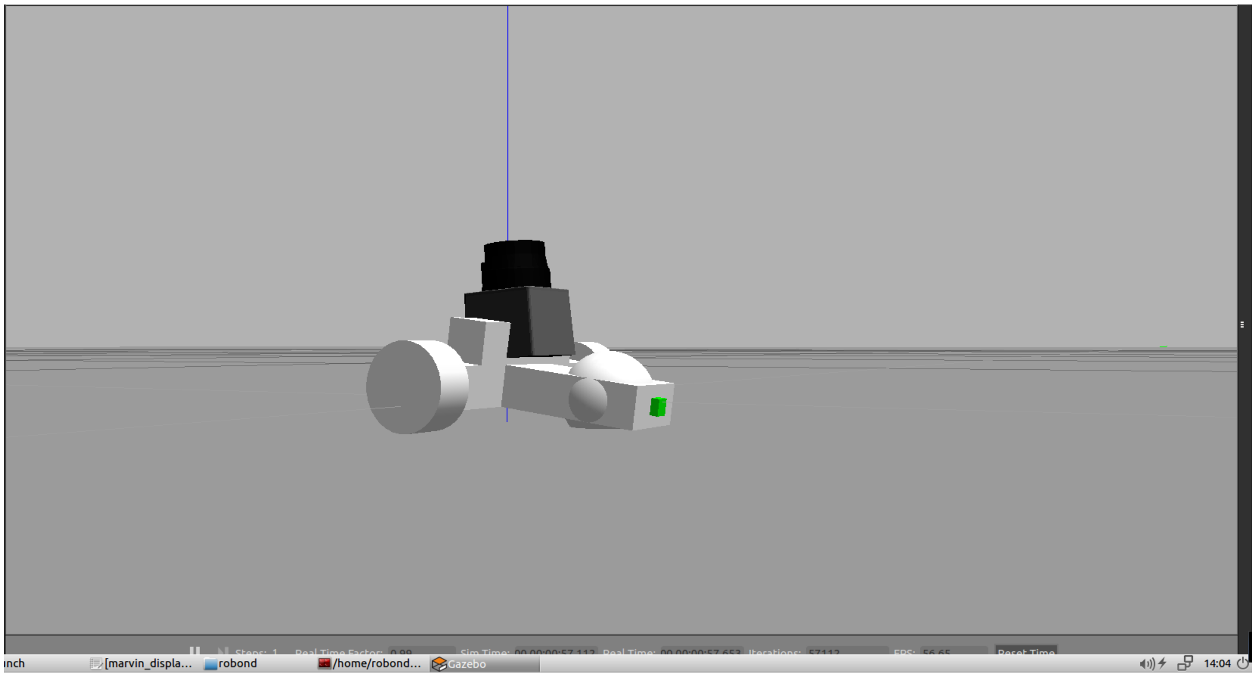


Figure 1 Marvin Bot

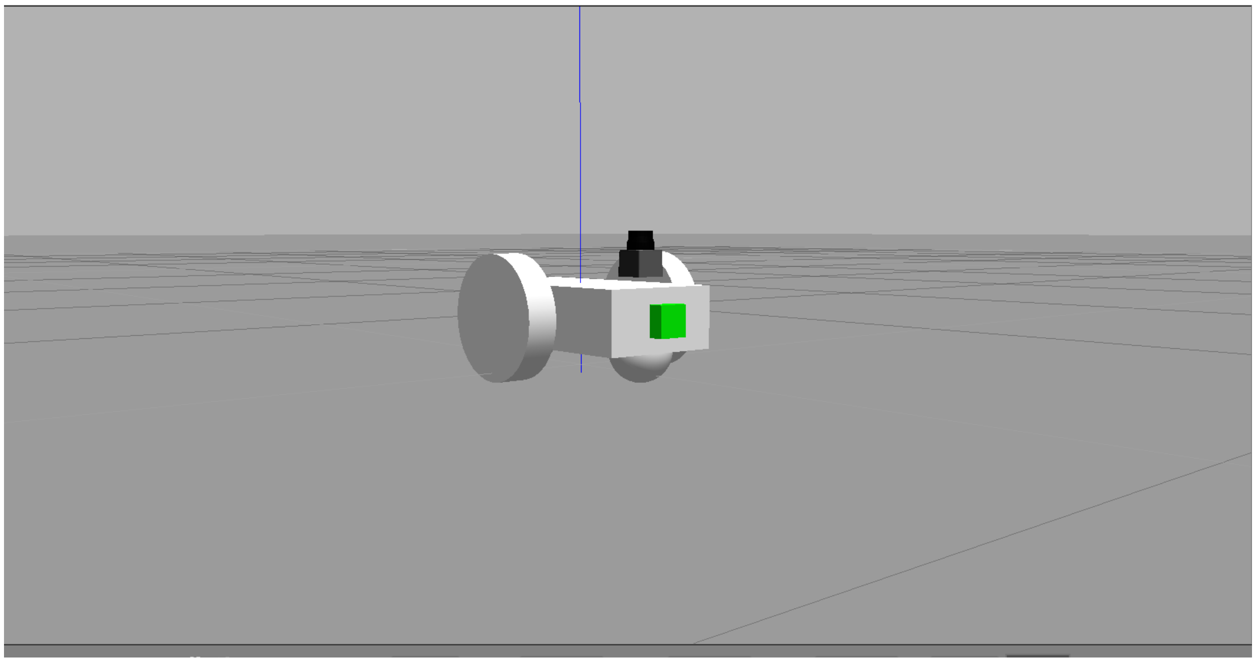


Figure 2 Udacity Bot

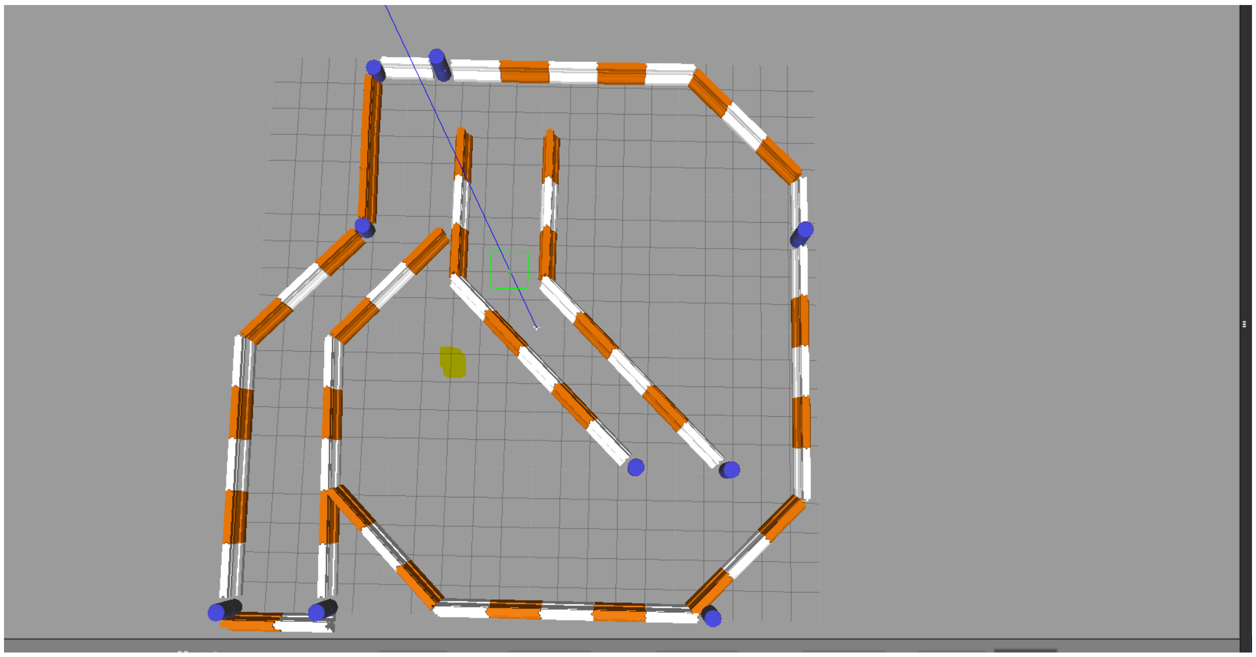
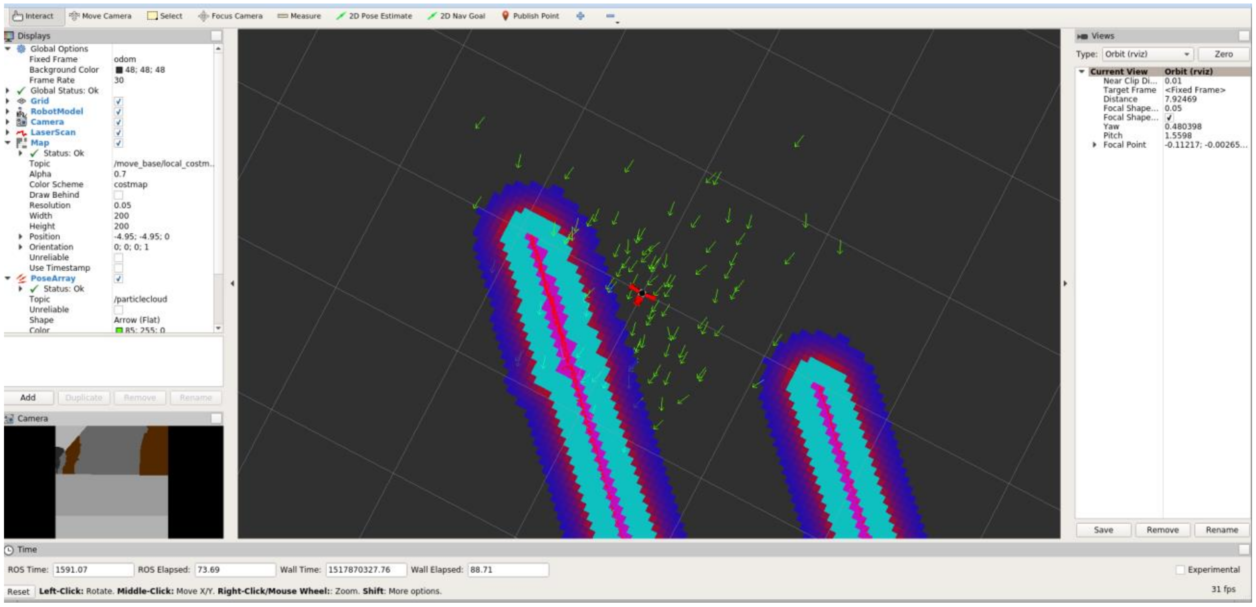


Figure 3 Jackal Race World



```
Global Options
  Fixed Frame          odom
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
Grid                   ✓
RobotModel             ✓
Camera                 ✓
LaserScan              ✓
Map                    ✓
  Status: Ok
  Topic                /move_base/local_costm...
  Alpha                0.7
  Color Scheme         costmap
  Draw Behind
  Resolution           0.05
  Width                200
  Height               200
  Position             -4.95; -4.95; 0
  Orientation          0; 0; 0; 1
  Unreliable
  Use Timestamp
PoseArray              ✓
  Status: Ok
  Topic                /particlecloud
  Unreliable
  Shape                Arrow (Flat)
  Color                85; 255; 0
```

```
Type: Orbit (rviz)
Current View           Orbit (rviz)
  Near Clip Di...      0.01
  Target Frame         <Fixed Frame>
  Distance             7.92469
  Focal Shape...       0.05
  Focal Shape...       ✓
  Yaw                  0.480398
  Pitch                1.5598
  Focal Point          -0.11217; -0.00265...
```

```
ROS Time: 1591.07   ROS Elapsed: 73.69   Wall Time: 1517870327.76   Wall Elapsed: 88.71
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
```

Figure 4 Pose Array

BACKGROUND

Robots for this this project where simulated using the Gazebo simulation environment provided with ROS. Each Robot was built by using the Gazebo sdf format from primitive shapes.

Specifications for the "Udacity Bot" where provided by the Udacity course materials. As can be seen in Figure 2 the Udacity bot is equipped with two relatively large wheels, a laser range scanner, and a camera. These rest on a rectangular platform supported by two caster wheels. To allow the robot to move around freely the friction settings for the caster wheels have been set to zero. The robot controls its wheels using a differential drive package provided by ROS.

Marvin, as can be seen in Figure 1, was designed after experimenting with the Udacity bot. Marvin was designed to improve on the Udacity model in terms of speed and agility. Marvin has two small wheels at the back of the robot for propulsion. Mounted on a beam between the two wheels is a laser range scanner, which has been slightly rotated to allow the sensor to map more of the world around it without the beams intersecting with the worlds floor. During initial design of the Marvin it was discovered the robot would "tilt" upwards pushing its front edge into the air. To get around this issue a counterweight was placed on the front platform of the robot to keep the robot stable as it moved around. To avoid having the counterweight interfere with Marvin's navigation system this counterweight was configure in a similar manner to the Udacity Bot's caster wheels. Finally, a camera was placed on the forward section of the robot's front platform. While this camera does not feed into the AMCL package, it could potentially be used as a detector for random navigational hazards spread throughout the road ahead.

Both the Udacity bot and Marvin have been combined into a single ROS package named "udacity_bot". This "udacity_bot" package contains several launch files configured to make experimentation easier. Launch files and their respective functions are summarized in the table below:

Launch File	**Description**
udacity_world.launch	This loads the robot description file for the udacity bot and the jackal_race world (provided in the course material) inside the Gazebo physics simulator. In addition, it loads Rviz with a suitable configuration file for visualizing the udacity bots state.
amcl.launch	This loads both the amcl ROS package and the move_base package from the ROS navigation stack. It also specifies parameters and parameter files for each package.
marvin_world.launch	Similar to the udacity_world.launch file, except it loads the relevant files for the Marvin bot.
marvin_amcl.launch	Similar to the amcl.launch file, except it loads the appropriate amcl and move_base settings for the Marvin bot.

The simulations can be started by running the appropriate launch file pairs (udacity_world.launch + amcl.launch / mavin_world.launch + marvin_amcl.launch). Once both launch files have been run, the users should see a similar display to that seen in Figure 4.

Our primary goal in this project was to solve the localization problem. In this problem we are provided with a map of the environment and we want to find and/or track the robot's location with the respect to the provided map over time. This problem is important for applications where a robot needs to navigate around its environment safely. Imagine for the moment a hospital robot that needed to bring important medical supplies to a surgery room. If it cannot ascertain its own location, how could it plan a path to the hospital room? Possibly worse, what if its believed location is incorrect for some reason (perhaps some nefarious criminal has moved it) – it may plan a path that leads to the wrong room altogether!

Two common methods of solving this issue where discussed in the Udacity course materials. Kalman Filters, and Monte-Carlo Localization. We choose to use a version of Monte-Carlo Localization for this project, however it is useful to briefly discuss the two approaches for the sake of understanding why this choice was made.

Kalman filters represent the location of the robot as a unimodal gaussian distribution centered around the robots most probable location on the map. Kalman filters track a robots movement throughout the map in three steps. First the filter predicts the robot's new location using the odometry command provided to the robot combined with the robots motion model. As this motion tends to be noisy, this step tends to add noise to the model widening the gaussian distribution. Secondly the robot uses its measurement model to obtain a prediction of the robots location based on measurement data provided by the robots sensors. Thirdly a weighting factor called "the Kalman Gain" is calculated and used to create a weighted average between the predicted location and the measured location. This final prediction is the Kalman filters current representation for the robots location.

Monte-Carlo Localization (MCL) represents the location of the robot as set of orientated particles spread throughout the map. Each particle represents a potential location and orientation of the robot on the map. MCL begins with an initial set of guesses (or particles) for the robots location, randomly drawn from some prior distribution. When the robot moves, the particles are moved in accordance with the robots odometry input and the provided movement model. Next the measurements from the robots censors are taken into account, this is used to assign aa probability to each particle based on how likely that guess is based on the measurements. A new set of particles is then generated by sampling from the existing particle set based on those probabilities. (i.e. particles with a higher probability have a greater chance of being chosen to live until the next iteration). This process is then repeated for the robots next movement. The idea here is that over time eventually only particles that correspond with the robots actual orientation and position will remain active.

MCL has a few advantages over the use of a Kalman filter. First it is much more straightforward to implement. Kalman filters require a lot of matrix operations which can be cumbersome to perform without the use of a good library, this can be avoided in MCL. Kalman filters cannot easily represent a multi-modal distribution, which means if its equally likely that the robot is in the living room and the dining room a Kalman filter cannot represent that. An MCL however can represent nearly any distribution thanks to its use of particles as opposed to an explicit Gaussian distribution. This means if the robot could in both places, it can be represented by simply having some particles in the living room and some in the dining room. Finally, a Kalman filter requires roughly a set amount of processing power to implement. If we must represent the robot and say 52 landmarks, all those must be tracked. With MCL we can tweak the number of particles to match the available processing power, allowing (at least for some applications) a more efficient localization system.

For this project we used an ROS package to perform MCL. Specifically, we used the amcl package which uses a sampling specific sampling technique to adapt the number of particles used on each iteration based on how uncertain the robot is of its current location[1]. When the robot is uncertain of it location (generally during the beginning of the process) a large number of particles will be generated, when it is fairly certain of its location (generally the case when tracking the robots movements after having narrowed down its original location) only a handful of particles are generated. Parameters for the amcl package will be discussed in the Model Configuration section.

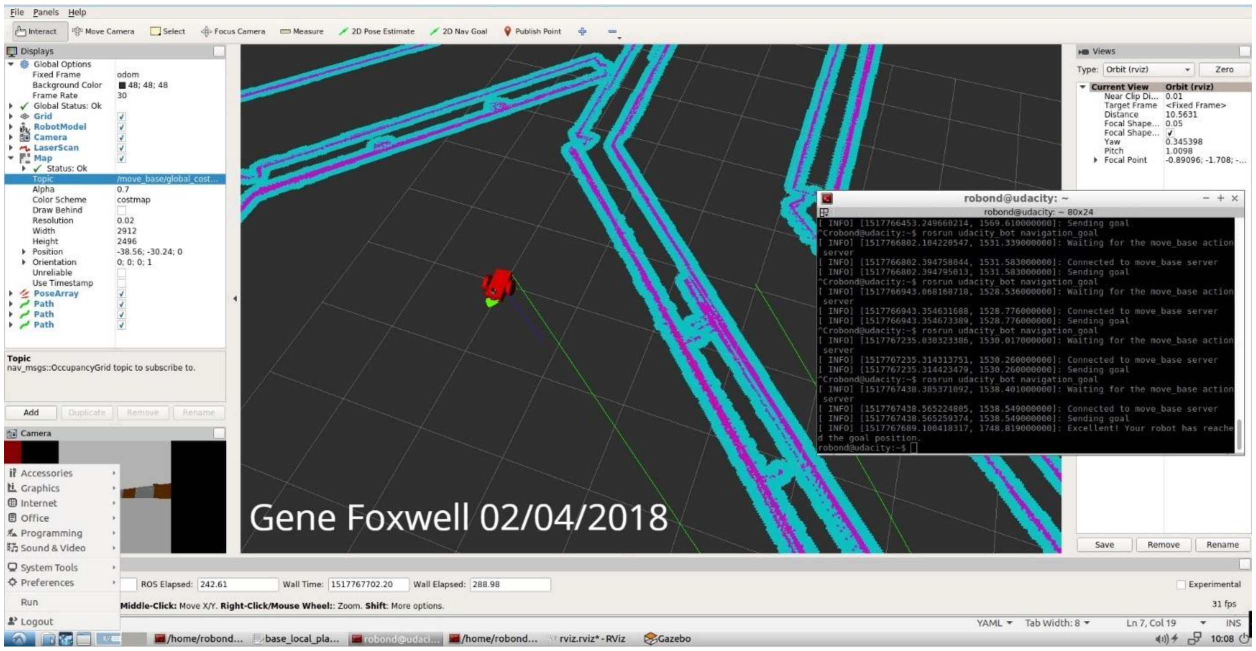
Path planning and movement of the robot was handled using the ROS Navigation stack. This stack had two main components that required configuration Costmap and the Base Local Planner. The Costmap (which was split into both a global and local system) represented information about obstacles in the world. This was used to help the planner determine which map cells the robot could drive into and which it should avoid. It also contained settings for generating higher costs for movement that brought the robot too close to an obstacle rather than directly into it. The Base Local Planner was used to computer the odometry commands required to follow the higher-level plan generated by the navigational system. As with amcl package the parameters that were configured for this will be discussed in the Model Configuration section.

[1]Adapting the Particle Size in Particle Filters though KLD-Sampling
http://www.robots.ox.ac.uk/~cvrg/hilary2005/adaptive.pdf Deiter Fox

Finally, once the parameters for the various packages had been set, the system was tested against a test case provided by Udacity. The navigation_goal utility provided a 2D navigation goal to the ROS Navigation stack which triggered the robot build and follow a path to the goal state. The robots progress was monitored via Rviz to keep track of how well it was localizing itself as it went along. Typically, both Robot where able to localize themselves quickly with results similar to that seen in Figure 7 occurring in the few seconds of operation.

RESULTS

Final results for both the Udacity Bot and the Marvin Bot can be seen in Figure 5 and 6. In each case the robot is waiting at the prescribed goal provided by the Udacity navigation_goal utility with its final pose estimates showing up underneath the robot as a set of green arrows. In both cases the remaining particles are centered tightly around the robots location with all arrows pointing the direction of the robot.



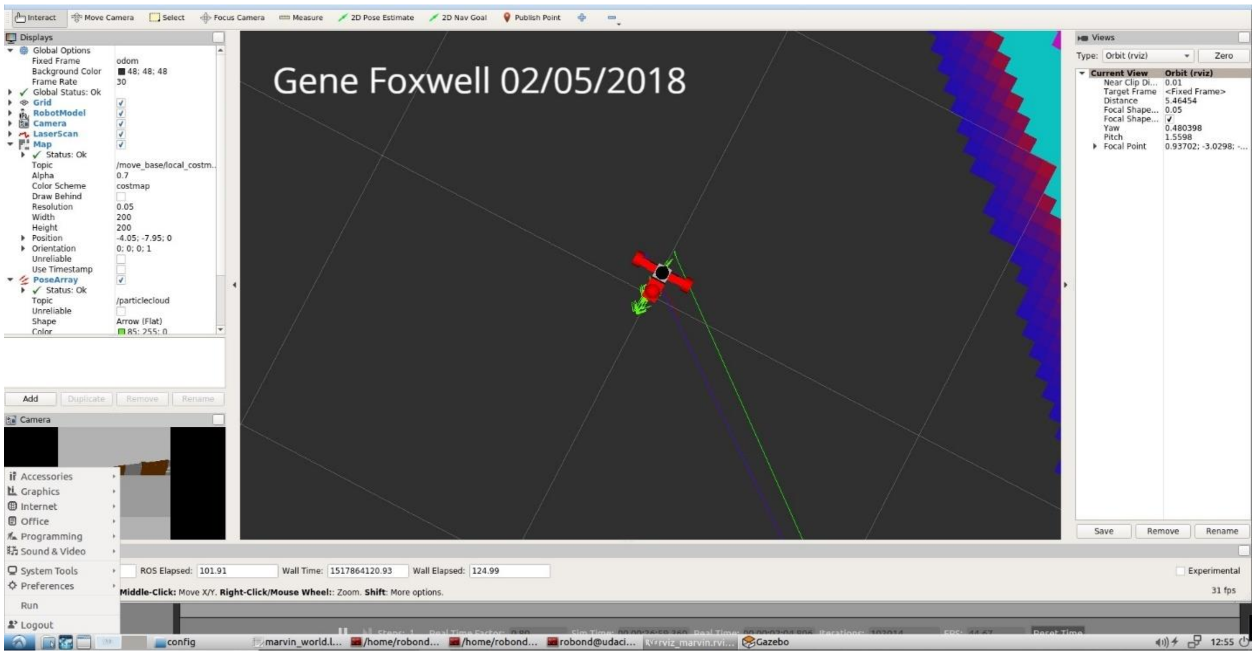
```
Global Options
  Fixed Frame          odom
  Background Color     48; 48; 48
  Frame Rate           30
  Global Status: Ok
Grid                   ✓
RobotModel             ✓
Camera                 ✓
LaserScan              ✓
Map                    ✓
  Status: Ok
  Topic                /move_base/global_cost...
  Alpha                0.7
  Color Scheme         costmap
  Draw Behind
  Resolution           0.02
  Width                2912
  Height               2496
  Position             -38.56; -30.24; 0
  Orientation          0; 0; 0; 1
  Unreliable
  Use Timestamp
PoseArray              ✓
Path                   ✓
Path                   ✓
Path                   ✓

Topic
nav_msgs::OccupancyGrid topic to subscribe to.
```

```
Type: Orbit (rviz)
Current View     Orbit (rviz)
  Near Clip Di...  0.01
  Target Frame     <Fixed Frame>
  Distance         10.5631
  Focal Shape...   0.05
  Focal Shape...   ✓
  Yaw              0.345398
  Pitch            1.0098
  Focal Point      -0.89096; -1.708; -...
```

```
robond@udacity: ~
robond@udacity: ~ 80x24
[ INFO] [1517766453.249660214, 1569.610000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1517766802.104220547, 1531.339000000]: Waiting for the move_base action
 server
[ INFO] [1517766802.394758044, 1531.583000000]: Connected to move_base server
[ INFO] [1517766802.394795013, 1531.583000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1517766943.068168718, 1528.536000000]: Waiting for the move_base action
 server
[ INFO] [1517766943.354631688, 1528.776000000]: Connected to move_base server
[ INFO] [1517766943.354673389, 1528.776000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1517767235.030323386, 1530.017000000]: Waiting for the move_base action
 server
[ INFO] [1517767235.314313751, 1530.260000000]: Connected to move_base server
[ INFO] [1517767235.314423479, 1530.260000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1517767438.385371092, 1538.401000000]: Waiting for the move_base action
 server
[ INFO] [1517767438.565224805, 1538.549000000]: Connected to move_base server
[ INFO] [1517767438.565259374, 1538.549000000]: Sending goal
[ INFO] [1517767689.100418317, 1748.819000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$
```

```
ROS Elapsed: 242.61   Wall Time: 1517767702.20   Wall Elapsed: 288.98   Experimental
Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
YAML   Tab Width: 8   Ln 7, Col 19   INS
/home/robond...   base_local_pla...   robond@udaci...   /home/robond...   rviz.rviz* - RViz   Gazebo   10:08
```

Figure 5 Udacity Bot Localization



Displays

Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
Grid	✓
RobotModel	✓
Camera	✓
LaserScan	✓
Map	✓
✓ Status: Ok	
Topic	/move_base/local_costm...
Alpha	0.7
Color Scheme	costmap
Draw Behind	☐
Resolution	0.05
Width	200
Height	200
Position	-4.05; -7.95; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐
PoseArray	✓
✓ Status: Ok	
Topic	/particlecloud
Unreliable	☐
Shape	Arrow (Flat)
Color	🟩 85; 255; 0

Views

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip Di...	0.01
Target Frame	<Fixed Frame>
Distance	5.46454
Focal Shape...	0.05
Focal Shape...	✓
Yaw	0.480398
Pitch	1.5598
Focal Point	0.93702; -3.0298; -...

ROS Elapsed: 101.91 Wall Time: 1517864120.93 Wall Elapsed: 124.99 Experimental

Middle-Click: Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

Figure 6 MarvinBOT Localization

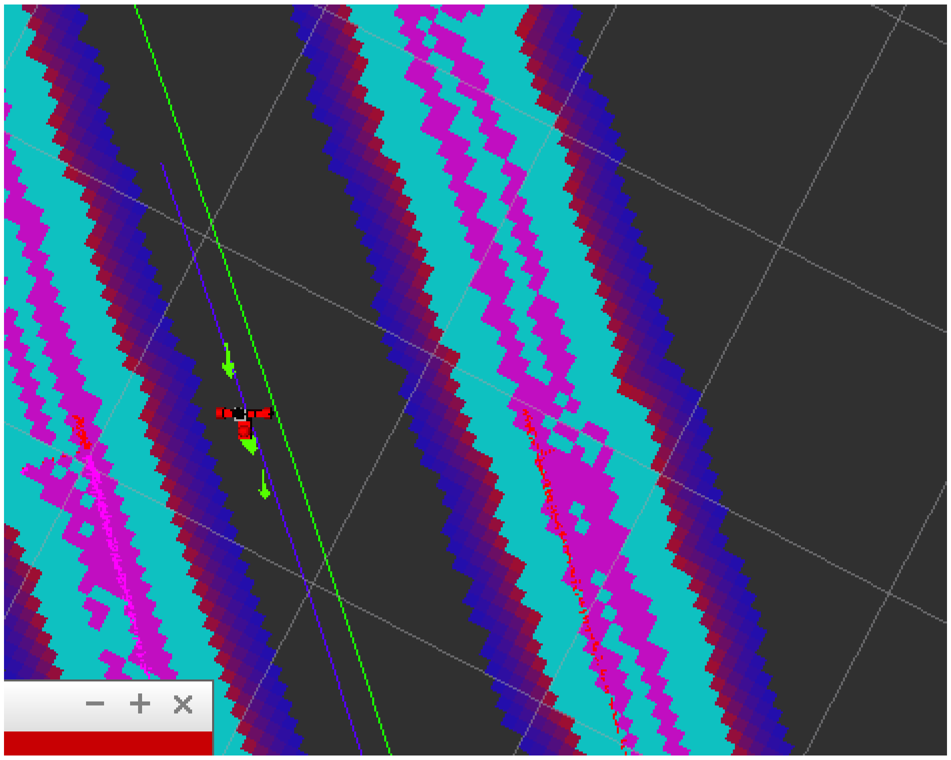


Figure 7 Localization in Progress

MODEL CONFIGURATION

For this section only, settings that have been directly considered will be discussed. As this system has hundreds of possible parameters it did not make sense to make mention of every parameter that was simply left as default.

UDACITY BOT CONFIGURATION

Udacity Bot AMCL Settings

Parameter	Value	Notes
min_particles	10	See max_particles
max_particles	100	This parameter was set to smallest value that seemed to consistently work. As more particles require more processing power it was desired to keep the maximum and minimums reasonable small.
transform_tolerance	0.25	This was set to the smallest value that did not result in causing issues with the simulation itself. Smaller values than this resulted in warnings regarding the map, or robot location along with "shaking" behavior in Rviz.
update_min_d	0.05	Minimum distance the robot had to travel before generating a new set of particles. The approach taken here was to accept fewer particles but to generate them far more often than the default value of 0.2 meters.
update_min_a	0.1	Minimum change in angle (in radians) the robot must make before generating a new set of particles. As with update_min_d this was deliberately set much smaller than the default pi/6 with the idea that generating fewer particles at a higher rate would result in better localization.

Udacity Bot Base Local Planner Settings

Parameter	Value	Notes
sim_time	2.0	Amount of time the simulation should plan ahead for its odometry. This was set by trail and error moving up and down in increments of 0.5.
meter_scoring	True	
xy_goal_tolerance	0.05	Tolerances where set by trail and error. Much smaller than this and they resulted in the robot occasionally reach its goal, but just spinning around it or driving over it trying to get closer to the exact location than was possible.
yaw_goal_tolerance	0.1	
pdist_scale	2.5	A weight for how much play should be allowed between the controller and provided path. A setting of 2.5 (directly between the min of 0 and max of 5) seemed to allow the robot enough play to make it around the final turn during the process. Setting it at 5.0 occasionally caused the robot get stuck when attempting to turn the final bend.
controller_frequency	10	This was reduced from a default of 20 to help mitigate an issue where the navigation stack would complain it could not update the odometry at the desired rate. This warning can still occur but is much rarer than when left at the default.

Udacity Bot Common Costmap Parameters

Parameter	Value	Notes
obstacle_range	9.0	The maximum distance away from the robot where an obstacle registered by the laser range finder should be

		registered as an obstacle on the cost map. This value was chosen based on the specifications of the laser range finder provided.
raytrace_range	10.0	
transform_tolerance	0.2	
robot_radius	0.2	While the robot is not technically circular, it was easier to model if we assume that it is. This parameter tells the planner to treat the robot as being a circle with radius 0.2m.
Inflation_radius	0.2	This sets the maximum distance from the obstacles at which a cost should be incurred. For this robot it was sufficient to set this to be the same as the robot itself. Setting this too large seemed to cause problems with the robots ability to efficiently transverse the map.

Udacity Bot Global Cost Map Settings

Parameter	**Value**	**Notes**
update_frequency	5.0	How often the global costmap should be updated.
publish_frequency	2.0	How often global costmap data was published.

Udacity Bot Local Cost Map Settings

Parameter	**Value**	**Notes**
update_frequency	5.0	
publish_frequency	2.0	
Width	10.0	These where reduced as the costmap was spitting out warnings regarding being unable to update itself in time. A smaller local costmap mitigated this issue greatly.
height	10.0	

MARVIN BOT CONFIGURATION

Marvin Bot AMCL Settings

Parameter	Value	Notes
min_particles	10	
max_particles	100	
transform_tolerance	0.25	
update_min_d	0.05	
update_min_a	0.1	

Marvin Bot Base Local Planner Settings

Parameter	Value	Notes
sim_time	0.5	This was set to a smaller value than the Udacity Bot to account for the smaller size and faster speed of this model. Setting this value higher causes the robot to move very eractically and in some cases to simply drive around in circles.
meter_scoring	True	
xy_goal_tolerance	0.2	
yaw_goal_tolerance	0.1	
pdist_scale	5.0	Set to highest value to encourage the robot to stick heavily to the pre-planned path. As this robots agility was much greater than the Udacity Bots this robot was set to follow the plan as close as possible.
controller_frequncy	10	
max_vel_x	0.45	Limitations on the robots speed and acceleration had to be set to avoid the robot from getting out of control. Without these limitations the robot would occasionally drive so far it would lift its front end off the ground, causing it to bounce around – greatly effecting the sensor readings and making localization nearly impossible.
min_vel_x	0.1	
max_vel_theta	1.0	
min_in_place_vel_theta	0.4	
acc_lim_theta	3.2	

acc_lim_x	1.5	
acc_lim_y	1.5	

Marvin Bot Common Costmap Parameters

Parameter	**Value**	**Notes**
obstacle_range	8.0	
raytrace_range	9.0	
transform_tolerance	0.25	
robot_radius	0.18	This was adjusted to consider Marvin Bot's smaller size.
inflation_radius	0.4	A larger inflation rate was used here to help the robot get around the lower turn of the planned path without getting stuck.

Marvin Bot Local Costmap Parameters

Parameter	**Value**	**Notes**
update_frequency	5.0	
publish_frequency	2.0	
width	10	
height	10	
resolution	0.05	As this is a smaller robot than the Udacity Bot a smaller resolution for the local costmap was tried.

Marvin Bot Global Costmap Parameters

Parameter	**Value**	**Notes**
update_frequency	5.0	
publish_frequency	2.0	
width	40	
height	40	

DISCUSSION

Both the Marvin Bot and Udacity Bot were able to meet their performance goals. Admittedly the Udacity Bot was able to get better results than the Marvin Bot. Part of the reason for this may be that the Marvin Bot can be a bit unstable – especially at higher speeds. This problem tends to manifest itself during the robot's initial acceleration in which occasionally the front end will lift slightly before bouncing back on the ground. This causes the laser scanners POV to change dramatically introducing a short lived systematic error in the robots measurements.

One problem to consider with any localization system is the so-called "kidnapped robot" problem. In this scenario someone has teleported the robot from where it thought its original location was to some other random place on the map. Solving the kidnapped robot problem with a MCL based system requires the introduction of random particles to the system – otherwise the only poses that will remain after the robot has functioned for sufficient length of time will be those corresponding with the last know location. Fortunately, the amcl module provides for this with its use of KLD sampling. For this sampling method we get a wider range of particles if the robot becomes more uncertain about its location. In the kidnapped robot problem this would mean that very likely the measurements at the new location would cause the robot to become very uncertain of its current location generating more and more particles (up to the set maximum) until it had enough particles to sufficiently relocalize itself.

That said, while the amcl module may be able to handle the kidnapped robot problem, handling this issue with the current settings of the Udacity Bot and the Marvin Bot would be difficult. These robots have had their minimum and maximum particles setting set to allow fast computation with emphasis on the tracking problem. They may not be producing enough particles at these settings to robustly handle having been randomly transported to another location on the map.

Localization systems such as this could have many uses in an industrial setting. For many manufacturing and some commercial environments a robust map of the environment is already available. A robot equipped with such a map could be used to dynamically move parts, equipment, and other merchandise dynamically throughout the environment to where they are needed most. Such a robot could also be used to ferry hospital supplies through medical centers, possibly reducing the workload of the nurses. A robot that can localize itself in your home could be used to help take care of senior citizens by bringing them their food, medicine etc. Anywhere a map exists, and some form of dynamic behavior is wanted could easily benefit from a robot equipped with a localization system like the amcl shown here.

FUTURE WORK

To keep processing time under control, it was decided to keep a low maximum for the number of particles to generate on each iteration. As a result, these robots are likely not robust to failure states such as the kidnapped robot problem. One direction then for future work would be to address this issue directly – programmatically kidnapping the robot and tweaking the parameters further until a new set is found that can be shown to be robust to kidnapping without drastically increasing the processing time for the individual MCL iterations.

Marvin bot could also stand for a few improvements. First it might be useful to adjust the weight of the front platform a little further to see if the issue where the robot occasionally bounces can be further mitigated (or perhaps removed entirely) without reducing the robots performance. We could also look into integrating the onboard camera to help detect cases where the sensors may be bouncing around and either discard such sensor readings entirely or allow for some form of transform that could feed into the measurement model to take into account unexpected changes in the robots spatial configuration. An alternative to trying to infer changes in the robots spatial orientation could be addition an IMU sensor.

Finally, Marvin isn't exactly an all-terrain robot. Its small size and slight instability means that any unexpected litter on the floor could potentially cause the robot to lose track of itself. It may be worthwhile to investigate changing it differential drive system from simply being two circular wheels to something more track like. Such a change should increase the robots stability and allow for a higher maximum speed – allowing it to complete the course faster. (Not that speed was a goal of this project of course).