Where Am I?

Project for Udacity's Robotics Software Engineer Program

Leroy Friesenhahn

***Abstract*—The "Where am I" projects explore the problem associated with all mobile robots. Where am I? Where do I want to go? How do I get there? This project explores two different methods of determining the robot's location. Using ROS, Gazebo, and Rviz, a 3D robot is constructed to evaluate different localization methods used to navigate from start to end. Using software to simulate a robot and its capabilities reduces the cost and time of building a physical robot.**

Beginning with a specified two wheel robot, the project evaluates different parameters to determine the most successful combination needed to perform the robot's navigational task. A camera and laser sensor is used to obtain data on the robot's environment (distance from objects, etc.). The goal is to navigate from an initial point, thru a maze and stop at the predetermined end point.

A second robot is designed and tested in the same manner. Success is achieved when both robots are able to complete their navigational tasks.

INTRODUCTION

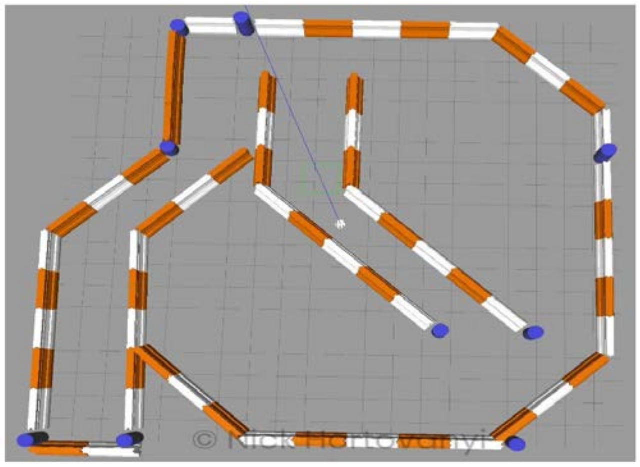
THE problem with mobile robots is the need to know its location. Sensors provide information on the robot's environment but are plaque with noise and mechanical inaccuracies.

Noise and mechanical inaccuracies is a problem with all sensors. Acknowledge the misleading information coming from the sensor and reducing its impact on the data is the main goal of this project.

Using the power of ROS, Rviz and Gazebo, a 3D model of a robot (one supplied robot and one custom robot), is developed to explore the pros and cons for the different localization filters. The robot model is used to simulate a mobile robot with two sensors (camera and laser sensor). This information is used to determine the robot position.
To prove the effectiveness of localization filters, a robot is created and given a field to navigate.

The task consist of navigating from a preselected starting point, thru the maze (the map is given) and arrive at the designated end point. Although many different localization filters are available, the Advanced Monte Carlo Localization (AMCL) is selected.

The task is complete when the robot(s) (supplied and custom) performs its navigation task and arrives at the endpoint with a minimum particle cloud(high probability the robot is at the location it has determined to be.



Simulated map provided by Clearpath Robotics

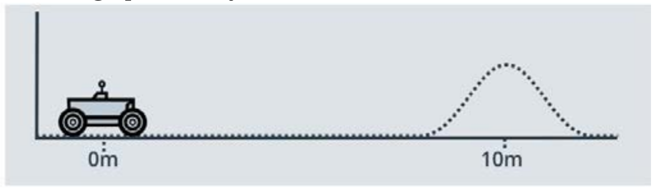
The selection of the best parameters for the task provides the challenge for this project.

BACKGROUND/FORMATION

How many times have you asked the question? Where am I? How do I get to the restaurant? Which roads, sidewalks etc. are closed due to construction or an accident? For humans it is a way of life. For mobile robots it is known as localization.

Robot localization can be accomplished using different techniques. In this lesson, Kalman, Extended Kalman, Monte Carlo, and Adaptive Monte Carlo Localization (AMCL) filters were discussed.

Kalman and Extended Kalman filters are used in linear systems. The new location can be represented as a gaussian function. Based on the initial location and the movement of the robot, the new location of the robot can be determined with a high probability.

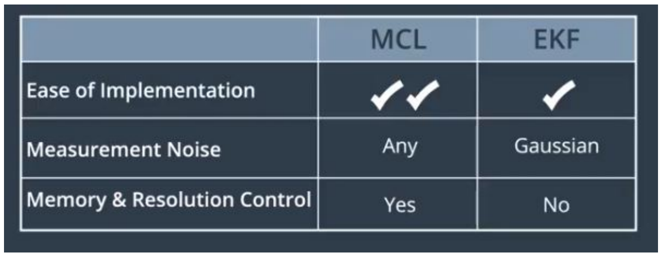


Probability of the robot's new location

Kalman filters starts with an initial state estimate. The Kalman filter performs a measurement update using the sensor data followed by a state prediction (location).

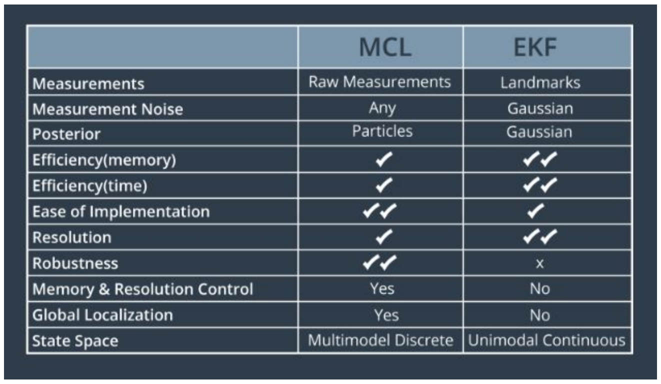
The Extended Kalman filter assumes the motion is linear and have location represented by a unimodal Gaussian function. Since most mobile robots operate in a non-linear fashion (does not always drive in a straight line) Using the first two terms of the Taylor series to compute the Gaussian approximation, the EKF can be applied to a now linear problem.

AMCL are used in nonlinear systems. AMCL can handle almost any kind of model by discretizing the problem into individual "particles". Each particle represents one possible state (position and orientation) of the robot. The location of the robot is determined by the distribution of the particles. The following charts show the pro's and con's for MCL and EKF filters.



	MCL	EKF
Ease of Implementation	✓✓	✓
Measurement Noise	Any	Gaussian
Memory & Resolution Control	Yes	No

Monte Carlo Filter vs Extended Kalman Filter



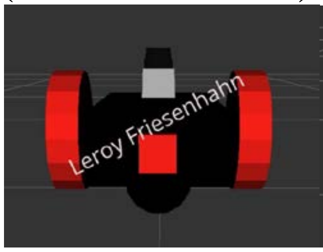
	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	✓	✓✓
Efficiency(time)	✓	✓✓
Ease of Implementation	✓✓	✓
Resolution	✓	✓✓
Robustness	✓✓	x
Memory & Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodal Continuous

Pro's and Con's of Monte Carlo Filter vs Extended Kalman Filter

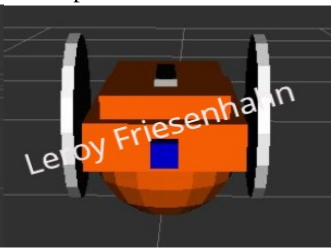
Built in ROS is a localization package called Adaptive Monte Carl Localization (AMCL)..This project will use AMCL to determine the robot's location.

RESULTS

Two robots were constructed. A standard robot (specification given) was implemented. A custom robot (different from the standard) was implemented.



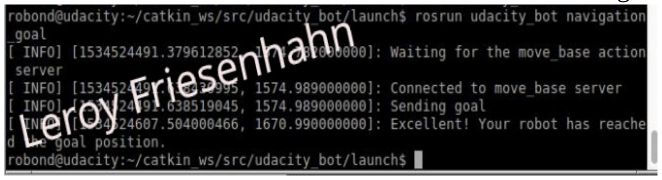
Standard Robot



Custom Robot

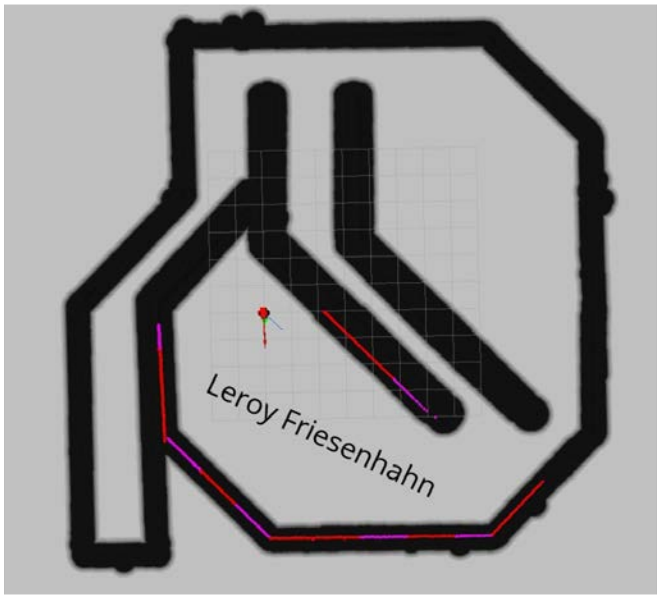
Although the two robots are powered by two wheels, the custom robot has larger and thinner wheels. The custom robot has a larger chassis with a laser sensor mounted differently from the standard robot. Both robots have the same mass.

Both robots achieved the required results. The standard robot outperformed the custom robot in tracking to its destination. The standard robot AMCL particles converge quicker than the custom robot. Although the robot mass were similar the size of the custom robot was larger.

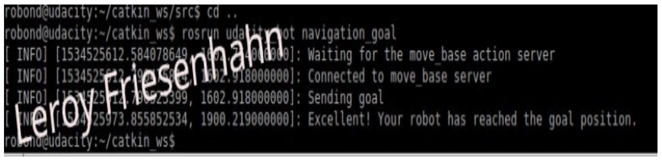


```
robond@udacity:~/catkin_ws/src/udacity_bot/launch$ rosrun udacity_bot navigation
_goal
[ INFO] [1534524491.379612852, 1[illegible]000000]: Waiting for the move_base action
 server
[ INFO] [1534524491.638430995, 1574.989000000]: Connected to move_base server
[ INFO] [1534524491.638519045, 1574.989000000]: Sending goal
[ INFO] [1534524607.504000466, 1670.990000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~/catkin_ws/src/udacity_bot/launch$
```

Standard robot results

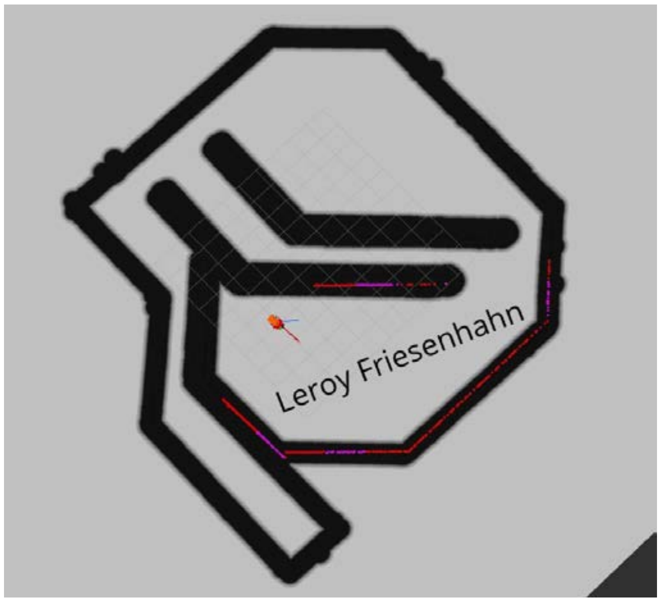


Standard robot reaching it destination



```
robond@udacity:~/catkin_ws/src$ cd ..
robond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1534525612.584078649, 16[illegible]]: Waiting for the move_base action server
[ INFO] [1534525[illegible], 1602.918000000]: Connected to move_base server
[ INFO] [1534525612.790523399, 1602.918000000]: Sending goal
[ INFO] [1534525973.855852534, 1900.219000000]: Excellent! Your robot has reached the goal position.
robond@udacity:~/catkin_ws$
```

Custom robot result



Custom robot reaches goal

MODEL CONFIGURATION

The standard robot was launched using the amcl.launch file. The custom robot was launched using the amcl_custom.launch file. The following configuration was used to tune the standard and custom robot.

To improve the robot's location uncertainty the following parameters were used in the AMCL and AMCL_custom launch file:

update_min_d = 0.01
update_min_a = 0.005

To save on compute resources the number of particles analyzed were changed:

min_particles = 20
max_particles = 80

The minimum range of the laser was set to eliminate the effect of the wheels especially from the custom robot

laser_range = 0.4

The following configuration files were changed to improve the robot performance and meet the robot's objective.

.

base_local_planner_params.yaml

The trajectory planner is responsible for calculating the velocity command used to drive the robot. After analysis of the robots response to the goal influence and global path, the following parameters were modified

pdist_scale: 1.0
gdist_scale: 0.3

local_costmap_params.yaml

Local costmap size effects the ability of the robot to navigate around corners. The goal has influence over the local costmap. After experimentation, the values were set.

width: 4.0
height: 4.0

costmap_common_params.yaml

To improve the robots ability to updating its reading based on the laser sensor(add or remove obstacles).

optical_range = 4.0
raythean_range = 4.0

To prevent the robot from bumping off the wall a radius was defined (padding) around the walls to help in calculating the global path

inflation_radius = 0.6

local_costmap_params.yaml & global_costmap_params.yaml

Parameter were modified to match the performance of the computer used to run the simulation

publish_frequency = 3.0
update_frequency = 5.0

DISCUSSION

The project took a considerable amount of time to setup and test. In the process of adjusting the many parameters, considerable amount of time was required to test and readjust parameters already set. With so many variables, the performance many times depended on minor changes into related parameters Considerable time was spend on researching additional parameter available that was allow the robot to perform its required task.

The hardest task wastuning the robot to follow the global path. If the endpoint appear on the other side of the wall the robot had a tendency to spin in circles. Pdist_scale and gdist_scale eventually alleviated this problem. Failure to recognize update task timeout, masked the problem with following the global path. Once this error was eliminated, the solution was found.

Conclusion/Future Work

The project was frustrating but interesting. This was a very challenging project which only touches on the possibilities of designing and testing a robot in software. Considerable experience was obtained. Although considerable learning occurred, even more time is needed to fully appreciate and use the ROS package. I hope to apply to use this information to allow my students in building and testing their robot for this year's First Tech Challenge.