# Udacity Project: Where Am I?

Brad Huang

July 19, 2018

## 1 Abstract

## 2 Introduction

The global localization problem tries to determine the pose of a mobile robot in a known environment using sensor measurements. The challenges of localization lie in the uncertainties of the robot motions and noises of the sensor measurements. Localization algorithms, such as Extended Kalman Filter (EKF) and Monte Carlo Localization (MCL), aim at filtering out the noises and accurately identifying the position of the robot in the given environment.

The *Where Am I?* project explores using the Adaptive Monte Carlo Localization (AMCL) algorithm to localize a mobile robot in a simulated environment with a provided map. Accurate localization results require tuning parameters of the algorithm to the specific robot sensors and the maze environment. Using the localization results of the algorithm, the robot can navigate through a maze to reach a designated goal pose.

## 3 Background

### 3.1 Localization Problem

As described in Probabilistic Robotics (Thrun et al. 2005), localization estimates the pose of a robot relative to an external reference frame. Localization problem is an important part of robotics, since understanding of the environment and successful navigation are both paramount in performing many tasks, such as exploration of the environment, delivery of objects, and mapping the surroundings. There are multiple types of localization problems, and this project considers the local localization problem. The local localization problem has information about the robot's initial pose relative to the static map of the environment. While exploring the environment, the robot can localize itself through a combination of sensor measurements and movements. In the real world, many environmental and internal conditions can make the sensor measurements noisy, while motors and actuators of the robots are not flawless and can incur additional uncertainties. Thus, solutions to the localization problem require filtering out these noises from the sensor measurements.

### 3.2 Localization Algorithms

Most localization algorithms, including EKF and MCL, succeed by accruing measurements over time and combining them to accurately estimate the robot's pose. Both the EKF and the MCL are widely used localization algorithms. The Extended Kalman Filter is an algorithm that models the uncertainties of sensor measurements and robotic motions as Gaussian noises, and uses iterations of measurement update and state prediction to decrease the variances/covariances of the estimated quantities, which can include velocity, position, orientation ... etc. On the other hand, the Monte Carlo Localization algorithm, also known as particle filtering, uses particles of varying poses to represent the probability distribution of the robot's pose, and resamples the set of particles by the likelihood of each particle based on measurements of landmarks each time the robot moves.

| Property | EKF | MCL |
|---|---|---|
| Measurement Noise | Gaussian Distribution | Any |
| Memory Efficiency | Great | Adjustable |
| Time Efficiency | Great | Adjustable |
| Resolution | Great | Adjustable |
| Global Localization | Incapable | Capable |
| Robustness | Average | Great |

Table 1. Comparison between EKF and MCL algorithms.

However, the EKF and MCL algorithms differ in many aspects (Table 1). The MCL algorithm can model noise of any distribution, is capable of performing global localization, and is generally more robust in noisy environments. In addition, the MCL algorithm can adjust its memory and time efficiency by changing the number of particles representing the pose's probability distribution.

The advantage of being able to adjust resource consumption is further refined with the Adaptive Monte Carlo Localization (AMCL) algorithm. Probabilistic Robotics (Thrun et al. 2005) explained that the AMCL algorithm differs from the MCL with the amount of resamples it takes at every iteration, by accounting for the likelihood of the sensor measurements. This allows AMCL to save significant computational resources compared to the MCL.

## 4 Results

### 4.1 Robot Model

Using the Udacity robot as an example, I created a robot loosely based on a racecar by adding front portion, a driver seat and a rear wing. The front wheels are simplified while the back wheels are preserved. The laser sensor was placed on top of the driver seat, while the camera is moved to the front of the car.

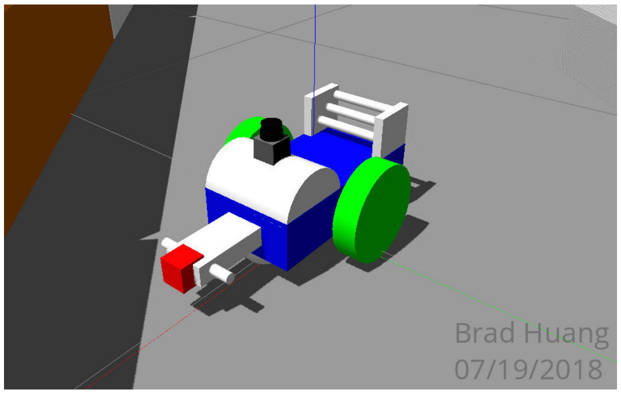

Figure 1. The racecar robot.

## 4.2 Reaching the Goal Position

Both robots get to the goal position in about 60 seconds (including the delays coming from internet connection), and both followed a straightforward path from the starting position, or the origin, to the goal pose. The particles are displayed along the way, and although the localization results were not always accurate while the robot is moving, the results improve through time and the eventual pose arrays were close to the desired accuracy.

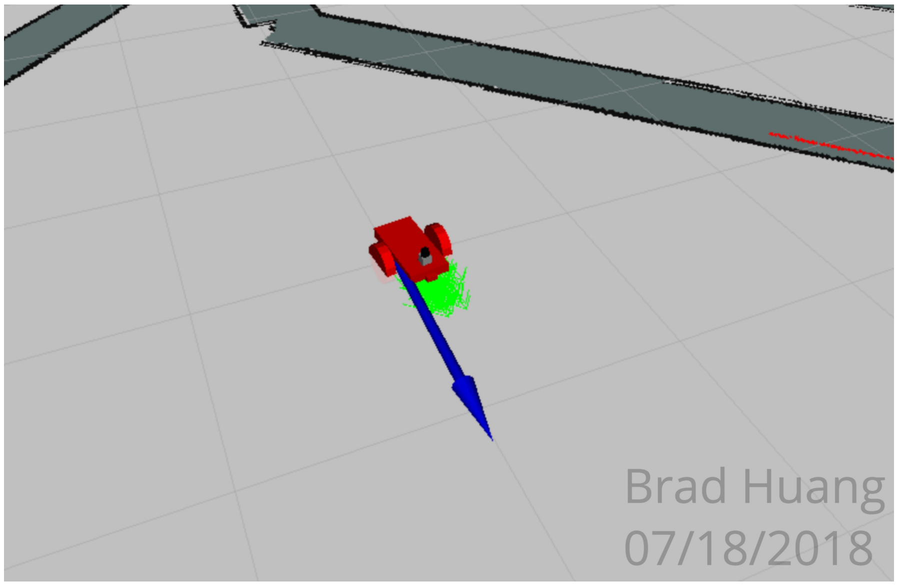

Figure 2. Udacity robot reaching the goal position.

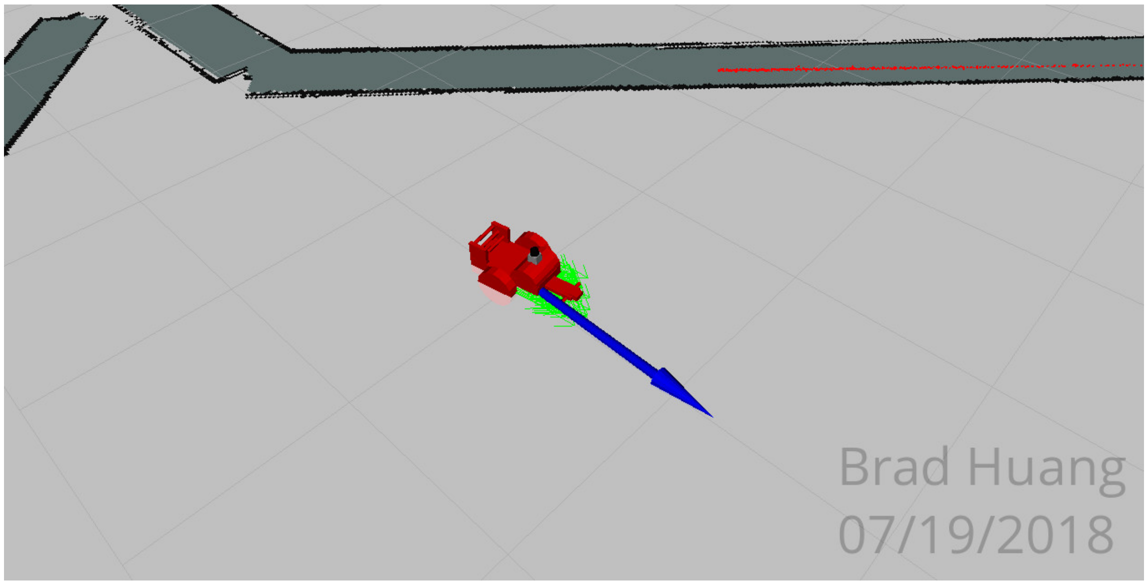

Figure 3. Racecar robot reaching the goal position.

## 5 Model Configuration

### 5.1 Robot Model

The racecar robot has two property differences compared to the original robot. First, the racecar robot is longer but keeps the same width, thus having bigger inertia. Second, the sensors are placed at locations more extruded from the car body. The original robot's camera was blocked by the wheels and the laser sensor's scans were sometimes blocked too.

### 5.2 Parameter Update

For the `acml` and `move_base` packages, their parameters have been updated to achieve better resource allocation and more accurate localization and navigation results. To save resource consumption, I modified the `acml.launch` parameters (Table 2), the `global_costmap` parameters (Table 3) and the `local_costmap` parameters (Table 4). Even though reducing the maximum number of particles might make the probability distribution estimate be more coarse, and reducing the maximum number of laser beams might reduce scan quality, these changes did not seem to affect the eventual results. In addition, I chose to keep the resolutions of the cost maps in order to preserve accuracy.

| Parameter | Default | New | Reason |
|---|---|---|---|
| min_particles | 100 | 50 | Resample fewer particles when measurement likelihood is low. |
| max_particles | 5000 | 200 | 200 particles are enough when the confidence is high. |
| laser_max_beams | 30 | 20 | 20 beams is sufficient for the scans. |

Table 2. Parameter changes in `acml.launch` for efficiency improvements.

| Parameter | Default | New | Reason |
|---|---|---|---|
| update_frequency | 50 | 10 | Map update loop is not fast enough. |
| publish_frequency | 50 | 10 | Map update loop is not fast enough. |
| width | 40 | 20 | Global map only needs to cover 20x20 grids. |
| height | 40 | 20 | Global map only needs to cover 20x20 grids. |

Table 3. Parameter changes in `global_costmap` for efficiency improvements.

| Parameter | Default | New | Reason |
|---|---|---|---|
| update_frequency | 50 | 10 | Map update loop is not fast enough. |
| publish_frequency | 50 | 10 | Map update loop is not fast enough. |
| width | 20 | 5 | Local map only needs to cover 5x5 grids. |
| height | 20 | 5 | Local map only needs to cover 5x5 grids. |

Table 4. Parameter changes in `local_costmap` for efficiency improvements.

To get more accurate localization results, I modified the `acml.launch` parameters (Table 5). The most important changes here are the changes to `update_min_a` and `update_min_d` - since the robot moves at a slow pace, frequent filter update with smaller intervals of motion would keep the particles consistent with the robot and the probability density estimation would be much more accurate.

| Parameter | Default | New | Reason |
|---|---|---|---|
| update_min_a | 0.2 | 0.1 | Update the particles more frequently. |
| update_min_d | $\pi/6$ | $\pi/12$ | Update the particles more frequently. |
| odom_alpha | 0.2 | 0.05 | The odometry measurements in the project are exact. |

Table 5. Parameter changes in `acml.launch` for accuracy improvements.

To facilitate navigation of the robot, I updated several parameters for the `costmap_2d` package (Table 6). The inflation radius is set so that the narrow passages of the maze can still allow the robot to pass, while the obstacle and raytrace ranges are for correctly identifying obstacles in the environment.

| Parameter | Default | New | Reason |
|---|---|---|---|
| obstacle_range | 0.0 | 1.2 | Update the particles more frequently. |
| raytrace_range | 0.0 | 1.2 | Update the particles more frequently. |
| inflation_radius | 0.0 | 0.4 | The odometry measurements in the project are exact. |

Table 6. Parameter changes in `move_base` for navigation improvements.

Lastly, note that the `transform_tolerance` parameter for the `costmap_2d` has been updated to 0.35 to allow some short delays in transform data while the transform data is still updated consistently.

## 6 Discussions

Reducing the global and local map sizes is the biggest factor in making the map update and control loops efficient. This allows the effects of tuning other parameters, including navigation and localization accuracy, to be more transparent. Without the optimization, the robot moves in a jittered path and takes a much longer time to reach the goal.

From the pictures of the robots reaching the goals, we can observe that the position accuracy is lower than the orientation accuracy. Increasing the update frequency of the particle filtering by the distance traveled should make the position accuracy better.

## 7 Future Work

Given more computation resources, I would like to run the simulation more times to reduce the `update_min` parameters to increase the particle filter update frequency, since the localization accuracy should be further improved. In addition, I would like to explore different model configurations, since I attempted creating four-wheel robots but were not able to finish in time.