# Where Am I?

Christopher Ohara

**Abstract**—Localization is a key topic in the operation of autonomous robotics. Utilizing powerful libraries in ROS, two robot models are created and simulated to navigate to a goal location on a known map, using Gazebo and RViz. The robots utilize an adaptive Monte Carlo Localization (AMCL) algorithm that receives sensor data from a camera, odometer, and Hokuyo rangefinder. Parameter tuning is performed to achieve optimal behavior by updating coordinates from sensor measurements to ensure proper navigation. The result is that both the benchmark and custom robots were able to successfully arrive at the target destination.

**Index Terms**—Robotics, Udacity, Localization, ROS, SLAM, EKF, AMCL, Path Planning, Navigation.

## 1 Introduction

GAINING accurate data is a crucial focal point in the performance, predictability, and safety in the field of robotics (especially autonomous). As all sensor types have limitations, sensor fusion using a multiple-sensor combination measuring system (MCMS) approach can reduce impacts of noise, ground slippage (odometry), inaccurate GPS readings, nebulous sight for computer vision, etc.

This project combines sensor data from an odometer, hi-res camera and a Hokuyo UST-20LX scanning laser rangefinder. These hardware devices are attached to an autonomous vehicle that is tasked with reaching a goal while avoiding collisions. The Kalman Filter and an Adaptive Monte Carlo Localization algorithms are explored.

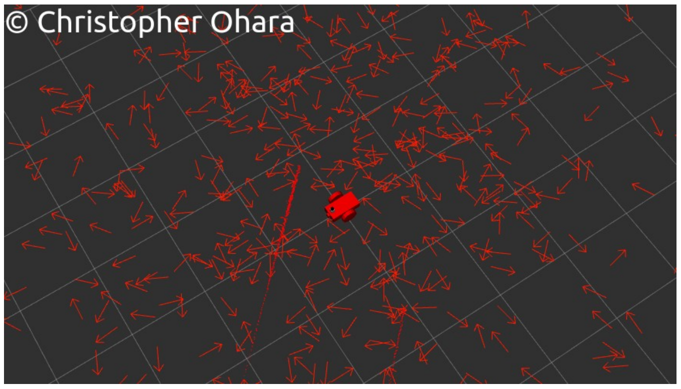


Fig. 1: Initial noise as interpreted by RViz.

## 2 Background

A common misunderstanding about GPS is its reliability. Currently, smartphones are accurate up to 4.9m roughly 95% of the time, without taking into consideration obstructions like buildings and tunnels. An autonomous robot relying solely on GPS data would place both the user and robot in harms way (consider only being in the correct car lane 95% of the time). Therefore, sensor fusion is a mandatory requirement for meeting the function requirements of autonomous robotics.[1]

The most common approaches to interpreting data from sensors is utilizing filters, particularly the Kalman Filter and the Particle Filter. These filters have many different ”flavors” (e.g. EKF or MCL) that allow for high precision in navigation. As an extension of the Recursive Bayesian Estimation technique (Bayes Filter), the PDF of unreliable and continuous measurements from a Markov process allow for a believe, $Bel$, to be estimated and updated. This information can be processed sequentially, allowing for continuous measurements and updates.

### 2.1 Kalman Filters

The general Kalman filter is a derivative of the Bayes filter that is used for linear quadratic estimation. The Kalman Filter has two steps. The first of which is the prediction step, where *a priori* state estimates and error covariances are calculated. Next, an update step is completed, in which these values are updated *a posteriori*. Measurements continue to be received and this process estimates the true states recursively. Due to its requirements towards Gaussian (or normal) distributed curves, it has some limitations. While this method is very beneficial for simplistic, linear systems, it fails to deliver high precision in stochastic environments.

Two common flavors of the Kalman Filter are the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). The EFK is relatively easy to implement and can be used in non-linear systems. The EFK's main task is to arrive at a Taylor expansion equivalent that allows a piece-wise linearization around the mean ($\mu$). The UKF, on the other hand, does not require linearization around the mean, by using a transform that maps various points onto a Gaussian and is very suitable for non-linear systems.[2]
*NB: all forms of KF do not assume the noise will be linear.*

### 2.2 Particle Filters

Particle Filters are also a derivative of the Bayes Filter. However, in its vanilla form, it is able to deal with non-linear systems by discretizing random samples into ”particles”. The more samples that are available, the more accurate the estimate will be. However, in practice, a high number of samples can become problematic (as was found during the experiment). The Monte Carlo Localization (MCL) genetic algorithm is very good at the mutation-selection of particles. The samples are taken from the PDF and given a weighted probability. In order to avoid problems in weight disparity, re-sampling is undertaken and particles with negligible weight values are replaced.[3]

Adaptive Monte Carlo Localization (AMCL) is a flavor of the MCL that implements Kullback-Leibler distance-sampling (KLD), which allows for accurate pose tracking of a robot on a known map. The AMCL has takes inputs of laser-based map, laser scans, and transform messages. It then outputs pose estimates that are used for localization. An important aspect of the AMCL is that it is able to transform inputs (i.e. laser scan data) to the intended frame (odometry, base). This allows for the optimization of techniques like Dead Reckoning, which are generally not optimal for arriving at a goal and only used in non mission-critical autonomous applications.[4]

### 2.3 Comparison / Contrast

In general, Kalman and Particle Filters are similar. They both have prediction and update states, that allow for the refined and fairly accurate estimations. The primary difference is in usage, as the Kalman filter is only suitable for linear systems and the particle filter is intended for non-linear systems. Kalman filters are less adaptable at the trade off being less intensive and easier to implement. Due to the nature of this project, only particle filters (AMCL) were used. However, it is occasionally beneficial to two both filters simultaneously.

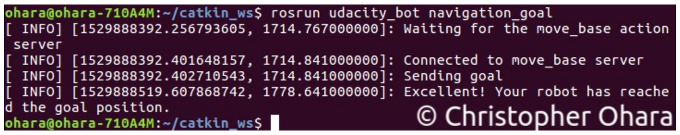
## 3 Simulations

The setup consisted of an Intel Core i5-7400T 2.4GHz processor with 12GB (SODIMM) DDR4 memory and an Intel integrated chipset. Due to the built-in graphics accelerator and lack of an Nvidia GPU, visualizations occasionally crash and are quite intensive for the overhead. This is a known issue and more of an inconvenience than limiting-factor. The build was compiled utilizing the ROS Kinetic distribution on Ubuntu 16.04 LST OS. Modeling and navigation were completed using Gazebo and RViz.

Two robots were created for comparison. The first of which, is a benchmark model that performs well on the map provided by Udacity, labeled as ”udacity_bot”. A custom MkII version was created by altering the URDF (.xml-style) specification file, named ”ohara_bot”. Since the model and simulator have a built in physics engine, a large variation in the parameters will require a great deal of tuning for the differential drive controller.

### 3.1 Achievements

Both version of the robot were able to successfully complete the course. This was not a simple task, as parameter tuning for the Mk.II required careful trial-and-error iterations due to the physics engine and impact that estimation algorithms have on behavior.



```
ohara@ohara-710A4M:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1529888392.256793605, 1714.767000000]: Waiting for the move_base action
 server
[ INFO] [1529888392.401648157, 1714.841000000]: Connected to move_base server
[ INFO] [1529888392.402710543, 1714.841000000]: Sending goal
[ INFO] [1529888519.607868742, 1778.641000000]: Excellent! Your robot has reache
d the goal position.
ohara@ohara-710A4M:~/catkin_ws$
```

Fig. 2: Official results from terminal output

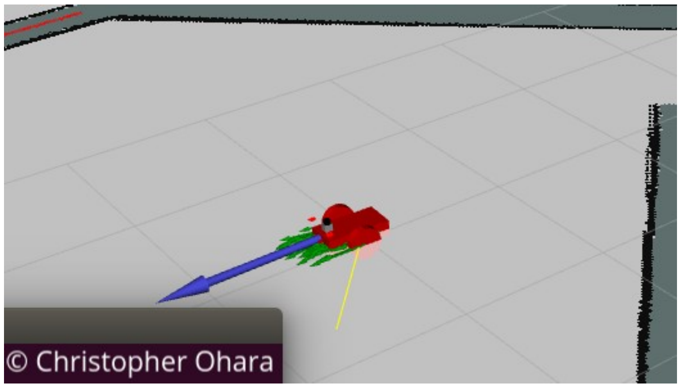


Fig. 3: Mk.I - udacity_bot at desired goal, as rendered in RViz

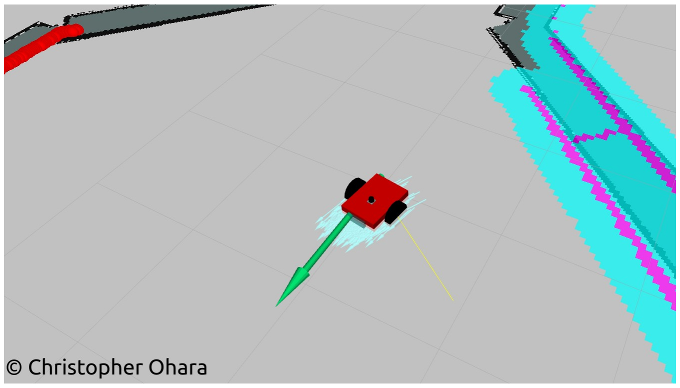


Fig. 4: Mk.II - ohara_bot at desired goal, as rendered in RViz

### 3.2 Benchmark Model

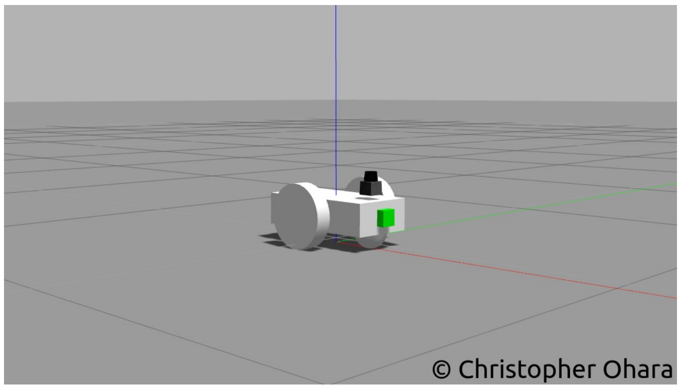


Fig. 5: Mk.I - udacity_bot, benchmark model in Gazebo

#### *3.2.1 Model design*

The initial design has a length of 0.4m, width of 0.2m and height of 0.1m. It includes two additional casters that prevent the robot from having sporadic behavior once the differential drive begins to propel the robot. For sensor hardware, there is one camera, one odometer sensor, and one Hokuyo rangefinder.

#### *3.2.2 Packages Used*

The packages include AMCL, map_server, and move_base. The AMCL allows for the algorithm allows localization to be implemented during the path planning and navigation. The map_server package transforms the given .world file to be transformed into an interactive environment. Finally, the move_base is responsible for actually driving the robot to the desired goal while avoiding obstacles.

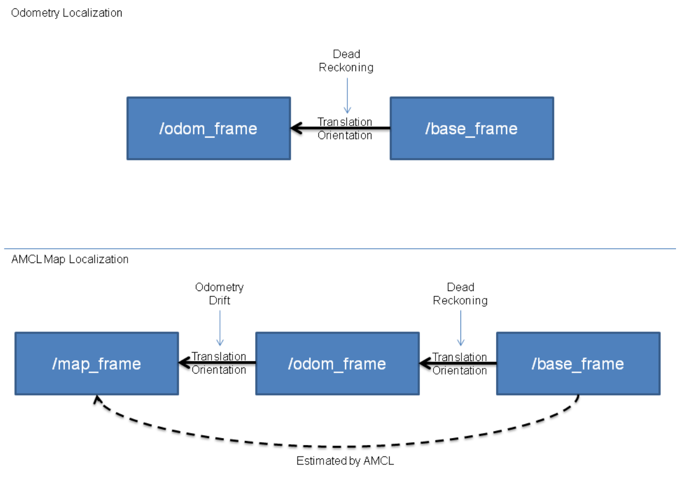


Fig. 6: Overview of AMCL Map Localization, from http://wiki.ros.org/amcl

### 3.2.3 Parameters

For the Mk.II, a variety of critical parameters were included for proper operation as given in the table below. The odom_alpha parameter is tasked with specifying the expected noise in the odometry's translation estimate from the rotational component of the robot's motion. The laser parameters improve the performance of the Hokuyo, under the condition that these parameters sum to a value of one (percentile based).

TABLE 1: Mk.I ACML Parameters

| *Parameter* | *Value* |
|---|---|
| *Localization* | |
| transform_tolerance | 0.10 |
| update_min_d | 0.05 |
| update_min_a | 0.20 |
| min_particles | 20 |
| max_particles | 500 |
| *Odometry* | |
| odom_alpha1 | 0.025 |
| odom_alpha2 | 0.025 |
| odom_alpha3 | 0.025 |
| odom_alpha4 | 0.025 |
| *Laser* | |
| laser_z_hit | 0.96 |
| laser_sigma_hit | 0.02 |
| laser_z_rand | 0.02 |
| laser_likelihood_max_dist | 2.00 |

The different costs maps also have several adjustable parameters. The update and publish frequencies need to be low enough (relative to the host machine) to avoid errors and warnings thrown in the terminal regarding the loop update. This is a relatively common problem in soft real-time systems. The obstacle_range, raytrace_range, and inflation_radius parameters are important regarding how much cost is incurred based on the relative distance the robot is to local obstacles. If these values are too high, it is found that the robot will not navigate through the narrow areas. If these values are too low, the robot will become willingly deadlocked, driving directly into a wall or in a circular movement repeatedly.

TABLE 2: Mk.I Costmap Parameters

| *Parameter* | *Value* |
|---|---|
| *local_costmap* | |
| update_frequency | 7.5 |
| publish_frequency | 2.5 |
| *global_costmap* | |
| update_frequency | 5.0 |
| publish_frequency | 2.5 |
| *costmap_common* | |
| transform_tolerance | 0.4 |
| obstacle_range | 2.0 |
| raytrace_range | 2.5 |
| inflation_radius | 0.4 |
| *base_local* | |
| yaw_goal_tolerance | 0.4 |
| xy_goal_tolerance | 0.2 |
| sim_time | 5.0 |

## 3.3 Personal Model

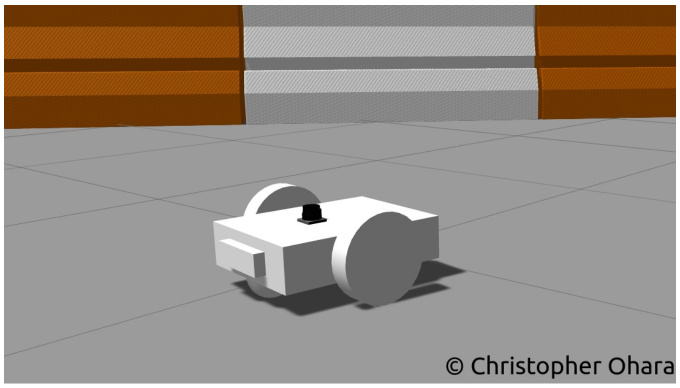


Fig. 7: Mk.II - ohara_bot, custom model in Gazebo

### 3.3.1 Model design

The Mk.II design has a length of 0.4m, width of 0.3m and height of 0.1m. It retains the two casters for stability. For sensor hardware, the camera has been widened by three times the initial value (from 0.05m to 0.15m), one odometer sensor, and one Hokuyo rangefinder.

### 3.3.2 Packages Used

The Mk.II utilizes the same packages as the Mk.I. This is due to the similar goal, structure, and planned operation for the robot. Similarly, comparisons are subject to less ambiguity when most of the factors, including packages, are kept as control variables.

### 3.3.3 Parameters

The critical parameter value changes are primarily the transform_tolerance and laser_min_range. Changing transform_tolerance sets the maximum amount of latency between transforms. If the value is not in the proper range, a

TF error will be returned. The laser_min_range was needed to prevent the robot from interfering with itself during operation. The alpha values were also decreased to 20% of the previous value. This number was arrived at through trial-and-error. The number of particles was also increased to return better performance of the AMCL algorithm.

TABLE 3: Mk.II ACML Parameters

| *Parameter* | *Value* |
|---|---|
| *Localization* | |
| transform_tolerance | 0.50 |
| update_min_d | 0.05 |
| update_min_a | 0.20 |
| min_particles | 100 |
| max_particles | 800 |
| *Odometry* | |
| odom_alpha1 | 0.005 |
| odom_alpha2 | 0.005 |
| odom_alpha3 | 0.005 |
| odom_alpha4 | 0.005 |
| *Laser* | |
| laser_z_hit | 0.96 |
| laser_sigma_hit | 0.02 |
| laser_z_rand | 0.02 |
| laser_min_range | 0.30 |
| laser_max_range | 30.0 |
| laser_likelihood_max_dist | 2.00 |

Some parameter additions were given in the base_local including the yaw_goal and xy_goal tolerances. This allowed for more flexibility in describing the goal state. If the requirements are too strict, the robot make another attempt at arriving at the precise coordinates. Prior to proper tuning, the Mk.II would occasionally become dead-locked, as it would hover around the intended goal, even passing through it, without the program accepting its position as correct since both the heading direction (yaw) and location coordinates (xy) must simultaneously be met.

TABLE 4: Mk.II Costmap Parameters

| *Parameter* | *Value* |
|---|---|
| *local_costmap* | |
| update_frequency | 7.5 |
| publish_frequency | 2.5 |
| *global_costmap* | |
| update_frequency | 5.0 |
| publish_frequency | 2.5 |
| *costmap_common* | |
| transform_tolerance | 0.5 |
| obstacle_range | 3.0 |
| raytrace_range | 5.0 |
| inflation_radius | 0.35 |
| robot_radius | 0.3 |
| *base_local* | |
| yaw_goal_tolerance | 0.3 |
| xy_goal_tolerance | 0.2 |
| sim_time | 5.0 |

## 4 Results

The Mk.I has a slightly better time of reaching the goal at 102s. The Mk.II arrived at the goal at 118s on its best run. Occasionally, both models would drive in the opposite direction upon initialization, change their heading by 180° and then barrel down the narrow path towards to the goal.

Occasionally, both the robot and the map would have unexpected behavior. Ironically, this would almost lead to the "kidnapped robot problem", as the map would re-render when using the roslaunch command for the amcl.launch file, rotating the map overlay. The robot would become very confused about which way to go based on the perceived differences between the goal direction and the sensor data.

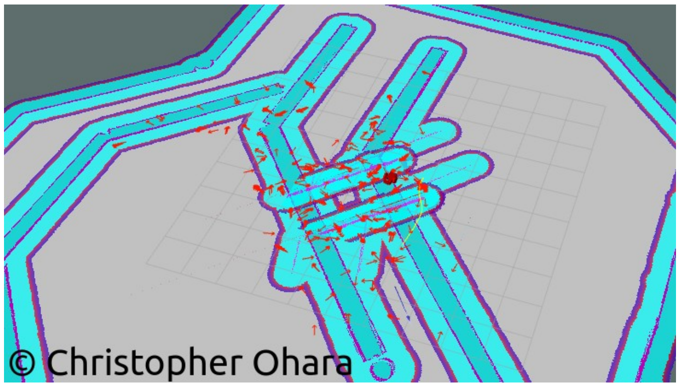


Fig. 8: Map transformation inversion when launching the amcl.launch file in RViz

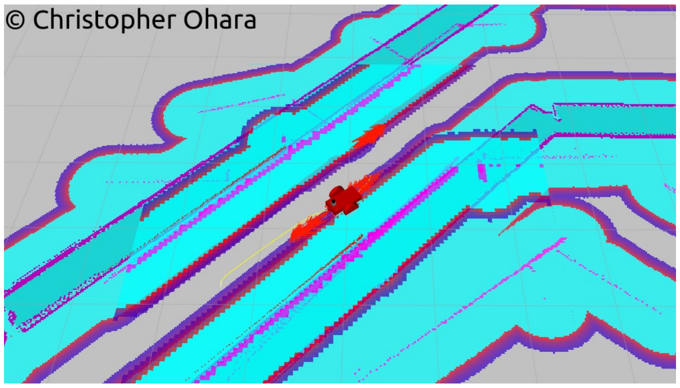


Fig. 9: Map overlay errors resulting in abnormal cost map and pose array characteristics

### 4.1 Localization Results

#### 4.1.1 Benchmark

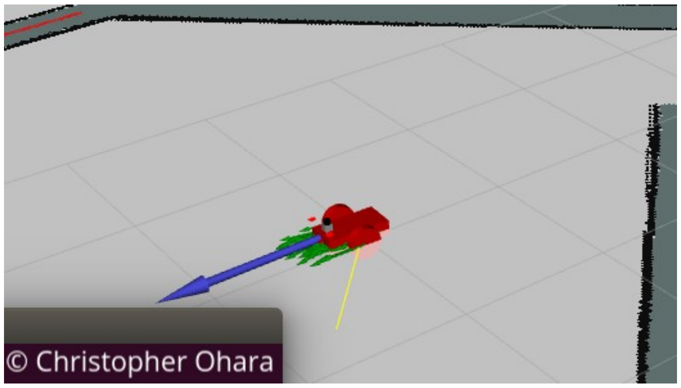


Fig. 10: Mk.I - udacity_bot at desired goal, as rendered in RViz

#### 4.1.2 Student

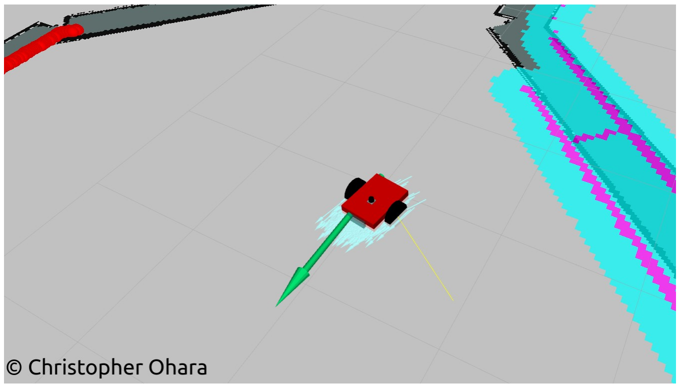


Fig. 11: Mk.II - ohara_bot at desired goal, as rendered in RViz

### 4.2 Technical Comparison

For the Mk.II, the width was increased to improve stability during driving. However, the Mk.I performed better at navigating the narrow path areas, as the robot's radius was smaller and it was less concerned with driving into walls. Decreasing the Mk.I also performed better with the sensor readings, as the increased size of the Mk.II cause some self-interference with the sensors.

## 5 Discussion

The initially planned design for the Mk.II was to increase the height to be more representative of a regular vehicle. However, this caused a few issues with both the differential drive physics and the vehicles sensors. Even attempting to move the Hokuyo to the middle of the vehicle compared to the front end, caused the vehicle to be confused by the difference in it's cost map (due to increased size) and deciding where it could safely navigate. Therefore, these designed were abandoned due to the increase amount of non-intuitive parameter tuning that would incur (as the behavior is stochastic).

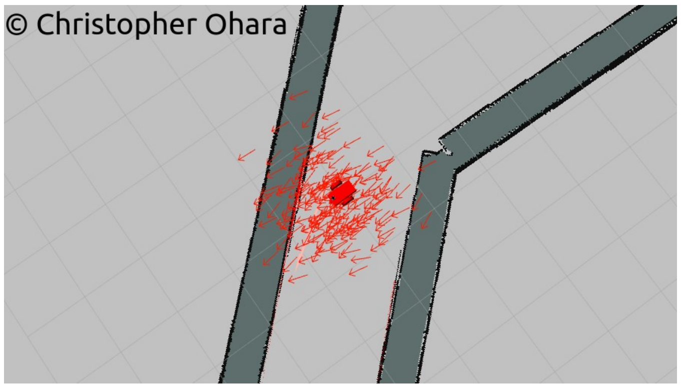


Fig. 12: Mk.I - udacity_bot with converging sensor fusion in pose array

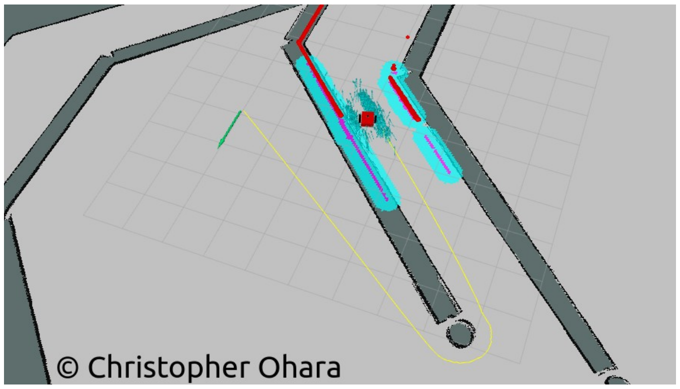


Fig. 13: Mk.II - ohara_bot determining the optimal path to the goal state

### 5.1 Performance

The Mk.I had an overall better performance with respect to navigating around corners and with time. This is likely due to the narrow shape of the robot which allows it to slalom through narrow areas without increasing the estimated probability of collision.

### 5.2 Other Applications

In the event of a ”kidnapped robot situation”, neither the default MCL or EKF algorithms will result in satisfactory localization. However, if the robot has retained knowledge of objects via landmark detection, an Information Filter can be used in conjunction with the AMCL to gain its' bearings.

The MCL could work well in the industry domain during certain nearly ”static” operations. In a smart factory (i.e. Siemens lights-out factory), robotic arms have a range of operations that are generally planned. However, if a conveyor belt delivering products/components that have a random distribution, the MCL might be useful with some computer vision techniques for the robotic arm (like KUKA robotics) to re-orient itself during the manufacturing process.

## 6 Conclusion / Future work

While the project was successfully completed, there are still a lot of areas to explore. ROS has many parameters that can be utilized for just these three sensors. There are a plethora of available sensors that can be added and the shape of the robot can change performance as well. The next steps will be to deploy similar methods on Soft Robotics' Nao robot. This robot suffers from many problems in localization, landmark detection, and obstacle avoidance.

Human-Robot Interaction is a growing field of importance, though the technology is lacking currently. The Nao robot can already employ the MCL and EFKs in operation, though the MCL seriously slows down the computational time. In fact, the EFK is preferred, even though inaccurate, as the robot behaves ”good enough” with its state estimations for localization. Trade-offs between accuracy and processing speed will likely always be a challenge for embedded systems.

## REFERENCES

[1] ”GPS Accuracy”, GPS.gov: Agricultural Applications, https://www.gps.gov/systems/gps/performance/accuracy/

[2] *Fox, Dieter*, KLD-Sampling: Adaptive Particle Filters, papers.nips.cc/paper/1998-kld-sampling-adaptive-particle-filters.pdf.

[3] *Thrun, Sebastian and Burgard, Wolfram and Fox, Dieter*, Probabilistic Robotics (Intelligent Robotics and Autonomous Agents), 2005, ISBN (0262201623), The MIT Press.

[4] ”ROS Wiki”, ros.org: AMCL, wiki.ros.org/amcl, http://wiki.ros.org/amcl