Where Am I?

pook, chun

Abstract—This project utilize ROS packages to implement localization of mobile robots inside a provided map in the Gazebo and RViz Simulation environment. A robot was designed to leverage on AMCL to implement the monte carlo localization (MCL) and Navigation Stack move_base package to define and navigate to goal position.

Index Terms—Robot, particle filter, monte carlo localization, Localization.

1 Introduction

Kalmen filters and Monte Carlo Localization are two most common localization algorithm, each has its own pros and cons.

Localization refers to the problem where the Robot determines where it is located with respect to its environment, and how it estimates its position and orientation within the map using information gathered from noisy sensor measurements.

Using Adaptive Monte Carlo Localization (AMCL) and Navigation Stack move_base packages in ROS environment, a custom robot was implemented which localize itself in a predefined environment and be able to move from current position to final goal position.

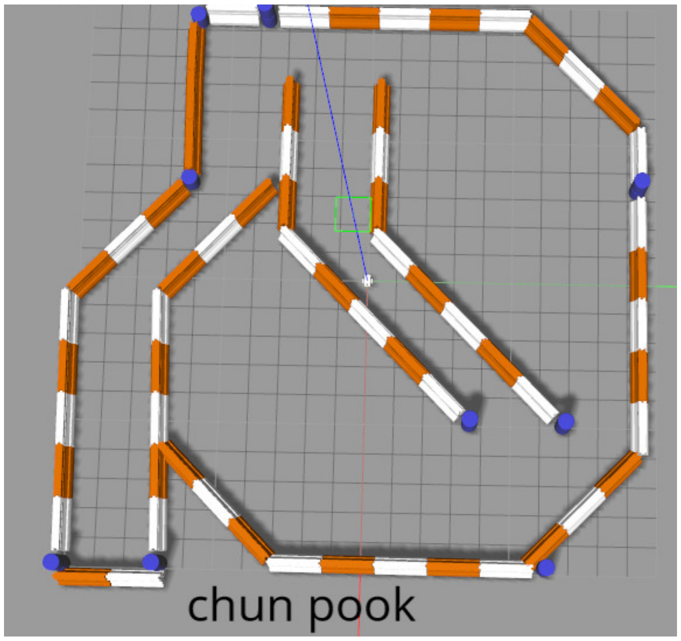


Fig. 1. Top View of robot and its environment

2 Background

In order for the robot to plan for future action, it must be able to determine its location and orientation with respect to its environment using noisy data gathered from the sensor. Kalman filter and Monte Carlo Localization (particle filter) are two most common localization algorithm being used.

2.1 Kalman Filters

Kalman filter is an algorithm that make use of noisy data on a linear system with Gaussian error to update the estimate of its system state.

The algorithm is a two step process, where in the prediction step, it produces estimate of the current state with uncertainties. Then in the measurement step, it incorporates the noisy measurement data by applying weighted average to the data with more weight being given to estimate with higher certainty. And then the whole process is repeated indefinitely.

when the system is a linear gaussian system, kalman filter is very efficient. But real world application is rarely linear, nor it is unimodal gaussian. To overcome these limitation, extended kalman filter can be used.

The extended kalman filter is a variation of kalman filter, it works by linearizing the non linear measurement or motion.

2.2 Particle Filters

Monte Carlo Localization initially generate number of particles scattered randomly in space and orientation throughout the map. As robot moves around, it gathered the noisy measurement from its sensor, which is used to update the weight for each particle based on the difference between the robot's actual pose vs the predicted pose. This weight serve to indicate the importance of the particle. Particle with higher weight are more likely to survive during the re sample process.

```
Algorithm MCL($X_{t-1}, u_t, z_t$):
    $\overline{X_t} = X_t = 0$
    for m = 1 to M:
        $x_t^{[m]} = motion\_update(u_t, x_{t-1}^{[m]})$
        $w_t^{[m]} = sensor\_update(z_t, x_t^{[m]})$
        $\overline{X_t} = \overline{X_t} + < x_t^{[m]}, w_t^{[m]} >$
    end for
    for m = 1 to M:
        draw $x_t^{[m]}$ from $X_t$ with probability $\propto w_t^{[m]}$
        $X_t = X_t + x_t^{[m]}$
    end for
    $return X_t$
```

2.3 Comparison / Contrast

Kalman filter and Monte Carlo localization (particle filter) algorithm has its own pros and cons, which is summarized in the table below.

As shown in the table below, Particle filter is relatively easy to implement, work with non linear and non gaussian system. This allows particle filter be applied to various applications. Also the computational memory requirement and its performance can be fine tuned by controlling by the number of particles. For these reasons, particle filter will be used to implement robot localization in this project.

TABLE 1
Extended Kalman filter(EKF) Vs Monte Carlo Localization(MCL)

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

3 Simulations

The Robot was setup to run up in ROS environment, using AMCL package for localization and navigation stack move_base package for defining goal position and navigate to that goal position.

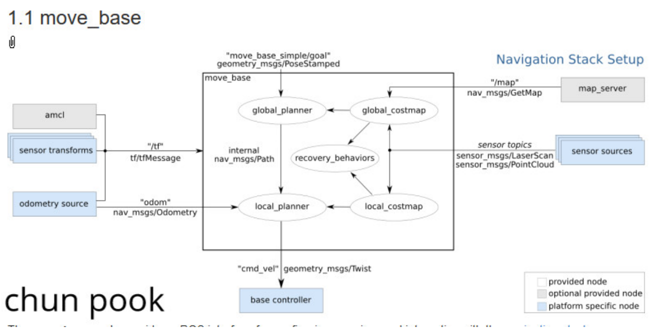


Fig. 2. amcl and navigation stack node diagram taken from ros wiki

Two simulations has been carried out, one for the udacity bot and one for the custom built robot, with the goal of having the robot being able to localize itself and navigate from its start position to the goal position.

3.1 Achievements

Both udacity and custom built robot was able to successfully reach the goal position after tuning the configuration parameters.

As can be seen from figure below, initially both the udacity and custom built robot, has high uncertainty as where they are.

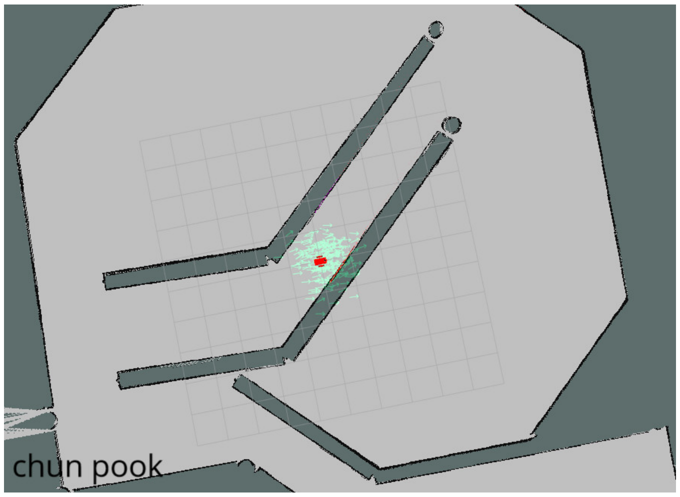


Fig. 3. udacity bot has high uncertainty initially

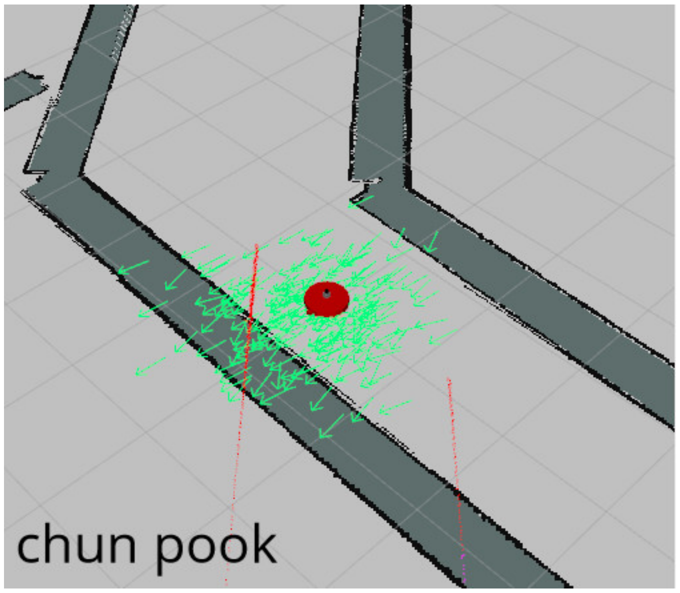


Fig. 4. custom built bot has high uncertainty initially

As the robot start moving around, the localization algorithm was able to improve its estimation gradually, and start to converge. see below the snapshot taken for the udacity as it moves towards the goal position, the system is getting more certain of its position.

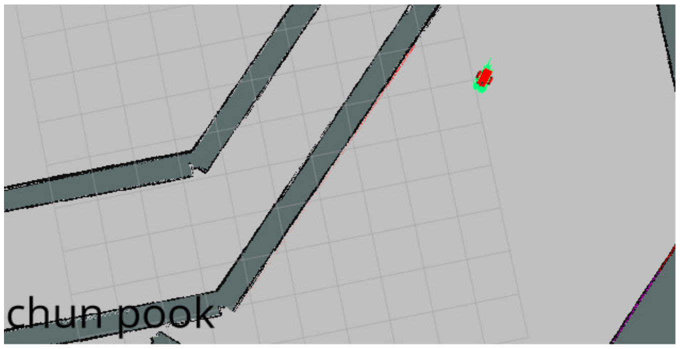


Fig. 5. udacity bot moving towards the goal position

Similarly for the custom built robot, it is getting more certain of it position as it moves towards the goal position.

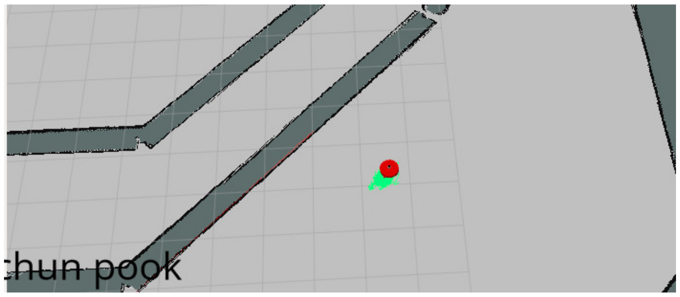
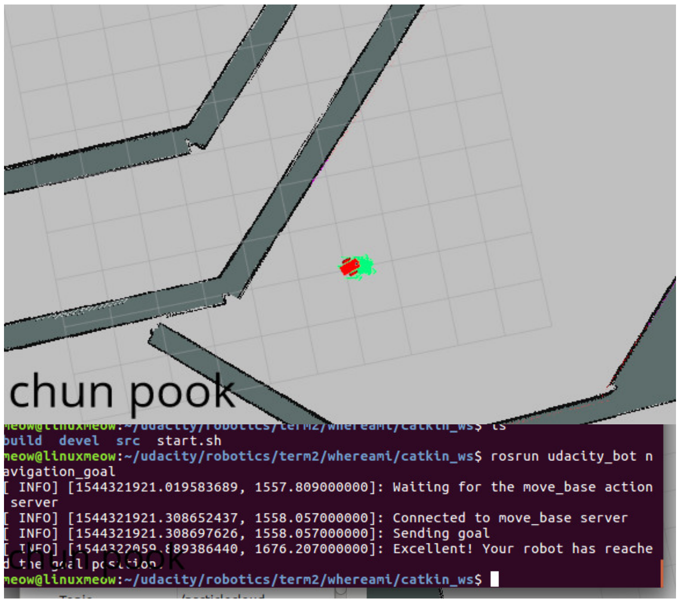


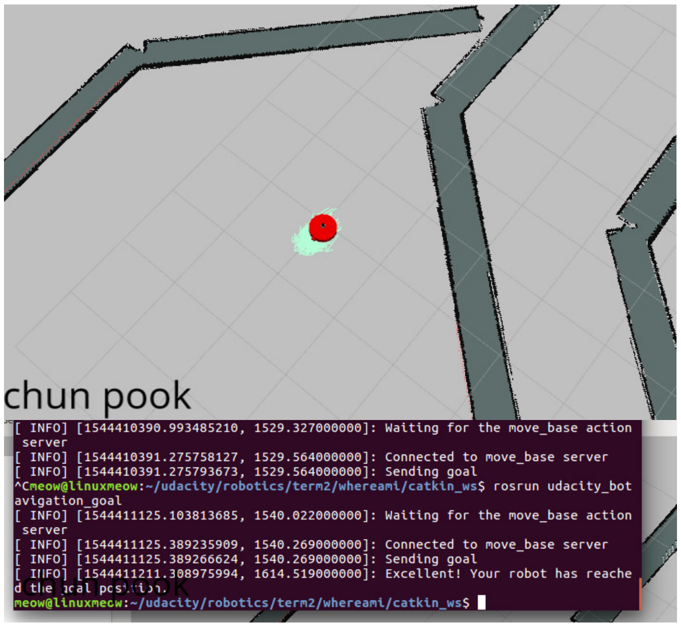
Fig. 6. custom built bot moving towards the goal position

And both the benchmark udacity and custom built robot was able to reach the goal position.



```
meow@linuxmeow:~/udacity/robotics/term2/whereami/catkin_ws$ ls
build  devel  src  start.sh
meow@linuxmeow:~/udacity/robotics/term2/whereami/catkin_ws$ rosrun udacity_bot n
avigation_goal
[ INFO] [1544321921.019583689, 1557.809000000]: Waiting for the move_base action
 server
[ INFO] [1544321921.308652437, 1558.057000000]: Connected to move_base server
[ INFO] [1544321921.308697626, 1558.057000000]: Sending goal
[ INFO] [1544322059.889386440, 1676.207000000]: Excellent! Your robot has reache
d the goal position.
meow@linuxmeow:~/udacity/robotics/term2/whereami/catkin_ws$
```

Fig. 7. udacity bot reached goal position



```
[ INFO] [1544410390.993485210, 1529.327000000]: Waiting for the move_base action
 server
[ INFO] [1544410391.275758127, 1529.564000000]: Connected to move_base server
[ INFO] [1544410391.275793673, 1529.564000000]: Sending goal
^Cmeow@linuxmeow:~/udacity/robotics/term2/whereami/catkin_ws$ rosrun udacity_bot
avigation_goal
[ INFO] [1544411125.103813685, 1540.022000000]: Waiting for the move_base action
 server
[ INFO] [1544411125.389235909, 1540.269000000]: Connected to move_base server
[ INFO] [1544411125.389266624, 1540.269000000]: Sending goal
[ INFO] [1544411211.308975994, 1614.519000000]: Excellent! Your robot has reache
d the goal position.
meow@linuxmeow:~/udacity/robotics/term2/whereami/catkin_ws$
```

Fig. 8. custom bot reached goal position

3.2 Benchmark Model

Below is a snapshot of the benchmark model, it is a two wheeled rectangular shape robot, with front camera and a laser scanner on the top front.

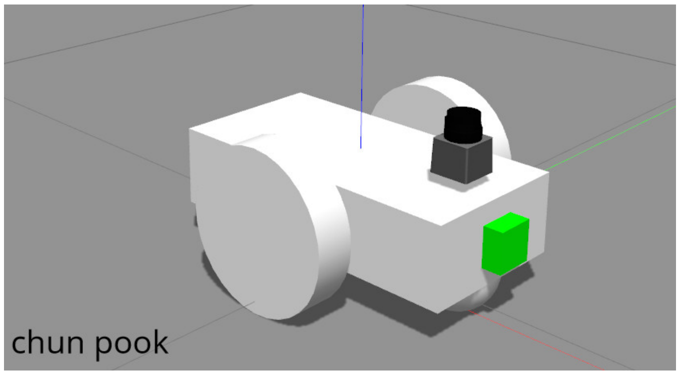


Fig. 9. udacity bot

3.2.1 Model design

The Benchmark robot was provided by udacity, with the following configuration defined in the urdf file.

TABLE 2
Udacity Benchmark Bot

chassis	mass size	15.0 0.4 x 0.2 x 0.1
back caster	radius	0.0499
front caster	radius	0.0499
left wheel	mass cylinder length cylinder radius	5.0 0.05 0.1
right wheel	mass cylinder length cylinder radius	5.0 0.05 0.1
camera	mass box size child link parent link joint origin	0.1 0.05x0.05x0.05 camera chassis 0.2,0,0,0,0,0
hokuyo	mass box size child link parent link joint origin	0.1 0.1x0.1x0.1 hokuyo chassis 0.15,0,0.1,0,0,0

3.2.2 Packages Used

The following packages are used in this project.

- amcl
- navigation stack

And the following topics are used to allow communication between components

- udacity_bot/laser/scan to map to scan for amcl, move_it package
- image_raw used in gazebo
- cmd_vel to map to cmd_vel for move_it package
- odom to map to odom for move_it package
- camera_info used in gazebo

3.2.3 Parameters

In order to have the robot moved from the starting position to its final goal position. Parameters needs to be tune iteratively. The parameters are listed below

The higher the number of particles used, the more accurate the localization is. However the computational overhead also increase, and compromise has to be made between accuracy and system response time.There is an issue with delay or latency allowed between transforms, hence the transform_tolerance has to adjusted shown in the configuration below.The parameters related to odometry's rotational and translational expected noise was also adjusted to the value below.

AMCL Parameter

transform_tolerance	0.2
min_particles	10
max_particles	200
odom_model_type	diff-corrected
odom_alpha1	0.010
odom_alpha2	0.010
odom_alpha3	0.001
odom_alpha4	0.001
update_min_d	0.05
update_min_a	0.05

In costmap configuration file, Various parameters are tuned, like inflation_radius that determines the minimum distance between robot and obstacles. raytrace_range which is used to clear and update the free space in the costmap as the robot moves. And Obstacle_range which determines the maximum range sensor reading that will result in an obstacle being put into the costmap, which can help with discarding noise.

TABLE 3
costmap Parameter

map_type	costmap
obstacle_range	5.0
raytrace_range	10.0
transform_tolerance	0.3
inflation_radius	0.6

Then in base local planner and local configuration file, the update frequency, publish frequency and controller frequency was adjusted to reduce the load on the system.

TABLE 4
base local planner Parameter

holonomic_robot	false
controller_frequency	10.0
sim_time	1.0
pdist_scale	0.1
gdist_scale	0.8
occdist_scale	0.01
meter_scoring	true
publish_cost_grid_pc	true

TABLE 5
local costmap

global_frame	odom
robot_base_frame	robot_footprint
update_frequency	3.0
publish_frequency	5.0
width	3.0
height	3.0
resolution	0.05
static_map	false
rolling_window	true

3.3 Personal Model

3.3.1 Model design

The custom built bot was designed to be circular in shape as shown below. With the laser scanner sitting at the center of the robot. In addition, it has 2 small wheels at the bottom, plus an extra castor wheel allowing it to move smoothly.

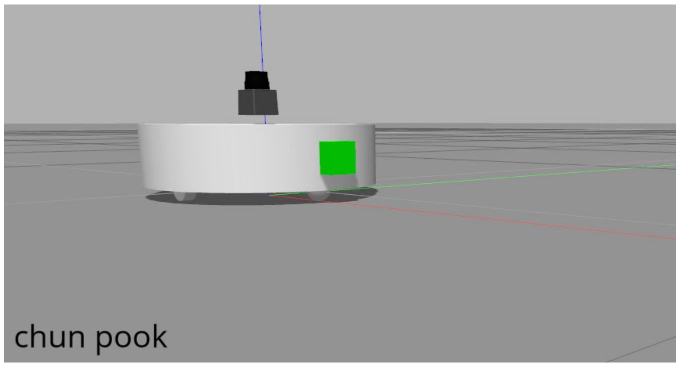


Fig. 10. udacity bot

TABLE 6
Udacity Benchmark Bot

chassis	mass cylinder radius cylinder length	10.0 0.2 0.1
left wheel	mass cylinder length cylinder radius	1.0 0.02 0.02
right wheel	mass cylinder length cylinder radius	1.0 0.02 0.02
castor	mass sphere	1.0 0.02
camera	mass box size child link parent link joint origin	0.1 0.05x0.05x0.05 camera chassis 0.2,0,0,0,0,0
hokuyo	mass box size child link parent link joint origin	0.1 0.1x0.1x0.1 hokuyo chassis 0,0,0,0,0,0
left wheel hinge	friction damping	0.1 0.1
right wheel hinge	friction damping	0.1 0.1
castor joint	friction damping	0.0 0.0

3.3.2 Packages Used

same as udacity bot above

3.3.3 Parameters

Most of the parameters are exactly the same as the udacity bot above, except the following

TABLE 7
my bot parameters

AMCL	odom_alpha3	0.010
AMCL	odom_alpha4	0.010

4 Results

4.1 Localization Results

Both robots was able to localize itself and navigate from starting position to the final goal position. And while it is moving towards the goal, they never collide with the wall and was able to recover if it get stuck.

4.1.1 Benchmark

The udacity robot took around 4 to 6 seconds for particle filters to converge, and took around 70 to 80 seconds to move to its destination. At the starting position, there are some wandering around initially when the robot starts moving. Sometimes the robot was going in wrong direction, but it recovers and was able to reach the goal position.

4.1.2 Student

The Custom robot took around 5-10 seconds for particle filters to converge, and took around 45 to 50 seconds to move to its destination.

4.2 Technical Comparison

The custom robot has a lighter weight, and smaller friction on its wheel. It is moving faster than the benchmark model moving to its goal. The laser on the custom robot was placed at the center of the robot, this making the scanning symmetrical around the robot.

5 Discussion

The custom robot seems to be able to move to the final destination faster, probably due to its smaller weight, and smaller friction.

Occasionally the robot will wander around and then recover. To improve the localization accuracy, more particles can be used, parameters like obstacle_range, raytrace_range, publish_frequency can be fine tuned, but this will increase the computational burden on the system, and render the system unresponsive, so care has to be taken to balance the accuracy vs system resource requirement.

Particle filters was expected to work in a kidnapped robot scenario, as the algorithm relies on randomly generating particles scattered throughout the map initially, and converge into the area where the robot likely to be, as more noisy sensor data are collected. If the robot is then kidnapped to another location the random walk should be able to re-discover where it is located.

6 Conclusion / Future work

Both Robots were able to localize itself and navigate from starting to final goal position, without colliding with obstacle and was able to recover if it got stuck.

At the starting position, Robot has high uncertainty, and sometimes wandering around in the wrong direction, more fine tuning is needed. There are parameters like raytrace_range, obstacle_range that can be fine tune to mitigate the effect. But current system quickly ran into performance problem, the system was not processing update quick enough, causing time out and the robot becomes unresponsive.

The Robots are only ran in simulated ROS environment. future work will include deploying to the actual robot platform like NVIDIA Jetson TX2.

REFERENCES

1) udacity, Robotics nanodegree, lesson on Monte Carlo Localization
2) http://wiki.ros.org/move_base, ros wiki