Abstract

This project outlines the robot has been developed based on the lecture notes and also a modified version of the robot model. The robots are simulated in Gazebo and RViz. Given a map (an environment), the robot is able to navigate to the given end goal. The project depends on Adaptive Monte Carlo Localization (AMCL) library for navigation.

Introduction

This project is to teach a robot on how to localize itself and also able to navigate around its environment, reaching its end goal. Localization is a hard problem. And there are different difficulties, e.g the robot's initial pose is unknown in a global localization or when it is "kidnapped", while in local localization there is uncertainty in pose. Not knowing it's initial pose, makes it hard for the robot to be able to know it's pose based on ground truth, and makes it difficult to be able to reach its end position. There are two ways for a robot to self localized: Kalman Filter and Monte Carlo Localization (MCL). This project presents the results of implementing Adaptive Monte Carlo Localization (AMCL) which is similar to MCL but it dynamically adjusts the number of particles over a period of time.

Background

Localization is important for a robot to accurately determine its pose and understanding its environment. Localization pose many challenges, where the amount of information and environment determine the difficulty of the problem. In increasing difficulty order:

1. Position tracking/local localization: the robot knows its initial pose, however it needs to be able to estimate its pose as it moves around with consideration of uncertainty in its motion surrounding its region
2. Global localization: the robot does not know its initial pose, but must determine its pose relative to the ground truth map. This has more uncertainties than local localization
3. Kidnapped robot: the robot may be moved to a new location randomly and poses the same challenges as global localization

Depending on the tasks, it is essential for a robot to be able to localize itself in its environment. These are mostly for robots that moves around its environment to achieve a goal or perform a task, and not for static robots that are fixed to a position.

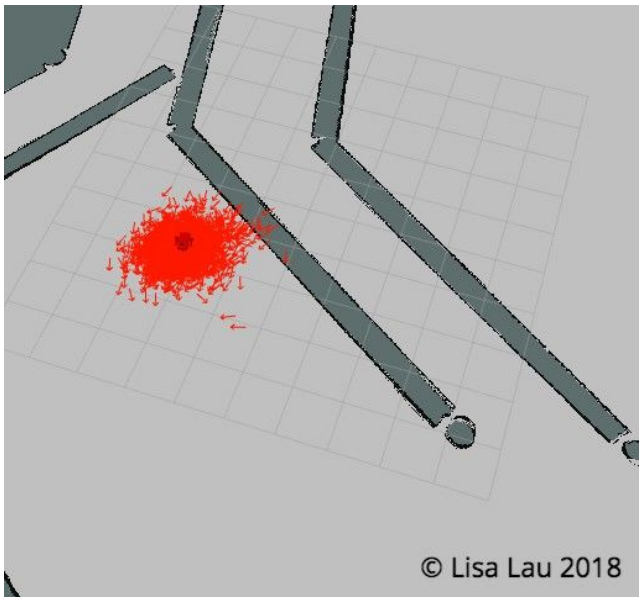
Kalman Filter is one of the algorithm used for localization. It is fast compared to other algorithm and able to take data with a lot of uncertainty or noise in the measurements and provide a very accurate estimate of the real value.

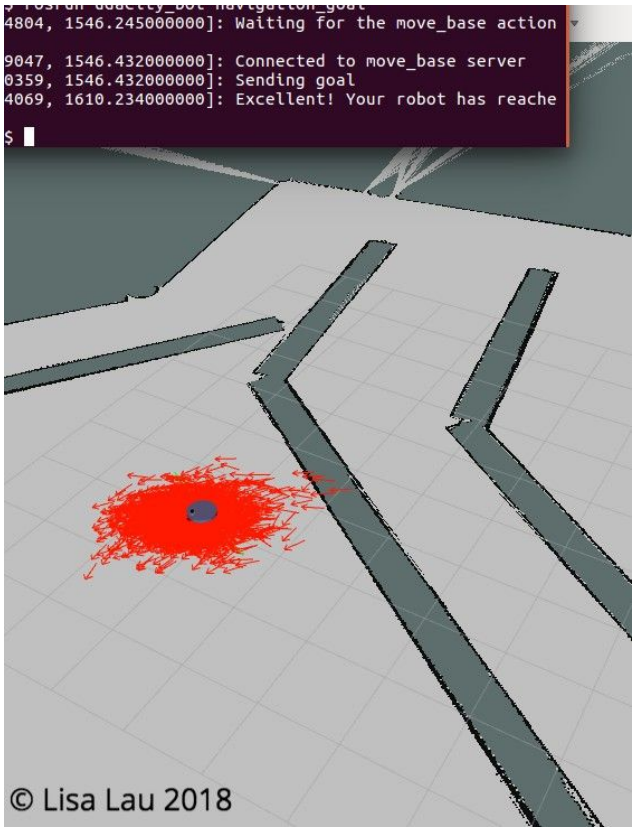
Monte Carlo Localization (MCL) is one way to solve localization problem, and is one of the most popular algorithm. It allows the robot to knows its pose. However it is only limited to local and global localization problem. This algorithm is unable to solve kidnapped robot problem.

The table below shows the difference between MCL and EKF.

	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

Results


© Lisa Lau 2018


4804, 1546.245000000]: Waiting for the move_base action
9047, 1546.432000000]: Connected to move_base server
0359, 1546.432000000]: Sending goal
4069, 1610.234000000]: Excellent! Your robot has reache
$
© Lisa Lau 2018

Model Configuration

Several parameters were tuned to be able to achieve the result - robot reaching the end goal, which is position at (0.995, -2.99) with orientation of (0.0, 0.0, 0.0, 1.0).

Local costmap

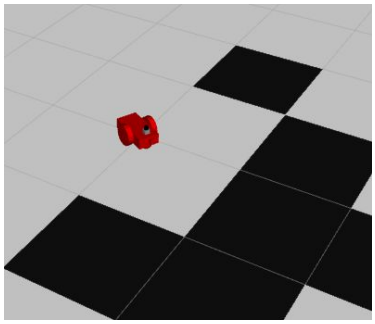
These parameters is to specify the behaviour associated with the local costmap.

Update and publish frequency

This was lowered as there were some warning on the terminal about frequency not met. It did not have huge impact on the performance of the robot unless it is very low, i.e. 1.0.

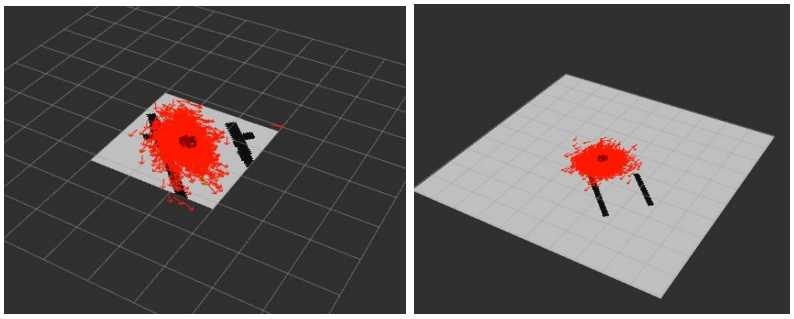
Resolution

The resolution was kept at 0.05 as at a much higher resolution, the accuracy drops, where the the local map is no longer a line, but just grids. And any slight obstruction is enlarged.



Width and Height

Only the width and height for the local map has been decreased. The image below shows the difference between a width/height of 3.0 and 10.0. Basically it means the coverage of the local map, a bigger map means it takes consideration more information of its surroundings with a wider range. These parameters have been tweaked so the sizes are enough for the current path only and would not be affected by path outside its region.
(left) 3.0 (right) 10.0



Base local planner

This is useful when it has an end goal, where a path or a trajectory to the goal position is calculated, and the robot is required to navigate to that path.

Yaw and XY goal tolerances

These values have been tweaked to ensure that the robot post is as close to the goal position

Sim_time, meter_scoring and pdist_scale

These value has been tweaked to ensure that the robot follows the given path more closely. Tweaking number helped as there were times where the robot strayed away from the calculated trajectory.

Common costmap

Obstacle and raytrace range

This has to be a non 0, and anything bigger than 1.0 seems to work for the robot very well.

Transform tolerance

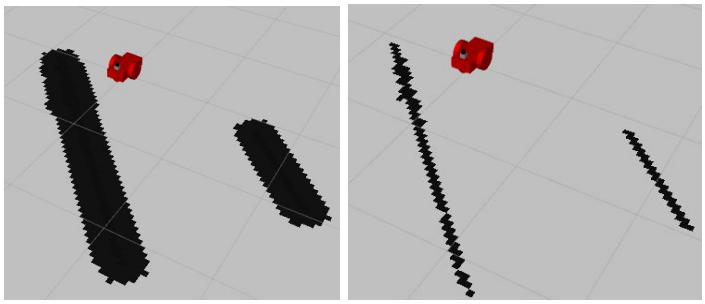
This is important as it decides the longevity of the transform being published for localization purposes. Too high of a value would cause a significant delay or latency. However the lower the value, the more processing power it would require. So there is a tradeoff between a too low or too high value. 0.3 was chosen as the robot seems to react well.

Robot radius

The radius is set to 0.2 as taken from the base size

Inflation radius

The radius means the robot will treat all path with that distance away from obstacles as having equal obstacle cost. The local and global costmap would reflect this value in any obstacles it sees.
(left) 3.0 radius (right) 0.0 radius



Discussion

There were many hiccups in the project, and it was hard to debug the exact causes. First was the robot not responding at all initially. After trying all the parameters, it was the range values that has been set to 0, that were causing this problem. Next was getting the origin values for the right and left wheel links incorrect. From RViz it seems like it has been set correctly, even though the link was not. This results in the robot getting stuck occasionally and it was hard to understand whether it was due to bad parameter tuning or not. Fixing the link origins results in the robot moving smoothly, although it still would get stuck around the origin, as the trajectory was pointing it towards the end goal, but through the wall. By reducing the size of the local costmap, it made sure that only that range is taken into account when calculating the trajectory, this ensures that the robot tries to find a route that makes sense to its current local map. With this change, the robot is able to reach the end goal, with one caveat, it tries to go along the wall and takes a long detour to the bottom of the map even though there is a shorter route to the end goal. The last tweak was updating the base local parameters, where we tweaked parameters to ensure the robot tries to follow the given trajectory.

According to the lecture, MCL does not work for kidnapped robots. This may be due to problem of accurately detecting its pose at all times, especially a failure to localization if it is kidnapped to an area where there is an absence of particles in some when the particles converges. As AMCL dynamically adjusts the number of particles over a period of time, it may help is recalculating the its position, although it may take more computational power and time.

MCL and AMCL would be useful in many places where localization is needed. In the household, this is useful for a robot vacuum cleaner to be able to self localize and make sure it cleans every

part of the house. In a packaging/distribution centers, i.e Amazon, it'll be useful for robots to be able to deliver packages from one place to another without any human help.

Future Work

There is a trade of in accuracy and processing time, i.e parameters like transform tolerance is a good example. A lower latency is required if the robot is to be able to react real time, but it means that the robot would need to be able process all the points within the given time. This is more essential, together with update/publish frequency, in a dynamic environment where the map is updated frequently. Modifying the robot model from rectangular to cylinder does give the robot an advantage, where it seems like it is moving quicker to the end goal as compared to the previous shape where it was a rectangle. May also be worth considering other sizes, and weights as a lighter robot means quicker, however it does also mean less stable. Additional sensors would be useful, such as tactile or pressure sensor, where it is able to identify different objects or motion sensors, where it would be able to detect if something is moving and react accordingly.