Where Am I (Localization Project)

Abstract:

This paper presents robot localization in ROS using the Adaptive Monte Carlo Localization (AMCL) package. The paper discusses various parameters, how they were tuned for the provided robot and difficulties in the parameter tuning. Furthermore, a new robot is created with different size of the robot and wheels and different sensor positions. Effect of these changes on previous set parameters is discussed. At the end, future enhancements in localization are discussed.

Introduction:

This paper discusses how to solve the robot localization problem using ROS AMCL (Adaptive Monte Carlo Localization) package. The ground truth map is provided and move_base package is used for the robot path planning. The goal of the project is to reach the given goal position in the map, with minimal and denser pose array.

Initially, the robot pose array consists of many particles which are spread widely around the robot. The particles have random positions and orientations. At this point, the uncertainty in robot's position is quite high. As the robot starts moving towards its goal, the AMCL package gathers data from its laser and camera sensor and attempts to localize the robot. However, to achieve quicker and accurate localization, lot of parameters need to tweaked to find their optimum values. This paper discusses effects of various parameters on robot localization and how the localization was achieved.

Later on, the robot model is changed and the localization problem is solved for the new robot. Effects of various AMCL and Move Base parameters is discussed. The parameters needed to be changed is also discussed.

Background:

When a robot is moving in real world, it uses various sensors and its odometry data to localize itself in the given map. However, the sensors such as laser, radar, camera, etc., are noisy. Additionally, the robot odometry is also noisy. For example, the robot does not move exact distance as it was expected to. Therefore, as the robot moves, the sensor and odometry noises add up resulting in large error in the robot's position. Consequently, the robot will be lost. The localization algorithms help the robot to take data from multiple sensors and remove noise to achieve a better prediction of the robot's position.

There are four types of localization algorithms -

1. Extended Kalman Filter (EKF)
2. Markov Localization
3. Grid Localization
4. Monte-Carlo Localization (MCL)

Here I will briefly discuss the first and the forth algorithms which were introduced in the class. The Kalman Filter algorithm has two major steps, state prediction and measurement update. These steps are continuously executed in loop. EKF is an extended version of Kalman Filters. It uses Taylor series to approximate nonlinear function so that it can use gaussian error distributions. The Kalman filter can achieve quite accurate estimate of robot's position with only a few sensor updates. The MCL algorithm, also known as particle filter, uses a number of particles with random positions and orientations, along with the sensor measurements, to localize the robot. Each of this particle is a pseudo particle that resembles the robot. These particles are resampled each time the robot moves and senses its environment.

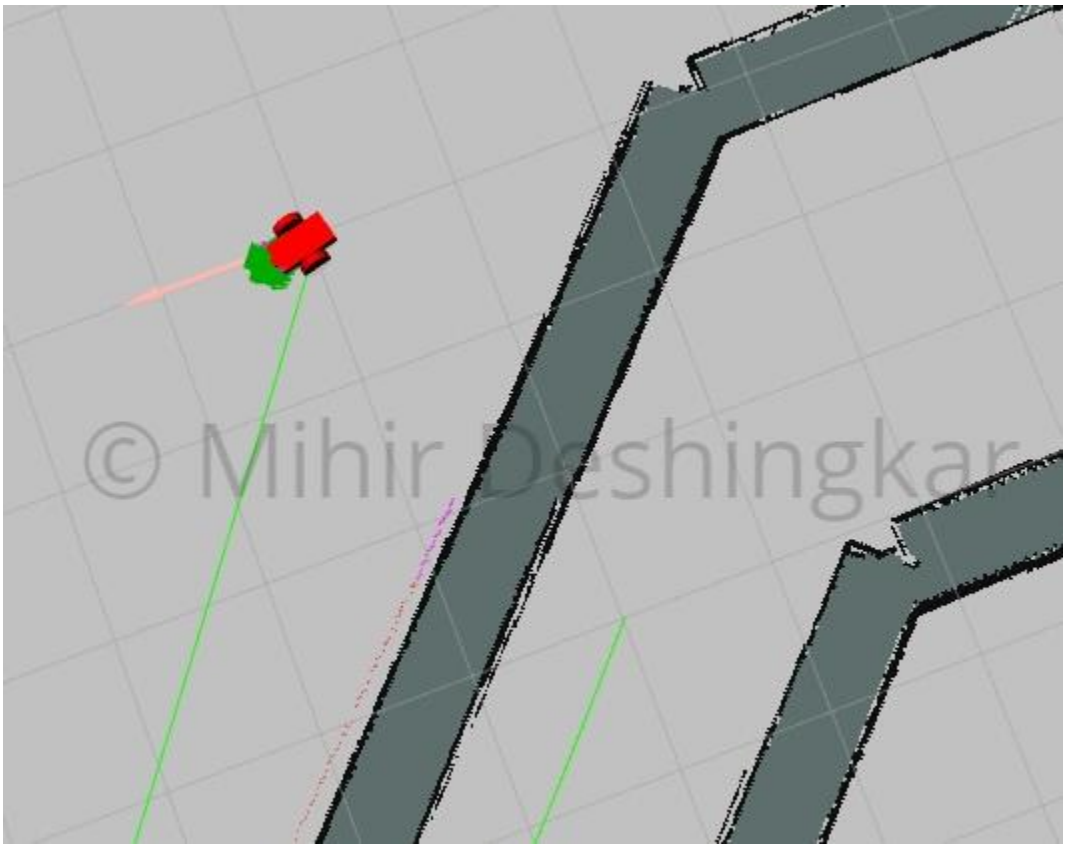
MCL algorithm has certain advantages over EKF as given below:

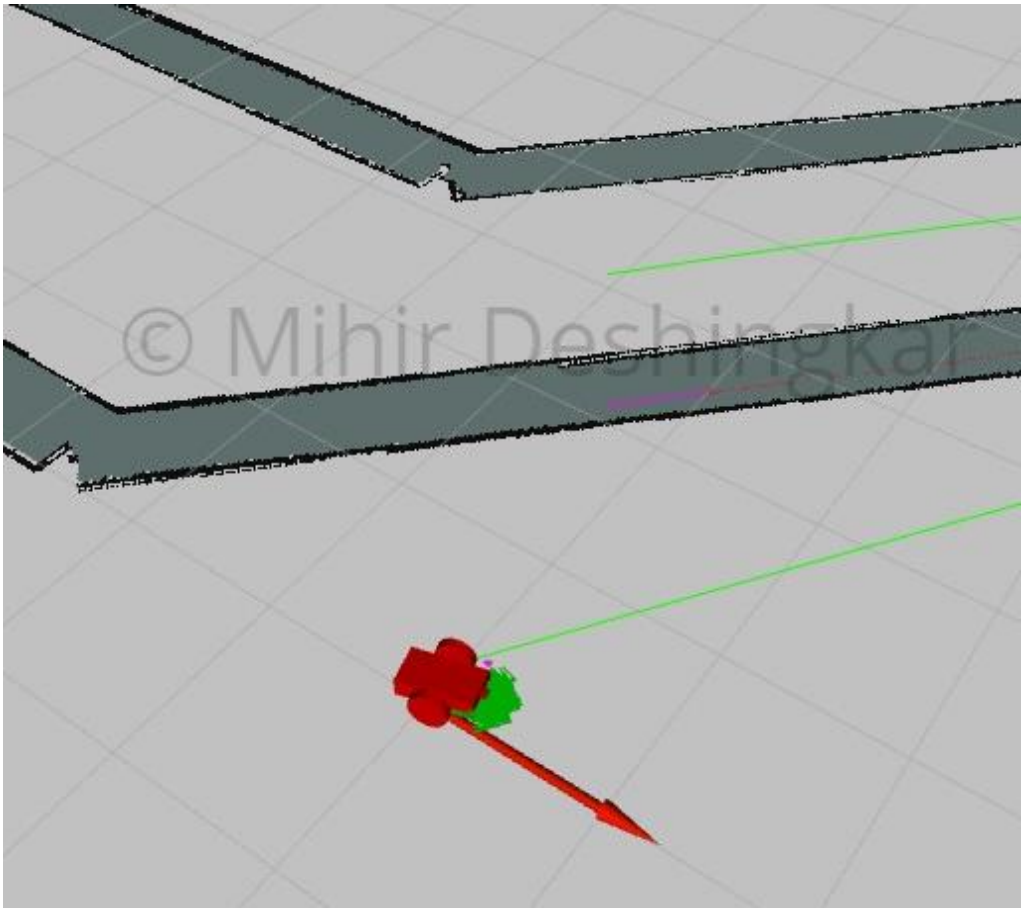
1. Easier to implement
2. MCL represents non-gaussian distributions and can approximate any other practical distribution
3. Computational memory and resolution of the solution by simply changing the number of particles

For the above advantages, many real-world robots today implement MCL. In this project, Adaptive Monte Carlo Localization (AMCL), which is a variation of MCL is used. The AMCL dynamically adjusts the number of particles over a period of time, and consequently offers a significant computational advantage over MCL.

Results:

Udacity robot at goal position –



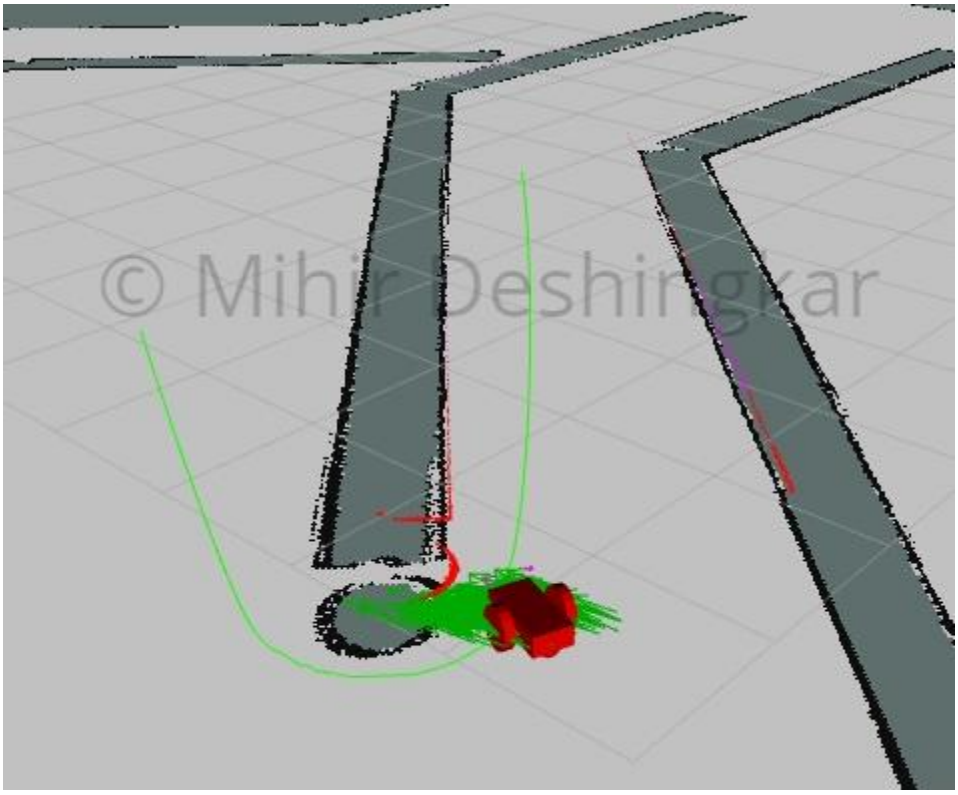


My robot at goal position –

Model Configuration:

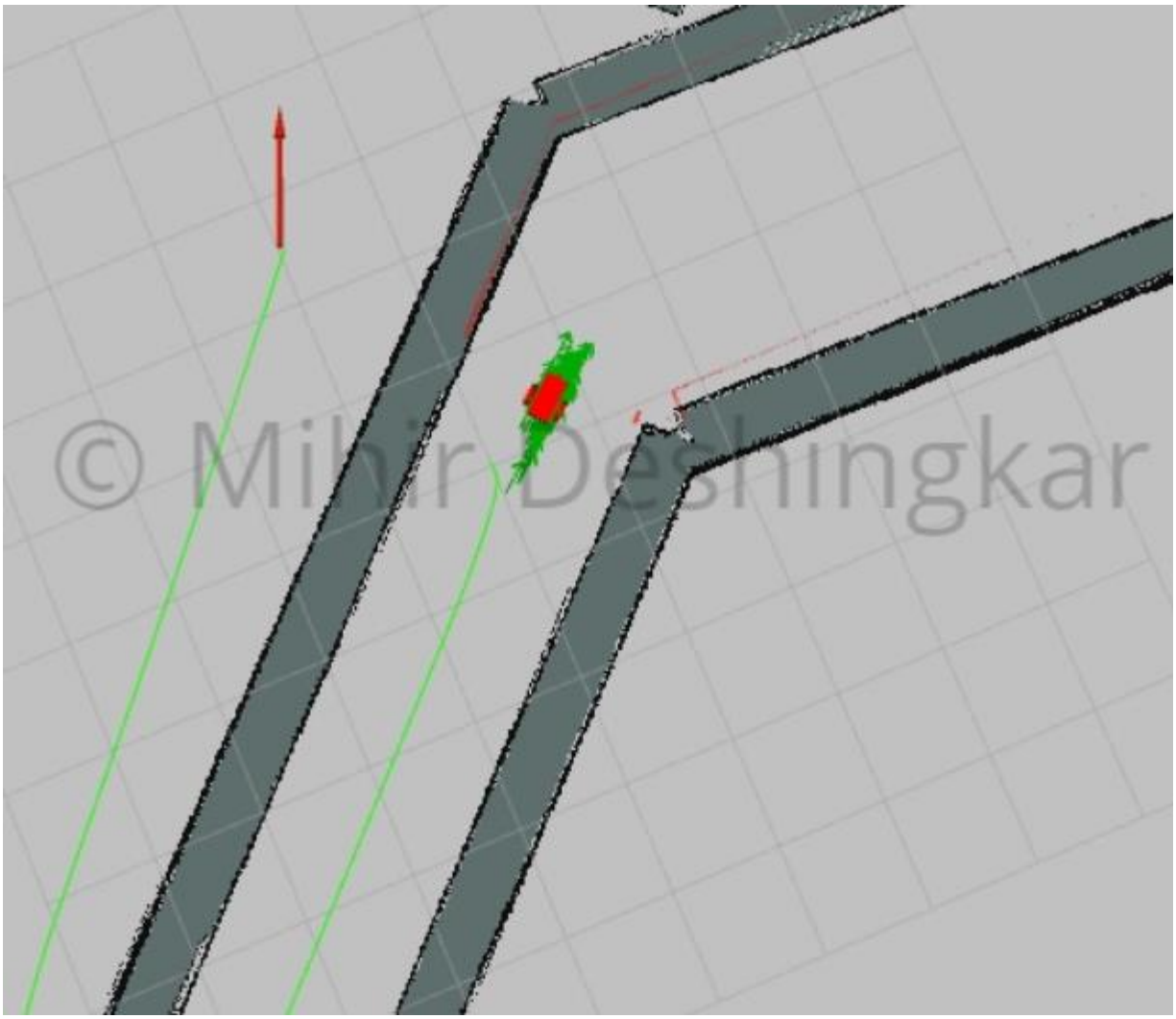
Initially, the robot was not moving. The problem was solved by changing the transform tolerance parameter. Next, the focus was put on the localization problem of reducing the particle cloud as the robot moved. For this, various AMCL parameters were tweaked. However, the robot still had difficulties in two main areas.

Firstly, the robot would get stuck during turning around an obstacle. This is shown below –

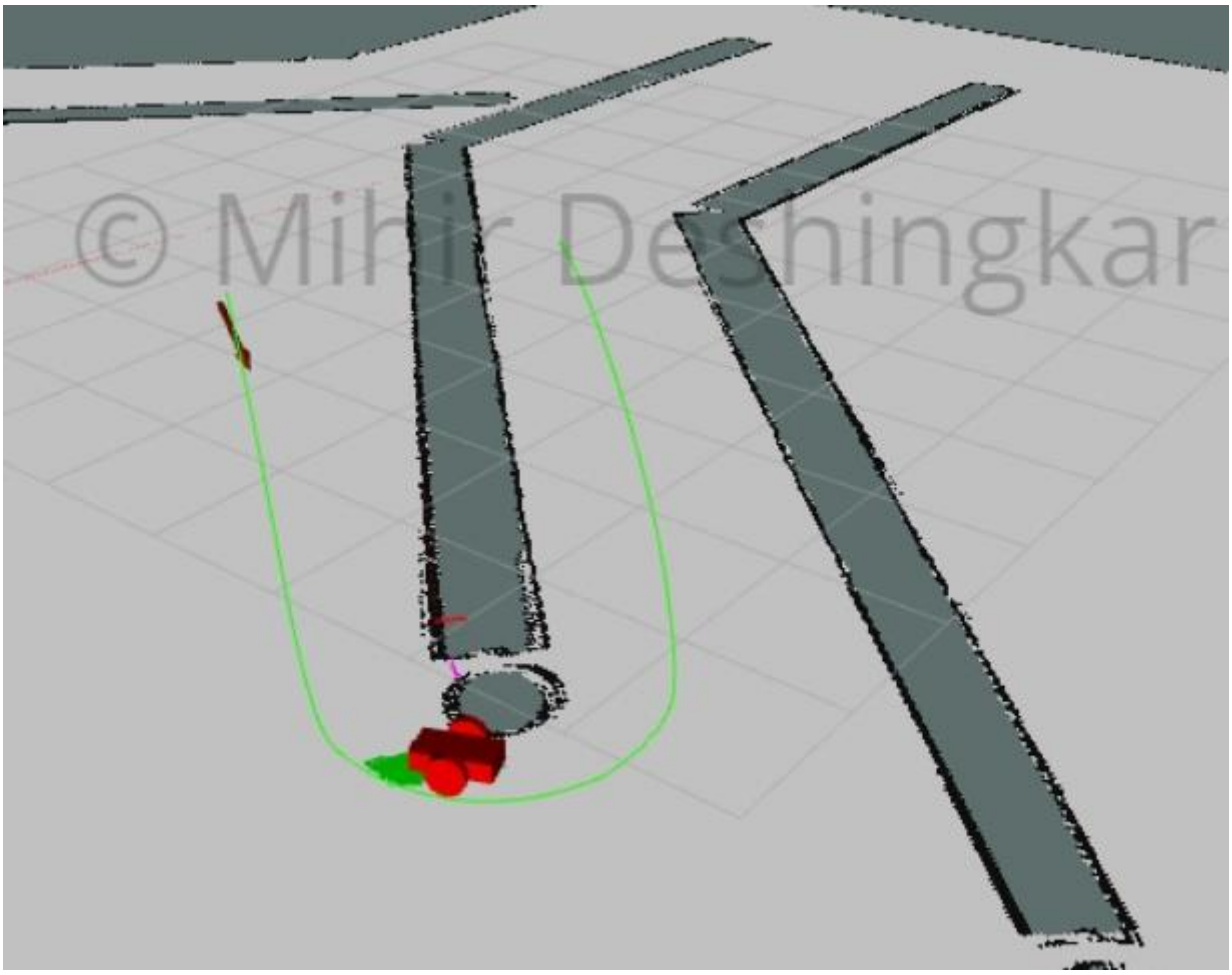


To solve this, various methods such as increasing the map size in local costmap parameters file, increasing the resolution by changing the resolution of local costmap from 0.05 to 0.02 were tried. Additionally, the map publish and subscribe frequencies were reduced, obstacle_range was increased, raytrace_range was tweaked, robot radius and inflation radius were tweaked, min and max particles were tweaked, update_min_d and update_min_a were reduced, odom alphas were reduced (amcl_parameters). This helped the robot to take a U-turn around the obstacle to reach goal position on the other side. However, the robot did not consistently reach the goal position. It rotated when taking the U-turn.

Secondly, for certain goal positions as shown below, the robot travelled in the opposite direction of the desired path and showed a random behaviour.



After tweaking many relevant parameters in base_local_planner.yaml such as sim_time, min_vel_x, max_vel_x, min_vel_theta, max_vel_theta, heading_lookahead, heading_scoring, pdist_scale, gdist_scale, occdist_scale it was found that reducing the width and height of local costmap in local_costmap_params.yaml solved the issue. The value was reduced from 30 to 2. The robot travelled sufficiently close to the global path and also took smooth turn around the obstacle to reach the desired goal position. This is illustrated in image below –



The significance and effect of various AMCL and move_base parameters is explained below.

AMCL parmaters (amcl_params.yaml)

1. transform_tolerance:
This is the time with which to post-date the transform that is published to indicate that this transform is valid. When the navigation was first launched, the robot did not move to goal position. This was because the transform_tolerance value was high (0.2). It started moving after changing the transform_tolerance to 0.1. Ultimately, the value was set to 0.12 for optimal performance.

2. min_particles and max_particles:
These parameters specify the range for the number of particles used for localization. When the robot is highly uncertain about its position, the robot prediction as particles equal to max_particles. As the robot localizes itself, AMCL reduces the particles. However, the number of particles will always be greater than or equal to min_particles. The value of max_particles was reduced to reduce the computations. The value of min_particles was reduced to reduced the final number of particles. The final value for these is min_particles = 10 and max_particles = 1000.

3. first_map_only:
When this parameter is set true, the AMCL will only use the first map that it subscribes to, rather than updating it each time a new one is received. This parameter was set true. A significant improvement was observed in the stability in Rviz. When the value of this parameter was false, the map in Rviz flickered a lot.

4. odom alphas:
odom_alpha1 and odom_alpha2 specify the noise in odometry's rotation estimate from robot's rotational and translational motions respectively. odom_alpha3 and odom_alpha4 specify the noise in odometry's translational estimate from robot's translational and rotational motions respectively. Only after significantly reducing the values of odom_alpha1, odom_alpha2, odom_alpha3 and odom_alpha4, the particle cloud started to reduce in size as the robot moved. Optimum value for all was found to be 0.001.

5. update_min_d and update_min_a:
The update_min_d and update_min_a are respectively the translational and rotational movements required before performing filter update. Reducing these values increases the required processing. The values of these parameters were changed from 0.2 and pi/6 to 0.05 and pi/12 respectively. The result was that the particle cloud size reduced very quickly in short distance.

6. resample_interval:
This specifies the number of filter updates required before performing resampling. Reducing the value of this parameter from 2 to 1 also helped in speeding up the reduction of particle cloud size consequently speeding up localization.

move_base Parameters

Local and Global Costmap Parameters (local_costmap_paramas.yaml, global_costmap_paramas.yaml)

1. update_frequency and publish_frequency:
The update_frequency and publish_frequency specify the frequency in Hz with which the map is updated and published respectively. For both local and global costmaps, these two frequencies were set to 5. For higher frequencies (10Hz and above), warning messages 'map update loop missed its desired frequency' were displayed as the it reached the computation limitation.

2. width and height:
These are the width and height in meters of either local or global costmapFor global costmap. These parameters were set to 40 for global costmap and 2 for local costmap.
3. Resolution:
This is the resolution of the map. Higher the resolution, more computational capacity is required. For higher resolutions, such as 0.02, the warning message "control loop missed the desired frequency" was continuously displayed. At resolution of 0.05, these warning messages did not occur.

Common Costmap Parameters

1. obstacle_range:
The obstacle_range parameter determines the maximum range sensor reading that will result in obstacle being updated into cost map. Obstacle range was increased to 5.0 from its default value of 2.5. This helped the robot plan a better path and move smoother, as the obstacles up to 5 meters distance from the robot were updated.

2. raytrace_range:
The raytrace_range sets the distance which the robot tries to clear in front of it from the robot base. To avoid collision with obstacles and keep safe distance from the obstacles, this parameter was set to! 0.5.

3. robot_footprint:
This parameter is used to set the footprint of the robot. In case the robot is circular, robot_radius parameter is set instead of robot footprint. Robot footprint was set to [[-0.2, 0.2], [0.2, 0.2], [0.2, -0.2], [-0.2, -0.2]] in order to include the wheels. When the wheels were not included in the footprint, the robot did not reach the target location.

4. robot_radius:
This parameter is set in case of circular robots. However, the rectangular robot was also tested by setting this parameter. Even though robot is not circular, the robot reached the target when an optimum radius of 0.3 was set to include the wheels. Reducing this radius to 2 resulted in irregular robot movements.

5. inflation radius:
This is the distance from the obstacles up to which the robot will treat the cost of the path equal to that of obstacle. This value was set to be 0.4 to sufficiently stay away from the obstacles. When the value was reduced to 0.2, the robot did not turn smoothly around the corners and started rotating.

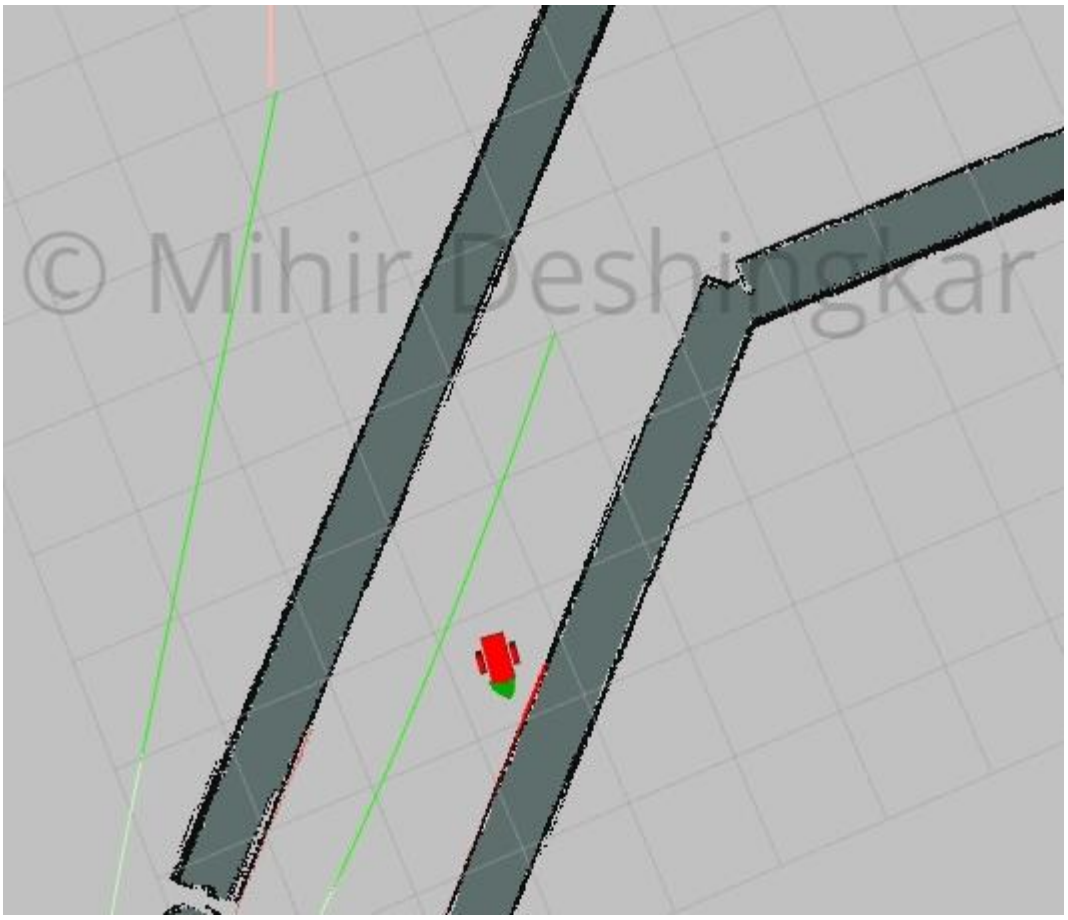
Base Local Planner Parameters

1. sim_time:
This parameter sets the amount of time to forward-simulate the trajectories in seconds. The optimum value for this parameter was found to be 1.0. Either increasing or decreasing its value even by a small amount resulted in unstable system.

2. yaw_goal_tolerance and xy_goal_tolerance:
These two parameters determine how accurately the robot reaches the given goal. To achieve the goal, sufficiently smaller values of 0.05(rads) and 0.1 were used for ya_goal_tolerance and xy_goal_tolerance respectively.

3. meter_scoring:
This parameter determines whether the gdist_scale and pdist_scale parameters should assume that the goal_distance and path_distance are expressed in units of meters or cells. Meter scoring was set to true to make the robot robust against changes of costmap resolution.

4. pdist_scale, gdist_scale, occdist_scale:
The pdist_scale parameter determines how closely the robot follows the global path. The gidist_scale parameter determines how closely the robot follows local path. The occdist_scale determines how strongly the robot stays far from obstacle. Increasing the value of pdist_scale caused the robot move in a small zig zag motion. When the value was decreased, the robot strayed away from the given path. This effect is demonstrated in image below.



Increasing the gdist_scale resulted in robot being stuck at the starting position. Therefore, it was set to default value of 0.01. Below is a table which shows the effect of pdist_scale and gdist_scale on the robot.

Sr. No.	**pdist_scale**	**gdist_scale**	**Result**
1	0.6	0.8	Reached goal
2	1.6	1.0	Reached goal
3	5.0	2.0	stuck
4	5.0	5.0	Reached goal (slightly zig-zag path)

5. min_vel, max_vel:
This is the minimum forward velocity assigned for the base in meters per second. It should large enough to ensure that the robot overcomes the friction and actually moves. This was kept to a default value of 1.0. Increasing this velocity to higher values (for example, 2.0) resulted in robot being drifted away from the desired global path. The max_vel parameter is the maximum velocity of the robot. At its default value of 0.5, the robot reached the goal. Increasing it to 0.8 helped in reaching the goal faster. However, when this parameter was further increased to 1.0, the robot travelled closer to the global path.

6. min_theta, max_theta:
These parameters set the minimum and maximum angular velocities for the robot. The min_theta value was set to be a default of 0.5. The max_theta parameter value had to be increased to 3.0 for the robot to start moving from the starting position. At its default value, the robot was stuck at its starting position.

My Robot:
This section describes the design changes made in the Udacity's provided robot and the parameters that were changed to achieve the localization. Also note that the launch commands are changed as below –

1. roslaunch mihir_bot mihir_world.launch
2. roslaunch mihir_bot amcl.launch
3. rosrun mihir_bot navigation_goal_mihir

Design Changes –
The robot box size is 0.5x0.3x0.2. The wheels are in front by 0.1 (r=0.2) and only one caster wheel is used behind at -0.15 to balance the robot (r=0.6). Also, sensor locations are changed. Camera is to the right of Y-axis of base by 0.05 (0.25, 0.05, 0.0) and laser is to the left of Y-axis of base by 0.05. Despite asymmetric sensor locations, robot was able to localize successfully. To achieve this localization, following parameters were changed.

List of changed parameters from the previous robot –

1. Max_particles:
Reduced from 1000 to 500 to remove warning messages.

2. inflation radius:
Reduced from 0.4 to 0.25. The main reason for this is that the robot size is now increased and therefore for inflation radius of 0.4, the robot did not move from the initial position. As the robot is initially facing the wall, a larger inflation radius will lead robot to perceive that it is already in the inflation radius range and therefore is hitting the obstacle.

3. max_vel_x:
This parameter was reduced to 0.6 from 0.8. For higher values, the robot did not move properly or did not move at all.

4. max_vel_theta:
This parameter was reduced to 0.2 from 0.3 for optimal performance.

5. acc_lim_x and acc_lim_y:
These parameters were reduced to 2.0 from 2.5 which resulted in better path following.

6. pdist_scale:
The pdist_scale parameter was changed to 1.6 from 0.6.

transform_tolerance:
This parameter was set to 0.11 to avoid all the warning messages 'control loop missed its desired frequency'.

odom_alpha1 and odom_alpha4:
Both these parameters were set to 0.005 (previously 0.001). Reducing these parameters resulted in undesirable robot behaviour, such is it started rotating randomly. The probable reason for this could be that this robot inherently causes some translation as it cannot purely rotate.

robot_footprint:
This was changed to [[-0.3, 0.3], [0.3, 0.3], [0.3, -0.3], [-0.3, -0.3]] to include the entire robot.

Discussion:

Both robots successfully achieve their main goal of reaching the target locations and also achieving the desired orientation. The particle size is sufficiently small and concentrated around the robot at goal locations. For 'my robot', the particle size and spread is slightly larger than the Udacity's robot because the robot cannot turn around the vertical axis passing through the centre of mass of the base. As a result, when the robot reaches the goal position and turns to achieve the desired orientation, inherent translational motion is introduced. Therefore, the robot has to again perform some translational motion to reach the goal position. This results in slightly greater particle size when the robot reaches the goal.

A square robot footprint was used to reduce the processing time (as compared to rectangular footprint). In achieving the goal, various AMCL and Move Base parameters were tuned and their effect on robot was understood. However, the robots slightly deviate from the global path provide. This can probably be avoided by better tuning of pdist_scale and gdist_scale parameters.

The current localization for the robots would not work for kidnapped robot scenario. This is because the particles are currently spread around the robot. Therefore, if the robot is kidnapped far from its previous belief, there will be no particles in that region where the robot actually is. This problem might be solved using wider particle spread to cover entire map.

AMCL algorithm will face many challenges when used in industries. One of the reasons is the required computational capacity. In industries, errors are expensive. Also, robot will be surrounded by environment with humans and therefore safety is utmost important. For these reasons, the robot localization accuracy needs to be really high. Consequently, a high-resolution map will be required, and the particle resolution needs to be increased as well. Therefore, large on board computational capacity will be required. Furthermore, the map in industries will keep on changing. Therefore, the robot will also need to update the map along with the localization.

Future Work:

Various ways to improve the robot accuracy and decrease processing time are discussed below.

1. The robot accuracy can be increased by increasing various parameters such as max_particles, map_update_rate, map resolution. However, this will result in increased computations which can lead to errors in robot localization if the system does not have the required computational capacity. Therefore, a trade off must be found between localization accuracy and computational capacity.

2. The robot localization accuracy can further be increased by adding more sensors. More sensors will give more data about the surroundings. By combining data from all these sensors, the

localization accuracy will be increased. However, more sensors will also need more computational capacity to process and combine their data.

3. The processing time can be reduced by changing the robot footprint. For example, using a square or a circular footprint results in increased processing speed. Comparatively, the rectangular robot footprint requires much more processing time.

Reference Links:

1. AMCL parameters- http://wiki.ros.org/amcl#Parameters
2. Local and Global costmap parameters- http://wiki.ros.org/navigation/Tutorials/RobotSetup#Costmap_Configuration_.28local_costmap.29_.26_.28global_costmap.29
3. Common Costmap parameters- http://wiki.ros.org/costmap_2d#costmap_2d.2BAC8-layered.Parameters
4. Base local planner parameters- http://wiki.ros.org/base_local_planner#Parameters
5. Optimizing trajectory and computational time - http://wiki.ros.org/teb_local_planner/Tutorials/Frequently%20Asked%20Questions
6. https://pdfs.semanticscholar.org/760e/0a4ff74275294a8c55c4548e0f712f34c262.pdf