Robot Localization Using AMCL ROS Package: "Where I Am?" Project

Navid Safaeian

Abstract—The goal of this project is to build a ROS Navigation stack (package) including localization using **Adaptive Monte Carlo Localization (AMCL)** and configure a number of ROS packages that can be used in conjunction to accurately localize and navigate a mobile robot inside a provided map in the Gazebo and RViz simulation environments. Robot localization and navigation to a goal location is the process of determining where a mobile robot is located with respect to its environment and how can send a goal location to the mobile robot without hitting the map obstacles. Localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robots own location is an essential precursor to making decisions about future actions. In this paper Extended Kalman Filter (EKF) localization algorithm will be compared to Adaptive Monte Carlo localization (AMCL) algorithm then AMCL performance will be tested in two different simulated robot configurations.

1 Introduction

This project is conducted by using ROS Navigation Stack to accurately localize a robots pose and orientation as the robot navigates through a maze and successfully reaches its desired destination in simulation. The Adaptive Monte Carlo Localization (AMCL) method was used to successfully achieve this challenge derived from noisy sensor readings from the robot and flawed odometry of motor and maybe undesired environment conditions. Localization algorithms can helps with the robot filter out the noisy sensor reading and leading to correctly modification of the robot localization within a given map environment (Fig. 1). Exploring, adding, and tuning parameters for the AMCL and ROS move_base packages are some of the main goals of this project. To properly test this goal, the Adaptive Monte Carlo Localization was applied to two robots in simulation. One that is predefined for the project, udacity_bot, therefore setting the benchmarks for performance, and the other was designed as a my custom robot to meet or exceed the standards set which can be called as safa_bot.

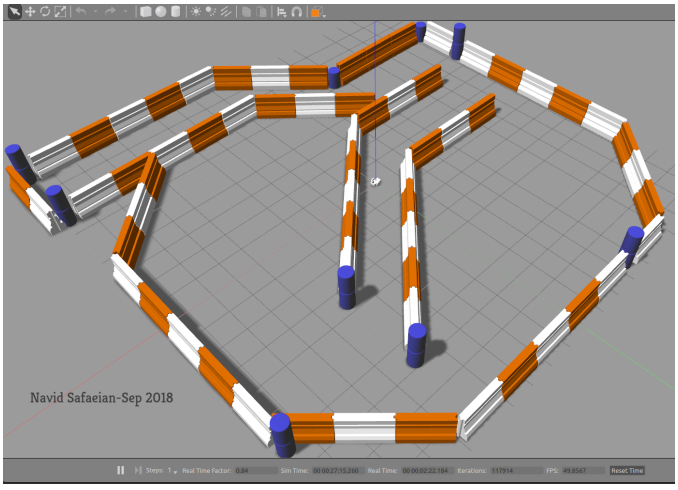

Fig. 1. Jackal Race Map

2 Background

There are several approaches for a robot to perform localization numerically. The most popular would be Markov localization, Grid localization, Kalman Filter localization, and Monte Carlo Localization. However, here in this project, two approaches will be further discussed: Kalman Filter and Monte Carlo Localization (Particle filter).
The localization problem for a robot on a map can be accurately found by filtering noisy sensor data using probabilistic methods. Localization problems assume to be categorized in three types: the simplest one is **Position tracking** in which the robots initial position is known and the algorithm keeps track of the robot's position as it moves. In the **Global Localization**, the initial position is unknown and needs to be determined at as the robot moves. This problem combines the uncertainties from measurements and actions in a cycles to achieve a precise estimate of the location.
Finally, the hardest localization problem is called **Kidnaped robot** which commonly refers to a situation where an autonomous robot in operation is carried to an arbitrary location. This is particularly difficult for bayesian algorithms to recover from since they preserve an internal belief that interfere with the new robot state.

2.1 Kalman Filters

The Kalman filter has long been regarded as the optimal solution to many tracking and data prediction tasks. Its use in the analysis of visual motion has been documented frequently. Kalman filter (KF) localization, is a linear quadratic estimator that uses a Gaussian probability density representation of robot position and scan matching for localization. Unlike Markov localization, Kalman filter localization does not independently consider each possible pose in the robots configuration space. Interestingly, the Kalman filter localization process results from the Markov localization axioms if the robots position uncertainty is assumed to have a Gaussian form. The disadvantage of KF is the assumption that motion and measurements are linear, and the assumption that state space can be represented by

a unimodal Gaussian distribution. Both assumptions are not true in most of robots hence a non-linear algorithm will be a better fit for mobile robots. Most mobile robots will execute non linear motion like a curve in Fig. 2. Non linear actions will result in non-Gaussian posterior distributions that cannot be properly modeled by a closed form equation.

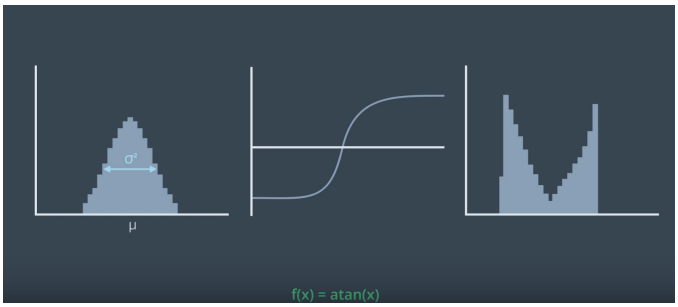

Fig. 2. Gaussian prior (left) subject to a non-linear action ($atan(x)$-center) results in a non-Gaussian posterior (right).

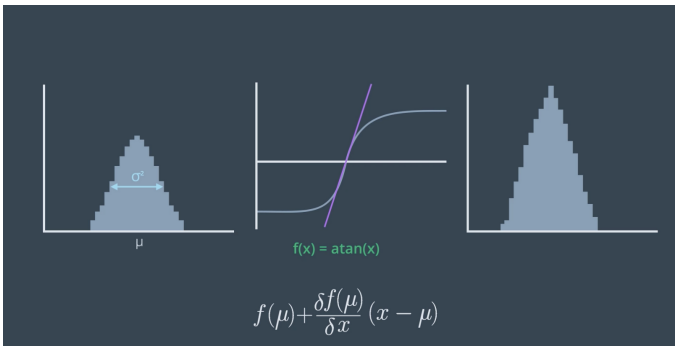

Fig. 3. Linear approximation (in purple) of the $atan(x)$ function around x = 0

The Extended Kalman Filter, or EKF, linearizes this non-linear problem via Taylor Expansion. This approximates the nonlinear function by a linear function tangent at the mean of the Gaussian (Fig.3). The linear estimate is only valid for a small section of the nonlinear function, however, if the linear function is placed at the best estimate, the mean, and updated at every step, the resulting estimates becomes truer. Thus, the EKF inherits from the Kalman Filters fundamental belief representation but the EKF differs in that its certainty is only an estimate rather than correct, as the case with the Kalman Filter.

2.2 Particle Filters

The Particle filter algorithm also known as the Monte Carlo Localization (MCL) is the most popular localization algorithm in robotics. This algorithm uses particles to localize the robot. Each particle has a position and orientation, representing the guess of where the robot is positioned. These particles are re-sampled every time the robot moves by sensing the environment through range-finder sensors, such as lidars, sonars, RGB-D cameras, among others. After a few iterations, these re-sampled particles eventually converge with the robots pose, allowing the robot to know its location and orientation. The MCL algorithm can be used for both Local and Global Localization problems, and is not limited to linear models [1].

2.3 Comparison / Contrast

EKF is great for position tracking problems and work well if the position uncertainty is small. If the EKF is implemented for the right system, it is more time and memory efficient compared to the MCL. However, the EKF performs really poor in more general global localization problems. Global localization is where the MCL shines, making it more robust than the EKF. By also adding random particles in the particle set, it also solves the kidnapped robot problem. The accuracy and computational costs for the MCL can be tweaked through setting the size of the particles. EKF, on the other hand, has great accuracy but is heavy in resources due to the matrix computations in the algorithm. The figure below (Fig 4) summarizes the comparison of the two implementations:

	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

Fig. 4. Comparing MCL with EKF

3 Simulations

In this project, ROS packages are implemented to accurately localize the two simulated robots to test performance of AMCL package inside a custom map in the Gazebo and RViz simulation environments. The ROS implementations include building a mobile robot for simulated tasks, creating a ROS package that launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack, and tuning specific parameters corresponding to each package to achieve the best possible localization and navigation results. The navigation stack is based on the mobile robot configurations in a particular manner in order to run. The movement was driven by a C++ node that provided navigation goal.

3.1 Achievements

The two simulated mobile robots (benchmark model and the own model) navigate and reach the specific goal position by using provided C++ node while localizing itself along the way. The tuning process for the parameters regarding ROS AMCL and move base packages were a long and iterative one, but ultimately it enabled the robots with the localization and navigation capability.

3.2 Benchmark Model: Udacity_bot

3.2.1 Model design

The mobile robot (Udacity_bot) model is built by creating its own Unified Robot Description Format (URDF) file. For this model, a cuboidal base (box size=.4 .2 .1) with two caster wheels is created (Fig. 5). The caster wheels will help stabilize this model and it can lead to uniform weight distribution, and then the robot is prevented from tilting along the z-axis at times. There is a single link, with the name defined as chassis, encompassing the base as well as the caster wheels. Two wheels are added to the robot which each wheel is represented as a link and is connected to the base link (the chassis) with a joint. For this robot, two sensors, a camera and a laser rangefinder, are added. Table 1 and Table 2 show some of the specifications for the sensors and body.

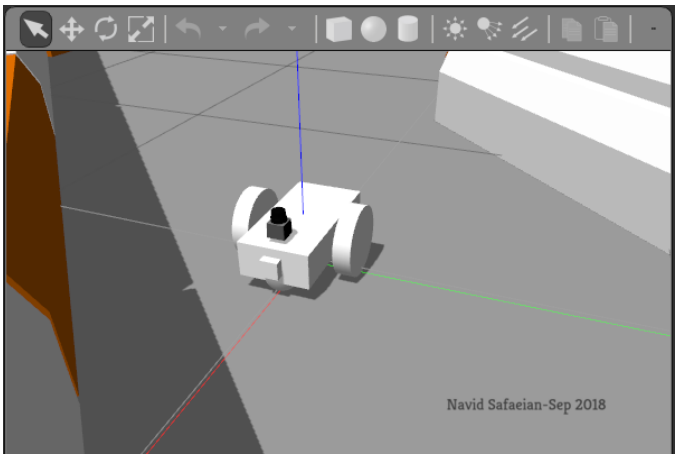

Fig. 5. Udacity_bot

TABLE 1
Camera Sensor and Laser Sensor: Udacity_bot

Camera Sensor	
link name	camera
link origin	[0, 0, 0, 0, 0, 0]
joint name	camera joint
joint origin	[0.2, 0, 0, 0, 0, 0]
geometry	box with size 0.05
joint parent link	chassis
joint child link	camera
Laser Sensor	
link name	hokuyo
link origin	[0, 0, 0, 0, 0, 0]
joint name	hokuyo joint
joint origin	[0.15, 0, 0.1, 0, 0, 0]
geometry	box with size 0.1 for collision
geometry	a mesh for visual
joint parent link	chassis
joint child link	hokuyo

3.2.2 Packages Used

In order to launch the mobile robots, a ROS package was created. The robot package structure was designed as shown below. Udacity_bot Package includes **meshes**, **URDF**, **worlds**, **maps**, **rviz**, **src**, and **config**.

TABLE 2
Udacity_bot Body

Udacity_bot Body		
Part	Geometry	Size
Chassis	Cube	0.4 x 0.2 x 0.1
Back and Front Casters	Sphere	0.0499 (radius)
Left and Right Wheels	Cylinders	0.1 (radius), 0.05 (length)

This robot package, along with the Navigation Stack and AMCL packages were crucial for a complete simulation of a mobile robot performing and successfully solving the localization problem.

3.2.3 Parameters

To obtain most accurate localization results, several parameters were added, tested and tuned. These parameters are related to AMCL package which localize the robot [2], and move base package which helps navigate the robot to the goal position by creating or calculating a path from the initial position to the goal.
Table 3 shows Localization parameters in the AMCL node. For the project, min particles is set to 10 and max particles is set to 300 as an input. This range is tuned based on the system specifications. Table 4, Table 5 and TABLE.5 show move base parameters in the configuration file. In Table 4, Local or Global costmap parameters specify the behavior associated with the local (or global) costmap. In Table 5, the laser sensor is defined as the source for observations for the list of parameters common to both types of costmaps. In Table 6, the set of parameters in the configuration file customize the particular behavior. The base local planner package provides a controller that drives a mobile base in the plane [3].

TABLE 3
AMCL Parameters: Udacity_bot

Overall Filter	
min particles	10
max particles	300
initial pose for x, y, a	0.0, 0.0, 0.0
Laser	
laser model type	likelihood field
laser z hit	0.997
laser z rand	0.003
laser max beams	90
Odometry	
odom model type	diff
odom alpha1	0.2
odom alpha2	0.2
odom alpha3	0.2

For the project, max velocity x is set to 0.5 and min velocity x is set to 0.1 as an input. For more information, the ROS Wiki Page provides in-depth descriptions of each and every one of the parameters used for this project as well as other parameters that can be explored further.

TABLE 4
Local and Global Costmap Parameters: Udacity_bot (benchmark model) & Safa_bot (personal model)

Local costmap	
global frame	odom
robot base frame	robot footprint
update frequency	15.0
publish frequency	15.0
width	5.0
height	5.0
resolution	0.05
static map	false
rolling window	true
global costmap	
global frame	map
robot base frame	robot footprint
update frequency	15.0
publish frequency	15.0
width	20.0
height	20.0
resolution	0.05
static map	true
rolling window	false

TABLE 5
Common Costmap: Udacity_bot

Common Costmap	
map type	costmap
obstacle range	2.5
raytrace range	3.0
transform tolerance	0.2
footprint	[[.2, .2], [.2, -.2], [-.2, -.2], [-.2, .2]]
inflation radius	0.2
observation sources	laser scan sensor
laser scan sensor	sensor frame: hokuyo, data type: LaserScan, topic:/udacity_bot/laser/scan, marking: true, clearing: true

3.3 Personal Model: Safa_bot

3.3.1 Model design

A mobile robot (Safa_bot) is shown in Fig. 6 that is built for simulated tasks. The model was developed based on new Unified Robot Description Format file (URDF). In this file, the size and geometry of the robot is designated and set to have physical configuration such as inertial and collision parameters. The Safa_bot has a cylindrical shape chassis with a radius of 0.2 and length of 0.1. Two casters (spherical shape) provide a balance distribution with a radius of 0.0499, and two wheels with a cylindrical shape of 0.1 radius and a length of 0.05 that are represented as individual links (Fig. 6).
The two wheels are connected to the chassis via continuous joints, meaning they are free to rotate around the joint axis. The Safa_bot is also equipped with two on-board sensors, a camera and a laser range-finder. All detailed specification are shown in Table 7 and Table 8.

3.3.2 Packages Used

Safa_bot is based on Udacity_bot. For mobile robot Safa_bot, no ROS package had to be built and it is hosted within the udacity_bot package such that it leverages most of the udacity_bot package.

TABLE 6
Base local Parameters: Udacity_bot

TrajectoryPlannerROS	
holonomic robot	false
meter scoring	true
yaw goal tolerance	0.05
xy goal tolerance	0.10
occdist scale	0.05
sim time	1.0
pdist scale	0.6

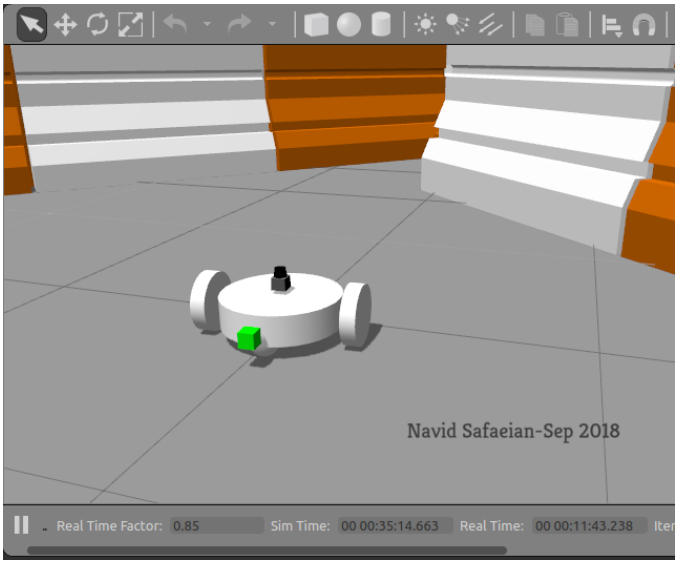

Fig. 6. Safa_bot

TABLE 7
Safa_bot Body

Safa_bot Body		
Part	Geometry	Size
Chassis	Cylindrical	0.2(radius), 0.1(length)
Back and Front Casters	Sphere	0.0499 (radius)
Left and Right Wheels	Cylinders	0.1(radius), 0.05(length)

3.3.3 Parameters

To obtain most accurate localization results, specific parameters of each package were added, tested and tuned. These parameters are related to AMCL package which localize the robot, and move base package which helps navigate the robot to the goal position by creating or calculating a path from the initial position to the goal. The Udacity_bot parameters of local and global costmap were reused for Safa_bot, although the ROS AMCL node parameters, the costmap common parameters, and base local planner parameters used was slightly different as the body and shape of Safa_bot was dramatically different.
Table 9 shows Localization parameters in the AMCL node. For the project, min particles is set to 10 and max particles is set to 200 as an input. This range is tuned based on the system specifications.

TABLE 8
Camera Sensor and Laser Sensor: Safa_bot

Camera Sensor	
link name	camera
link origin	[0, 0, 0, 0, 0, 0]
joint name	camera joint
joint origin	[0.25, 0, 0, 0, 0, 0]
geometry	box with size 0.05
joint parent link	chassis
joint child link	camera
Laser Sensor	
link name	hokuyo
link origin	[0, 0, 0, 0, 0, 0]
joint name	hokuyo joint
joint origin	[0, 0, 0.1, 0, 0, 0]
geometry	box with size 0.1 for collision
geometry	a mesh for visual
joint parent link	chassis
joint child link	hokuyo

TABLE 9
AMCL Parameters: Safa_bot

Overall Filter	
min particles	10
max particles	200
initial pose for x, y, a	0.0, 0.0, 0.0
Laser	
laser model type	likelihood field
laser z hit	0.95
laser z rand	0.05
laser max beams	30
Odometry	
odom model type	diff-corrected
odom alpha1	0.01
odom alpha2	0.01
odom alpha3	0.01

Table 10, Table 11 show move base parameters in the configuration file. In Table 10, the laser sensor is defined as the source for observations for the list of parameters common to both types of costmaps. In Table 11, the set of parameters in the configuration file customize the particular behavior. For the project, max vel x is set to 0.1 and min vel x is set to 0.08 as an input.

4 Results

Both robots reached the goal after tweaking parameters for ROS AMCL and move base packages. Udacity_bot and Safa_bot navigate to a specific goal position with a specified goal tolerance parameters in the ROS move base package. The navigation is achieved for localization.

4.1 Localization Results

At the start of the simulation, the both mobile robot's particles (in the benchmark and the own robot) were uniformly spread and after the iterative computational AMCL algorithm and re-sampling the particles eventually narrowed down to a small group of particles close to the robots as the simulated robots proceeded to the goal. At the goal, the robots particles had a tight grouping pointing towards the orientation of the goal. Fig. and Fig. show the both robots at the start position. Fig. and Fig. show the both robots at the goal position.

TABLE 10
Common Costmap: Safa_bot

Common Costmap	
map type	costmap
obstacle range	3.0
raytrace range	3.0
transform tolerance	0.3
robot radius	0.35
inflation radius	0.6
observation sources	laser scan sensor
laser scan sensor	sensor frame: hokuyo, data type: LaserScan, topic:/Safa_bot/laser/scan, marking: true, clearing: true

TABLE 11
Base local Parameters: Safa_bot

TrajectoryPlannerROS	
holonomic robot	false
meter scoring	true
yaw goal tolerance	0.05
xy goal tolerance	0.10
occdist scale	0.05
sim time	1.0
pdist scale	0.5
gdist scale	1.0
max vel x	0.1
min vel x	0.08
max vel theta	2.0
acc lim x	5.0
acc lim y	5.0
acc lim theta	5.0

4.1.1 *Benchmark: Udacity_bot*

Fig. 7 and Fig. 8 show the Udacity_bot position at the start position with uniformly random distribution of particles (PoseArray) and at the goal position with narrowed down particles (PoseArray) in RViz, repectively. It takes about 4 min for Udacity_bot to reach the goal with no collision with obstacles and takes almost a smooth path to the goal.

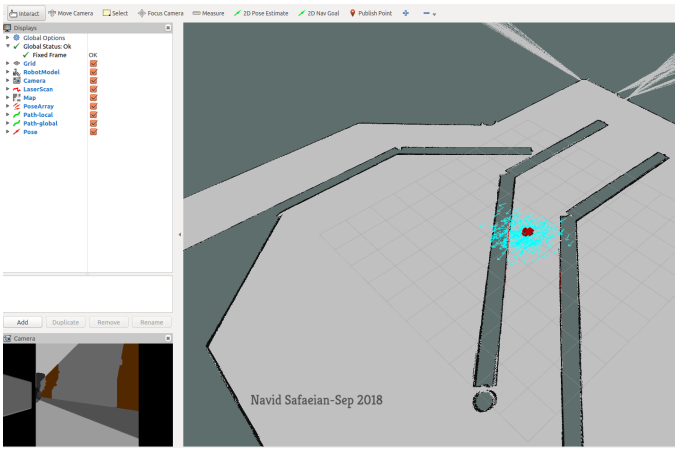

Fig. 7. Udacity_bot at the start position

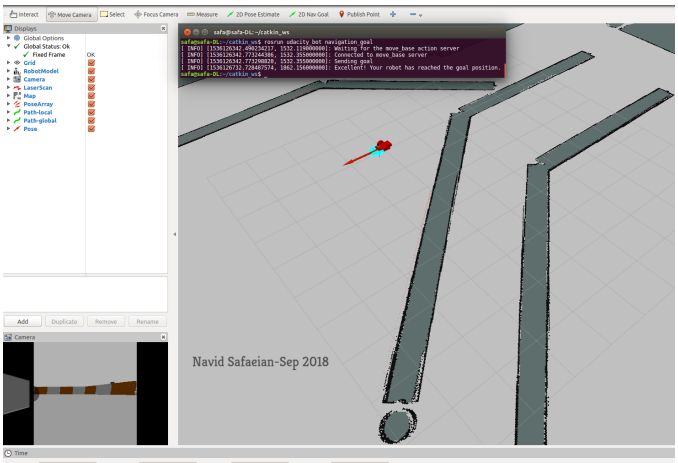

Fig. 8. Udacity_bot at the goal position

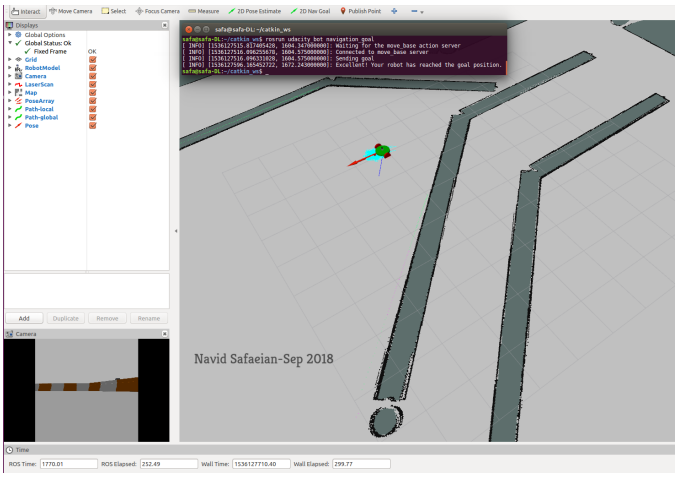

Fig. 10. Safa_bot at the goal position

4.1.2 *Student: Safa_bot*

Fig. 9 and Fig. 10 show the Safa_bot position at the start position with uniformly random distribution of particles (PoseArray) and at the goal position with narrowed down particles (PoseArray) in RViz, respectively. It takes about 4 min for Udacity_bot to reach the goal with no collision with obstacles and completely takes a smooth path to the goal.

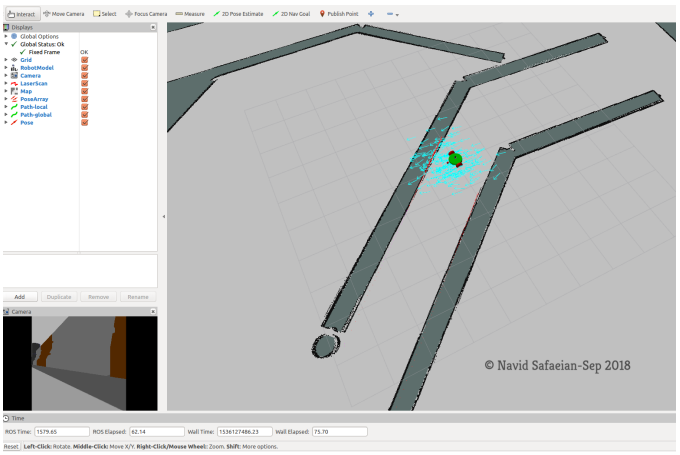

Fig. 9. Safa_bot at the start position

4.2 Technical Comparison

Safa_bot performs better than Udacity_bot. Both robots were able to smoothly navigate to the goal avoiding all obstacles in the world map, although Udacity_bot had a little distance with global path in its way to the goal and it takes about 4 min to reach the goal, but on the other hand Safa_bot takes about 1 min (much faster than Udacity_bot) to reach the goal with taking complete smooth path.

5 Discussion

5.1 Topics

- Safa_bot performance was better by considering the time required to reach goal and the trajectory taken by both robots.
- In kidnapped robot problem commonly refers to a situation where an autonomous robot in operation is carried to an arbitrary location. The kidnapped robot problem creates significant issues with the robot's localization system and it differs from the global localization problem in that the robot almost know of somewhere else at the time of the kidnapping. The kidnapped robot problem is often used to test a robots ability to recover from catastrophic localization failures. The small number of particles for noise filtering in regular MCL algorithm is not able to cope with the kidnapped robot issue. One of the solution could be increase in particle samples which leads to heavy computational problem or using more effective algorithm such as MCL/AMCL.
- The various types of scenario associate with localization problem such as industrial and commercial cleaning robot, warehouse robot, and performing inspection tasks in industrial and agricultural area.
- MCL/AMCL in an industry domain would used to localize robustly in manufacturing environment where mobile manipulators perform various tasks with high accuracy, such as painting, cutting, welding.

6 Conclusion / Future Work

In brief summary, both robots with two different design and URDF were able to reach the goal position using two different AMCL and move base nodes' parameters can be modified in both cases to achieve better results or make the robot ready for certain tasks/types of obstacles.
The next steps to this project is to implement the software on a physical robot using the Jetson TX2. This will be more challenging problem since it now involves hardware, software, and physical properties, such as friction and gravity, which will all be taken into account in designing a new robot or modifying the existing design and tweaking configurations. Another step could be implementing AMCL for 3-Dimensional localization problems (application in drone and arm actuator robots) by using a 3D Laser range-finder instead of the 2D Hokuyo sensor.

Some modification could be implemented in the model to increase accuracy and/or decrease processing time. For the trade-offs in accuracy and processing time, all that is needed is to benchmark some of the processing times and choose the parameters that best meets the constraints of processing time or accuracy required. Furthermore, the addition of more sensors and the modification of the robot base size also might be required for the improvement.

REFERENCES

[1] S. T. Dieter, Fox and W. Burgard, *LATEX: Probabilistic Robotics*. 2005.
[2] B. P. Gerkey, *LATEX: Ros wiki: Amcl package summary; http://wiki.ros.org/amcl*. Ros wiki, 2018.
[3] K. Zheng, *LATEX: Ros navigation tuning guide*. 2016.