Where Am I

Seyfi Gozubuyuk

Abstract—In this project, the aim is to utilize ROS packages to accurately localize a mobile robot inside a provided map inside a provided map in the Gazebo and RViz simulation environments. The overview of the tasks can are as follows, building two mobile robots for simulated tasks, creating a ROS package that utilizes AMCL package, exploring specific parameters.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization, ROS, AMCL, Robot Model.

1 Introduction

The problem is to localize the robot in a known map environment. It is critical to know the exact location of the robot, to create a path from the starting point to the goal. It is also crucial to localize the robot to prevent any collision with obstacles.
Noise from the sensors and the actuators makes it harder to estimate the robot position. There is a need for the location algorithms, to decrease the uncertainty caused by the noise. The most common algorithms are Kalman Filter and Monte Carlo Localization (Particle Filter). This project test the Adaptive Monte Carlo Localization with two different robot models. The Udacity Bot and the xbot. The project includes building the robot models and tuning the parameters. The environment contains Gazebo and RViz on ROS Kinetic. The ROS packages required to run this project are AMCL, move base, map server and navigation.

2 Background

The problem is to localize the robot, as mentioned earlier. The map is known; however, the sensors are not precise. There is noise to affect the measurements coming from the sensors, and the controls are not perfect to move the vehicle to the desired location. Therefore, we need to apply some techniques to use noise measurements.

2.1 Kalman Filters

Kalman Filters use noisy measurements and the actuation commands to estimate the location of the robot. There are two steps in Kalman Filters. They are predict and update. Predict step forecast the position after actuation command is applied, whereas the update step uses the measurements to update the location. Kalman filters need to have Gaussian distributions to work. Therefore only linear operations are allowed on the probability.
Since the robot model is not entirely linear, there is a need for transforming nonlinear equations to linear equations. Taylor Series is the method for linear approximation. With linearization, the Kalman Filter become Extended Kalman Filter. [1]

2.2 Particle Filters

Particle Filter, also known as Monte Carlo Localization, starts with randomly generating particles. Each particle represents a possible location for the robot. The steps are similar to the Kalman Filter. After the motion, the position of each particle updates, and the uncertainty increases. After the measurements come from the sensors, the algorithm performs another update and the uncertainty decreases. [2]

2.3 Comparison / Contrast

Particle Filter can work with any distribution whereas Kalman Filter only works with a Gaussian distribution. Particle Filter takes raw measurements; however, Kalman Filter requires landmarks. The posterior is particles for Particle Filter and Gaussian for Kalman Filter. Kalman Filter is more efficient and provides more resolution; on the other hand Particle Filter is easier to implement and more robust. Only Particle Filter has Memory and Resolution control and can provide a solution for Glocal Localization. Particle Filter has Multimodal Discrete state space, and Kalman Filter has unimodal continuous.

3 Simulations

The simulation environment consists of ROS (Kinetic), Gazebo and RViz. The ROS packages are AMCL, move base, map server, and navigation. There are two different robot models to test on the simulation.

3.1 The Udacity Bot

The Udacity Bot has the robot model that described in the lesson. The robot has a chassis, two casters, two wheels, a laser sensor and a camera. Figure 1 and Figure 2 shows the Udacity bot in Gazebo and RViz.

3.2 The xbot

The xbot is the slightly modified version of the Udacity Bot. It has a sensor base link which is in a rectangular box shape. This link is connected to the chassis and the laser sensor and the camera. Figure 3 and Figure 4 shows the xbot in Gazebo and RViz.

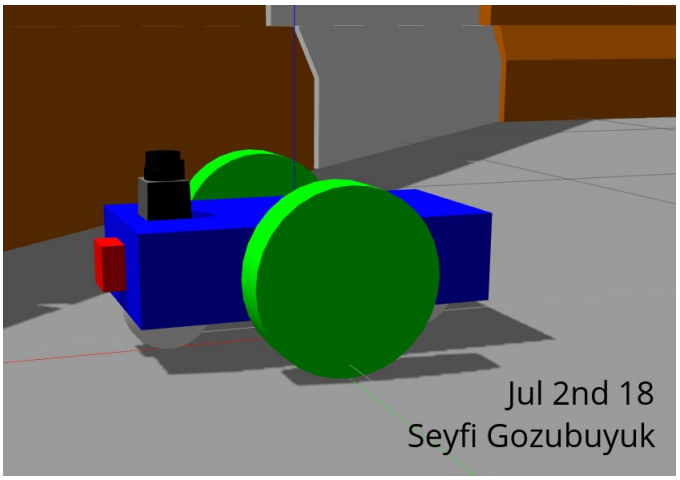

Fig. 1. Udacity Bot in Gazebo

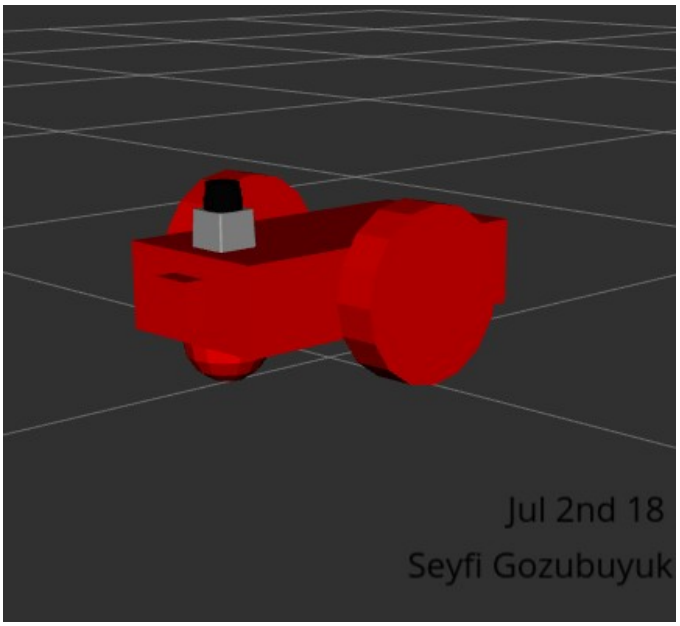

Fig. 2. Udacity Bot in RViz

3.3 Achievements

The Udacity bot and the xbot were able to reach the goal location. However, Udacity first went in the opposite direction to find a way to goal. After recognizing there was no way to goal, they turned back and the reach to the target. xbot directly moved in the correct direction; however, it moves in a curvy path.

3.4 Benchmark Model

3.4.1 Model design

The main part of the Udacity bot is the chassis. It has a box shape with dimensions 0.4 x 0.2 x 0.1. It has two casters, one at -0.15 0 -0.05 and the other at 0.15 0 -0.05. There are two wheels connected to the chassis at 0 -0.15 0 and 0 +0.15 0. // The laser sensor located at position relative to the chassis 0.15 0 0.085. It has a cubic shape with length 0.1. The camera sensor is connected to the chassis at 0.2 0 0, and it has a cubic shape with lenght 0.05. The difference between the

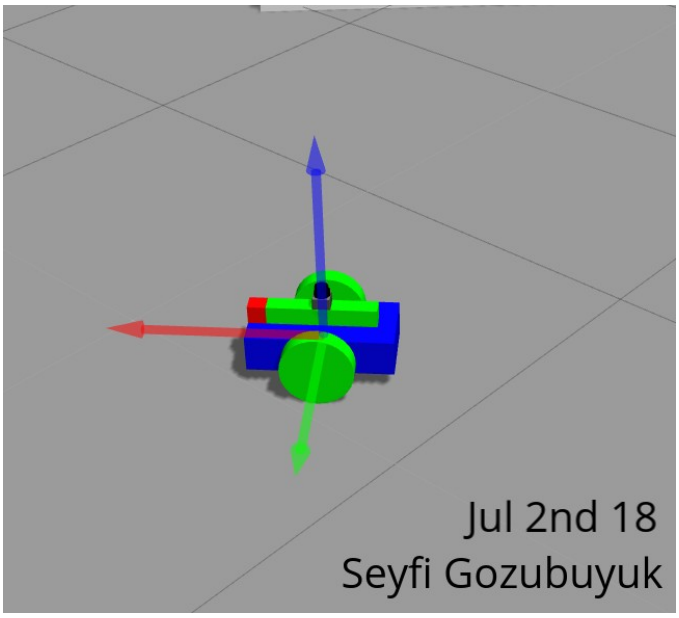

Fig. 3. xbot in Gazebo

Fig. 4. xbot in RViz

xbot and the Udacity bot is the link named sensor_base. The camera and the laser sensors are connected to this link. The link has the dimensions of 0.3 0.05 0.05. It is connected to the chassis at location 0 0 0.075. The camera is connected to sensor_base at 0.175 0 0 and the laser sensor connected to the sensor_base at 0 0 0.05.

3.4.2 Packages Used

The packages used are as follows:

- ros-kinetic-navigation
- ros-kinetic-map-server
- ros-kinetic-move-base
- ros-kinetic-amcl

Figure 5 and Figure 6 shows the topics for the Udacity Bot and the xbot.

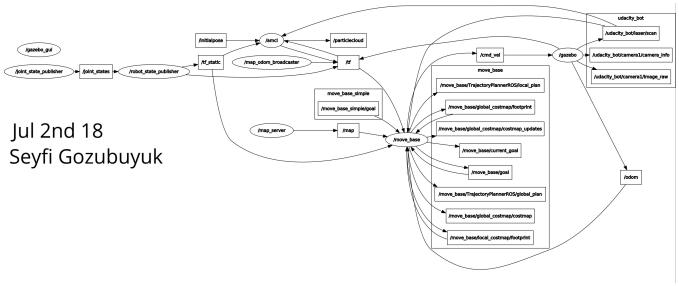

Fig. 5. Udacity Bot Rosgraph

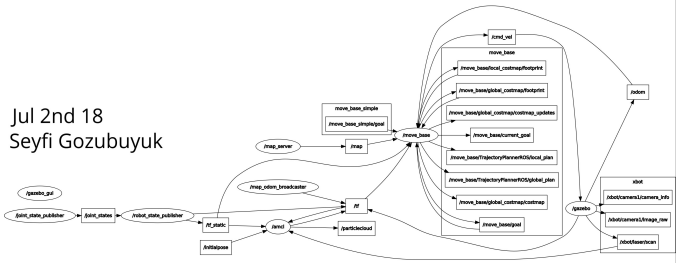

Fig. 6. xbot Rosgraph

3.4.3 Parameters

The AMCL node parameters for Udacity bot is given in Table 1 [3].

TABLE 1
AMCL Parameters for Udacity Bot

Parameter	Value	Default
odom_frame_id	odom	odom
odom_model_type	diff-corrected	likelihood_field
base_frame_id	robot_footprint	base_link
global_frame_id	map	map
transform_tolerance	0.2	0.1
min_particles	15	100
max_particles	250	5000
initial_pose_x	0.0	0.0
initial_pose_y	0.0	0.0
initial_pose_a	-0.7853981634	0.0 * pi
update_min_d	0.2	0.2
update_min_a	0.5235987756	pi/6
recovery_alpha_slow	0.001	0.0 (disabled)
recovery_alpha_fast	0.1	0.0 (disabled)
laser_model_type	lfp*	lfp*
laser_min_range	-1.0	-1.0
laser_max_range	-1.0	-1.0
laser_max_beams	30	30
laser_z_hit	0.95	0.95
laser_z_rand	0.05	0.05
odom_alpha1	0.005	0.2
odom_alpha2	0.005	0.2
odom_alpha3	0.010	0.2
odom_alpha4	0.005	0.2

lfp* = likelihood_field_prob

The parameters for TrajectoryPlannerROS in the base local planner is given in the Table 2

TABLE 2
Base Local Planner Parameters Udacity Bot

holonomic_robot	false	true
controller_frequency	7.0	20.0
meter_scoring	false	true
xy_goal_tolerance	0.05	0.1

TABLE 3
Costmap Common Parameters Udacity Bot

map_type	costmap
obstacle_range	5.0
raytrace_range	6.0
transform_tolerance	0.2
inflation_radius	0.5
observation_sources	laser_scan_sensor

The common parameters for costmap is given in the Table 3

The global costmap parameters are given in the Table 4

TABLE 4
Global Costmap Parameters Udacity Bot

global_frame	map
robot_base_frame	robot_footprint
update_frequency	5.0
publish_frequency	5.0
width	40.0
height	40.0
resolution	0.025
static_map	true
rolling_window	false

The global costmap parameters are given in the Table 5

TABLE 5
Local Costmap Parameters Udacity Bot

global_frame	odom
robot_base_frame	robot_footprint
update_frequency	5.0
publish_frequency	5.0
width	40.0
height	40.0
resolution	0.05
static_map	false
rolling_window	true

The differences between Udacity bot and xbot are given in the Table 6.

4 RESULTS

4.1 Localization Results

4.1.1 Udacity Bot

The duration to reach the target for the Udacity Bot is not always the same. In one run, this duration is around 120 seconds. The screenshot of this run is available at Figure 7.

However, on another run it took around 250 seconds, this can be seen from the Figure 10. The convergence time

TABLE 6
Parameter Differences Between Udacity Bot and xbot

File or Pkg	Parameter	Udacity Bot	xbot
AMCL	min_particles	15	100
AMCL	max_particles	250	1000
Common	obstacle_range	5.0	6.0
Common	raytrace_range	6.0	9.0
Common	inflation_radius	0.5	1.0
Common	robot_radius	N/A	0.5
Global	update_frequency	5.0	3.0
Global	publish_frequency	5.0	3.0
Local	update_frequency	5.0	1.0
Local	publish_frequency	5.0	1.0
Local	width	40.0	3.0
Local	height	40.0	3.0
Local	resolution	0.05	0.01

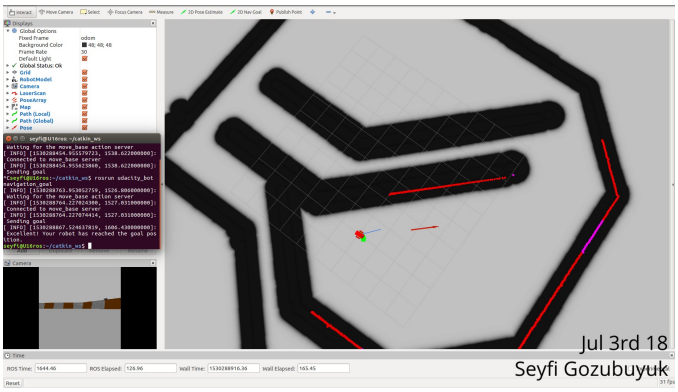

Fig. 7. Udacity Bot at Goal Run 1

for the particles is less then 10 seconds, the Figure 9 shows the converged particles. The robot traveled to the opposite direction for 95 seconds. The Figure 8 contains the turn back point.

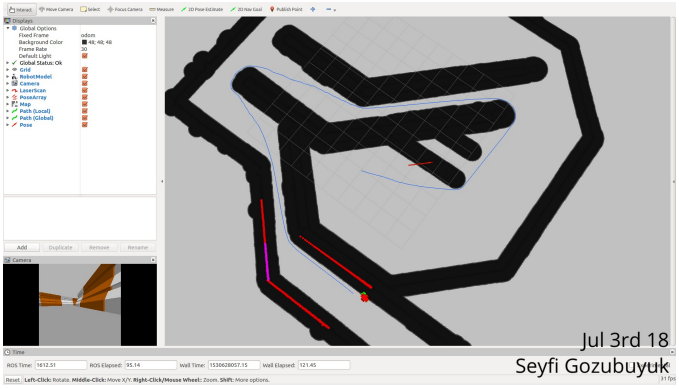

Fig. 8. Udacity Bot - Turn Back

4.1.2 xbot

The convergence time of the particles are between 5 seconds and 10 seconds for the xbot, and it is very similar to the one for the Udacity bot. Figure 11 shows the particle convergence time for the xbot. xbot reaches to the goal position around 50 to 60 seconds, this can be seen from the Figure 12.

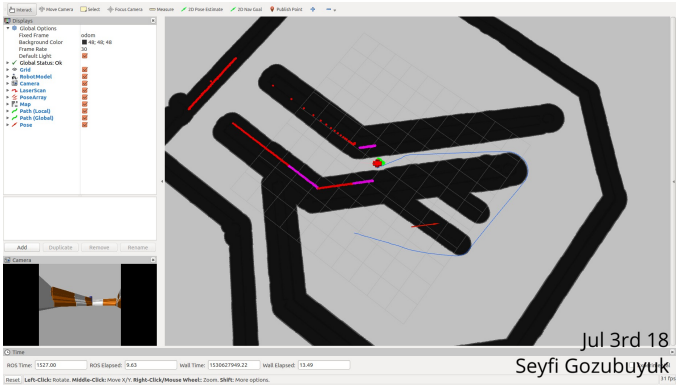

Fig. 9. Udacity Bot - Particle Converge

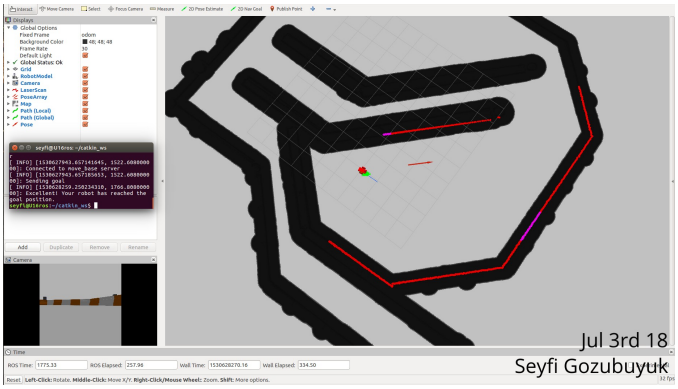

Fig. 10. Udacity Bot at Goal Run 2

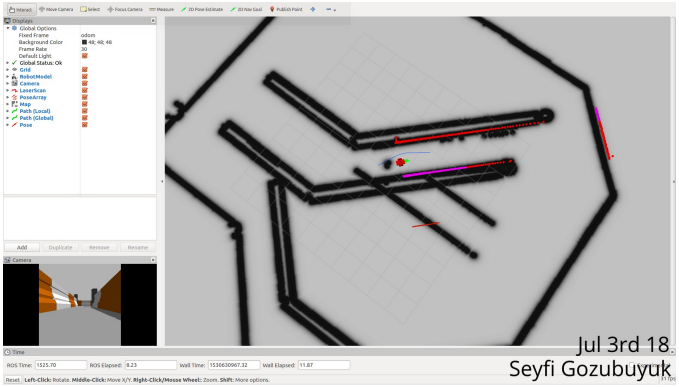

Fig. 11. xbot - Particle Converge

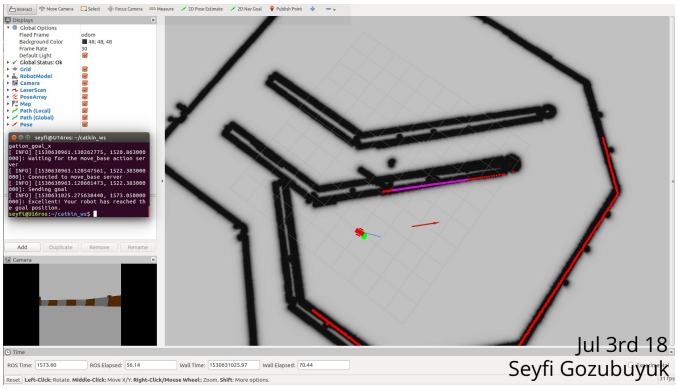

Fig. 12. xbot at Goal

4.2 Technical Comparison

The main difference between two robot models is the _link, and the sensors are connected to this link, instead of the chassis. There are differences on the parameters selected, which can be seen from the Table 6. The reason for the better performance of the xbot is the differences in the parameters.

5 Discussion

The two robot models were able to reach the target. The Udacity Bot first went to opposite direction, whereas the xbot directly moved to the goal direction. The reason for this difference was the differently set parameters. It is required to tune the parameters for the Udacity Bot, but due to time limitations, the tunning operation was marked as a future work.

The convergence time for both of the robots were quite similar. Which shows that the AMCL package parameters were almost the same. Actually, only the number of particles parameter was different. As a result, these parameters do not have much effect on localization performance.

5.1 Topics

- xbot performed better. It took less time to reach the goal position.
- The different costmap parameters let xbot to perform better. The xbot directly started following the global path.
- The 'Kidnapped Robot' problem is randomly moving the robot to another location. It is a challenge to recover from [4]. The current version of the xbot failed to recover from changing its location on Gazebo. The changed position of the xbot is in the Figure **??**. Figure 14 and Figure 15 show the situation nearly one minute later; and Figure 16 shows when the AMCL was aborted due to the failure of generating a plan.
- Localization can be performed in scenarios in which the map is known, and the location of the robot does not randomly change.
- MCL/AMCL algorithms are a proper choice in industry domains where the map is known and small. The larger the map, the higher the number of particles; which will result in increased computational costs. They work better in closed environments, such as warehouses and stores. The reason for that is to have obstacles, columns, and barriers. The laser sensor will get the distances from them providing more information to localization.

6 Conclusion / Future Work

It is required to tune the parameters for the Udacity Bot to make it reach the goal position without going the opposite direction first. Another task to work on is kidnapped robot problem. When a robot moved to a random location on Gazebo, it should still be able to reach the target.

Testing the algorithm on the Jetson TX2 and comparing the results with the current results is a waiting task for the future work. Upon receiving successful results, the algorithms in the project can be used on a real robot such a vacuum cleaner or an autonomous forklift.

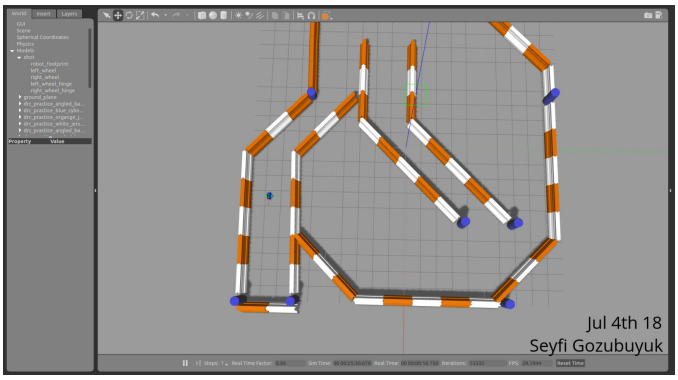

Fig. 13. xbot - Kidnapped Location

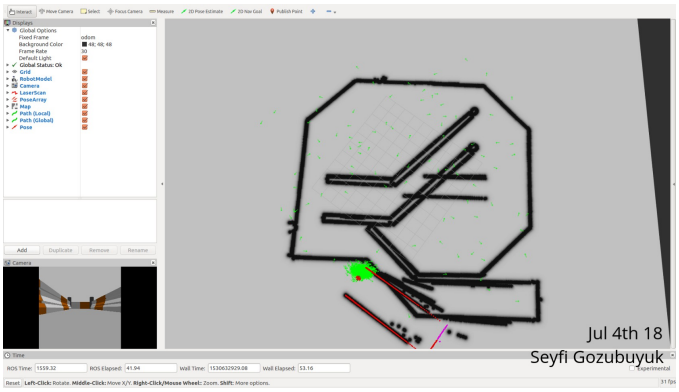

Fig. 14. xbot - Kidnapped 1

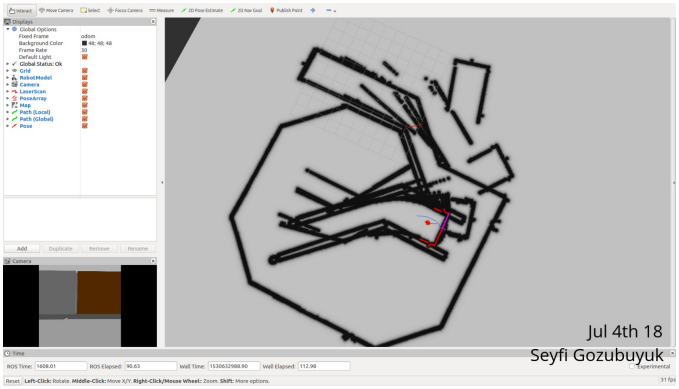

Fig. 15. xbot - Kidnapped 2

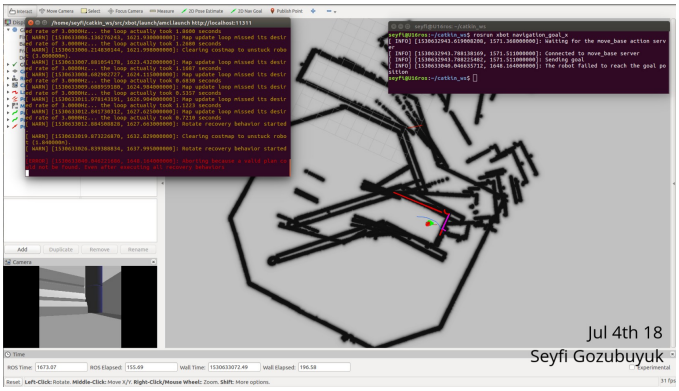

Fig. 16. xbot - Kidnapped Failure

6.1 Hardware Deployment

The project was deployed in the simulation environment. The machine has an I7 CPU, 8GB RAM, and 650M GPU. The configuration was able to run the project, but it needed to decrease the update_frequency and publish_frequency. In addition, there were limitations on the resolution of the costmaps. With more powerful CPU and GPU, it will be possible to increase the update and publish frequencies and the resolutions. Increasing the resolution helped xbot to perform better compared to the Udacity Bot.
More powerful hardware will enable to test on different and bigger maps. Using a Jetson TX2 would increase the performance of this project.

REFERENCES

[1] Kalman Filter, "Kalman filter — Wikipedia, the free encyclopedia," 2016. [Online; accessed 02-July-2018].
[2] Particle Filter, "Particle filter — Wikipedia, the free encyclopedia," 2016. [Online; accessed 02-July-2018].
[3] K. Zheng, "Ros navigation tuning guide.," 2016. [Online; accessed 02-July-2018].
[4] Kidnapped robot problem, "Kidnapped robot problem — Wikipedia, the free encyclopedia," 2016. [Online; accessed 03-July-2018].