CSCI-632 Mobile Robot Programming
Final Project Report
Vishal Vijay Kole (vvk3025), Mounika Alluri (ma8658), Shih-Ting Huang (sh3964)

1 Introduction

Change the file path for the map in getPath of safetogo.py file and init function of Mapper class in mapGUI.py. Run the following files simultaneously after launching the Gazebo file:

- Run the localization node
  rosrun project-name localization.py
- Run the safegoto file
  rosrun project-name safegoto.py input.txt

Figure 1 shows the flow of nodes for the project. After waking up, the robot safe wanders without running into obstacles until localization. Once localized, it publishes global pose to the robot. This global path is used by A* algorithm to find an optimal path with way-points to the destination. This path is then executed by the robot with obstacle avoidance for any obstacles that are not integrated into the map.

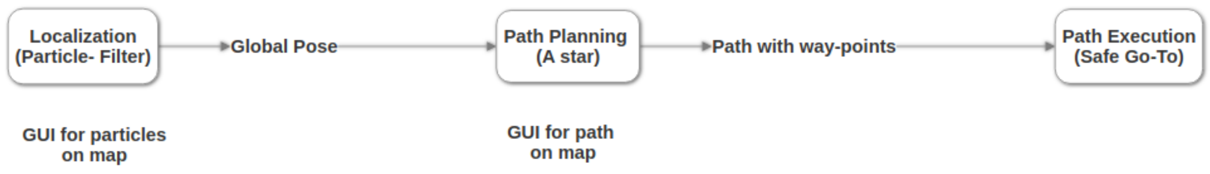


Figure 1: ROS Nodes for SLAM

2 Localization

(a) Algorithm

We implemented the particle filter algorithm to solve the localization problem with the given map, Figure 2. Every particle has attributes Pose() and weight. The Pose() contains global x and y coordinates and orientation value. First, we initialized 16 $\times$ 16 locations, Figure 3, with 5 different orientations, Figure 4, around each possible starting points. Hence, we have $16 \times 16 \times 5 \times 6 = 7680$ initial particles with weight = 0.5.
Second, we have a predict function to compute the difference between previous odometry data and current odometry data from the robot and apply the difference to all particles. Third, we modify the weight of each particle base on the corresponding map reading open distance values. We compute the distance to the wall on the map by using the Euclidean distance metric form the current pixel to the wall pixel, only for specified angles of the robot. The map reading function can return the left, right and front open distance values base on the map given particle global location and orientation. We compare the return

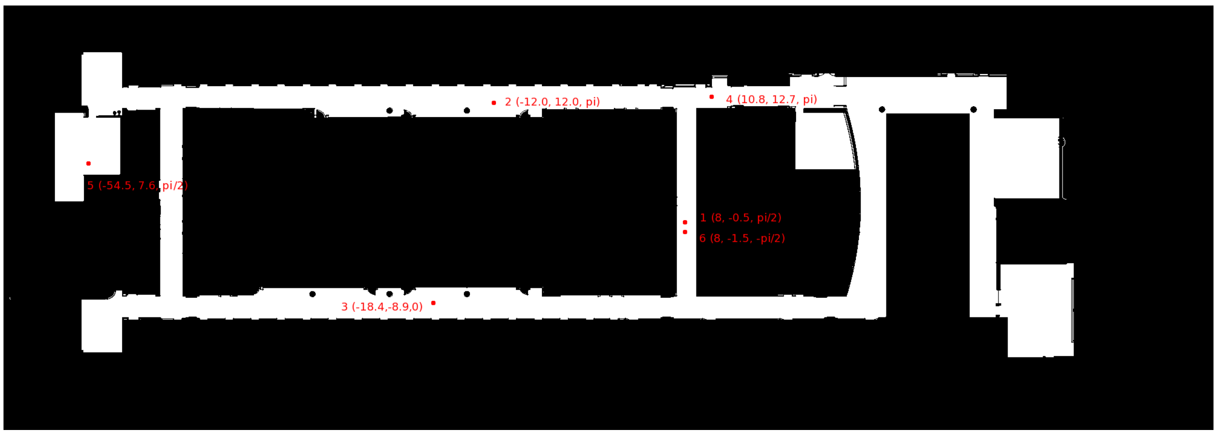


Figure 2: The given map with 6 possible initial locations

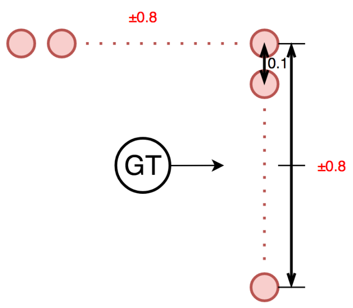


Figure 3: 16×16 initial particle locations

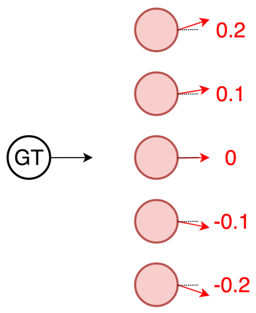


Figure 4: Five different orientations

value with sonar and laser sensor data. If the difference is under a threshold, we multiply the weight with percent-change, otherwise, we divide the weight with percent-change. Fourth, if the number of particles is lower than half of the original size, we perform the re-sampling function. The re-sampling function sorts the particles by weight and randomly generates particle from the heaviest weight with increasing/decreasing x or y coordinates value between 0.1 to 0.2, and randomly choose orientation difference among -0.1, 0, or 0.1 radians. The generating loop continues until the number of particles equals to the initial number of particles. Fifth, the robot will safely wander to explore the environment and provide more space information to modify weights of particles to help localization. Sixth, the converge function will check if all particles converge to a close area under a certain threshold. If particles have converged, the mean Pose() value of all particles' will be treated as a robot's location and orientation.

(b) Simulation

Starting location 1

Video: `https://youtu.be/eBJFBz7LyWc`
Particles from location 2, 3, 4, and 5 were eliminated because of low probabilities, only particles from location 1 and 6 remained at the beginning after few seconds. Both of them were in the same hallway where the side distance was the same. They were able to be distinguished until one cluster reaches the T-intersection. It converged at point (8.49,

11.54, 1.51) in 152 seconds.

Starting location 2

Video: `https://youtu.be/JuIq3j97OGI`
It is easy to get confused between starting location 2 and 3. They both have a similar distance of hallway and front open space. But, once the robot started passing over the pillar and made a turn, particles started converging. This simulation converged at point (-15.91, 12.4, 3.38) in 64 seconds.

Starting location 3

Video: `https://youtu.be/6AZ28dtRNxs`
The situation is similar to starting location 2. The simulation converged at point (-13.48, -8.34, 0.01) in 72 seconds. The starting location 2 and 3 are most likely to be converged at the wrong location due to the similarity of left, right, and front open space distances.

Starting location 4

Video: `https://youtu.be/R0SzC3PslVY`
Because of the side distances, this starting location can easily converge. It converged at point (10.03, 12.87, 3.1) in 13 seconds.

Starting location 5

Video: `https://youtu.be/GcLCWU6tNVg`
This starting location is also easy to converge successfully because of the side distances. It converged at point (-53.9, 8.09, 1.6) in 11 seconds.

Starting location 6

Video: `https://youtu.be/q2hm5eZtfEY`
This is a similar situation as starting location 1. Particles from location 2, 3, 4, and 5 were eliminated, and only particles from location 1 and 6 remained at the beginning after few seconds. They were able to be distinguished until one cluster reaches the T-intersection. It converged at point (12.08, -8.34, 0.04) in 177 seconds.

3 Path planning

(a) Algorithm

Once the localization is done, it publishes the pose of the converging point to '/localization' topic. The path planning program, safegoto.py, subscribes to the '/localization' topic. Once a location is published, it will start to get the shortest path to destinations. The main idea of our path planning is based on the A* search algorithm. The valid steps are based on the given map's pixels (open cells) with a safe distance (14 pixels) away from occupied cells. We check this for all the valid pixels in the map and it gives us a safe path(not too close to the wall) to the destination. We have used the Euclidean distance as the heuristic distance and it works well for our application. The program will then generate a valid path(these would be the pixels in the map). It will then convert the pixels into global coordinates and deletes 19 consecutive pixels and keeps the 20th one to create way-points. The next step is to transform way-points from global coordinate to local coordinate(the local coordinates are the robot coordinates which change for each starting point). Then, it performs safe-goto toward the destination based on the way-points.

(b) Simulation

Video: `https://drive.google.com/open?id=1KGq6OPEN0yz9VdlACjPwNIcJUa18Iesx`
Video: `https://drive.google.com/open?id=1W6vOTzQzdhDQE_wtl0Q1CXhJWC81bFwr`
Video: `https://drive.google.com/open?id=14PPVv0Z8l9ejWOXFiN1eNIJBR9maS8So`
The above videos are the A-star path planning algorithm results for different points in the map. We are showing all the possible points it consider but it selects only the shortest and valid points for final way-point generation.

Once localization converges, the GUI is closed and localization node is shut down. The A* star node uses this GUI to print path on the map. Figure 5 shows the path between two points in the map.

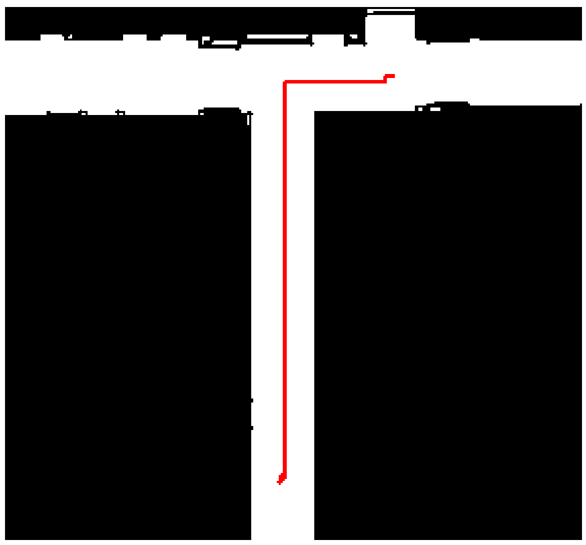


Figure 5: The path A-star algorithm computed

Initial implementation of GUI takes the complete path from A* star node and plots it on the GUI. This did not help when there was no path with the specified conditions. Hence, this was modified so that the path is represented on GUI as nodes are being added to heap of A* implementation. If the distance of destination from walls is less than safe distance or the destination is too close to the walls, there is no optimal path. In these cases, the path planner tries to go forward and backward along the initial path.

4 Software testing and observations

We tested the project on Gazebo before testing it on hardware and below are the links to the video of end to end execution of the project.

Video: `https://drive.google.com/open?id=1Cqg76KO6twYS3xS2BhMwG9scApS2M3H5`
Video: `https://drive.google.com/open?id=1JQ8aDrVjVeq5z5lFrxvfguLNQk64CouL`

(a) Things that worked

- The localization algorithm was able to converge at all the 6 locations successfully.
- A-star algorithm is implemented with the modified heuristic function to avoid getting close to the walls also works as expected
- Path execution is perfect if the initial location of the robot is accurate. It is also able to avoid obstacles like the wall and pillar.

(b) Things that did not work

- The localizations, although able to detect between the locations correctly, sometimes fails to localize for the symmetric starting points correctly.
- Convergence for the particles is also not perfect and computes a point which is close to the robot, but not the actual position of the robot. This adds to the error and affects the further process.
- The continuous world and the discrete map interaction adds some error. For example, The distance to the wall in the map and the actual distance is not perfect for the exact locations. This we think is because of the resolution of the map and the discrete representation , it skews the readings.
- Angle error and computation errors accumulate and it slowly takes the robot off path.
- Long and complicated paths are not possible without any further modifications to the program to avoid or minimize the errors.

5 Hardware testing results

We tested localization with starting location 1, 2, and 4 using Clark, Table 1. Since the laser sensor did not function correctly, we modified our program to use sonar data. Left, right and two sonar sensors at the center are used for the safe wander without running into obstacles for localization.

Table 1: Convergence results on hardware

Starting location	Converged point (x, y, θ)	Time (second)	Result
1	$\times$	$\times$	Wrong
2	(-18.2, 12.08, 3.24)	84	Correct
4	(-0.03, 12.38, 3.21)	131	Correct

Starting location 2

The localization also successfully converged to Clark's global location (-18.2, 12.08, 3.24) in 84 seconds, Figure 6 and Figure 7.

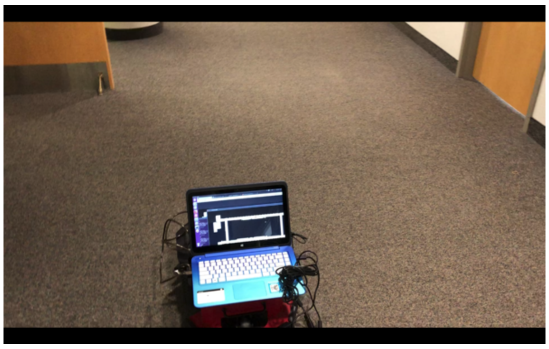


Figure 6: Clark's location

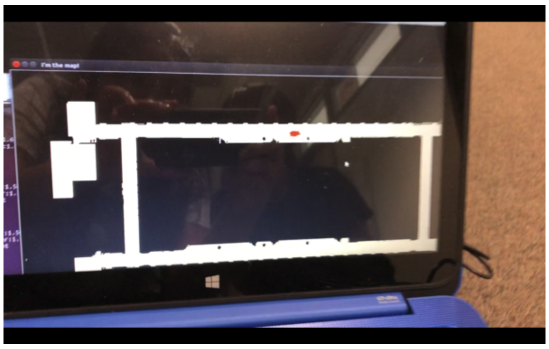


Figure 7: Localization result

Starting location 4

The localization successfully converged to robot's global location (-0.03, 12.38, 3.21) in 131 seconds, Figure 8 and Figure 9. In the simulation, robot always turns left at the T-intersection. Instead of turning left, Clark went straight every time we tested it. It is because the detected sonar data was larger than the safe distance. Hence, the robot didn't make a turn.

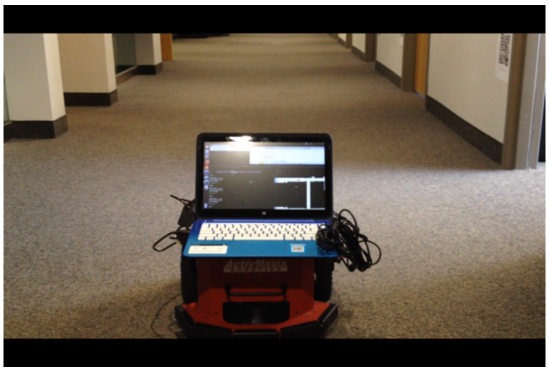
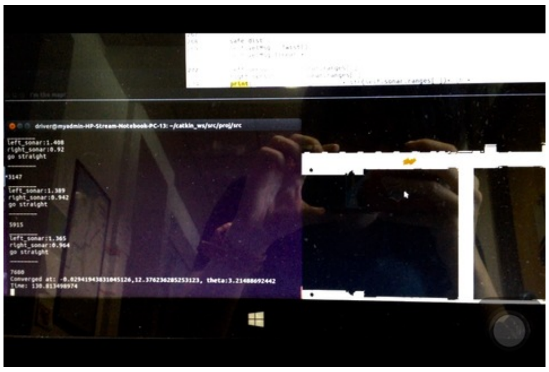


Figure 8: Clark's location



```
driver@myadmin-HP-Stream-Notebook-PC-13: ~/catkin_ws/src/proj/src
________
left_sonar:1.408
right_sonar:0.92
go straight
__________
3147
________
left_sonar:1.389
right_sonar:0.942
go straight
__________
5915
________
left_sonar:1.365
right_sonar:0.964
go straight
__________
7600
Converged at: -0.029419438310045126,12.3762362085253123, theta:3.21488692442
Time: 130.813498974
```

Figure 9: Localization result

Starting location 1

This did not converge correctly. Clark started from the middle of the narrow hallway, Figure 10. In the localizing process, it narrows down to two clusters, Figure 11. However, it converged to the wrong cluster, in the end, Figure 12.

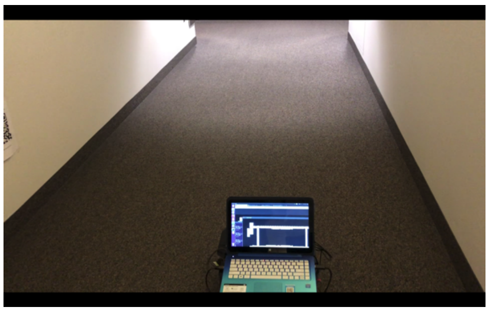


Figure 10: Clark's location

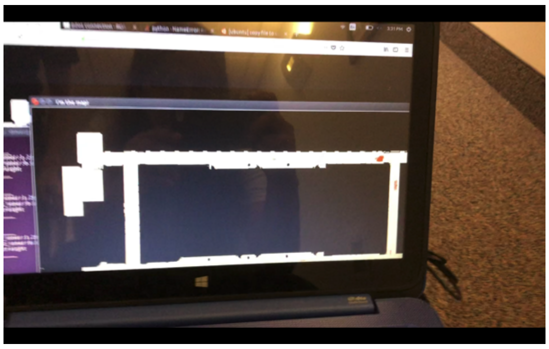


Figure 11: Localization processing

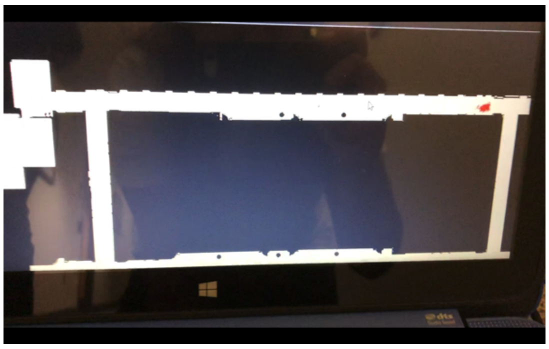


Figure 12: Localization result