Intelligent Mobile Robots Assignment 1 Creating a 2D Map of an Unknown Environment

Arpit Sharma
P Number: P2613237,
Intelligent Systems, and Robotics MSc,
De Montfort University, Leicester, England.

Contents

1 Introduction

For the assignment, two-dimensional maps are constructed according to the binary mapping technique(without correction) and Bayes theorem(with correction) using the occupancy grid method, and the results are compared visually according to the accuracy, reliability, computational and time efficiency. Both the techniques are discussed in detail. The coordinates of the occupied points are dynamically printed in an excel file, coordinates.xlsx. The robot is controlled by a keyboard using the "turtlebot teleop" package and made to wander randomly while mapping the environment. "Vigorous movements"[Robc] are avoided as they can diminish the quality of the map.

Ultimately, a good approximation of the environment is achieved by both the techniques in the experiment, but the one using Bayes Theorem has far better quality.

1.1 Turtlebot3 Burger

Figure 1: Real Turtlebot3 Burger

Turtlebot3 Burger is very "small"[Ltd] in size and cheap but still doesn't lack any feature when compared to other Turtlebot3 models. The model is open for expansion and customizable, which means more external sensors and "mechanical parts"[Robb] can be added to it. It has an inbuilt laser sensor that covers 360 degrees to measure the distance from the obstacles and has been used in the experiment. This makes it very suitable for making prototypes and research.

Figure 2: LDS-01 sensor

An inbuilt 2D laser scanner, LDS-01 where LDS stands for 'Laser Distance Sensor', has been used. It detects the distance from the obstacles in all angles in the plane(360 degrees), which makes the mapping very fast. It supports both, "USB and UART interfaces"[Roba].

2 Map Construction Technique

2.0.1 Using Binary mapping

For map construction, odometry, laser scan data are used. There are two coordinate systems used: a static global coordinate system and a mobile one with respect to the robot's position. The current position (x_r, y_r) of the robot is got from the optometry system after every time frame. We just need the x and y coordinates, and the heading (yaw or θ_r in the below diagram) of the robot with respect to a stationary global frame for mapping purposes. The laser sensor of the robot gets the distance detected for all the angles in its range.

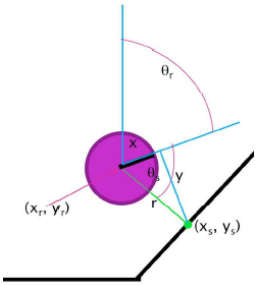

Figure 3: Diagram for understanding the mathematics behind the Technique

In the figure above, we get the object distance, r from the laser scanner, and the angle of the sensor offset, θ_s. From just these two values, we find the local co-ordinates of the obstacle with respect to robot, (x_s, y_s) by the following equations. The robot radius didn't have much impact as it was very small as compared to the distances being detected.

$$x_s = \cos\theta_s * (r + robot\ radius)$$
$$y_s = \sin\theta_s * (r + robot\ radius)$$

Now, to get the global coordinates of the obstacle from the local coordinate system, a linear and angular transformation has to be applied. To perform both of them at the same time, a rotation matrix is used as shown below. the

$$\begin{bmatrix} x_g \\ y_g \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_r & -\sin\theta_r & x_r \\ \sin\theta_r & \cos\theta_r & y_r \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

After getting the global co-ordinates (x_g, y_g) of the obstacle, the coordinates must be converted into the grid cells (x_c, y_c) for mapping, according to the resolution and the origin of the grid map, and then, that cell must be made black by making the probability of that cell being occupied to 1.0 as shown below.

```
 # co-ordinates of the origin of the map (-3.5,-3.5)
 x_o = map.origin_x
 y_o = map.origin_y

# positioning the origin in the map to the center of the grid
# and mapping points to cells in the grid
x_c = int((x_p + abs(x_o)) / map.resolution)
y_c = int((y_p + abs(y_o)) / map.resolution)
```

The process is repeated every time an obstacle is detected, and, eventually, a 2D map gets constructed in the following stages as shown in the figure below.

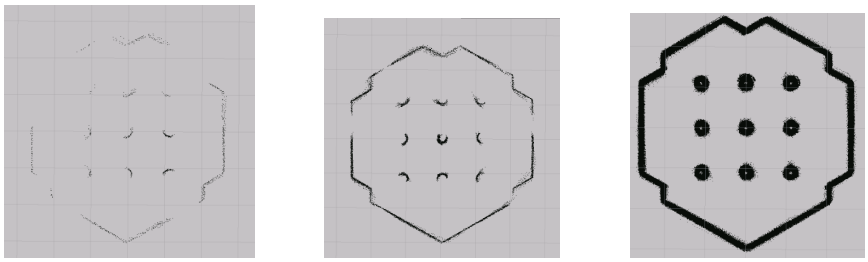

Figure 4: Stages in the Binary mapping (without correction)

2.0.2 Using Recursive Bayes theorem

In this technique, most part of the above technique is used, but instead of straight away assigning a probability of 1.0 to the occupied cells, a probability little less than 1.0 is assigned to it and the points in its vicinity as there is always a margin for error in the sensor reading in the real world. It is much more computationally expensive and time-consuming than the previous technique, but it is much more reliable as it upgrades the probability of a cell being occupied 'recursively' using the Bayes theorem making it much more promising and robust. That is why it is more desirable than the previous technique.

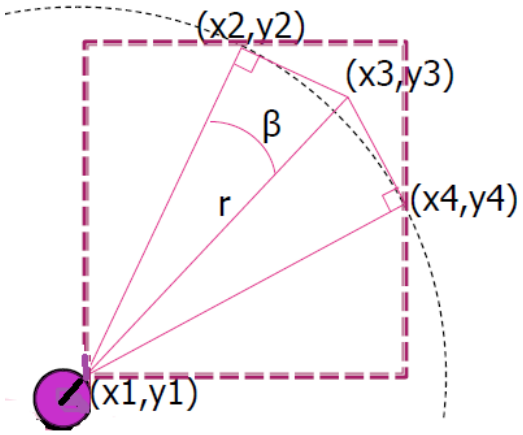

Figure 5: Bayes Theorem Diagram

From the above diagram, it can be seen, that we form a cone(a 60^0 cone was used in the experiment, meaning β was kept 30^0). The more certain, we are if the obstacle lies in the line of sight of the heading of the robot ($\theta_s = 0$). As the sensor offset increases, (i.e. the direction error increases), the probability of it being occupied decreases. Therefore, (x_3, y_3) has nearly 1.0 probability of being occupied and is higher than (x_2, y_2) and (x_4, y_4). Also, the probability for the point (x_2, y_2) is equal to that for (x_4, y_4) as their direction errors are equal. The recursive Bayes theorem has been used for calculating the probabilities as follows:

$$\frac{(E|H_t) * P(H|E_{t-1})}{(1 - E|H_t)) * (1 - P(H|E_{t-1}))}$$

where P(H) is the prior probability of the hypothesis, P(E) is the prior probability of the evidence, $P(E|H_t)$ is the probability returned by the model, $P(H|E_{t-1})$ is the posterior probability of the hypothesis.

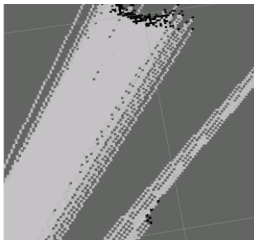

Figure 6: Gradient in the mapping using Bayes Theorem

It can be seen from the figure above that some region has shades of grey because those points were away from the obstacle and hence, assigned a probability of being occupied between 0.0 and 1.0.

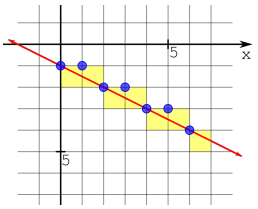

Figure 7: Bresenham's Line Algorithm[con]

As we need the cells between the robot and the obstacle to draw the line between them(as shown in the figure above) and to assign a probability to each of them, Bresenham's Line Algorithm[con] was used for the purpose using the bresenham python library which was installed simply using pip install, and then imported in the mapper.py class. It returns an array of top left vertices of the cells. The appropriate probabilities calculated by the Bayes theorem were assigned to each cell in the array returned by it.

The probability of any cell should never be absolutely 0.0 or 1.0 because the algorithm doesn't improve as the Bayes theorem returns the same values of 0.0 or 1.0, over and over again.

The probabilities are constantly updated as long as the robot continues to move in the environment, making the probabilities of occupied cells closer to 1.0 and of empty cells, closer to 0.0.

The process is repeated every time an obstacle is detected, and, eventually, a 2D map gets constructed in the following stages as shown in the figure below.

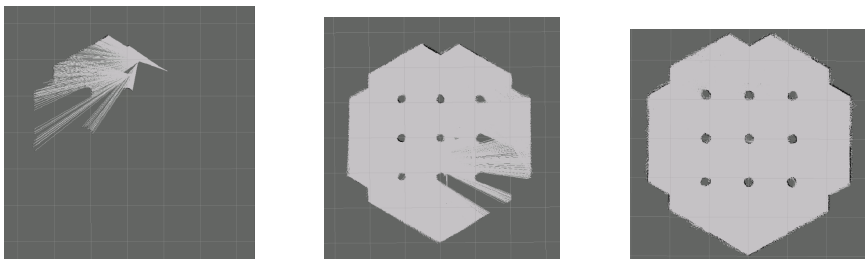

Figure 8: Stages in the mapping using Bayes Theorem (with correction)

3 Software Implementation

3.1 ROS development studio

"A Platform to Learn ROS-based Advanced Robotics Online" has been used in the experiment using the "free starters pack". The entire experiment has been performed on the website in the ROS project which the website calls 'ROSjects'. The following are the components of the studio.

3.2 ROS Terminal

It is the backbone of the ROS development environment. All the commands have been types in the command in four different terminals in order as follows:

3.2.1 Terminal 1

```
cd ~
cd catkin_ws/src
cd move_bot/src
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

3.2.2 Terminal 2

```
cd ~
cd catkin_ws/src
cd mapping/src
chmod +x avoid_obstacle.py
python avoid_obstacle.py
```

3.2.3 Terminal 3

```
rosrun rviz rviz
```

3.2.4 Terminal 4

```
python -m pip install bresenham

python -m pip install XlsxWriter

cd ~
cd catkin_ws/src
cd mapping/src
```

```
chmod +x mapper.py
chmod +x map.py
chmod +x helper_functions.py
python mapper.py
```

3.3 Gazebo

It provided the simulated environment in which the robot can be seen to wander randomly in the environment because of the code in the avoid$_obstacles.pyfile$.

3.4 RViz

The RViz tool is an official 2D and 3D "visualization tool of ROS". Almost all kinds of data from sensors can be viewed through this tool. It has been used for the visualization of the generated map. The points in the environment are represented by a grid in the map to have an approximation of the environment being mapped. A suitable resolution must be selected.

It is opened by launching the following command in the ROS terminal,

```
rosrun rviz rviz
```

The map class located in the map.py file in the mapper package draws the map, which is visualized on the screen of the tool by clicking the 'Add' button at the bottom, selecting the 'By topic' tab, and then selecting 'Map' under the '/map' section. Hence, the map gets displayed. The map dynamically updates as the robot moves according to its laser scan data and odometer readings.

For the experiment, different resolutions were tried for mapping the 5 meters x 5 meters environment. To fully cover the map with some margin, the dimensions of the map were kept 7 meters x 7 meters to fully cover the environment. The default resolution was 0.1 that would constitute a 70x70 cell grid, hence containing 4900 grid cells as each cell in the grid would represent 0.1 meters x 0.1 meters or 10 centimeters x10 centimeters. The resolution was changed to 0.01 with the width and height of 700 grid cells each, and a total of 490,000 grid cells of 0.01 meters x 0.01 meters. This increase in resolution really improved the quality of the map as there were more cells to map the points.

The origin was kept in the center of the grid for both the resolutions with the coordinates of (-3.5, -3.5).

3.5 Code Editor

the entire coding has been done in the default editor. The code was divided into 3 files each containing three different classes, HelperFunctions.py, Map.py,

and Mapper.py, to improve the readability of the code.

The mapper.py class subscribes to two topics namely, "odom" and "scan" for retrieving the "Odometry" and "LaserScan" data respectively. Their respective callback methods get called when the odometer and the laser scanner publish data to their topics.

4 Testing and Results

The coordinates of the occupied cells are stored in an excel file, which gets edited dynamically as the robot maps the environment.

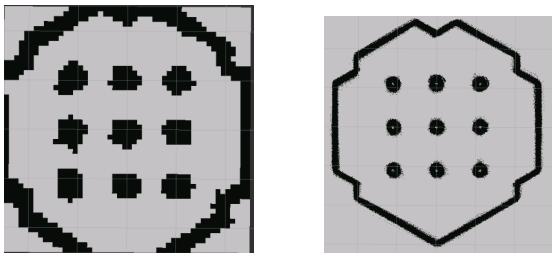

Figure 9: Effect of resolution on the effect of mapping. 0.1 and 0.01 in order(left to right)

On changing the resolution from 0.1 to 0.01, the quality of the map incredibly increased, as shown in the figure above, because of the reasons discussed in section 3.4..

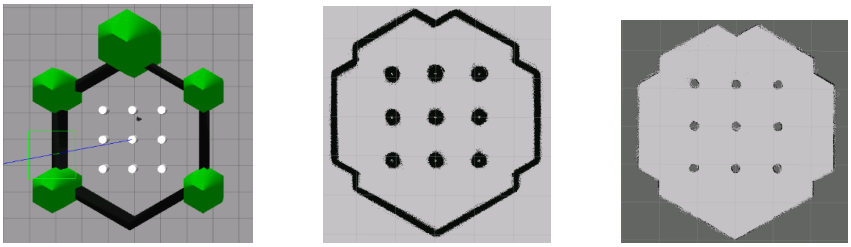

Figure 10: Simulated environment, Binary map and Bayes Theorem map in order(left to right)

From the above figure, it can be seen that the lines drawn with Bayes theorem are much thinner as compared to those drawn using just binary mapping proving that with Bayes theorem, much higher accuracy and precision are achieved.

The Binary mapping takes much less time and computational power but doesn't upgrade, meaning that if the sensor wrongly detected an obstacle, which is very much possible in the real world because of noises and the error in the readings of the odometer, and the "uncertainty in sensory information"[NUS12], those errors cant be corrected and hence, it will show much more occupied cells than there actually are. This is very problematic if very high accuracy is needed.

While using Bayes Theorem, different cone-angles(β) were tried. It was realised that on increasing it, the computational power consumption increased as indicated by a higher CPU usage, but the mapping became faster. To balance these two constraints, a medium value of cone-angle of 60^o ($\beta = 30^o$) was selected.

5 Conclusion

Much higher accuracy and precision are achieved with the Bayes theorem as compared to binary mapping but, at the cost of higher computational power and time.

In case of the Bayes theorem, the probabilities of interest don't change immediately as the new values are based on old values. Therefore, outliers in the sensors readings don't have any drastic effect, over time.

Mapping can further be improved if a more sophisticated, robot localization technique is used as the mapping highly depends on the accuracy of the robot's position in the environment. In Bayes theorem, there is a trade-off between time and computational power. On decreasing the cone angle, β , as shown in figure 5, the computational power decreases but the time taken to map the environment increases, and vice-versa.

References

[NUS12] KS Nagla, Moin Uddin, and Dilbag Singh. "Improved occupancy grid mapping in specular environment". In: *Robotics and Autonomous Systems* 60.10 (2012), pp. 1245–1252.

[con] Wikipedia contributors. *Bresenham's line algorithm*. URL: `https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm`. (accessed: 05.04.2021).

[Ltd] Cyberbotics Ltd. *Robotis' TurtleBot3 Burger*. URL: `https://www.cyberbotics.com/doc/guide/turtlebot3-burger?version=develop`. (accessed: 04.04.2021).

[Roba] Open Robotics. *LDS-01*. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/. (accessed: 04.04.2021).

[Robb] Open Robotics. *Overview*. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#overview. (accessed: 04.04.2021).

[Robc] Open Robotics. *SLAM*. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/#map. (accessed: 04.04.2021).