Task4 ELA408 Occupancy grid algorithm

Jacob Johansson*, Mudar Ibrahim†
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden
Email: *jjn20030@student.mdu.se, †mim20004@student.mdu.se

***Abstract*—The occupancy grid mapping algorithm is one of the most important techniques in mobile robotics. As this mapping algorithm is beneficial for creating maps of an environment based on sensor readings. This paper explored the implementation of The occupancy grid mapping algorithm using MATLAB and tested it on the provided Freiburg Campus and Orebro dataset. The chosen method utilizes LIDAR sensor data and information about the robot's position to update an occupancy grid, representing the algorithm's probability in a grid format. The result demonstrates the efficiency of the algorithm in generating accurate maps, which help robot navigation and obtain a better understanding that clarifies the surrounding area.**

I. Introduction

Autonomous navigation systems are a critical aspect of mobile robotics, as they provide the robot with the ability to understand, interact, and interpret with the surrounding environment accurately. To obtain this type of environmental understanding, the robot must be able to use the input data from the sensors to create and update specific and detailed maps of the surroundings. One of the most effective and common methods of doing this is occupancy grid mapping, this method takes the environment and divides it into a grid of cells, where each cell has a probability of being free or occupied. This approach is particularly effective as it allows the robot to make decisions based on the input sensor data on a probabilistic map. The occupancy grid mapping algorithm is a very useful technique of mobile robotics due to it is characteristics that provide the robot with the ability to understand uncertainty. Using this method, robots can build detailed maps of their surroundings, which is critical for tasks such as autonomous navigation, obstacle avoidance, and decision-making[1] [2].

This paper explored the implementation of the occupancy grid mapping algorithm using MATLAB, where data from a LIDAR sensor from the Freiburg Campus and Orebro. The two datasets were chosen from the collection provided online [0]. By using this data, we can develop and test the algorithm in a realistic environment. The goal of this study is to create a reliable and accurate occupancy grid map, which can help robots navigate autonomously and avoid obstacles effectively[0]. The project also includes path planning on the created grid map using two known algorithms, A* and Dijkstra's algorithm. These algorithms will be used for path planning from one end of the map to the other. As well as investigate the algorithm's performance on the end of the maps to gain knowledge of how effective and useful they are in different situations. Path planning is usually used for the navigation of a robot to find an efficient path to reach the desired destination. Path planning ensures that the robot navigates through complex environments safely, as well as avoiding all possible obstacles that can appear, making real-time decisions based on sensor data and pre-constructed maps.

This study focuses on several important aspects of the algorithm, including how sensor data is transformed into a grid and how probabilities of occupied and free cells are updated over time. Also investigate how different parameters, such as the grid resolution and the maximum range of the sensor, affect the accuracy and performance of the map. Finally, Evaluate the effectiveness of the algorithm by comparing the generated map with the real environment and analyzing how well the robot can navigate based on this map[2]

Figure 1 demonstrates how the robot understands and analyzes the surrounding environment [0].

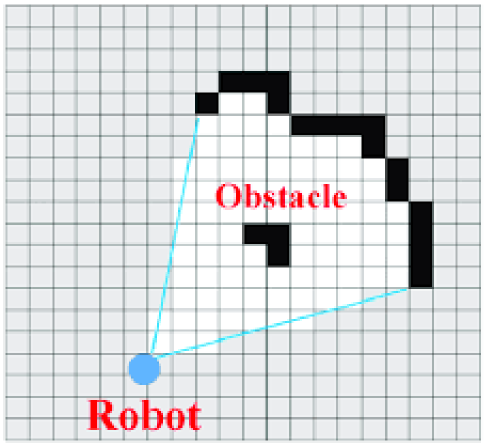

Figure 1. Figure of the Obstacle detection from the robot's perspective.

II. Method

Occupancy mapping

The environment is represented as a discrete grid where each cell in the grid represents a small area of the environment, containing the probability that the area is occupied by an obstacle. Each cell can be in three states, shown in 1.

$$\text{state}(cell_i) = \begin{cases} \text{Free} & \text{if } 0 \leq P_{cell_i} \leq P_{\text{max free}} \\ \text{Occupied} & \text{if } P_{\text{min occupied}} \leq P_{cell_i} \leq 1 \\ \text{Unknown} & \text{otherwise} \end{cases} \quad (1)$$

At each time step *t* the occupancy grid map takes a position and LIDAR measurements and calculates which cells the

LIDAR points fall in the grid. Assuming a Markovian update process, the estimation of the occupancy probability of cell i given the history of measurements up to time *t* is given by the Bayesian probability function [2]. This is shown in 2.

$$P(\text{cell}_i|z_{1:t}, x_{1:t}) = \left[1 + \frac{1-P(\text{cell}_i|z_t, x_t)}{P(\text{cell}_i|z_t, x_t)} \cdot \frac{P(\text{cell}_i|z_{1:t-1}, x_{1:t-1})}{1-P(\text{cell}_i|z_{1:t-1}, x_{1:t-1})} \cdot \frac{P(\text{cell}_i)}{1-P(\text{cell}_i)}\right]^{-1} \tag{2}$$

For computational efficiency, the log-odds notation is used during calculations, presented in equation 3.

$$l(cell_i|z_{1:t}, x_{1:t} = log(\frac{P(cell_i|z_{1:t}, x_{1:t})}{1-P(cell_i|z_{1:t}, x1:t)}) \tag{3}$$

Which can be simplified using equation 4.

$$l(cell_i|z_{1:t}, x_{1:t} = l(cell_i|z_t, x_t) + l(cell_i|z_{1:t-1}, x_{1:t-1}) - l(cell_i) \tag{4}$$

Where $l(cell_i|z_t, x_t)$ is the inverse sensor model, $l(cell_i|z_{1:t-1}, x_{1:t-1})$ is the previous log-odds value of $cell_i$, and $l(cell_i)$ is the prior log-odds value of $cell_i$.

When building the occupancy map for each data set (Orebra and fr campus), a grid resolution of 10 cm was chosen to get a fine-grained map of the Orebro map, and 1 meter for the Fr campus map. This resulted in a grid size of 750x750 for the Orebro map and 300x300 for the fr campus map. The probability value of a cell being occupied Poccupied was set to 0.75 while the probability value of a cell being free Pfree was set to 0.25. The prior probability Pprior was set to 0.5. This was found to be sufficient enough for the application. A probability map was constructed with all cells being set to a probability value of 0.5 to represent the unknown whether each cell was occupied. In order to optimize the algorithm, the probability map was then converted to the log-odds notation with equation 5.

$$\text{logOddsMap} = log(\frac{\text{probabilityMap}}{(1-\text{probabilityMap})}) \tag{5}$$

Due to each robot pose being local with reference coordinate (0, 0) and each sensor reading being relative to the robot pose at time t, the robot position (x, y) was transformed to the coordinate system of the built map by equation 6

$$\text{gridPosition} = round(\frac{\text{localPosition}}{\text{gridResolution}}) + round(\text{probabilityMap} \cdot 0.5) - round(\frac{\text{averageLocalPosition}}{\text{gridResolution}}) \tag{6}$$

, where *averageLocalPosition* represents the average position of the robot in the data set to center the mappings around the center of the probability map. This made it possible to reduce the total size of the probability map. With each given sensor reading, representing the distance from the robot to the hit object), the end position of the respective sensor reading was calculated using equation 7.

$$\begin{bmatrix}\text{localSensor}_x \\ \text{localSensor}_y\end{bmatrix} = \begin{bmatrix}\text{localPosition}_x \\ \text{localPosition}_y\end{bmatrix} + \text{sensorValue}_i \cdot \begin{bmatrix}\cos(i-90) \\ \sin(i-90)\end{bmatrix} \tag{7}$$

Where $sensorValue_i$ is the *ith* value of the sensor reading at time t, representing the angle i from the data sets respectively. 90 degrees was subtracted to keep the values relative to the heading of the robot. After calculating the robot position and the end position of each sensor reading, the Bresenham line algorithm was used to get all cells relative to the line between the robot position and the end position [0]. A parameter, OccupancyCellSize, was used to tune how many cells from the end cell that should be considered occupied. A value of 1 or 2 was found sufficient enough. Due to an error in the LIDAR sensor, the max range of sensor readings to trust was adjusted specifically for each data set. 20% and 25% respectively for the Orebro and Fr campus set resulted in a sufficient mapping.

Optimization

With a grid size of 750x750 and 300x300, the worst-case scenario of visiting each cell during path planning would result in 562.500 and 300.000 cells visited respectively. Given the tunable gridResolution parameter, uncertainties in the sensor readings, and reduced number of cells to visit during path planning, the Occupancy Inflation strategy was used to enhance the robustness and avoid cells near obstacles [0]: Due to the path planning being dependent on whether a cell is occupied in order to visit it or not, and the occupancy map being fine-grained by the *gridResolution* parameter, an inflation algorithm was used to propagate the occupancy value of occupied cells to neighboring cells within a radius parameter. The algorithm checks if a cell has a high enough probability, which was set to 0.5, to be occupied. If so, then all neighboring cells within the radius are also set to have the same probability of being occupied as the checked cell. A radii of 1 for Orebro and 0 for fr campus was found to be sufficient.

Path planning

Both A* and Dijkstra's algorithms in a static environment were used during path planning due to the environments already being mapped. The start and goal cells were chosen accordingly after occupancy grid mapping in order to experiment and test the robustness of the system in different scenarios.

Dijkstra's Algorithm: Dijkstra's algorithm methodically searches for the shortest paths from a specified starting cell to every other cells on a map. It does this by keeping a priority queue of cells and consistently choosing the cell that has the smallest total cost from the starting cell during each time step *t*.

A Algorithm:* A* is an improved technique of Dijkstra's algorithm by integrating heuristic data to converge the search towards the goal. It evaluates, not just the confirmed cost from the staring cell, but also an anticipated cost to the destination cell.

III. Result

The implementation was tested using the Freiburg Campus and Orebro datasets. The occupancy grid generated effectively represents the environment, accurately marking free and occupied cells. The robot's path is plotted on the grid, providing a visual representation of the robot's trajectory and the corresponding sensor readings. The algorithm efficiently updates the grid with each sensor reading, demonstrated by the computational time recorded. The final occupancy grid map highlights the areas of occupancy and free space, enabling robust navigation for autonomous robots.

The algorithm plots the robot's path on the generated grid. It visualizes the robot's trajectory and the data that the sensor captured during the movement of the robot. The result shows the navigation of the robot through the surrounding environment and how the robot continuously updates the grid and makes decisions based on these updates.

The comparison between A* and Dijkstra's algorithm is illustrated in figure III, which demonstrates the efficiency of A* over Djikstra's algorithm in path planning on the Orebro map. The result indicate that A* is generally more efficient in finding optimal paths compared to Dijkstra's algorithm due to the heuristic information. Furthermore, the outcome of path planning for the fr campus scenario is elaborated on in IV. Figure 4 and 6 shows the path planned for the Orebro map. 4 visualizes how the path always tends towards the goal as the shortest path.

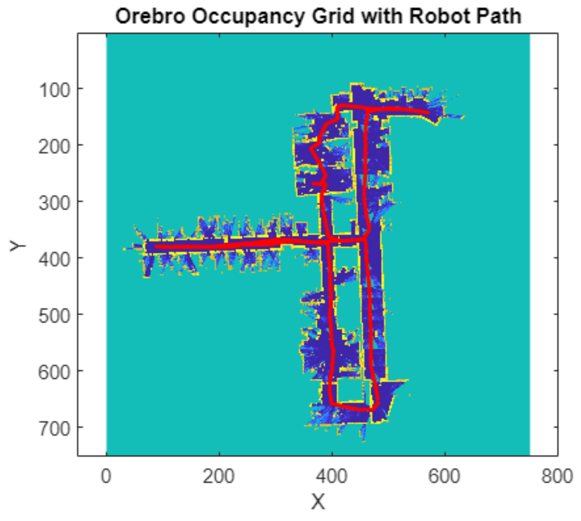

Figure 2. Occupancy mapping Orebro.

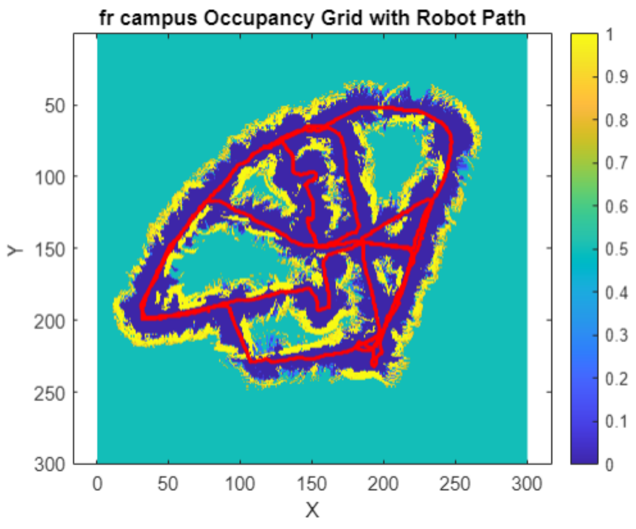

Figure 3. Occupancy mapping fr campus.

Map	A*	Dijkstra's
Orebro	17.5 s	66.2 s
fr campus	4.2 s	3.6 s

IV. Discussion

The implemented occupancy grid mapping algorithm in MATLAB has proven to be a useful method for creating detailed maps of the surrounding environment using LIDAR sensor data. By using this method, the robot has been able to effectively deal with uncertainties and noise in input sensor data, resulting in reliable maps useful for autonomous navigation and environmental understanding. One of the main advantages of occupancy grid mapping is its ability to deal with uncertainty. By assigning each cell in the grid a probability of being busy or free, the algorithm can make robust judgments even when the sensor data is uncertain or noisy. This is critical for robots to make safe and informed decisions when navigating complex. Furthermore, by using MATLAB, the algorithm can be easily adapted and extended for different applications and scenarios. Regardless of the positive result our algorithm could obtain, there are several suggested improvements for future work. A possible improvement is to optimize the algorithm for real-time applications which may mean reducing the computation time or improving the efficiency of the algorithm. The second suggestion is to investigate advanced sensor data fusion methods to further improve map accuracy and reliability.

The integration of occupancy grid mapping with path planning algorithms such as A* and Dijkstra provides a great solution for autonomous navigation. A* efficiently finds optimal paths by using heuristics to guide its search, leading to faster computations and shorter paths compared to Dijkstra's algorithm. However, Dijkstra's exhaustive approach ensures the shortest path but tends to be slower due to its lack of heuristic guidance. However, regarding the

0.45

(a) t

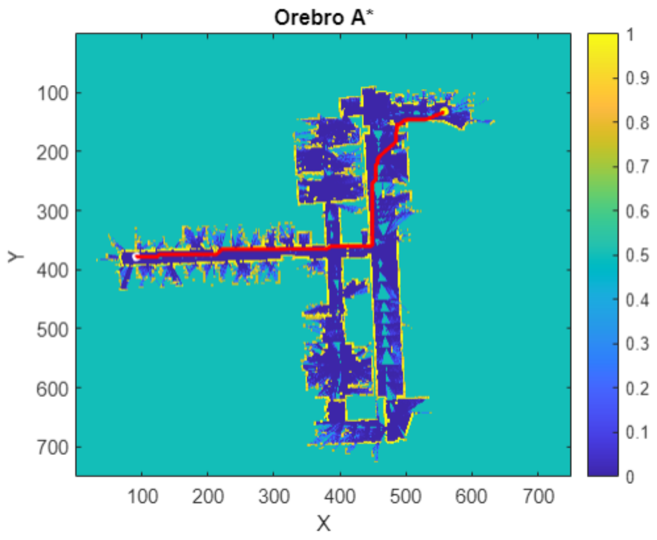

Figure 4. A* path planning of Orebro.

0.45

(a) t

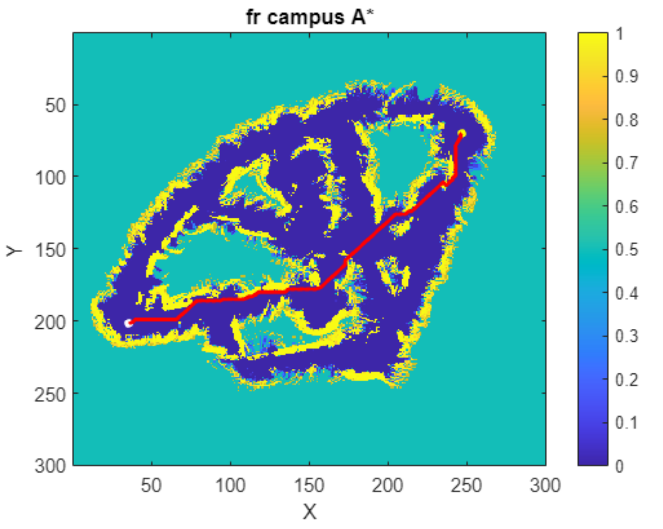

Figure 5. A* path planning of fr campus.

0.45

(a) t

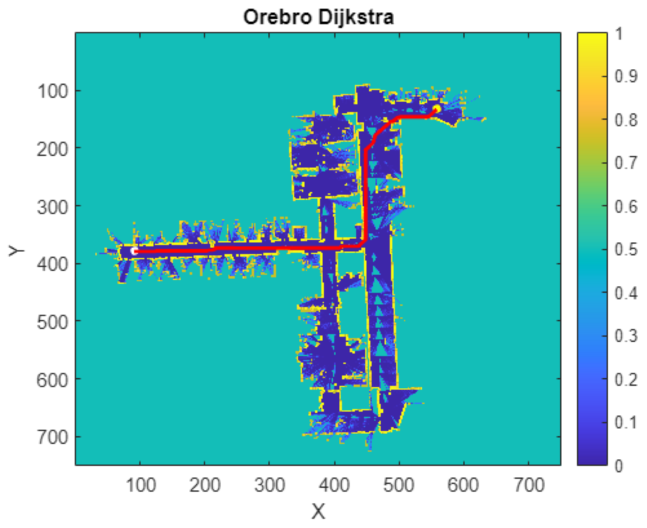

Figure 6. Dijkstra's path planning of Orebro.

0.45

(a) t

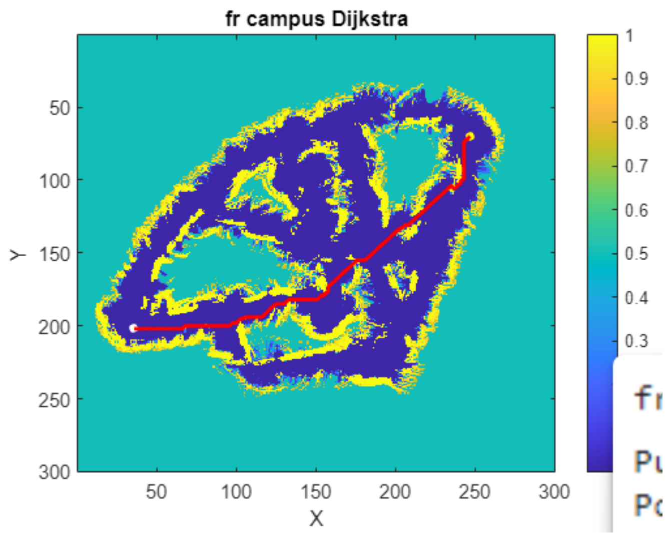

Figure 7. Dijkstra's path planning of fr campus.

result of the fr campus path planning, Diijkstra's algorithm was computed in less time. This could be due to multiple factors, such as it being more sparse, having more uniformly free cells, and the choice of start and goal cells. In the more enclosed environment of Orebro, A* clearly illustrates it's performance over Dijkstra's algorithm.

An important parameter in our algorithm is GridResolution, which refers to the size of each cell in the grid. A higher GridResolution means that more cells represent the same obstacle, giving a more detailed map. For example, if the GridResolution is 10 cm, it means that each cell is 10x10 cm instead of 1x1 meter. This allows the robot to navigate closer to obstacles, as more cells are calculated at a finer resolution instead of just one cell representing a larger area. However, the disadvantage of a higher GridResolution is that it requires more computation time because more cells must be updated and managed. On the other hand, a lower GridResolution results in faster calculations but less detailed maps, which can affect the robot's ability to navigate near obstacles safely. Consider an obstacle that occupies an area of 50x50 cm with a GridResolution of 1x1 meter, the entire obstacle would be represented by a single cell, which means that the details of the exact shape and location of the obstacle are lost. With a GridResolution of 10x10 cm, the same obstacle would be represented by 25 cells (5x5), providing a much more detailed and accurate representation of the obstacle in the map. This enables the robot to navigate with greater precision and to avoid obstacles more safely.

However, the downside of using a higher GridResolution is that it increases the computation time and resources required to update and manage the map. When the grid resolution is higher, the map consists of more cells, each representing a smaller area of the environment. This will lead to the system needing to process a larger number of cells during path planning. This increased processing load results in longer computational times and higher memory usage. The path planning was done in static environments, but computational cost is still important.

Finding a balance between GridResolution and computational resources is therefore an important aspect of our algorithm. By exploring and trying with different GridResolution settings, we can optimize map accuracy and performance for different applications. This is an important area for future work and development, where we can explore how different sensor data and environments affect the choice of GridResolution and how we can improve the efficiency of the algorithm without compromising the quality of the map.

References

[1] "path-motion-planning," 2024, accessed: 2024-06-02. [Online]. Available: https://www.mrpt.org/tutorials/programming/path-motion-planning/path_planning_over_occupancy_grid_map/.

[2] "Occupancy$_{g}rid_{a}lgorithm,''$ 2024, $accessed:$ $2024-06-02.[Online].Available:$

Accessed: 2024-06-02. [Online]. Available: https://www.ipb.uni-bonn.de/datasets/

"occupancy-grids," 2024, accessed: 2024-06-02. [Online]. Available: https://se.mathworks.com/help/robotics/ug/occupancy-grids.html.

"Occupancy$_{g}rid_{m}ap,''$ 2024, $accessed:$ $2024-06-02.[Online].Available:$

"Stachnisslab$_{(}uni-bonn.de),''$ 2024, $accessed:$ $2024-06-02.[Online].Available:$

""og-map: An efficient robocentric occupancy grid map for large-scene and high-resolution lidar-based motion planning." 2023, accessed: 2024-06-02. [Online]. Available: https://arxiv.org/pdf/2302.14819