Comparison between A* and Dijkstra Algorithms using Occupancy Grid in an Autonomous Vehicle

by

Abrish Sabri

Bachelor Thesis in Computer Science

Prof. V. Confused
Bachelor Thesis Supervisor

Date of Submission: May 14, 2019

Jacobs University — Focus Area Mobility

With my signature, I certify that this thesis has been written by me using only the indicates resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Signature Place, Date

Abstract

Consider this a separate document, although it is submitted together with the rest. The abstract aims at another audience than the rest of the proposal. It is directed at the final decision maker or generalist, who typically is not an expert at all in your field, but more a manager kind of person. Thus, don't go into any technical description in the abstract, but use it to motivate the work and to highlight the importance of your project.

(target size: 15-20 lines)

Contents

1 Introduction

Map is a spatial model of a robots environment and the process to build a map is called mapping. Occupancy grid mapping refers to a family of computer algorithms in probabilistic robotics for mobile robots which addresses the problem of generating maps from noisy and uncertain sensor measurement data, with the assumption that the robot pose(position and orientation) is known [1]. Information about the environment can be collected from sensors in real time or be loaded from prior knowledge. Laser range finders, bump sensors, cameras, and depth sensors are commonly used to find obstacles in the robots environment.

Occupancy grid map is an array with occupancy variables. The term Occupancy is defined as a random variable. A random variable is a function that maps the sample space to the real numbers. Each element of an occupancy grid is represented with a corresponding occupancy variable which is an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. Occupancy grid mapping requires bayesian filtering algorithm to maintain an occupancy grid map. Bayesian filtering applies recursive update to the map. A robot can never be certain about the world so we use the probabilistic notion of the occupancy instead of occupancy itself [1].

Method that is using occupancy grid divides area into cells (e.g. map pixels) and assign them as occupied or free. One of the grid cell is marked as robot position and another as a destination. All the grid cells are independent of each other. The occupancy probability if the grid cell is occupied is: $p(m_i) = 1$, if the grid cell is not occupied: $p(m_i) = 0$, and if there is no given knowledge of the grid cell: $p(m_i) = 0.5$. The state of each grid cell is assumed to be static.

Finding the trajectory is based on finding shortest line that do not cross any of occupied cells. This is a challenging problem in developing autonomous systems because minimising the risk of collisions takes away the efficiency of most navigation strategies. This problem is subject to uncertainty and partial information regarding the state of the vehicle, the obstacles, and the responses of the vehicle to inputs. Robust strategies for safe and efficient navigation require replanning to compensate for uncertainty and changes in the environment. The success of such strategies is dependent on the quality of sensing, planning and control, and on the temporal interactions between those tasks [2]. This work involves comparing two algorithms tested in a simulation that incorporates these sensors to find the most efficient trajectory plan. It focuses on finding a continuous path that can be followed from an initial configuration (or state) to a goal configuration. A safe path is one that prevents collisions with obstacles, and an efficient path is one which minimises cost.

For this thesis, I compare Dijkstra and A* in the simulation. Dijkstras algorithm is a classic algorithm for finding the shortest path between two points due to its optimisation capability. Dijkstra is a breadth-first-search (BFS) algorithm for finding the shortest paths from a single source vertex to all other vertices. It processes vertices in increasing order of their distance from the source, which are also called root vertices. The shortest path between two vertices is a path with the shortest length (i.e. least number of edges), also called link-distance.

- Let $G = (u, v)$ be a weighted undirected graph, with weight function $w : E \mapsto R$

mapping edges to real-valued weight. If $e(u, v)$, then we write $w(u, v)$ for $w(e)$.

- The length of a path $p =< v_0, v_1, v_2, v_3...v_k >$ would be the total of the weight of its constituent edges as in:

$$length(p) = \sum_{1}^{k} w(v_{i-1}, v_i)$$

- The distance from u to v, denoted by $\delta(u, v)$ is the length of the minimum path if there is a path from u to v and ∞ is otherwise.

The general idea of Dijkstras algorithm is to report vertices in increasing order of their distances from the source vertex while constructing the shortest path tree edge by edge; at each step adding one new edge, corresponding to the construction of the shortest path to the current new vertex. This is accomplished in the following steps:

- Maintain an estimate $d[v]$ of the $\delta(s, v)$ of the shortest path for each vertex v.
- Always $d[v] \geq \delta(s, v)$ and $d[v]$ equals the length of a known path ($d[v] = \infty$ if we have no path so far).
- Initially, $d[s] = 0$ and all other $d[v]$ values are set ∞. The algorithm will then process the vertices one by one in some order. The processed vertex's estimate will be validaated as being the real shortest distance; i.e. $d[v] = \delta(s, v)$.

The term processing a vertex u means finding new paths and updating $d[v]$ for all $v \epsilon adj[u]$ if necessary. The process by which an estimate is updated is called relaxation. When all vertices have been processed, $d[v] = \delta(s, v)$.for all v [3]. The path computed using the classic Dijkstras algorithm is the shortest; however, it may not be the most feasible.

The A* algorithm is an informed search procedure. It computes the shortest path between two nodes in a graph. While Dijkstra's algorithm blindly chooses the next node available, the A* algorithm uses heuristic that estimates the distance from any node to the target to choose the best node leading to the target. This estimate acts as a archetype for the algorithm and speeds up the computation.

During the A* algorithm, each node has one of the three following states:

- **The node is unknown:** The node has not been processed yet and no way from the start to this node is known.
- **The node is in the priority queue:** We know some way leading to this node, but there may be a shorter way.
- **The node is fully processed:** We know the shortest path from the starting node to it.

The algorithm first adds the starting node to the empty priority queue. Each node in the priority queue has an f-value. This value is the sum of the distance from the starting node to this node and the estimate of its distance to the target node. The node with the smallest f-value leads the priority queue and will be processed next. The algorithm now takes the node with the minimum f-value from the priority queue until the queue is empty or a path to the target node has been found. If the node taken from the queue is the target node, then the algorithm has found the shortest path and terminates. If the priority queue becomes empty, then no path from the start to the target is possible and the algorithm terminates. After processing a node from the priority queue, its neighbors are inspected. The algorithm distinguishes three cases:

- **The neighbor has already been processed:** Then the algorithm does nothing.
- **The neighbor is already in the priority queue:** If the current path is a shortcut, update its f-value.
- **The neighbor is not in the priority queue:** Compute the f-value of the node and add it to the priority queue.

Each time when updating the cost of some node, the algorithm saves the edge that was used for the update as the predecessor of the node. At the end of the algorithm, the shortest path to each node can be constructed by going backwards using the predecessor edges until the starting node is reached. If a node cannot be reached from the starting node, then its cost stays infinite [4].

Dijkstras Algorithm is guaranteed to find the shortest path given the input graph. A* is guaranteed to find the shortest path if the heuristic is never larger than the true distance. As the heuristic becomes smaller, A* turns into Dijkstras Algorithm. As the heuristic becomes larger, A* turns into Greedy Best First Search [5].

The major disadvantage of Dijkstras algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources. Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path. Dijkstra's algorithm has an order of n^2 so it is efficient enough to use for relatively large problems. The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources and as a result it generally makes it slower than A* [6].

2 Statement and Motivation of Research

In recent years, a whole lot of conclusive research results have been concluded using Dijkstras Algorithm and A* (which is an optimized version of Dijkstra Algorithm) to be the most fitting method for path planning in autonomous vehicles. [3] proposes a robotic path planning using a multilayer dictionary which provides more comprehensive data structure for Dijkstras algorithm in an indoor environment application where GPS coordinates and compass orientation are unreliable. Another group of researchers in [7] analyzed node-based optimal algorithms based on 3D path planning. Node based optimal algorithms deal with nodes and arcs weight information. Their task is to find the optimal path through calculating the cost by traversing through the nodes. Essentially, two main examples of node-based optimal algorithms are Dijkstra and A*. Their survey concluded these type of algorithms cannot be further optimize the result beyond the decomposition of the environment. The results of these kind of algorithms rely much on the preconstructed graph and can be combined with other methods to achieve global optimal. [8] developed an interpolation based method for ideal cost-to-go function based on Dijkstra and A*. It produced an effective method for estimating feedback of a plan and determined the shortest path for motion over a simplicial complex of an arbitrary dimension. The paper also demonstrated that the computational cost is significantly reduced by implementing an A* like heuristic.

There also has been some extensive research comparing the two algorithms A* and Dijkstra in terms of their computational cost, efficiency and simplicity. After considerable amount of analysis of the two algorithms [6] Dijkstra is significantly similar to A*, except there is no heuristic; H is always 0 because of which the algorithm expands out equally in all directions. In other case, A* scans the area only in the direction of destination. Thus, Dijkstra ends up exploring a sizeable area before the target is found making it slower than the A*. However, both of the algorithms have their own importance; Dijkstra is mostly used when the destination of the target is unknown while A* is mainly applied when both the source and the destination is known. As an illustration, there is a delivery service unit that needs to pick up some delivery trucks of some kind. It may know where several truck areas are but it wants to go to the closest one. Here, Dijkstras is better than A* because we dont know which one is closest. The only alternative is to repeatedly use A* to find the distance to each one and then choose that path. There are probably countless similar situations where the location whereabouts might be known but not know where it might be or which one might be closest. Therefore, A* is better when we both know the initial point and final point. A* is both thorough (finds a path if one exists) and optimal (always finds the shortest path) if the admissible heuristic function is used.

In the field of path planning for motion in virtual environments and artificial intelligence such as games, there have been some group of researchers focusing on navigation to find the optimized path [[9],[10],[11]]. The authors in [10] compared various path finding algorithms in unmanned aerial vehicles for detecting targets and keeping them in its sensor range in various environments and their performance compared to establish and monitor a path for communication. K. Khantanapoka and K.Chinnasarn [11] compared the path finding algorithms in intelligent environments in order to find the contrast in their time and space complexity. Sathyaraj et al. [9] developed a path planning strategy which let the randomly deployed autonomous robots in the environment move forward till an obstacle is met. For each agent (robot), an estimation of the relative location of the obstacle nearby according to the measurements of infrared sensors was deduced which let the common agent diverge before collision. Since common agents are used to imitate real people, this moving strategy satisfies the real situation that people walk around. For pursuer, it needs to navigate to specific position when evader is located therefore the researchers applied Dijkstra algorithm for path planning.

On the contrary, instead of focusing only on finding the optimized path using the Dijkstra and A*, the proposed method in this thesis uses the sensors implemented in the simulation. This thesis is based on a simulation in the loop procedure that incorporates real observations into the simulation in an effortless manner by synchronization of simulated conditions with real-world data. The simulation helps in analyzing and optimizing critical components like robot localization considering the components behavior in deep sea robotic operations and shows the benefit of the presented simulation in the loop approach on the DexROV research project. The SIL framework synchronizes simulated and real-world data by incorporating environmental and spatial feedback collected from field-trials which provides an augmented virtual environment reflecting environmental/spatial conditions from real-world missions to test, benchmark and compare behaviors of system modules, preserves the benefits of continuous system integration to perform such benchmarks using real or simulated components or a combination of both and allows to perform tests on distributed deployment, interfaces/pipelines, data regressions/degradation, and fault recovery/safety [Appendix A]. This simulation is provided by Jacobs University Bremen under the supervision of Dr. Francesco Maureli, Dr. Szymon Krupinski and Arturo

Gomez Chavez.

The proposed method takes advantages of the prior knowledge that the sensors provide such as camera view, position, resolution, height and width of the grid. Since the simulation illustrates an underwater vehicles environment with a vessel nearby, the physics of the vehicle is disabled in order to get a more precise measurement of the mapping for simplicity. Additionally, making this measurement off-center than the real world calculations. For simplicity, it is assumed that the prior knowledge of the simulation/ map services is believable. The occupancy grid map is based on real-time and it updates with the detection based on camera image. The contribution focuses on planning the global path for autonomous navigation, however, in this thesis, the focus will be in detecting which grid cell is occupied or free in the occupancy grid and the time it takes to find the optimal path from starting position to the goal.

3 Representation of the Data

As mentioned above in this paper, the two algorithms implemented are Dijkstra and A* for finding the optimized path for the robot used in the simulation. A generative implementation of these two algorithms could have been constructed by explicitly coding basic rules in some logic or formal grammar. However, due to the utility of the simulation extended expertise was required. It is of crucial importance how data set is represented and what information is retrieved from the data set. Subsections 3.1, 3.2 and 3.3 describe the representation of simulation performed, data collected, and data utilized, respectively. Subsection 3.4 describes the procedures of selecting and manipulating the data obtained through the simulation.

3.1 Representation of Simulation performed & ROS

Since the simulation is part of the DexRov research project, evidently it is run on the ROS. ROS (Robot Operating System) provides tools and libraries to help software developers simplify the task of creating complex and robust robot behavior across a variety of robot applications. ROS greatly aids in displaying the valuable data needed to perform the task required such as width and height between the travelled distance, resolution, the initial position of the robot and the goal it needs to reach. By running the command rostopic echo /projected_map the data is presented in the form of pose (position and orientation) [1]. The simulation is initiated by launching a vehicle, vessel and octomap launch file; followed by starting Rviz (ROS visualization), a 3D visualizer for displaying sensor data and state information from ROS. It displays live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data, and more. Rviz is an essential part of this research paper. The robot moves through a joystick and the software displays the live representation of the trajectory mapped by the robots movement [2].

```
#INITIAL POSITION IN METERS (START)
pose:
  position:
    x: 10.6115179532
    y: -12.484507967
    z: 2.94719725414
  orientation:
    x: -0.00106181641047
    y: 0.00037097801211
    z: -0.943184935889
    w: -0.332266326362
---
```

```
#FINAL POSITION IN METERS (GOAL)
pose:
  position:
    x: 1.21908376738
    y: 4.85488249354
    z: 2.88283420207
  orientation:
    x: -0.000842838538483
    y: -0.000929650401635
    z: -0.67355026858
    w: 0.739140352754
---
```


Figure 1: Data set displayed for the source and destination of the robot through Rviz

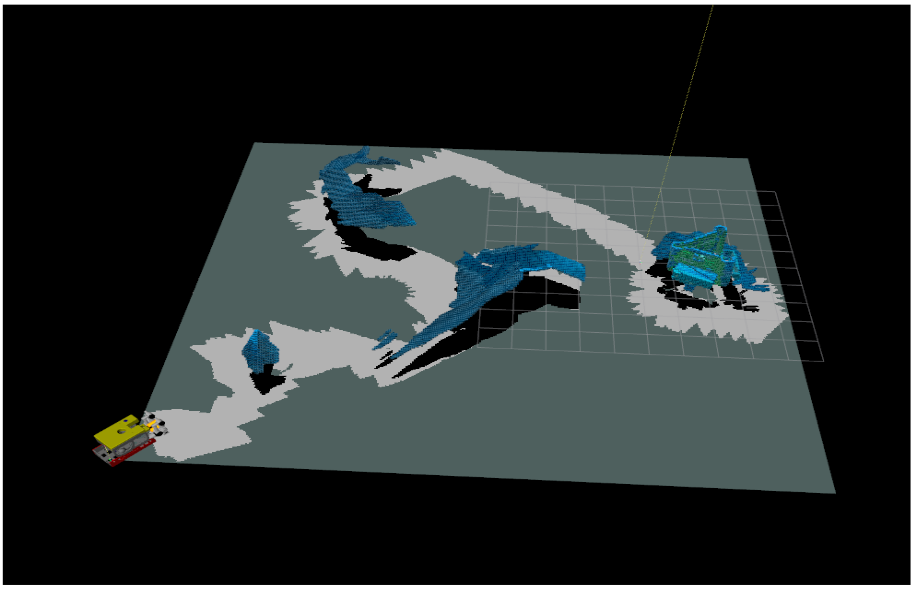

Figure 2: Projected map rendered via Rviz

Gazebo is used to simulate the robots route. This software produces the actual simulation of the underwater environment where the robot, vessel is shown by a third body. This part of the software involves disabling the physics of the environment. The software acts as an eye while Rviz collects all the data and utilizes this information to produce an occupancy grid and an octomap [3].

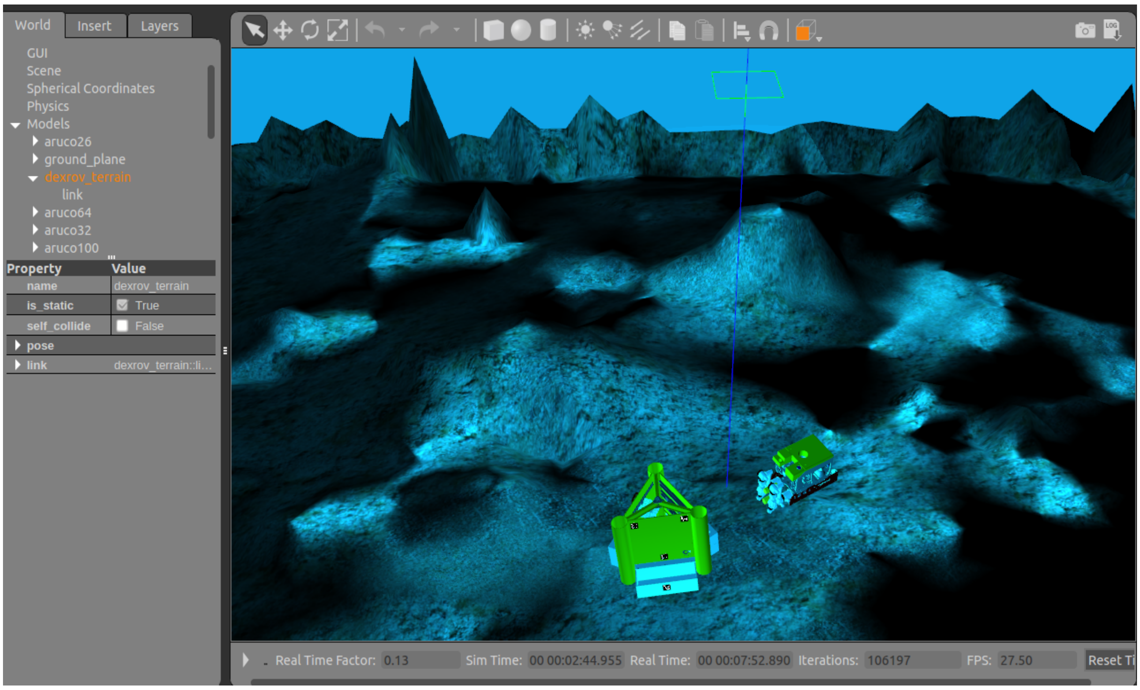

Figure 3: Simulation of the environment via Gazebo

3.2 Representation of Data:

The data is displayed in left tab of Rviz. It exhibits vital things about the ROS Topics that have been enabled and are currently running. For this simulation, there are at least 150 ROS topics active varying from stereo, depth, and underwater camera / left and right , each having their numerous parameters such as image_raw, state, rotation, etc. Topics containing information about rov, gazebo, octomap, projected map are also active. The data collected for this research exists of stereo camera, ROV model, projected_map, pose, grid, and octomap. Rviz also generates a text file that displays the map as a 1D array. The text file is indeed applied in the usage of dijkstra and A* algorithm. The data obtained in the text file is displayed as an array of 100, 0 and -1. 100 implies that the cell in the map is occupied/blocked, 0 implies that the cell is free/empty while -1 implies that the cell is unexplored. Based on this information the data is further probed on.

3.3 Representation of Data Set/Text file:

The data set is displayed in a 1D array. The data is converted into a 2D array by using the width and height values given. This data is then converted to be represented as a graph to be used in the path planning algorithms by traversing through each cell one by one and checking the cells neighbours. Logically, each cell would have 8 neighbours at most. The graph is formulated by comparing the values between the neighbour cell and the current cell. If the value of the cell in the map is 100, 0 , -1, in the graph it would have a value of 100,1,50, respectively. After creating the graph, the shortest path and the best path from source to the destination is found respectively through the two path planning algorithms. The path (display of various cells) is extracted from the cells in the map and multiplied by the resolution. These values are stored as X and Y values into a YAML file to be used in the launch file.

4 Evaluation of the Investigation

This section discusses criteria that are used to evaluate the research results. Make sure your results can be used to published research results, i.e., to the already known state-of-the-art.

(target size: 1-3 page)

5 Conclusions

Summarize the main aspects and results of the research project. Provide an answer to the research questions stated earlier.

(target size: 1/2 page)

References

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[2] Andrew B Walker et al. Hard real-time motion planning for autonomous vehicles. *Walker, A.—" Hard Real-Time Motion Planning for Autonomous Vehicles PhD thesis, Swinburne University*, 2011.

[3] Syed Abdullah Fadzli, Sani Iyal Abdulkadir, Mokhairi Makhtar, and Azrul Amri Jamal. Robotic indoor path planning using dijkstra's algorithm with multi-layer dictionaries. In *2015 2nd International Conference on Information Science and Security (ICISS)*, pages 1–4. IEEE, 2015.

[4] Lisa Velden. An informed search for the shortest path.

[5] Introduction to a*.

[6] Abhishek Goyal, Prateek Mogha, Rishabh Luthra, and Neeti Sangwan. Path finding: A* or dijkstras. *International Journal in IT and Engineering*, 2(01), 2014.

[7] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, 2016:5, 2016.

[8] Dmitry S Yershov and Steven M LaValle. Simplicial dijkstra and a* algorithms for optimal feedback planning. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3862–3867. IEEE, 2011.

[9] Mengzhe Zhang. *Path planning for autonomous vehicles*. PhD thesis, 2014.

[10] B Moses Sathyaraj, Lakhmi C Jain, Anthony Finn, and S Drake. Multiple uavs path planning algorithms: a comparative study. *Fuzzy Optimization and Decision Making*, 7(3):257, 2008.

[11] K. Khantanapoka and K. Chinnasarn. Pathfinding of 2d 3d game real-time strategy with depth direction a algorithm for multi-layer. In *2009 Eighth International Symposium on Natural Language Processing*, pages 184–188, Oct 2009.

A title

B title 1