Wheeled Waiter Robot Path Finding

Submitted to:

Dr. Hereid

Created by:

Samuel Arigo, Randy Jung, Eshwar Pamula

ME/ECE 5463, SP 2024 (9637)

The Ohio State University

Columbus, Ohio

4/22/2024

Introduction and Problem Statement:

In recent years, there has been a noticeable trend towards automation in various industries. In particular, the hospitality sector is exploring more automation to integrate it more into their fields to replace the human workforce. The restaurant industry, in particular, oftentimes can pose challenges to its employees. Examples of such challenges include the physical demands like walking around non-stop for hours, carrying large, heavy, and even hot trays of food, and having to navigate through busy aisles which could all lead to stress and/or injury. Any error on the part of the employee can lead to an accident that could affect both customers and employees alike. Additionally, workers can endure mental challenges as well due to rude and unsavory customers who blame everything that goes wrong on the waiter. In response to these challenges, a desire to automate a waiter's tasks has grown to provide improved efficiency, safety, precision, and optimization. By developing an automated mobile robot tailored to the needs of a restaurant, the overall quality of service will improve while reducing the risks posed. The team's objective is to design a robot that can do just that by developing a navigation system to allow it to quickly and efficiently go from one point in the restaurant to another while avoiding all obstacles that might be in its path.

Literature Review:

To perform the duties of a waiter, a mobile robot must be able to determine an optimal path through a restaurant layout. Babatope, Samuel, et al. explored and compared prevalent mobile robot path planning algorithms. The paper made a distinction between global and local path planning by stating that "Global path planning algorithms are frequently applied to static workspaces having static obstacles and the robot has comprehensive knowledge of this workspace," whereas local path planning involves real-time path decisions based on sensor data [1]. Assuming that a restaurant layout is fixed and already known, global path planning

algorithms seem suitable. Popular global path planning algorithms explored in [1] include Dijkstra's Algorithm and the related A* Algorithm, which can find optimal paths through a map consisting of a known set of nodes and node connections.

An example of Dijkstra's Algorithm can be imagined from Figure 1, where Umam, Faikul, et al. designed a set of nodes and node connections to represent a restaurant's aisles (left) and utilized Dijkstra's Algorithm to compute optimal paths in the corresponding path graph (right), weighted by the distance between the nodes. A waiter robot used the algorithm to travel from, as an example in the paper, node 0 to node 7 via the shortest path in the path graph: 0 to 1 to 8 to 7 with a route length of 1581 [2].

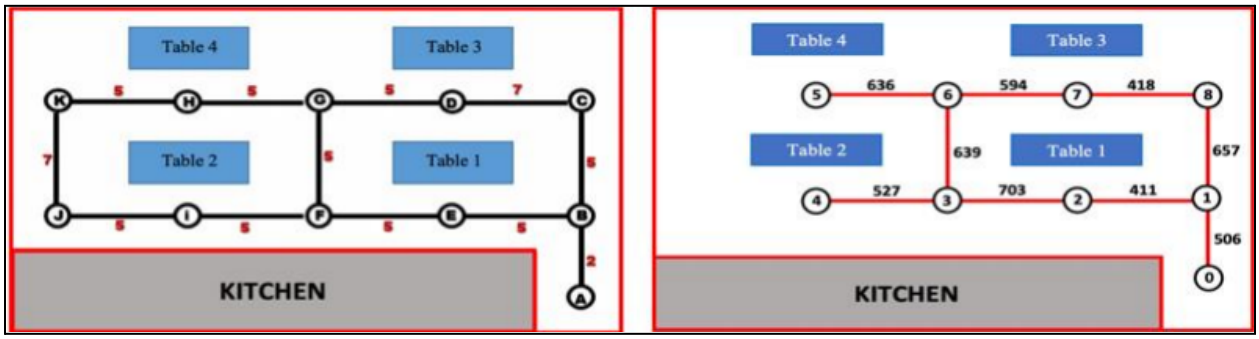

Figure 1. Path Graph for Dijkstra's Algorithm in a Restaurant Layout
This figure depicts a set of nodes and node connections to represent a waiter robot's available walkway (left) and the related path graph where Dijkstra's Algorithm was successfully applied in [2] (right).

Now that desired global paths can be identified in a static restaurant environment, the design of a waiter robot itself must be considered such that it can accurately track these paths. By utilizing MATLAB's Simulink, Corke, Peter, et al. modeled a controller that outputs tracking velocities and steering angles based on a desired arbitrary path. A wheeled robot was simulated under this controller for a desired path and its motion was plotted: Figure 2 (left) depicts the desired path in dotted red and the actual path taken by the modeled robot in blue [3].

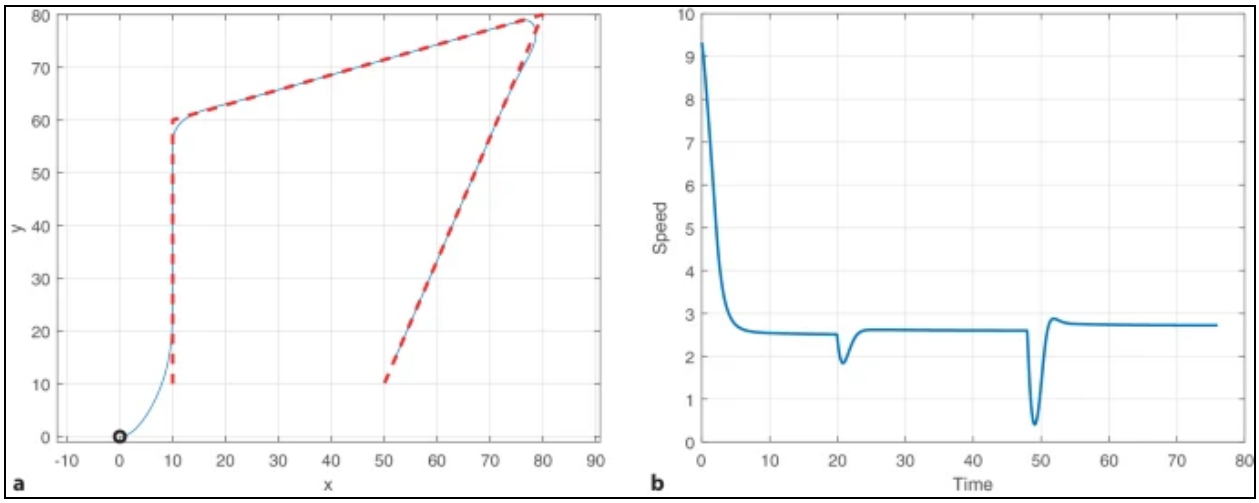

Figure 2. Control System Performance for Tracking a Desired Path

This figure illustrates the performance of the Simulink control system in [3]. The left plot compares an arbitrary desired path (dotted red) to the mobile robot's actual path taken during the simulation (blue). The right plot depicts the speed of the mobile robot during the course of the simulation.

Tracking error can be interpreted as the difference between the desired path and the robot's actual path. Notice how the speed dips as the robot makes corner turns in Figure 2 (right). Also, notice how the robot did not perfectly track the red-dotted path at these turns or near the beginning of the simulation at the origin. This is due to the nature of the controller designed by Corke, Peter, et al., which employed a PID (proportional integral derivative) controller and utilized a negative feedback loop of the robot's xy position [3]. The design minimized the tracking error in the steady state (for one desired velocity and direction) but still experienced transients when turning (for changing the velocity or direction). A proper controller must be designed such that tracking is accurate and that the transient arc paths do not cause collisions with, for a restaurant application, the corners of a table.

Although the wheeled robot used in Figure 2 was based on a bicycle kinematic model, a differential-drive mechanism may be used for a waiter robot. A simple differential-drive robot travels by changing the rotation speeds of its two wheels independently. A controller's tracking

velocities and turning rates can simply be translated to the rotation speeds of a differential-drive robot's two wheels. The equations for this conversion are discussed in [3].

So far, only a known static environment has been considered for a restaurant layout. In reality, customers may serve as unknown dynamic obstacles. Hu, Huosheng, et al. proposed a method of deviating from a predefined path to avoid nearby moving obstacles. The paper heavily relied on statistics, expounding on the uncertainties of both a dynamic obstacle's position and a mobile robot's sensor measurements. An ellipse surrounding an obstacle conveys a region where it most likely exists, much like an electron probability cloud. A safety circle was also established around the mobile robot. The concept was to predict a path that would not allow the ellipse and the circle to intersect [4]. A simpler method, both conceptually and computationally, comes in the form of the fuzzy logic method, which "uses human-supplied rules (If-Then) and converts these rules to mathematical" implementations [1]. Ghaemi, Sehraneh, et al. created a fuzzy controller for a mobile robot to avoid dynamic obstacles. The paper defined certain states for the robot and obstacles, based on sensor data. For example, if an obstacle is NEAR, SLOW, and to the LEFT of the robot, then set the mobile robot to go SLOW and go RIGHT. These instructions were translated (or "defuzzified") to values that could control the robot [5]. Such a fuzzy logic method can be used to make local path decisions to avoid restaurant customers. For example, a fuzzy controller could have an If-Then statement that directs the waiter robot to stand off to the side of a restaurant aisle and allow people to walk past.

Proposed Approach:

The team's approach to this proposed challenge is to model the waiter mobile robot in MATLAB. In this simulation, the robot will move from a set starting point to one of several fixed endpoints to represent starting in the "kitchen" and ending at a "table". As it moves, it will have to avoid both static and dynamic obstacles. The team will initially begin with static obstacles

before adding dynamic obstacles once the robot can avoid static objects without fail. These obstacles will be located in a simulated restaurant with a set layout that does not change in between runs to keep the robot in a consistent setting. To do this, the team plans on utilizing the A* Algorithm to allow for global path planning as this will allow the robot to path plan around the static obstacles. Then, to avoid dynamic obstacles, the team plans on using fuzzy logic for local path planning. The plan will be a hybrid of those two strategies as it will allow the robot to easily avoid any obstacles in its path safely and efficiently.

Algorithm Development:

After consideration from the team, they decided to scale down their approach to focus primarily on the static obstacle avoidance of the robot. The new approach was designed around using the A* method in combination with a control system and a differential drive representation of the robot. The goal behind this design was to use A* to find the path for the robot, the control system to determine the linear and angular velocity which the robot must go, and the differential drive to provide the wheels with their different corresponding velocities. Absent from this design is the avoidance of dynamic obstacles through the use of fuzzy logic. The team determined that it would have been too challenging and time-consuming for the length and scope of the project presented to them so they decided to focus on improving the static avoidance instead.

The A* algorithm is a method of pathfinding that finds the shortest path between two points on a given graph or, in the case of this project, a map. The algorithm takes the two points and goes from node to node, considering the cost and estimating the remaining distance at each node. A* prioritizes the nodes with a lower total estimated cost, using those nodes to expand until the algorithm finds the cheapest possible path when it terminates. This method is guaranteed to find the shortest possible route between the two points, making it very advantageous to use when considering a purely static environment. It was for this reason that the

team chose to use this method. By turning the layout of the restaurant into a static "graph" or map, the robot could use the A* algorithm to determine the shortest path it would need to take, improving its efficiency.

To create the A* algorithm, the following pseudocode was used. The first step was to create two lists: "open list" and "closed list." The open list referred to the set of nodes to be evaluated while the closed list consisted of all the nodes that had already been evaluated. To start, the open and closed lists are empty and the current node is the start node. A loop is started, evaluating the current node, removing it from the open list, and adding it to the closed list. The neighbors of the current node are scouted. Any neighbor that is in the closed list or is not traversable is skipped. Among the available neighbors, if the neighbor is not on the open list or the new path to the neighbor is shorter than its old path, the neighbor's cost is established or reduced, and the neighbor is put into the open list if it was not already, and the "parent" of the neighbor is set to the current node. Finally, the point in the open list with the lowest total cost is the next current node. This loop will repeat until the current node is the end node, and the function will terminate as the shortest path has been found.

To calculate the total cost of each cell, two costs needed to be calculated and then summed. The first of these costs is the node heuristic (H) cost. A diagonal heuristic (the distance to the end node if diagonal movement is allowed) was used, which can be calculated using the following equations:

$$xDiff = abs(currentNode.x - endNode.x) \quad (1)$$

$$yDiff = abs(currentNode.y - endNode.y) \quad (2)$$

$$heuristicCost = D * (xDiff + yDiff) + (D2 - 2 * D) * min(xDiff, yDiff) \quad (3)$$

After getting the H cost, the node G cost then had to be added to get the total cost. This cost did not require any explicit equations. Instead, for any cardinal direction movement from a

current node, a neighboring node's G cost was set to 1 (D = 1) plus the current node's G cost, and for diagonal movement, $\sqrt{2}$ (D2 = $\sqrt{2}$) was added instead. Adding the H and G cost resulted in the total cost that the algorithm used in its decision-making.

In addition to the A* algorithm, a control system was used. A control system at its core is used to manage, direct, and regulate the behavior of the system which it is controlling. It collects and determines information about the different states of the system which are then able to be processed and returned to allow for adjustment of the system's behavior. In regards to this project, the control system was focused on the control of the angular and linear velocities of the robot. After getting a final path from the A* algorithm, the control system would be able to estimate and output values required for the linear and angular velocities of the drive. By doing this, the robot will be able to move much better and adapt to any kind of path or variation by adjusting the speed in real-time. MATLAB's Robotics System Toolbox contained a controllerPurePursuit object that was used to implement a controller for this project.

The differential drive model of the robot considers two different drive wheels independent of each other. These wheels are considered to have motors, so they can be controlled and powered separately. By using this model in correlation with the control system, the robot can adjust and adapt to any reasonable situation that it could be put into. It will primarily increase its ability to turn which is very important in ensuring that the robot makes it from its start point to its end point. MATLAB's Robotics System Toolbox contained a differentialDriveKinematics vehicle model that was used to implement the robot dynamics.

By combining the three different methods and algorithms, the team was able to create a good flow for the robot. It started off by using the A* method to determine the shortest possible path between the point at which it would be starting and the point at which it needed to end at. This path was guaranteed to be the shortest, ensuring that the robot would be as efficient as possible since it wouldn't be wasting any extra unnecessary movement. Once the path was

created, the control system used the given path to determine the wheel speeds at which the robot needed to go at each point along the path. These speeds were then used to control the differential drive as it moved along the path. This allowed the robot to move smoothly both linearly and when turning to significantly reduce the possibility of crashes or failures as it moved.

Simulation Results and Analysis:

The A* path-planning method was realized in MATLAB code. A matrix of ones and zeros was created to represent the fixed restaurant floor plan nodes, with ones representing traversable nodes and zeros representing non-traversable nodes (walls and tables). Additionally, the start node and end (desired) node were each defined as [i, j], where i represents the node row and j represents the node column in the map matrix. For the following demonstration, the map, end node, and start node can be seen in the left part of Figure 3. The path-planning algorithm considered diagonal movement in addition to cardinal directions (up, down, left, and right). The final simulation using these parameters and a diagonal heuristic can be seen in the right part of Figure 3.

```
%Define restaurant map. 0's are tables/walls
%The top left is defined as the origin
map = [ ...
    1, 1, 1, 1, 1, 1, 1, 1; ...
    1, 1, 0, 1, 0, 1, 0, 1; ...
    1, 1, 1, 1, 0, 1, 0, 1; ...
    0, 1, 0, 1, 0, 1, 0, 1; ...
    0, 1, 1, 1, 1, 1, 0, 1; ...
    1, 1, 0, 1, 0, 1, 0, 1; ...
    0, 1, 1, 1, 0, 1, 0, 1; ...
    0, 1, 0, 1, 0, 1, 0, 1; ...
    1, 1, 1, 1, 1, 1, 0, 1; ...
    ];

%Define a start and end (desired) node
desiredNode = [9,1];
startNode = [1,8];
```

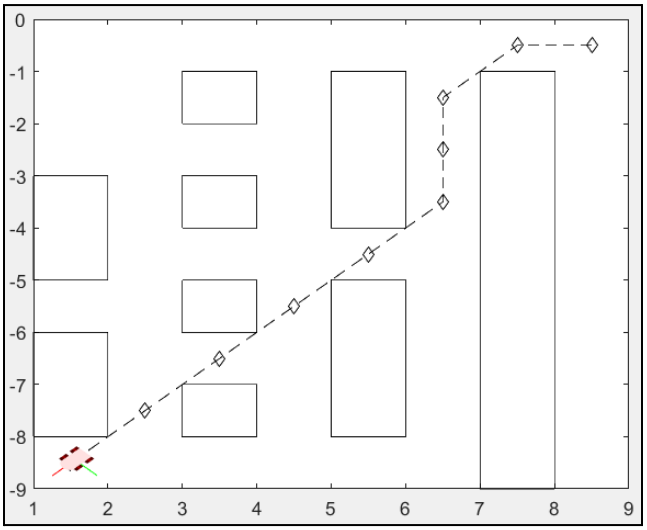

Figure 3. MATLAB Map, Parameters, and Simulation
This figure depicts the restaurant map and parameters (end and start nodes) defined in MATLAB code (left) as well as a snapshot of the trajectory-following animation (right). Ones represent traversable nodes and zeros represent

tables or walls. For this specific simulation, the desired end node was in row 9, column 1 (bottom left of the map) and the start node was in row 1, column 8 (top right of the map).

Intuitively, it can be said that the robot took the shortest path from the top right of the restaurant to the bottom left. Although the robot seems to have collided with the corners of tables and walls along its diagonal paths, the rectangles are simply models of obstacles that may be smaller in reality. For example, the original floor plan included oval-shaped tables, so there are no corners to collide with. For a more detailed analysis of the A* implementation, Figure 6 below depicts the F (total) costs of each node, which are the summations of the G costs and the H (heuristic) costs.

	1	2	3	4	5	6	7	8
1	Inf	Inf	Inf	Inf	13.4853	12.0711	11.4853	Inf
2	Inf	Inf	Inf	Inf	Inf	11.4853	Inf	10.8995
3	Inf	Inf	Inf	Inf	Inf	11.4853	Inf	11.4853
4	Inf	Inf	Inf	13.4853	Inf	11.4853	Inf	12.0711
5	Inf	Inf	13.4853	12.0711	11.4853	12.0711	Inf	Inf
6	Inf	13.4853	Inf	11.4853	Inf	13.4853	Inf	Inf
7	Inf	12.0711	11.4853	12.0711	Inf	Inf	Inf	Inf
8	Inf	11.4853	Inf	13.4853	Inf	Inf	Inf	Inf
9	11.4853	12.0711	13.4853	Inf	Inf	Inf	Inf	Inf

Figure 6. Node Total (F) Costs

This figure depicts the total (F) costs, or the sum of the heuristic (H) costs and the G costs. Up, down, left, and right directions cost 1 unit while diagonal movements cost sqrt(2) units. For this simulation, the start node was in the top right of the map [1,8] and the end node was in the bottom left of the map [9,1]. The final path is labeled in red. Inf values represent either nodes that are non-traversable (tables or walls), nodes that have not been explored yet, or the start node.

Notice how every node on the final path (labeled by red boxes in Figure 6) has the same total (F) cost of 11.4853, which represents the total path cost: a total of six diagonal movements (each costing sqrt(2) units) and three cardinal-direction movements (each costing 1 unit) results in about 11.4853. Any deviation from this path would result in a higher total cost or an obstacle

collision (marked by Inf in Figure 6, except for the start node). The algorithm also correctly detected that, even though the node at row 2, column 8 had the lowest cost (10.8995), its downward path eventually became too costly. That path is a dead end. A more in-depth analysis of the A* implementation can be seen in the associated video.

Conclusion:

Through the use of the A* method in combination with a control system and differential drive model, the team was successfully able to create a simulation of a robot that could perform the required task of moving around the static obstacles in a restaurant. This robot was able to navigate both efficiently and safely and was able to avoid any obstacles as it moved. By developing this functional navigation algorithm, the team was able to prove that there is lots of potential for a future of automated waiters. The implications of this success could also be spread to other forms of business as the algorithm is very adaptable and can be adjusted easily to account for any change in an environment.

If given more time to work and develop the algorithm, the team's next steps would be to incorporate fuzzy logic into the overall algorithm. This addition would give the robot the ability to avoid moving obstacles in addition to static ones. In real-life situations, this would let the robot move around people walking around while it delivers food. This would be a very important feature to allow for safety in the restaurant environment by avoiding accidents that might happen due to the collision of the robot and a customer.

Overall, this project proved that there is a promising future for automation in the restaurant field among others. This new technology will be invaluable moving forward and will benefit many people in their day-to-day lives. Working alongside technology as it grows is key to ensuring that both workers and customers benefit from the technology to create a better, safer, and more efficient workplace.

References:

[1] Babatope, Samuel, et al., “Quantitative Performance Review of Wheeled Mobile Robot Path Planning Algorithms,” Gazi University Journal of Science, pp. 765-785, Jan. 2021, https://dergipark.org.tr/en/download/article-file/1281982

[2] Umam, Faikul, et al., “Optimal Mapping Trajectory for Autonomous Robot Waiter,” Technology Reports of Kansai University, Vol. 62, pp. 6429-6435, Dec. 2020, https://simpelmas.trunojoyo.ac.id/backend/assets/uploads/lj/LJ20211011635739155548.pdf

[3] Corke, Peter, et al., “Mobile Robot Vehicles,” Robotics, Vision, and Control, pp. 127-166, May 2023, https://link.springer.com/chapter/10.1007/978-3-031-07262-8_4

[4] Hu, Huosheng, et al., “Navigation and Control of a Mobile Robot among Moving Obstacles,” Proceedings of the 30th IEEE Conference on Decision and Control, pp. 698-703, Dec. 1991, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7db3c6a402ce0bb9740929d128bb5ee1f1cce5b1

[5] Ghaemi, Sehraneh, et al., “A fuzzy multi-stage path-planning method for a robot in a dynamic environment with unknown moving obstacles”, Robotica, Vol. 33, pp. 1869-1885, May 2014, https://www.cambridge.org/core/journals/robotica/article/abs/fuzzy-multistage-pathplanning-method-for-a-robot-in-a-dynamic-environment-with-unknown-moving-obstacles/42F93113F5339A65AE7C91A342554CC7