Research Report: Evaluating Algorithms for a Robot Tasked with Waiting Tables in the Hospitality Industry

Team members: Zofia Dukała (S4778200) , Mekhola Doha (S4790650), Eman Ansari (S4317394), Katarzyna Kapuścińska (S4723708)

Supervisor: Dr. Marcelo Goulart

1. Begin by stating your research question and objectives, making sure they are clearly stated in the introduction of the report.

Research Question: Which algorithm is most suitable for an a-priori pathfinding robot tasked with waiting tables in the hospitality industry?

As the new wave of automation grows wider in scope, the range of tasks it is aimed towards also increases. Automation today is not only aimed to achieve routine tasks that can be codified into a series of steps, but rather also strives to incorporate more dynamic non-routine tasks such as self-driving cars or surgical robots. In this research, we focus on automating the task of waiting tables at a restaurant and more specifically, programming a robot optimized to carry orders to customers in the most efficient way. The main objectives of this research are then to a) identify the key requirements of such a robot, b) integrate two well known pathfinding algorithms, A* and Dijkstra's, capable of handling these requirements with the robot interface to, c) visualize and comparatively evaluate the performance of two algorithms using the Webots simulation environment and d) Determine which algorithm is optimum and what makes it better suited for the task.

2. State your academic deliverables, including the most recent and relevant research on the topic.

For our academic deliverable, we will create a presentation to showcase the workings of our maze-solving robot simulation within a restaurant layout designed in Webots. We will include key parts of the simulation to provide a clear visual understanding of the working principles behind our maze-solving robot.

In recent research, Alamri et al. (2021) explores the complex field of robotics and focuses in particular on the algorithms that enable autonomous robots to navigate and solve mazes. This study divides maze-solving algorithms into two categories: known environments, where the robot is already familiar with the layout of the maze, and unknown environments, where it must find its way through on its own. The paper provides a thorough assessment of the effectiveness and applications of several maze-solving algorithms, including Wall-follower, Tremaux's

algorithms, Flood-fill, Dead-end, and Maze-routing for known environments. Additionally, the paper discusses the design, system integration, and real-world deployment challenges of these algorithms as they are implemented in various robotic systems.

Alamri et al.'s classifications provide important insights when relating this research to our study on choosing the best algorithm for an a-priori pathfinding robot in the hospitality sector. Algorithms that can adapt to 'known environments' and be flexible enough to handle unforeseen obstacles are imperative in a restaurant setting, where the layout may be known but dynamic changes are common due to customer movement and table configurations. Therefore, it could be useful to incorporate and assess adaptive versions of well-known algorithms as the A* algorithm and Dijkstra's shortest path method using Webots simulation.

In the current landscape, machine learning and artificial intelligence are increasingly utilized to enhance the learning capabilities of robots across diverse sectors such as the hospitality industry. A recent study by Kulaç et al. (2023) emphasizes the advancements in expanding the role of traditional industrial and agricultural sectors into service-oriented applications, propelled by decreasing costs and high accessibility of robotics technologies. The study developed an algorithm for an autonomous mobile robot, which was programmed to map its environment, identify operational stations, and perform transport tasks between these stations according to work orders. This system was created using NI LabVIEW and tested in RobotinoSIM, then physically validated with the Robotino platform. The system utilized an RGB camera, odometry, and QR codes for precise tracking, enabling it to complete 50 work orders without making any mistakes. This research also further demonstrated its potential for broader service sector applications.

3. Clearly and concisely describe the methods used in the research, including the procedures, participants, and materials.

Requirements and Assumptions

For the purpose of simplicity, we designed this robot with the assumption that its only job is to collect the orders from a designated collection point (the point A in our original layout) and then deliver the orders to a given number of tables before coming back to its starting position. In delivering the orders, the robot must find the optimized shortest path, such that it is able to visit each table once and return to its starting position while ensuring the lowest cost. In this context, the cost is measured as the total distance it has to cover. The algorithms we chose to integrate for this task are the path-finding algorithms A* and Dijkstra’s. The robot’s task is to calculate the shortest path a-priori to any movement it makes for each sequence of orders entered by the user. This is because in a restaurant setting, it would not be time efficient for the robot to be finding the shortest path while delivering.

Additionally, we assume that any such robot would have a maximum capacity for the amount of orders it can carry at a time, which is determined by the hardware design it is subject to. Since our research is not focused on the hardware design as such, our programme has the added functionality to adjust the order carrying capacity of the robot depending on the number of orders received from the user input. Other logistic assumptions included are that a) there are no obstacles present in the path of the robot, b) orders from different tables are placed on different shelves of the robot in order for customers conveniently know which items to take, and c) the waiting time for each table's stop is dependent on a weight sensor placed on the shelves to indicate when items have been fully removed.

Restaurant layout

Given the aforementioned, we first begin with designing the layout of our restaurant. As visualized in fig 1.0, this design was intended to be compatible with a realistic floor plan of a restaurant. The figure was generated using Canva webtool.

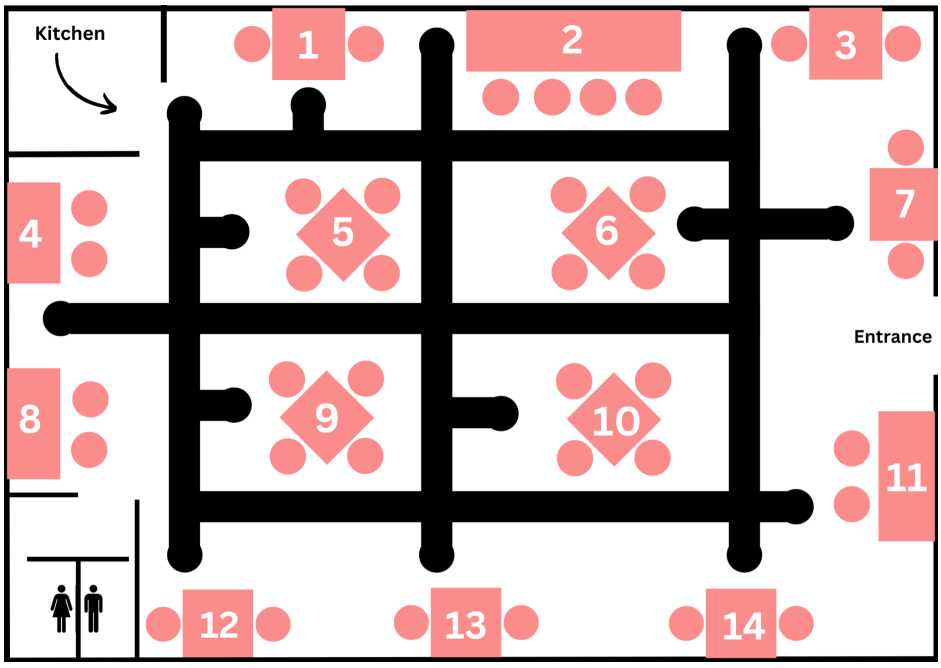


Fig. 1.0, Restaurant layout

Representation Methodology

The path of the robot is indicated by the network of nodes and edges in black. We proceed by adding this layout as our floorplan in our Webots world file. Since our chosen algorithms both require a graph with labeled nodes, edges to other nodes as well as their weights, we sought to manually create two dictionaries. The first dictionary (named "coordinates") contains the labels and euclidean coordinates of each node. In order to compile the correct "coordinates" dictionary,

we placed an e-puck robot with a GPS module at each node of the layout and recorded the respective x, y and z coordinates. This dictionary is then used to calculate the exact distances between each node. These distances then served as the weights assigned to the final dictionary (called “graph”) which contains each node, the edges connecting it to other nodes and the weights of each edge. Using the ‘networkx’ library in python, we visualize the graph as shown in fig. 1.1.

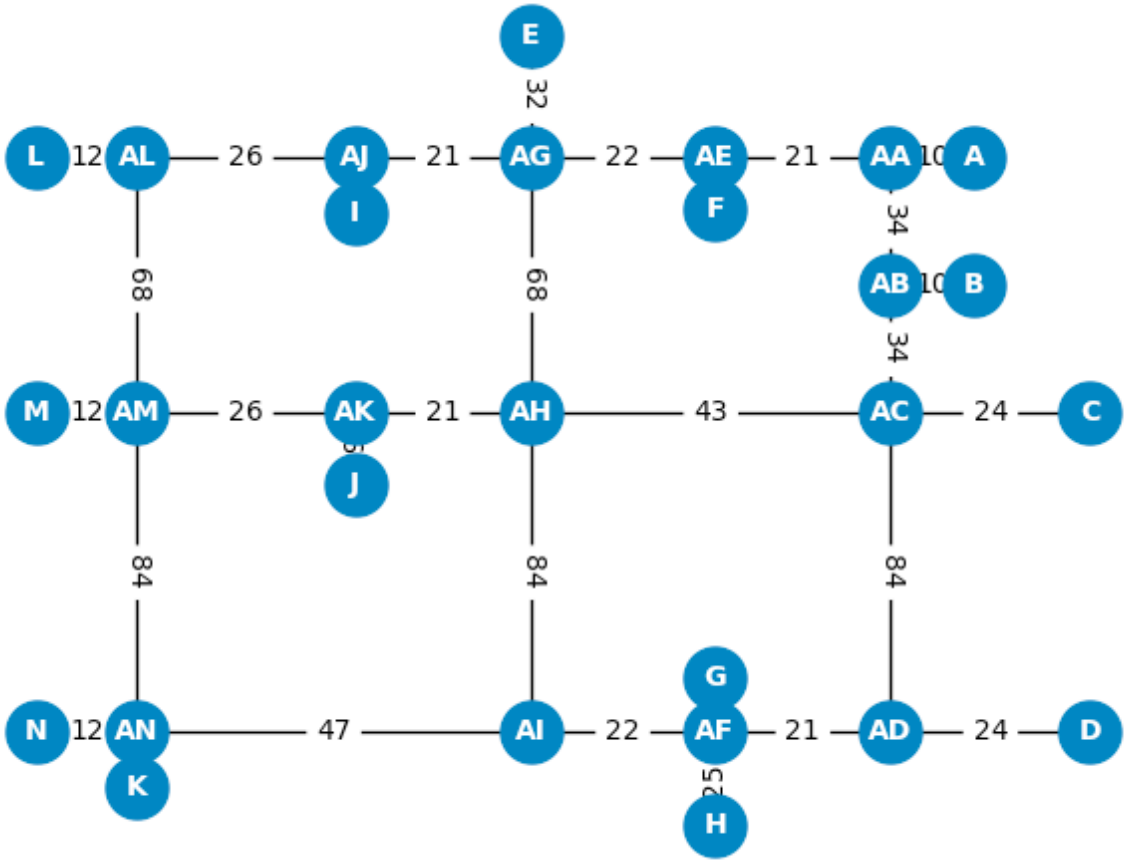


Fig. 1.1 Network visualization

Algorithm Integration into E-puck Controller

In our python implementation, we integrate the algorithms into two separate functions that take 3 input arguments, 1) a graph of the nodes, 2) a 2D list containing a sequence of destination nodes (representing the food orders placed on the robot for a single trip), 3) an initial node which is the fixed position representing the point near the kitchen where the food is collected. This function then outputs a sequence of tuples representing the shortest path that visits each node in the input list once before returning to the initial node. Lastly, we use this sequence of tuples as input to a

function that translates this sequence of nodes into their corresponding euclidean coordinates and is able to calculate the movements the robot must make (in terms of its exact speed and rotation) to arrive at each coordinate described in the list of tuples sequentially.

Algorithm Methodology

A search algorithm*

This algorithm is a powerful tool, which employs the graph traversal method. It is particularly useful for finding a shortest path from the current point to the goal, taking into consideration assigned weights. In the context of a restaurant simulation, where the task of the algorithm is to efficiently navigate the robot to serve food to customers, A* is particularly advantageous, since the heuristic function enables prioritizing paths that appear to lead more directly towards the destination, potentially reducing the overall journey time for the robot. The algorithm dynamically adjusts its pathfinding as it progresses, allowing it to effectively handle complex restaurant layouts and unexpected obstacles, ensuring the robot serves customers efficiently. The A* algorithm calculates the shortest path by using the following function:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the path cost from the start node to node n, while $h(n)$ is a heuristic estimating the path cost from node n to the goal. In the restaurant environment, the heuristic will depend on the allowed movements in the grid we designed. $f(n)$ then represents the total cost of the path. A* searches the nodes that have the lowest $f(n)$ values first. The path cost $g(n)$ keeps the algorithm from going too far from the start node, while the heuristic $h(n)$ moves the path towards the end goal (Akdogan, A., 2021). A* can then find the most efficient route in a varying environment, such as the dynamic layout of a restaurant with tables and moving customers.

Dijkstra's algorithm

In our restaurant situation, the use of this algorithm ensures that the robot takes the shortest possible path. It's initialized with a starting node and assumes infinite cost for all other nodes that are not known yet. This algorithm iteratively processes all of the unvisited neighboring nodes, selects the neighboring node with the lowest path cost, and updates their path costs with adding the travel cost from the current to the next node. It then moves to the node with the minimal cost. At the end, it has visited all possible nodes and updated all possible nodes, through which it can calculate the shortest path from the starting node to any given node on the graph (Özkan, M., et al, 2022). It focused on the path cost $g(n)$ and can be understood as an example of A* algorithm, where the heuristic $h(n)$ is equal zero:

$f(n) = g(n)$

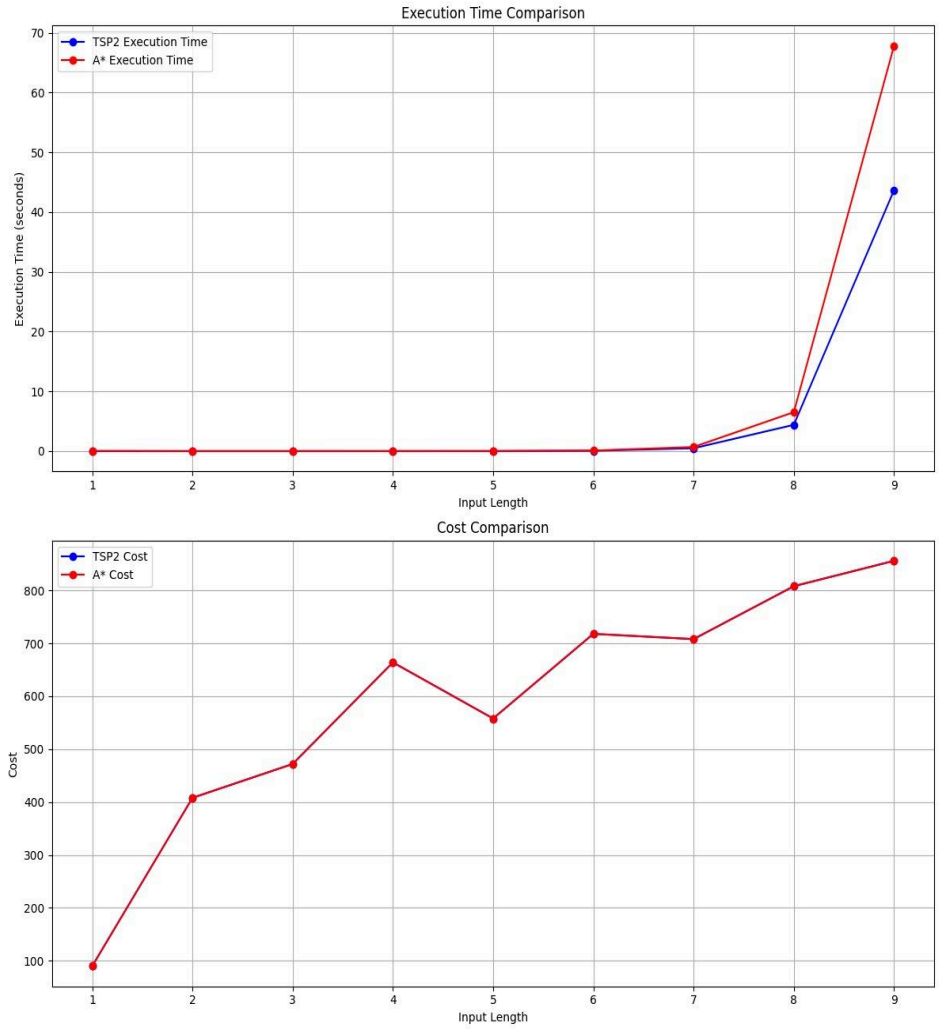
It is less efficient then A* because it employs brute force by calculating the costs for all possible nodes. We hypothesize that A* will generally perform better in real-world applications due to its use of heuristics, which effectively guide the search, while Dijkstra's algorithm, lacking such guidance, explores all possible paths from the start node. This makes Dijkstra's algorithm more reliable for environments where paths have uniform costs and no heuristic estimation is possible or needed.

5. Draw conclusions based on the results of the research, and discuss the implications of the findings.

Performance Analysis

Subsequently we assess the performance of each algorithm in the simulation. By comparatively evaluating the results of these algorithms, we aim to determine which is more efficient within this context. We developed 2 additional layouts that have increased complexity with a progressively higher number of nodes and edges. We test each algorithm's performance based on measuring execution time and cost across the three progressively complex layouts.

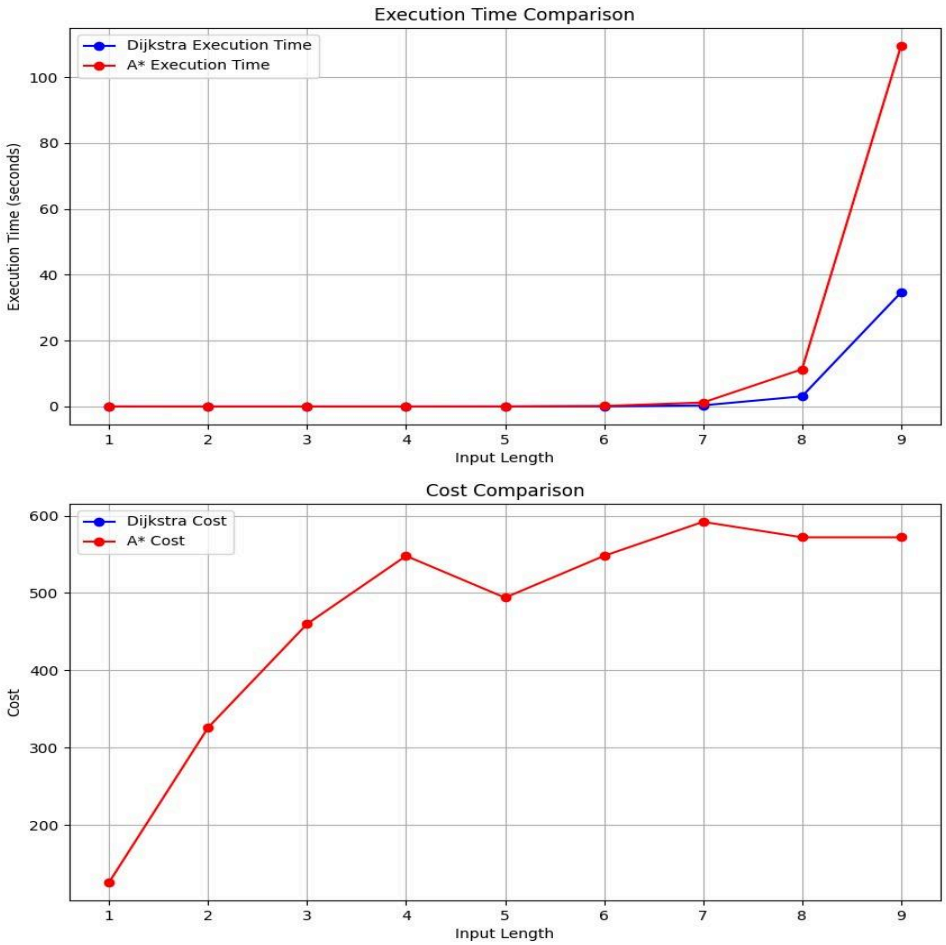
Original Layout



2.1

The above graphs compare the TSP2 (Dijkstra) and A* algorithms in terms of execution time and cost for sequences of varying lengths. Both algorithms show low and stable execution times for input lengths 1 to 7, but experience a significant increase for input lengths 8 and 9, with A* reaching around 70 seconds and TSP2 (Dijkstra) a bit over 40 seconds at the longest sequence. In terms of cost, the A* algorithm's costs increase with longer input lengths, and the TSP2 (Dijkstra) cost line overlaps with the A* cost line, indicating similar cost performance, with costs ranging from approximately 100 to over 700.

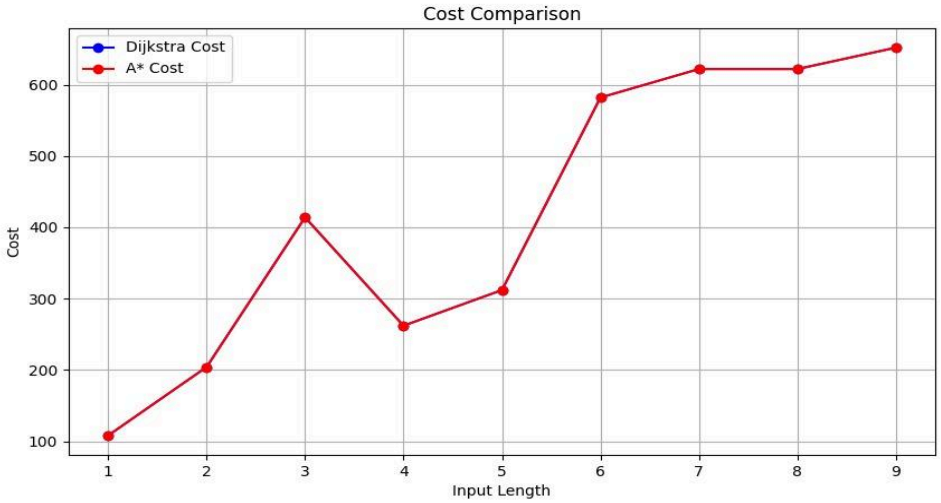
Layout 2



2.2

In Layout 2, both the Dijkstra and A* algorithms show stable and low execution times for inputs ranging from lengths 1 to 6, after which there is a notable increase in execution time at length 7, with A* experiencing a particularly sharp rise by length 9. Regarding costs, Dijkstra shows a consistent upward trend across all input lengths, while A* showcases some fluctuations, which is mostly noticeable when there is an increase at length 6 before stabilizing. This behavior suggests that while both algorithms maintain efficiency in simpler scenarios, A* faces increased computational demands as complexity increases.

Layout 3



2.3

Both algorithms execute very slowly for Layout 3 after input length 6, increasing somewhat at length 7, and peaking for A* at length 9. Up to length 6, the cost trends show comparable patterns; however, starting at length 7, A* has much higher computation time, which keeps rising, indicating higher execution costs in complex situations. This indicates that performance will change as complexity rises, with A* becoming more sensitive to input complexity and influencing execution time and related expenses.

Results and Discussion

General findings for more complex layouts (Layout 2 and 3) show that, whereas Dijkstra and A* perform similarly well in lower complexity scenarios, their performance diverges dramatically in higher complexity scenarios. As complexity increases, Dijkstra's performs much more quickly than A*.

This behavior could potentially be attributed to the heuristic not being well aligned with the more complicated layouts, leading to A* losing its 'guided search' advantage compared to Dijkstras while still maintaining the computational overhead from each call to the heuristic function. The aforementioned suggests that while A* in theory is more efficient in more complex graphs, the efficiency really depends on how well suited the heuristic is.
Additionally, differences in implementation structure and design of the functions could also be contributing to the execution time disparities. Since the algorithms have to iteratively run for all permutations of each sequence, when the sequence gets larger in size, the permutations also increase. Though this is the case for both algorithms, the structural differences in implementation are significant in execution time.

6. Include a discussion of any limitations of the research and suggestions for future research.

Limitations

The subject and methodology of the project we decided on, has significant logistical limitations, which resulted in designing a digital robot, as opposed to a real physical robot. As previously mentioned in the earlier midterm report, our initial idea for the project was to build a maze-solving robot using Arduino and LabView. We then began researching aspects of the robot such as sensors, wheels and other hardware components impacting the performance of the robot in real life applications of maze solving. After identifying all the required parts, we realized that the project would not be feasible without a dedicated budget, and so we decided to instead design and implement a digital simulation using WeBots. While digital simulations like Webots provide a controlled environment to effectively test theoretical models and algorithms, they lack the complexity and unpredictability of real-world environments. In Webots, we also make use of GPS and compass sensors, the effectiveness of which cannot be tested fully in such simulations. The absence of physical interactions with real obstacles limits the ability to accurately measure sensor effectiveness and robot performance. Therefore, the results from our digital simulation might not be the same as from a physical robot navigating an actual maze.

Future research should primarily focus on implementing a physical robot. By building a physical model, we could assess how well the sensors perform under various conditions such as lighting variations, surface textures, or unexpected obstacles. This approach would provide more insights into the practical limitations of robotic navigation systems. Further studies should also explore the integration of more advanced maze-solving algorithms, including those using machine learning techniques. Alamri et al. (2021) proposed various maze-solving algorithms, however, they state that there is yet no perfect maze-solving algorithm for autonomous movement.

Machine learning could enable the robot to learn from and adapt to its environment dynamically, improving its performance. Testing more maze-solving algorithms would provide a broader understanding of which methods are most effective under specific conditions and why. Additionally, generating a more complex maze than our current restaurant floor plan could also be beneficial for testing the algorithms and the robot's ability more effectively.

(2700 words)

References:

1. Akdogan, A. (2021). *Understanding A* path algorithms and implementation with python*. Medium. https://towardsdatascience.com/understanding-a-path-algorithms-and-implementation-with-python-4d8458d6ccc7

2. Alamri, H., Alamri, S. A., Alshehri, S., Alshehri, W., & Alhmiedat, T. (2021). Autonomous Maze Solving Robotics: Algorithms and systems. 10.18178/ijmerr.10.12.668-675

3. Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics*. **4** (2): 100–107.

4. Kulaç, N., & Engin, M. (2023). *Developing a machine learning algorithm for Service Robots in industrial applications*. MDPI. 10.3390/machines11040421

5. Özkan, M., Tabakoğlu, A., Uygun, E. G. & Anbaroğlu, Berk (2022). A Computationally Reproducible Approach to Dijkstra's Shortest Path Algorithm. Advanced Geomatics, 2(1), 01-06

6. Shrinivaas, B. (2023). *Dijkstra - the person & algorithm*. Medium. https://medium.com/@balajeraam/dijkstra-the-person-algorithm-5016ebe9468#62c9