

Path Generation for Robot Navigation using Weighted Shortest Path Algorithm from Source to Target using Artificial Intelligence Concepts

Dr. G.V. Jayaramaiah¹ and Dr. T.C. Manjunath²

¹Professor, Department of Electrical & Electronics Engineering, Dr. Ambedkar Institute of Technology, Bangalore-56, Karnataka

²Professor, Dean (R & D), Department of Computer Science & Engineering, IoT, Cyber Security & Blockchain Technology, Rajarajeswari College of Engineering, Bangalore, Karnataka
Email: tcmanju@iitbombay.org

Abstract—This study presents an improved robot path planning method using Dijkstra's algorithm with path weighting. Integrating weights enhances navigation efficiency from source to goal in various environments, optimizing trajectory decisions for improved performance in complex scenarios.

Index Terms— SCARA, Robot, Simulation, Algorithm.

I. INTRODUCTION

In recent years, robot navigation has become one of the most fascinating and fast-evolving areas in robotics and automation. Whether it's a warehouse robot moving goods, an autonomous vehicle driving through traffic, or a drone mapping unknown terrain, the ability of robots to plan and follow optimal paths from one point to another is at the heart of their performance. Navigation isn't just about moving—it's about doing so intelligently, safely, and efficiently, even in unpredictable environments. One of the key challenges in robot navigation lies in determining the best path between a start point (source) and a destination (target). It's not always a straightforward journey. The robot must often navigate through obstacles, tight spaces, or uneven terrain. That's where path planning comes in—a process that helps the robot figure out *how* to move, *where* to move, and *when* to adjust its route [1].

To make this possible, mathematical models called graphs are used to represent the environment. Each possible position the robot can occupy becomes a *node*, and the connections between those positions are known as *edges*. The weight assigned to each edge typically represents the cost or difficulty of moving between those two points—this could mean distance, time, energy consumption, or even risk level. Now, simply finding any path is not enough. Robots must find the *best* one. That's where weighted shortest path algorithms come into play. These algorithms, like Dijkstra's or A*, help robots evaluate and compare different paths based on edge weights, ultimately selecting the one that minimizes cost and maximizes efficiency. In dynamic and real-time applications, this becomes critical [2].

What makes this process even more powerful today is the integration of Artificial Intelligence (AI). AI introduces adaptability, learning, and decision-making into the system. Instead of relying solely on pre-coded

logic, robots can now analyze real-world data, learn from previous navigation experiences, and make intelligent choices in unfamiliar situations. With AI-based models, robots can predict changes in the environment, adapt their paths in real-time, and even collaborate with other agents in multi-robot systems. Techniques such as machine learning, fuzzy logic, neural networks, and reinforcement learning are actively used to enhance path-planning algorithms, making them more context-aware and resilient [3].

In this approach, weighted shortest path algorithms provide a strong foundation, while AI acts as the brain that continuously refines and improves the path generation strategy. Together, they create a navigation system that isn't just algorithmically efficient but also behaviorally intelligent—capable of learning and reacting just like a human would. The combination of these technologies has enabled groundbreaking applications—from automated delivery systems in urban environments to search-and-rescue robots operating in disaster zones. The paths generated are no longer static; they're dynamic, responsive, and capable of self-optimization. Robots can now go beyond “A to B”—they can choose *how best* to get there [4].

As robotics continues to intersect with AI, the design and implementation of intelligent path generation models will become more sophisticated. This also opens up exciting research directions: How can AI help robots handle completely unstructured environments? How do we balance speed, safety, and learning in path planning? This paper explores these questions and more, focusing on how weighted shortest path algorithms, when enhanced with AI-driven decision-making, can revolutionize robotic navigation. It aims to bridge theory with practical implementation, bringing together the best of algorithmic planning and artificial intelligence for smarter, safer, and more adaptive robotic systems [5].

II. LITERATURE SURVEY

PAPER 1: Route Planning for an Autonomous Robotic Vehicle Employing a Weight-Controlled Particle Swarm-Optimized Dijkstra Algorithm| by Subaselvi Sundarraj *et.al*. A reliable approach to the route planning method is weight-controlled particle swarm-optimized Dijkstra algorithm (WCPSODA). The Dijkstra algorithm seeks the route with the minimum distance between two points by going in the opposite direction of the length of the paths. In mapping with a single source, it addresses the minimum path issue by determining the shortest route between the origin and destination nodes. In this study, the conventional Dijkstra algorithm is enhanced to address the route planning issue in the square warehouse framework setting. In the square surroundings area, there are several smallest paths of equal length. The Dijkstra method only keeps one intermediary node, making it possible to search for just one fastest route. By setting aside any nodes that are equally far from the origin node as the intermediary nodes, we enhance the Dijkstra method [6].

PAPER 2: Path Planning for Mobile Robot Navigation in Unknown Indoor Environments Using Hybrid PSOFS Algorithm| Mohd Nadhir *et.al*. This paper discusses A* algorithm which is one of the variants of Dijkstra's algorithm and commonly used to find the shortest path in a weighted directed graph. Unlike Dijkstra's algorithm which only can explore one possible path at a time, the algorithm can explore multiple possible paths at a time from the origin vertex, until one of the paths ends at the destination vertex. The efficiency of the algorithm in pathfinding is higher than Dijkstra's algorithm. The algorithm is mainly used for global search in static environments. This paper is aimed to solve the problem of planning the optimum path for a mobile robot to navigate in unknown indoor environments [7].

PAPER 3: Dynamic Path Planning Algorithms with Load Balancing Based on Data Prediction for Smart Transportation Systems by Ning Sun *et.al*. In this paper, a traffic path planning algorithm based on data prediction named TPPDP is proposed. A prediction model is designed to predict the traffic state of the road segment in the future based on historical traffic data and current traffic information, and based on the predicted result, which is the average travel speed of road segments in the next time slot, TPPDP algorithm is proposed to plan a path with the shortest travel time. It discusses the Dijkstra Algorithm and its improved versions in detail.

PAPER 4: Improved Dijkstra Algorithm for Mobile Robot Path Planning and Obstacle Avoidance by Shaher Alshammrei *et.al*. where, the optimal path planning avoiding obstacles is among the most attractive applications of mobile robots (MRs) in both research and education. In this paper, optimal collision-free algorithm is designed and implemented practically based on an improved Dijkstra algorithm. To achieve this research objectives, first, the MR obstacle-free environment is modeled as a diagraph including nodes, edges and weights. Second, Dijkstra algorithm is used offline to generate the shortest path driving the MR from a starting point to a target point. During its movement, the robot should follow the previously obtained path and stop at each node to test if there is an obstacle between the current node and the immediately following node.

PAPER 5: A Survey of Path Planning Algorithms for Mobile Robot by Karthik Karur *et.al*. Here, the path planning algorithms are used by mobile robots, unmanned aerial vehicles, and autonomous cars in order to

PAPER 6: Dijkstra Algorithm based intelligent path planning with topological map and wireless communication by Lyle Parungao *et.al*. In this paper, an idea of intelligent path planning was introduced. Using information received from a server that is transmitted through a wireless communication, the data is processed to edit a provided topological map, with which a shortest path calculation will be executed. Information about the destination and blocked road parts is sent to a mobile robot using a wireless ad hoc network communication. The mobile robot will process this information in a topological 2D array map and ignore the blocked parts of the road for the shortest path calculation based on Dijkstra's algorithm. Aside from the intelligent path planning, an automated driving algorithm was also implemented using infra-red sensors installed on the mobile robot to navigate the robot to its destination.

A graph is a non-linear data structure consisting of vertices and edges. Vertices are sometimes called nodes, and edges are lines or arcs that connect any two nodes in a graph. More formally, a graph consists of a set of vertices (V) and a set of edges (E). The graph is denoted by $G(V,E)$. Graphical data structures are a powerful tool for representing and analyzing complex relationships between objects or entities. They are particularly useful in areas such as social network analysis, recommender systems and computer networks. In the field of sports informatics, graph data structures can be used to analyze and understand the dynamics of team performance and player interactions on the field. Imagine the game of soccer as a web of connections, where the players are the nodes and their interactions on the field are the edges. This web of connections is exactly what the graph data structure represents, and is the key to understanding team performance and player dynamics in sports.

Vertices: Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labelled or unlabeled.

Representation of Graphs - There are two ways to store a graph: Adjacency Matrix

Adjacency List

Adjacency Matrix

In this method, the graph is stored in the form of the 2D matrix where rows and columns denote vertices. Each entry in the matrix represents the weight of the edge between those vertices as shown in Tables I.

TABLE I: ADJACENT GRAPH LIST TABLE COMPONENTS OF THE PARAMETERS

	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9
V0	0	1	0	0	1	0	0	1	0	0
V1	1	0	1	1	0	1	0	0	0	0
V2	0	1	0	0	0	0	1	0	0	1
V3	0	1	0	0	0	0	0	1	1	0
V4	1	0	0	0	0	1	0	0	0	0
V5	0	1	0	0	1	0	1	0	0	0
V6	0	0	1	0	0	1	0	0	0	1
V7	1	0	0	1	0	0	0	0	1	0
V8	0	0	0	1	0	0	0	1	0	1
V9	0	0	1	0	0	0	1	0	1	0

VI. ADJACENCY LIST – REPRESENTATION OF THE PATH OF THE ROBO AS A GRAPH

This graph is represented as a collection of linked lists. There is an array of pointer which point to the edges connected to that vertex. The journey of a robot can be represented by a diagram, defining a set of nodes and edges that model the relationship between different places of the robot and the environment. Nodes represent specific locations or waypoints, and edges represent possible transitions or connections between those stations. Graphs can be used to model various data structures, for instance, an Octree. Octrees have applications in things such as 3D computer graphics, spatial indexing, nearest neighbour searches, finite element analysis, and state estimation. In robotics especially, octrees have been leveraged via the creation of the OctoMap Library, which implements a 3D occupancy grid mapping approach. This provides data structures and mapping algorithms that not only assist in mobile robot navigation and mapping, but also helps in path planning for manipulators in cluttered environments.

Define nodes: Identify key locations or waypoints around the robot and environment. Each of these locations becomes a node in the graph. Nodes can represent locations on a grid, coordinates in continuous space, or specific landmarks in the environment.

Create edges: Create connections between nodes by defining edges. An edge between two nodes indicates that the robot can move from one place to another. Edges can be associated with weights or costs that represent distance, time or other factors involved in moving from one node to another.

Graph Presentation: Choose an appropriate graphical representation, such as an adjacency matrix or an adjacency list, to capture the relationships between nodes and edges. The representation should allow efficient passage and path planning.

VII. ALGORITHMS FOR FINDING SHORTEST PATH IN A GRAPH – BREADTH FIRST SEARCH (BFS)

Breadth-First Search (BFS) is one of those fundamental algorithms that pops up often when working with trees or graphs. It's widely used for exploring or searching through these structures. The idea behind BFS is fairly intuitive: you start at the source node and explore all its immediate neighbors first, then move outward layer by layer. It's like exploring a network of rooms in a building—you check all the rooms on the current floor before going up to the next one. Now, BFS is especially suited for scenarios where all edges in the graph are either equally weighted or completely unweighted. In such cases, BFS guarantees the shortest path from the source to any other node in terms of the number of edges, which makes it extremely useful for simple pathfinding problems.

But things get a bit more interesting when we deal with edges that have binary weights—meaning, each edge has either a weight of 0 or 1. This is where the standard BFS falls short, and a variant called 0-1 BFS becomes valuable. The 0-1 BFS is a clever twist on the original algorithm that allows us to handle these kinds of graphs

much more efficiently. So, what makes 0-1 BFS special? Instead of using a regular queue like in traditional BFS, 0-1 BFS uses something called a deque (short for "double-ended queue"). This allows elements to be added or removed from both the front and the back, which is key to how the algorithm manages distances. Here's the trick: when the algorithm encounters an edge with a weight of 0, it adds that node to the front of the deque. If it encounters an edge with a weight of 1, it adds the node to the back. This ensures that nodes reached with a lower cumulative weight are processed before those with a higher one—almost like how Dijkstra's algorithm works, but faster in this specific case.

Because of this simple yet effective mechanism, 0-1 BFS can compute the shortest path distances in graphs with binary weights in linear time—that's $O(V+E)$, where V is the number of vertices and E is the number of edges. That's significantly more efficient than traditional shortest path algorithms for this specific use case. Of course, this method only works when the graph's edges are strictly 0 or 1. It's not a drop-in replacement for Dijkstra's algorithm, which handles more general cases with arbitrary positive weights. But when the conditions are right, 0-1 BFS is a powerful and lightweight tool.

Another advantage is that 0-1 BFS naturally preserves the predecessor of each vertex, which is useful when reconstructing the shortest path after traversal. This makes it not just fast, but practical when you're working on real-world navigation or network routing problems. In summary, if you're working with graphs where edges are either free (weight 0) or cost a single step (weight 1), 0-1 BFS gives you a smart and efficient way to compute shortest paths. It keeps the simplicity of BFS but tailors it for binary-weighted graphs using a deque for optimal performance.

IX. DIJKSTRA'S ALGORITHM

Dijkstra's Algorithm is one of the most well-known and widely used techniques in computer science for solving shortest path problems in a graph. Specifically, it helps you find the shortest path from a single source node to all other nodes in the graph. So, if you're trying to figure out the most efficient route from one city to every other city on a map, Dijkstra's is a go-to method. This algorithm falls under the category of Greedy Algorithms—meaning it makes a series of choices, always selecting what seems best at the moment, and it turns out that for graphs with only positive edge weights, this approach actually works perfectly. Traditionally, if you use an adjacency matrix to represent the graph, the algorithm runs with a time complexity of $O(V^2)$, where V is the number of vertices. That's fine for small graphs, but can get quite slow as the graph grows. However, if you switch to using an adjacency list along with a priority queue (like a min-heap), you can speed things up significantly. In that case, the time complexity becomes $O((V + E) \log V)$ —a big improvement, especially when you're dealing with large and sparse graphs. One important thing to keep in mind, though: Dijkstra's Algorithm doesn't work with negative edge weights. That's its main limitation. If a graph has any edges with negative costs, Dijkstra's can give incorrect results because it assumes that once it's found the shortest path to a node, there's no need to revisit it—which isn't always true when negative weights are involved. All in all, Dijkstra's is an efficient and practical choice for many real-world pathfinding problems—as long as you're dealing with positive weights only.

X. FLOYD WARSHALL

The Floyd-Warshall algorithm, named after its creators Robert Floyd and Stephen Warshall, is a foundational method in both computer science and graph theory. It's best known as a dynamic programming algorithm used to find the shortest paths between all pairs of vertices in a graph. One of the standout features of this algorithm is that it can handle negative edge weights, which many other algorithms like Dijkstra's cannot. Floyd-Warshall works by gradually improving the path estimates between every possible pair of nodes. It does this by considering each node as a potential intermediate stop between two others, and updates the shortest known path if a better one is found through that node. It's particularly efficient when dealing with dense graphs, where most of the possible connections between nodes actually exist.

However, this exhaustive approach can also be a drawback. Since the algorithm looks at all combinations of nodes, it becomes computationally heavy, especially for larger graphs. The time complexity sits at $O(V^3)$, which may be fine for small graphs or when we truly need all-pairs shortest paths, but not so great when speed is a concern. In the context of robot navigation, where path weights can vary significantly due to environmental conditions or energy consumption, simpler algorithms like BFS or 0-1 BFS often fall short. These methods assume uniform or binary weights, which rarely reflect real-world robot path planning needs. That's why Floyd-

Warshall might seem like a good fit at first—it's thorough and handles complex weights—but its performance can become a bottleneck.

XI. CONCLUSIONS

To sum up, our case study shows that the proposed Dijkstra-based algorithm works well for robot path planning and obstacle avoidance. It can quickly recalculate paths in real time, allowing the robot to adapt smoothly even when new obstacles appear. Despite its time complexity being proportional to the square of the number of nodes, the algorithm performed efficiently in indoor settings where the environment is relatively small. Using an Arduino board proved to be a practical choice, offering enough processing power and memory to run the algorithm effectively.

REFERENCES

- [1] Syed Abdullah Fadzli, Sani Iyal Abdulkadir, Mokhairi Makhtar, Azrul Amri Jamal, "Robotic Indoor Path Planning using Dijkstra's Algorithm with Multi-Layer Dictionaries", *Conf. Paper*, 2015.
- [2] Huijuan Wang, Quanbo Yuan, Yuan Yu, "Application of Dijkstra algorithm in robot path- planning", *Journal Paper*, 2011.
- [3] Masato Noto, Hiroaki IS, "A Method for Shortest Path Search by Extended Dijkstra Algorithm", *Journal Paper*, 2002.
- [4] Mr. S. Julius Fusic, Mr. P. Ramkumar, Dr. K. Hariharan, "Path planning of robot using modified dijkstra Algorithm", *Journal Paper*, 2018.
- [5] Saif Allah M. Abgenah, Azrul Amri Jamal, Syed Abdullah Fadzli, "Multi-Robot Path Planning using Dijkstra's Algorithm with Multi-layer Dictionaries", *Journal Paper*, 2021.
- [6] Subaselvi Sundararaji, R. Vijaya Kumar Reddy, Mahesh Babu Basam, Gururaj Harinahalli Lokesh, Francesco Flammini, Rajesh Natarajan, "Route Planning for an Autonomous Robotic Vehicle Employing a Weight-Controlled Particle Swarm-Optimized Dijkstra Algorithm", *Journal Paper*, 2023
- [7] Shaher Alshammrei, Sahbi Boubaker and Lioua Kolsi, "Improved Dijkstra Algorithm for Mobile Robot Path Planning and Obstacle Avoidance", *Journal Paper*, 2022.