Received: 22 March 2023 | *Revised:* 29 April 2023 | *Accepted:* 1 May 2023
DOI: 10.37917/ijeee.19.2.11 *Early View* | *December 2023*

Iraqi Journal for Electrical and Electronic Engineering
Original Article

Open Access
IJEEE University of Basrah College of Engineering

An Efficient Path Planning in Uncertainty Environments using Dynamic Grid-Based and Potential Field Methods

Suhaib Al-Ansarry *, Salah Al-Darraji, Dhafer G. Honi
Department of Computer Science, College of Education for Pure Sciences, University of Basrah, Basrah, Iraq

Correspondance
* Suhaib Al-Ansarry
Department of Computer Science,
College of Education for Pure Sciences,
University of Basrah, Basrah, Iraq.
Email: suhaib.alansarry@uobasrah.edu.iq

Abstract
Path planning is an essential concern in robotic systems, and it refers to the process of determining a safe and optimal path starting from the source state to the goal one within dynamic environments. We proposed an improved path planning method in this article, which merges the Dijkstra algorithm for path planning with Potential Field (PF) collision avoidance. In real-time, the method attempts to produce multiple paths as well as determine the suitable path that's both short and reliable (safe). The Dijkstra method is employed to produce multiple paths, whereas the Potential Field is utilized to assess the safety of each route and choose the best one. The proposed method creates links between the routes, enabling the robot to swap between them if it discovers a dynamic obstacle on its current route. Relating to path length and safety, the simulation results illustrate that Dynamic Dijkstra-Potential Field (Dynamic D-PF) achieves better performance than the Dijkstra and Potential Field each separately, and going to make it a promising solution for the application of robotic automation within dynamic environments.

Keywords
Robotic, Path Planning, Dijkstra, Potential Field, Static Obstacle, Dynamic Environment.

I. Introduction

Designing the path for an autonomous robot is a crucial task, especially when operating in dynamic environments where obstacles can appear or disappear unexpectedly in real-time. The objective of planning the robot's path is to identify a secure and efficient route from the initial location to the desired endpoint, considering the existence of obstacles. [1] In dynamic environments, the primary challenge is to devise a path that is both safe and optimal while allowing the robot to adapt to any changes in the environment and modify its course accordingly. Finding a path that is safe from potential threats and efficient to meet the objective requires an extensive understanding of the environment and the robot's capabilities. This involves analyzing data from various sources such as sensors, cameras, and LIDAR (Light Detection and Ranging) to detect obstacles and other hazards, and then employing a suitable algorithm that considers the robot's motion constraints, obstacle avoidance, and optimality criteria to compute an optimal and safe path.

Several planner algorithms have been discussed in the previous literatures. A* is a heuristic search algorithm that uses a cost function to evaluate the distance from the starting point to the goal, considering the appearance of obstacles. [2] Dijkstra is a graph-based algorithm that finds the shortest path between two nodes [3], while the Rapidly Exploring Random Tree (RRT) is a sampling-based algorithm that generates a tree structure and finds a path by connecting the starting and goal points. [4], [5] Artificial Potential Field (APF) is a technique that uses a potential field to guide the robot away from obstacles based on the gradient of the field. [6] Despite the success of these traditional path planning algorithms, they have limitations in dynamic environments. [7] For example, A* and Dijkstra are computationally intensive and may not

be suitable for real-time response, while RRT is not well-suited for environments with dynamic obstacles. Potential field obstacle avoidance is a promising solution for dynamic environments, but it can result in sub-optimal paths that are not safe or efficient and may also fall in the local minimum regions.

L. Li-sang et al. describe a smart obstacle avoidance car for autonomous driving, using Simultaneous Localization and Mapping (SLAM) technology to generate a map of an indoor unknown environment. [7] Other researchers focus on finding the shortest mobile route in a virtual environment using the Modified Dijkstra's algorithm. [8] D. S. Nair and P. Supriya present a new approach for planning besides collision avoidance within the system of robot navigation using Modified Temporal Difference Learning (MTDL). [9] On the other side, another method that presents a solution for the problem of constrained coverage path planning for Unmanned Aerial Vehicles (UAVs) is proposed. [10] X. Li, proposes a relaxed Dijkstra algorithm to plan the path of a robot within a real time large-scale and obstacle-intensive environment. [11] Particle Swarm Optimization (PSO) algorithm is applied to improve the initial path of Dijkstra to get the final shortest path. [12] Also, an improvement to the Dijkstra algorithm for optimizing the shortest path for the direct navigation of ships has been proposed. [13] In 2022, a new method proposes an energy-efficient local path planning algorithm for Autonomous Ground Vehicles (AGVs) by merging the Potential Field algorithm with future movement prediction. [14] The algorithm of Dijkstra to find the optimal path in real-time for a single unmanned surface vehicle in Portsmouth Harbour is introduced. [15] B. Kasmi and A. Hassam (2021) proposed a two-part path planning method for mobile robot navigation, using cell decomposition for offline computation and a potential field method for online computation. [16] Finally, an analytical method for path planning for a UAV to attack multiple moving targets in a dynamic environment is presented. [17] W. Yang et al. operate to address the problems of "local minimum" and "unreachable target". The improved method modifies the direction and influence range of the gravitational field, increases the virtual target and evaluation function, and increases gravity to solve these problems. [18] These studies offer valuable insights into various path planning and obstacle avoidance techniques for robotic systems, which can be useful in a range of applications. [19], [20], [21]

In this research, we present an improved method for path planning that combines the Dijkstra path planning algorithm with the Potential Field obstacle avoidance method to address the aforementioned challenges (dealing with the dynamic obstacles and overtaking the regions of local minima). The Dijkstra algorithm is used to generate multiple paths from the source point to the target one, and the potential field algorithm is used to filter the only desirable paths (paths near to the shortest one and have a responsible cost) to be assigned as alternative paths in case of the basic path (shortest one) intersect by any dynamic obstacle, then evaluated the safety (obstacle-free) of each path that generated by Dijkstra and select the next optimal one. The proposed method generates connections between the alternative paths, allowing the robot to switch between various paths if it faces a dynamic obstacle on its current route in real-time. This allows the robot to continue toward its target while avoiding obstacles in a safe and efficient manner. The proposed method also handled the issue of trapping within the region of local minima which PF suffered from, making the path suitable for use in a wide range of mobile robot applications.

The research is distributed as follows: Section II presents the methodology for the proposed method and the problem formulation. In section III, experiments are conducted and the results are illustrated to validate the algorithm. Finally, the conclusion is made in Section IV.

II. Proposed Method

The method suggested involves utilizing a combination of the Dijkstra path planning algorithm and potential field obstacle avoidance to determine the most efficient path in constantly changing environments. It consists of two distinct phases: Path Generation and Path Selection.

A. Path Generation

The algorithm of Dijkstra is considered a common graph traversal method that is used to locate the path with the shortest length between two vertices in a graph with non-negative edge weights. It is known for its efficiency and is commonly used in various applications, such as finding the optimal route between two nodes on a map or finding the shortest path in a computer network.

The algorithm operates as shown in Figure (1) by iteratively exploring nodes within a graph, starting from the initial node and visiting its neighbors. The distance between the source node and each neighboring node is calculated and then selects the neighbor with the shortest distance. This strategy is continued till every single node has been examined, and the path with the lowest distance has been determined between the source node and all other nodes in a graph.

In the path generation phase, the Dijkstra algorithm is used to generate multiple paths as shown in Figure (2) between the start and goal points, considering the presence of obstacles in the environment. The algorithm creates a graph structure, where nodes represent the possible locations of the robot, and edges represent the connections between the nodes. The algorithm then finds the shortest path from the source point

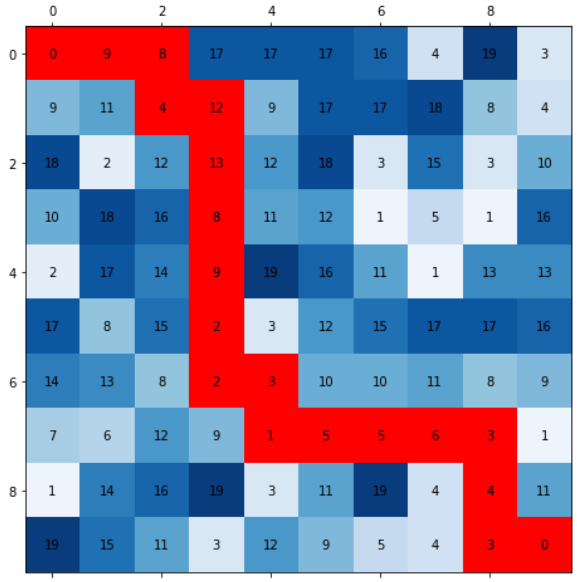

Fig. 1. Dijkstra Map Representation

to the goal by searching the graph structure for the minimum distance between nodes.

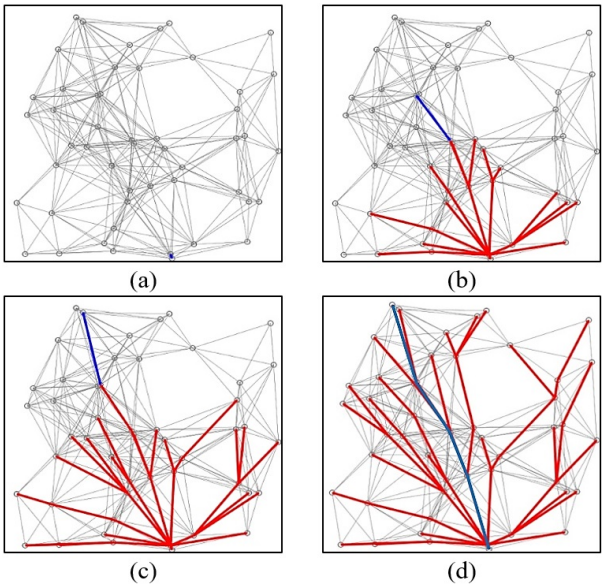

Fig. 2. Path Generation

The goal here is to generate candidate paths from the current robot position to the position of the goal. This is accomplished by using Dijkstra's shortest path algorithm, which finds the optimal path for each two points in a graph. By generating and storing the multiple candidate paths, the proposed method provides the robot with multiple options for reaching the goal position and increases the robustness of the path planning process. Algorithm (1) describes these steps clearly.

Algorithm-1: Path Generation

```
Initialize : graph, start, end
dist = float("inf")
dist[start] = 0
previous = {node: None for node in graph}
unvisited = graph.keys()
while unvisited:
  current = min(unvisited, key=lambda x: dist[x])
  if current == end:
    break
  unvisited.remove(current)
  for neighbor, weight in graph[current]. items():
    if neighbor not in unvisited:
      continue
    new_dist = dist[current] + weight
    if new_dist < dist[neighbor]:
      dist[neighbor] = new_dist
      previous[neighbor] = current
return previous, dist[end]
```

- The ShortestPath function takes three inputs: graph, start, and end. Graph is a dictionary of dictionaries representing the graph structure, where the keys are the nodes, and the values are dictionaries of neighbors and weights. Start and end are the starting and ending nodes, respectively.
- The distances dictionary is initialized with the value float("inf") for all nodes, representing that the distance is unknown. Set the distance equal to 0 between the start node to itself.
- The previous dictionary is initialized with None for all nodes, representing that there is no previous node in the path yet. The unvisited list is initialized with all the nodes in the graph.
- The while loop continues until all nodes have been visited or the end node has been found.
- In each iteration, the node with the minimum distance is selected as a current node. If it is the end node, the loop is terminated. The current node is removed from the unvisited list.
- For each neighbor of the current node, the distance from the start node to the neighbor is calculated as the sum of the distance from the start node to the current node and the edge weight connecting the current one and the

neighbor. If this new distance ¡ current distance to the neighbor, the distances and previous dictionaries are updated accordingly.

- The previous dictionary and the distance starting from the source node to the end one is returned as the result of the algorithm.

B. Path Selection

The Potential Fields is a path planning algorithm that calculates a path between the start and goal positions while avoiding collisions that may occur in the environment. The algorithm starts by initializing the current position to the start point and runs a loop with a maximum amount of iterations. In each iteration, the attractive force vector is calculated depending on the current position and the goal location, while the repulsive force vector is calculated depending on the current location and the obstacles in the configuration. The vector of the total-force is then calculated by adding the attractive and repulsive force vectors and the current position is updated by moving a small step in the direction of the total-force vector. The distance between the current location and goal one is checked, and if it is below a predefined threshold, a path to the goal has been found and the algorithm returns the path. If the maximum number of iterations is reached and no path is found, the algorithm returns ”No path found”. In this phase, the algorithm of the Potential Field (PF) is used to evaluate the safety of each path generated by the Dijkstra algorithm. The algorithm creates potential fields, where the gradient of these fields as shown in Figure (3) guides the robot away from obstacles and toward the goal point. The virtual path represents by black dashed line and the actual path represents by green dashed line. The attractive forces act as blue rows, while repulsive forces as red rows.

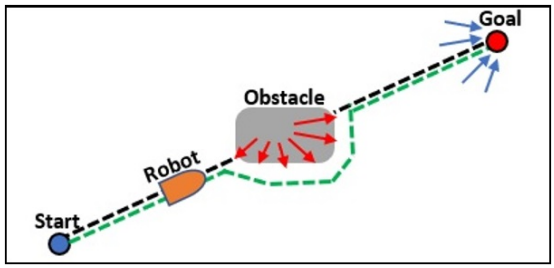

Fig. 3. Potential Field

The safety of each path is evaluated based on the potential field, and the path with the safety highest score is chosen as the optimal path. Additionally, the proposed method generates connections between the paths, allowing the robot to switch to different paths if it encounters a dynamic obstacle in its current position. The connections are generated based on the potential field, allowing the robot to switch toward another path while still avoiding obstacles safely and efficiently. The proposed method operates in real-time and can handle movable obstacles, making it suitable for use in a wide range of applications. The method is designed to find a path that is both short and safe, while also allowing the robot to react the variations in the environment and adjust its path accordingly. Figure (4) demonstrates the representation of the Artificial Potential Fields (APF).

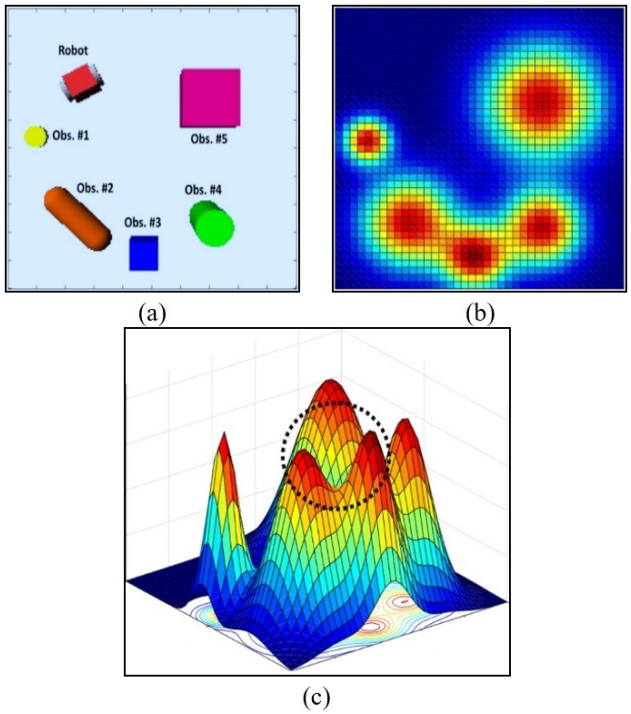

Fig. 4. Potential Fields Representation (a) 3D Environment (b) 2D Field (c) 3D Field

The main mathematical equations used in the potential field algorithm are as follows:

1. **Attractive Force Vector Calculation:** The attractive force vector is calculated using Equation (1):

$$F\_attr = k\_attr * (goal - currentPos) \quad (1)$$

Where: F_attr: Attractive force vector, k_attr: Attractive force constant (positive scalar value), goal: Goal position, currentPos: Current position.

2. **Repulsive Force Vector Calculation:** The repulsive force vector is calculated using Equation (2) and (3) for each obstacle in the environment:

$$F\_rep = k\_rep * (1/distance - 1/d0) * direction \quad (2)$$

$$direction = (obs - curPos)/||obs - curPos|| \quad (3)$$

Where: F_rep: Repulsive force vector, k_rep: Repulsive force constant (positive scalar value), obs: Obstacle, position curtPos: Current position, d0: Desired minimum distance between the obstacle and the robot.

The goal here is to select the optimal path from the multiple candidate paths generated in the Path Generation phase. This is accomplished by employing the Potential Field to calculate the safety of each candidate path and combining this information with the length of the path to determine the overall cost of each path. The path with the lowest cost is then selected as the optimal path, providing the robot with the best trade-off between safety and efficiency in reaching the target position. Algorithm (2) shows these steps clearly.

Algorithm-2: Path Selection

```
Initialize = robot_pos, target, obstacle
attr_force = k_attr * (target - robot_position)
rep_force = [0, 0]
for obstacle in obstacles:
    distance = dist(robot_position, obstacle)
    direction = normalize(robot_position - obstacle)
    rep_force += k_rep * (1/distance - d0) * direction
total_force = attr_force + rep_force
return normalize(total_force)
```

- This algorithm, takes three inputs: robot_pos, target, and obs. robot_pos represents the current position of the robot, the target represents the desired target position, and obstacles are a list of obstacles in the environment.
- The attractive force is calculated as k_attr * (target - robot_pos), where k_attr is a constant that determines the strength of the attractive force. The attractive force acts to pull the robot toward the target position.
- The repulsive force is initialized as [0, 0]. For each obstacle in the obstacles list, the distance between the robot and the obstacle is calculated, and the direction away from the obstacle is calculated as normalized (robot_pos - obs). The contribution of the obstacle to the repulsive force is calculated as k_rep * (1/distance - d0) * direction, where k_rep is a constant that determines the strength of the repulsive force, and d0 is a constant that represents a desired minimum distance between the robot and obstacles. The repulsive force is updated with the contribution of each obstacle.
- The total_force is calculated as the sum of the attractive force and the repulsive force.
- The algorithm returns the normalized total force, representing the direction that the robot should move in to avoid obstacles while still reaching the target position. The normalization step ensures that the magnitude of the force is always 1, which helps in controlling the speed of the robot's motion.

C. Problem Formulation

The problem of the (Dynamic D-PF) method is to find a safe and efficient path for a robot to navigate from a starting position to a target position in a 2D environment that may contain dynamic obstacles. The solution to this problem is formulated mathematically as a combination of path length and obstacle avoidance, and the method returns the optimal path with the minimum cost.

Let's assume the following variables and their definitions:

- s: starting position of the robot.
- t: target position.
- O: a set of obstacles.
- p: a candidate path from s to t.
- l(p): length of the path p.
- f_obs(p): the rep_force exerted by obstacles on path p.
- cost(p): cost of path p, defined as the combination of the path length and the safety of the trajectory.

The mathematical formulation of the (Dynamic D-PF) method can be defined as follows:

1. Generate candidate paths using the Dijkstra algorithm:

$$p\_candidates = Dijkstra(s,t) \quad (4)$$

2. For each candidate path, calculate the safe trajectory using the Potential Field, for p in p_candidates:

$$safe\_trajectory = PotentialField(s,p,O) \quad (5)$$

3. Evaluate the cost of each path: for p in p_candidates:

$$cost(p) = l(p) + f\_obs(p) \quad (6)$$

4. Select the path which has a minimum cost:

$$p\_optimal = argmin\_pcost(p) \quad (7)$$

The method then returns the optimal path, p_optimal, which is the minimum cost path. The cost of each path is calculated as the sum of the length of the path and the repulsive force exerted by obstacles on the path. The length of the path can be calculated using any suitable distance metric, and the repulsive force can be calculated using the method of Potential Field.

D. Dynamic D-PF Method

We present -in this section- the Dynamic Dijkstra-Potential Field method (Dynamic D-PF). The goal here is to implement the selected optimal path as shown in Figure (5) in a real-time potential field environment and respond to any dynamic obstacles that may arise during the execution of the path. This is accomplished by using the connections generated in the Path Generation phase, which allow the robot to switch between candidate paths if it encounters a dynamic obstacle on its current path.

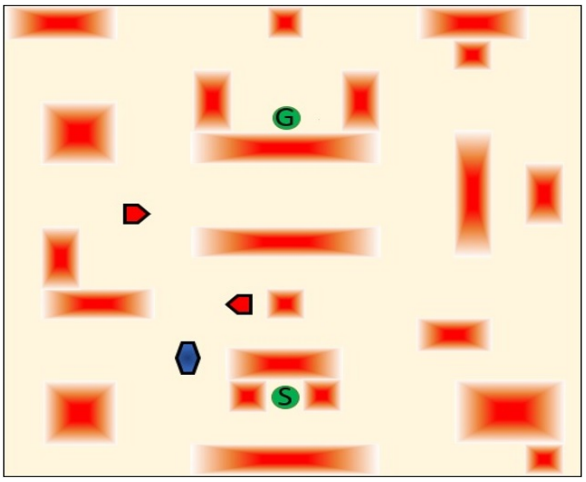

Fig. 5. Potential Fields Heat Map with Dynamic Obstacles

By combining these two algorithms, the proposed method (Dynamic D-PF) as illustrated in Algorithm (3) provides a flexible and efficient way to find an optimal path (short and safe) in real-time within a 2D environment. The combination of the multiple paths generated by the Dijkstra and the Potential Field method provides robustness and adaptability in the presence of dynamic obstacles.

Algorithm (3), takes as input the robot's current position, the target position, and the obstacles in the environment. It first generates multiple paths from the robot's current position to the target one using Dijkstra's algorithm. Then, for each path generated by Dijkstra's algorithm, it applies the Potential Field method to find a safe trajectory along the path, avoiding obstacles along the way. If a safe trajectory is found, the path and its associated safe trajectory are added to the list of paths. The EvaluatePath function calculates the cost of each path and its associated Potential Field trajectory by summing the length of the path and the safety of the safe trajectory along the path. Finally, the path with the lowest cost is chosen as the optimal path and returned as the result. It is worth mention that A* and Dijkstra are algorithms to locate the shortest route between two points within a graph or a network. However, there are some key differences between them.

Algorithm-3: Dynamic D-PF

```
Initialize = robot_pos, target, obstacle
paths = [ ]
path_candidates = Dijkstra(robot_pos, target)
for path in path_candidates:
    safe_trajectory = PotentialField(robot_pos, path, obs)
    if safe_trajectory:
      paths.append((path, safe_trajectory))
optimal_path = None
optimal_cost = infinity
for path, safe_trajectory in paths:
    path_cost = EvaluatePath(path, safe_trajectory)
    if path_cost < optimal_cost:
      optimal_path = path
      optimal_cost = path_cost
return optimal_path

function EvaluatePath(path, safe_trajectory):
   path_length = Length(path)
   safety = Safety(safe_trajectory)
   return path_length + safety
```

- **Heuristic Function** One of the main differences between A* and Dijkstra is that A* utilizes a heuristic function to guide the search toward the goal, while Dijkstra does not. The heuristic function estimates the distance from a node to the goal, which helps A* to prioritize nodes that are closer to the goal, potentially leading to faster search times.
- **Optimality** Dijkstra is guaranteed to reach the goal with the minimum distance between two points, whereas A* is only guaranteed to find an optimal path if the heuristic function used is admissible, meaning it never overestimates the actual distance to the goal.
- **Memory Usage** A* typically uses more memory than Dijkstra, as it keeps track of more information about each node in order to calculate the heuristic function. However, this additional memory can often lead to faster search times.
- **Time Complexity** In terms of time complexity, the algorithm of Dijkstra has the worst time complexity:

$O(|E|+|V|log|V|)$, where E is the set of edges and V is the set of vertices. On the other hand, A* has a worst-case time complexity of O($b\hat{d}$), where b is the branching factor of the graph and d is the depth of the optimal solution. However, in practice, A* often performs better than Dijkstra due to its use of the heuristic function.

As a conclusion, the proposed dynamic path planning method (Dynamic D-PF) can be illustrated through the blocks diagram. Figure (6), shows the steps clearly.

III. Experimental Results

The proposed method was evaluated through a series of experiments in a simulated 2D environment. The performance of the (Dynamic D-PF) was evaluated in terms of path length, safety, and execution time. The path length was measured as the total distance traveled by the robot starting from the initial location to the target one. The safety of the path was evaluated by the number of times the robot encountered a dynamic obstacle and had to switch to a different path. The execution time was measured as the time it took for the robot to complete the path from start to finish. A variety of scenarios were tested, with different numbers and types of static and dynamic obstacles, and different levels of complexity in the environment. The start point depicts as an orange circle and the goal point as a green circle. The shortest global path (Dijkstra modified-path) is represented by the red solid line. The dynamic real-time path (generated by PF) is shown by the solid green line and the final resulting path (generated by Dynamic D-PF) is represented by a black dotted line, which is calculated based on the proposed method. The grey dotted line represents the desired path-net, generated using the (Dijkstra algorithm) to cover most free configurations. Static obstacles are denoted by black blocks, while dynamic obstacles are represented by blue wide arrows. Finally, the robot is acted by a blue polygon. The simulation is built using Lenovo intel Core i7 8Gen laptop with a Linux 20.04, Python-3 under the Robotic Operating System (ROS) framework, and Rviz environment.

A. Experiment-1: (Global Path - Static)

In this experiment, the comparisons have been conducted by implementing (A*, Dijkstra, and the Dijkstra-PF) on a static offline map of size (300*150 cm) for (100 runs). The planner efficiency as the experimental results illustrated in Table I below, is measured based on the time-consuming, cost, and path length until reaching the goal.

The simulation results illustrate that the time and cost consuming of A* is less than both Dijkstra and Dijkstra-PF (D-PF) respectively, due to exploring the only regions with minimum cost based on the heuristic function, while it cannot deal with the dynamic obstacles. On the other side, Dijkstra has to explore all the regions' free obstacles which cause to increase the cost compare to A*. The Dijkstra-PF method works to explore the desirable regions that have acceptable costs. Moreover, in this experiment, the behavior that the potential field takes influences the resulting path making it shorter and smoother compared to that of A* and Dijkstra

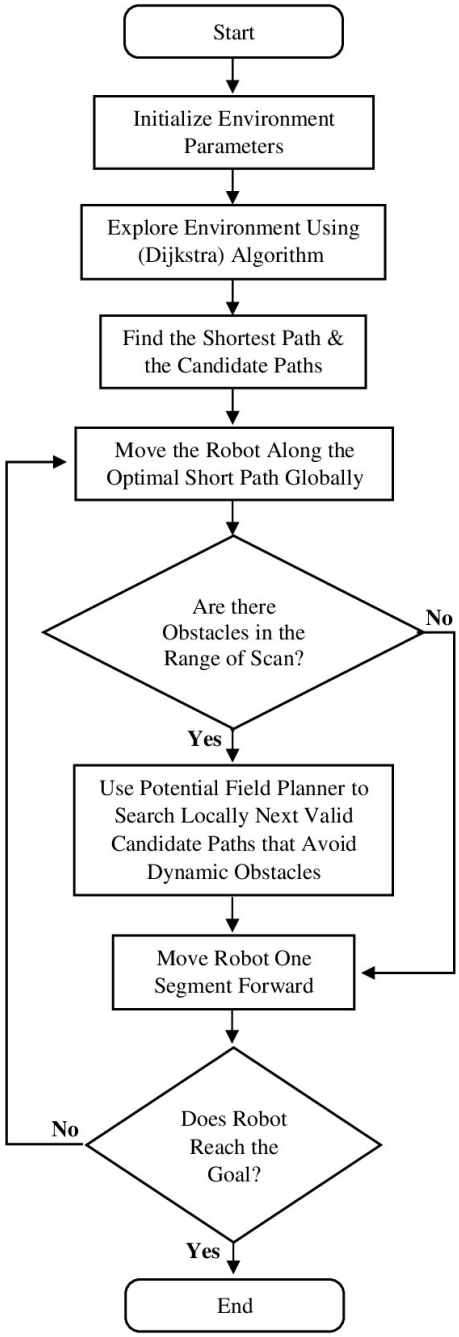

Fig. 6. Blocks Diagram of Dynamic D-PF

TABLE I.
PLANNING METHODS COMPARISON (STATIC MAP)

Method	Time (sec.)	Cost (node)	Length
A*	3.40	311	350 cm
Dijkstra	7.88	453	365 cm
Dijkstra-PF	6.04	440	335 cm

respectively. Figure (7) shows the static map with global paths that generates using the aforementioned methods above.

B. Experiment-2: (Local Path - Dynamic)

In this experiment, the proposed (Dynamic D-PF) has been tested on a dynamic map of size (600*600 cm) and the results of (50 runs) have been evaluated. As demonstrated in Table II, the range of both time and cost values was reasonable between the highest and lowest value. Finally, the standard deviation was low and the median value average was also convincing. In this experiment, the Dynamic D-PF has the ability to deal with the dynamic obstacles based on employing the potential field in case of facing moving obstacles accidentally where the forces of repulsive and attractive guide the robot away from the obstacles and move toward the goal by selecting the nearest alternative safe path to be adopted. Figure (8) simulates the dynamic path planning.

TABLE II.
DYNAMIC PLANNING BASED ON ROBOT VELOCITY (8 CM/SEC) AND THE DECISION TIME

Dynamic D-PF	Time (sec.)	Cost (node)	Length
STD.	2.2	34.6	18.3 cm
Mean	114.5	1040	820 cm
Median	113.2	1025	810 cm
Lowest Value	112	1004	800 cm

The results of the experiments showed that the proposed method (Dynamic D-PF) performed well in terms of path length, safety, execution time, and demonstrating its effectiveness as a path planning solution in dynamic 2D environments. In comparison, the proposed method was found to provide a good balance between (A* and Dijkstra) based on path length, safety, and execution time. Moreover, the Dynamic D-PF in this stage can deal with the dynamic obstacles (moving obstacles that appear accidentally) due to the benefit of merging the Potential Field (PF) and Dijkstra which makes the Dynamic D-PF detects the obstacles efficiently and find another safe path quickly.

The algorithm of Dijkstra is capable of generating the shortest optimal path by searching all the available configurations

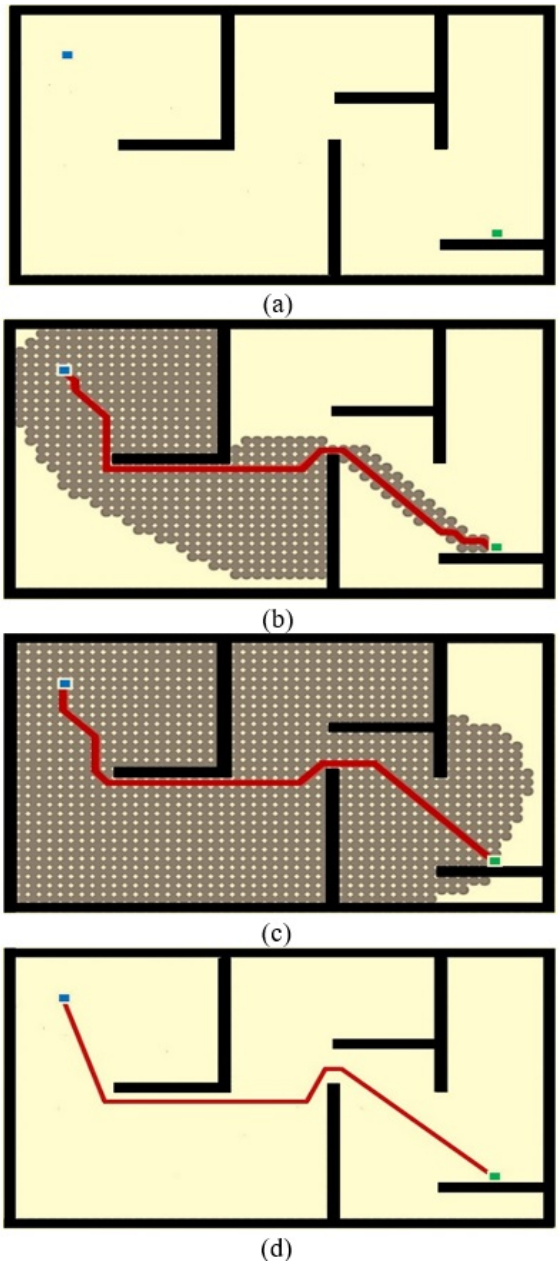

Fig. 7. Offline Path Planning (a) Static Map (b) A* Path (c) Dijkstra Path (d) Dijkstra-PF Path

(spaces) within the given environment. Although the A* algorithm utilizes a heuristic function to identify a single optimal path without the need to explore the entire map, it requires less time than the Dijkstra algorithm. However, in the context of this work, it is necessary to create and store the desired paths, except for those that extend far from the goal. This is done to provide alternative paths that can be used when dynamic obstacles unexpectedly intersect with the global path

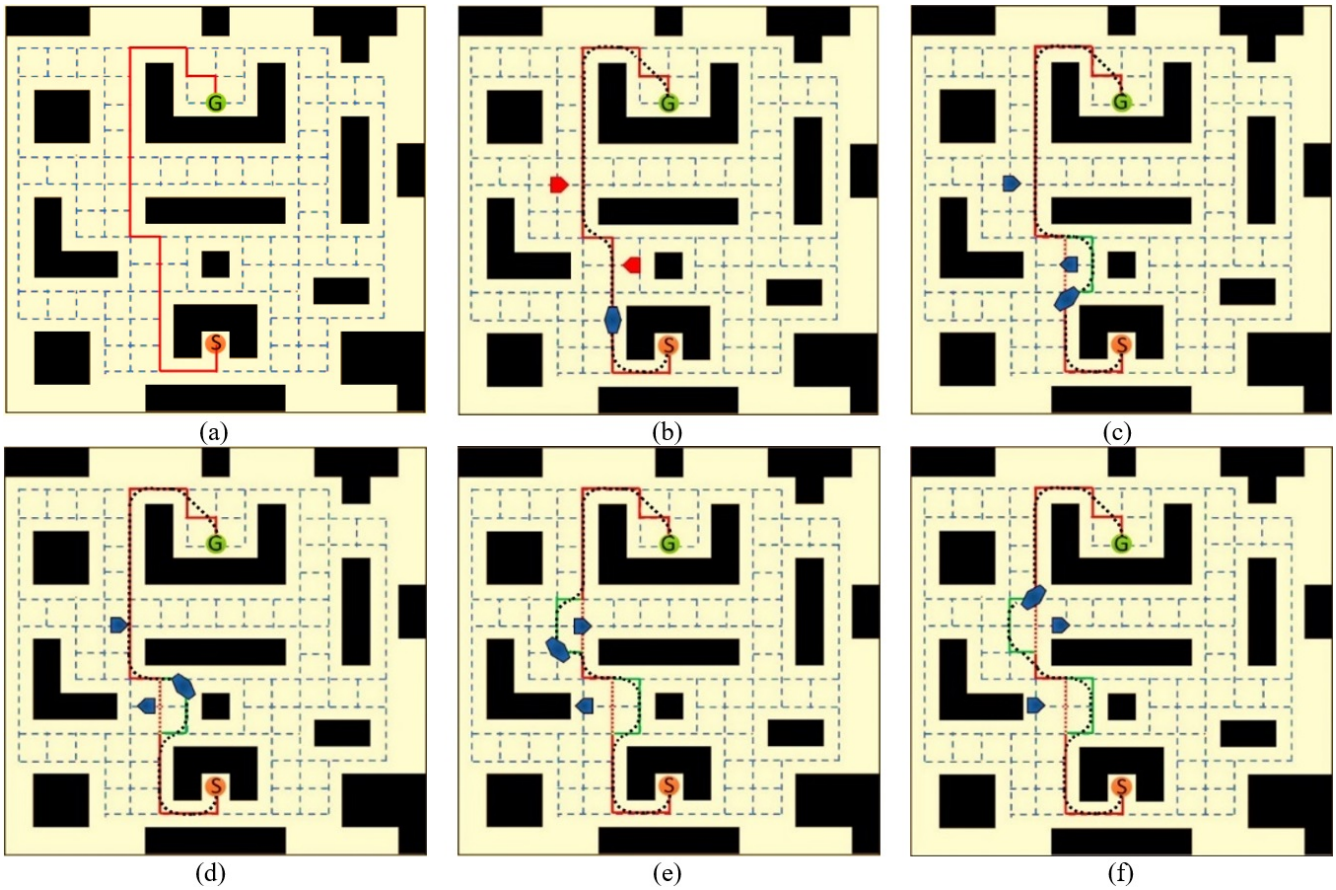

Fig. 8. Snapshots of Real-Time Dynamic Path Planning

of the Dijkstra algorithm.

IV. Conclusion

In this paper, a new path planning method was proposed that combines the strengths of Dijkstra's algorithm and the Potential Field obstacle avoidance method to provide a robust and efficient solution for online path planning in 2D environments with dynamic obstacles. The proposed method (Dynamic D-PF) consists of two phases: Path Generation, Path Selection. The results of the experiments showed that the proposed Dynamic D-PF method outperformed the basic methods (Dijkstra and Potential Field) in terms of path Optimality (short and smooth) and safety (obstacles-free). The results also demonstrate the effectiveness of the proposed method and provide evidence for its possible use in real-world applications. On the other side, the proposed method has some limitations (takes more time compared to Dijkstra and Potential Field each separately). Future work will focus on improving the robustness and efficiency of the proposed method. Additionally, the proposed method will be tested in more complex and realistic environments, such as 3D environments and environments with multiple robots. These improvements and extensions will further demonstrate the potential of the proposed method for mobile robotics applications.

Conflict of Interest

The authors have no conflict of relevant interest to this article.

References

[1] M. Castillo-Lopez, P. Ludivig, S. A. Sajadi-Alamdari, J. L. Sanchez-Lopez, M. A. Olivares-Mendez, and H. Voos, "A real-time approach for chance-constrained motion planning with dynamic obstacles," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3620–3625, 2020.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] D. B. Johnson, "A note on dijkstra's shortest path algorithm," *Journal of the ACM (JACM)*, vol. 20, no. 3, pp. 385–388, 1973.

[4] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[5] S. Al-Ansarry and S. Al-Darraji, "Hybrid rrt-a*: An improved path planning method for an autonomous mobile robots.," *Iraqi Journal for Electrical & Electronic Engineering*, vol. 17, no. 1, 2021.

[6] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[7] L. Li-sang, L. Jia-feng, Y. Jin-xin, H. Dong-wei, Z. Jishi, J. Huang, and P. Shi, "Path planning for smart car based on dijkstra algorithm and dynamic window approach," *Wireless Communications & Mobile Computing (Online)*, vol. 2021, 2021.

[8] S. J. Fusic, P. Ramkumar, and K. Hariharan, "Path planning of robot using modified dijkstra algorithm," in *2018 National Power Engineering Conference (NPEC)*, pp. 1–5, IEEE, 2018.

[9] D. S. Nair and P. Supriya, "Comparison of temporal difference learning algorithm and dijkstra's algorithm for robotic path planning," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1619–1624, IEEE, 2018.

[10] S. M. Ahmadi, H. Kebriaei, and H. Moradi, "Constrained coverage path planning: evolutionary and classical approaches," *Robotica*, vol. 36, no. 6, pp. 904–924, 2018.

[11] X. Li, "Path planning of intelligent mobile robot based on dijkstra algorithm," in *Journal of Physics: Conference Series*, vol. 2083, p. 042034, IOP Publishing, 2021.

[12] H. I. Kang, B. Lee, and K. Kim, "Path planning algorithm using the particle swarm optimization and the improved dijkstra algorithm," in *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, vol. 2, pp. 1002–1004, IEEE, 2008.

[13] Z. Cheng, H. Zhang, and Q. Zhao, "The method based on dijkstra of multi-directional ship's path planning," in *2020 Chinese Control And Decision Conference (CCDC)*, pp. 5142–5146, IEEE, 2020.

[14] R. Szczepanski, T. Tarczewski, and K. Erwinski, "Energy efficient local path planning algorithm based on predictive artificial potential field," *IEEE Access*, vol. 10, pp. 39729–39742, 2022.

[15] Y. Singh, S. Sharma, R. Sutton, and D. Hatton, "Towards use of dijkstra algorithm for optimal navigation of an unmanned surface vehicle in a real-time marine environment with results from artificial potential field," 2018.

[16] B. Kasmi and A. Hassam, "Comparative study between fuzzy logic and interval type-2 fuzzy logic controllers for the trajectory planning of a mobile robot," *Engineering, Technology & Applied Science Research*, vol. 11, no. 2, pp. 7011–7017, 2021.

[17] X. Chen, X. Chen, and G. Xu, "The path planning algorithm studying about uav attacks multiple moving targets based on voronoi diagram," *International Journal of Control and Automation*, vol. 9, no. 1, pp. 281–292, 2016.

[18] W. Yang, P. Wu, X. Zhou, H. Lv, X. Liu, G. Zhang, Z. Hou, and W. Wang, "Improved artificial potential field and dynamic window method for amphibious robot fish path planning," *Applied Sciences*, vol. 11, no. 5, p. 2114, 2021.

[19] G. Klančar, A. Zdešar, and M. Krishnan, "Robot navigation based on potential field and gradient obtained by bilinear interpolation and a grid-based search," *Sensors*, vol. 22, no. 9, p. 3295, 2022.

[20] J. Luo, Z.-X. Wang, and K.-L. Pan, "Reliable path planning algorithm based on improved artificial potential field method," *IEEE Access*, vol. 10, pp. 108276–108284, 2022.

[21] H. Qin, S. Shao, T. Wang, X. Yu, Y. Jiang, and Z. Cao, "Review of autonomous path planning algorithms for mobile robots," *Drones*, vol. 7, no. 3, p. 211, 2023.