Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 5 (2023)

Path Planning for Heterogeneous Robots in ROS based Warehouse Environment

[1]Smt. Thilagavathy R, [2]Dr. Sumithra Devi K A

[1] Department of ECE, DSATM, Bengaluru, India
[2] Department of Data Science, DSATM, Bengaluru, India

Abstract— Autonomous robots have become an inevitable necessity in many applications with defense and scientific interest. Unmanned rovers are currently playing a crucial role in the field of planetary exploration. Path planning is an essential part of any autonomous robot. This task must be executed in a manner that ensures the robot identifies a viable route without encountering any obstacles in its path. In order to address this challenge, various path planning algorithms, such as Dijkstra, A*, Dynamic A* (D*), Dynamic A* Lite (D* Lite), Rapidly exploring Random Tree (RRT), and Probabilistic Road Map (PRM), have been explored, implemented, and analyzed using Python Idle 3.11. Our research has demonstrated that, for heterogeneous robots, path planning involved the integration of the A* algorithm for global path planning and the Dynamic Window Approach (DWA) algorithm for local path planning.

Index Terms— Autonomous robots, Rovers, Dijkstra, A*, D*, D* Lite, RRT, PRM, DWA.

1. Introduction

Navigation, a critical aspect of mobile robotics, refers to the precise determination of the robot's position, the formulation of a path plan, and the subsequent adherence to that planned path. The mobile-robots are capable of moving around in their environment and carrying out intelligent activities autonomously, thus having extensive realistic applications, including Locomotion & exploration, payload transportation, surveillance, cleaning services & rescue works. For a mobile robot to execute its functions effectively, it requires three essential components: localization, mapping, and path planning. Initially, the robot's position must be accurately determined in relation to its immediate environment. Subsequently, the robot relies on a map to recognize its surroundings and navigate accordingly. Thirdly, the robot must engage in path planning to determine the optimal route for accomplishing a specific task[1]. The primary task in navigation is either to reach a predetermined goal or to follow a predetermined path without any collisions. Autonomous navigation is subdivided into four main subtasks [2] (see Fig. 1) The sensory system captures the robot's surrounding environment (Perception). 2) Determining the robot's position within the environment (localization). 3) The robot determines its maneuvering strategy to safely reach the destination without encountering any collisions (path-planning). 4) The robot's motions are controlled to follow the desired path (motion control). Out of the four mentioned tasks, path planning stands out as a crucial area of focus in this research. This paper explores various contemporary methods for robot path planning in the context of guiding mobile robots within a warehouse environment.

Path planning can be categorized into two main types: point-to-point and complete coverage. A complete coverage path planning is carried out when it is necessary to inspect all positions, as is the case for a cleaning robot, for example. Point-to-point path planning is executed when the objective is to move from the initial position to the goal position [3]. In the context of mobile robotics, path planning typically involves the discovery of a collision-free, shortest, and smooth route from a given starting point to a desired destination while minimizing the path cost. The intricacy of the issue escalates as the system's degrees of freedom increase. The determination of the most suitable path is contingent upon various constraints and conditions, such as prioritizing the shortest route between endpoints or achieving the minimum travel time while avoiding collisions. In certain cases, constraints and objectives may be intertwined, as in the effort to minimize energy consumption without surpassing a predefined travel time threshold. Path planning can be applied in environments that are fully known, partially known, or completely unknown, where data is acquired from sensors mounted on the system and used to update environmental maps, thus guiding the motion of the robot or

autonomous vehicle.

The robot's operational environment is rarely static, and it often has many moving obstacles. This environment is called dynamic environment and is representative of the real-world. The robot will need to decide how to proceed when one of these obstacles is obstructing its path without a collision probability. Path planning algorithms are it well- typically categorized into distinct groups, specifically global and local methods. Global path planning, also known as global navigation planning, relies on a priori knowledge, such as an environmental map, to formulate the path, making suited for planning within static environments. Local path planning, also known as obstacle avoidance planning, dynamic path planning, or real-time navigation planning, involves adjusting the path as the robot moves through the environment in response to any changes in the surroundings [4].

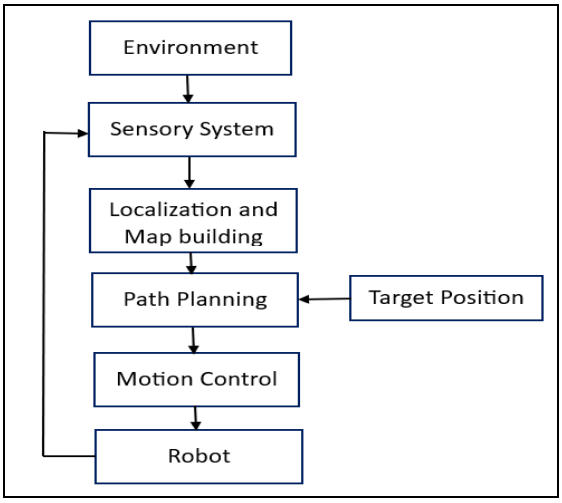


Fig 1: Basic steps involved in Robot Navigation

The remainder of the paper is structured as follows: Section II discusses related research; Section III elucidates path planning principles; Section IV offers an overview of several path planning algorithms along with their pseudocode; Section V presents and evaluates the outcomes of experimental simulations and the implementation of path planning algorithms; Section VI outlines the conclusion and outlines potential future research directions.

2. Related Works

Numerous literature reviews have been undertaken in the early stages to explore motion planning for multiple robots. The widely focused are mobile robots, and then followed by UAVs and autonomous underwater vehicles (AUVs). All the papers have discussed path-planning strategies from classical to emerging AI techniques, such as reinforcement learning (RL) and machine learning (ML). Different motion planning techniques are analyzed with a prime focus on highway planning and UGVs [5]. Decision-making and path-generation concepts are discussed to elaborate motion planning. Findings reveal that a huge number of algorithms are reviewed in this chapter. This study encompasses not only cutting-edge research but also suggests decomposition methods for highway motion planning and encourages autonomous driving. Various methods developed on motion planning policy are reviewed [6].

This chapter is focused on mobile robots in an unstructured environment but has explored some studies on UAVs. The conventional and emerging deep reinforcement learning (DRL) methods that involve multi-robot systems, meta learning, and imitation learning are enlightened. The primary obstacles preventing real-time applications are believed to be limited theoretical progress and a lack of interpretability. Researchers survey studies that applied ML for control and motion planning in the navigation of mobile robots [7]. They conduct a comparative analysis between machine learning (ML) approaches and traditional methods within the framework of navigation. Findings reveal that classical navigation issues are required to be examined with an ML perspective. It further evaluates that despite advances, classical approaches are unable to solve navigation

problems. A comprehensive and clear understanding related to opportunities, limitations, relationships, and the future of different motion planning algorithms is presented [8].

Traditional algorithms to policy gradient reinforcement learning algorithms are discussed for intelligent robots. This study paves the way for improved motion planning algorithms. Optimization techniques for motion planning of unmanned aerial vehicles (UAVs) are addressed in Reference [9]. Findings reveal that swarm-based optimization approaches are preferred by researchers due to their exceptional ability in complex scenarios. A survey is carried out to evaluate various methods of motion planning and task planning for the cooperative working of multiple mobile robots [10]. A taxonomy based on system capabilities is proposed in this study that applies to single-robot systems and multi-robot systems. Various motion planning methods, from classical to reinforcement learning (RL) approaches, are reviewed for single-robot and multirobots [11]. It covers different types of robots such as UAVs, wheeled mobile robots (WMRs), AUVs, etc. It concludes that motion planner based on RL is model-free and achieves integration of both the local and the global planner but shows various limitations that hinder its real-time applications. The conclusion drawn is that an RL-based motion planner is model-free, leading to the integration of both the local and global planners. However, it exhibits several limitations that impede its real-time application.

3. Path Planning

Path planning involves identifying a sequence of feasible robot configurations called trajectories that allow moving a robot from its initial stage to its final destination with collision avoidance and obstacle avoidance for completing a given task. It involves various variables such as robots' dynamics, kinematics, environment, and task constraints. Path planning optimizes a robot's motion by enhancing its throughput and minimizing its cycle time. It can be utilized to assess a process's feasibility and to estimate potential problems before deploying robots. The primary challenge in the advancement of robots, particularly autonomous vehicles, lies in devising a mechanism by which they possess the capability to formulate plans in various situations. Therefore, plan planning is essential in the deployment of multiple robots in an environment consisting of obstacles. The degree of plan planning problem varies according to a couple of factors whether all the obstacles' information regarding their locations, sizes, and motion, is known before the deployment of the robot or whether the obstacles are stated or dynamic in an environment [12].

Path planning algorithms include several classification methods, they are distinguished according to the available environmental knowledge. The main categories of path planning are classical algorithms and soft computing techniques. Classical algorithms are comprised of cell decomposition, road map, Voronoi-diagram, and potential field. Conversely, fuzzy logic, hybrid and evolutionary approaches, and artificial neural networks (ANN) are examples of soft computing techniques. As per a recent investigation of multi-robot systems, path-planning methods are classified into four main groups: classical approaches, heuristic algorithms, artificial intelligence (AI) techniques, and bio-inspired algorithms [13]. Figure 2 shows classification of these path-planning algorithms.

Classical approaches usually involve a predefined graph that requires high computational space and time. These techniques do not ensure completeness and are not capable of re-plan the path in the application. These methods can be divided into three categories: sampling-based, graph-based, and artificial potential field (APF)[14]. The heuristic approaches solve the problems that cannot be addressed by other approaches and estimate an approximate solution rather than an exact solution. Hence, these algorithms are sometimes referred to as approximation algorithms. These methods produce cost functions to assess the path and are readily applied. It explores a subspace within the search space and generates results that are in the proximity of optimality. Moreover, they require lower space and runtime. A-star (A*) search algorithm and D* algorithm are extensively applied through heuristic algorithms [15]. Intelligent systems have garnered increased attention in recent times.AI techniques are developed to overcome the limitations of traditional reinforcement learning. AI-based algorithms and models possess self-learning abilities and have completed characteristics for the path planning of multiple robots with faster convergence. Researchers have focused more on machine learning (ML) algorithms, reinforcement learning (RL), neural networks (NN), fuzzy logic, etc [16]. Bio-inspired techniques are inspired by the behaviour of animals and use particles to generate paths. They are primary algorithms for the path planning of multiple robots because they show computational efficiency and have powerful implementations.

Bio-inspired methods encompass various approaches, such as particle swarm optimization (PSO), pigeon-inspired optimization (PIO), ant colony optimization (ACO), genetic algorithm (GA), gray wolf optimizer (GWO), and various other bio-inspired techniques [17].

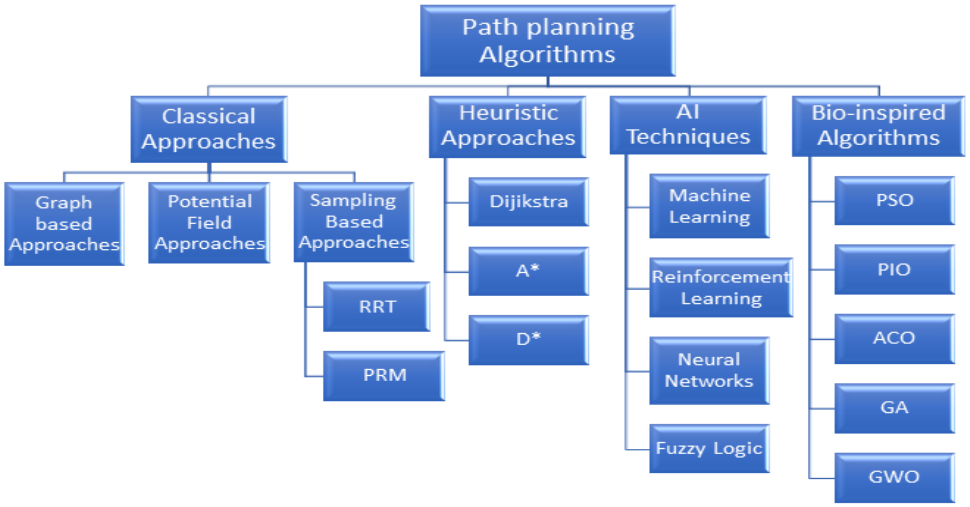


Fig 2: Classification of Path planning Approaches

4. Path Planning Algorithms

A. Dijkstra Algorithm

One of the earliest algorithms for path planning is the Dijkstra algorithm. Edsger Wybe Dijkstra introduced Dijkstra's algorithm (DA), the Dutch scientist, in 1956 and published in 1959 [18]. This algorithm sequentially explores the most promising subnodes based on the greedy principle, and it utilizes the relaxation method to enhance the path selection. Ultimately, the optimal path is recorded in a readable list, thus addressing the optimal path planning challenge. The Dijkstra algorithm yields more favorable planning outcomes when dealing with smaller-scale map data, satisfying the specified requirements. Nevertheless, in cases where the map data volume is extensive, the planning results are suboptimal and fail to meet the planning requirements.

In a weighted graph, the weight of a path corresponds to the cumulative sum of the weights of its constituent edges. This algorithm operates under the assumption that there are n vertices, and the distances between each pair of connected points are known, with at least one connection existing between any two vertices. The objective of this algorithm is to identify a path with the shortest overall length, starting from the initial point and concluding at the target point. At each step, the algorithm selects the shortest distance from the starting point, and then updates the distances of other vertices based on a distance variable x (equation 1). For each vertex v:

$$Dist(v) = \min(dist(v), dist(v) + w(x,v)) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Where w(x,v) = Weights of the arc between x and v

The global path planning for the robot utilizes the Dijkstra algorithm, with the algorithm outlined in Figure 3. Initially, the robot's navigation commences by establishing both the starting point and the destination point. Subsequently, two arrays are created to facilitate the storage of points for the path to be determined and the points for the path that has been ascertained. Following this, the algorithm calculates the distances between the central point and the eight adjacent points, utilizing the starting point as the reference center. Subsequently, the algorithm saves the point with the shortest distance, designates this point as the new center, and proceeds to calculate the distances between the starting point and the neighboring points based on this updated center point. for each of the calculated points, we selected the solution with the smallest distance. This process continues iteratively, calculating adjacent points until the target point is reached, ultimately yielding the shortest path planning route as the output. Broadly speaking, the algorithm assesses and contrasts the node weights within the

graph from a global standpoint, thereby determining the global shortest path [19].

Algorithm 1: Dijkstra Algorithm

```
1.  Input: Map, Starting point S, Target point T
2.  Output: The shortest path SP
3.  Init openlist[], closedlist[]
4.  While (openlist[] is not empty)
5.          Center = The point with shortest distance in the openlist[]
6.          Put Center from openlist[] to closedlist[]
7.          If (Center is not T) Then
8.             Extract path, break
9.          Else iterate over eight adjacent points
10.             If (point is in closedlist[] or encounter an obstacle) Then
11.                 Skip this point
12.             Else cumulative the path
13.                 If (point is in openlist[]) Then
14.                     If (the cumulative distance<the original recorded distance) Then
15.                         Replace the data
16.                     Else add the point to openlist[]
17.                     End If
18.                 End If
19.             End If
20.             Arrange the points in openlist[] from small to large
21.         End If
22. End While
```

Fig 3: Pseudocode of the Dijkstra Algorithm

B. A Star (A*) Algorithm

The A* Algorithm is a well-known and widely used path planning algorithm for graph traversal. A* functions in a manner akin to Dijkstra's algorithm, with the key distinction being that it directs its search toward the most promising states, which can potentially reduce the computational effort required, as noted in reference [20]. A* is commonly employed to seek a solution that is close to optimal [21] based on the available dataset or node. A* is predominantly utilized in static environments, but there are scenarios in which this algorithm finds application in dynamic environments as well [22]. Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described A* algorithm in 1968. It is an enhanced version of Edsger Dijkstra's algorithm. The A* algorithm [23] is an additional graph-based path planning technique employed to assist the robot in discovering the optimal path within grid-decomposed static grid maps. The environment, which includes both free spaces and obstacles, is represented using a collection of uniform, regular grids. A* employs a heuristic-based variation of the Dijkstra algorithm to achieve an optimal solution for the robot. One limitation of A* is its utilization of uniform grid representation, which necessitates allocating significant memory resources for areas that might not be traversed or lack obstacles.

While Dijkstra's Algorithm is effective in finding the shortest path, it can be inefficient as it spends time exploring directions that do not hold promise. Greedy Best First Search is inclined to explore promising directions; however, it may not necessarily identify the shortest path. The A* algorithm combines the actual distance from the start with an estimated distance to the goal, enabling it to find paths to a specific location. A* prioritizes paths that appear to be progressing closer to the goal. It is a best-first search algorithm, it solves problems by searching among all the possible paths to the goal location and selects the one path that incurs the smallest cost i.e. least distance travelled or shortest time taken to reach the goal location. A* is represented using weighted graphs and initiates its journey from a designated node within the graph. It proceeds to build a tree of paths originating from this node, incrementally extending these paths until one of them reaches the goal node or target position. The pseudocode for the A* algorithm is depicted in Figure 4. A* extends paths that are already more cost-effective by employing the following function (equation 2):

$$f(n) = g(n) + h(n)\text{……..(2)}$$

where,
n = the last node on the path
f(n) = total projected cost of the path via node n
g(n) = the current cost to reach node n
h(n) = Projected cost from n to target.

This is the heuristic part of the cost function. The heuristic is problem-specific. In order for the algorithm to determine the genuine shortest path, the heuristic function must be admissible, implying that it should never provide an overestimate of the actual cost to reach the nearest goal node. The heuristic function can be computed through a variety of methods:

Algorithm 2: A* Algorithm

```
1.  Input: Map, Starting point S, Target point T
2.  Output: The shortest path SP
3.  Init openlist[], closedlist[]
4.      Let open list have only the starting node
5.      Let closed list empty
6.  While (the destination node has not been reached):
7.  consider the node with the lowest f score in the open list
8.    If (this node is destination node) Then
9.    We are finished
10.   Else If (put the current node in the closed list and look at all of its neighbors)
11.   For (each neighbor of the current node):
12.         If (neighbor has lower g value than current and is in the closed list)
13.         replace the neighbor with the new, lower, g value current node is now the
14.         neighbor's parent
15.         Else If (current g value is lower and this neighbor is in the open list).
16.         replace the neighbor with the new, lower, g value change the neighbor's parent
17.         to current node
18.         Else If this neighbor is not in both lists:
19.         add it to the open list and set its g .
20.         End If
21.     End If
22. End While
```

Fig 4: Pseudocode of the A-Star or A* Algorithm

- **Manhattan Distance:**

In this method h(n) is computed by (eqn. 3) calculating the total number of squares moved horizontally and vertically to reach the target square from the current square. Here any obstacles and diagonal movement are ignored.

$$h = |x_{start} - x_{destination}| + |y_{start} - y_{destination}|\text{……..(3)}$$

- **Euclidean Distance Heuristic:**

This heuristic is slightly more accurate than its Manhattan counterpart. If we try run both simultaneously on the same maze, the Euclidean path finder favors a path along a straight line. This is more accurate, but it is also slower because it has to explore a larger area to find the path (eqn. 4).

$$h = \sqrt{(x_{start} - x_{destination})^2 + (y_{start} - y_{destination})^2}\text{……..(4)}$$

C. Dynamic A Star (D*) Algorithm

Building upon the A* algorithm, Anthony Stentz introduced the Dynamic A* algorithm, commonly known as the D* algorithm, in 1994. The D* algorithm is characterized as a reverse incremental search algorithm. It is also considered a graph search algorithm. It's named as such because it shares similarities with the A* algorithm, with the key distinction being its dynamic nature. This means that the cost functions can evolve and change over time, or during the problem-solving process. The pseudocode for the D* algorithm is

depicted in Figure 5.

Algorithm 3: D* Algorithm

```
1.  while (!openList.isEmpty()) {
2.      point = openList.getFirst();
3.      expand(point);
4.  }
5.
6.  void expand(currentPoint) {
7.    boolean is Raise = is Raise(currentPoint);
8.    double cost;
9.    for each (neighbor in currentPoint.getNeighbors()) {
10.     if (is Raise) {
11.       if (neighbor.nextPoint == currentPoint) {
12.         neighbor.setNextPointAndUpdateCost(currentPoint);
13.         openList.add(neighbor);
14.       } else {
15.         cost = neighbor.calculateCostVia(currentPoint);
16.         if (cost < neighbor.getCost()) {
17.           currentPoint.setMinimumCostToCurrentCost();
18.           openList.add(currentPoint);
19.         }
20.       }
21.     } else {
22.       cost = neighbor.calculateCostVia(currentPoint);
23.       if (cost < neighbor.getCost()) {
24.         neighbor.setNextPointAndUpdateCost(currentPoint);
25.         openList.add(neighbor);
26.       }
27.     }
28.   }
29. }
30.
31. boolean isRaise(point) {
32.   double cost;
33.   if (point.getCurrentCost() > point.getMinimumCost()) {
34.     for each(neighbor in point.getNeighbors()) {
35.       cost = point.calculateCostVia(neighbor);
36.       if (cost < point.getCurrentCost()) {
37.         point.setNextPointAndUpdateCost(neighbor);
38.       }
39.     }
40.   }
41.   return point.getCurrentCost() > point.getMinimumCost();
42. }
```

Fig 5: Pseudocode of the D-Star or D* Algorithm

In simpler terms, it can be described as the process of path planning in an environment where comprehensive data about the terrain and obstacles may not be readily accessible, meaning that certain obstacles in the environment and the complete traversal cost information between each grid are not known in advance. The database containing costs and obstacle information should be continually updated whenever sensors onboard the robot detect a deviation from the previously recorded data. The D* algorithm manages a robot's state until it is eliminated from the open list. Concurrently, it computes a sequence of states along with back pointers, which are used either to guide the robot to the goal position or to adjust the cost in response to a detected obstacle, and subsequently, the affected states are added to the open list. States within the open list are handled until the path cost from the present state to the goal becomes lower than a designated minimum threshold. Any changes in cost are propagated to the subsequent state, enabling the robot to proceed by following back pointers within the updated sequence toward the goal, as detailed in reference [24]. D* has been shown to be more than 200 times faster than an optimal re-planner, as indicated in references [24][25]. The primary limitation of the D* algorithm is its considerable memory usage in comparison to other D* variants, as highlighted in reference [26]. The formula for distance measurement (equation 5),

$$H(x)=H(y)+C(y, x)\ldots\ldots\ldots(5)$$

Where,
H(y) - The measurement of the separation between the target point and point x, and
C (y, x) - The distance measurement between point y and point x can be substituted with the real, physical distance between these two points within the algorithm.

Much like A*, D* also keeps an OPEN list of states. This list is utilized for disseminating information regarding modifications to the arc cost function and for computing path costs to states within the space. Each state S is linked with a corresponding tag t(S), with its value designated as "NEW" if it has never been part of the OPEN list, t(S)="OPEN" when S is currently included in the OPEN list, and t(S)="CLOSED" if it has been removed from the OPEN list. In contrast to the A* algorithm, which commences at the initial point and proceeds towards the goal by employing a heuristic function, D* initiates its journey from the target position T and works backward towards the initial position, without employing a heuristic similar to A*. Again, quoting Stentz, "For each state S, D* maintains an estimate of the sum of the arc costs from S to T given by the path cost function h (T, S). Given the proper conditions, this estimate is equivalent to the optimal (minimal) cost from state S to T, given by the implicit function O (T, S). For each state S on the OPEN list (i.e., t(S) =OPEN), the key function, k (T, S), is defined to be equal to the minimum of h(T,S) before modification and all values assumed by h(T,S) since S was placed on the OPEN list[27]. The key function classifies a state S on the OPEN list into one of two types: a RAISE state if k (T, S) be min (k(S)) for all S such that t(S) =OPEN. The parameter k_{min} represents an important threshold in D*: path costs less than or equal to k_{min} are optimal, and those greater than k_{min} may not be optimal. The parameter k_{old} is defined to be equal to k_{min} prior to most recent removal of a state from the OPEN list. If no states have been removed, k_{old} is undefined."

D. Dynamic A Star Lite (D* Lite) Algorithm

The D* Lite algorithm is a path planning technique introduced by Koenig S and Likhachev M, building upon the Life Planning A* (LPA*) algorithm. The principal distinction between D* Lite and LPA* lies in the search direction, where D* Lite substitutes the target point goal in the Key definition with the relevant data for the starting point start [28]. The D*_lite algorithm initiates by conducting a backward search within a specified map set to determine an optimal path. While advancing towards the target point, the algorithm handles the appearance of dynamic obstacle points by conducting searches within the local scope. The incremental algorithm offers the benefit of having completed the path search for each point. In situations where an obstacle point prevents further progress along the original path, the data from the incremental search can be repurposed to promptly re-plan an optimal path from the current obstruction point, allowing the robot to resume forward movement. Figure 6 depicts the Pseudocode for the D* Lite algorithm.

E. Rapidly Exploring Random Tree (RRT) Algorithm

The Rapidly Exploring Random Tree (RRT) method was originally introduced by LaValle [29]. It is a randomized path planning approach that employs sampling algorithms. It begins with an initial point as the root node and incrementally constructs a roadmap tree from randomly drawn samples (leaf nodes). The primary goal of RRT is to efficiently explore a substantial portion of the configuration space [30]. When the leaf nodes within the random tree encompass the goal point or enter the goal region, it becomes feasible to identify a path from the initial point to the goal point [31]. The RRT algorithm is well-suited for addressing path planning challenges under both holonomic and non-holonomic constraints [32].

An advantage of the RRT method is that it does not necessitate the modeling of the entire planning space. This

Algorithm 4: D* Lite Algorithm

```
1.  Function Key(s)
2.      Return
3.          [min(g(s), rhs(s)) + h(s start, s) + km ; min(g(s), rhs(s))]
4.  Function Update Vertex(s):
5.      If s ? sgoal Then
6.          rhs(s) = min s'?Succ(s) (cost(s,s') + g(s'))
7.      End If
8.      If s ?OPEN Then
9.          OPEN.remove (s)
10.     End If
11.     If g(s) ? rhs(s) Then
12.         OPEN.insert(s, key(s))
13.     End If
14. Function ComputePath():
15.     While
16.         OPEN.TopKey() < Key(sstart)
17.         OR rhs(sstart) ? g(sstart) Do
18.             kold = OPEN.TopKey()
19.             s = OPEN.pop()
20.             If kold < Key(s) Then
21.                 OPEN.insert(s, key(s))
22.             Else If g(s) > rhs(s) Then
23.                 g(s) = rhs(s)
24.                 forall s' ??Pred(s) do
25.                     UpdateVertex(s')
26.                 End
27.             Else
28.             g(s) = 8
29.             forall s' ??Pred(s) U{s} do
30.                 UpdateVertex(s')
31.             End
32. End While
```

Algorithm 4: D* Lite Algorithm (contd...)

```
1.  Function main()
2.      forall s ??S do
3.          g(s) = rhs(s) = 8
4.      End
5.      slast = sstart
6.      OPEN = Ø
7.      rhs(sgoal) =0; km = 0
8.      OPEN.insert(sgoal, key(sgoal))
9.      ComputePath()
10.     While sstart ? sgoal do
11.         sstart = argmin s'?Succ(sstart) (cost(sstart,s') + g(s'))
12.         Move to sstart
13.         Scan for cell changes in the environment
14.         If Cell changes detected Then
15.             km = km + h(slast , sstart)
16.             slast = sstart
17.             forall s ??CHANGES do
18.                 Update cell s state
19.                 forall s' ??Pred(s) U{s} do
20.                     UpdateVertex(s')
21.                 End
22.             End
23.             ComputePath()
24.         End If
25.     End While
```

Fig 6: Pseudocode of the D-Star Lite or D* Lite Algorithm

Algorithm exhibits an extensive coverage of the search space and possesses a wide search range, enabling it to explore unknown regions to a significant extent. However, it also faces the challenge of having a notably high computational cost. Various adaptations and enhancements to the RRT have been put forward by researchers to address these issues. Some of these modifications include the Goal-Bias RRT algorithm, Bi-RRT algorithm, RRT-Connect algorithm, Extend RRT algorithm, Local-Tree-RRT algorithm, Dynamic RRT algorithm, among others. In the goal-bias algorithm, the sampling point is the target node, and it allows for control over the probability of selecting the target point within the algorithm. Before continuing the search, the Dynamic RRT algorithm suggests trimming and combining procedures to remove invalid nodes [33-35].

RRT operates by constructing two trees: one originating from the starting position and the other with its root at the ending position. The two trees exchange roles, with one tree adding a node, and the other attempting to establish a connection to it, repeating this process until a path is discovered. While this may appear to be an inefficient and somewhat non-deterministic approach, it is, in fact, quite swift (capable of planning a new path without significant interruptions for recalculation) and typically manages to find a path on its initial attempt in nearly all cases where a viable path exists. Figure 7 depicts the RRT algorithm's pseudocode.

F. Probabilistic Road Map (PRM) Algorithm

The probabilistic roadmap (PRM) was initially introduced by Kavraki et al. in 1996 [36] with the primary aim of facilitating path planning for a robot within a static workspace. PRM has the ability for multiquery planning [37]. Road maps refer to the connectivity of the robot's configuration free space plotted on a 1-dimension curve or lines. The roadmap has likewise been named highway strategy [38], withdrawal approach, and skeleton system [39]. The optimal path is determined by calculating the distances along the edges, which represent the connections between randomly generated nodes on the map. Nodes are generated by sampling from obstacle-free points on the map, after which these nodes are connected to each other, and the path cost is subsequently assessed. The rapid random cluster strategy delivers results more swiftly than other algorithms but does not guarantee the shortest path, as noted in reference [36]. Figure 8 depicts the PRM

algorithm's pseudocode.

Algorithm 5: RRT Algorithm

```
1.  EPSILON = max length of an edge in the tree;
2.  K = max number of nodes in the tree;

3.   rrt(start,end){
4.          startNode = new Node at start;
5.          startTree = new RRTree(startNode);
6.          endNode = new Node at end;
7.          endTree = new RRTree(endNode);
8.          return makePath(startTree,endTree);
9.  }

10. makePath(t1,t2)
11. {
12.         qRandom = new Node at random location;
13.         qNear = nearest node to qRandom in t1;
14.         qNew = new Node EPSILON away from qNear in direction of qRandom;
15.         t1.add(qNew);
16.         if(there's a path from qNew to the closest node in t2)
17.         {
18.           path = path from qNew to root of t1;
19.           path.append(path from qNew to root of t2);
20.           return path;
21.         }
22.         else if(size(t1) < K)
23.         {
24.           return makePath(t2,t1);
25.         }
26.         Else
27.         {
28.           return FAIL;
29.         }
30. }
```

Fig 7: Pseudocode of the RRT Algorithm

A. Dynamic Window Approach (DWA) Algorithm

The DWA is a velocity-based local planner that calculates the optimal collision-free ('admissible') velocity for a robot required to reach its goal. It translates a Cartesian goal (x, y) into a velocity (v, w) command for a mobile robot. There are two primary objectives: the calculation of a valid velocity search space and the selection of the optimal velocity. The space to be searched, is constructed from the set of velocities which produce a safe trajectory i.e. allow the robot to stop before colliding, given the set of velocities the robot can achieve in the next time slice given its dynamics 'dynamic window'. The optimal velocity is selected to maximize the robots clearance, maximize the velocity and obtain the heading closest to the goal (See Figure 9)[40].

In contemporary times, while numerous path planning methods for robots continue to emerge, many of these algorithms primarily operate in static environments and often overlook the intricacies of dynamic environments. Path planning algorithms designed for dynamic environments are still in a relatively early stage of development, and the research addressing these issues may exhibit certain limitations. Undoubtedly, research on real-time planning in dynamic environments holds a prominent place in the research agenda due to its challenging and practical applications in technology and everyday life.

5. Experimental Results And Discussion

The cited algorithms in the previous section were implemented in simulation environment in this

section. This experiment was run in the Windows 10 Enterprise LTSC on a computer with an Intel Core i5-6200U CPU and 4GB RAM. To illustrate the advantages of the algorithms in various aspects, such as search speed, the number of visited nodes, rotations, path selection, and path length, a dedicated experimental environment is established, and the fundamental algorithms are subjected to a comparative analysis. The experiments aim to establish a starting point and a target point environment.

Algorithm 6: PRM Algorithm

```
1.  INITIALIZE()
2.  for all (cell ? cells) do
3.      cell.dist = distPointQueryLine(cell.origin);
4.  end for
5.  OPEN = {parentCell(ninit ); CLOSED = Ø;
6.  GROWPRMINCELL(cell)
7.  numSamples = 0;
8.  while (numSamples < nodeIncrementPerCell) do
9.      sample random configuration cnew in cell;
10.     cell.numTrials++;
11.     if (isFreeConfig(cnew)) then
12.       add node nnew to N ;
13.       connect nnew to neighbors, add edges to E;
14.       update cell.numComponents of cell;
15.       numSamples++;
16.     end if
17. end while
18. cell.numSamples += numSamples;
19. updateOccupancy(cell);
20. updateConnectedness(cell);
21. updateValue(cell);
22. SOLVEQUERY(ninit ,ngoal )
23. INITIALIZE()
24. add ninit , ngoal to N ;
25. connect ninit , ngoal to neighbors, add edges to E;
26. loop
27.    if (OPEN = Ø) then
28.       report failure;
29.    end if
30.    cell = takeFirst(OPEN );
31.    GrowPRMInCell(cell);
32.    PerformRandomWalksInCell(cell);
33.    if (cell.occupancy > occupancyThresh or cell.numSamples = maxNodesPerCell) then
34.       CLOSED ? cell;
35.    else
36.       OPEN ? cell;
37.    end if
38.    for all (neighbor ? neighbors(cell)) do
39.       if (neighbor ? / CLOSED) then
40.          OPEN ? neighbor;
41.       end if
42.    end for
43.    sortAscendingByValue(OPEN );
44.    if (parentComp(ninit ) = parentComp(ngoal)) then
45.       path = findShortestPath(G);
46.       if (path satisfies quality condition) then
47.          return path;
48.       else
49.          publish path;
50.       end if
51.    end if
52. end loop
53. MAIN()
54. create array cells; N = Ø; E = Ø; G = (N, E);
55. for all (query ? queries) do
56.     solutionPath = SolveQuery(query.ninit,query.ngoal);
57. end for
```

Fig 8: Pseudocode of the PRM Algorithm

Algorithm 7: DWA Algorithm

```
1.  BEGIN DWA (robotPose, robotGoal, robotModel)
2.      desiredV = calculateV(robotPose, robotGoal)
3.      laserscan = readScanner()
4.      allowable_v = generateWindow(robotV, robotModel)
5.      allowable_w = generateWindow (robotW, robotModel)
6.      for each v in allowable_v
7.          for each w in allowable_w
8.              dist = find_dist(v,w,laserscan,robotModel)
9.              breakDist = calculateBreakingDistance(v)
10.             if (dist > breakDist) //can stop in time
11.                 heading = hDiff(robotPose,goalPose, v,w)
12.                 clearance = (dist-breakDist)/(dmax - breakDist)
13.                 cost = costFunction(heading,clearance, abs (desired_v - v))
14.                  if (cost > optimal)
15.                      best_v = v
16.                      best_w = w
17.                      optimal = cost
18.                      set robot trajectory to best_v, best_w
19.                 end if
20.             end if
21.         end for
22.     end for
23. END
```

Fig 9: Pseudocode of the DWA Algorithm

A. Dijkstra Algorithm

World: 100x100, Starting point = -20x-20, Target point = 50x60, No. of random Obstacles = 9, Grid size = 2.0, Robot radius = 1.0. Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by Dijkstra algorithm for randomly generated static world is as in fig. 10.

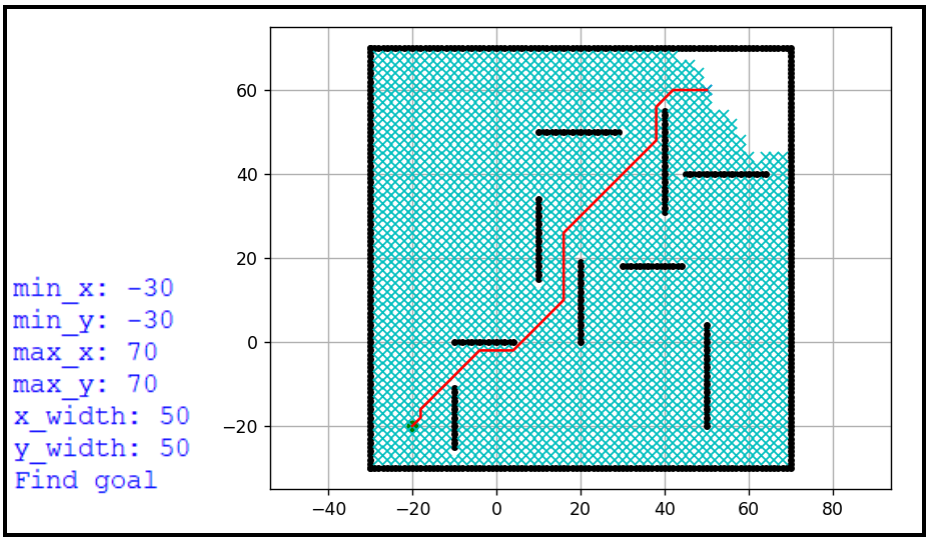


Fig 10: Path planned by Dijkstra algorithm for randomly generated world

B. A Star (A*) Algorithm

World: 100x100, Starting point = -20x-20, Target point = 50x60, No. of random Obstacles = 9, Grid size = 2.0, Robot radius = 1.0. Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by A* algorithm for randomly generated static world is as in fig. 11.

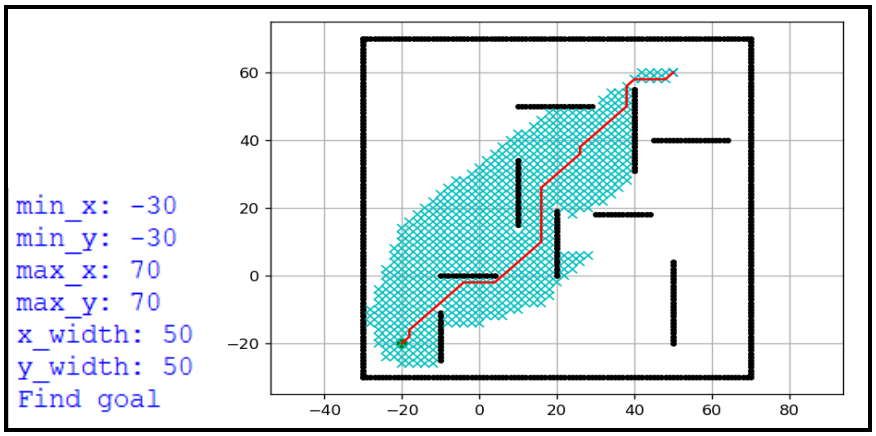
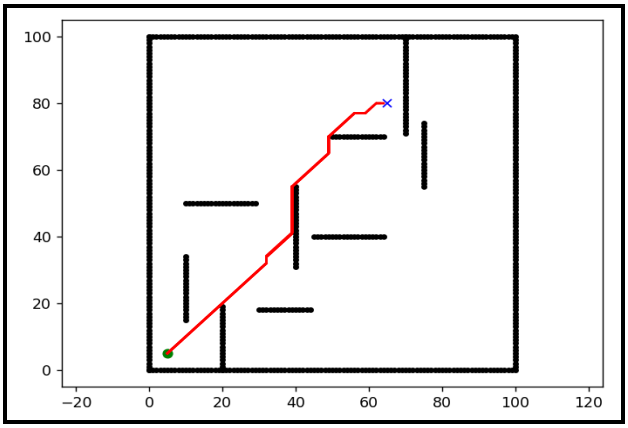


Fig 11: Path planned by A* algorithm for randomly generated world

C. Dynamic A Star (D*) Algorithm

World: 100x100, Starting point = 5x5, Target point = 65x80, No. of random Obstacles = 9, Grid size = 2.0, Robot radius = 1.0. Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by D* algorithm for randomly generated static world is as in fig. 12.



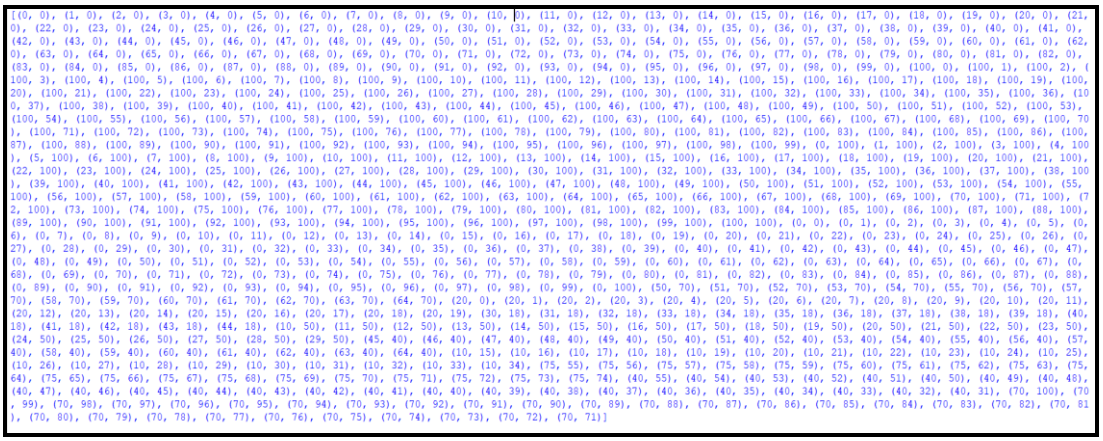


Fig 12: Path planned by D* algorithm for randomly generated world

D. Dynamic A Star Lite (D* Lite) Algorithm

World: 100x100, Starting point = -20x-20, Target point = 50x60, No. of random Obstacles = 9, Grid size = 2.0, Robot radius = 1.0. Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by D* Lite algorithm for randomly generated static world is as in fig. 13 &14.

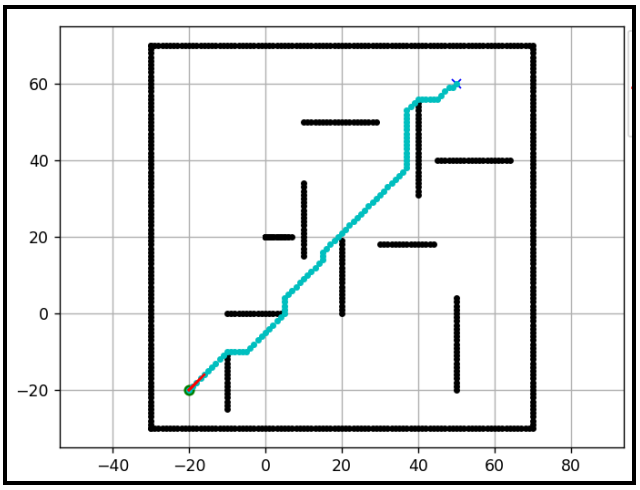


Fig 13: Path planned by D* Lite algorithm without obstacle for randomly generated world

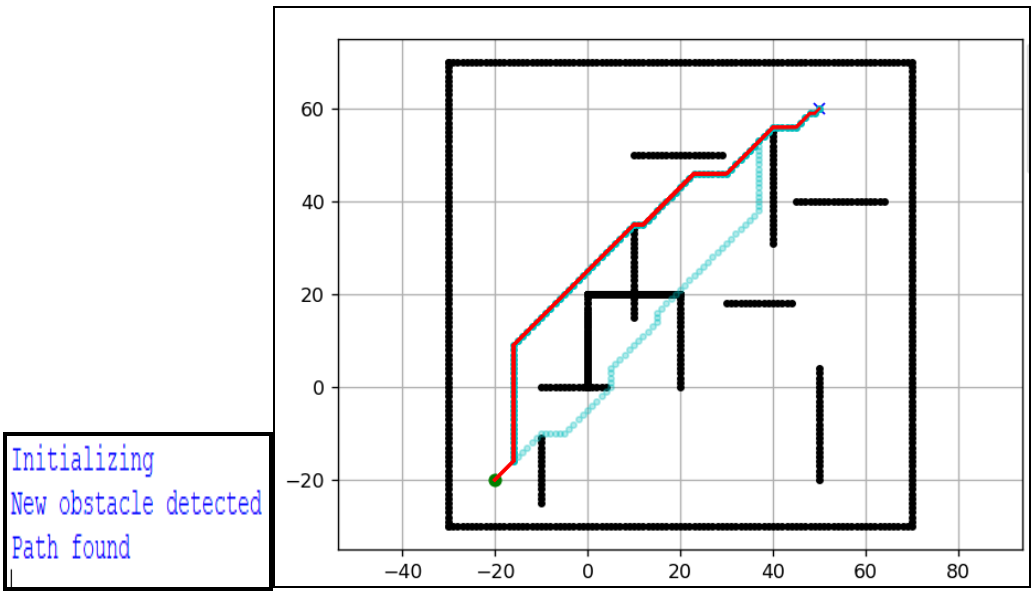


Fig 14: Path planned by D* Lite algorithm with obstacle for randomly generated world

E. Rapidly Exploring Random Tree (RRT) Algorithm

World: 16x16, Starting point = 0x0, Target point = 10x12, No. of random Obstacles = 8, Grid size = 2.0, Robot radius = 0.8, Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by RRT algorithm for randomly generated static world is as in fig. 15.

F. Probabilistic Road Map (PRM) Algorithm

World: 100x100, Starting point = -20x-20, Target point = 50x60, No. of random Obstacles = 9, Grid size = 2.0, Robot size = 5.0. Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by PRM algorithm for randomly generated static world is as in fig. 16.

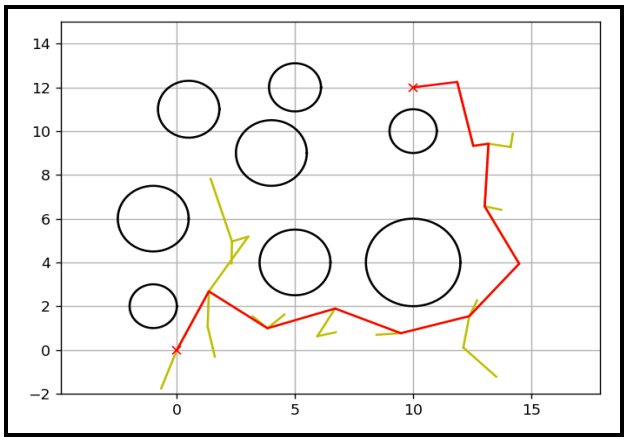


Fig 15: Path planned by RRT algorithm for randomly generated world

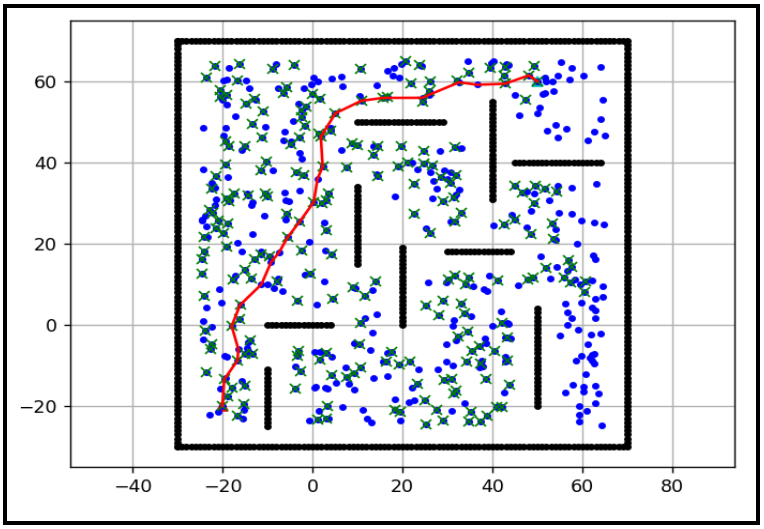


Fig 16: Path planned by PRM algorithm for randomly generated world

G. Dynamic Window Approach (DWA) Algorithm

World: 16x16, Starting point = 0x0, Target point = 10x10, No. of random Obstacles = 26, Grid size = 2.0, Robot radius = 0.8, Colour coding --> Black: Obstacle, Green: Starting point, Cyan: Target point, Red: Path. The Path planned by DWA algorithm for randomly generated dynamic world is as in fig. 17.

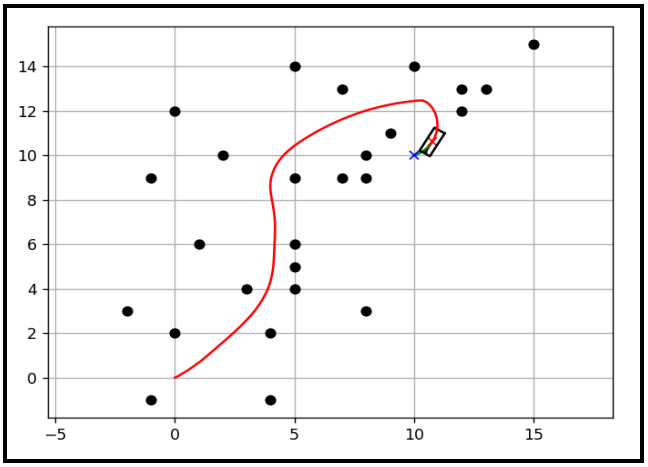


Fig 17: Path planned by DWA algorithm for Dynamic environment

All of these results are one solution to the path planning algorithms. All the algorithms are producing different results in each operation. All the algorithms are 3 times and path distance and consumption time of them are saved. The table I presents the search speed in seconds for the aforementioned algorithms.

Table I: Comparison of Search Speed (in seconds) for Path Planning Algorithms in a Static Environment

Trial No	Dijkstra	A*	D*	D* Lite		RRT	PRM
				Without obstacle	With obstacle		
1	31.73	5.70	14.01	17.51	25.10	9.03	13.07
2	31.66	5.65	13.95	17.10	24.12	8.45	12.90
3	31.95	5.64	13.94	16.80	24.20	6.09	13.85

6. Conclusion

Efficiently discovering the optimal path for heterogeneous robots, which minimizes both time and distance, is crucial for successfully accomplishing their designated tasks. Within this context, the efficiency of several heuristic approaches, including Dijkstra, A*, D*, Dynamic A* Lite, and sampling-based techniques such as RRT and PRM algorithms, is deliberated and assessed using Python. These methods are then compared. These algorithms have been subjected to testing for path planning under various start-goal point configurations and obstacle configurations. In almost all the cases, A* can be used if the environment has been completely mapped already, i.e. the entire topography of the environment is known, and will remain so during the course of the robot mission. Also, it doesn't have huge computational requirements and can be implemented easily. The D*, on the other hand may be used in those environments which are dynamic and whose topography isn't completely known. But in the comparative study, it shows that D* has a larger computational delay in cluttered environments. This happens because the D* keeps a record of the cost from each of the grids to its neighbouring grids, unlike the A*. The RRT and PRM algorithms are rather slow algorithms. Also it gives a non-optimal solution, doesn't guarantee that the solution will be obtained even if it exists. But it comes in handy in highly cluttered environments. In fact this must be selected in such environments only if the planner can afford the high memory requirements. The path planning was executed by amalgamating the A* algorithm for global path planning with the Dynamic Window Approach (DWA) algorithm for local path planning. Simulation and emulation experiments were performed to compare and validate the outcomes related to map construction and path planningFuture work will encompass the development of algorithms that incorporate Artificial Intelligence, enhancing their responsiveness and features to align with the demands of modern technology.

Data Availability Statement: Data sharing not applicable.
Conflicts of Interest: The authors declare no conflict of interest in preparing this article.

References

[1] X. Wang, X. Wang, and D. M. Wilkes, "An Automatic Scene Recognition Using TD-Learning for Mobile Robot Localization in an Outdoor Environment," in Machine Learning-based Natural Scene Recognition for Mobile Robot Localization in An Unknown Environment, Springer, 2020, pp. 293–310.

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots. Cambridge, MA, USA: MIT Press, 2011.

[3] M. Virt, S. Siroki, Á. Nyerges, and V. Tihanyi, "An Online Path Planning Algorithm for Automated Vehicles for Slow Speed Maneuvering," in 2019 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE), 2019, pp. 97–102.

[4] Connell, D., & La, H. M. (2017). "Dynamic path planning and replanning for mobile robots using RRT", in 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017.

[5] Claussmann L, Revilloud M, Gruyer D, Glaser S. A review of motion planning for highway autonomous driving. IEEE Transactions on Intelligent Transportation Systems. 2019;21(5):1826-1848

[6] Sun H, Zhang W, Runxiang Y, Zhang Y. Motion planning for mobile robots—Focusing on deep reinforcement learning: A systematic review. IEEE Access. 2021;9:69061-69081
[7] Xiao X, Liu B, Warnell G, Stone P. Motion planning and control for mobile robot navigation using machine learning: A survey. Autonomous Robots. 2022;46(5):569-597
[8] Zhou C, Huang B, Fränti P. A review of motion planning algorithms for intelligent robots. Journal of Intelligent Manufacturing. 2022;33(2):387-424
[9] Israr A, Ali ZA, Alkhammash EH, Jussila JJ. Optimization methods applied to motion planning of unmanned aerial vehicles: A review. Drones. 2022;6(5):126
[10] Antonyshyn L, Silveira J, Givigi S, Marshall J. Multiple mobile robot task and motion planning: A survey. ACM Computing Surveys. 2023;55(10):1-35
[11] Dong L, He Z, Song C, Sun C. A review of mobile robot motion planning methods: From classical motion planning workflows to reinforcement learning-based architectures. Journal of Systems Engineering and Electronics. 2023;34(2):439-459
[12] Ali, Z. A., Israr, A., & Hasan, R. (2023). Survey of methods applied in cooperative motion planning of multiple robots. Motion planning for dynamic agents InTechOpen. https://doi.org/10.5772/intechopen.1002428
[13] Lin S, Liu A, Wang J, Kong X. A review of path-planning approaches for multiple Mobile robots. Machines. 2022;10(9):773
[14] Lin C, Han G, Jiaxin D, Bi Y, Shu L, Fan K. A path planning scheme for AUV flock-based internet-of-underwaterthings systems to enable transparent and smart ocean. IEEE Internet of Things Journal. 2020;7(10):9760-9772
[15] Hu Y, Yao Y, Ren Q, Zhou X. 3D multi-UAV cooperative velocity-aware motion planning. Future Generation Computer Systems. 2020;102:762-774
[16] Chang H, Chen Y, Zhang B, Doermann D. Multi-UAV mobile edge computing and path planning platform based on reinforcement learning. IEEE Transactions on Emerging Topics in Computational Intelligence. 2021;6(3):489-498
[17] Yu Z, Si Z, Li X, Wang D, Song H. A novel hybrid particle swarm optimization algorithm for path planning of UAVs. IEEE Internet of Things Journal. 2022;9(22):22547-22558
[18] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959.
[19] Jiang S, Wang S, Yi Z, Zhang M, Lv X. Autonomous Navigation System of Greenhouse Mobile Robot Based on 3D Lidar and 2D Lidar SLAM. Front Plant Sci. 2022 Mar 10;13:815218. doi: 10.3389/fpls.2022.815218. PMID: 35360319; PMCID: PMC8960995.
[20] Ferguson, D.; Likhachev, M.; Stentz, A. A Guide to Heuristic-based Path Planning. In Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, USA, 5–10 June 2005; pp. 9–18.
[21] Gass, S.I.; Harris, C.M. Near-optimal solution. In Encyclopedia of Operations Research and Management Science; Gass, S.I., Harris, C.M., Eds.; Springer: New York, NY, USA, 2001; p. 555. [CrossRef]
[22] Hernández, C.; Baier, J.; Achá, R. Making A* run faster than D*-lite for path-planning in partially known terrain. In Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS, Portsmouth, NH, USA, 21–26 June 2014; Volume 2014, pp. 504–508.
[23] Seo, D. J., & Kim, J. (2013). Development of autonomous navigation system for an indoor service robot application. Paper presented at the 2013 13th International Conference on Control, Automation and Systems (ICCAS 2013).
[24] Stentz, A. Optimal and Efficient Path Planning for Unknown and Dynamic Environments; The Robotics Instilute Carnegie Mellon University: Pittsburge, PA, USA, 2003; Volume 10.
[25] Nilsson, N.J. Principles of Artificial Intelligence; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1980.

[26] Stentz, A. The D* Algorithm for Real-Time Planning of Optimal Traverses. 2011. Available online: https://www.ri.cmu.edu/ pub_files/pub3/stentz_anthony__tony__1994_2/stentz_anthony__tony__1994_2.pdf (accessed on 12 October 2023).
[27] Sasi Kumar, Gopikrishnan & Shravan, BVS & Gole, Harshit & Barve, P & Ravikumar, Lingam. (2011). "Path Planning Algorithms: A comparative study", National Conference on Space Transportation Systems (STS – 2011) At: Vikram Sarabhai Space Centre (VSSC), Thiruvananthapuram, India, December 2011.
[28] Zhuozhen Tang1 and Hongzhong Ma, An overview of path planning algorithms, IOP Conference Series: Earth and Environmental Science, Volume 804, 1. Fossil Energy & Geological Engineering, 2021. DOI 10.1088/1755-1315/804/2/022024
[29] LaValle, S. M. (2006). Planning algorithms: Cambridge university press
[30] Karova, M., Zhelyazkov, D., Todorova, M., Penev, I., Nikolov, V., & PETKOV, V. (2015), "Path planning algorithm for mobile robot", Paper presented at the Proc. Int. Conf. Appl. Computer Science, 2015.
[31] Xinyu, W., Xiaojuan, L., Yong, G., Jiadong, S., & Rui, W. (2019), "Bidirectional Potential Guided RRT* for Motion Planning", IEEE Access, 2019, 7, 95034-95045.
[32] Korkmaz, M., & Durdu, A. (2018), "Comparison of optimal path planning algorithms", Paper presented at the 2018 14th International Conference on Advanced Trends in Radio elecrtronics, Telecommunications and Computer Engineering (TCSET), 2018.
[33] DOSHI A, POSTULA A J, FLETCHER A, et al. Development of micro-UAV with integrated motion planning for open-cut mining surveillance [J]. Microprocessors and Microsystems, 2015, 33 (8) 6:829-835.
[34] Journal of Northwest University (Natural Science Edition), 2018, 48(05): 651-658. (in Chinese with English abstract)
[35] Shi K, Denny J, Amato N M. Spark PRM:Using RRTs within PRMs to efficiently explore narrow passages[C]//2014 IEEE International Conference on Robotics and Automation, ICRA 2014. Piscataway: IEEE, 2014:4659-4666.
[36] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.
[37] M. N. A. Wahab, S. Nefti-Meziani, and A. Atyabi, "A comparative review on mobile robot path planning: classical or meta-heuristic methods?" Annual Reviews in Control, vol. 50, pp. 233–252, 2020.
[38] B. K. Patle, G. Babu L, A. Pandey, D. R. Parhi, and A. Jagadeesh, "A review: on path planning strategies for navigation of mobile robot," Defence Technology, vol. 15, no. 4, pp. 582–606, 2019.
[39] M. J. Gilmartin, "Introduction to autonomous mobile robots roland siegwart and illah R. Nourbakhsh," Robotica, MIT Press, vol. 23, no. 2, pp. 271-272, 2005.
[40] Tomovi, Aleksandar. "Path Planning Algorithms For The Robot Operating System." (2014).