Optimal Path Planning for Autonomous Mobile Delivery Robots

Prasanna Marudhu Balasubramanian
Department of Robotics Engineering
University of Maryland
College Park, Maryland 20740, USA
prasbala@umd.edu

Chayan Kumar Patodi
Department of Robotics Engineering
University of Maryland
College Park, Maryland 20740, USA
ckp1804@umd.edu

Abstract*— Obtaining the path from initial position to final position for the robot which should be hurdle free even though the whole vicinity is filled with obstacles should be the talk of the moment while working on finding the way for the robot. The whole idea behind this project is planning the paths for autonomous bots which are the future, while using multiple algorithms that will give us the ideal path in the most advantageous time so that the robot can move around in the cluttered space. To execute motion, the steps of an algorithm can be summed up in two lines. First it will scan the map or retrieve a stored map and second it will create a path accordingly. In order to achieve the ideal path as output, various methods are created, modifying the architecture of the algorithm present to develop an idealised algorithm for the applications of such delivery robots. Results and conclusion from the researches done till date, prove that algorithms are the key to generate and produce an optimal path for motion and it also proves that certain algorithms are suitable enough, so as to use them in real time. This report shares some light on and gives a perspective of the pros and cons of multiple path planning algorithms existing in present time.

***Keywords*—optimal path, planning, mobile robots, A-star, Dijkstra, RRT, delivery robot, Breadth First Search, heuristic cost ,visibility graph, euclidian distance, obstacle avoidance.**

I. Introduction

Path planning is a very popular field in the world of robotics and it plays a very vital role in traversing the mobile robots autonomously between node points [1]. It is the major aspect in the autonomous mobile robotics field. It enables to find an ideal pathway for the bots to traverse from an initial point to final point in a map space having hittable objects. In this process of motion planning, it is quite often that ideal path is needed to be ideal and optimal [2]. Navigation is process of creating a path or route that a mobile robot must perform to move without colliding with obstacles from one position to the other position by not following the wrong path. The 3 major concern for the navigation includes localization, robot motion control and optimal path planning [5]. Among these 3 areas, path planning to reach the goal is most vital area of concern. It helps in finalising the best pathway for the robot to transport within the map space [6].

Optimal path planning guides the mobile mobile vehicle or robot to trace an ideal path not getting into any of the obstacle space in the map. Optimal path planner should have the ability to eliminate the uncertain situations while searching in the uncertain map to reduce robot impact on the object and for finding the ideal path in lesser duration typically when path need to be altered regularly [4]. Representing the Robot in the C-Space (configuration space) and the implementing the algorithm are the two major points while path planning is considered and they are related to each other and have bigger to obtain the ideal path in minimal time [3].

In the following sections, the idea behind the different algorithms taken into consideration is briefly presented, and then the method in which they are implemented is discussed in detail. Consecutively, some simulation results are shown, and then the efficiency of each method is discussed based on the experimental results for these algorithms.

The description about each of the following sections described in this paper are discussed in the following. The Section II – Path planning algorithms gives a detailed information about the type of algorithm taken into consideration in this project and how each algorithm works. The Section III – Methods describes how each algorithm is implemented in this project to enable the movement of the robot starting from the very first node to the end node by staying away the obstacles and hindrances in the map. Advantages of each algorithm over the other is described in this section. The section gives the basic idea behind the background of this project. The computational advantage and complexity between the algorithm are also mentioned in this section. The Section IV discusses about the previous researches and findings in the sphere of autonomous mobile robots and planning. The ideas as well as the concepts inferred from the previous research papers are described in this paper. The complexity and practical issues seen in implementing these algorithms and general mathematical approach for implementing these algorithms are observed from these previous research papers and described under this section. The outcome of each of these researches are also briefly discussed. The Section V describes about the simulation of each of these algorithms in the obstacle cluttered environment or space. The simulation results from each of the algorithm and the path planning response from them in the obstacle space are mentioned here. The simulation output results with respect to each of the algorithm can be visualized in this section. The Section VI compares the response from each of these algorithms and shows the tabulation consisting of details of all the algorithms including their Time of execution, Number of explored nodes, and more. The better algorithm amongst all the compared algorithms for the functionality of the robot is described in this section. Section VII shows the conclusion we have reached from this research after analysing all the aspects. Section VIII discusses the future scope for this approach.

The following assumptions are made in this project paper:

1. The obstacles are shape with nodes in set B. Let Vetex= B *U* [s,t] [10].
2. The robot is be observed as a point robot traversing among the obstacle space.
3. The obstacles move linearly with the constant speed.
4. The autonomous mobile robot has constant velocity and spontaneous deceleration and acceleration values.

II. PLAN OF ACTION

When it comes to the delivery robots apart from planning the path we need this process to happen in less time and this planned path should be an optimal path with less distance between its start and the goal point. Path planning methods noticed the complexity in computation. [6]. Basically, the short and ideal path between start and goal can be obtained through the Dijkstra algorithm, graph search, DFS algorithm, BFS algorithm, and A-Star algorithm and the other well-known algorithms including RRT, Depth First Search (DFS), RRT*, Visibility Graph and Weighted A-Star for delivery robot path planning application will be considered in this paper. From the comparison of the various factors including the algorithm with less time complexity, the one that gives the shortest path, and the one that is feasible based on the map the result will be summarized.

The list of algorithms taken for comparison of path planning for autonomous mobile robot are:

- A-Star
- Dijkstra
- RRT
- Weighted A*
- Breadth First Search (BFS)
- Depth First Search (DFS)
- RRT-Star (RRT*)
- Visibility Graph

The above-mentioned algorithms are compared based on the factors mentioned earlier and the optimal path planning algorithm is obtained.

III. METHODS

As mentioned above we are taking into consideration the above algorithms for finding an optimal path for our application and the methodology for implementing these algorithms are described below in detail,

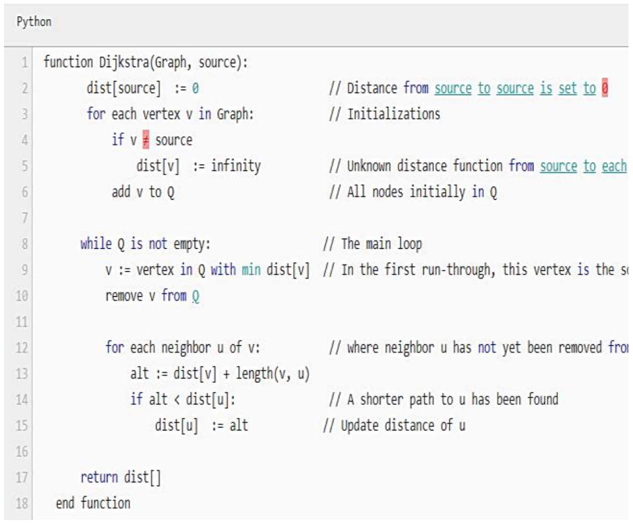
1) Dijkstra Algorithm:

Dijkstra helps us calculate the shortest path between nodes in the graph. The steps we follow for Dijkstra algorithm are as follows:

- We initialize the distances according to the general notion.
- Then we start with the start node and calculate distances to the adjacent nodes.
- We then pick the next node with minimal distance and we repeat the distance calculation for the adjacent node.
- If a new minimal distance is found, we update the distance to the value.
- This is repeated till the goal node is found.

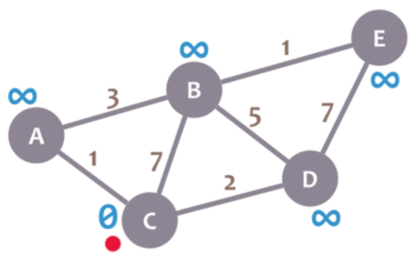
The above-mentioned steps are explained in the Pseudo code below.

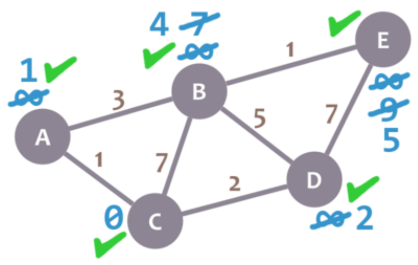
Pseudocode:


Python

```
function Dijkstra(Graph, source):
      dist[source]  := 0                    // Distance from source to source is set to 0
      for each vertex v in Graph:           // Initializations
          if v ≠ source
              dist[v]  := infinity          // Unknown distance function from source to each
          add v to Q                        // All nodes initially in Q

     while Q is not empty:                 // The main loop
         v := vertex in Q with min dist[v]  // In the first run-through, this vertex is the so
         remove v from Q

         for each neighbor u of v:          // where neighbor u has not yet been removed fro
             alt := dist[v] + length(v, u)
             if alt < dist[u]:              // A shorter path to u has been found
                 dist[u]  := alt           // Update distance of u

     return dist[]
  end function
```

Dijkstra steps [13]:



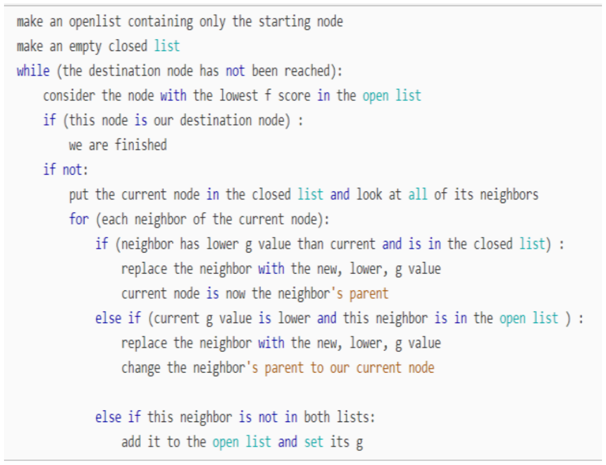


2) A-Star (A*) Algorithm:

This is the most commonly used algorithm in finding short path application because of its efficiency.

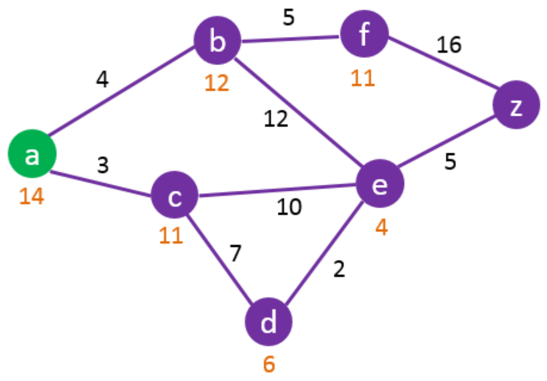
What makes A-star different from other path planning algorithms is the formulae: f = g + h. The f,g,h get calculated every time a new node is created , where f is the total cost of the node , g is the distance b/w current node and start node and H is heuristic distance (can be Manhattan or Euclidian) – estimated distance from the current node to the goal node.
It is an modified version of Dijkstra, it avoids exploring the non-promising nodes using the heuristic distance to the goal node.

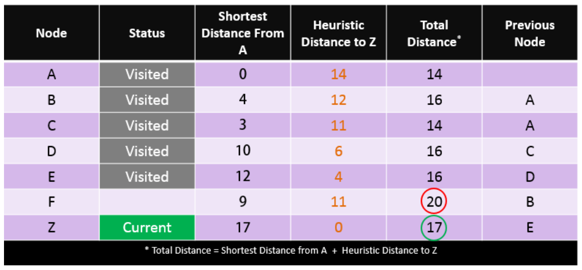
Psuedo Code:



```
make an openlist containing only the starting node
make an empty closed list
while (the destination node has not been reached):
    consider the node with the lowest f score in the open list
    if (this node is our destination node) :
        we are finished
    if not:
        put the current node in the closed list and look at all of its neighbors
        for (each neighbor of the current node):
            if (neighbor has lower g value than current and is in the closed list) :
                replace the neighbor with the new, lower, g value
                current node is now the neighbor's parent
            else if (current g value is lower and this neighbor is in the open list ) :
                replace the neighbor with the new, lower, g value
                change the neighbor's parent to our current node

            else if this neighbor is not in both lists:
                add it to the open list and set its g
```

A-Star Steps [13]:





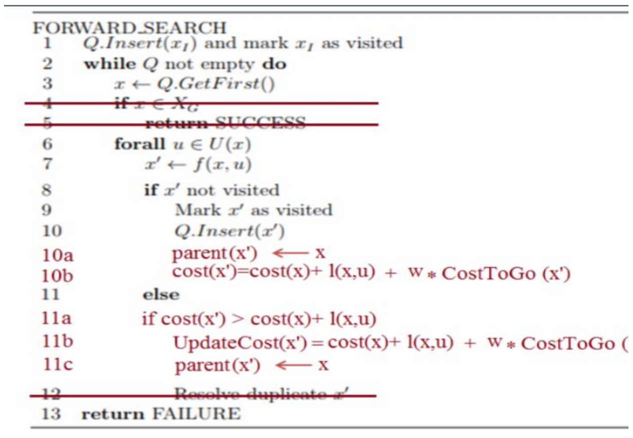
Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Visited	10	6	16	C
E	Visited	12	4	16	D
F		9	11	20	B
Z	Current	17	0	17	E

* Total Distance = Shortest Distance from A + Heuristic Distance to Z

3) Weighted A*:

Weighted A* is a more modified version of A* algorithm. The formula mentioned above gets updated to f = g+εh . Weighted A* search trades off optimality for speed. Parameter ε biases the search towards the states that are closer to the foal state.

Pseudocode Weighted A*[Lecture – Session 6]:

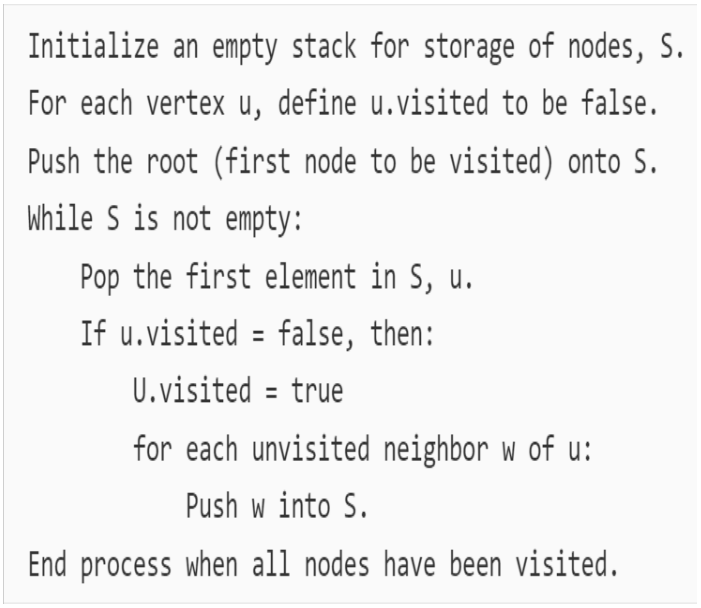


```
FORWARD_SEARCH
1    $Q.Insert(x_I)$ and mark $x_I$ as visited
2    while $Q$ not empty do
3        $x \leftarrow Q.GetFirst()$
~~4        if $x \in X_G$~~
~~5            return SUCCESS~~
6        forall $u \in U(x)$
7            $x' \leftarrow f(x,u)$
8            if $x'$ not visited
9                Mark $x'$ as visited
10               $Q.Insert(x')$
10a              parent(x') $\longleftarrow$ x
10b              cost(x')=cost(x)+ l(x,u) + w $*$ CostToGo (x')
11           else
11a          if cost(x') > cost(x)+ l(x,u)
11b              UpdateCost(x') = cost(x)+ l(x,u) + w $*$ CostToGo (
11c              parent(x') $\longleftarrow$ x
~~12             Resolve duplicate $x'$~~
13   return FAILURE
```

4) Depth-First Search:

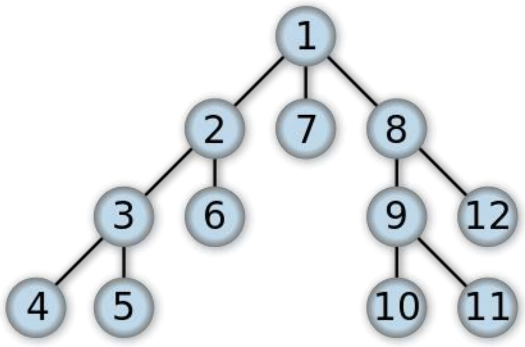
DFS uses backtracking. It is a recursive algorithm. In DFS, once we move forward and there is a dead end along the current path, we move backwards on to the same path to find more nodes to go through.

Psuedo Code :



```
Initialize an empty stack for storage of nodes, S.
For each vertex u, define u.visited to be false.
Push the root (first node to be visited) onto S.
While S is not empty:
    Pop the first element in S, u.
    If u.visited = false, then:
        U.visited = true
        for each unvisited neighbor w of u:
            Push w into S.
End process when all nodes have been visited.
```

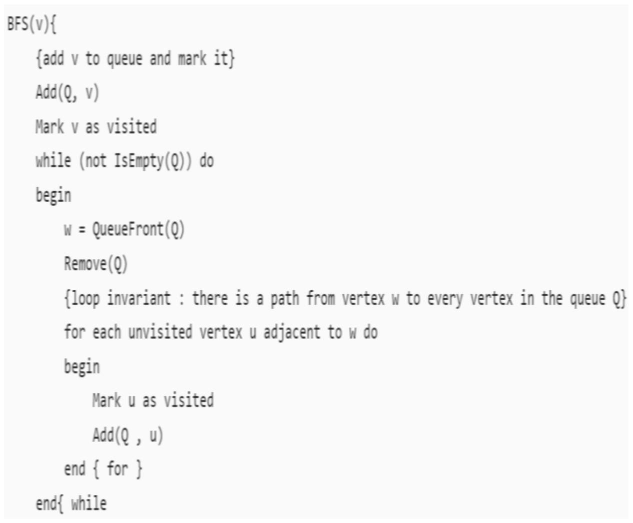
DFS Node Exploration[15]:



5) BFS (Breadth First Search):

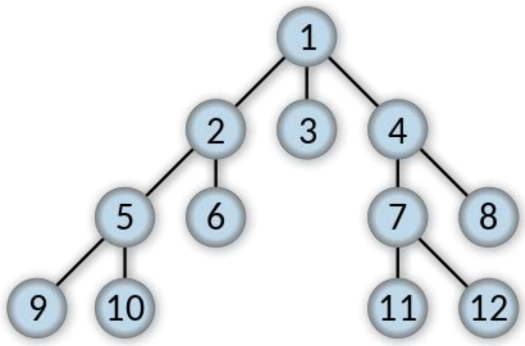
BFS algorithm is a traversing one and we start from the selected node and we explore the neighbour nodes which are directly connected to the source node. We then move forward to the next level neighbour nodes. We then move horizontally and visit all the nodes of the current layer.

Psuedo Code:



```
BFS(v){
    {add v to queue and mark it}
    Add(Q, v)
    Mark v as visited
    while (not IsEmpty(Q)) do
    begin
        w = QueueFront(Q)
        Remove(Q)
        {loop invariant : there is a path from vertex w to every vertex in the queue Q}
        for each unvisited vertex u adjacent to w do
        begin
            Mark u as visited
            Add(Q , u)
        end { for }
    end{ while
```

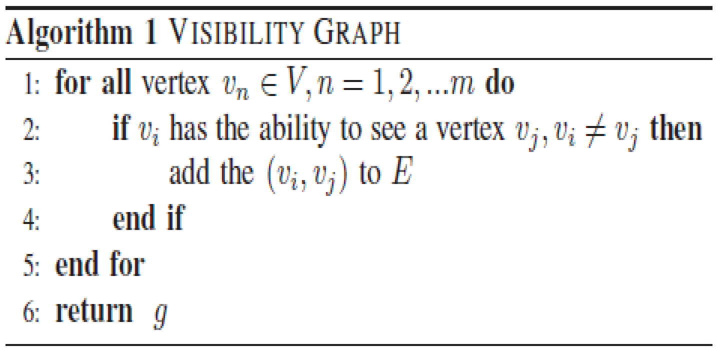
BFS Node Exploration [15]



6) Visibility Graph:

In Visibility Graph approach each node in the graph is a point location and each corresponding edge represents a visible bond in between them. The shortest path between two obstacles follows a straight line except at the vertices of the obstacle , where it may turn [12].

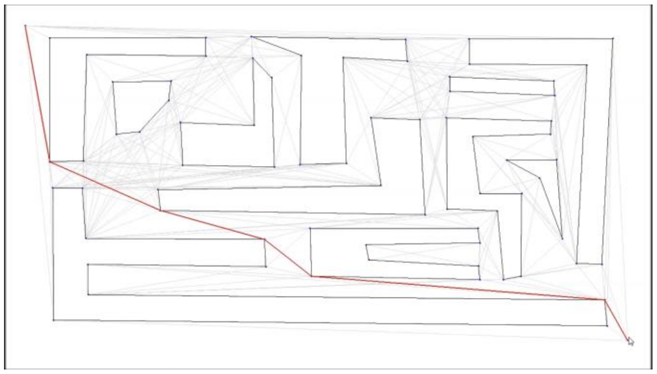
Psuedo Code:



Algorithm 1 VISIBILITY GRAPH

```
1: for all vertex $v_n \in V, n = 1, 2, ...m$ do
2:     if $v_i$ has the ability to see a vertex $v_j, v_i \neq v_j$ then
3:         add the $(v_i, v_j)$ to $E$
4:     end if
5: end for
6: return $g$
```

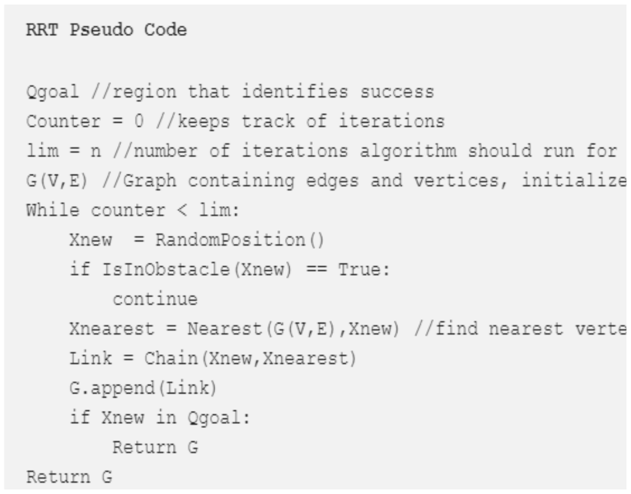
Visibility Graph Exploration:



7) RRT Algorithm:

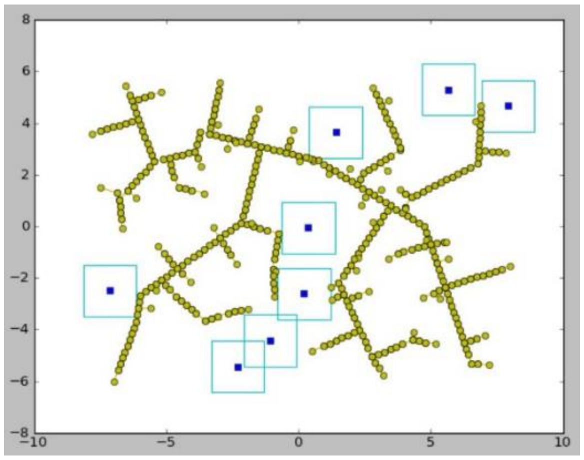
The premise of RRT is actually quite straight forward. Points are randomly generated and connected to the closest available node. Each time a vertex is created, a check must be made that the vertex lies outside of an obstacle. Furthermore, chaining the vertex to its closest neighbour must also avoid obstacles. The algorithm ends when a node is generated within the goal region, or a limit is hit [14].

Pseudocode:



```
RRT Pseudo Code

Qgoal //region that identifies success
Counter = 0 //keeps track of iterations
lim = n //number of iterations algorithm should run for
G(V,E) //Graph containing edges and vertices, initialize
While counter < lim:
    Xnew  = RandomPosition()
    if IsInObstacle(Xnew) == True:
        continue
    Xnearest = Nearest(G(V,E),Xnew) //find nearest verte
    Link = Chain(Xnew,Xnearest)
    G.append(Link)
    if Xnew in Qgoal:
        Return G
Return G
```

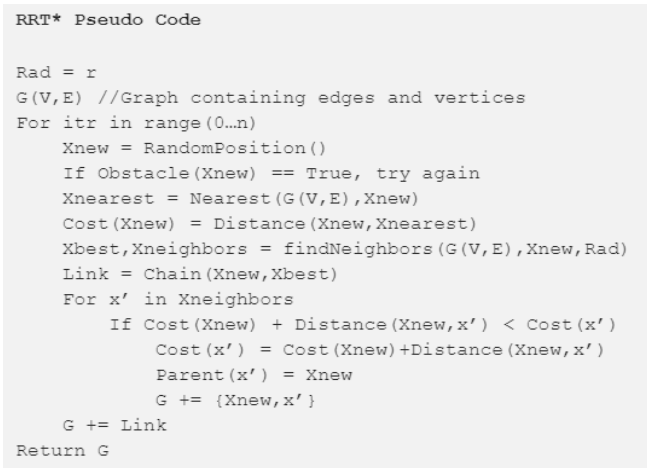
RRT Node exploration[14]:



8) RRT* (RRT-Star) Algorithm:

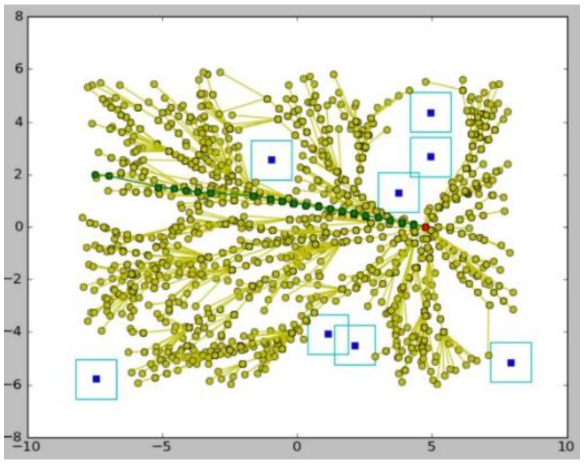
RRT* is an optimized version of RRT. RRT* records the distance each vertex has travelled relative to its parent vertex. This is referred to as the ‘cost’ of the vertex. After the closest node is found in the graph, a neighbourhood of vertices in a fixed radius from the new node are examined. If a node with a cheaper ‘cost’ than the proximal node is found, the cheaper node replaces the proximal node.
After a vertex has been connected to the cheapest neighbour, the neighbours are again examined. Neighbours are checked if being rewired to the newly added vertex will make their cost decrease. If the cost does indeed decrease, the neighbour is rewired to the newly added vertex [14].

Pseudocode:



```
RRT* Pseudo Code

Rad = r
G(V,E) //Graph containing edges and vertices
For itr in range(0…n)
    Xnew = RandomPosition()
    If Obstacle(Xnew) == True, try again
    Xnearest = Nearest(G(V,E),Xnew)
    Cost(Xnew) = Distance(Xnew,Xnearest)
    Xbest,Xneighbors = findNeighbors(G(V,E),Xnew,Rad)
    Link = Chain(Xnew,Xbest)
    For x’ in Xneighbors
        If Cost(Xnew) + Distance(Xnew,x’) < Cost(x’)
            Cost(x’) = Cost(Xnew)+Distance(Xnew,x’)
            Parent(x’) = Xnew
            G += {Xnew,x’}
    G += Link
Return G
```

RRT* Node Exploration[14]:



The above mentioned algorithms will be implemented* and they will be compared based on the simulation response. The factors which are taken into consideration for comparing these algorithms are Minimal Time, Shortest Path, Number of node created, Cost and feasibility.

Simulation Criteria for checking algorithms:

1. The Algorithm checked for one start point and one goal point with different radius of robot.
2. Also check for One start point and Multiple Goal points (Since for the delivery robot there will be multiple goal/delivery locations)
3. Check by varying the start and goal points
4. Check for farther goal node points

IV. PREVIOUS RESEARCH

There are wide range of researches that were carried out in the path planning algorithm and among which the researches which were carried out related to finding the time efficient algorithm, optimal path resulting algorithm, smoother path based algorithm, application based efficient algorithm and algorithm which is ideal for reaching the multiple goal points are observed for comparing the optimal path planning algorithm for delivery robots. Some of the research papers considered and gone through in doing this project and their drawbacks comparing to out intiative are mentioned below,

1) Previous researchers including, *Donald, Brooks and Kambhampti* [8,9] were working on a way to find a path with best time by modifying a suitable approach on mapping the workspaces area with an efficient pictuarisation of free space and obstacles region and also implement an appropriate algorithm to look for and modify those data structures by using only the graph search algorithm and Dijkstra algorithm. .

2) The research done by '*S. M. Masudur Rahman Al-Arif'* on the topic *"Comparative Study of Different Path Planning Algorithms: A Water based Rescue System"* compares the A*, Dijkstra and Graph Search algorithms for obtaining the optimal path and they are not considering the optimal time [11].

3) The research by '*Felipe Haro and Miguel Torres*' on the topic '*The Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Environments'* compare three path planning methods for omni-directional robots, which are based on a) the Bug algorithm b) the Potential Fields algorithm, and c) the A* algorithm for finding ideal path and not taking consideration the other existing algorithms like Dijkstra [6].

4) The research by *'Emili Hernandes'* on the topic '*A comparison of homotopic path planning algorithms for robotic application'* compares algorithms RRT and A* for underwater navigation by autonomous vehicles.

5) '*Milena Karova'* done a research on the topic '*Path planning algorithms for an autonomous mobile robot'* comparing only the D* and A* path planning algorithms.

6) '*Pradeep Jayabala'* researched on '*Path planning of autonomous mobile robots: A survey and comparison'* comparing the Graph Search, Potential Field and Vector Field algorithm.

There are still more previous researches that we have analysed and not all are mentioned here.

There is still various types of algorithms to search and find the optimal path between nodes and not all have been discussed and analysed in the previous researches. Previous researches comes to a conclusion that different algorithm type may complete the same task in less or more time, space, effort etc.
Each type of algorithm has its own speciality , pros and cons and for that reason our outputs are compared and discussed in detail in Section V

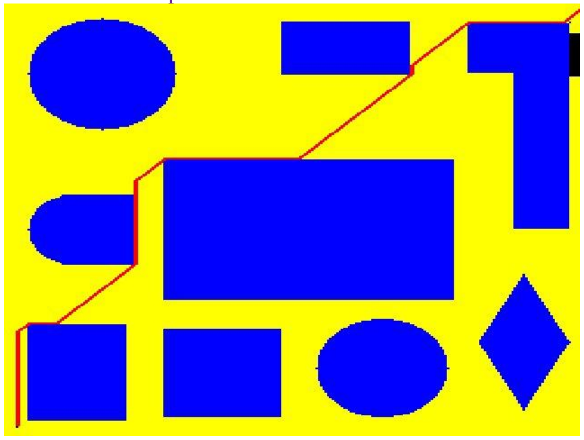
V. SIMULATION OUTPUTS

The Algorithms are executed and simulation results are obtained for different start and goal points for each algorithm. The simulation results are shown below for all the algorithms with description.

1) BFS Simulation:

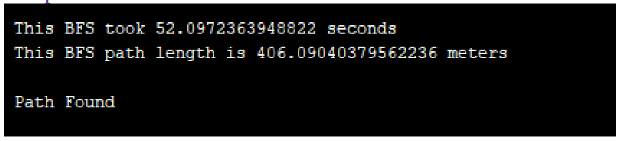
Points 1: Start Point (5,5) , Goal Point (249, 246)

Simulation Output



The Time for obtaining the path from BFS algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

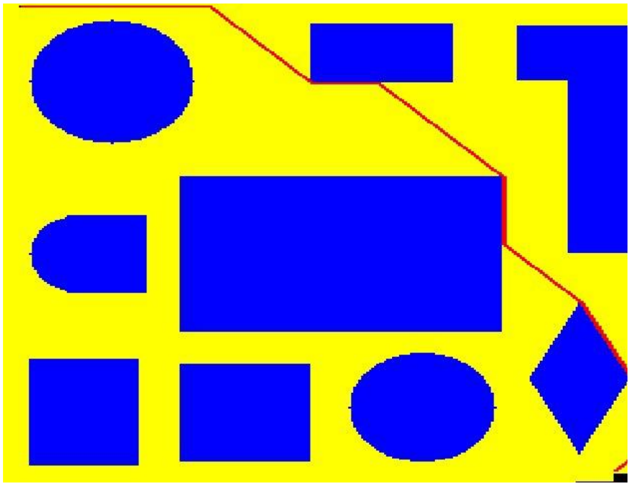
Output Console:



```
This BFS took 52.0972363948822 seconds
This BFS path length is 406.09040379562236 meters

Path Found
```

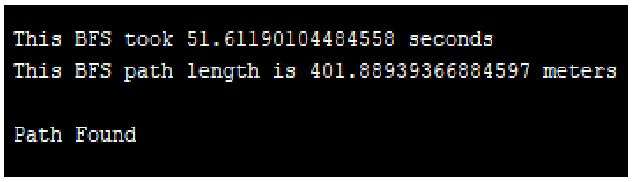
Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output



The Time for obtaining the path from BFS algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

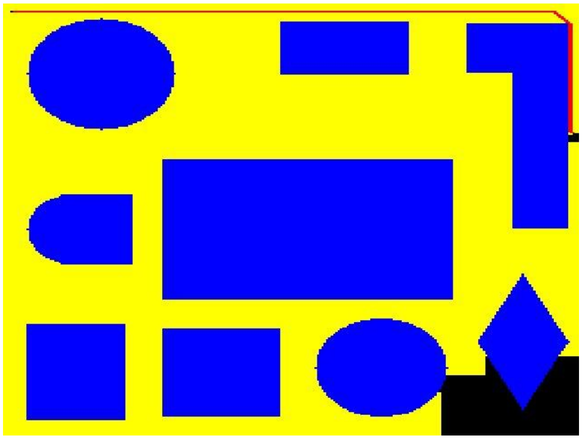
Output Console:



```
This BFS took 51.61190104484558 seconds
This BFS path length is 401.88939366884597 meters

Path Found
```

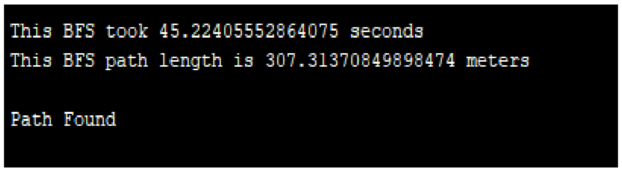
Points 3: Start Point (3,249) , Goal Point (246, 175)

Simulation Output



The Time for obtaining the path from BFS algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:

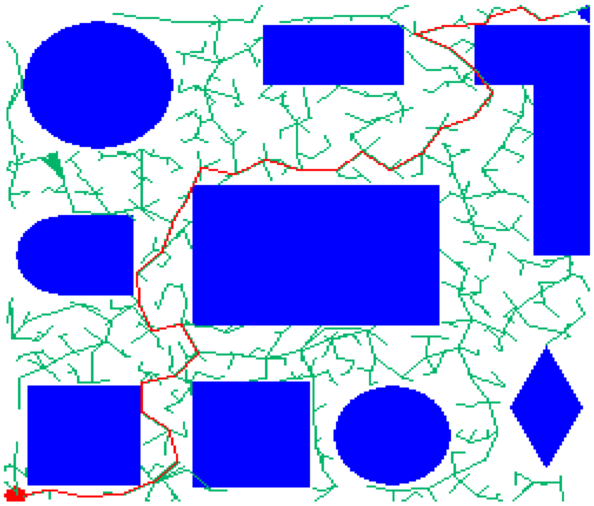


```
This BFS took 45.22405552864075 seconds
This BFS path length is 307.31370849898474 meters

Path Found
```

2) RRT Simulation:

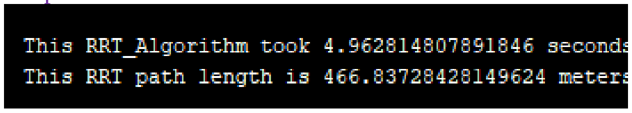
Points 1: Start Point (5,5) , Goal Point (249, 246)

Simulation Output



The Time for obtaining the path from RRT algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

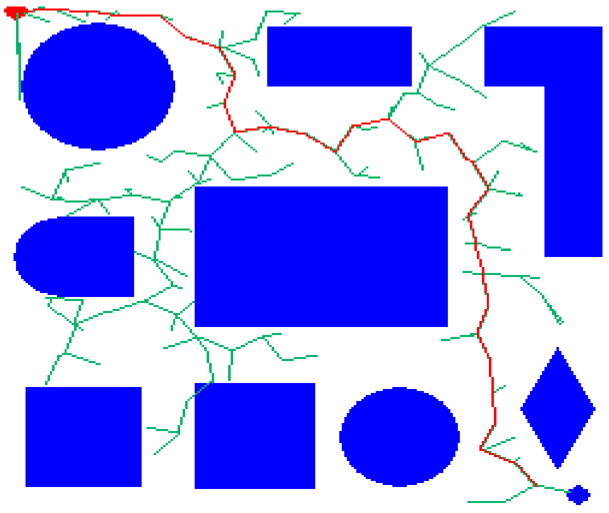
Output Console:



```
This RRT_Algorithm took 4.962814807891846 seconds
This RRT path length is 466.83728428149624 meters
```

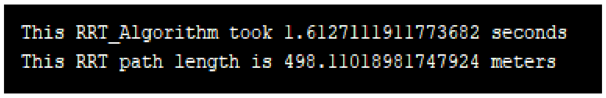
Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output



The Time for obtaining the path from RRT algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

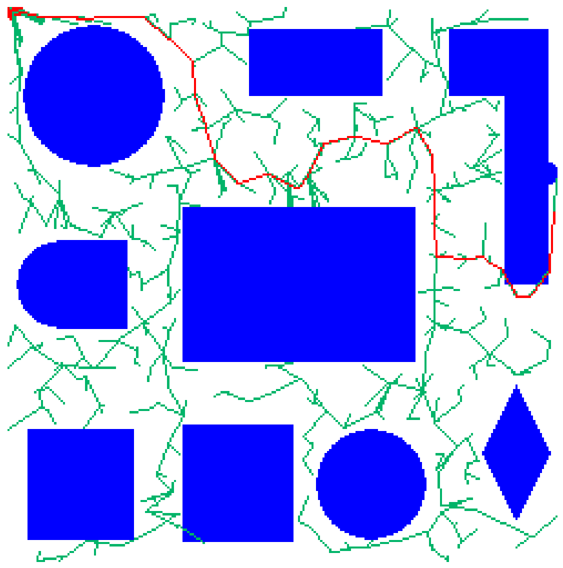
Output Console:



```
This RRT_Algorithm took 1.6127111911773682 seconds
This RRT path length is 498.11018981747924 meters
```

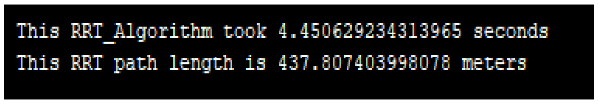
Points 3: Start Point (3,249) , Goal Point (246, 175)

Simulation Output



The Time for obtaining the path from RRT algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:

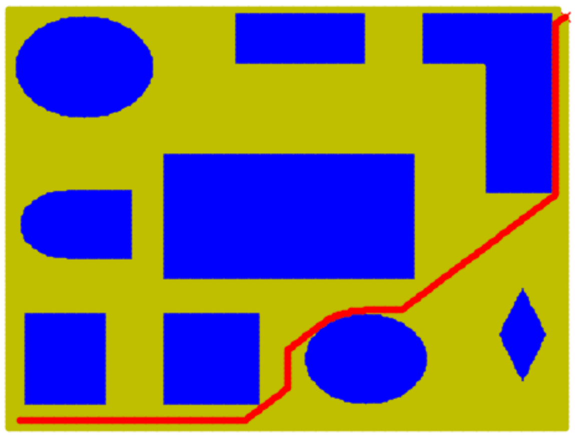


```
This RRT_Algorithm took 4.450629234313965 seconds
This RRT path length is 437.807403998078 meters
```

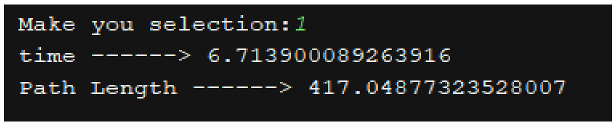
3) Dijkstra Simulation:

Points 1: Start Point (5,5) , Goal Point (249, 246)

Simulation Output



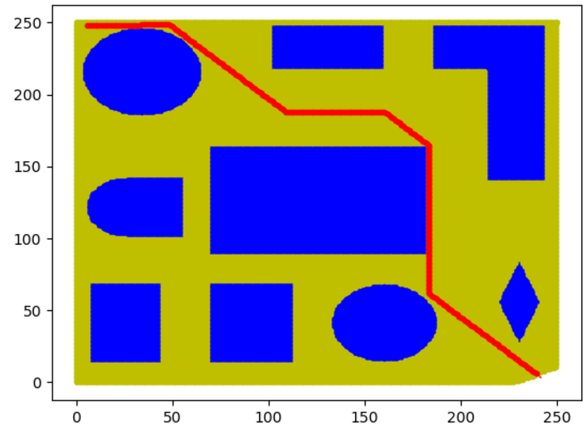
The Time for obtaining the path from Dijkstra algorithm and path distance between the points considered can be seen in the python console as shown below in the image.



```
Make you selection:1
time ------> 6.713900089263916
Path Length ------> 417.04877323528007
```

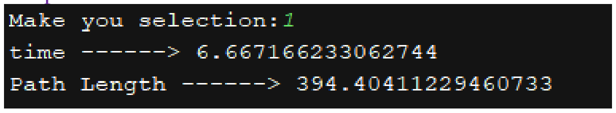
Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output



The Time for obtaining the path from Dijkstra algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

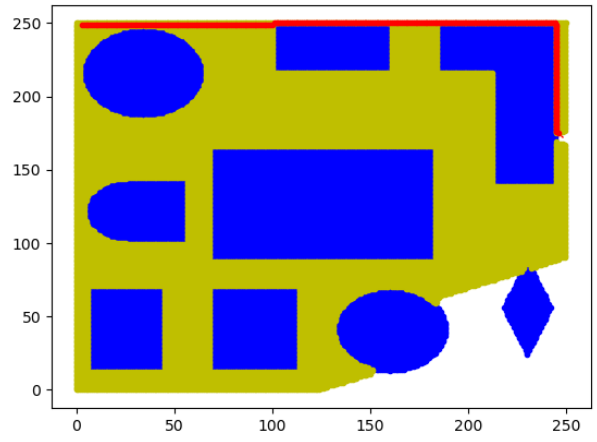
Output Console:



```
Make you selection:1
time ------> 6.667166233062744
Path Length ------> 394.40411229460733
```

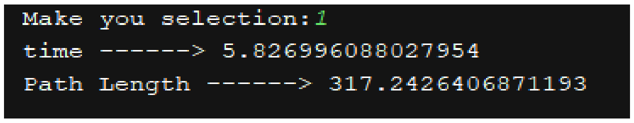
Points 3: Start Point (3,249) , Goal Point (246, 175)

Simulation Output



The Time for obtaining the path from Dijkstra algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:

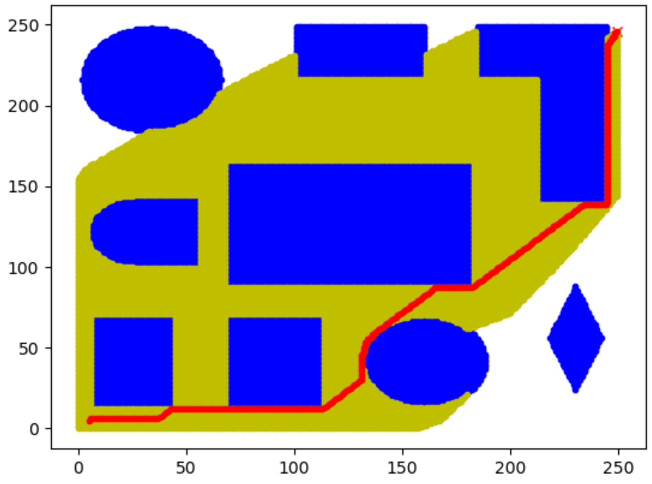


```
Make you selection:1
time ------> 5.826996088027954
Path Length ------> 317.2426406871193
```

4) A-Star Algorithm:

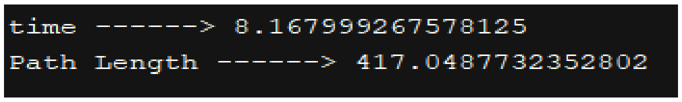
Points 1: Start Point (5,5) , Goal Point (249, 246)

Simulation Output



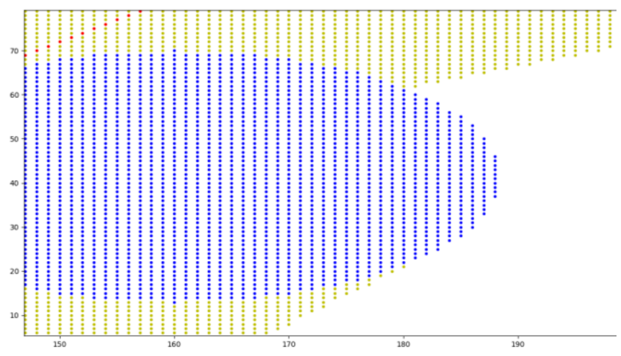
The Time for obtaining the path from A-Star algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:



```
time ------> 8.167999267578125
Path Length ------> 417.0487732352802
```

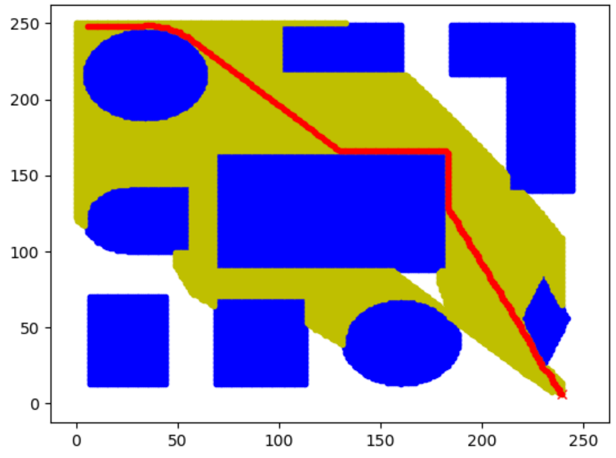
Note: It looks like the node exploration is happening inside the obstacles , but that is not true. The reason it looks like that in the above figure , is because of the way we are plotting it and also scaling issues.



The above figure is the magnified version of node exploration and it shows that no nodes are formed inside the obstacle space.

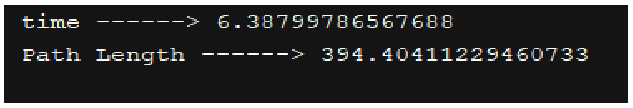
Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output



The Time for obtaining the path from A-Star algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

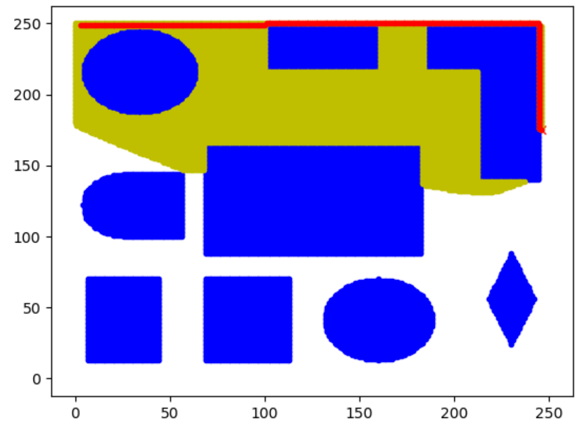
Output Console:



```
time ------> 6.387997865676 88
Path Length ------> 394.40411229460733
```

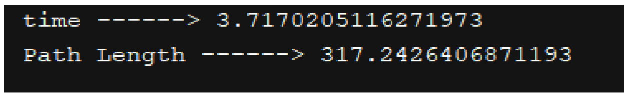
Points 3: Start Point (3,249) , Goal Point (246, 175)

Simulation Output



The Time for obtaining the path from A-Star algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:

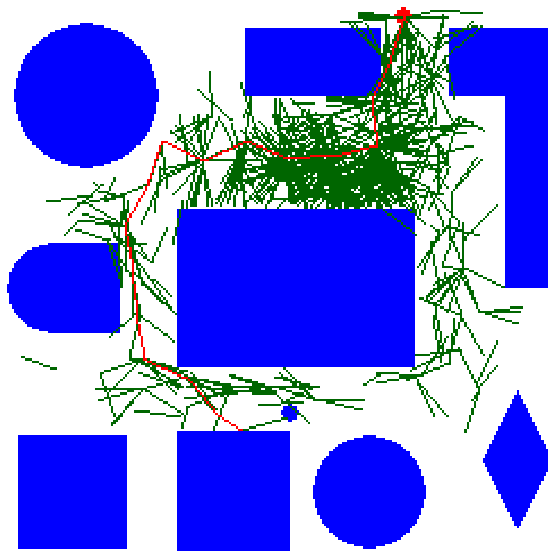


```
time ------> 3.7170205116271973
Path Length ------> 317.2426406871193
```

5) RRT_Star (RRT*) Simulation:

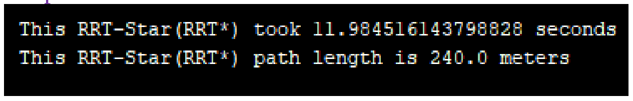
Points 1: Start Point (65,60) , Goal Point (225, 246)

Simulation Output



The Time for obtaining the path from RRT* algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

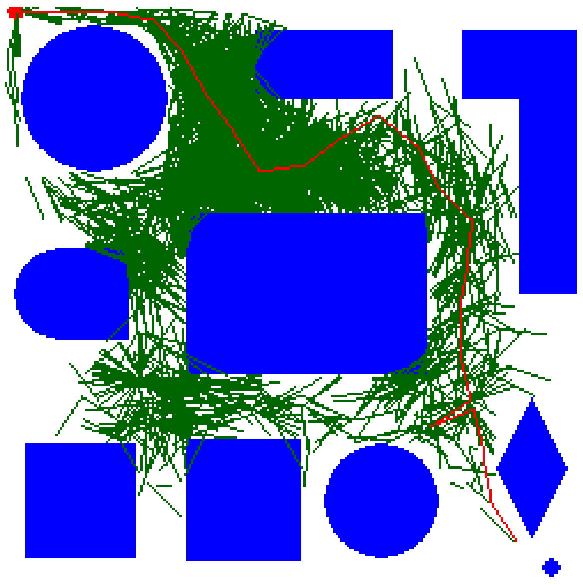
Output Console:



```
This RRT-Star(RRT*) took 11.984516143798828 seconds
This RRT-Star(RRT*) path length is 240.0 meters
```

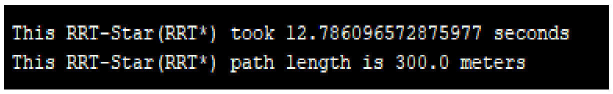
Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output



The Time for obtaining the path from RRT* algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

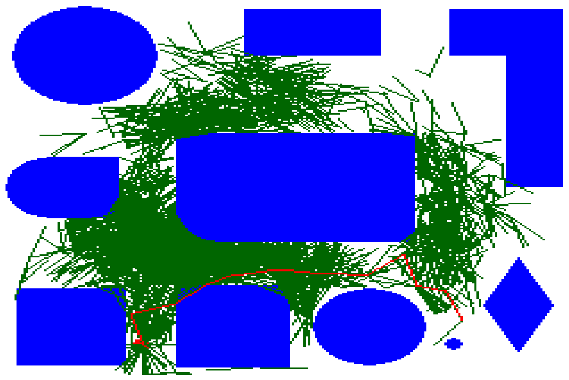
Output Console:



```
This RRT-Star(RRT*) took 12.786096572875977 seconds
This RRT-Star(RRT*) path length is 300.0 meters
```

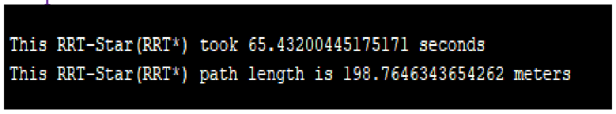
Points 3: Start Point (60,30) , Goal Point (230, 32)

Simulation Output



The Time for obtaining the path from RRT* algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:



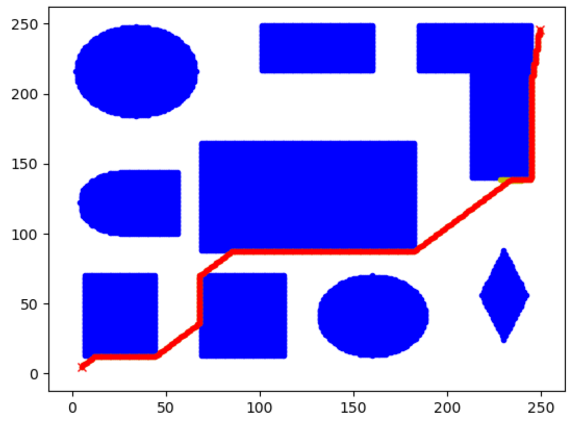
```
This RRT-Star(RRT*) took 65.43200445175171 seconds
This RRT-Star(RRT*) path length is 198.7646343654262 meters
```

6) Weighted A-Star Algorithm: (Weight =4)

In weighted A-star simulation even when the node exploration occurs, it is not visible in the output window as can be seen in the below figures. This happens since we are plotting 1000 nodes in a single iteration and since the nodes explored in case of 1st and 2nd points are less than 1000, the node exploration is not visible in the GUI.

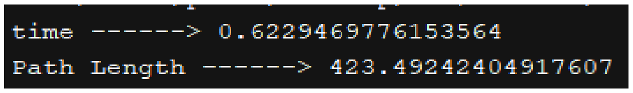
Points 1: Start Point (5,5) , Goal Point (249, 246)

Simulation Output



The Time for obtaining the path from Weighted A-Star algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

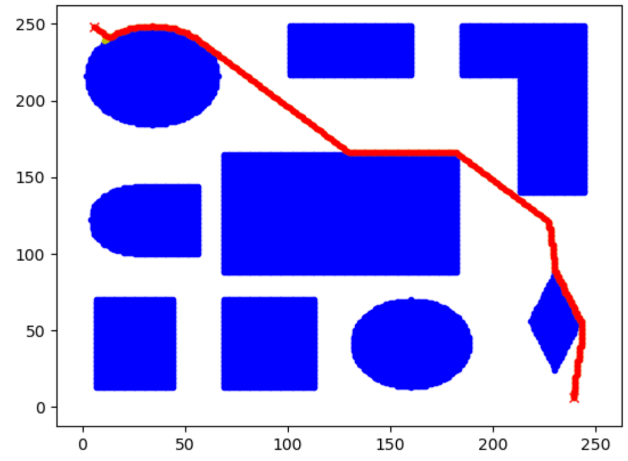
Output Console:



```
time ------> 0.6229469776153564
Path Length ------> 423.49242404917607
```

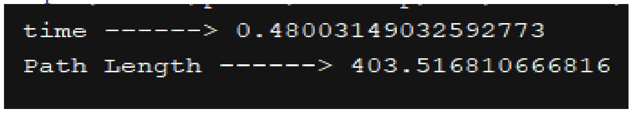
Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output



The Time for obtaining the path from Weighted A-Star algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

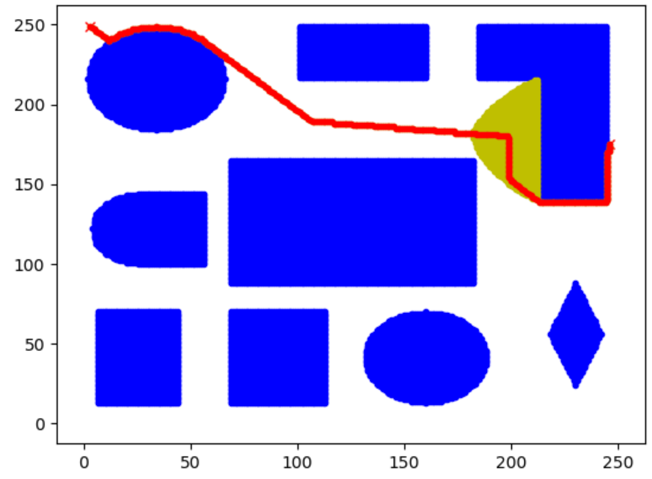
Output Console:



```
time ------> 0.4800314903259277З
Path Length ------> 403.516810666816
```

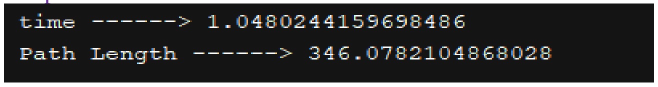
Points 3: Start Point (3,249) , Goal Point (246, 175)

Simulation Output



The Time for obtaining the path from Weighted A-Star algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:



```
time ------> 1.0480244159698486
Path Length ------> 346.0782104868028
```

7) Visibility Graph:

Points 1: Start Point (5,5) , Goal Point (249, 246)

Simulation Output

Figure with Visible Edges and Path

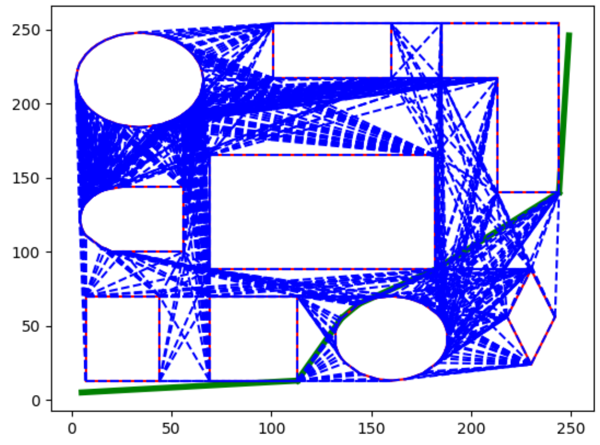
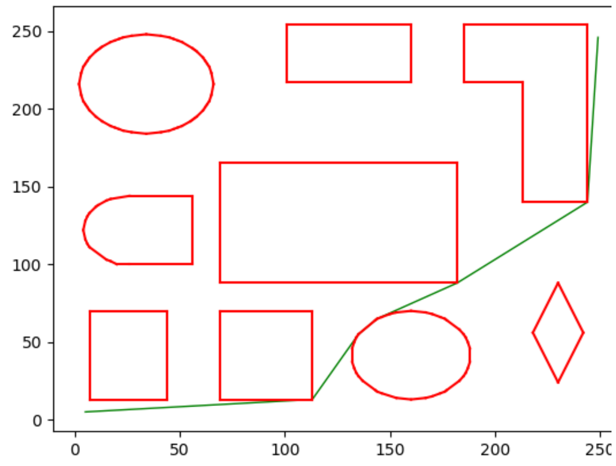
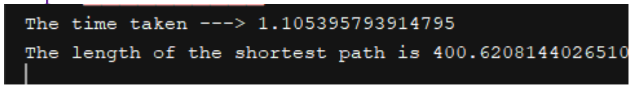


Figure with only Path



The Time for obtaining the path from Visibility Graph algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:



```
The time taken ---> 1.105395793914795
The length of the shortest path is 400.6208144026510
```

Points 2: Start Point (6,248) , Goal Point (239, 6)

Simulation Output

Figure with Visible Edges and Path

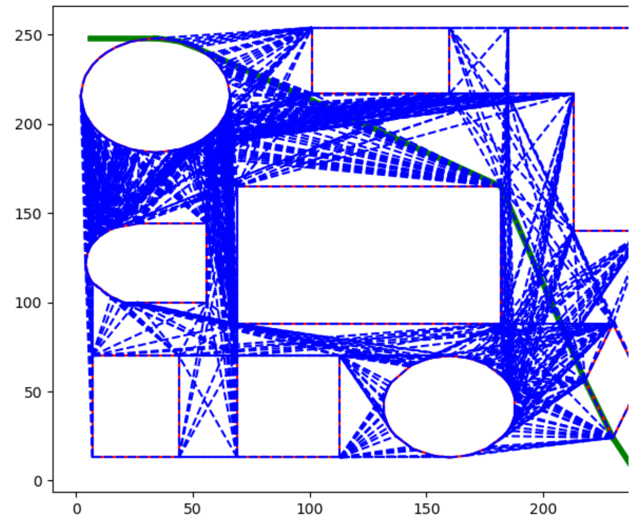
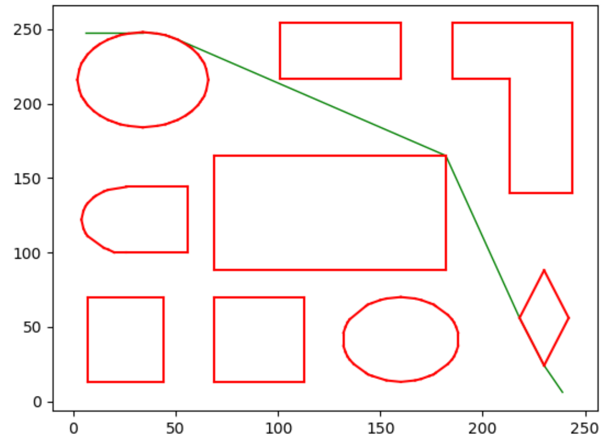
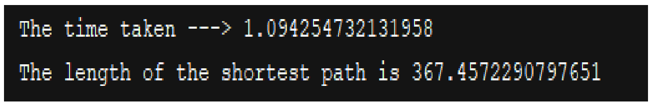


Figure with only Path



The Time for obtaining the path from Visibility Graph algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:



```
The time taken ---> 1.094254732131958
The length of the shortest path is 367.4572290797651
```

Points 3: Start Point (3,249) , Goal Point (246, 175)

Simulation Output

Figure with Visible Edges and Path

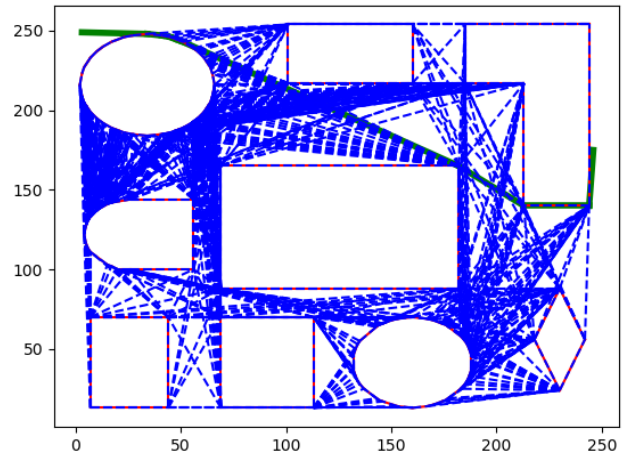
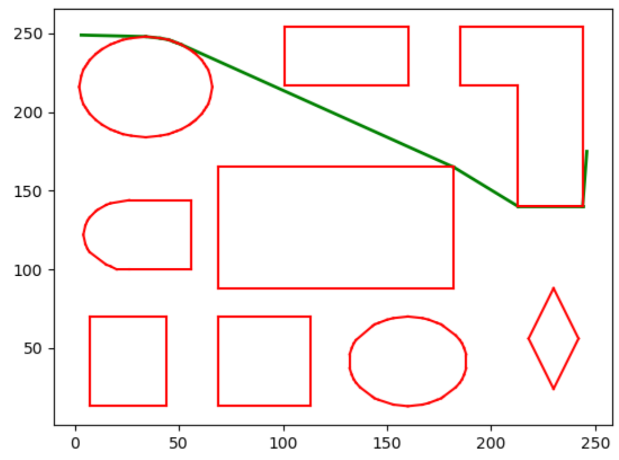
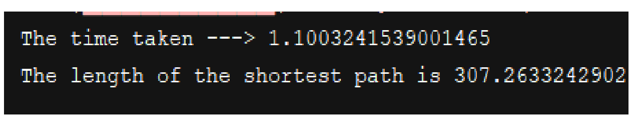


Figure with only Path



The Time for obtaining the path from Visibility Graph algorithm and path distance between the points considered can be seen in the python console as shown below in the image.

Output Console:



```
The time taken ---> 1.1003241539001465
The length of the shortest path is 307.2633242902
```

VI. RESULT COMPARISION

1) BFS r:

Start Points	Goal Points	BFS_Time(sec)	BFS_Distance (m)
5,5	249,246	52.0972	406.0904
6,248	239,6	51.6119	401.8893
55,24	202,24	8.5002	157.1837
79,228	244,70	31.3172	252.87
122,26	97,249	49.2647	276.296
57,35	208,209	48.4822	288.267
241,20	6,157	49.1833	323.2497
246,213	64,120	29.6566	278.5512
3,249	246,175	45.224	307.3137
180,245	130,45	35.2217	250.8822

VI. CONCLUSION

When the simulation results for all the algorithm taken into consideration are obtained, they will be analyzed and compared on the basis of the time and other factors which are mentioned earlier and the final outcome will be mentioned here.

VII. REFERENCES / BIBLIOGRAPHY

[1] N. Sariff and N. Buniyamin, "An Overview of Autonomous Mobile Robot Path Planning Algorithms", 4th Student Conference on Research and Development (Scored 2006), June 2006.

[2] Jagadish Chandra Mohanta , Dayal Ramakrushna Parhi , Saroj Kumar Patel , "Path planning strategy for autonomous mobile robot navigation using Petri-GA optimisation", Computers and Electrical Engineering, vol. 37, pp. 1058–1070, 2011.

[3] Dominik Henrich, Christian Wurll and Heinz Worn, "On-line path planning by heuristic hierarchical search", IECON '98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society, vol. 10,pp.1109, August 1998.

[4] Koichi Kondo," Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration," IEEE Transactions on Robotics and Automation, vol. 7, no. 3, June 1991.

[5] William Rone and Pinhas Ben-Tzvi," Mapping, localization and motion planning in mobile multi-robotic systems." Robotica (2013) volume 31, pp. 1–23, 2013.

[6] Haro, F.; Torres, M., "A Comparison of Path Planning Algorithms for Omnidirectional Robots in Dynamic Environments," Robotics Symposium, 2006. LARS '06. IEEE 3rd Latin American, vol., no., pp. 18, 25, 26-27 Oct. 2006.

[7] Anthony (Tony) Stentz." Optimal and Efficient Path Planning for Partially-Known Environments," Conference Paper, Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 4, pp. 3310 - 3317, May, 1994.

[8] Rodney A. Brooks," Solving the Find-Path Problem by Good Representation of Free Space," IEEE Transaction on systems, Man and Cybernetics, Vol. SMC-13, NO. 3, March/April 1983.

[9] SUBBARAO KAMBHAMPATI AND LARRY. DAVIS," Multiresolution Path Planning for Mobile Robots," IEEE JOURNAL OF ROBOTICS AND AUTOMATION, VOL. RA-2, NO. 3, SEPTEMBER 1986.

[10] Christos Alexopoulos and Paul M. Griffin," Path Planning for a Mobile Robot," IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. 22, NO. 2, MARCH/APRIL 1992.

[11] S. M. Masudur Rahman Al-Arif, A. H. M. Iftekharul Ferdous, Sohrab Hassan Nijami," Comparative Study of Different Path Planning Algorithms: A Water based Rescue System", International Journal of Computer Applications, Volume 39– No.5, February 2012 .

[12] J. H. Zhou and H. Y. Lin, "A Self-Localization and Path Planning Technique for Mobile Robot Navigation," 2011 World Congress on Intelligent Control and Automation (WCICA 2011), Taipei, Taiwan, June 21-25, 2011.

[13] https://www.brilliant.org

[14] https://medium.com/@theclassytim/robotic-path-planning-rrt-and-rrt-212319121378

[15] https://en.wikipedia.org