# A Comparative Study of Planning Algorithms for Robot in a Dynamic Maze Environment

Ghanem Jamal Eddine
Robotics Engineering,
A. James Clark School of Engineering
University of Maryland,
College Park, USA
gjamaled@umd.edu

Sneha Nayak
Robotics Engineering,
A. James Clark School of Engineering
University of Maryland,
College Park, USA
snehanyk@terpmail.umd.edu

Shreya Gummadi
Robotics Engineering,
A. James Clark School of Engineering
University of Maryland,
College Park, USA
shreyag@terpmail.umd.edu

*Abstract*— **A maze is a complex, compound form of obstacle space. This study aims to simulate a real-life maze-like hazardous environment unreachable by humans but accessible by robots. The study will approach this problem in three different, state-of-the-art algorithms: A*, Rapidly exploring Random Tree Star (RRT*), and Probabilistic Road Map (PRM). The search algorithms will be programmed and established in both MATLAB and Python. It is expected to get much more improved and tuned results than by using search algorithms like Breadth First Search (BFS) and Depth First Search (DFS).**

***Keywords— A*, RRT*, PRM, Heuristic, Optimal Path, Sampling-based Planning, Random Obstacles, V-REP, Simulation.***

## I. Introduction

Autonomous robots are a growing advancement in the field of tracking technology, especially applications in inspection, manipulation, and mapping. Home and office environments are typical places for autonomous robots, and they are especially necessary in hazardous field environments that are not suitable or even accessible by humans. This paper will analyze the replication of such environments with a defined obstacle space that will replicate a typical hazardous environment. The robot will have to navigate from the start point till the end point while avoiding any obstacle it might face in order to avoid collisions. The robot is assumed to move in an 8-connected space and at a constant speed.

The environment is replicated by defining a maze. A maze is a complex, compound form of obstacle space. Mazes have been studied rigorously in planning applications as they have many applications in the real world. A maze can simulate a hazardous environment. It can be used to simulate the parking space for an autonomous vehicle [12].

Our aim is to implement this project using three different search algorithms: A*, RRT*, and PRM. These advanced algorithms are better suited for robot mapping in hazardous environments especially because it is expected from the robot to execute and run as fast and as accurate as possible. Thus, making these algorithms superior to blind search like Breadth-First Search and Depth-First Search and uniform cost search like Dijkstra where processing time continues until all the nodes have been popped from the priority que, meaning shortest paths to every node have been determined.

A* will be implemented using MATLAB and RRT* along with PRM will be implemented using Python to show differences in compiling time and run time across the two platforms.

## II. Previous Studies

The paper "Mobile Robot Remote Path Planning and Motion Control in a Maze Environment" [3] has defined a similar maze for the mobile robot in a maze environment problem.

The study implements blind algorithms like BFS and DFS, where the heuristic is not considered. In the paper, the control system of the robot is designed thoroughly and uses the "Xbee wireless Communication Modules" which provides communication between the PC-based controller and the mobile robot [3].

Another paper, "Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game" [4] implements the aforementioned algorithms in a simple Tetris-like game that allows the user to input a start and goal node as well as predefined obstacle shapes in the obstacle space. The paper tests all three algorithms to find path length, time it took to reach goal point, and number of computed nodes. According to the paper [4], all three algorithms had the same length of 38 nodes. BFS was the slowest, with a time of 0.8 ms, while Dijkstra was the fastest, with a time of 0.3 ms which is 0.05ms faster than A*. However, A* computed much less nodes, reaching 323 nodes, while both Dijkstra and BFS computed 738 nodes [4].

The paper "Probabilistic roadmaps for path planning in high-dimensional configuration spaces."[5], gives the basic algorithm for the PRM algorithm and implements it on a 7 revolute joint with fixed base in an obstacle environment. The paper implements a customized local planner for path planning and compares it to the PRM Kavraki et al. It compares the number of nodes expanded, collision checks, time taken for the construction of roadmap and the success rate. The paper further explores a 5 revolute joint free base robot and a 4 degree of freedom robot. The paper concludes that PRM can solve certain problems that are beyond the capabilities of other existing methods.

The paper "Sampling-based algorithms for optimal motion planning" [9], talks about rigorously analyzing the asymptotic behavior of the cost of the solution returned by stochastic sampling-based algorithms as the number of samples increases. It mentions the drawbacks of using sampling-based algorithms which yield non optimal cost values. To overcome this issue, the paper introduces RRT* and PRM*, which has proven to be asymptotically optimal [9], i.e., such that the cost of the returned solution converges almost surely to the optimum.

The paper Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions", gives an overview of RRT* in recent application and about its importance in the future. It also talks about variations in RRT* to overcome the drawbacks of the traditionally used RRT* algorithm. This paper addresses the issue of [10] optimal path planning for mobile robots using RRT* sampling based. The selection of a particular variation of RRT* [10] is based on the similar concepts such as type of environment information available, the structure of the tree and the constraints managed by approach.

## III. Objective And Assumtions

The objective of the project is to generate an optimal path given the map of the maze and the start and end points. It has to ensure safe planning in order o avoid obstacles i.e. dead ends in the maze and random obstacles in between. Random Obstacles have been introduced in order to replicate the occurrence of dynamic obstacles [6][7] in real world. The three algorithms are to be then compared based on their performance, ability to handle randomness of obstacles and computation time.

In order to do so, certain assumptions have been made. The robot is assumed to move in an 8-connected space. The project assumes that the maze environment is always known. A common maze obstacle space is designed and the same map has been used in all of the algorithms for uniformity and fair comparison. The map has also been initialized as a configuration space that takes into account the radius of the robot and a reasonable clearance. Since the working robot for this paper is the Turtlebot2, the robot radius of 17.7 cm is set, while a reasonable clearance of 13 cm is selected. The map is shown in fig 1, and the configured map is shown in fig 2.

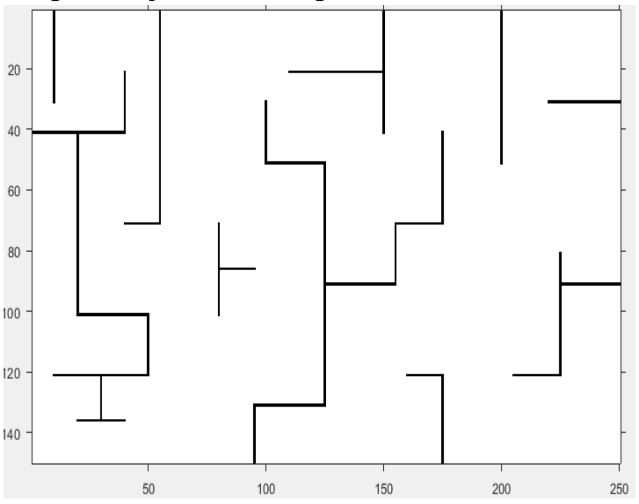


**Figure 1 Maze Environment.**

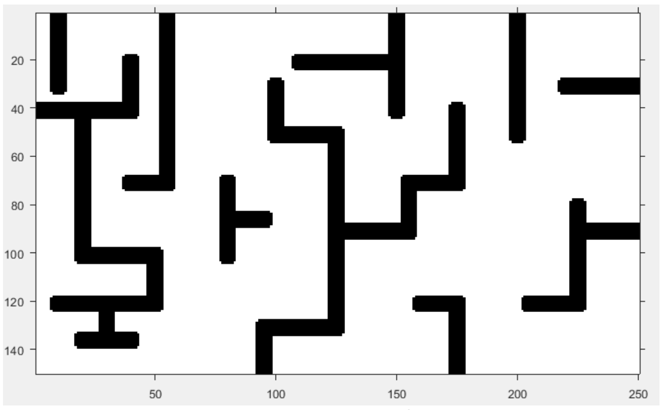


**Figure 2 Maze Environment Configuration Space**

The random obstacles as shown in fig 2, are generated at initialization and are known before the algorithm takes effect. Random obstacles simulate a real-world environment where the environment is unpredictable and dynamic, hence, the obstacle space changes each time the code is executed. The obstacles can resemble humans in the obstacle space, or even unplanned obstacles the robot might face and will need to avoid.

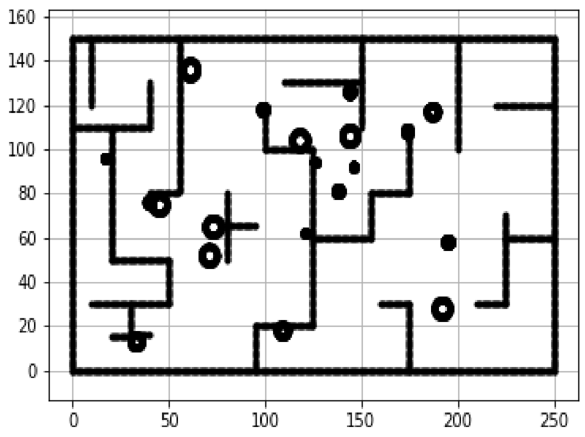


**Figure 3 Random Obstacles in the Maze**

The obstacles are simple polygons with random radii and locations, and the robot moves with a constant speed.

The maze environment is replicated on V-REP, as shown in fig 3, to simulate each search method using the RemoteAPI to connect to both MATLAB and Python. Furthermore, Turtlebot2 has been used as the mobile robot.

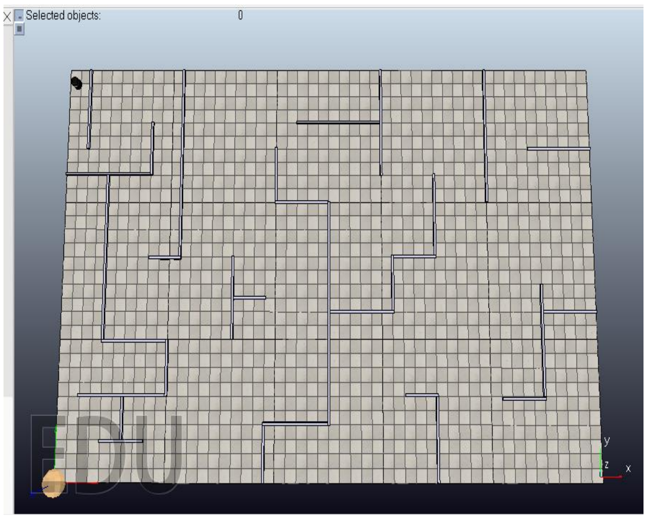


**Figure 4 VREP Maze Environment with Turtlebot2**

## IV. Method

### 1. A* Algorithm

A robot typically wants to reach the goal node as quickly as possible from the starting node. What A* does is that at each step, it picks the node with the lowest F where

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

Here, n is the previous node, g(n) is the cost of the path from the start node to n (or cost-to-come), and h(n) is the heuristic that estimates the cost of the cheapest path from n to the goal node (or cost-to-go).

A* is a heuristic search, which means it searches with information about the goal.

A* search finds the optimal solution to problems as long as the heuristic is admissible, which means it never overestimates the cost of the path to the from any given node.

The time complexity for A* is O(N), where N is the number of edges in the graph [2].

A simple A* Pseudocode is defined as the following:

```
List = PriorityQueue()
List.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not List.empty():
current = List.get()

if current == goal:
break

for next in graph.neighbors(current):
new_cost = cost_so_far[current] +
graph.cost(current, next)
if next not in cost_so_far or new_cost <
cost_so_far[next]:
cost_so_far[next] = new_cost
    priority = new_cost + heuristic(goal, next)
    List.put(next, priority)
    came_from[next] = current
```

Where heuristic is the Manhattan distance from any node to goal.

The issue with A* is that when both, the start and goal nodes are at the same height but with an obstacle in between (ex: start at top left and goal at top right), the algorithm will favor searching towards the right (because heuristically, it is the shortest path) until all nodes across the obstacle are explored, as shown here:

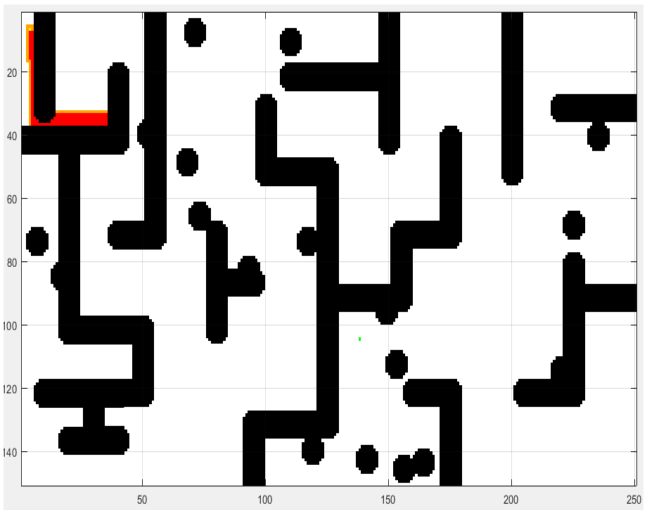


**Figure 5 MATLAB A* Path 1**

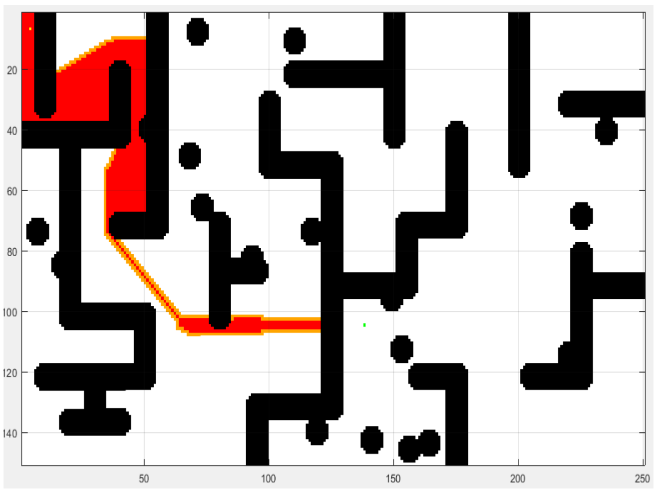


**Figure 6 MATLAB A* Path 2**

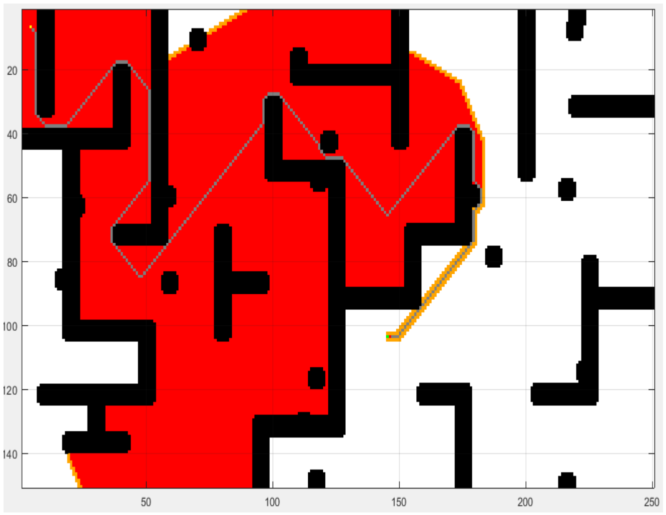


**Figure 7 MATLAB A* Start to Goal**

As shown, the A* algorithm uses the least Manhattan distance heuristic to search for the goal and backtrack the optimal path from start to goal.
Next, a MATLAB to VREP RemoteAPI connection has been implemented to simulate and validate the A* results as shown in fig 8 and 9. The video of the VREP simulation is provided in the supplementary files of this paper. Below are snapshot of the VREP simulation of the Turtlebot2 carefully navigating from the start point to the goal point. A thorough analysis of the computation time and run time is provided in Section V. Results and Conclusions.

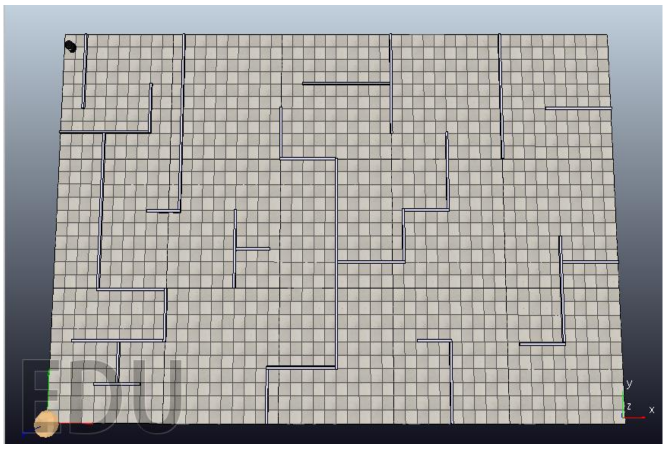
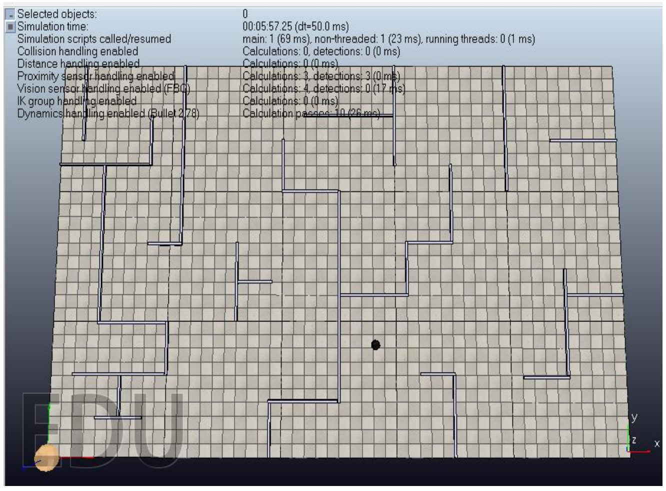


**Figure 8 A* VREP Simulation 1**



```
Selected objects:                           0
Simulation time:                            00:05:57.25 (dt=50.0 ms)
Simulation scripts called/resumed           main: 1 (69 ms), non-threaded: 1 (23 ms), running threads: 0 (1 ms)
Collision handling enabled                  Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                   Calculations: 0 (0 ms)
Proximity sensor handling enabled           Calculations: 3, detections: 3 (0 ms)
Vision sensor handling enabled (FBO)        Calculations: 4, detections: 0 (17 ms)
IK group handling enabled                   Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)     Calculation passes: 10 (26 ms)
```

**Figure 9 A* VREP Simulation 2**

The algorithm still shows the optimal path between the start and goal nodes but run time would be much faster if the algorithm can avoid obstacles more efficiently. This issue can be tackled by using the upcoming methods: RRT* and PRM.

## 2. RRT*

The RRT* algorithm is the most extensively used algorithm in recent years, due to is feasible motion planning ability and optimality with low run time. They are probabilistic complete algorithms and have natural support for solving high dimensional complex problems [11].
This algorithm works better in a dynamic environment as compared to A* and PRM. It also has proven to yield optimal path with a non-holonomic robot with differential constraints. This algorithm uses random sampling to pick out a node at random and check the node that is closest to it and makes it its parent, if there aren't any obstacles obstructing the path between the nearest node and itself. When it does find the closest node to it, it forms a new edge and then is added to the list of explored nodes. Then within a radius 'r', the algorithm finds all existing nodes that are close to this new explored node, and if the cost of the new node summed with the heuristic distance, is less than the cost of each of the nodes within the set radius 'r', the parent of each of these nodes is rewired to the new node and hence, their cost is updated.

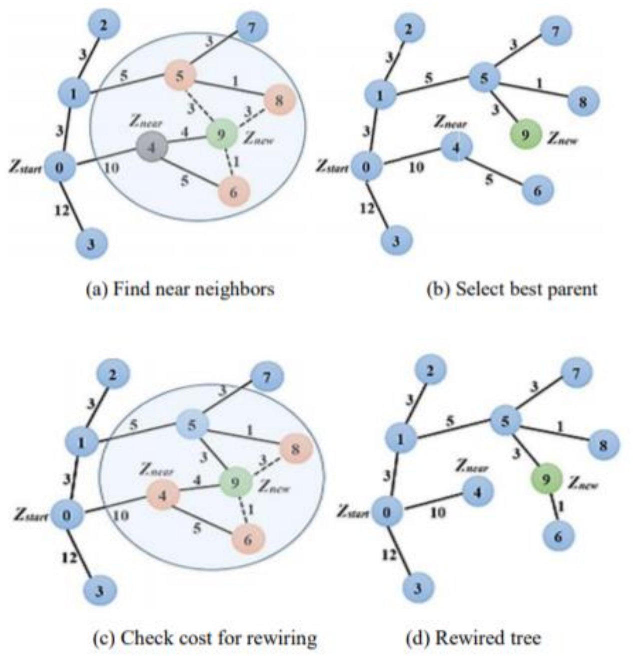


**Figure 10 Random Sampling, Nearest Node search and Rewiring [11]**

The pseudo code for this algorithm, is as follows [9]:

```
Algorithm 1. T = (V, E) ← RRT*( $z_{init}$)
1 T ← InitializeTree();
2 T ← InsertNode(Ø, $z_{init}$, T);
3 for $i$=0 to $i$=$N$ do
4     $z_{rand}$ ← Sample($i$);
5     $z_{nearest}$ ← Nearest($T$, $z_{rand}$);
6     ($z_{new}$, $U_{new}$) ← Steer ($z_{nearest}$, $z_{rand}$);
7     if Obstaclefree($z_{new}$) then
8         $z_{near}$ ← Near($T$, $z_{new}$, $|V|$);
9         $z_{min}$ ← Chooseparent ($z_{near}$, $z_{nearest}$, $z_{new}$);
10        $T$ ← InsertNode($z_{min}$, $z_{new}$, $T$);
11        $T$ ← Rewire ($T$, $z_{near}$, $z_{min}$, $z_{new}$);
12 return $T$
```

**Figure 11 RRT* Pseudo code [9]**

When implemented with our obstacle space, the following graphs showing node explorations, were generated at different times.

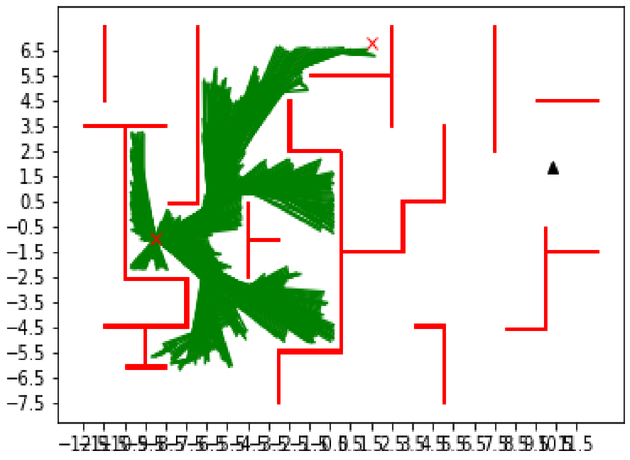


Figure 12 Node Exploration at t=20s

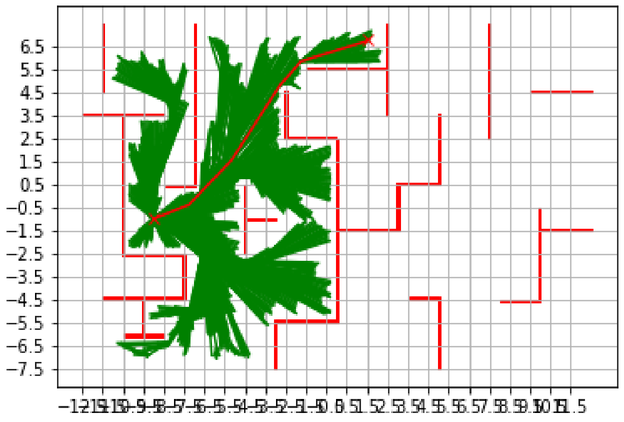


Figure 13 Final Path

This algorithm explores the entire obstacle space before finding the most optimal path. This is the reason why convergence rate slows down. To introduce an element of complexity, we have generated random obstacles between a range of 10 to 20. Figure 14 shows the node exploration of the maze environment for 10 obstacles.

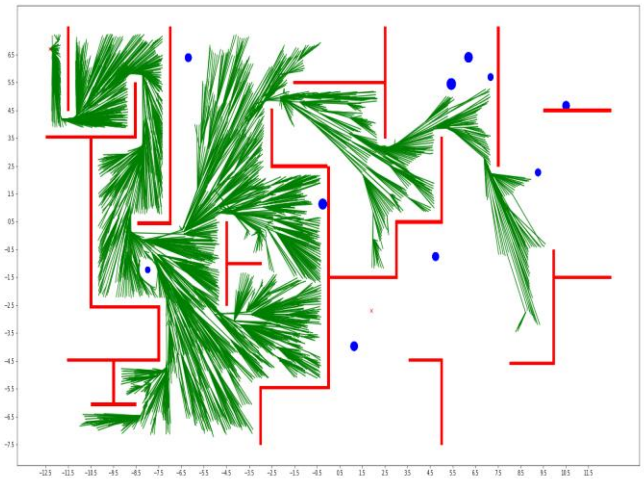


Figure 14 Node Exploration at t=100s and obstacles=10

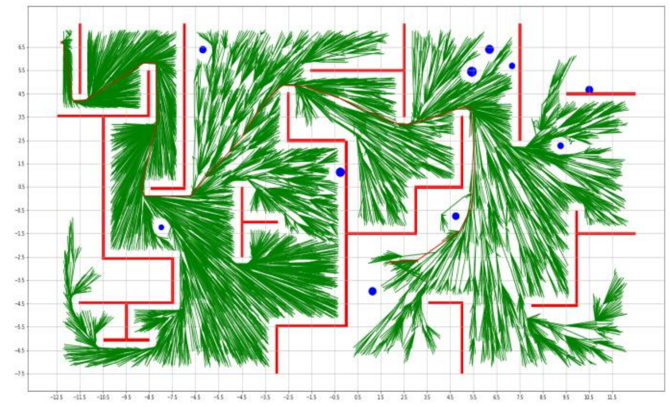
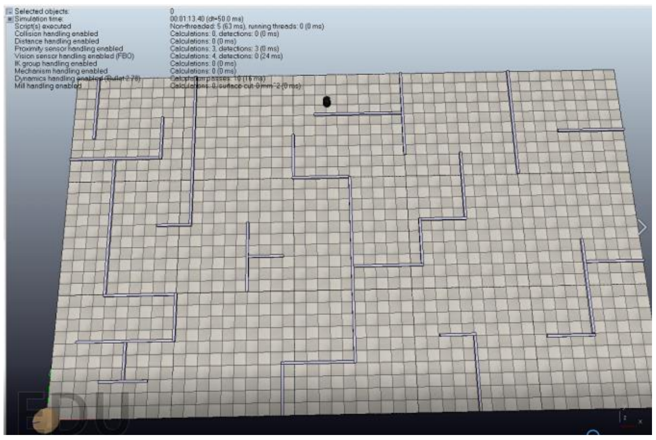


Figure 15 Node Exploration at t=360s and obstacles=10

As the number of iterations increase, RRT* improves its path cost gradually due to its asymptotic quality [11] .The code with the given start and goal points takes a total of 22 seconds to run. RRT* computes the optimal path from start to goal node and is single queried. It is a probabilistically complete algorithm with a time complexity of O(log n). Works the best when in robot is in a dynamically changing environment. Though, it improved path cost but on the other hand it also slowed down convergence rate of RRT* [11].
We further implemented this on TurtleBot2 using the VREP simulation software to traverse the path generated by the TurtleBot2.



```
Selected objects:                      0
Simulation time:                       00:01:13.40 (dt=50.0 ms)
Script(s) executed                     Non-threaded: 5 (63 ms), running threads: 0 (0 ms)
Collision handling enabled             Calculations: 0, detections: 0 (0 ms)
Distance handling enabled              Calculations: 0 (0 ms)
Proximity sensor handling enabled      Calculations: 3, detections: 3 (0 ms)
Vision sensor handling enabled (FBO)   Calculations: 4, detections: 0 (24 ms)
IK group handling enabled              Calculations: 0 (0 ms)
Mechanism handling enabled             Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78) Calculation passes: 10 (16 ms)
Mill handling enabled                  Calculations: 0, surface cut 0 mm^2 (0 ms)
```

Figure 16 Vrep Simulation

3. Probabilistic Roadmap (PRM)

The PRM algorithm [8] is a sampling based planning module that computes a collision free path for the robot. It has a time complexity of O(NlogN) for N nodes. This algorithm overcomes the failure of A* to compute paths for high dimensional configurational space and high degrees of freedom within a reasonable time. PRM has two phases: the construction or learning phase and the query or path generating phase.
In the learning phase, the algorithm generates a roadmap (graph) i.e. the topology of free space. Initially, the graph is empty. A random node is chosen with random co-ordinates and checked if it belongs in the free space. If it does, the node is added to the graph as a vertex. Next, the K nearest neighbors of the node are expanded and are connected if feasible. This is stored as an edge in the

graph. This is done for N number of nodes which generates the roadmap.

The pseudo code for the above is as follows [5]:

```
N = ø
E = ø
loop
α: select a random configuration
check if α is in free space
N ← NU{ α}
for each c belongs neighborhood(α)
    if (not sameconnected(c, α)
        connect(c, α)
        E ← E U{{c,n}}
```

In the query phase, given an initial point and a goal point a planning algorithm is implemented to find a path between the two in the generated roadmap. This project uses Dijkstra algorithm for generating path.

The algorithm is well adapted for the known maze environment since it saves the roadmap and uses it for path generation as shown in fig 17. The blue nodes show the roadmap whereas the green nodes shows the node exploration with Dijkstra.

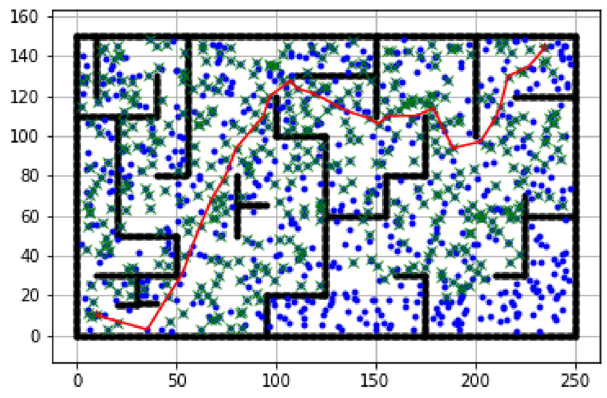


**Figure 17 PRM algorithm for static maze environment.**

In the project the PRM algorithm has also been implemented on a maze environment with random obstacles as shown in the figure below. The map in fig 18 does not generate a final path as the circular random obstacles block the path.

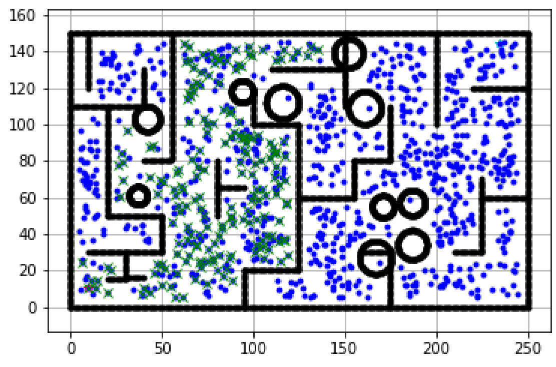


**Figure 18 PRM algorithm(obstacles=10).**

The final path is shown in fig 19 below.

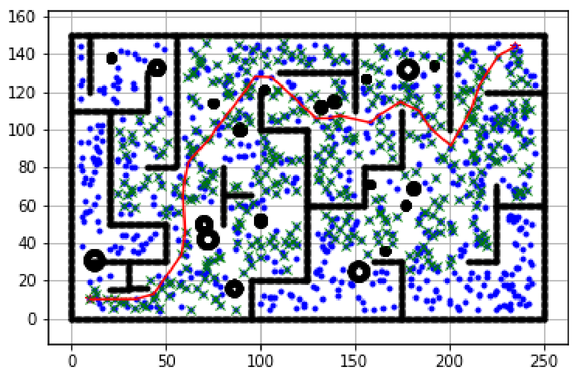


**Figure 19 Final Path (obstacles =20).**

The success of the PRM algorithm depends on factors such as number of nodes to be generated for the roadmap, the number of neighbors to be explored, the distance of the neighbors from the node etc. These parameters have to be tuned and selected optimally in order to get the path. If the number of nodes in the roadmap are too less the path planning algorithm will be unable to find a path since it can't explore beyond the nodes in the roadmap. PRM is a probabilistically complete algorithm i.e. the probability of getting a solution increases with time and depends on other factors

The final path generated is simulated in V-REP using the Python API as shown in fig 20 . The video is provided in supplementary files.

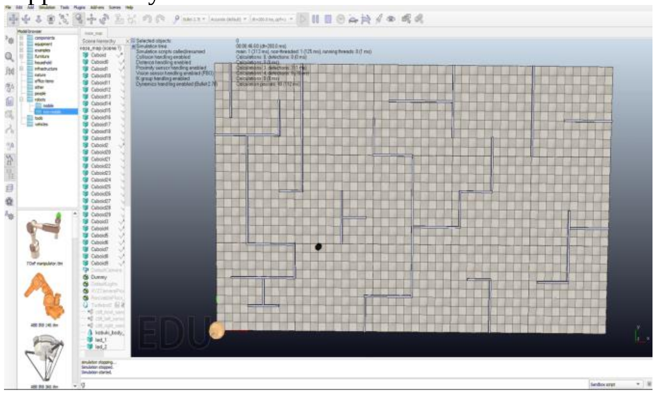


**Figure 20 V-REP Simulation.**

## V. Results And Conclusions

A*, RRT* and PRM algorithms are compared based on their computation time, nodes explored and ability to deal with random obstacles as shown in table 1.

| **Property** | **A*** | **RRT*** | **PRM** |
|---|---|---|---|
| Platform | Matlab | Python | Python |
| Time Complexity | O(N) | O(logN) | O(N*logN) |
| Completeness | Complete. | Probabilistic Complete. | Probabilistic Complete. |

| | | | |
|---|---|---|---|
| Solution | Optimal | Optimal | Depends on path search algorithm. |
| Time (maze only) | 4.878s | 361.883s | 2.28360s |
| Time (with random obstacles) | 4.784s | 560.723s | 4.84917s |
| No. of nodes generated (maze) | 13043 | 10689 | Roadmap: 503 Dijkstra: 392 |
| Query | Single | Single | Multi |

**Table 1. A* vs RRT* vs PRM algorithms**

RRT* and PRM and both sampling-based algorithms and thus have an advantage over A* in their ability to handle higher dimensional configuration space and higher degrees of freedom of the robot.

PRM works the best for static obstacle space as it generates a roadmap and uses it for future iterations. Whereas the other two perform node exploration for every new start and goal point thus increasing the computation time. RRT* is the slowest as it explores the entire space. The comparison is as shown in fig 21.

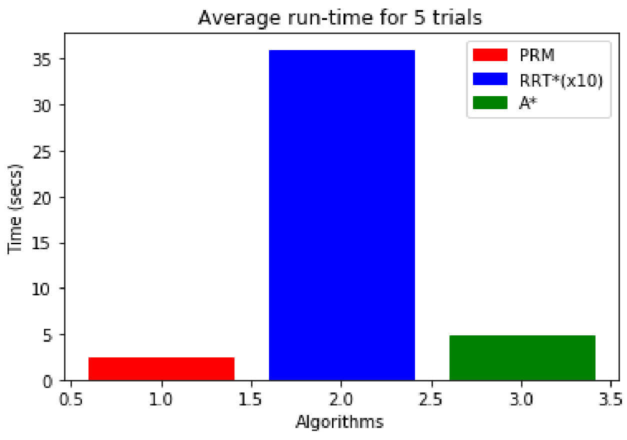


**Figure 21 Computation Times.**

On the other hand, for random obstacles, PRM though fast is similar to A* since it has to generate a new roadmap every time a start and goal point is given. This also helps conclude that PRM would fail for dynamically moving obstacles.

PRM fails if the number of nodes generated aren't enough which is not observed in A* and RRT*.

Although RRT* generates the most optimal path by exploring the entire configuration space, its rate of convergence is slow. This is the reason why the run-time of this algorithm is the slowest among the three.

## VI. Future work

The algorithms can be further improved to take into account dynamically moving obstacles by doing planning and then re-planning. The number of algorithms compared in this study can be increased by taking into account other algorithms like PRM*, weighted A* etc.

## Acknowledgment

We would like to thank Professor Reza Monfaredi for his constant support and guidance. We are grateful for having him as a mentor and for helping us in every aspect of this project. He has provided useful incites on improving performance of the project and in making it more robust.

## References

[1] C. Alexopoulos, P. Griffin. (1992) "Path Planning for a Mobile Robot" IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 2.

[2] W. Seo, S. Ok, J. Ahn, S. Kang and B. Moon, "An Efficient Hardware Architecture of the A-star Algorithm for the Shortest Path Search Engine," *2009 Fifth International Joint Conference on INC, IMS and IDC*, Seoul, 2009, pp. 1499-1502.

[3] S. Jose, A. Anthony. "Mobile Robot Remote Path Planning and Motion Control in a Maze Environment" 2nd IEEE International Conference on Engineering and Technology (ICETECH)-2016.

[4] Permana, S.D., Bintoro, K.B., Arifitama, B., Ade, & Syahputra, A. (2018). "Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game".

[5] L. E. Kavraki, P. Svestka, J. -. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566-580, Aug. 1996

[6] Jarad Cannon and Kevin Rose and Wheeler Ruml , Real-Time Motion Planning with Dynamic Obstacles, Proceedings of the Fifth Annual Symposium on Combinatorial Search

[7] Priyanka Sudhakara and Velappa Ganapathy, Path Planning of a Mobile Robot using Amended A-Star Algorithm, I J C T A, 9(37) 2016, pp. 489-502.

[8] David Hsu,Jean-Claude Latombe, Hanna Kurniawati, On the Probabilistic Foundations of Probabilistic Roadmap Planning.

[9] S. Karaman, E. Frazzoli, "Sampling-based algorithms for optimal motion planning", *Int. J. Robot. Res.*, vol. 30, no. 7, 2011.

[10] I. Noreen, A. Khan, Z. Habib. "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions". *IJACSA, Vol. 7, No. 11, 2016*

[11] I. Noreen, A. Khan, Z. Habib." A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms". *IJACSA*, VOL.16 No.10, October 2016

[12] Liu, Changliu & Wang, Yizhou & Tomizuka, Masayoshi. (2017). Boundary layer heuristic for search-based nonholonomic path planning in maze-like environments. 831-836. 10.1109/IVS.2017.7995819.