NUST

NATIONAL UNIVERSITY
OF SCIENCES & TECHNOLOGY

COURSE: ARTIFICIAL INTELLIGENCE

PROJECT: PATH PLANNING ROBOT

SUBMITTED BY: UMAIR KHAN

SUBMITTED TO: DR. YASAR AYAZ

Path Planning Robot

Umair khan
School of Mechanical & Manufacturing Engineering (SMME), National Unoversity Of Sciences and Technology, Islamabad, Pakistan

***Abstract*—The terminology "optimal path planning" defines to the process of determining the safest, quickest, and the smoothest path between two points. The two points are the initial and final points. It is really important part in many aspect in robotics application. There are many application requires path planning with accuracy, safety and reliability. Many robotic applications, including as autonomous vehicles, surveillance operations, agricultural robots, and planetary and space exploration missions, require this duty. RRT* (Rapidly Exploring Random Tree Star) is a well-known sampling-based planning method. It has become really popular because of its dimensional complex problems. There are two methods used in this project to solve this problem; RRT and A star (*). Considering rapid demand of this field, this paper represents the development of an agent based on path planning approaches. Both the algorithms are compared and we'll show the performance of both the algorithms.**

Keywords*— Path planning, Robot navigation, Optimal Path, RRT, A star*

I. INTRODUCTION

Path planning refers to the development of collision-free paths from an initial state to a defined destination state at an optimal or near optimal cost. When examining various robot applications and restrictions, optimal criteria may be based on one or more characteristics, such as shortest physical distance, smoothness, minimal risk, decreased fuel demands, maximum area coverage, and low energy consumption.

As a result, in the context of route planning for robots, an optimum path refers to finding a viable plan with optimal performance based on the application's defined criteria [1]. The holonomic and non-holonomic limitations have an impact on optimal path planning. The term non-holonomic, according to LaValle, refers to differential constraints (restrictions on permitted velocities) that are not totally integrable, such as car-like robots, while holonomic constraints, such as robotic arms, are completely integrable [1]. Because of its multiple applications in autonomous automobiles, path planning algorithms are crucial for mobile robot motion planning [2]. RRT* was a game-changing innovation in high-dimensional path planning by Karaman and Frazzoli [3]. If sufficient run time is available, RRT* always converges to an optimum solution. With multiple successful applications, RRT* has had significant success in tackling high-dimensional complex problems.

II. RRT* TECHNIQUE

In order to have a better grasp of this study, this part covers key path planning principles linked to RRT*. Prior to discussing RRT* many techniques, it is necessary to first establish the core operations of RRT*. These methods are included in all RRT* variations, although how they are implemented varies between planners and applications.

A. Formulation of Problem

The configuration space is where RRT*-based techniques operate. This configuration space is a collection of all the robot's conceivable transformations. The aim is to discover an optimum collision-free path between the initial init z and goal z states in Z free with the lowest path cost in the shortest time feasible t ∈ R, where R is a collection of real numbers.

*B. Tree Expension for RRT**

Instead of one long path, RRT* creates a tree of several short pathways that are randomly ordered. From the starting state, it creates a tree. Start z to find a way to the desired condition z-goal the tree is with each iteration, the quality of the product improves. Each iteration includes a The sampling process chooses a random state, such as zander rand from room for setup Z The sample was chosen at random zander rand is not accepted if it is found in Zob If, on the other hand, it's in Z is liberated. then a close relative say the node z closest is searched in tree T according to a predefined set of criteria metric If this is the case, zander rand consists of Z is liberated. As well as being accessible to z closest then, based on the predetermined step size, a local planner is used. Connects it to the tree and

Algorithm 1. $T = (V, E) \leftarrow RRT^*(z_{init})$

```
T ← InitializeTree();
T ← InsertNode(Ø, z_init, T);
for i=0 to i=N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(T, z_rand);
    (z_new, U_new) ← Steer (z_nearest, z_rand);
    if Obstaclefree(z_new) then
        z_near ← Near(T, z_new, |V|);
        z_min ← Chooseparent (z_near, z_nearest, z_new);
        T ← InsertNode(z_min, z_new, T);
        T ← Rewire (T, z_near, z_min, z_new);
return T
```

inserts it zrand rand and z closest. Otherwise, using a steering function, the planner creates a new node znew and adds it to the tree by linking it to the nearest z. The ability of RRT* to explore regions in Z free space is known as Voronoi bias. To verify that the link between znew and nearest is free of collisions, a collision checking mechanism is used. The expansion is shown in the figure Fig.1.

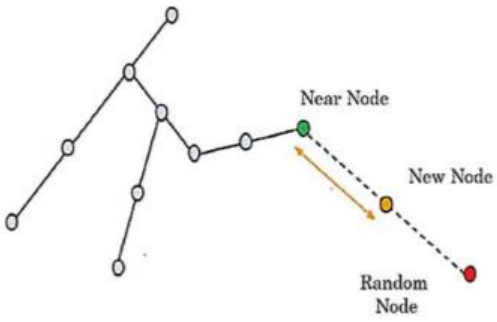


Fig. 1. RRT* Tree expansion [4]

This is the introduction how RRT* works.

III. .A (*) STAR

One of the most well-known path planning algorithms is the A* method, which may be used in either metric or topological configuration space [5]. This is the combination of heuristic function and the step cost function. The evaluation function is given by:

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

Here n is the path's final node, g(n) is the cost of the path from the start point to n, and h(n) is the heuristic function that predicts the cost of the cheapest way from n to the objective in formula . In general, g(n) is a constant number, but h(n) varies depending on the path taken. The best path can only be found by selecting an appropriate h(n) value. Currently, there is no set standard for selecting h(n). If the function h(n) changes, the projected path may alter. Under the necessity, we can set the h(n). Manhattan distance is the heuristic function with the highest frequency for choosing the most appropriate kind of h(n). It takes into account the minimal cost D of travelling from one site to the next. As a result, the heuristic function may be set at D Manhattan distance times. The A*(star) Algorithm is shown below.

Algorithm 1: *A*(S, G)*

```
1  for each state X in the environment model:
2    t(X) = new;
3  Initialize the OPEN list to be empty; Place state S on the OPEN list; t(S) = open; g(S) = 0;
4  while t(G) ≠ closed:
5    Find the state, X, on the OPEN list with minimum estimated path cost;
6    for each neighbor Y of X:
7      if Y is not in a forbidden region
8        if t(Y) = new
9          p(Y) = X; g(Y) = g(X) + c(X, Y); f(Y) = g(Y) + h(Y); Place Y on the OPEN list; t(Y) = open;
10       else if t(Y) = open
11         if g(X) + c(X, Y) + h(Y) < f(Y)
12           g(Y) = g(X) + c(X, Y); f(Y) = g(Y) + h(Y); p(Y) = X;
13   Delete X from the OPEN list; t(X) = closed;
14   if OPEN = NULL
15     return NO-PATH;
16 The optimal path from S to G is obtained.
```

III ALGORITHM SIMULATION

The Simulation is divided in two parts, the first is to find the optimum path using RRT* and the second is to find the path with the help of A*(Star) search Algorithm. But before moving to these Algorithms, we'll get familiar with the simulation procedure.

A. Software Description

In this section, we'll briefly discuss how to use this software/ application. There are few steps to get familiar with these will help you to watch how both the algorithms works, The user interface of the software, the steps are listed below;

1) The first step is to start the simulation, doing this will load values in the Initial position, goal position and DH parameters for the corresponding robot.You can update the coordinate by your choice as well. Shown in the Fig.2.

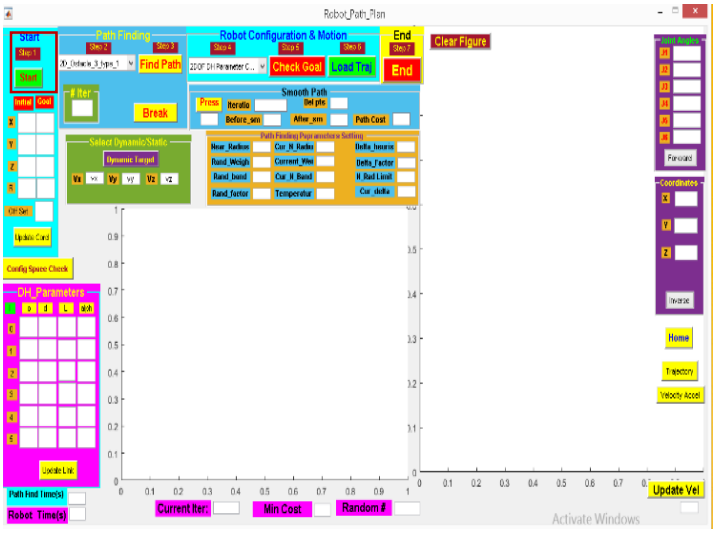


Fig.2. Shows Press start to load default parameters. You can change start & goal location (if desired).

2) In this step you'll have to select the type of obstacle from dropdown Manu

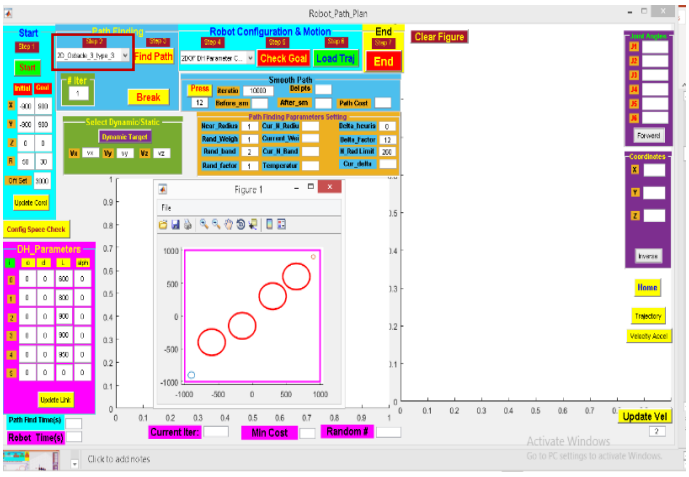


Fig.3. Selection of Obstacles from the manu

3) You can set the number of iterations using the button “#Iter”, shown below in the figure.

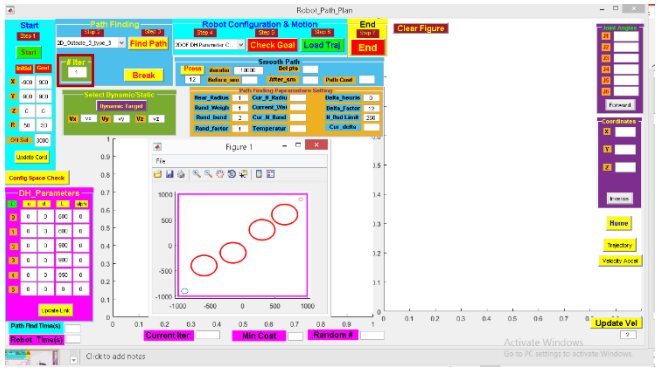


Fig.4. shows the selection of the Iterations

4) In this software you are privileged to decide either the goal should be static or dynamic, you want the goal to be dynamic then select the velocity components, shown below in the Fig.5.

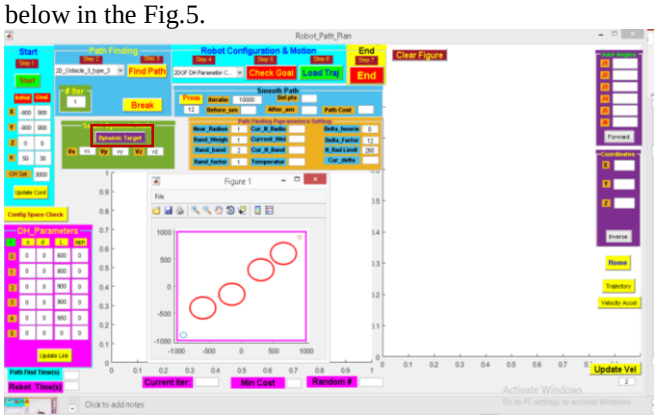


Fig.5. Select dynamic or static goal position Select velocity component

5) On the very next step you are asked to find the path so you should press the button “Find Path”. Shown below;

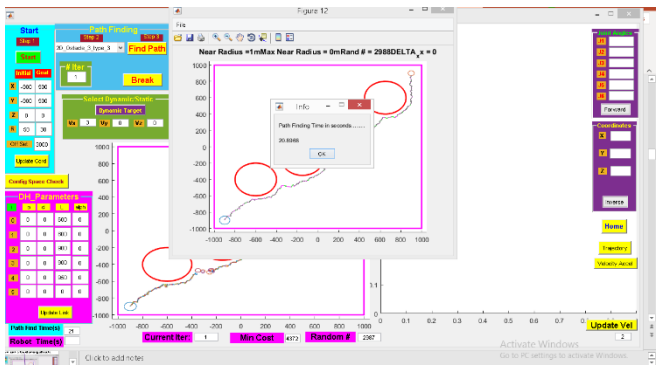


Fig.6. Press ‘Find Path’ to start optimum path algorithm

6) We have used simulated annealing to avoid local minima, because there are chances to get stuck on that point, so this novel idea will help to get out of that, as shown in below fig

Fig.7. stimulated annealing

7) In the next step you’ll see an optimal path among all the paths

Fig.8. Minimum path is found among all paths

8) RRT* or A*(star) algorithm will display the min cost and the time it takes to reach the goal

Fig.9. shows the time

9) The software will display the parameters related to optimization

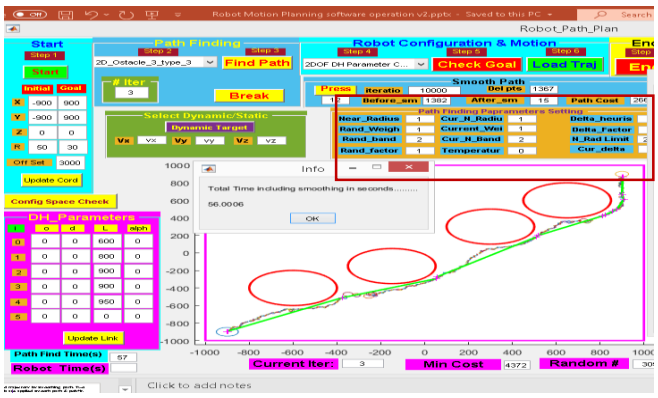


Fig.10. shows the parameters

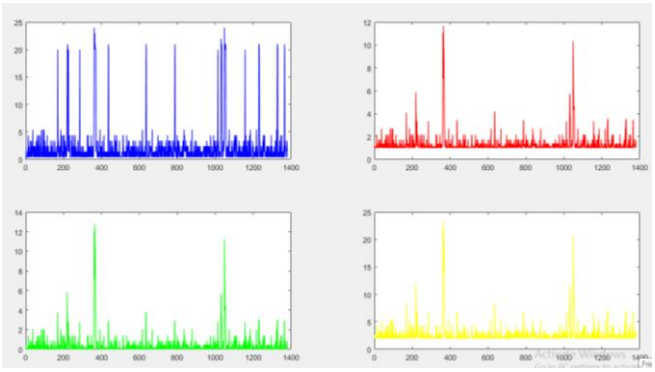


Fig.11. shows the how it located minimum path with the help of simulated annealing

10) We can select the degree of freedom for out simulated agent as well, the selection of the robot configuration is shown below. You can the dh parameters with the help of "update link" the robot configuration will change accordingly

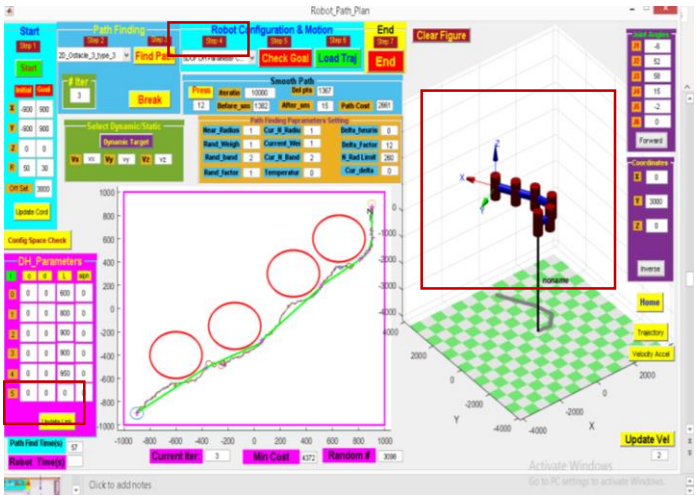


Fig.12. shows the robot configuration

11) You can view the trajectory in 3d by pressing the "load trajectory

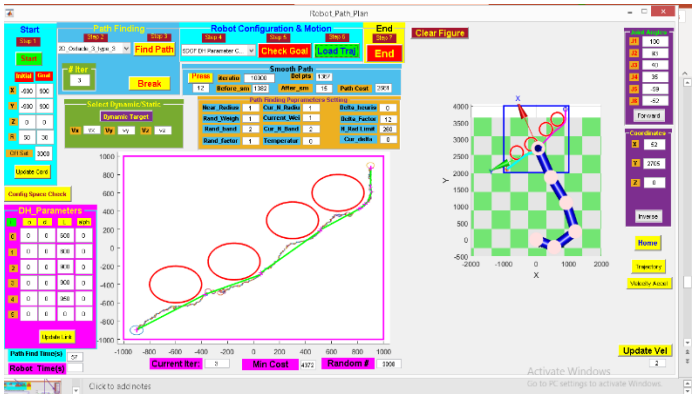


Fig.13. Shows how get this achieve

12) In the last, you'll be able to see the trajectory time and the option to end this simulation, shown in the fig.

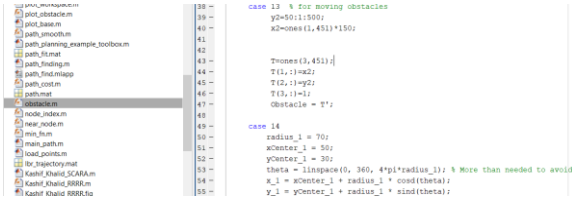
Fig.14. Shows time to reach goal state

B. *Obstacle Creation*

If we analyze the code we'll find that every obstacle values pass are passed in a variable "a".

```
clf(figure);
a = get(handles.obstacle_selection,'Value');
```

So the next is to locate the Obstacle_selection so that we can change the values and create the obstacle of our own. After analyzing the code we are able to locate from where the obstacles are passed to the variable "a". there are number of cases for each obstacle. So I created my own obstacle that looks like a concentric circles and a straight wall like obstacle.



The obstacles are shown in the below figure

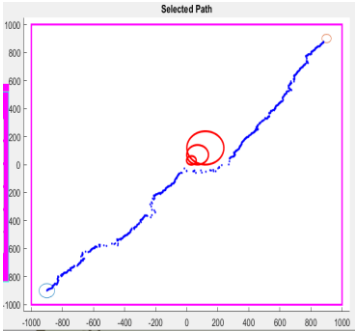


Fig.15. Using RRT*

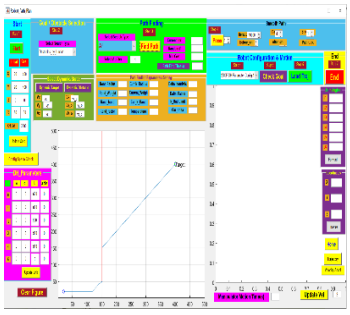


Fig.16. Using A*(star)

C. *A* (star)*

As we know that A star search algorithm is used for finite states but this problem is having continuous state space, so first we have to divide the environment in NxN blocks so that the algorithm treat environment as finite state and will create the nodes accordingly as we mentioned above in the section of A*(star). And the algorithm is also defined above

D. *Comparison*

Path planning is a crucial and important issue in the field of robotics. In this project, we analyze RRT* and A*(star). As we know that A*(star) is well known for finding

path between two points; initial and final. So this is a optimum path finding algorithm between two points, whereas Random tree (RRT) family uses random sampling via state space and then converges to the optimal path. By simulating both the algorithms we have come to conclusion that RRT* is more faster than A*(star) but the path found by the RRT is significantly longer than A*(star) [6].

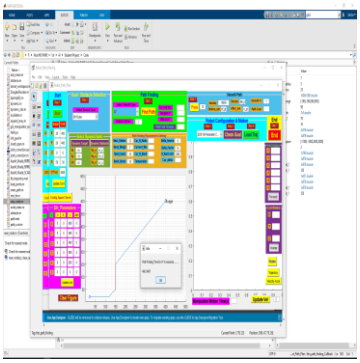


Fig.17. A*(star)

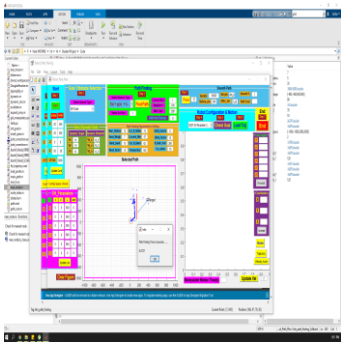


Fig.18. RRT*

IV Conclusion

This project help us to get familiar with real time simulation of the path planning and get in touch with the algorithms like how they work in real life, how would we be able to achieve our goal using different algorithms. This project brings the theoretical knowledge to practical to some extent. The difference between different algorithms in real world is something really fascinating.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] X. Lan and S. Di Cairano, "Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT," in *2015 European control conference (ECC)*, 2015, pp. 2360–2365.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[4] I. Noreen, A. Khan, and Z. Habib, "A comparison of RRT, RRT* and RRT*-smart path planning algorithms," *Int. J. Comput. Sci. Netw. Secur.*, vol. 16, no. 10, p. 20, 2016.

[5] S.-G. Cui, H. Wang, and L. Yang, "A simulation study of A-star algorithm for robot path planning," in *16th international conference on mechatronics technology*, 2012, pp. 506–510.

[6] P. Lajevardy and M. A. Oskoei, "A Comparison Between RRT * and A * Algorithms for Motion Planning in يد بايد ريد سم رح تد بک تد ابد ور ک رحد تم رد ود يد دعب هسد ياقد م مد تد يروگ لا يد اه * RRT و * A يد ارب ,مظعا ذ ايد و سوم ط يحم يد اه هديد چ يپ" no. August, 2015.