Roadmap-Based Motion Planning in Dynamic Environments

To implement path planning of any robot or
robots in configuration spaces with dynamic and static obstacles.

Author Name
Harsh Bharat Kakashaniya
Koyal Bhartia
Aalap Rana

We use different path planning algorithms to find path of robots in
dynamic environment to simultaneously start from different points and reach different goals.

Date : 5/7/2019

Roadmap-Based Motion Planning in Dynamic Environments

Harsh Kakashaniya, Koyal Bhartia and Aalap Rana

Abstract— **In this paper, an advanced technique is offered for path planning in dynamic surroundings, that is, searching a path for a robot in an environment which encompasses both dynamic and static obstacles.The method suggested develops a pragmatic algorithm based on the roadmap for the static part of the scene. The roadmap generated is used to form a time-optimal trajectory from start position to goal position without colliding with the non-static obstacle. The final path is obtained in two phases the first phase is the local level search where a sub-optimal trajectory is developed using the depth-first search (DFS) and in the following phase, an optimal path is obtained taking in consideration time as one of the parameters and using A* search algorithm. The solution obtained is applicable to all types of robots, in any configuration space where the motion of obstacles is unconstrained but should be known beforehand. Furthermore, the velocities of obstacle also should remain constant throughout the process. This approach can be applied to the multi-robot system.**

Index Terms— **Dynamic Planning, PRM , Local Trajectory , Global trajectory, State Time space , Probe , A* , KD Tree**

I. INTRODUCTION

Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.[1] As mentioned in above definition that the desired movement must satisfy the constraints , there are two main kind of obstacles in planning which are namely static obstacles and dynamic obstacles. As the name suggests that static obstacles have a fixed position and orientation in configuration space which does not vary with time,in contrast to dynamic obstacles. Numerous planning algorithm have been suggested and successfully implemented for motion planning with static obstacle space but due to the complexity associated with dynamic constraints this part of planning is relatively untouched and has huge scope of exploration.This project aims to find an optimal solution to the planning problem which has both static and dynamic obstacle in the environment with no constraints in movements, provided we have prior knowledge about the initial position of the obstacles. This paper uses probabilistic road-map planner (PRM) algorithm for finding the feasible path. [3] The main principle behind PRM algorithm is that it takes random samples from the configuration space of the robot, tests them for whether they are in the free space, and uses a local planner to attempt to connect these configurations to other nearby configurations. The starting and goal configurations are added in, and a graph search algorithm is applied to the resulting graph to determine a path between the starting and goal configurations. [2] The implementation of PRM algorithm is done in two stages. The first stage is known as construction phase where a road-map is formed by approximating the possible movement,and then it is connected to neighbors depending on the condition mentioned as parameters such as the nearest neighbour. The configurations and connections are added to the graph when the network becomes dense. The second stage is the query phase, where the path is created for movement from initial state to goal state. The algorithm is highly proficient for fixed obstacles, however when applied to moving obstacles it produces undermined results as the configuration space is extremely transient. Therefore, the implementation of traditional PRM algorithm is slightly modified by using it in levels. The first level is dedicated to road-mapping for static obstacles which is pre-processed without additional dimension of time and the second level does planning in query state for dynamic obstacles. Hence in this level we take in to consideration the space and time parameter simultaneously. [4] The proposed method is effective for both free-flying robots and articulated robots which have six degree of freedom. The algorithm can also be applied to multiple-robot motion planning problem in a confined dynamic environment. The proposed algorithm can be applied to robots which can be used in real situations at industries, airports, restaurants and hospitals.

II. PROBLEM ELABORATION

A. State Time Space

The path planning approach for immobile obstacle space can be solved taking into consideration only the configuration space, that is all the possible collision free positions of the robot from start to goal configuration. However, when we deal with a mobile obstacle we need to upgrade our configuration space,thus now our new configuration space has a time component in it. Thus, now we call this newly developed configuration space as the State Time-Space. The advantage of converting the configuration space to State Time Space are as follows: 1) The rigid robot can be thought of as a point robot. 2) Dynamic obstacles can now be considered to be static in state time space.

B. Roadmap

The paper uses the roadmap approach to calculate collision free vertices and edges in configuration space for static obstacles. Thus, the roadmap is formulated in the pre-processing step; vertices thus formed may include the start and goal nodes. If the start and goal nodes are not included in the vertices they are connected in the post-processing phase. Here we have taken the undirected roadmap as it is easy to implement and computationally fast. The approach is suitable for path planning in a dynamic environment because we just have to deal with mobile obstacles in the query phase. This also breaks the complexity of configuration space to a 1D structure. There are both advantages and disadvantages of using the roadmap approach, the advantage is that the constraints for movements can be taken into consideration in the preprocessing step which reduces the execution time. However, the whole algorithm uses the local path developed so if the local path developed is not accurate and smooth the whole trajectory would not be ideal. [9]

C. Problem Description

We can now define our problem in more clear perspective, which is to evaluate a realizable trajectory on an estimated roadmap for the robot starting from a point in state time-space configuration at time T_0 and reaching the desired goal at time T_f, without colliding with the obstacles in the space.

III. Method

Following is a brief overview of the steps that will be implemented in the project. The broad two approaches are:

A. Local Approach :

In pre-processing stage, we consider only static obstacles and build nodes in map using PRT and roadmap from start to end with A* search algorithm. Hence this phase is responsible for converting our problem for space search to 1D problem. And in future we will never need the explore nodes and use search algorithm again for the same environment. Hence now due to this we can integrate time in the process to take care of dynamic obstacles. The roadmap is built for this static part of the scene without the dynamic obstacles and without any additional dimension for time. A two-level approach is used to find the trajectory of the objects. Here the trajectories on single edges of the roadmap are found in an implicit grid in state-time space. The global approach is an extension of this local approach. On the global level, the local trajectories are coordinated using an A*-like search to find a near-time-optimal global trajectory in the entire roadmap. This is been implemented and output can be seen in Figure 7,8,9 .

B. Global Approach

To find a trajectory from a point to another, algorithms like Dijkstra search does not solve the purpose. This is because it is not always best to arrive as early as possible on each of the vertices of the road-map as a scenario may occur that there may exists an obstacle at the next stage. The following is done to implement the solution:
We have a roadmap with a start and a goal at time T_0. The start is ”s” of interval tree at T_0. Then we expand the probes to form neighbor vertices which tries to reach unvisited free interval on the edge’s destination. When probe arrives on the reachable interval on its destination vertex it continues to search for the upcoming interval. We have to consider the returning probes too and thus they are launched on the incoming edge. All the probes are stored in the priority queue, and in each iteration the probe with highest priority is expanded, and this process is done repeatedly until the goal is reached or the queue becomes empty. Thus, the program terminates when the goal is reached or the queue becomes empty ,in the second case of termination there is no possible path for the given input and a messaged is displayed to the user. However, sometimes deadlock condition may arise and thus to handle such scenarios we put a upper bound on the time and if the execution time goes above this upper bound we terminate the program.

- **The State-Time Grid** The distance travelled by the robot along the edge of a roadmap, represents the robot’s configuration as a variable x. This ranges from 0 to 1 corresponding to the source and destination respectively. Thus the resulting state-time space followed by the robot is 2D consisting of the pair (x,t) $\in$ [0,1] x [t0,∞]; [10] Following is an illustration of the same:

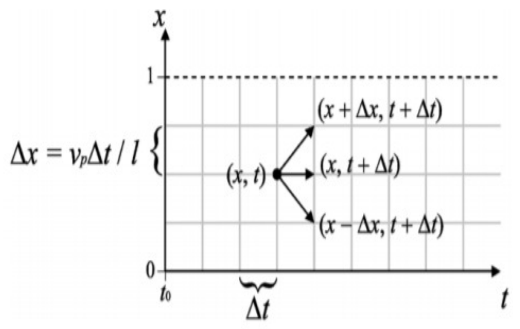

x
1
0
t_0
t
Δt
$\Delta x = v_p \Delta t / l$
(x, t)
$(x + \Delta x, t + \Delta t)$
$(x, t + \Delta t)$
$(x - \Delta x, t + \Delta t)$

Fig. 1: State Time grid of a roadmap edge

- **Finding a Local Trajectory** Following is the algorithm followed: A path is first chosen towards goal. If there is an obstacle in the path, the second option is for the robot to wait till the object passes and then continue travelling towards the goal. Pre-calculation of the trajectory is done if the object is huge. Once the robot reaches the goal, and then it tracks an obstacle, it comes back if needed and again traverses towards the goal. This has been illustrated below in the diagram. The computation of these steps uses the A* algorithm with consideration of cost of motion.

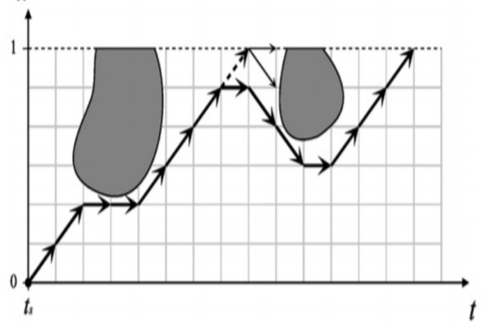


Fig. 2: Finding the shortest trajectory from (0,t_s) to x = 1

- **Global Trajectory** In the previous section, computation of a local trajectory on a single edge was discussed. Here, we will show how the global trajectory from a start vertex to a goal vertex through the roadmap is found. As with the local trajectories, the algorithm will find an approximately time-optimal trajectory.

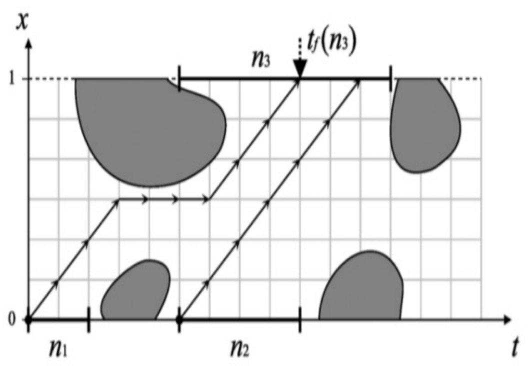


Fig. 3: State-time grid showing free intervals on the vertices and trajectories between them

As shown in the figure above, we have a interval tree to track all robots motion. Interval tree has start vertex as roots. For example as shown in the figure robot A which starts from the origin reaches the goal in the time $t_f(n_3)$. Now Robot B which reaches there at some time after $t_f(n_3)$ will not be allowed as robot A is already reached at the same position. This is possible only if A displaces itself from that position. Else robot B will have to wait or take an alternative path to reach its goal. In other words, the first trajectory subsumes the latter.

- **A Probe** The probe is the main conceptual object of our algorithm. It saves priority in path of Robots. It explores the roadmap in search of a global trajectory. During the usage of A* algorithm to track the probes [8], the following function is used.

$$f(p) = g(p) + h(p) \tag{1}$$

Here, f(p) gives an estimate of the cost of the time-optimal global trajectory. g(p) is the cost of the trajectory between the start vertex and the current state-time of p , and h(p) is a lower-bound . estimate of the cost of the time-optimal trajectory between the current state-time of and the goal vertex. As explained above, the probe concept is used to find the global optimum global trajectory considering collision free path.

IV. Assumptions

- There is a upper bound on the velocity of the robot.
- There is no constraints on shape and motion of dynamic obstacles as long as we have prior knowledge of its motions.
- There is no restriction on robot's motion.
- The road map for static obstacles is computed in pre-possessing phase without time as a parameter.

V. Software

The algorithm mentioned in the paper is implemented using Python 3.

VI. Innovation

- We have implemented KD tree for saving nodes generated using PRM which ease in the process of searching of 2D points and if needed can be upgraded to higher dimension like 3D in case of Drone path planning. [12]
- In pre-processing, DFS was used by the authors of paper to calculate the static path. We are using A* algorithm which will be time efficient for some configuration of Start and End points.
- Map generation was done using Minkowski sum which enable us to use the configuration for different radius robots.

VII. Results to be shown from previous study

Below is the image of the results reported by the reference paper we are working on.

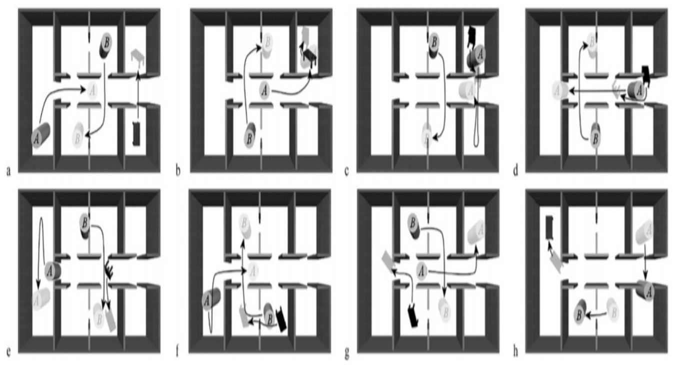


Fig. 4: Illustration of Algorithm implementation

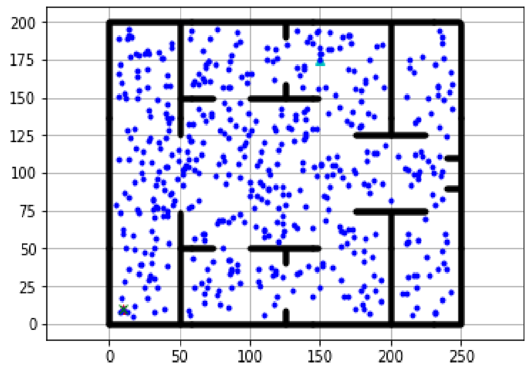


Fig. 5: Nodes created using PRM

In the above figure a table has to move from the lower-right room (a) to the left room (h). The table first moves to the upper-right room to find room for avoiding dynamic obstacle A (a-b). Then it moves in the slipstream of A through the passageway (c–d). The table then moves to the lower room to find room for avoiding obstacle B (e). After this, it moves in the slipstream of B to enter the left room (f). Obstacle A is moving toward the right part of the scene, so the table can safely reach its goal position (g–h).

Following is the results we get after the pre-processing phase:

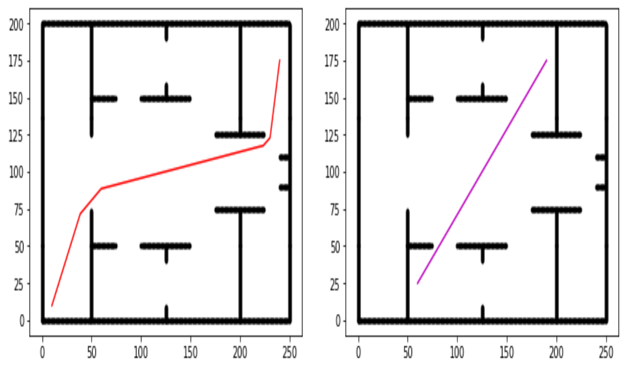


Fig. 6: Pre-processed path generated for Robot1

As shown below we analyze the results in the paper and discuss the focus on the following parameters in our dynamic planning conditions as follows:

1) **Road map size (edges and vertices)**: If we increase number of edges it will give a smooth and optimal path. Hence we will show this result with change in different number of nodes per map.

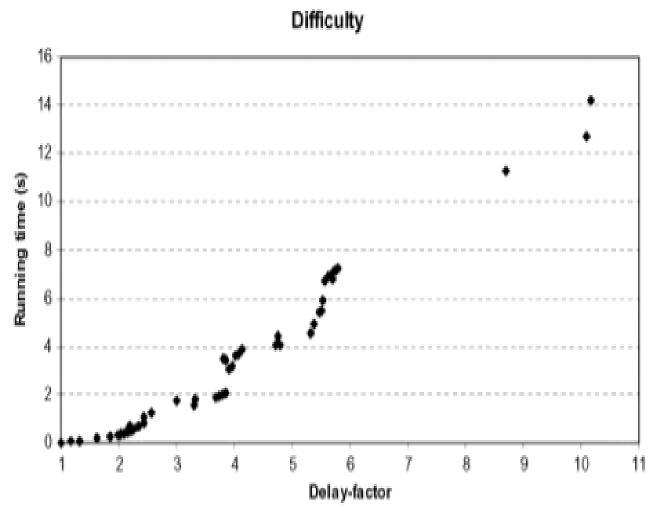


Fig. 7: Scatter-plot of 55 experiments with diff. delay factors

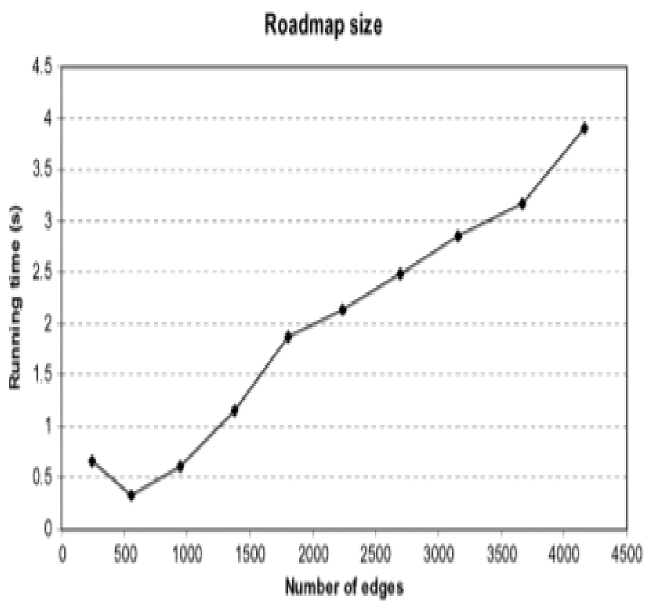
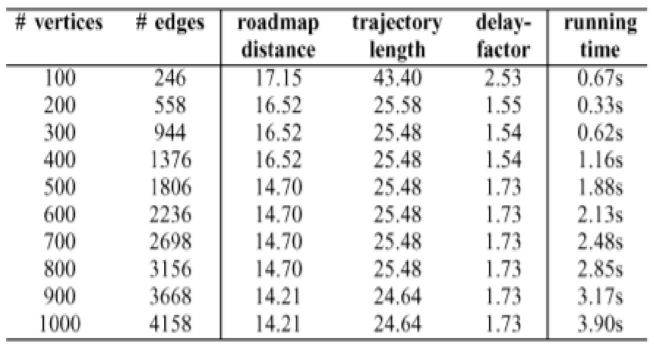


Fig. 8: Running time for roadmaps of various sizes

2) **Time Step**: When time step increases run time decreases. Hence we will check our map with different time steps and note the change in running time and prove the linearity. This can be shown from the result given below as time step decrease vertices increase which results in increase in running time.
3) **Delay factor**: As shown in data table delay factor decrease with increasing edges till a particular threshold after which it remains constant.
4) **Other parameters**: There are other parameters which affects the performance as DOF for manipulator. But in our case we have mobile robot so above are the parameters which are to be taken care off.

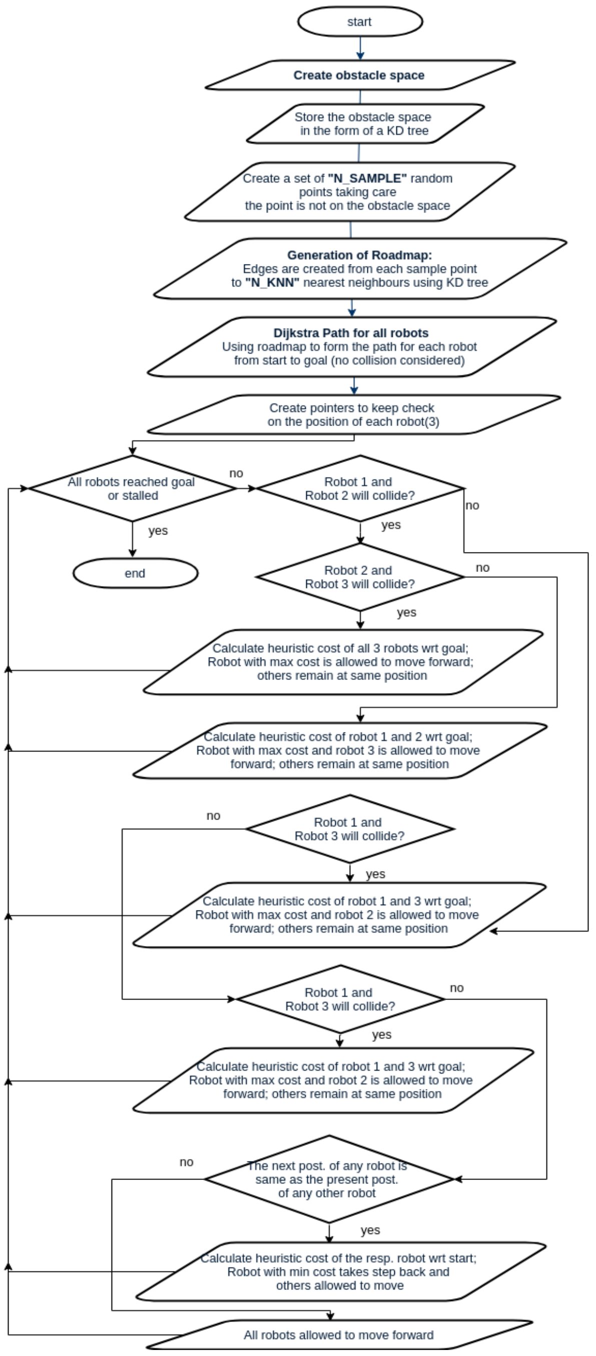
This results will be shown with the animation of multi-robot environment with static and dynamic obstacles and provide animation as a point robot in the environment.



# vertices	# edges	roadmap distance	trajectory length	delay-factor	running time
100	246	17.15	43.40	2.53	0.67s
200	558	16.52	25.58	1.55	0.33s
300	944	16.52	25.48	1.54	0.62s
400	1376	16.52	25.48	1.54	1.16s
500	1806	14.70	25.48	1.73	1.88s
600	2236	14.70	25.48	1.73	2.13s
700	2698	14.70	25.48	1.73	2.48s
800	3156	14.70	25.48	1.73	2.85s
900	3668	14.21	24.64	1.73	3.17s
1000	4158	14.21	24.64	1.73	3.90s

Fig. 9: Paper Results for various Roadmap Sizes

VIII. Flowchart of Implementation



start
Create obstacle space
Store the obstacle space in the form of a KD tree
Create a set of "N_SAMPLE" random points taking care the point is not on the obstacle space
Generation of Roadmap: Edges are created from each sample point to "N_KNN" nearest neighbours using KD tree
Dijkstra Path for all robots Using roadmap to form the path for each robot from start to goal (no collision considered)
Create pointers to keep check on the position of each robot(3)
All robots reached goal or stalled
no
yes
end
Robot 1 and Robot 2 will collide?
no
yes
Robot 2 and Robot 3 will collide?
no
yes
Calculate heuristic cost of all 3 robots wrt goal; Robot with max cost is allowed to move forward; others remain at same position
Calculate heuristic cost of robot 1 and 2 wrt goal; Robot with max cost and robot 3 is allowed to move forward; others remain at same position
no
Robot 1 and Robot 3 will collide?
yes
Calculate heuristic cost of robot 1 and 3 wrt goal; Robot with max cost and robot 2 is allowed to move forward; others remain at same position
Robot 1 and Robot 3 will collide?
no
yes
Calculate heuristic cost of robot 1 and 3 wrt goal; Robot with max cost and robot 2 is allowed to move forward; others remain at same position
no
The next post. of any robot is same as the present post. of any other robot
yes
Calculate heuristic cost of the resp. robot wrt start; Robot with min cost takes step back and others allowed to move
All robots allowed to move forward

Fig. 10: Flowchart of code implementation

As shown in the above, we first use PRM to create a random set of points which is used for the creation of roadmap using the specified no of edges. [11] Dijkstra is then used to get the path of the robots from the start to the goal. This locally generated path is used to check for collisions in the global approach. If 2 or more robots collide, the one with max heuristic distance from the goal is given priority to move. In case of no collision but possible stall in the next step, the robot which is less far from the start is made to step backwards.

IX. Paper implementation results

A. Collision free conditions

We have formed a user defined functions comparerobots(),increment(),checkradius(),and globalpath() to implement the global level trajectory. We at this stage take into consideration time as one of the parameters.The global level trajectory is formed when the function globalpath() function is called. The function has three condition corresponding to four possible cases. The cases are discussed in detail below:

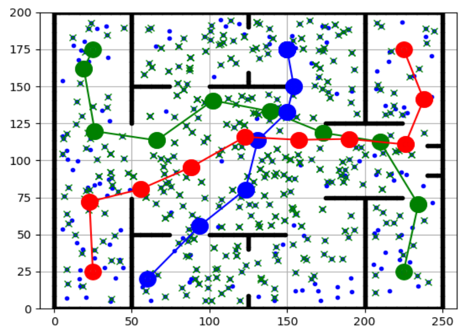


Fig. 11: Collision free movement for three robots

B. Case of collision between three dynamic obstacle

The globalpath function takes seven inputs which are the path coordinates i.e rx1,rx2,rx3,ry1,ry2,ry3, and robot radius and return node. A function called checkradius() is called to check the collision among the dynamic obstacles. It takes two co-ordinates of path formed and finds the difference between them and if the distance is less than the robot radius then they would collide and thus the checkradius() function returns True. This check for collision is carried out for all the three moving robots. if no collision is noticed then the robots are allowed to move on their desired path .If there is collision for all the three robots then the heuristic cost is calculated for all the robot and the one with least cost is allowed to move and rest robots have to wait. Then, the robot with maximum heuristic cost among the one which were at wait is allowed to move in the next cycle.

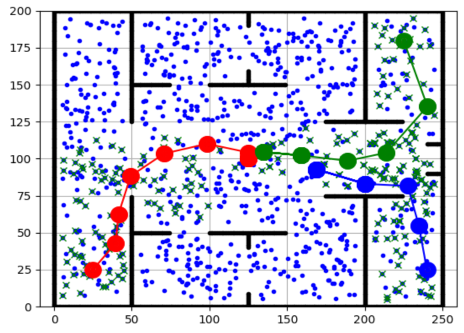


Fig. 12: Collision of 3 robots for three robots

C. Case of collision between any two robots

The globalpath() function takes seven inputs which are the path coordinates i.e rx1,rx2,rx3,ry1,ry2,ry3, and robot radius and return node. A function called checkradius() is called to check the collision among the dynamic obstacles. It takes two co-ordinates of path formed and finds the diffrence between them and if the distance is less than the robot radius then they would collied and thus the checkradius() function returns True. This check for collision is carried out for all the three moving robots. if no collision is noticed then the robots are allowed to move on their dezired path .If there is collision between two robots the three robot is allowed to move.However,the heuristic cost for the colliding robots is calculated and the one with maximum cost is allowed to move while the other robot waits. In the next cycle the robot which was stationary moves towards the goal.The above mentioned case give rise to total three cases.

D. Case when the path of 2 more robots overlap

The global path function takes inputs like which are the path coordinates i.e local path, and robot radius and returns node. A function called increment() is called when the path for more than one robots is same.The increment() function checks that the node generated in the next interval for any robot is not the node for a robot in the present interval, if this condition occurs then the robot which is nearest to goal is allowed to move forward and the robot which is far might have to reverse its path and get back to start.

Thus, the globalpath() function covers all possible scenarios and also incorporates the concept of advancing and returning probe. So by these methods we prioritize the robot and make them reach there goal.

Below considered is a case where start of one robot is end of other and visa-versa. So this give high probability of collision but due to A* implementation it is focused on the goal and takes different paths to reach the goal.

Below shown is an implementation of the above considered case where red and green robots have goals opposite to each other.

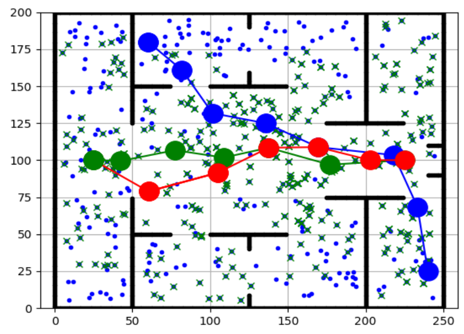


Fig. 13: When start and goal for two robots are complementary

E. Effect of change in edge length

This is an study of change in edge length below given is a figure of very less edge length and moderate number of points. It is difficult to generate road map and hence we do not get a path to goal.

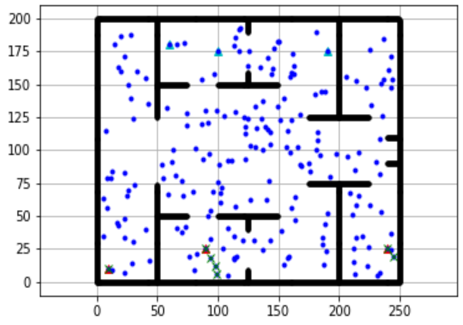


Fig. 14: when the maximum edge length is very less path may not be formed in some case

The below mentioned plot is obtained by increasing the edge length to 30 units and thus all paths are formed as seen in the plot.It can be noticed that the path is more curved and longer.The path has larger number of abrupt changes.

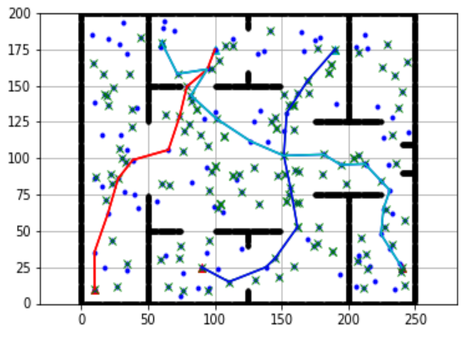


Fig. 15: Path formed for all robots

The plot is obtained by further increasing the maximum edge length to 40 units and thus the path formed is much smaller compared to that formed with edge length lesser than 40 units. The path is much smooth as the number of abrupt changes in direction is less compared to the previous plot.

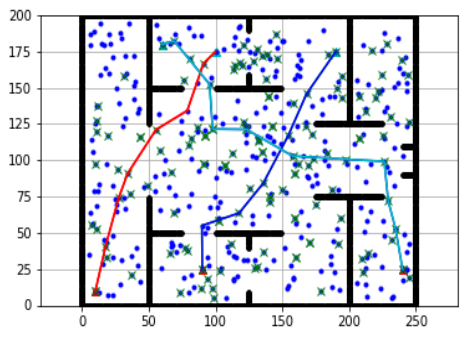


Fig. 16: When edge length is increased

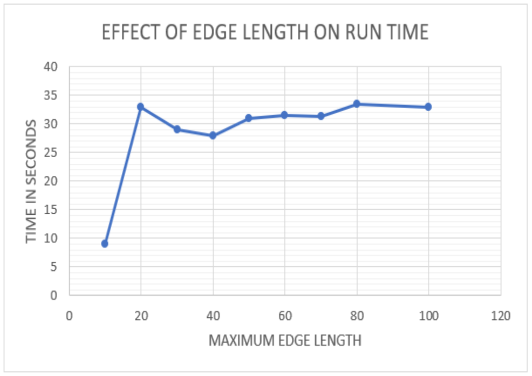


Fig. 17: Effect of edge length on run time

F. *Effect of change in total number of samples*

This is a study of how change in the total number of samples would affect the formation of final trajectory. The plot was formed keeping all the parameters constant except the total number of samples.It was found that the path was not formed for all robots.When the samples were less.

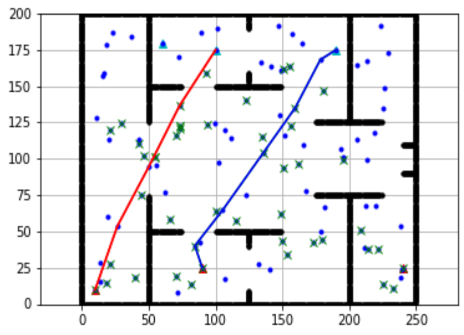


Fig. 18: when the number of sample is less the path is not formed

When the number of sampling points were increased it was found that the trajectory were obtained.But the computational time increased.The graph rightly illustrates the relationship between program execution time and the number of sample points.The execution time was found to increase with the number of samples.

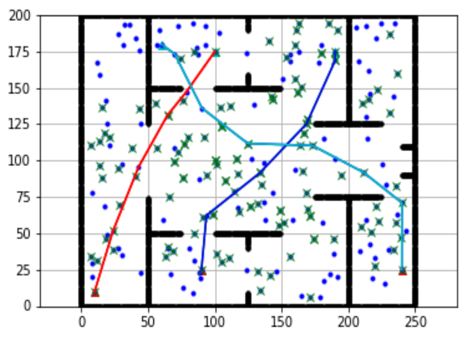


Fig. 19: increasing the number of samples to form path for all robots

G. *Effect of change in probe generation on run time*

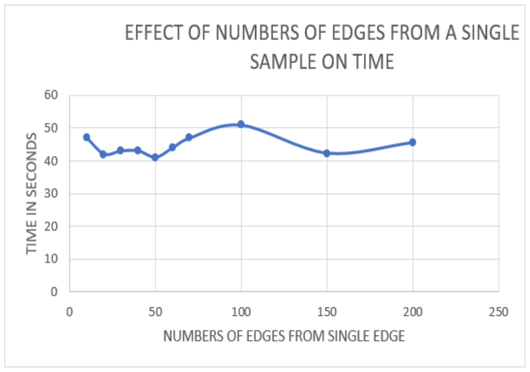


Fig. 20: Effect of change in number of probe from single edge on run time

X. Conclusion

The algorithm works in two level in the local level we use depth-first search to find trajectories and on the global level we have implemented A* algorithm.The method implemented through this paper can over come the drawbacks of Fujimura method [7] for path planning which is applicable only to point robots. The algorithm could be applied to multirobot application where computational time is very critical in path planning.The computation time for finding optimal path is very less when the obstacles are not present or away from the computed path.

XI. acknowledgment

We would like to thank Professor Reza Monfaredi for his constant support and guidance. He helped us in every step of the project from the selection to the simulation stage. We are grateful to have had his encouragement. We would also like to thank to the TA's: Ashwin Goyal and Utsav Patel for providing us with useful incites on improving the robustness of the project.

XII. Future Work

1) The local path is developed independently meaning without taking into consideration other local trajectories for different obstacles. Thus, the local path developed is not optimal, we would try to develop an advanced algorithm which would do parallel path planning taking into consideration the local trajectories of other obstacles. Thus, reduce the computation in the global level.
2) There are situations when we need to delete the probe before its stack becomes empty. This would save unnecessary expansions and checks for the collision. This can be explained by considering the following state time-space graph where the advancing probe returns to the source vertex, and thus it is clear that no kind of expansion can propagate it towards the goal vertex. Hence, the probe stack containing all the probe needs to be deleted to prevent the above-mentioned situation. Thus, we can consider the above mention scenario and modify our algorithm to overcome such shortcomings.

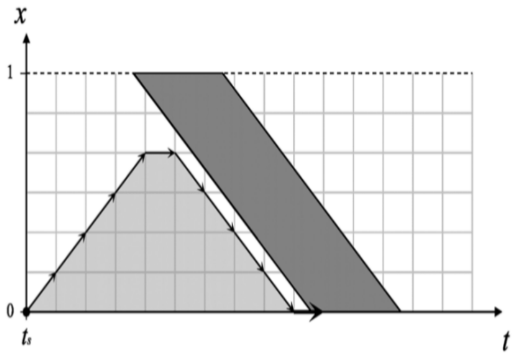


Fig. 21: Situation where probe can be deleted before its stack becomes empty.

3) The probes were considered independent, i.e. they do not affect the behavior of each other when present on the same edge. However, in the case of actual situation multiple probes may explore the same common part of the state-time-space. Thus, to overcome this situation we can allow only one probe to proceed on the same edge by providing global priority. This can be explained by the below illustrated example, consider 4 probes p1, p2, p3 and p4.P1, and p2 are returning probes whereas p3 and p4 are advancing probes. They are priorities in the list according to the following rules to avoid multi-probe exploration on the same edge. The rules are as follows:
    a) Returning probes precede advancing probes.
    b) Returning probes are sorted in reverse order of their start times ,i.e., there turning probe that started the latest is in front of the list.
    c) Advancing probes are sorted in order of their start times, i.e., the advancing probe that started latest is at the end of the list. Thus the list generated would be (p2, p1,p3,p4) the first probe is allowed to explore as being the nearest and then the order follows.

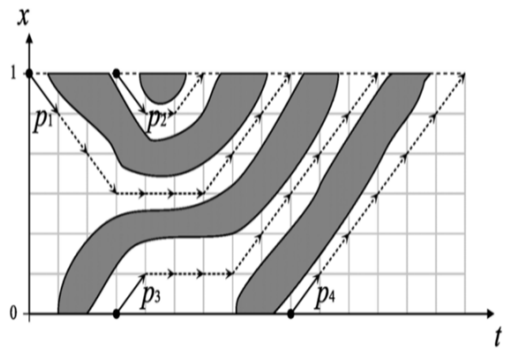


Fig. 22: Four probes on the same edge, and their projected traces (dotted)

4) We are enhancing our knowledge in data structure and algorithms so that we can replace the existing data structure to form a less time complex algorithm which can be executed in bare minimum time for global level planning.
5) The path is very dangerous as the path developed is passing very close to the obstacles and thus there is a high risk of collision with dynamic obstacles. We can try to integrate dynamic protection shield to guarantee a safe distance from dynamic obstacles.

References

[1] http://en.wikipedia.org/wiki/Probabilisticroadmap
[2] http://en.wikipedia.org/wiki/Motionplanning
[3] P. Svestka,“Robot motion planning using probabilistic road maps,” Ph.D. dissertation, Utrecht Univ., Utrecht, The Netherlands, 1997.
[4] P. Fiorini and Z. Shiller, ”Time optimal trajectory planning in dynamic environments”, J. Appl. Math. Comput. Sci. vol. 7, no. 2, pp. 101–126, 1997
[5] M. Erdmann and T. Lozano-Pérez,“On multiple moving objects”, Algorithmica, vol. 2, pp. 477–521, 1987.
[6] Time-minimum routes in time-dependent networks,IEEE Trans. Robot. Autom., vol. 11, no. 3, pp. 343–351, Jun. 1995.
[7] K. Fujimura, Motion Planning in Dynamic Environments. Tokyo, Japan: Springer-Verlag, 1991.
[8] J.-C. Latombe, Robot Motion Planning. Boston, MA: Kluwer, 1991.
[9] D. Nieuwenhuisen and M. H. Overmars, “Useful cycles in probabilistic roadmap graphs,” in Proc. IEEE Int. Conf. Robot. Autom., 2004, pp. 446–452
[10] T. Fraichard, “Trajectory planning in a dynamic workspace: A “state-time” approach,” Adv. Robot., vol. 13, no. 1, pp. 75–94, 1999
[11] Atsushi Sakai, https://github.com/AtsushiSakai/PythonRobotics/tree/master/PathPlanning/ProbabilisticRoadMap
[12] Jon Louis Bentley, ”Multidimensional binary search trees used for associative searching”,Volume 18 Issue 9, Sept. 1975