Low-Cost Path Planning in 2D Environment using A* Algorithm by Considering Slope of the Obstacle

Shivam Jaiswal * **Dr. Soumya S** **

* *School of Robotics, Defence Institute of Advance Technology, Pune, India (e-mail: shivamjaiswal095@gmail.com)*
** *School of Robotics, Defence Institute of Advance Technology, Pune, India (e-mail: soumya_s@diat.ac.in)*

Abstract: This paper deals with path generation of a mobile robot for an environment having slopes. In general, most of the path planning algorithms will take care of 2-dimensional or full 3-dimensional path planning whereas in the proposed method we are considering 2-dimensional environments along with slopes and travelable obstacles, which can be viewed as another layer of a 2-dimensional environment over the first one, and vehicle can manoeuvre over the obstacle rather than taking long road. For this purpose, A* algorithm is chosen for the path planning in this work, by considering its accuracy, efficiency and easiness in extending to Dynamic environment, D* algorithm. This can help mobile robot to conserve energy and reduce time taken to reach a goal. Algorithm presented here tries to find a path, in a 2D environment, with obstacle having slope, by travelling over them rather than going around. Ability to travel over an obstacle is basically based on the angle of the inclination, which can be estimated using sensor, like range finder, Lidar and others.

Keywords: A* algorithm, shortest path, sloped environment, ground robot, autonomous vehicle

1. INTRODUCTION

When it comes to path planning, there are number of algorithms and methods present in the literature. Few of the most common methods are Bug-0, Bug-1, Gradient based methods, Dijkstra, A*, RRT, D* and many more, Choset (2005), Stentz (1994), Vonasek (2009). These algorithms have their own advantages and limitations as compared to each other. Most widely known method of path finding algorithm is A* algorithm, which can be easily extended to D*. Most path planning algorithms are based on the initial position of the mobile robot and the goal position. The objective of these planners is to find an optimal path from start position to goal position. An optimal path can be a path which uses less time, less energy, or an optimal combination of both. It depends on the type of task, safety factor, or environment. There can be multiple optimal paths too in same situation, e.g., a path which is safe for humans if humans are travelling in an autonomous vehicle, may not be time efficient or may require more energy, but in this case safety factor will be given higher priority, but in other situation if vehicle is transporting some good, safety can be neglected to priorities time and energy factor. Even after choosing the optimal safe path, the robot should identify the less cost path among the chosen safe path.

Most number of available literatures can be classified into 2-dimensional and 3-dimensional path planning. Zhang (2017), Yang (2020), Wenzheng (2019) and Gajjar (2017) deals with the 2D path planning while Carsten (2006), Ergezer (2013), Li (2020) and Wu (2020) deals with 3D path planning. A few numbers of literatures are present which deals with the path planning of an environment in between them that is 2.5D environment, a path planning algorithm specifically for the sloped environment. Gu (2011) developed a method for such an environment by considering outdoor environment as non-flat surface while this paper tries a different approach for the same problem. The novelty of the paper lies in the perspective of the environment considered for this work. We considered environment as two layers of 2D environment connected by a slope. This can be visualized as two paper sheets placed parallelly one over the other and is connected by slopes. By considering environment as 2D layers rather than using usual 3D or 2.5D environments, system will become easy to implement and may reduce time as taken by an algorithm used for specifically for 3D environment.

This paper presents an algorithm based on A* algorithm to find a low-cost path between a start position, and a goal position of the vehicle. This algorithm utilizes the a vehicle's ability to travel over obstacle, for inclination less than a *critical angle*, and generate a path which involve both, some part on ground surface and some part over the obstacles. *critical angle* can be a maximum angle, for upward slope, or a minimum angle, for downward slope, at which a mobile robot or vehicle can travel easily and without losing its control when traveling in downward direction. Since moving on slope is costlier then moving on a 2D plane, as moving on slope has two cost factor, movement in horizontal as well as in vertical direction, this algorithm tries to find a way with less cost using Interpolation method for *node cost*, cost of travelling from

one node to its neighbouring node. This algorithm uses an 8-point grid-based method to travel to its neighbouring node. Traveling from one node to other node requires some cost to be paid (in form of Energy) depending on the type of node and its position with respected to previous node. In section 2, 8-point grid system and its cost function will be described in a 2D plane and some background on A* algorithm. Then in section 3, grid division and its cost calculation will be derived. In section 4 algorithms will be presented and will be described. Finally, in section 5 results of the few test cases will be presented.

2. BACKGROUND

2.1 8-point grid system and its costs

There are generally 2 type of grid-based systems, a 4-point grid-based system, in which nodes in left, right, front and back are considered but not nodes in diagonals, while an 8-point grid-based system uses all nodes, i.e. 8 neighbouring nodes, Choset (2005). For this paper workspace on the both the level, i.e., on ground as well on obstacle, is divided into 2D grids called nodes. Each node or grid can be of same size or different sizes, depending on performance requirement or environmental factors. For this path planning algorithm, 8-point grid system and uniform division of the workspace will be considered.

Suppose we have a vehicle or robot with dimensions $l*b*h$ (length * breadth * height), with a safety factor included in it and a workspace of the size $L*B$ (length * breadth). Considering grids have equal lengths and equal breadths, number of grids in length and breadth side is

$$n_{\mathrm{l}} = L/l \tag{1}$$

$$n_{\mathrm{b}} = B/b \tag{2}$$

Cost for grid system for traveling from one node to its neighbouring node is *node cost* C_{nh} (in horizontal plane), and cost for traveling from one node to any node in the workspace is its total cost, which is the sum of all node cost in its path. Since there can be various paths, total cost will be different for different paths.

For an 8-point grid-based system node cost is given by

$$C_{\mathrm{nh}}((x_0, y_0), (x_{\mathrm{i}}, y_{\mathrm{j}})) = \begin{cases} 1 & \text{for nondiagonal nodes} \\ 1.414 & \text{for diagonal nodes} \end{cases} \tag{3}$$

where,

$$i\ ,\ j \in \{-1, 0, 1\}\ \&\ i = j \neq 0$$

For total cost C_{t}, for traveling from one node to a goal node, say from (X_1, Y_1) to $(X_{\mathrm{n}}, Y_{\mathrm{n}})$ by a path P_1, a set of the travelable nodes to reach end point, is given by

$$C_{\mathrm{t}} = \sum_{i=1}^{n} C_{\mathrm{nh}i}((x_{\mathrm{i}}, y_{\mathrm{i}}), (x_{\mathrm{i+1}}, y_{\mathrm{i+1}})) \tag{4}$$

where $(x_{\mathrm{i}}, y_{\mathrm{i}})$ is an element from set P_1 at i^{th} position. We can change the node cost to travel to a specific node to our needs. E.g., cost for moving to goal point is generally considered as zero. See Fig 1 for illustration.

2.2 A Algorithm and cost association*

A* algorithm starts form one start node, stores the cost of the traveling to its neighbours by exploring them, after exploring all the neighbouring node, it moves to one of its neighbours and explore its neighbour, then moves to next node, this cycle goes on till it reaches goal, or explored whole environment. Using stored data, cost and nodes, A* finds the path with least cost, using the node cost stored, Hart (1968), Choset (2005).

Suppose we have a workspace divided into grids, as shown in Fig. 2. We have a set of all nodes S, each node s in S will have at least 8 adjacent neighbour and traveling from s to next neighbour node s_{next}, will have some cost as shown in Fig. 1. Moreover, travelling to a node containing obstacle will cost it a higher cost, let's say 10000. Now the node cost function (3) will be modified to

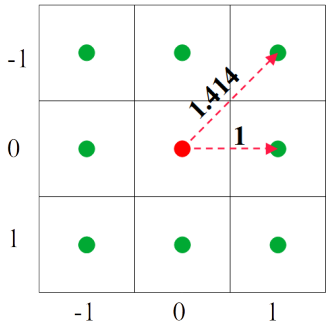

Fig. 1. 8 point-grid system. Nondiagonal path will cost 1 unit, while diagonal path will cost $\sqrt{2} \approx 1.414$ units, Choset (2005)

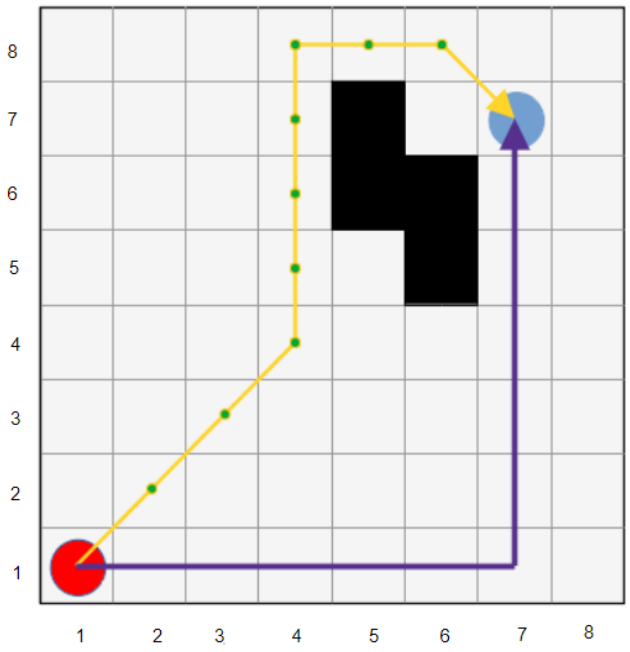

Fig. 2. Workspace. Red dot is start point and blue dot is goal point. For the two path, the cost of yellow path is 11.656 and violet path is 12, which are calculated using (4).

$$C_{\mathrm{nh}}(s, s_{\mathrm{next}}) = \begin{cases} 0 & \text{if } s_{\mathrm{next}} = s_{\mathrm{goal}} \\ 1 & \text{if } s_{\mathrm{next}} \in S_{\mathrm{ndia}} \\ 1.414 & \text{if } s_{\mathrm{next}} \in S_{\mathrm{dia}} \\ 10000 & \text{if } s_{\mathrm{next}} \in S_{\mathrm{obs}} \end{cases} \quad (5)$$

where,

- $s_{\mathrm{goal}} \in S$ is the position of goal node,
- $S_{\mathrm{ndia}} \in S$ is the set of nodes which are neighbour to s and are in nondiagonal nodes,
- $S_{\mathrm{dia}} \in S$ is set the of nodes which are neighbour to s and are in diagonal nodes,
- $S_{\mathrm{obs}} \in S$ is set the of nodes having obstacle

Before a robot can start moving, it need to scan the environment, divide it into nodes and generate a set, associating cost to all nodes based on (5). After this it need to extract shortest path P_{min} (set of nodes $s \in S$, which generate shortest part), from start node s_{start}, to the goal node s_{goal}, for the given cost definition. Once path P_{min}, is generated it can travel over it.

3. PRE IMPLEMENATATION

3.1 Workspace Generation from Environment

For this algorithm, environment is considered as a 2D surface with a slope for travelling from ground surface to the next level, one level stacked over the other surface. Each level acting as a new 2D plane. The obstacle which are accessible by any slope will act as next level. All 2D planes are divided with same grid structure as shown in the Fig. 3, giving a stacked grid layer. Each node will have either a free space, an obstacle, partial or fully filled obstacle or a slope (having a negative slope means going downward and positive slope means going upwards, with respect to the position of the robot) in it.

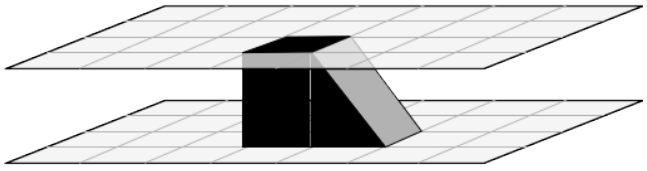

Fig. 3. Workspace shape, schematics of obstacle in workspace to visualize the shape of the workspace

Following considerations are used while implementing our algorithm

- A partial filled node with obstacle will be considered same as if it is completely filled, and discarding all other possibilities,
- The position of the start point and the goal point to be on ground surface,
- Number of the layers of 2D plane to be 2, i.e. ground and first layers,
- Each node is of same dimensions and in square shape,
- Workspace or environment is in a shape of square.

Suppose we have an environment of area $L * L$ units and area of ground mobile robot be $l * l$ units (safety region included), in horizontal plane. The total number of grids/nodes N, generated will be

$$N = (L/l)^2 \quad (6)$$

3.2 Cost Calculation

Node cost C_{n}, i.e. cost of traveling from one node to its neighbouring node and total cost of a path C_{t}, can be determined as explained in section 2. Also, node cost of traveling to a node with obstacle is infinite for ideal case. Apart from this there is another cost to be considered here, i.e., cost of traveling over a slope. A slope has two components horizontal and vertical, a horizontal node cost can be calculated as stated earlier, for a vertical node cost C_{nv}, consider a right-angle triangle with slope θ degree, having a horizontal node cost of C_{nh}, as shown in Fig. 4 then C_{nv} is given by

$$C_{\mathrm{nv}} = C_{\mathrm{nh}} * \tan\theta \quad (7)$$

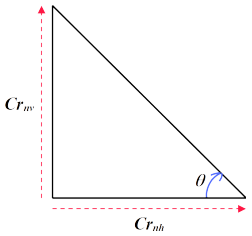

Fig. 4. Inclination, with slope angle θ and its horizontal and vertical costs

We will add a factor, α, which is required to adjust the vertical cost and it depends on the surface type, gravity acting on it and other factor's influencing its vertical cost. Hence (7) becomes

$$C_{\mathrm{nv}} = \alpha * C_{\mathrm{nh}} \tan\theta \quad (8)$$

and total node cost $\mathrm{C_n}$, becomes

$$C_{\mathrm{n}} = C_{\mathrm{nv}} + C_{\mathrm{nh}}$$

from (8)

$$C_{\mathrm{n}} = \alpha C_{\mathrm{nh}} \tan\theta + C_{\mathrm{nh}}$$
$$C_{\mathrm{n}} = C_{\mathrm{nh}}(1 + \alpha \tan\theta) \quad (9)$$

When slope is absent in a node, i.e., $\theta = 0$ total node cost C_{n}, will become equal to horizontal node cost C_{nh}. Also, when slope angle θ crosses its critical angle whether in negative Cr_{n}, or in positive Cr_{p}, direction, mobile robot will not be able to climb or descend the slope, hence its cost C_{n} will become infinite (for ideal case), for this purpose we considered a very high cost for this, 10000. Considering $\alpha = 1$ for this paper, from (3) we have, $C_{\mathrm{nh}} = 1$ for $s \in S_{\mathrm{ndia}}$ and $C_{\mathrm{nh}} = 1.414$ for $s \in S_{\mathrm{dia}}$, so final total node cost function will be

$$C_{\mathrm{n}}(s, s_{\mathrm{next}}) = \begin{cases} 0 & \text{if } Cr_{\mathrm{n}} < \theta < Cr_{\mathrm{p}} \ \& \\ & s_{\mathrm{next}} = s_{\mathrm{goal}} \\ 1 + \tan\theta & \text{if } Cr_{\mathrm{n}} < \theta < Cr_{\mathrm{p}} \ \& \\ & s_{\mathrm{next}} \in S_{\mathrm{ndia}} \\ 1.414(1 + \tan\theta) & \text{if } Cr_{\mathrm{n}} < \theta < Cr_{\mathrm{p}} \ \& \\ & s_{\mathrm{next}} \in S_{\mathrm{dia}} \\ \infty \text{ or } 10000 & \text{if } Cr_{\mathrm{n}} \geq \theta \geq Cr_{\mathrm{p}} \ \& \\ & s_{\mathrm{next}} \in S_{\mathrm{obs}} \end{cases} \tag{10}$$

Above node cost function is used in the implementation of the algorithm in this paper.

4. IMPLEMENTATION

Once workspace is generated and cost calculation has been formulated, we can begin implementing algorithm. As an input we have S_{slope}, a set of nodes which contains slope nodes. Considering two neighbouring nodes s_{i} and $s_{\mathrm{i+1}}$ having a slope of θ, where $s_{\mathrm{i}} \in S_{\mathrm{slope}}$ lies in lower level or nodes with slope, and $s_{\mathrm{i+1}}$ lies in upper level or nodes with obstacle, then a set $S_{\mathrm{slopeAngle}}$, of all such slope node and angle having $\{s_{\mathrm{i}}, s_{\mathrm{i+1}}, \theta\}$ as an element is,

$$S_{\mathrm{slopeAngle}}(i) = (s_{\mathrm{i}}, s_{\mathrm{i+1}}, \theta) \tag{11}$$

where θ is bounded in ($\mathrm{Cr_n}$, $\mathrm{Cr_p}$).

As discussed so far, algorithm is developed in two parts, Algorithm 1, "Workspace scan", and Algorithm 2, "Path extraction from the *finalSet*", where *finalSet* is the output from Algorithm 1 and input to Algorithm 2 and contains nodes in s and their neighbouring node along with the cost of traveling to neighbouring node, from start node.

4.1 Algorithm 1. Workspace scan

For a given data of obstacles and slope inclination with ground, in the environment, algorithm 1 generates a set of the data, $finalSet$, containing two neighbouring nodes, s and s_{next}, and the cost of traveling to the node, $\mathrm{s_{next}}$ from start point, $\mathrm{s_{start}}$. Cost to travel to the node is determined using (10).

In algorithm 1, first, indexes and $finalSet$ is initialized, line 1- 4. Then a loop is created, till all the nodes in S are explored, line 5, or the goal point is detected by the algorithm, if the goal is detected then algorithm 2 is executed, line 18. In a loop two variables are created, s and cost, to store current location and cost of traveling from s_{start} to s respectively, line 5-6. The current node, s, is checked for its existence in *closeSet*, line 9. A *closeSet* contains all the explored nodes. Neighbouring nodes of current node are explored, checked for slope inclination, line 11-12, where Cr_{n} is a negative limit and, Cr_{p} is a positive limit of θ. Neighbouring nodes and their cost are updated in $finalSet$, line 13-15. And lastly *closeSet* is updated with explored, line 22-23.

Algorithm 1 Workspace scan

```
1: i ← 1
2: j ← 1
3: k ← 1
4: finalSet[1] ← [s_start, s_start, 0]
5: while workspace is unexplored do
6:     s ← finalSet[k][2]
7:     cost ← finalSet[k][3]
8:     k ← k + 1
9:     if s ∉ closeSet then
10:        for s_next in neighbour of s do
11:            if s_next ∉ closeSet then
12:                if θ ∈ (Cr_n, Cr_p) [1] then
13:                    costSum = C_n(s, s_next) + cost
14:                    finalSet[i] ← [s, s_next, costSum]
15:                    i ← i + 1
16:                end if
17:            end if
18:            if s_next == s_goal then
19:                goto Path
20:            end if
21:        end for
22:        closeSet[j] ← s
23:        j ← j + 1
24:    end if
25: end while
```

Algorithm 2 Path extraction from $finalSet$

```
1: Label Path
2: if s_next == s_goal then
3: P_min[1] ← finalSet[end][2]
4: P_min[2] ← finalSet[end][1]
5: n ← 3
6: s_next ← finalSet[end][1]
7:     while s_next ≠ s_start do
8: Cl ← 10000
9:         for K in finalSet do
10:            if K[2] == s_next & K[3] < Cl then
11: P_min[n] ← K[1]
12: s_next ← K[1]
13: Cl ← K[3]
14: n ← n + 1
15:            end if
16:        end for
17:    end while
18: end if
```

4.2 Algorithm 2. Path extraction from finalset

Algorithm 2 extracts the shortest path from $finalSet$, by comparing cost data in the set. First it checks if path exists, line 2, then last data nodes, goal point and its neighbour is kept in a buffer set, P_{min}. Line 3-4, then the neighbour of goal point is taken into a buffer, line 6. Then its neighbours are checked for lowest cost, and node with lowest cost is stored into P_{min}, line 11. This node is also stored into a buffer with its cost, line 12-14 and is repeated until goal point is reached, line 7. After completion of the algorithm 2, set P_{min} contains the shortest path between start point and goal point.

5. RESULTS

Above algorithms were implemented in Matlab. An environment with obstacles, slopes, goal and start point was created, in different dimensions. As shown in Fig. 5, black blocks are obstacles, green circle, (4,4) is the goal, red circle (14,13) is the start point, green blocks with a red line are the slope and the line shows with which obstacle they are inclined to, and red star shows the direction of the path between start and goal point. To test the algorithm different test cases were conducted with different environment size (5X5, 10X10, 50X50, 100X100 and 150X150), different distribution of obstacles (random), and different slope's angle and direction. Few of the test cases are presented in Table 1. Two of these cases are shown in Fig. 5 and Fig. 6. As seen from the figures planner was successfully able to find a path over the obstacles.

Table 1. Results of the test cases, time taken by the algorithm to produce the path.

Size	Placement	Scan time (s)	Extraction time (s)
15 X 15	Fig. 5	0.2859	0.0092
15 X 15	Fig. 6	0.1959	0.0040
50 X 50	On diagonal	4.900	0.0380
100 X 100	On diagonal	54.69	0.246

In Table 1 goal and start points are kept on the diagonal. with no obstacle, to estimate the worst-case time.

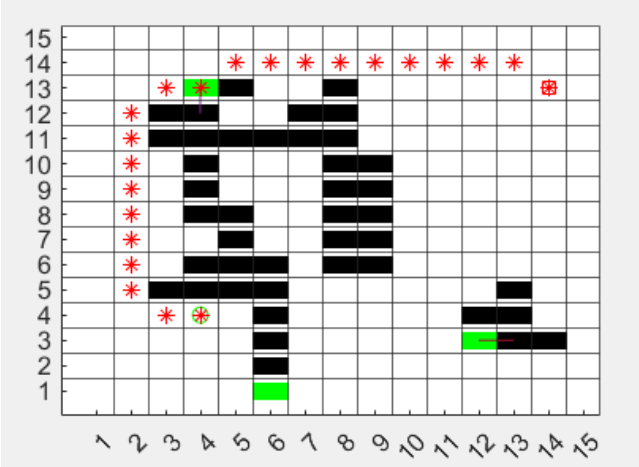

Fig. 5. Test 1, result after the test of 15 X 15 environment, red circle at (14, 13) is start point, green circle at (4, 4) is goal point, black blocks are obstacle, green block is the slope, red line shows with which obstacle slope is inclined and the red star shows the path.

A comparison of different paths and their *total cost* for test case, in Fig. 5, is tabulated in Table 2 and are shown in Fig 7. For Fig. 6 only one path is possible as in the figure. Total cost is based on the (10). For the test in Fig. 5 and Fig. 6 all the inclined slopes are of 45 degrees. For the calculation of total cost θ in (10) was consided as 45 degrees, which gives us cost of path red, yellow and green in Fig. 7 as 22.242, 28, 24.414 and red path in Fig. 6 as 31.21. In Fig. 7 the actual path generated by the algorithm was red path which has the lowest cost.

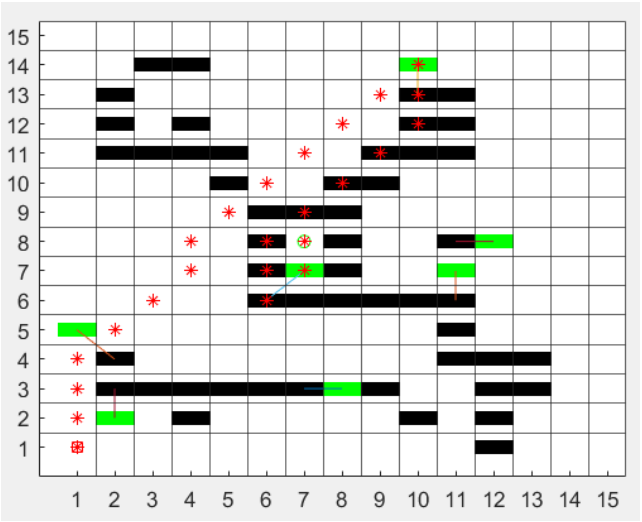

Fig. 6. Test 2, algorithm generating path on obstacles.

Table 2. Comparison of the different paths available for the robot to travel, based on their cost function

Test case in	Path	total cost
Fig. 7	red	22.242
Fig. 7	yellow	28
Fig. 7	green	24.414
Fig. 6	red	31.21

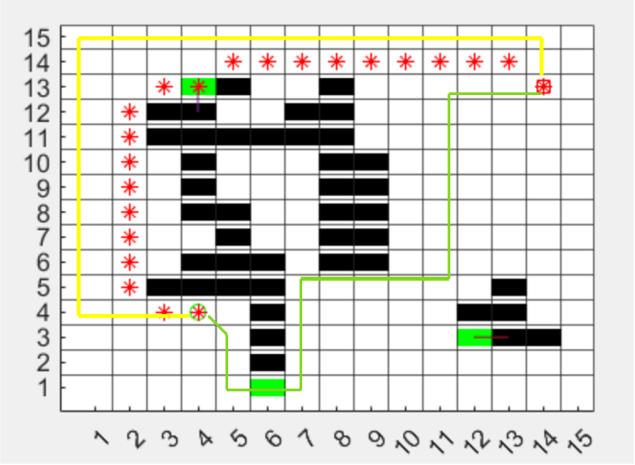

Fig. 7. Different possible paths for test case in Fig. 5

6. CONCLUSION

As seen in the results, the proposed method could successfully reach the goal point with minimum cost function. By utilizing a sloped path which has a less cost function as calculated from (10), algorithm was able to determine a less costly path as compared to another path present in workspace. Since the cost function for the sloped node is proportional to the angle of inclination ($tan(\theta)$) more will be the cost, making it harder for the planner to choose that path.

Next step to this paper is to optimize the program and algorithm to produce the results in much less time and to extend these algorithms to D star (D* algorithm is much close to a real-life situation, in which environment keeps

changing and use much less computation time than all other replanner algorithms.). So that the dynamic change in the environment can be considered by the algorithm to generate a new local path, rather than running same algorithm again for whole workspace.

REFERENCES

Carsten, J., Ferguson, D. and Stentz, A. (2006) 3D Field D: Improved Path Planning and Replanning in Three Dimensions, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3381-2286

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. S.(2005) *Principles of robot motion: theory, algorithms, and implementations*, MIT Press,2005,5

Ergezer, H. and Leblebicioglu, M. K. (2013) 3D path planning for unmanned aerial vehicles, *IEEE Signal Processing and Communications Applications Conference*, 1-4

Gajjar, S., Bhadani, J. and Rastogi, N. (2017) Complete coverage path planning algorithm for known 2d environment, *IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology*, 963-967

Gu, J. and Cao. Q. (2011) Path planning for mobile robot in a 2.5-dimensional grid-based map, *Industrial Robot: An International Journal*, Vol. 38 **3**, 315-321

Hart, P., Nilsson, N. & Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions On Systems Science And Cybernetics*, **4**, 100-107

Li, M. and Zhang, H. (2020) AUV 3D Path Planning Based On A* Algorithm, *Chinese Automation Congress*, 11-16

Stentz, A. 1994 *The D* algorithm for real-time planning of optimal traverses*

Vonasek, V., Faigl, J., Krajník, T. & Přeučil, L. RRT-path – A guided rapidly exploring random tree. 2009,12

Wenzheng, L., Junjun, L. and Shunli, Y. (2019) An Improved Dijkstra's Algorithm for Shortest Path Planning on 2D Grid Maps, *IEEE International Conference on Electronics Information and Emergency Communication*, 438-441

Wu, F., Cao, L., Guo, S. and Zhang, J. (2020) Design and Application of 3D Path Planning Model of Agricultural Automatic Robot, *International Conference on Computer Information and Big Data Applications*, 408-411

Yang, Y., Jiang, M. and Li, W. (2020) 2D Path Planning by Lion Swarm Optimization, *IEEE International Conference on Robotics and Automation Sciences*, 117-121

Zhang, C., Wang, J., Li, J. and Yan, M. (2017) 2D Map Building and Path Planning Based on LiDAR, *IEEE International Conference on Information Science and Control Engineering*, 783-787