Reliable Path Finding Technique for Mobile Robot

RAMA KANTA CHOUDHURY[1]
CHANDRA KANTA SAMAL[2]

M.A.I.T, IP University, New Delhi, India
AND College, Delhi University, New Delhi, India
[1]rkchoudhury1@gmail.com
[2]cksamal@gmail.com

Abstract - Path planning techniques of mobile robot (Automated Vehicle) is discussed in this paper. Though different researchers had proposed different path planning strategies, each plan has its own advantages and disadvantages. The goal of the research work is to develop an algorithm to find out an optimal path from source to destination along with the obstacles. The path planning algorithm not only minimizes the risk of collision but also reduces the planning time and creates a reliable path to reach the desired destination avoiding obstacles. The proposed algorithm is implemented to get the reliable path and compared with that of the existing algorithm to find the optimized path. The new approach is able to minimize the risk of collisions and travelling time with the help of different parameters and simulation software. It is proved through experimental results that the performance of the proposed algorithm is improves considerably and works efficiently when the shape and size of the image changes. It also turns closely at the corners of the obstacles and also reduces the number of steps without affecting the steps and corners. Time and space complexity analysis for this algorithm is experimentally tested and implemented.

Keywords*:* NFT, Path planning, Time Complexity, Space Complexity, DDA Optimization, Adj*.

(Received October 28th, 2020 / Accepted November 15th, 2020)

1. INTRODUCTION

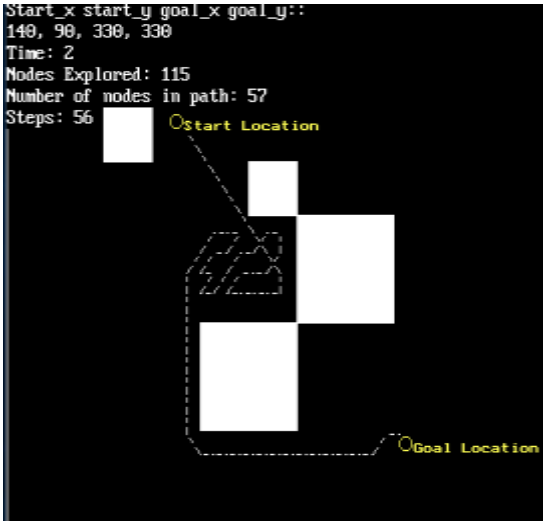
Path planning of a mobile robot is to determine a collision-free path from a starting point to a goal point optimizing a performance criterion such as distance, time or energy (distance being the most commonly adopted criterion). Based on the availability of information about environment, there are two categories of path planning algorithms, namely off-line and on-line. Path planning of robots in environments where complete information about static obstacles and trajectory of moving obstacles are known in advance is known as off-line or global path planning. When complete information about environment is not available in advance, mobile robot gets information through sensors and plans its path as it moves through the environment. This is known as on-line or local path planning. Essentially on-line path planning begins its initial path off-line but switches to on-line mode when it discovers new changes in obstacle scenario commonly used classic algorithms and evolutionary approaches of path planning of mobile robots are discussed. Review shows that optimization algorithms are computationally more efficient and hence are increasingly used in tandem with classic approaches [5].

The path planning algorithm contains various methods with different optimization techniques for optimization. The path planning algorithm developed for various platforms depends on the condition whether it is static or dynamic. Mobile robots are expected to work in many places such as factories, offices and so on. Nowadays, autonomous mobile robots are used in the environment where human beings are working together with robots. Since there are many stationary/moving obstacles in these environments, autonomous mobile robots should plan their own path to avoid not only stationary obstacles but also moving ones such as human workers and other robots. There are various methods available for path planning in the field of robotics, but planning or finding a path which is collision free, shortest and optimal is recent

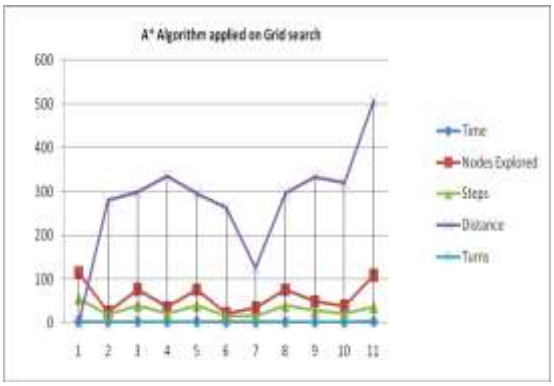
requirement for a robot in the field of robotics. Much of the work has been discovered for generating path in static environment where the obstacle in the environment are stationary But according to today's scenario it should be clear that a robot has to find a path up to the target efficiently when there are moving obstacles present in the environment. The efficiency of the algorithms is analyzed with space and time complexities, completeness and optimality with the help of various parameters. In this paper, A*, NFT algorithm are tested taking various parameters like steps, time, turns, nodes and turns for same shaped and different shaped obstacles[8,11,12].

2. A* SEARCH ALGORITHM

A* Search algorithm is one of the best and popular techniques used in path-finding and graph traversals. Informally speaking, A* Search algorithms [2], unlike other traversal techniques, have "brains". It is really a smart algorithm which makes it different from other conventional algorithms. It is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation). The analysis is shown in Fig.1(a,b) and Table.1.



(a)



(b)

Fig .1(a, b): Complexity analysis Path Planning of A* without optimization

Complexity analysis						
Start Point	Goal Point	Time	Nodes Explored	Steps	Turns	Distance
140, 90	330, 330	2	115	56	18	800.42043
90, 390	140 ,90	3	25	19	5	299.404301
130, 130	330, 330	2	77	41	10	559.010752
90, 380	240, 90	3	36	21	6	343.010752
150, 150	350, 350	2	76	40	13	544.010752
90, 390	130, 130	3	21	16	5	250.803236
130, 130	130, 380	2	35	16	0	240
150, 152	350, 350	3	75	41	13	544.010752
120, 350	260, 100	2	49	29	12	423.409677
80, 350	260, 80	2	39	21	8	338.61828
300, 120	120, 380	3	109	37	18	483.611828

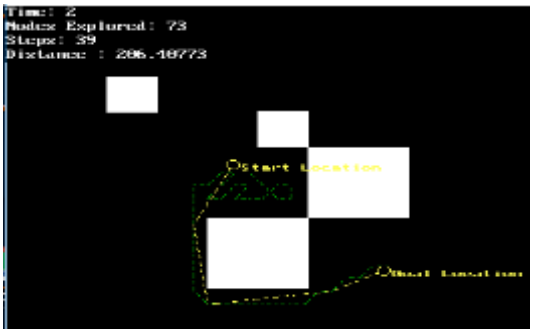
Table.1: Analysis for A* Algorithm without

Complexity analysis						
Start Point	Goal Point	Time	Nodes Explored	Steps	Distance	Turns
140, 90	330, 330	3	115	56	4.06. 83041	5
90, 390	140 ,90	3	25	19	279. 508484	1
130, 130	330, 330	3	77	41	298. 737732	3
90, 380	240, 90	3	36	21	336. 4609	3
150, 150	350, 350	3	76	40	295. 853668	3
90, 390	130, 130	2	21	16	263. 058929	1
130, 130	130, 380	2	35	16	125	1
150, 152	350, 350	2	76	40	295. 853668	3
120, 350	260, 100	2	49	29	334. 588684	3
80, 350	260, 80	2	39	21	320. 61417	3
300, 120	120, 380	3	109	37	507 .457092	5

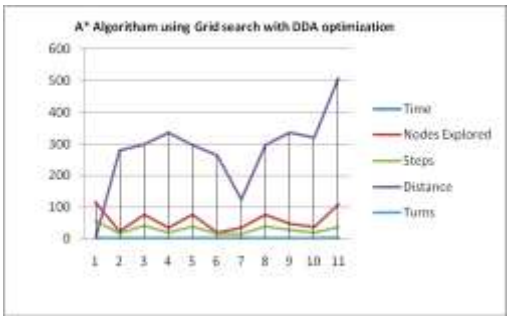
optimization for square shaped images

2.1 DDA optimization used on square shaped images

DDA optimization technique is applied on square shaped images using A*algorithm. The time and space complexity is tested using DDA optimization. Here the efficiency and the effectiveness are evaluated in four steps which are tabulated in Table 2. The simulated result and its graphical analysis is shown in Fig.2 (a,b).



(a)



(b)

Fig .2 (a,b) : Complexity analysis with DDA optimization of A* with optimization

Table.2: Analysis for A* Algorithm with DDA Optimization for square shaped images

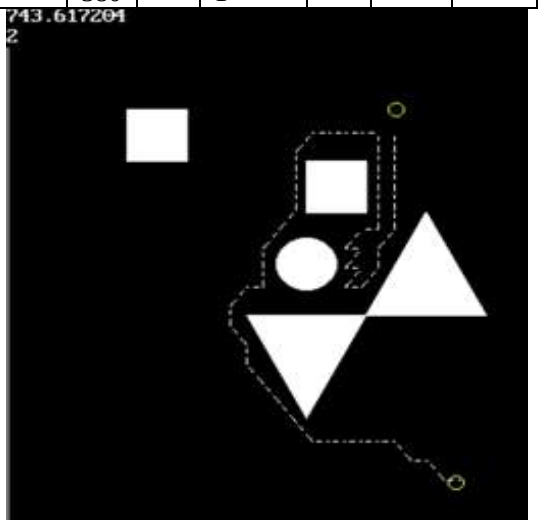
2.2 Path planning with different shaped images

As we know, A* algorithm works effectively with square shaped images, a step ahead is taken by introducing different shaped images like circles, rectangles and triangles to ascertain effectiveness in image processing and path planning of A* algorithm. The simulated result along with the graphical analysis is presented here as shown in Table.3 and Fig.3(a, b).

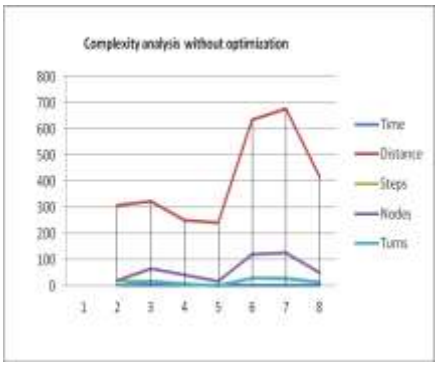
Complexity analysis						
Start Point	Goal Point	Time	Distance	Steps	Nodes	Turns
90, 390	140 ,90	3	308. 113892	19	20	1
90, 380	240, 90	3	324. 409668	17	65	18
90, 390	130, 130	2	250. 803223	6	41	7
130, 130	130, 380	2	240	1	17	0
120, 350	260, 100	2	634. 415039	30	119	29
80, 350	260, 80	3	678. 813965	28	126	27
300, 120	120, 380	3	416. 212891	13	51	13

Table .3: Analysis for A* Algorithm without optimization for different shaped images

Complexity analysis						
Start Point	Goal Point	Time	Distance	Steps	Nodes	Turns
90, 390	140 ,90	3	308.113892	2	20	1
90, 380	240 , 90	3	222.446198	2	65	2
90, 390	130 , 130	2	106.25441	6	41	7
130, 130	130 , 380	2	235	1	17	0
120, 350	260 , 100	2	664.122559	6	119	6
80, 350	260 , 80	3	726.091248	6	126	6
300, 120	120 , 380	3	416.212891	13	51	13



(a)



(b)

Fig .3(a, b): Complexity analysis without optimization for different shaped images for A*

2.3 DDA optimization used on different shaped images

Using A* algorithm, the DDA optimization was applied on images of different shapes through grid search. The simulation outcome is shown in Fig.4 and the graphical and tabular analysis is presented in Fig.4 and Table.4 respectively. Based on these tables the parameters of time and space complexity and its effectiveness are analyzed.

Table. 4: Analysis for A* Algorithm with DDA optimization for different shaped images

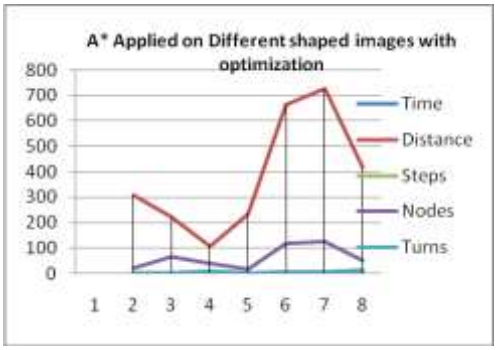
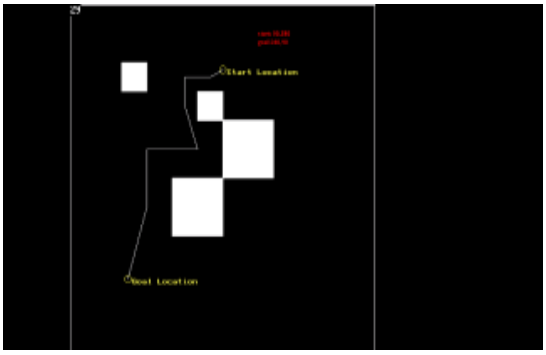


Fig .4 : Complexity analyses with DDA optimization for different shaped images

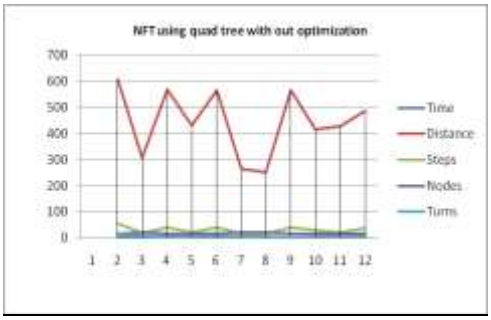
2.4 NFT algorithm on square shaped images without optimization

A NFT algorithm is tested here which is very popular in image processing. The complexity analysis for NFT algorithm is presented. The efficiency of the searched algorithm can be evaluated in four steps which are tabulated in Table 5. The output of the NFT algorithm and its graphical analysis is shown in Fig.5(a,b) .The graphical analysis is shown in Fig.5(a,b). The green and white line shows the un-optimized path

for NFT algorithm [3,5].



(a)



(b)

Fig .5 (a, b): NFT with Square shaped Images without Optimization

Complexity analysis						
Start Point	GoalPoint	Time	Distance	Steps	Nodes	Turns
140, 90	330, 330	4	606.66217	56	16	16
90, 390	140,90	3	307.263092	19	20	7
130, 130	330, 330	3	567.652405	41	17	4
90, 380	240, 90	3	430.009308	21	17	6
150, 150	350, 350	3	565.201	40	17	7
90, 390	130, 130	3	264.728241	16	21	6
130, 130	130, 380	3	251.209427	16	21	4
150, 152	350, 350	3	565.201294	41	17	2
120, 350	260, 100	3	413.245544	29	17	5
80, 350	260, 80	3	428.5895	21	17	5
300, 120	120, 380	3	485.7	37	17	6

Table.5: Analysis for NFT Algorithm without optimization for square shaped images

2.4.1 NFT algorithm on square shaped images with DDA optimization

A DDA optimization algorithm, applied on NFT is tested on square shaped images. The complexity analysis for NFT algorithm is presented. The efficiency of the algorithm can be evaluated in four steps which are tabulated in Table.6. The output of the NFT algorithm and its graphical analysis is shown in Fig.6 .The yellow line shows the optimized path for NFT algorithm.

Complexity analysis						
Start Point	Goal Point	Time	Steps	Nodes	Distances	Turns
140, 90	330, 330	4	8	16	403.181763	2
90, 390	140, 90	4	2	20	279.508484	2
130, 130	330, 330	3	2	17	362.4645	2
90, 380	240, 90	3	2	17	326.156311	2
150, 150	350, 350	3	2	17	363.6339	2
90, 390	130, 130	3	3	21	263.058929	0
130, 130	130, 380	3	1	21	125	0
150, 152	350, 350	3	2	17	363.633942	2
120, 350	260, 100	3	2	17	312.856	2
80, 350	260, 80	3	2	17	321.01	2
300, 120	120, 380	3	2	17	288.046	2

Table .6 : Analysis for NFT Algorithm with DDA optimization for square shaped images

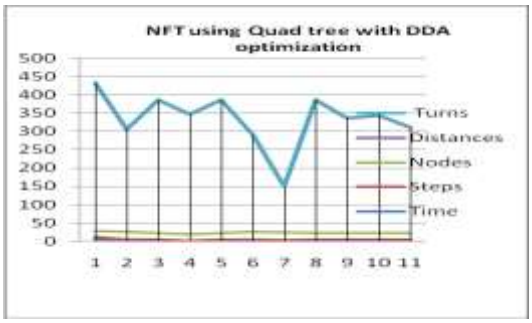


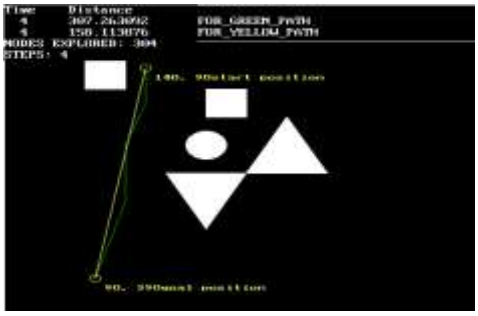
Fig .6: NFT using quad tree with DDA Optimization

2.4.2 NFT algorithm on different shaped images without optimization

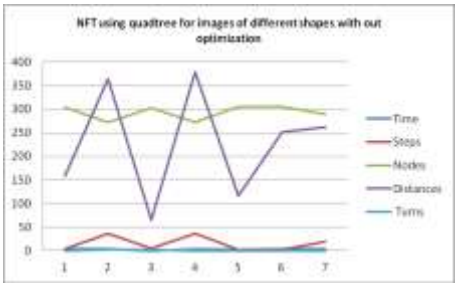
Here the algorithm is tested and simulated when the obstacles are different shaped images like circle, rectangle and square in a similar environment. The white blocks as shown in Fig.7 (a, b) are the images of different shapes, whereas green line is the un-optimized path of the robot. The space complexity is analyzed and presented in Table.7 and the graphical analysis is shown in Fig.7.

Complexity Analysis						
Start Point	Goal Point	Time	Steps	Nodes	Distances	Turns
90, 390	140 , 90	4	4	304	307.263092	3
130, 130	330, 330	4	36	272	553.510986	18
230, 230	240, 90	3	5	303	160.622574	19
150, 150	350, 350	4	36	272	569.356995	2
90, 390	130, 130	3	3	305	264.72821	2
130, 130	130, 380	4	3	305	251.209427	2
220, 380	160, 160	4	19	289	360.796722	2

Table .7: Analysis for NFT Algorithm without Optimization for Different Shaped Images



(a)



(b)

Fig 7 (a, b) : Complexity analysis NFT with Different shaped Images without Optimization (Green)

2.4.3 NFT algorithm on different shaped images with DDA optimization

Here DDA optimization technique with NFT algorithm is applied on different shaped images. The yellow line in Fig.7 and Fig.8(a, b) is the optimized path of the NFT algorithm. The DDA algorithm applied on images of different shapes is tested and the complexity analysis is carried out and shown in Table.8 and Fig.8.

Table.8: Analysis for NFT Algorithm with DDA optimization for Different shaped images

Complexity analysis						
Start Point	Goal Point	Time	Steps	Nodes	Distances	Turns
90, 390	140, 90	4	4	304	158.113876	0
130, 130	330, 330	4	36	272	364.208282	2
230, 230	240 90	3	5	303	65.764732	2
150, 150	350, 350	4	36	272	379.027039	0
90, 390	130, 130	3	3	305	117.046997	0
130, 130	130, 380	4	3	305	251.209427	0
220, 380	160, 160	4	19	289	262.48114	0

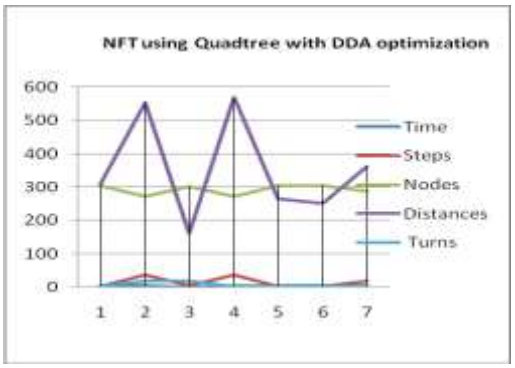


Fig.8: Complexity graph for different shaped images with optimization

3. PROPOSED ADJACENCY FINDING ALGORITHM (ADJ*)

It is imperative for a mobile robot to carry out tasks with utmost dedication and achieve the target by arriving at the destination.. The journey from its initial point to the target location should be seamless without any obstructions. For this reason a rigorous analysis of a host of other algorithms is undertaken before propounding an alternative or a new one. After assessing different algorithms i.e. A* and NFT on the basis of their performance, simulated results and applications with respect to time and space complexities, it was found that they were not completely flawless. The A* algorithm performed immaculately so long as the images were square shaped but with the slightest change in the shape of the image, the algorithm failed in optimum adaptation. The application of DDA optimized algorithm on the A* and the final outcome stand as a testimony to this. Apart from the inability to adapt with the changing shape of the image, it was also realized that A*consumed much time and space. We have analyzed the space and time complexity which has been concretized in tabular form and as well as graphically. Similar testing was undertaken with the NFT algorithm. Both square shaped images and irregular shaped images in the same environment were taken into account. The time and space complexity of NFT algorithm were also analyzed [3]. We arrived at the conclusion that the algorithm worked efficiently amidst square shaped images than irregular shaped images. The application of DDA optimization on NFT was carried out with the conclusion that, we need to try to improve the efficiency and effectiveness with space complexity on the basis of the parameters like time, distance, steps, nodes and turns. The emphasis was basically on time, distance and turns as other parameters are dependent on the speed of the processor. The exhaustive study on both A* and NFT algorithms undertaken with their outcomes, are presented in chapter 3 and chapter 4. After the analysis of the above two algorithms, along with its merits and demerits, we came up with a proposal of ADJ* algorithm on the basis of time and space complexity. The ADJ* algorithm was tested with similar and different shaped images. The simulated result in both graphical and tabular forms was put forward [9].

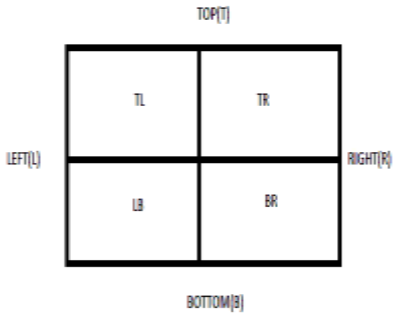
3.1 ADJ* using quadtree search

Image representation is highly essential for image processing. Region representation is an appropriate part of image processing. Neighbor finding is an important task to be performed on an image. It is difficult to find neighbors using the quad tree representation. It is very simple to find neighbors using matrix representation. In most cases, a block can have more than one neighbor in a given direction; the problem is then to find its neighboring blocks, in all directions, all relying solely on the quad tree structure of the image.

3.2 ADJ* notation and its application

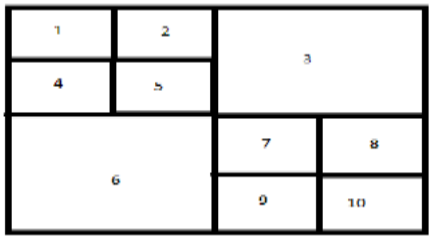
The quad tree initially came up with a strategy of subdividing images into quadrants, where same size and different size images are represented as

obstacles. This approach is based on a data structure in the form of a tree with multiple nodes called quad tree in which every single node comprises of four children. Fundamentally quad trees are two dimensional analog of octrees [9]. They are used to compartmentalize or divide a two dimensional space into four quadrants or specific areas repeatedly without halt. A quad tree has locations called leaves where records are stored at the tip of the leaves. The leaf fringe serve as the end points of data stored and the branch points serve as nodes. The number of branches or children forms the order of the quad tree. So the order of a quad tree is 4 as it has four siblings in each node which means that leaves in a quad tree is undoubtedly a power of 4. The depth of the quad tree is defined as the number of access operations needed to obtain the desired record. It is an approach where the complete tree is represented as a square and the square is divided into four sub-branches of equal size as shown in Fig.9(a). The row1 and column1 is represented as (R1C1) and row1column2 as (R1C2). (R2C1) and (R2C2) are shown in Fig 9(b). The image is divided into four quadrants each node has four children, each node is represented with (R1C1, R1C2, R2C1, R2C2). The location of the image is presented in Fig.9(c) and Fig.9(d).



(a)

(b)



(c)

Fig. 9: **(a)** Representation of image with notation **(b)** Numbering the image **(c)** Location of the position of an image **(d)** Numbering the image as per the quad tree.

The quad tree is a well organized composite configuration which helps to represent the image in a better way. The extra space that is required will be reduced and optimized path will be found. In this quad tree approach the image is first imposed in the square and is checked whether the image completely occupies the square or not, if not then the square is further sub-divided into sub-squares until and unless the image completely comes inside a square. Once it comes completely inside a square then it treated as "BLACK" otherwise it is treated as "GRAY"[9]. The black and white nodes are treated as the leaf nodes [9] whereas the "GRAY" node is further subdivided into sub-squares. The image along with its obstacle is presented in Fig 10, which illustrates the embedding of the image in the quadtree in BLACK node. The notation of the tree structure is represented as R1C1, R1C2, R2C1 and R2C2.

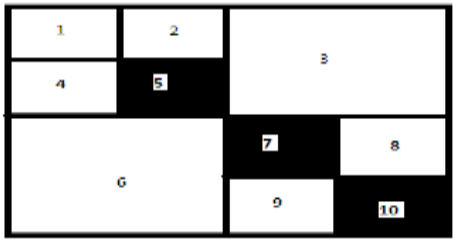


Fig .10: Complete region with obstacle

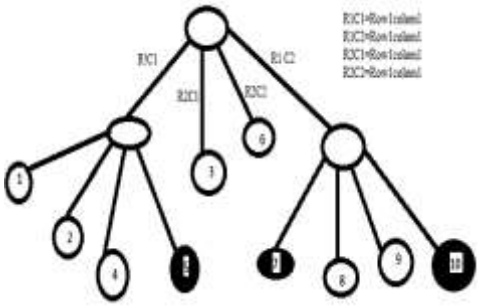


Fig. 11: Quad tree representation

Here, we assume that each node is stored with six fields, out of which first five fields contain pointers to the main field and its four sub fields, called as branches. In a Quad tree representation if M is representation M is the main node and it has four sub nodes then the main node represented as M node and Q is the branch, it will be treated as MAIN(M) and branch node will be represented as SUBFIELD(M,Q) that is M is main node and Q is its subfield node[6]. We can treat relative branches to its main by using subfield as SUBFIELD (M) having value of Q. if the block contains image which is within the node, then it will be treated as image block and is represented in "BLACK" with logic "1" and if the block does not contain any image it is represented in :WHITE" with logic "0" and "GRAY" zero or one. BLACK and WHITE are represented as terminal or leaf nodes whereas GRAY are non-terminal nodes. Let the four sides of a node be called as R1C1, R1C2, R2C1, R2C2. We use some functions involving certain quadrants and boundaries. ADJACENCY(R,S) is true, only if the branches S are adjacent to boundary R of the node block. REFLECTOR(R,S) represents the SONTYPE having block of equal size that is the nearest side of R(right) of the block having SONTYPE_VALUE 'S'. Example: REFLECTOR (T, BL) =TL, SAMEBOUNDARY(R1,R2) represents that block R1 and R2 are not adjacent blocks i.e. TR and BL. The detail is shown in Fig.11. The working of the algorithm is shown in Fig.12.

Adjacency	TL	TR	BL	BR
T(Top)	T	T	F	F
B(Bottom)	F	T	F	T
L(Left)	F	F	T	T
R(Right)	T	F	T	F

Table 1 Adjacency

Mirror	TL	TR	BL	BR
T(TOP)	BL	BR	BR	BL
B(Bottom)	TR	TL	F	T
L(Left)	BL	BR	TL	TR
R(Right)	TR	TL	BR	BL

Table 2 Reflector

Opposite Block	Block
TL	BR
TR	BL
BL	NE
BR	TL

Table 3 Opposite block

Block1 to Block2	TL	TR	BL	BR
TL	NO	T	L	NO
TR	T	NO	NO	R
BL	U	NO	NO	B
BR	NO	E	S	NO

Table 4 Same boundary

Table.9 : Boundaries and its Notation

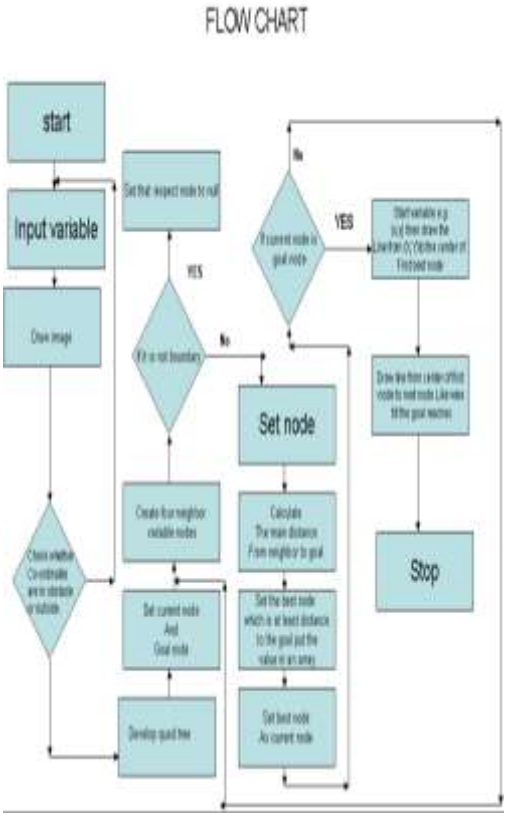


Fig .12: Flow chart for ADJ* Algorithm

3.3 Path planning and path coordination

Line1: Let the tree of an image be denoted by T. Each node of the tree has four branches branch 1, branch 2, branch 3 and branch 4.
Line2: There are some functions which are used for following functions.
Line3: Find_adjacency (Node, Direction)//Find the adjacency of the given node in given direction.
Line4: Find_small_adjacency (node) //Find small adjacency of given node.
Line5: Having common side (int m_1, int n_1, int m_2, int n_2) determines if two nodes are having a common side.
Line6: Assume that the tree is built and the current node is the one in which start point resides.
Line7.
do
begin
nn= find_ adjacency(current_node, TOP);
en= find_adjacency (current_node, RIGHT);
sn= find_adjacency(current_node, Bottom);
ww = find_adjacency(current_node,LEFT);
Line8: check if nn, en, sn, wn lie beyond the boundary of the map. If yes, delete such nodes else, begin to check if they contain smaller adjacent nodes in them
find_small_adjacency(nn);
find_small_adjacency(en);
find_small_adjacency(sn);

find_small_adjacency(wn);
end;
add all the adjacent nodes of equal or greater sizes to a list of neighbours.
Then find the path.
find_path( );
Line9: Repeat the above steps until current _node becomes the goal node.
Line10: find_path( ) finds the shortest path between start and goal node..
Line11: Shortening the path founded by Quad tree.
Let the points found by Quad tree algorithm be stored in an array P1 whose length is n.
A.From first element in P1 to third element check if a free or straight path exists which doesn't pass through an obstacle. If yes, remove the nodes in between and allow a new path to be formed between the first and the third node.
Repeat the process between the second element of this new array and third element.
B. This gets the path complexity O (n^2) in time
Space n=O(n) space.
Line12:
Let the path found by the quad tree algorithm be stored in any array P1 and let P1 contains n points.
A. Check the first element in the P1 to third element if a free path exists, if yes then store in a variable "Var"("var " initially contain 2nd "var").
B. Check from the first element to 4^{th} element in P1 if there is a path, then overwrite "Var" with 4^{th} element.
C. Repeat the process until all the elements are scanned in such a way.
D. Let the k^{th} element be stored in "var". Store it in a new array P(new)[ ], now repeat the above four steps from k^{th} variable, if the k^{th} variable is $(n=1)^{th}$ element of p[ ] then add two points in P(new).
E. At last join the points and get the optimal path complexity O (n^2) times and
space n = O(n).
begin
For i = 0: n-2
begin j=2: n
begin
if (path_bw(a[i],a[j]);
begin
var = a[j]
end;
end;
b[++counter]=var;
end
b[++counter]=a[n-1]
end;
A. Steps for same_side_adjacency**(**U,V)
Identify a same adjacency of node P in x-axis or y-axis direction
Line1: get node(U);
Line2: get direction(V);
Line3: return (son(If (BSD (V, son type(U) then same_side_adjacency (PARENT(U) ,V);
else PARENT(U);opposite (T(V , SONTYPE(U));
end;
B. STEPS FOR SAME_SIZE_ADJACENCY R(U,V)
Line1: Identify an adjacency of node P in x-axis and y-axis direction D, if the node is not present then return NULL.
begin
nodeU;
directionV;
node Q;
if not NULL(PARENT(U)) AND BSD (V,SONTYPE(U))
then find a common ancestor
Q <- SAME_CORNER_ADJACENCY(PARENT (U) ,V)
else
Q <- PARENT (U);
Follow opposite path to identify the adjacency.
return (if (not NULL(Q,REFLECT(d, SONTYPE(U))else Q;)));

3.4 Adjacency finding algorithm (ADJ*)

Adjacency Algorithm(ADJ*) is an incremental algorithm which finds shortest path between the goal and start nodes. The map on which the algorithm worked conventionally was the grid which consumed time in finding the shortest path. It is observed that if the map is represented in form of quad tree the search time is drastically reduced. The complete approach is given below.

1. The boundaries are set first under which the map is defined and the algorithm is followed.
2. It is required that when obstacles are added to this area a quad tree is formed.
3. Assign the root of the quad tree to the empty set in the first step.
4. Add an obstacle by giving the diagonally opposite coordinates, its height and width.
5. Starting from one of the coordinates check if the coordinate lies in the root.
   a. If yes, divide the root into four branches as explained earlier.
   b. Starting from the branch in the top left position check if the coordinate lies in the node.
   c. If yes, divide the node into four branches and repeat the steps again and again till the tree is extended to its maximum level. A suitable integral value is chosen so as to vary the resolution of the path formed.
   d. If no, then check if the coordinates lie in other branches and then repeat the steps. Exit if either the max level is reached or all the branches have been searched.

6. These steps are repeated for the remaining sides of the obstacle and for other obstacles, if there are any.
7. After the quad tree is formed and all the obstacles are added; the valid start and goal locations are taken.
8. Now Adjacency Algorithm can be easily applied to this system as follows:
a. The robot takes start node as the current and enters all the adjacent nodes of same, greater or smaller sizes in a priority queue called OPEN_LIST.
b. Then OPEN_LIST holds the list of all the adjacent nodes in an ascending order of their respective 'f' value,
where 'f' value = distance between the current node and goal node + distance between start and current node.
c. The adjacent node possessing the least f' value is considered the most promising node in progressing the search.
d. From the new current node again the adjacent nodes are taken in the OPEN_LIST and the process is repeated until the current node becomes the goal node.
e. The techniques used to find the adjacent nodes have been described in the above sections.

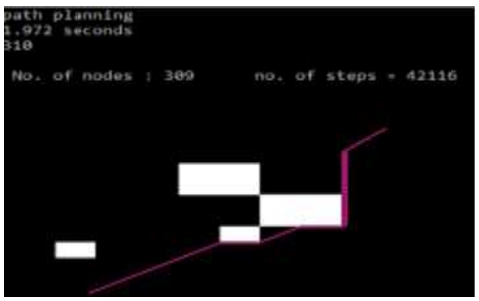
3.5 Simulation results

After an exhaustive analysis of A* and NFT algorithm, we have developed a new algorithm, ADJ* based on complexity. ADJ* algorithm has been implemented and was tested in C^{++} and the result has been illustrated in Table 10 and Table 11 represents a graphical representation of space complexity. To measure the effectiveness of the algorithm the parameters of distance, time, steps, nodes and turns were taken into account. The outcomes of these parameters are shown in Fig.13 (a, b, c).

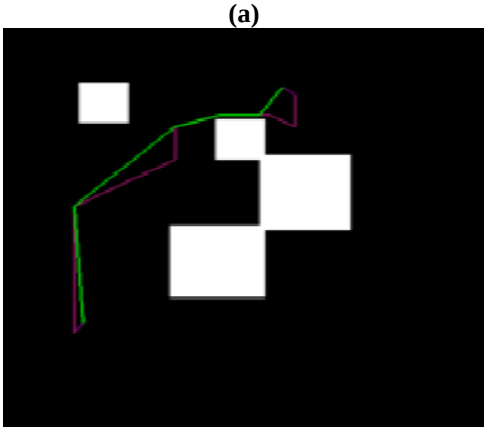
Complexity analysis

Start Point	Goal Point	Time	Steps	Nodes	Distances	Turns
90, 390	140, 90	0.103	575	4	390.104	4
90, 380	240, 90	0.412	4088	5	400.783	5
90, 390	130, 130	0.055	575	4	401.788	4
240, 90	230, 230	0.052	2	6	191.741	4
120, 350	260, 100	0.035	8	7	422.761	6
80, 350	260, 80	0.014	7	6	399.092	6
260, 120	220, 380	0.019	17	17	314.514	11
160, 160	220, 380	0.019	14	16	314.514	11

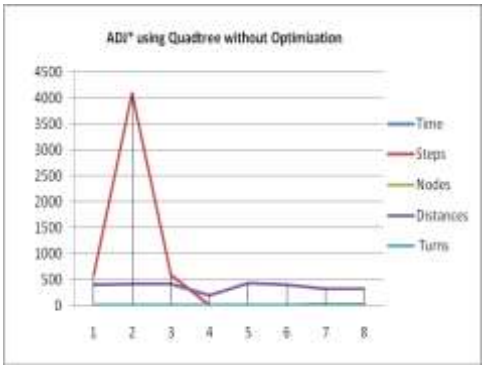
ADJ* algorithm imposed on square shaped images



(a)



(b)



(c)

Fig .13 (a, b, c) Simulation result of ADJ* Algorithm with square shaped Images

Table.10 : Analysis for ADJ* Algorithm without optimization for square shaped images

3.6 Proposed ADJ* algorithm on square shape images with DDA optimization

Table. 11: Analysis for ADJ* Algorithm with DDA optimization for square shaped images

Complexity analysis						
Start Point	Goal Point	Time	Distance	Steps	Nodes	Turns
90, 390	140 , 90	0.103	390.105	575	4	4
90, 390	130, 130	0.055	401.788	575	4	4
220, 380	160,160	0.019	315.514	14	16	11

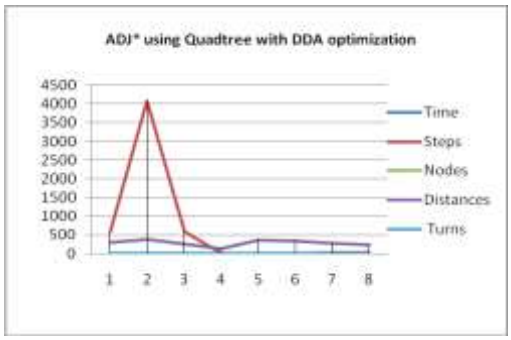


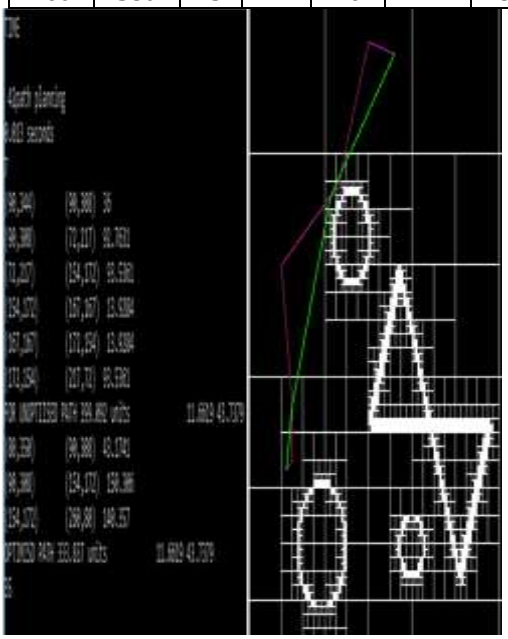
Fig .14: Space Complexity graph of ADJ* Algorithm with optimization

3.7 ADJ* algorithm on different shaped images without optimization

The ADJ* algorithm is also tested for different shaped images. Its output is shown in Fig.15(a, b). The space complexity and effectiveness are tested and analyzed related to time, distance, steps, nodes and turns which are presented in tabular in Table.12 and graphical form in Fig.16.

Table .12: Analysis for ADJ* Algorithm without optimization for Different shaped images

Complexity analysis						
Start Point	Goal Point	Time	Steps	Nodes	Distances	Turns
90, 390	140 , 90	0.103	575	4	304.874	2
90, 380	240, 90	0.412	4088	5	374.871	3
90, 390	130, 130	0.055	575	4	263.708	2
240, 90	230, 230	0.052	2	6	140.572	1
120, 350	260, 100	0.035	8	7	363.562	4
80, 350	260, 80	0.014	7	6	333.837	3
260, 120	220, 380	0.019	17	17	280.678	6
160, 160	220, 380	0.019	14	16	247.214	8



(a) **(b)**

Fig .15 (a, b) : Simulation result without optimization for different shaped images (pink)

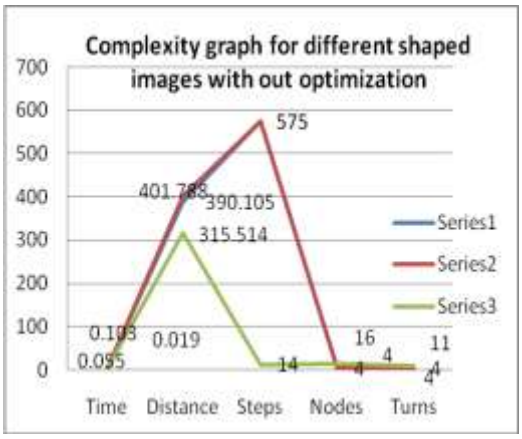


Fig. 16: Complexity graph for ADJ* for different shaped images without DDA optimization

3.8 ADJ* algorithm with DDA optimization for different shaped images

Table 13 : Analysis for ADJ* Algorithm with optimization for Different shaped images

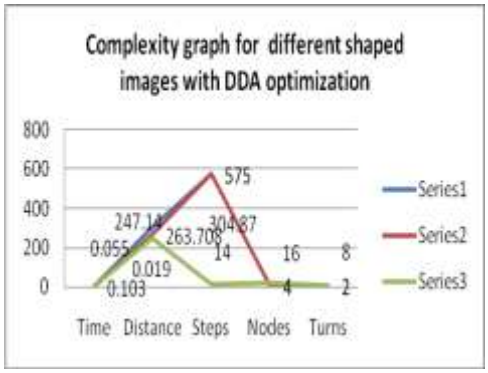


Fig. 17: Complexity graph for ADJ* for different shaped images with DDA optimization

4. CONCLUSION

An effective method to plan a path in a static environment having obstacles for a mobile robot is presented. The potential application of such designed algorithm is also discussed. The proposed algorithm is used on image processing, if the streaming images are coming from the source. It is a faster search algorithm as compared to other existing algorithms. When time and space complexities were calculated, it was found that it took less time as compared to other algorithms. The distance and time graph shows that it takes much less time than other path processing algorithms.

REFERENCES

[1] Dave Ferguson and Anthony Stentz , "Using interpolation to improve path planning: The field D* algorithm", *Journal of Field Robotics*, 23:79–101, 2006.

[2] Dave Ferguson and Anthony Stentz, "Anytime, dynamic planning in high-dimensional search spaces", In *Proc. IEEE International Conference on Robotics and Automation*, pages 1310–1315, 2007.

[3] Kunio Aizawa, Koyo Motomura, Shintaro Kimuru, Ryosuke Kadowaki, and Jia Fan, "Constant Time Nieghbor Finding in Quadtrees", *ISCCSP 2008*, Malta, 12-14, March, 2008.

[4] LaValle, S. M., "Planning Algorithms", *Cambridge University Press*, Cambridge, 2006.

[5] Lee, D.C. , "The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot", *Cambridge University Press*, New York, 1996.

Complexity analysis						
Start Point	Goal Point	Time	Distance	Steps	Nodes	Turns
90, 390	140 , 90	0.103	304 .87	575	4	2
90, 390	130, 130	0.055	263. 708	575	4	2
220, 380	160, 160	0.019	247. 14	14	16	8

[6] Likhachev M. and Koenig S., "Incremental A*", *In Proceedings of the Neural Information Processing Systems*, 2001.

[7] Lozano-Perez and T. Lozano-Perez, "Spatial planning: A configuration approach", *In IEEE Transactions on Computers*, volume C-32, pages 108–120, (1983).

[8] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys", *Acta Inform.*, vol. 4, no. 1, pp. 1–9, 1974.

[9] H. Samet, "Neighbor finding techniques for images represented by quadtrees", *Comput. Graph, Image Process*, vol. 18, no. 1, pp. 37–57, 1982.

[10] I. Gargantini, "An Effective Way to Represent Quadtrees", *Commun. ACM*, vol. 25, no. 12, pp. 905–910, December, 1982.

[11] Cui, S. G., Wang, H., Yang, L., "A Simulation Study of A* Algorithm for Robot Path Planning", *16th international*

conference on mechatronics technology, pp.506-510, 2012.

[12] Akshay Kumar Guruji, Himansh Agarwal, D. K. Parsediya, "Time-Efficient A* Algorithm for Robot Path Planning", *3rd International Conference on Innovations in Automation and Mechatronics Engineering, ICIAME 2016*. Available online at www.sciencedirect.com, Procedia Technology 23 (2016) 144 – 149, 2016.