Improved Route Optimization for Autonomous Ground Vehicle Navigation

Ibrahim Mohammed, Berat A. Erol, Ikram Hussain Mohammed, Patrick J. Benavidez, PhD, Mo Jamshidi, PhD
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249, USA
Email: md.ibrahm@hotmail.com, berat.erol@utsa.edu, ikramhussainmohammed@yahoo.com,
patrick.benavidez@utsa.edu, moj@wacong.org

***Abstract*—Mobile robot navigation is a very important exercise in all robotic application from a domestic household cleaner to highly dangerous life threatening situations. Path planning is the main issue related to navigation. It is very important for an autonomous mobile vehicle to navigate properly without any collision or unsafe conditions in its environment. Path planning in mobile robots must ensure the optimal route with least cost and collision free path. In this paper, an algorithm to reduce the path searching process by half of the traditional methods is proposed.**

***Index Terms*—Autonomous Navigation, Path Planning, Robot Operating System, Differential drive robot.**

I. Introduction

Navigation is an important task for an autonomous robot. The purpose of the task is to navigate from one point to another avoiding collisions with obstacles in the path. Once obstacles are mapped, it is possible to find the safest path to reach desired destination. By the nature of the mobility, the robot should use its resources efficiently; therefore, in addition to reaching the desired location safely, it should use the shortest way to reach there. There is no doubt that one can increase the number of concerns for mobile robotics easily by adding faulty sensor readings, odometry error, and latency to the problem. These several and highly dependent problems turn the robot navigation into a complex process. After the process of mapping, the most general among these problems are localization, path planning and motion control.

Localization is the first problem in the navigation of robot after mapping. Just as the name suggests, localization denotes the robot's ability to establish its own position and orientation within the frame of reference. The problem of motion control focuses on the actual mobility of the robot. After recognizing a destination and finding an ideal route, the robot has to be controlled to move according to the planned path, and reach the goal position. Path planning can be considered as the most important issue among the three, which happens to also be the focus of this paper. After knowing the destination for the robot to reach, a path must be planned in a way that the robot is able to attain the goal without any collisions. Path planning involves selecting and identifying an ideal route for the robot to traverse in the workspace area.

* This work was supported by Grant number FA8750-15-2-0116 from Air Force Research Laboratory and OSD through a contract with North Carolina Agricultural and Technical State University.

A. Path Planning

Path planning has been concentrated broadly in literature. An examination of path planning algorithms demonstrates that the best-first search calculations (A^* search) will keep running on various kinds of setup to determine an ideal path. It is started by the configuration space approach [1]; then, decomposed configuration space into graph of free, blocked and mixed cuboids followed by A^* search algorithm is introduced [2]. Then, for avoiding the collisions during the path, a rectangular shaped robot and polygonal obstacles described [3]–[5]. Later, a parametric expression for contact patches for moving polyhedral obstacles introduced [6].

Path planning has a major part in the navigation of mobile robots, since it enables the selection and identification of an appropriate route in a complex environment. Depending on the application, the workspace area differs, and thereby the problem of uncertainty in various domain arises. The process of determining an ideal path for the robot, within an ideal time, is influenced largely by the components of path planning.

B. Path Planning Algorithm

A generic path planning algorithm is required to meet various criteria as stated in [7]:

- *the resulting paths should have the lowest possible cost to prevent any indirection.*
- *it should be fast, correct and robust, if there is no collisions occurred during the navigation process.*
- *the algorithm should be generic for any scenarios- it should not be fully optimized for a specific map type.*

For instance, visibility graph (*VGraph*) algorithm was combined with the Dijikstras algorithm in [8]. In VGraph algorithm the path consists of straight line segments that are connected by a subset of vertices of obstacles, whereas most of the search algorithm looking for a solution with diagonal lines or connecting the edges. For execution of navigation, three activities must be carried out, which are mapping or modeling the environment, path planning, and driving systems. The

choice of suitable algorithm for every level of path planning process is crucial for navigating the robot successfully.

II. Methodology

A. A^* Search Algorithm

Traditional A^* algorithm, also known as Best-first Search algorithm, combines the advantage of the *least cost* and *greedy search* using a cost function [9]. The A^* search algorithm tries to minimize the function $f(n)$ expressed as:

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

The term $h(n)$ is a heuristic estimation of the remaining expense to get from node n to the objective node, and $g(n)$ indicates the cost from the beginning node to node n. The algorithm maintains two sets of lists throughout the search, which are open and closed lists. The open list stores all the nodes in sequences that are to be searched, and closed list store all the nodes visited by the search algorithm. The least $f(n)$ value is inserted into closed list. Extending a node implies placing it into the closed list, adding the neighbors into the open list, and assessing the cost function. The calculation stops when the objective of node is extended [9].

The decision of a decent heuristic calculation is essential, keeping in mind the end goal to accomplish both quality and proficiency of a search. As long as the heuristic underestimates the genuine cost, the shortest way is ensured; ironically, under estimating can lead to extension of nodes [5]. At the point when the heuristic is permitted to overestimate the distance to the objective, the A^* calculation has a tendency to extend node that lie on the immediate way to the objective before attempting different nodes. However, this can prompt slower approach, if the final way contains directions that lead far from the objective [9].

Despite the fact that it has been widely used, this established algorithm has a *time consuming* issue, since it does not have a heuristic data to handle until it achieve the objective [10]. This will bring a great deal of node being created to locate a shortest way, and to keep away from impediments which will in the long run make it slower [5]. As describe in [11] the A^* Search algorithm can be written as:

where $O$is an open list, or priority queue which includes the nodes that are subject to search at an iteration step, C is the closed list for the visited nodes so far, $Star(n)$ represent the set of nodes that are adjacent to n, and $c(n_1, n_2)$ is the length of the edge connecting n_1 and n_2 nodes.

B. Modified A^* Search Algorithms

Although the traditional A^* algorithm is most effective search algorithm in terms of low cost and path length optimization, it still takes more time in searching the end location and has a tendency of using more nodes than required. To overcome this limitation, [9] stated a way to increase the speed of the algorithm and make it an ideal path planner. The proposed alternative modifies the traditional A^* search algorithm to loose search on a fine grid by using trial vector that can cross several cells for fast computing.

Algorithm 1 A^* Search Algorithm

Input: A graph
Output: A path between start and end node

1: **repeat**
2: Pick n_{best} from O such that $f(n_{best}) \leq f(n)$, $\forall$n$\in O$
3: Remove n_{best} from O and add to C
4: If n_{best} = q_{goal}, EXIT
5: Expand n_{best}: $\forall$n$\in Star(n_{best})$ that are not in C.
6: **if** $x \notin$O **then**
7: add x to O.
8: **elseif** g(n_{best}) + c(n_{best},x) $<$ g(x) **then**
9: update x's backpointer to point to n_{best}
10: **end if**
11: **until** O is empty.

The main concept of this modification is to change the A^* candidate selection for successor nodes by two criteria. The first one is that the successor is not the parent node of the current node; and secondly, the successor does not contain an obstacle [12]–[14]. The new altered or modified approach utilizes the same criteria used by standard A^*; yet, as opposed to testing every candidate successor node individually, it tests more than one node together according to the possible direction of movement. It helps to eliminate the possible unsafe diagonal movements.

Another important feature of modified A^* is that it considers the size of the robot, and used as a parameter that must be inserted and included while selecting the candidate successor nodes. It uses the number of nodes that equal to the robot size while calculating the next node, ensuring that the successor have enough free space that enables the robot to move without a risk of collision.

For our purposes on this research, similarly for the traditional A^* algorithm, $f(n)$ function from the Equation (1) remains the same by definition for the modified A^* algorithm. The node which has the minimum value of $f(n)$ is a much better candidate to achieve the target. There are different versions of modified A^* algorithms with more accurate and efficient path planner, but for our case study the above discussed modified A^* algorithm is being used.

III. Case Study

In this research, a new way to reduce the number of step for searching the end node is proposed, three scenarios are prepared to compare results for different approaches. To test it in real time a 3D model of Autonomous Control Engineering (ACE) Laboratory of UTSA is constructed and Kobuki TurtleBot 2 mobile robot is used.

A. Test Bed Environment

For this research a static environment of ACE Lab was created. The lab is approximately 47 feet 60 inches long, 29 feet 90 inches wide and 9 feet 70 inches high. It has

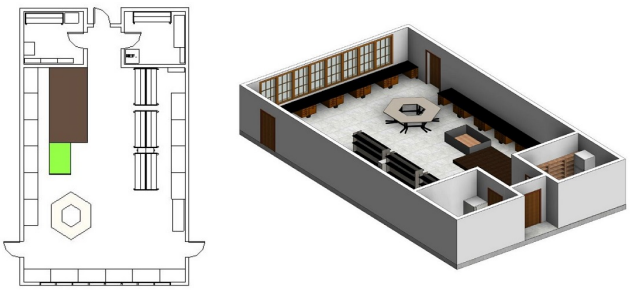

Fig. 1. Outline of the test-bed environment and its 3D model.

two separate rooms and windows on the opposite side of the entrance. Entire test-bed illustrated in REVIT and a 3D model of the test environment has created as shown in Fig. 1.

Since the results can be seen much before they are implemented in reality, we decided to implement the simulation approach in the beginning. Some of the advantages of simulation are the ability to diagnose source code that controls a particular resource, or a mix of resources, such as to simulate various alternatives without involving physical costs, to segregate into different stages which are beneficial for complex projects.

There are many widely used robotic simulators such as LabVIEW Robotics Simulator, V-REP, Webots, RobotStudio, Workspace, RoboDK, Gazebo, and etc. For this research, we used Gazebo, since it is compatible with Robot Operating System (ROS), which is required to operate the Kobuki TurtleBot2 [15]. The TurtleBot2 is a differential drive kinematics mobile robot developed for research and educational purposes. Gazebo uses a right-hand coordinate system where +Z is up (vertical), +X is forward (into the screen), and +Y is to the left. To import the environment in Gazebo, our 3D model need to be converted to either a STL format or an OBJ format.

B. Synchronizing the Simulated Environment with Real World

The Universal Robot Description Format (URDF) is an XML-like file type used heavily in ROS for simulation and testing. URDF can be used for indicating just the kinematic

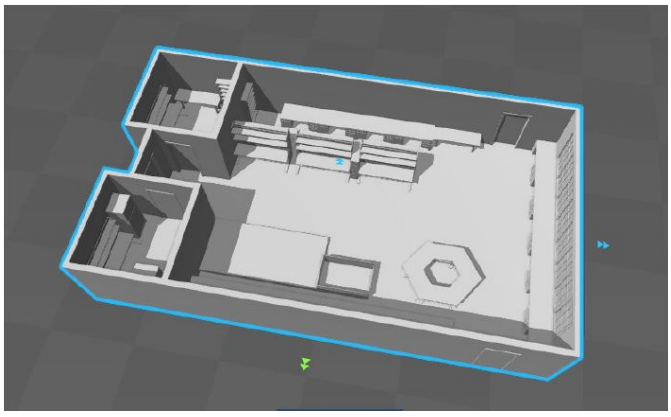

Fig. 2. Test-bed environment in STL format integrated to Gazebo.

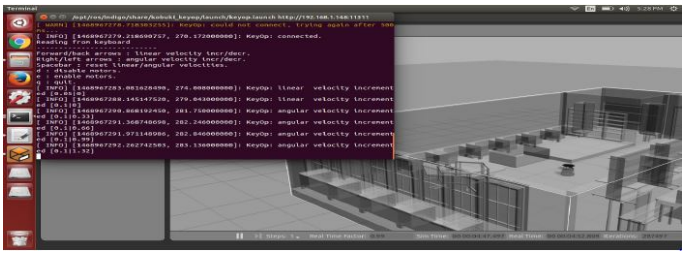

Fig. 3. Real time simulation and control of the mobile robot with ROS integration in the Gazebo environment of the test-bed.

properties of a robot; however, it cannot indicate the pose of the robot itself inside the world. Another drawback of URDF is that its capacities are restricted in contrast with the new Simulator Description Format (SDF). SDF was developed to fix some of the shortcomings of URDF, such as improper use of XML syntax [17]. SDF files are what Gazebo uses when performing simulation.

To work with SDF, an STL file is converted which has the test-bed environment model as shown in Fig. 2. The advantage of using SDF is we can change or modify the environment, such as inserting a new object. One can set the pose of an object without affecting surroundings. Once the environment has been established and the modifications have been made, a robot can be imported into the environment. Because of its satisfactory performance and compatibility with the ROS and Gazebo, we used TurtleBot2, as illustrated in Fig. 3.

C. Traditional Approaches

1) A Search Algorithm:* The heuristic function $h(n)$ for our case study can be expressed as:

$$h(n) = (l_h + l_v)/2 \tag{2}$$

where l_h is the number of the horizontal line and l_v is the number of the vertical lines. as explained above, n is the present node on the path, $g(n)$ is the cost of the path from the starting node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal node [11]. For instance, the cost function from one node to another node is 1 and for a diagonal movement from one node to another is 1.4- the Pythagoras value.

In other words, the cost is the diagonal value between two grid/pixels. As stated in Equation (1), the cost can be written as:

$$g(n) = g(n') + cost \tag{3}$$

where $g(n')$ is the previous node cost function. Note the cost of starting node is always zero. Once the open list and closed list are made, open list contains all the possible expansion of parent node.

As shown in the Fig. 4, the start and end nodes are denoted with red and light green points, respectively. Orange color boxes in the grid around the starting point are the next possible children nodes with red as parent. Depending on the next node in the closed list, the colored nodes are yellow boxes

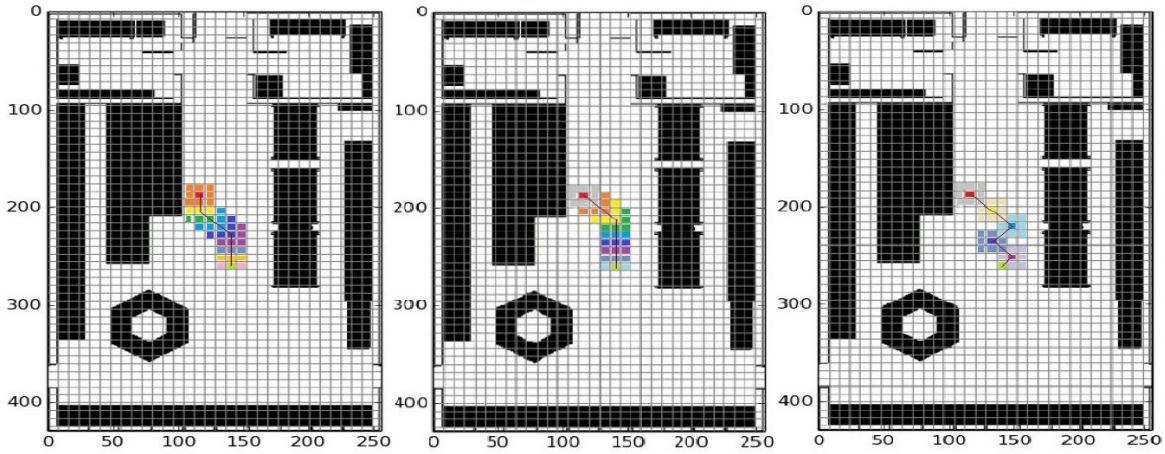

Fig. 4. Initial results for a known area with A^* Search, Modified A^* Search, and Proposed route optimization algorithm, respectively. Our aim is comparing the number of expanded nodes to reach the destination. It seems that the traditional methods are computationally expensive.

as children, followed by green, blue, indigo, purple, violet, orange, and lastly, the end point which is light green. The brown line shows the path and the total cost is 10.2, based on previous equations. Thus, the shortest and cheapest path is calculated very easily. A^* will provide the shortest and most inexpensive path, but it will not be seeking the smoothest path. Relative position between the current heading (direction) and the next move must be taken into consideration to improve the performance of the A^* path planner. Moreover, it does not take into consideration the size of the robot which may result in obstacle collision and inaccurate path, since the robot may be too large to pass the empty space. Thus, it can provide unsafe path for the robot. This limitation can be overcome by modified A^* algorithm.

2) Modified A^* Search Algorithm: As we described in previous section, $h(n)$ estimates the distance to the destination, which can be calculated as restated in [18]:

$$h(n) = \sqrt{(w_x - r_x)^2 + (w_y - r_y)^2} \tag{4}$$

an Euclidian distance, where (w_x,w_y) is the coordination of the aimed location, r_x and r_y are the horizontal and vertical distances from the previous heuristic. By calculating the Euclidean distance between the current node and the destination node, we can calculate an estimation distance that always is less than or equal to the actual distance [19]. For the same experimental setup shown in Fig. 4, the heuristic value can be calculated as $\sqrt{(9)^2 + (3)^2} = 9.5$. Then, this value is entered in closed list and $f(n)$ is expanded in all possible directions. Since it can jump to all nodes around the starting node, S, so the possible number of child node from parent node are 8. Now all the 8 child nodes are inserted in open list and the value $f(n)$ for all is calculated. The least $f(n)$ value is inserted into closed list. Note that there can be more than one least $f(n)$ at every iteration. To select a node from the successor node, it considers the next two adjacent node for diagonal movement, so that there should be no collision as discussed above. Again all the possible child node from their parent node are inserted into open list and $f(n)$ is calculated. Each node in the lowest $f(n)$ is chosen, and then is compared with a list of the current nodes to check whether the specified node is still better candidate or not. Otherwise, it continuous to look for the better node [20]. If $h(n)$ is increasing, then the direction may be wrong/opposite and this node can be ignored. These items from the closed list are ignored while searching for next possible node. The value from the closed list sets the path. Note if a value is being used twice for calculating next possible node, it is not inserted again in open list. The algorithm uses the previous value of that node instead of recalculating the $f(n)$.

In Fig. 4, the results for the modified A^* algorithm is demonstrated as well. The results are similar to A^* algorithm as the grid size is equal to that of the robot, but with consideration of robot size for collision avoidance.

D. Proposed Algorithm

In literature path planning works were done in order to reduce the time, increase the accuracy and reach the end node in shortest distance; however, the approach here is to find an inexpensive way to find a path with reduced number of steps in searching process. We are aiming to fulfill this by jumping from parent node to the successive node, leaving a node in between them, in the direction from the parent node towards the child node. The algorithm can work, if and only if the successor node and the parent nodes are having a common empty node in between them. If there is no path between the successive nodes then the path planner searches with different node. The path planner take into consideration only the nodes with the least $f(n)$ and with smallest $h(n)$ in that iteration. In this manner, the steps to search the destination reduced tremendously, compared to the previous algorithms.

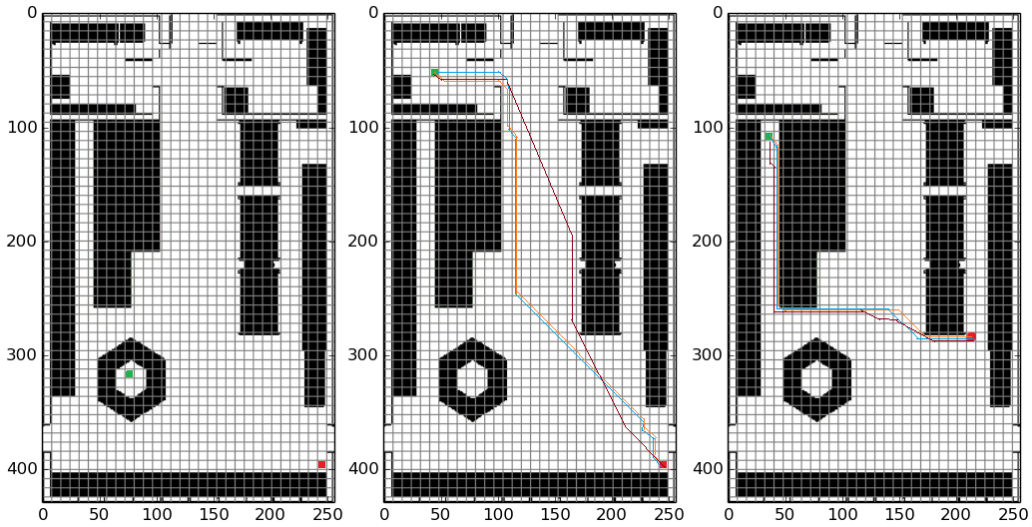

Fig. 5. Experimental results for path planning problem in a known area with A^* Search (orange), Modified A^* Search (blue), and Proposed route optimization algorithm (red).

It also takes into consideration the robot size. The advantages of this approach compared to previous path planners can be stated as following. Firstly, it is faster, decreases the steps for searching the end location by number of nodes visited. Secondly, the cost is reduced overall; and lastly, the number of expanded nodes are comparatively less than the previous mentioned approached.

The heuristic function $h(n)$ for this algorithm is the same construction as stated in Equation 4. where n is the present node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal [11]. The cost function from one node to another node is calculated as 10, and for a diagonal movement from one node to another is 14. Cost of 14 value is used by taking the diagonal value between the two grid/pixels and calculating the Pythagoras value between them which is multiplied with 10 that gives 14 while considering the absolute value. Cost for inclined movement is calculated by taking the square root of the squares of adjacent side and the opposite side. The moving cost $f(n)$ carries the same form as shown in Equations 1 and 3. Note that the cost of starting node is always zero. An open list and closed list is made, where open list contains all the possible expansion of parent node.

Same setup is considered, the start point is denoted with red color and the end location is denoted by light green color. Light grey boxes in the grid around the start location are the next possible children nodes with red box as the parent. Yellow box is the next available node depending on the calculations and the light yellow boxes represents all the next possible successor nodes followed by blue, navy blue and purple. We can see that after the purple, the path planner according to the proposed algorithm directly jumps to the light green end node and the algorithm breaks and returns the path. The brown line shows the complete path. As can be seen from Fig. 4, it is clear that the nodes to search the end results are much less compared to that of A^* and modified A^* algorithm.

IV. Simulation Results

To compare three different approaches, three scenarios are prepared. Test cases are built to analyze for time efficiency, lesser step to reach the aimed location, total number of the visited nodes and total cost for the test case for each individual algorithm. The Python programming language is used for testing the path planning algorithm.

The first case test whether the algorithm is working accurately by placing the start location inside the hexagonal table and setting the end location somewhere far. The second and third cases checks whether the A^* gives the shortest distance or not. The other purpose of these two test cases are to check the diagonal safety movement by modified A^* algorithm. For testing the algorithm, the environment is changed to a 2D grid with 0 as accessible path and 1 as non accessible path. The grid is made by taking into consideration the robot size. For each test case, the process starts with accessing the grid. The image is made 0 and 1 in which all the positions with 0 are clear routes and the rest are obstacles. Once the grid is called, the obstacle table is inserted in the algorithm. The start and end locations are defined by the user. Two separate lists are made for storing nodes. First, the open list to store all current node for calculating heuristic, the movable cost. Second, the closed list, which stores the path with visited nodes. The start node is inserted into closed list and then the open list is checked whether it is empty or not. If the open list is empty, it means there is no path and the algorithm breaks and displays no path found. If open list is not empty; then, all the neighboring nodes are selected to check whether destination is reached or not. If destination is reached, then the algorithm updates the closed list by adding the node and

TABLE I
COMPARATIVE RESULTS FOR PATH PLANNING PROBLEM OF SEARCH ALGORITHM FOR THREE CASES IN A KNOWN AREA AS ILLUSTRATED IN FIG. 5

	Case #1			Case #2			Case #3		
Search Type	# of nodes	Path to goal	Cost	# of nodes	Path to goal	Cost	# of nodes	Path to goal	Cost
A*	9	n/a	n/a	496	49	576	114	39	420
Modified A*	9	n/a	n/a	504	50	590	121	40	426
Proposed	5	n/a	n/a	206	24	566	79	20	411

displays the paths. Different than the traditional approaches, if destination is not reached, proposed algorithm will check whether the neighboring nodes are empty or not. If the nodes are not empty; then, the planner will again search with the neighboring nodes, instead of jumping to the successor node, and if again there is no path found then the search algorithm will terminate and returns a false value with *"No path found"*. If the neighbors are empty, then the planner jumps to the successor nodes, and calculate the heuristic and cost function. The least cost function is to be added to closed list, and the destination is checked. If destination is reached, the closed list is updated and path is displayed. If the destination is not reached, the planner again checks for open list and the process is continued until it finds a path, or the open list gets empty.

V. CONCLUSIONS AND FUTURE WORK

The research objective of this paper was to investigate A^* and modified A^* path planning algorithms, and to propose an advance modified version of path planning which provides result in less number of expanded nodes compared to that of the test algorithms in a known environment. The following concepts were identified as important components of the proposed algorithm: the search steps are nearly half the size of that compared to A^* and modified A^* search, the cost to search the destination is relatively less compared to other two algorithms and the number of expanded nodes is reduced tremendously, as shown in Table I. The proposed algorithm is fast, accurate and takes into consideration the size of robot.

In Fig. 5 Case 1, Case 2 and Case 3 are illustrated, from left to right, respectively. For all cases, the start point is marked by green and the goal is marked by red point. Since there is no path between the start location and end location for Case 1, the path planning algorithm terminates and returns a *false* value showing that when there is no accessible path between the start and end location, path cannot be created.

As it is noticed from the Fig. 5, for Case 2 and Case 3, the A^* Search algorithm highlights the cheapest and shortest path, but the turns are sharp which may result in collision. In comparison to that, the Modified A^* algorithm avoids the sharp turn by looking for adjacent nodes and create a collision free path. The proposed algorithm creates a path faster than the other two with less number of the expanded nodes. Following tables are comparing the final results for considering the time efficiency, total number of the visited nodes and the total cost overall.

REFERENCES

[1] T. Lozano-Perez, "Spatial planning: A configuration space approach," IEEE Transactions on Computers, Vols. C-32, pp. 108-120, 1983

[2] T. Lorenzo-Perez and R. A. Brooks, "A subdivision algorithm inconfiguration space for find path with rotation," IEEE Transactions on Systems,Man and Cybernetics, Vols. SMC- 15, no. 2, pp. 225-233, 1985.

[3] R. A. Brooks, "Solving the find path problem by representing free space as generalized cones," in A.I Memo No 674, Massachusetts Institute of Technology, May 1982.

[4] O. Takahashi and J. Schilling, "Motion planning in a plane using generalized Voronoi diagrams," IEEE Transactions on Robotics and Automation, vol. 5, no. 2, pp. 143-150, 1989.

[5] N. Sariff and N. Buniyamin, "An overview of autonomous mobile robot path planning algorithms," in IEEE 4th Student Conference on Research and Development, 2006.

[6] B. R. Donald, "A search algorithm for motion planning with six degrees of freedom," Artificial Intelligence, vol. 31, no. 3, pp. 295-353, 1987.

[7] C. Niederberger, e. Radovie and M. Gross, "Generic path planning for real time application," in Proceedings of the Computer Graphics International (CGI 04), 2004.

[8] C. Alexopoulos and P. M. Griffin, "Path planning for mobile robot," IEEE Transactions on Systems, Man, and Cybernetics, vol. 22, no. 2, pp. 318-322, 1992.

[9] C. W. Warren, "Fast path planning method using modified A* method," .IEEE International Conference on Robotics and Automation, pp. 662-667, 1993.

[10] Mohammed, Ibrahim. Optimal navigation of autonomous vehicles. Diss. THE UNIVERSITY OF TEXAS AT SAN ANTONIO, 2016.

[11] Choset, Howie M. Principles of robot motion: theory, algorithms, and implementation. MIT press, 2005.

[12] ElHalawany, B. M and e. al., "Modified A* algorithm for safer mobile robot navigation," in Modelling, Identification and Control (ICMIC), 2013 Proceedings of International Conference IEEE, 2013.

[13] J. Yao, C. Lin, X. Xie, A. J. Wang and C.-C. Hung, "Path Planning for Virtual Human Motion Using Improved A* Star Algorithm," in Seventh International Conference on Information Technology: New Generations (ITNG), 2010.

[14] C. B. Crous, Autonomous Robot Path Planning: M.Sc thesis, South Africa: University of Stellenbosch, 2009.

[15] Gallardo, N., Pai, K., Erol, B. A., Benavidez, P., and Jamshidi, M. (2016, July). Formation control implementation using Kobuki TurtleBots and Parrot Bebop drone. In World Automation Congress (WAC), 2016 (pp. 1-6). IEEE.

[16] T. Lozano-Perez and M. A. Wesly, "An algorithm for planning collisionfree paths among polyhedral obstacles," Communication of the ACM, vol. 22, no. 10, pp. 560-570, 1979.

[17] Kouba, Anis, ed. Robot Operating System (ROS): The Complete Reference. Vol. 1. Springer, 2016.

[18] A. Patel, "Heuristics: From Amits Thoughts on Pathfinding," Stanford University, 22 June 2016. [Online]. Available: http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html. [Accessed 16 July 2016].

[19] C. Hong-ying, X. Ting, W. Tao and H. Jin-yi, "The Research and Implementation of Three Stages Traffic Stations Intelligent Monitor Systems Based on GIS," in 2012 International Conference on Solid State Devices and Materials Science, Physics Procedia, 2012.

[20] S. M. Ayazi, M. F. Mashhorroudi and G. M, "Modified a* Algorithm Implementation in the Routing Optimized for Use in GIS," The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 40, no. 2, p. 69, 2014.