

Enhanced D* Lite Algorithm for Mobile Robot Navigation

Soh Chin Yun

School of Engineering, Monash
University, Sunway Campus,
Bandar Sunway, Malaysia.
mscysoh@gmail.com

Velappa Ganapathy

Faculty of Engineering,
University of Malaya
Kuala Lumpur, Malaysia.
velappa.ganapathy@um.edu.my

Tee Wee Chien

School of Engineering, Monash
University, Sunway Campus,
Bandar Sunway, Malaysia.
wct_1986@hotmail.com

Abstract— Mobile robot has been widely used in exploration and navigation and it is required to operate in domains that are completely unknown and dynamic. Unknown environment is where the locations of the obstacles are unknown and dynamic environment is where the locations of the obstacles might change with time. This research is focused on enhancing the existing D* Lite Algorithm. Existing D* Lite Algorithm is a goal-directed navigation algorithm and in this research, Enhanced D* Lite Algorithm is developed. Simulation of the Enhanced D* Lite Algorithm is created using MATLAB to verify feasibility and validity of the algorithm. Next, the Enhanced D* Lite Algorithm is implemented in real-time using Team AmigoBot™. The results obtained from both simulation and real-time implementation have proved the robustness and practicality of the enhanced algorithm.

Keywords — D* Lite Algorithm; Team AmigoBot™; MATLAB

I. INTRODUCTION

One of the mobile robot tasks is to use it for navigation purposes, such as navigation for exploring unknown planet, navigation for military purpose, goal directed navigation, path finding and more. The environment for navigation could be known or unknown with moving and static obstacles. For goal-directed navigation, a mobile robot has to move from the given starting position towards the goal position with the capability to avoid obstacles as well. The mobile robot should be also able to traverse in both known and unknown environments. Besides that, time to traverse from starting point to goal point is also an important aspect emphasized by researches. It is important for mobile robot to plan a shortest path from starting point to goal point. Moreover, the mobile robot must be able to re-plan its path quickly if there is a new obstacle in front or nearby.

There are few algorithms developed for goal-directed navigation such as A* Algorithm [1], D* Algorithm [2], and D* Lite Algorithm [3-5]. A* Algorithm [1] is an early developed popular graph search algorithm which finds the shortest path from a given initial start node to the goal node. A* Algorithm uses distance plus path cost function to determine the shortest path to the goal node. In A* Algorithm, node or square notation is used rather than coordinate because a map is divided into small grids or squares and nodes represent the center point of each grid.

D* Algorithm, which is also known as Stentz algorithm or Dynamic A* Algorithm, is developed by Anthony Stentz in 1994 [2]. It is better than A* Algorithm because it could be used in partially or completely unknown and

also dynamic environment. A* Algorithm is a simple algorithm which could be only used in static environment. But D* Algorithm is able to repair or update the map of the dynamic environment and it can re-plan quickly whenever it detects there is a new obstacle or an obstacle is removed on the way to goal node [2].

D* Lite Algorithm, which is one of most popular goal-directed navigation algorithm, is a reverse or backward searching method and it is able to re-plan from current position when there is a new obstacle blocking the path. It determines the same paths as D* Algorithm and moves the mobile robot the same way but it is algorithmically different from D* Algorithm. D* Lite Algorithm is implemented based on Lifelong Planning A* [6] Algorithm. It has been widely used for mobile robot navigation in unknown environment. Similarly, it divides the environment into grids. The path finding and robot's movement is from grid to grid.

From the study and review of existing D* Lite Algorithm, the main problems of the algorithm are mobile robot is traversing across obstacles' sharp corners, traversing in between two obstacles, and trapped in the 'U-Shaped' type obstacles. Also, there is no real-time implementation reported. To overcome the mentioned problems, Enhanced D* Lite Algorithm has been implemented.

In this research, the existing D* Lite Algorithm is investigated in detail and enhancements are emphasized on (i) preventing mobile robot from traversing across obstacles' sharp corners, (ii) avoiding complicated obstacles, (iii) preventing mobile robot from traversing in between two obstacles, (iv) creating virtual wall if necessary, and (v) removing unnecessary pathways to yield shortest path. Complicated obstacles could be U-shaped obstacles, or obstacles which may trap the mobile robot. The mobile robot should be able to escape out from those obstacles. Besides that, directly traversing across obstacles' sharp corners and in between two obstacles are not encouraged since there might be a chance the mobile robot would hit obstacles. Performance of the Enhanced D* Lite Algorithm in real-time implementation is also investigated in detail.

II. SYSTEM OVERVIEW

Consider a mobile robot navigation task in an environment with 'U-shaped' type obstacles as shown in Figure 1, where no information about the obstacles positions is provided to the mobile robot. It has to find a shortest path from the goal position towards starting position (back propagation). It will compute a shortest path from its current position with respect to the starting

position until it reaches starting position. White grids are known to be traversable, black grids are known to be obstacles, which are untraversable and round dot representing mobile robot.

Referring to Figure 1A, the mobile robot will traverse vertically up towards the starting position since that is the shortest path to reach starting position from goal position and it assumes all grids are traversable. As the mobile robot is traversing vertically towards starting position, it is trapped in the 'U-shaped' type obstacles as shown in Figure 1B. Eventually the sensors of the mobile robot will detect and realize that there is an obstacle blocking in front of the mobile robot. The mobile robot will continue to search for other grids and check if there is a new path to reach starting position. The only traversable grid now is the grid behind the mobile robot which means the mobile robot has to traverse out from the 'U-shaped' type obstacles. As the mobile robot traverses out from the 'U-shaped' type obstacles, virtual walls will be created to make sure the same grids would not be traversed again. This is shown in Figure 1C and Figure 1D. The mobile robot is able to escape out from the 'U-shaped' type obstacles and continue to find the shortest path in order to reach starting position. Referring to Figure 1D, as the mobile robot traverses across obstacle's sharp corner, a minimum clearance distance is provided between obstacle's sharp corner and mobile robot. This is important to prevent any damage to mobile robot. Finally, the mobile robot is able to reach starting position as shown in Figure 1E.

Once the mobile robot has reached the starting position, it will follow back the path found just now and return to goal position. No more path finding is required. For the shortest path found travelling from goal to starting position, if there are unnecessary pathways, it would be removed as shown in Figure 1H. This is the navigation strategy of Enhanced D* Lite Algorithm.

III. ENHANCED ALGORITHM PROPOSAL

Basically, the path finding principle of Enhanced D* Lite Algorithm is the same as existing D* Lite Algorithm. However, Enhanced D* Lite Algorithm has been implemented with five additional features which are mentioned in the introduction. Coordinate specification used to represent positions in Enhanced D* Lite Algorithm is (i,j) where i is the row and j the column. Enhanced D* Lite Algorithm is using back propagation searching method as in D* Lite Algorithm to find the shortest path. Back propagation searching method is where the path finding search will start from goal position towards starting position. Details of how the Enhanced D* Lite Algorithm works would be explained in steps as follows.

1. Calculate h (Heuristic) value

The first step of the enhanced algorithm is to calculate the h (heuristic) value. $h(i,j)$ indicates the h (heuristic) value in the i^{th} row and j^{th} column. Starting position is assigned with an h value of 0. Then, h value is incremented by 1 from grid to grid until all the grids are assigned with the respective h values as can be seen in Figure 2. Increment of h value could be done vertically,

horizontally, and diagonally with the condition that the smallest h value is taken.

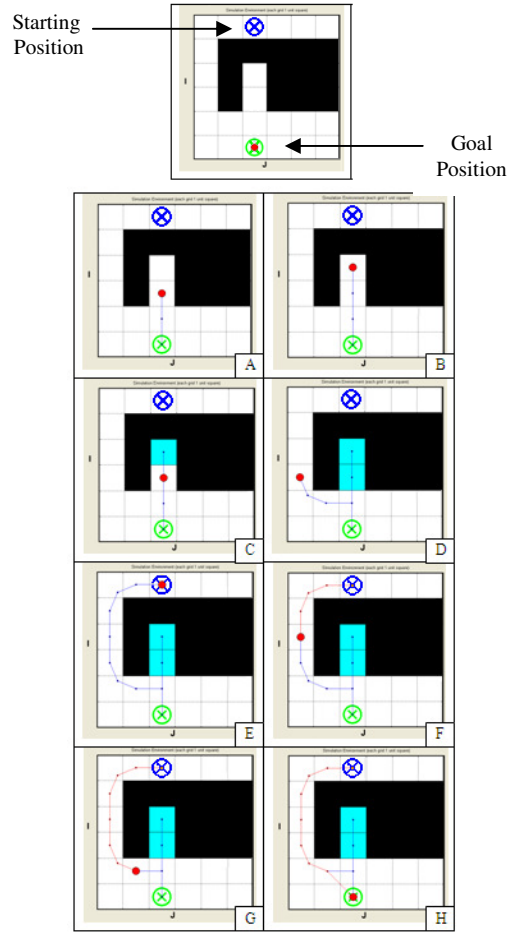


Figure 1: Illustration of the Navigation Strategy

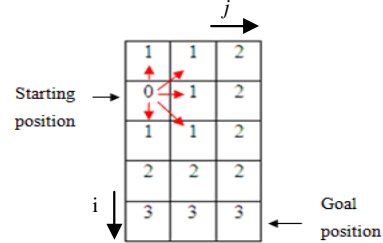


Figure 2: Calculate h (Heuristic) Value

2. Calculate rhs (Look Ahead Function) and g (Cost Function) values

Initially, each grid of the environment is assigned with g and rhs values of infinity. $rhs(i,j)$ is a look ahead function. rhs values will be re-calculated when the grids are analyzed. Formula of the rhs (Look Ahead Function) is shown as:

$$rhs(i, j) = \min[succ(i, j)(g)] + 1 \quad (1)$$

where g indicates the g values of $rhs(i,j)$ successors, $succ(i,j)$ is the successors of i^{th} row and j^{th} column.

$g(i,j)$ is the cost function and g value will be re-calculated when the grid is expanded. The formula of g is shown as:

$$g(i, j) = rhs(i, j) \quad (2)$$

3. Calculate $k1$ and $k2$ (Priority Queue Function) values

Basically, $k1$ and $k2$ are priority queue functions. $k1$ and $k2$ values would be calculated when the grid(s) is(are) analyzed by mobile robot. Grid with the smallest

$k1$ value will be expanded. Formula for calculating $k1$ and $k2$ are shown as:

$$k1(i, j) = rhs(i, j) + h(i, j) \quad (3)$$

$$k2(i, j) = \min[g(i, j), rhs(i, j)] \quad (4)$$

Path finding will stop when $k1$ value is equal to $k2$ value which means the mobile robot has reached the starting position.

4. Initialization

After defining all the variables and their respective functions in Enhanced D* Lite Algorithm, initialization is to be performed before path finding starts. First, rhs value of goal position is set to value of 0, $k1$ and $k2$ values of goal position are calculated and g value of goal position is set equal to rhs value of goal position.

5. Compute Shortest Path

Next, the surrounding grids will be analyzed using sonar sensors. This is to check whether the grid is free of obstacles and could be traversed. In this research, mobile robot Team AmigoBot™ [7] is used and it is built with eight sonar sensors. The structure of the Team AmigoBot™ is shown as:

$$S = [s1 \ s2 \ s3 \ s4 \ s5 \ s6 \ s7 \ s8] \quad (5)$$

Referring to Figure 3(a), $s1$ to $s6$ are the front sensors and $s7$ and $s8$ are the rear sensors.

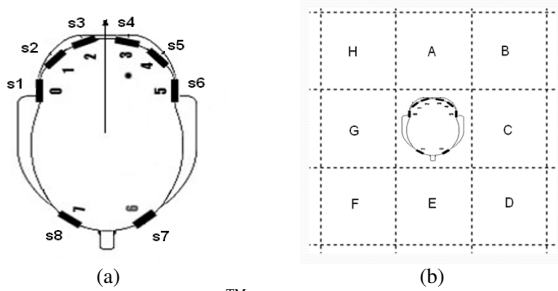


Figure 3: (a) Team AmigoBot™ [7]; (b) Grids A to H Surrounding the Team AmigoBot™

The position of the mobile robot when analyzing grids is shown in Figure 3(b). Basically, there are eight grids surrounding the mobile robot labeled grid A to grid H. Selection procedures of the grid(s) to be updated with new rhs , $k1$ and $k2$ values are shown in the flowchart Figure 9(a). The grid(s) updated with the new values would be the potential next grid to be expanded.

Referring to the flowchart in Figure 9(a), for each of the eight grids surrounding the mobile robot, firstly the MATLAB program will check whether the grid is free of obstacle by analyzing the sonar sensors readings. If the grid is not free of obstacle, it will be checked for second condition, whether the grid is already a virtual wall. If the grid is not a virtual wall, it will be checked for third condition, whether the grid is already in the closed list. If the grid is not in the closed list, it will be checked again for fourth condition to prevent mobile robot from traversing in between two obstacles. If this condition is not satisfied, then the grid's information will be updated which is the next step.

Referring to Figure 4 as an example, although traversing from grid (4,4) to grid (3,3) would give shorter path to reach starting position, but this action is

prevented. Text boxes shaded in the flowchart are the enhancements proposed.

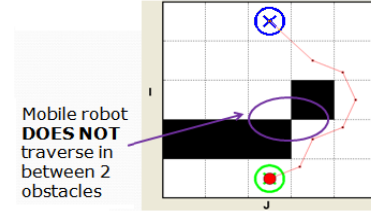


Figure 4: Prevention from Traversing in between Two Obstacles

6. Store and update grid information

Referring to the flowchart in Figure 9(a), if any of the eight grids satisfy all the four conditions, the grid would be updated with new rhs , $k1$ and $k2$ values. If any of the four conditions is not met, the next grid would be analyzed till all the eight grids are exhausted.

7. Priority queue

After analyzing all the eight grids, the updated grid(s) with the smallest $k1$ value would be selected where U is given by:

$$U = \min(k1) \quad (6)$$

Then, a grid in the selected list with minimum distance with respect to starting position would be selected and this will be the next grid to be expanded, (i,j) .

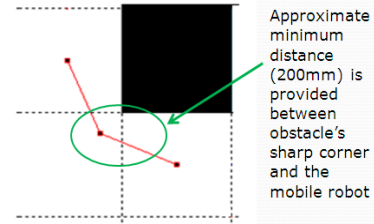


Figure 5: Minimum Clearance Distance is Provided between Obstacles' Sharp Corners and Mobile Robot

8. Traversing to the grid to be expanded

Before mobile robot traversing to the grid to be expanded, few more conditions need to be checked. Firstly, $g(i,j)$ value of the grid to be expanded is set to its $rhs(i,j)$ value. Then, the grid will be checked whether it has been traversed twice. If yes, a virtual wall would be created for the previously traversed grid and the grid to be expanded would be stored in the closed list. This is to make sure that the previously traversed grid would not be traversed again and the grid in the closed list would not be analyzed again. Then it will be passed to the next condition. This condition is to check if the grid to be expanded is a diagonal grid with an obstacle on its left, right, top, or bottom. If yes, the mobile robot will traverse to the grid to be expanded with a minimum clearance distance between mobile robot and obstacles which is shown in Figure 5. If no, the mobile robot will traverse directly to the grid to be expanded. Better illustration of this step is shown in the flowchart in Figure 9(a). If the grid to be expanded is free of obstacles, list of traversing actions are illustrated in Figure 6 (1 to 6). If the grid to be expanded is a diagonal grid with an obstacle on its left, right, top, or bottom, list of traversing actions are illustrated in Figure 7 (7 to 14).

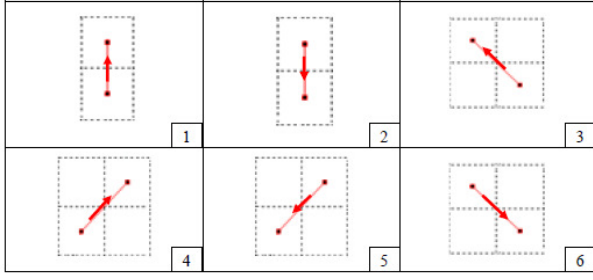


Figure 6: List of Traversing Actions if the Grid to be Expanded is Free of Obstacles

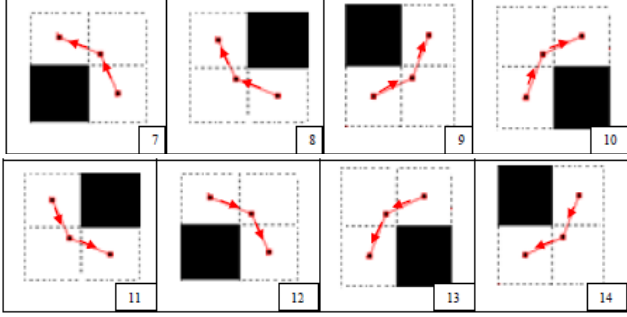


Figure 7: List of Traversing Actions if the Grid to be Expanded is a Diagonal Grid with an Obstacle on its Left, Right, Top, or Bottom

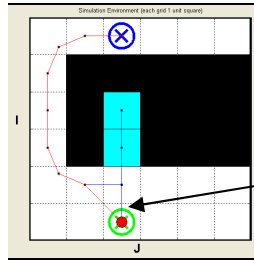


Figure 8: Virtual Walls are Created and Unnecessary Pathways are Removed

9. Return to goal position

After finding the shortest path, the mobile robot will return to the goal position. Basically, no more searching is required, path remembering function will be executed. If there are unnecessary pathways, it will be removed while the mobile robot returns. If there are additional obstacles blocking the returning path, step 5 to step 8 will be repeated to find the new shortest path. Thus, the current position will be assigned as goal position and the original goal position will be assigned as starting position.

The complete flowchart of Enhanced D* Lite Algorithm back propagation searching method from goal position towards starting position is shown in Figure 9. For returning path, mobile robot returns to the goal position by following the shortest path grids found. Movement of the mobile robot would be still from grid to grid to further analysis if there is any new obstacle blocking the shortest path found. Figure 10 illustrates the detailed flowchart of Enhanced D* Lite Algorithm back propagation returning method from the starting position to the goal position.

IV. MATLAB SIMULATION

Basically, MATLAB simulation is created to confirm

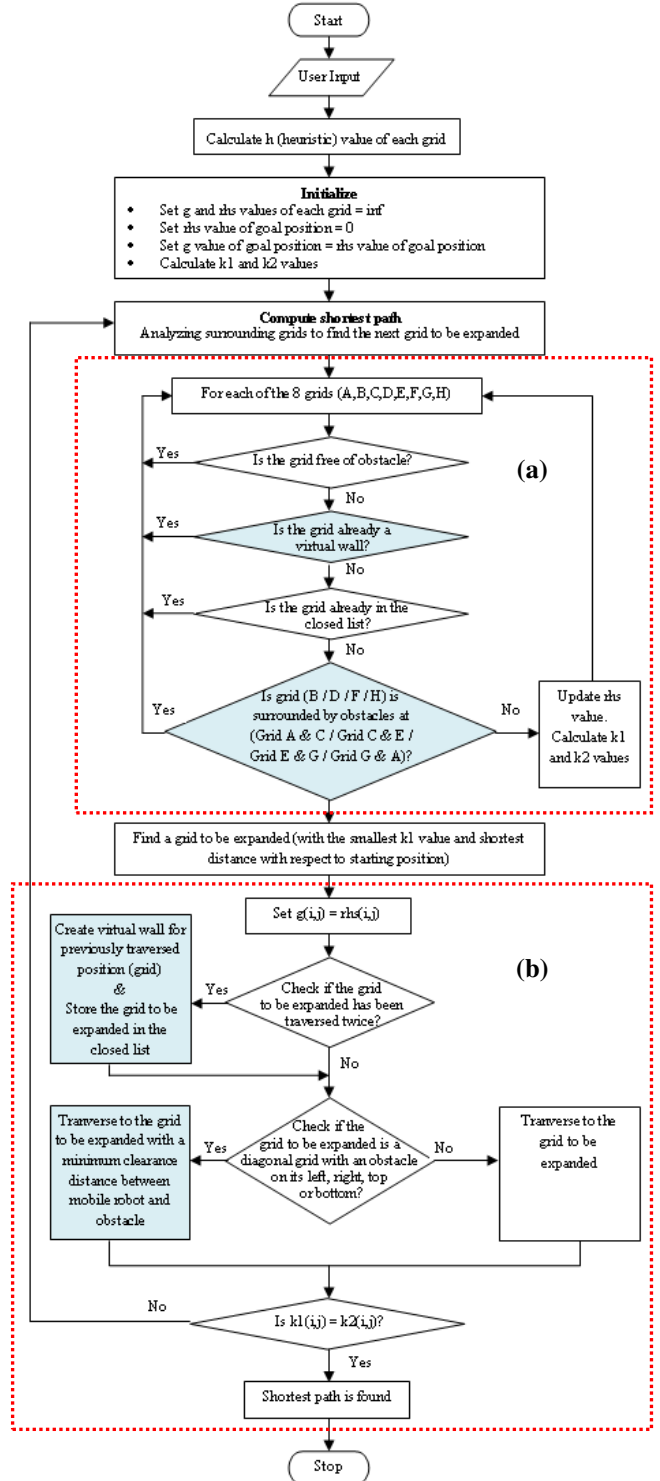


Figure 9: Complete Flowchart of Enhanced D* Lite Algorithm Back Propagation Searching Method from Goal Position Towards Starting Position. (a) Determine the Grid to be updated with new rhs , $k1$ and $k2$ values; (b) Determine the Traversing Action is to be taken and Virtual Wall(s) is(are) created if necessary

the feasibility of the enhanced algorithm. In MATLAB simulation, each grid is represented as one unit square.

Result of Test 1: Travelling from goal to start position

Referring to the result of Test 1, in which mobile robot had travelled from goal position to starting position (back propagation), a minimum clearance distance was

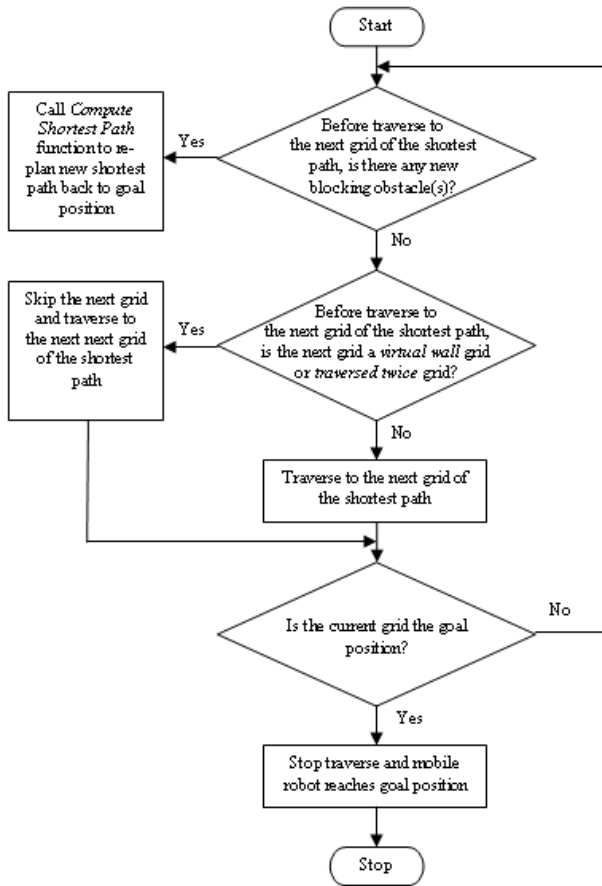


Figure 10: Detailed Flowchart of Enhanced D* Lite Algorithm Back Propagation Returning Method from Starting Position to the Goal Position

Test 1

Environment size	10 x 10
Starting position	1,1
Goal position	10,10
Number of Obstacles	34. Obstacles positions defined by user
Additional Obstacles added for returning path	Initially no, when test 1 repeated, additional obstacles for returning path are added.

Table 1: Input Parameters of Test 1

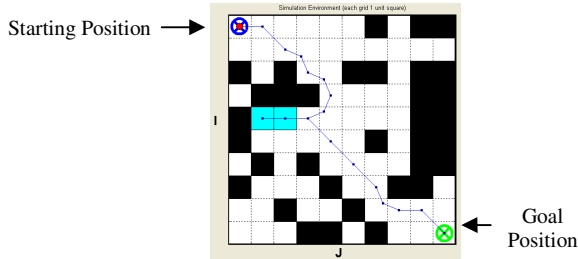


Figure 11: Travelling from Goal to Starting Position

provided between obstacles' sharp corners and the mobile robot and virtual walls were created so that the same grids would not be traversed or analyzed again. Moreover, mobile robot was also able to move away from 'U-shaped' type obstacles and it did not traverse in between two obstacles. Blue trace line (without considering the trace line covered with virtual walls) is the shortest path found by travelling from goal to starting position.

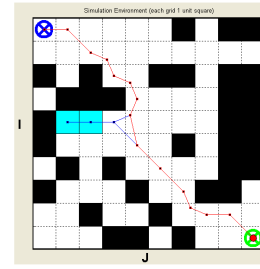


Figure 12: Returning from Starting to Goal Position without Additional Obstacles

Result of Test 1: Returning from starting position to goal position WITHOUT additional obstacles

Referring to Figure 12, it was the result obtained for mobile robot returning from starting position to goal position without additional obstacles. For returning path without additional obstacles blocking the initial shortest path found, path remembering function was executed to direct the mobile robot back to the goal position (grid by grid movement) and no more path searching was required. For the shortest path found travelling from goal to starting position, if there are unnecessary pathways, it would be removed.



Figure 13: Returning from Starting to Goal Position with Additional Obstacles

Result of Test 1: Returning from starting position to goal position WITH additional obstacles

The result of mobile robot returned from starting position to goal position with additional obstacles is shown in the Figure 13. While the mobile robot was directed by path remembering function to the next grid and also it detected that there was an obstacle, immediately the current position of the mobile robot became goal position and the original goal position became starting position. Then *compute_shortest_path* function was executed again to find the shortest path from the new goal position to the new starting position. From Test 1, all the five additional features of Enhanced D* Lite Algorithm were successfully implemented in the simulation. In overall, MATLAB simulation has validated the effectiveness of the Enhanced D* Lite Algorithm.

V. REAL-TIME IMPLEMENTATION

The enhanced algorithm is implemented on real life situation using Team AmigoBot™ [7] and the performance is observed. In real life situation, an environment might consist of obstacles, and the locations of the obstacles are unknown for the mobile robot. Moreover, the obstacles might move around from time to time. This is where the sonar sensors of the Team

AmigoBotTM play an important role in detecting obstacles.

Team AmigoBotTM manufactured by MOBILE-ROBOTS Inc. [7] is suitable to be used for this research due to its reasonable size, weight, and TCP/IP (wireless) control. Moreover, the surrounding body of Team AmigoBotTM is equipped with eight sonar sensors.

The working environment for real-time implementation of the enhanced algorithm is a 2500mm by 2500mm area. Within the environment area, there are 10 round-shaped type obstacles randomly arranged. A representation environment is also created using MATLAB to illustrate the real-time path taken by the mobile robot. It is scaled to 100:1 as compared to the actual environment. In this research, the actual environment is divided into grid size of 100mm x 100mm equally. However, the step size could be easily changed to smaller or larger value. Dimensions are in millimeters (mm), and coordinate specification is (x,y). Two tests have been carried out and the comparison of shortest path obtained in real-time implementation and MATLAB simulation.

Referring to the results of Test 2 and Test 3, the same path pattern was obtained for both real-time implementation and MATLAB simulation. For Test 2, no additional obstacles were added for the returning path. For Test 3, two additional obstacles were added for the returning path. When the sonar sensors of the mobile robot had detected additional obstacles in its return path, immediately *compute_shortest_path* function was executed to find the shortest path again from the current position.

VI. CONCLUSIONS

Path finding has been one of the major research topics in robotics field nowadays. Enhanced D* Lite Algorithm has proven its effectiveness in finding a shortest path with a given starting position and goal position and without any information about obstacles positions. In this research, five features explained in the introduction have been verified and proven with MATLAB simulation and real-time implementation. Mobile robot does not traverse across obstacles' sharp corners and does not traverse in between two obstacles, is able to move away from the 'U-Shaped' type obstacles, creates virtual walls if necessary and unnecessary pathways are removed to yield the shortest path. The results obtained from real-time implementation are compared with the MATLAB simulation to verify the practicality and robustness of the proposed algorithm.

Test 2 (Numbers in second column are in mm)

Environment size	2500 x 2500
Starting position	1450,2150
Goal position	1450,250
Number of Obstacles	10. Positions defined by user
Additional Obstacles added for returning path	NO

Table 2: Input Parameters of Test 2

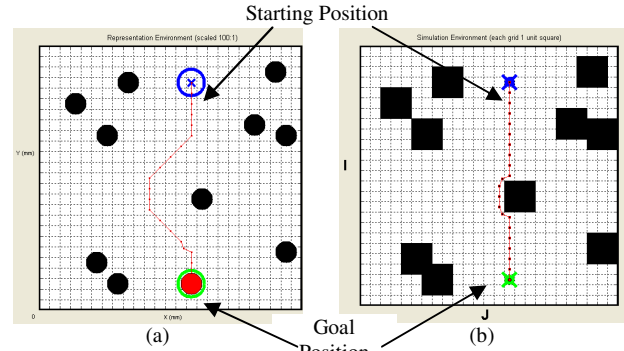


Figure 14: Result of Test 2 in (a) Real-Time Implementation Representation Environment; (b) MATLAB Simulation Environment

Test 3 (Numbers in second column are in mm)

Environment size	2500 x 2500
Starting position	1450,2150
Goal position	1450,250
Number of Obstacles	10. Positions defined by user
Additional Obstacles added for returning path	YES. (1050, 1350) (1250, 1250)

Table 3: Input Parameters of Test 3

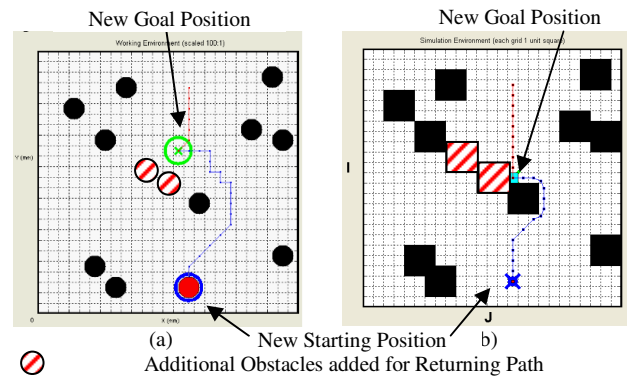


Figure 15: Result of Test 3 in (a) Real-Time Implementation Representation Environment; (b) MATLAB Simulation Environment

ACKNOWLEDGMENT

The authors thank Monash University Sunway Campus for the support of this work.

REFERENCES

- [1] Patrick Lester. "A* Pathfinding for Beginners", retrieved 20 May 2009, from <<http://www.policyalmanac.org/games/aStarTutorial.htm>>, 2009.
- [2] Anthony Stentz. *Optimal and Efficient Path Planning for Partially-Known Environments*, Carnegie Mellon University Robotics Institute, 1994.
- [3] Sven Koenig, Maxim Likhachev. *Improved Fast Replanning for Robot Navigation in Unknown Terrain*, IEEE International Conference on Robotics and Automation (ICRA '02), 2002.
- [4] Maxim Likhachev and Sven Koenig. *Incremental Replanning for Mapping*, Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems EPFL, Switzerland, 2002.
- [5] Sven Koenig, Maxim Likhachev. *Fast Replanning for Navigation in Unknown Terrain*, IEEE Transactions on Robotics, Vol. 21, No 3, June 2005.
- [6] Sven Koenig, Maxim Likhachev, David Furcy. *Lifelong Planning A**, Artificial Intelligence Volume 155, Issues 1-2, Pages 93-146, May 2004.
- [7] MobileRobots Inc., *Team AmigoBotTM Operations Manual* version 4, MobileRobots Inc., 2007.