IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. XII (Mar-Apr. 2014), PP 109-115
www.iosrjournals.org

D_s* Heuristic Approach using 'Safety Distance' for Agent Path Planning

Kiran K Ravulakollu[1], Shailashree K Sheshadri[1]
[1](Dept. of Computer Science and Engineering, Sharda University, Uttar Pradesh, India)

Abstract : *The efficiency in path planning algorithms is a crucial issue in mobile agents. For an artificial agent, observation from environment, navigational behavior and learning methods over occupancy grid maps are tools for effective path planning. The path generated by the conventional D* algorithm may lead to collision with the obstacles in real-time scenario and this issue is addressed by D_s* which uses `safety distance' phenomena, based on weighted cost function. The factors of distance and safety are considered simultaneously in the cost function, thereby demonstrated efficiency in path planning. An analysis on goal-directed navigation tasks in mazes using D_s* heuristic approach is carried out and the efficiency is evaluated based on two parameters - Path length and Execution time.*

Keywords: *D* search algorithm, Heuristic Search, Occupancy Grids, Path Planning, Safety Distance.*

I. INTRODUCTION

Mobile robot vision-based navigation has become the source of numerous research contributions, from the domains of both vision and control. Vision plays a very important role in applications such as localization, automatic map construction, autonomous navigation, path following, inspection, monitoring or risky situation detection. The mobile agent, which uses the artificial intelligence as the base for its functioning, is termed as mobile agent as it is independent of human interference and possesses the decision-making capability. Path planning for mobile agent is one of the critical tasks in unstructured, semi-structured and structured environments for navigation. In this process, it is essential to have a collision-free path between start and destination position continuously. There are two types for mobile agent path planning [1]: Global Planning or deliberative technique and local planning or sensor based planning.

Agent navigation includes the following challenges: Modeling the environment in the understandable way, finding collision-free path from source to destination and sensing the target. The major issue faced during the path planning is searching for a collision-free time-parameterized path. As the agents move in the terrain, they need to observe environment, analyze the trajectories based on the obstacles present and learn their behavior using heuristic based artificial intelligence techniques [2]. With the reduction of possible eventualities, planning process is expected to speedup.

There are various path-planning methods based on an environment map for mobile robots. For example, Dijkstra's algorithm [3]is a kind of most short-path search method, which it guarantees to find a shortest path. A* algorithm [4]is like Dijkstra's algorithm, and it can use a heuristic to guide itself. Fu and Xu [5] proposed one kind of A* algorithm based on restricted searching area to compute the fastest path. Chabini and LAN [6]extends the A* methodology to shortest path problems in dynamic networks, in which arc travel times are time dependent. A variety of algorithms have been developed to accomplish this re-planning described in [7][8][9][10][11][12]). In particular, the D* family of algorithms have been widely used for ground based mobile robot navigation [10][11][12], which are graph-based searches that extend A* while handling the planning and re-planning simultaneously.

Section II discusses the problem statement and the proposed solution i.e., the D_s* algorithm. Section III provides some simulation results of the proposed D_s* path planning method and some compared results are presented with D*. Finally, conclusions are summarized in section IV.

II. PATH PLANNING USING 'SAFETY DISTANCE'

For optimal path planning of agent, the D_s* heuristic based path planning algorithm is executed, which consists of: Image Acquisition, Preprocessing Phase and Path Evaluation Phase (D_s*).These algorithms D* and D_s*are graph-based searches that extend A* to efficiently repair solutions when changes are made to the graph. As such, they have been shown to be orders of magnitude more efficient than planning from scratch each time when new environmental information is received. Although these algorithms were initially designed to operate

over arbitrary graph representations of the environment, they have typically been used to plan over uniform 8-connected 2D grid representations for ground vehicles. The use of such representations has a number of limitations, most notably suboptimal solution paths and significant memory requirements. In an attempt to improve upon these limitations, interpolation-based re-planning algorithms have recently been developed [11] [13]. These algorithms produce much better solutions through both uniform grids (thus addressing the path quality limitation of standard 8-connected 2D grid-based planners) and non-uniform 2D grids (thus allowing for much less memory-intensive representations to be used).

1) Image Acquisition

The image of the environment is captured through the camera. At every stage, based on maximum detection range of camera, the image is captured which is visualized as grid. As the agent moves ahead, the field of view is incremented which leads to the generation of incremental grids. Integration of partial grids is considered the final image. As the external sources are used for acquisition, the care must be taken to avoid the noises and the distortion. Cameras are the fundamental robot inputs for the process of perception, and therefore the degree to which cameras can discriminate the world state is critical. Camera's noise induces a limitation on the consistency of readings in the same environmental state and, therefore, on the number of useful bits available from each reading [15]. Often, the source of noise problems is that some environmental features are not captured by the agent's representation and are thus overlooked. The navigation platform considered is a 2D representation of the environment in terms of occupancy grids. The free space and the obstacles are viewed as binary images. The configuration space determines the feasible paths available for navigation.

B. Preprocessing Phase

This phase deals with Localization, Mapping and obstacle detection as well. The environment under consideration comes via image acquisition. The preprocessing stage is essential to ensure that the agent understands and learns its environment visually. This phase must achieve the accurate results with minimal computational overhead.

1) Obstacle Detection

From the binary image, the obstacles are detected based on the pixel intensity (pixel value=0 is generally considered as obstacle) as shown in Figure1.

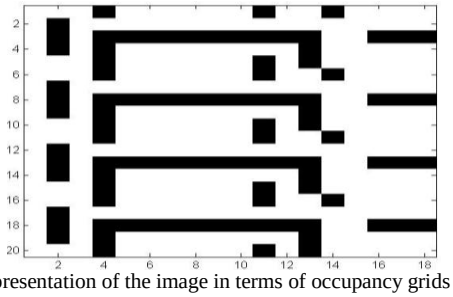

Figure1: The representation of the image in terms of occupancy grids – Binary Image

This image is subjected to corner detection and edge detection, in which Harris corner detector [16] is used to perform the initial task of locating corners. The edges are detected using threshold canny edge detector. The full edge detector will not be useful for detecting the obstacles on the image. The resulting edge detection image will leave only edges from objects that are evident on the image. Now, the boundaries of the obstacles are extracted and this helps the agent to analyze the free space for navigation as shown in Figure 2.

Figure 2: The boundaries of the obstacle detected

C. Path Estimation

There is a continuous path from Start(S) to Target (T) where S is the parent node. The Target (T) location is user-defined. The total distance D is estimated which is the Euclidean distance from S to T. There will be 'n' number of obstacles of definite geometric shape. The agent during navigation, when approaches the obstacle, then a point called focal point *F* is defined. As there are *n* obstacles, there will be *n* focal points. Let P be the total length of connected focal lengths F from S to T.

$$P = \sum_{n=1}^{s+1} (F_{n-1}, F_n) \qquad (1)$$

The total path P is evaluated as per the function (1) determines the cost of all the feasible paths generated at every focal point F. Let d_m denote the line between the current focal point and the Target (T). The Target point, Current focal point and previous focal point form a triangle. When, the agent is at S, the value of $d_m[0] = D$, which is the constant value. These values are accumulated in an array starting from 0 to last focal point before the target point. The further values are calculated as defined in (2):

$$d_m\,[1] = a = \sqrt{(b^2 + c^2\ - 2bccosA)} \qquad (2)$$

Where, b= distance between the current focal point and previous focal point obtained from the camera, c = previous value ofd_m.

D. Path Execution – D_s* Algorithm

The name of algorithm, D*, was chosen because it resembles A*, except that it is dynamic in nature that are cost parameters which change during planning. Provided that agent steps are properly coupled to the algorithm, D* generates optimal trajectories. With D*, we can obtain global knowledge of goal from any position in environment. Global knowledge is called in our case, as global path information is back-pointer for direction. Back-pointers are determined from cost calculation to all positions in agent space navigation. D* algorithm combines local and global planning by makes sufficient use of global environment from sensors. This algorithm is applied in partially known or changing environment. It produces an optimal path from start to goal by minimizing a predefined cost function. It has capability of rapid re-planning, and has been used in real time planning in challenging terrains. The path planning which Sojourner [17] used was D*, that made Sojourner avoid carefully distinguishing obstacles on the path and replan in real time for the next movement.

1) Issues in D Algorithm*

Commonly, '*shape*' of agent is ignored, and the position of agent is expressed by parameter of state. The D* algorithm mainly focuses on generating the shortest path which sometimes becomes less practical and obsolete in the complicated environment with many obstacles* sets agent transition from start to goal across directional arcs. It maintains a list of states queued for cost expansion, initially with goal state set to zero. This list is called the open list, as it contains states "open" for expansion. The state with the minimum path cost on the open list is repeatedly expanded, propagating path-cost calculations to its neighbours. As agent moves, it may detect new obstacles or absence of expected obstacles.

When obstacle is detected, arc of the path passing through this obstacle is marked with a large value and the adjoining states are set on open list for cost correction. Encountering unexpected obstacles will set off a "raise" wave, a wave of increasing cost, through neighbouring states. When this wave meets with states that are able to lower path costs, these "lower" states are put on the open list to recalculate new optimal paths. When it detects the absence of an expected obstacle, the arc of the path passing through this missing obstacle is assigned a small cost, denoting an empty space, and the adjoining state is put on the open list as a lower state, setting off a "lower" wave, a wave of decreasing cost. D* is able to determine how far the raise and lower waves need to propagate while providing the optimal path for robot traverse continuously [18].

2) D_s Algorithm*

The D_s* algorithm consists of three functions: *Safety-State* (Figure 3), *Process-State* and *Modify-Cost*. After applying the implemented algorithm on different grid maps, we found that it generates a path from the

start point in front of the robot to the goal location. This path is efficient in terms of the minimum cost required to reach the target (i.e. the cost function is the min path length). D_s* will provide the shortest, lowest cost path and also automatically gives the smoothest path. Relative position between the current position and that of the next move must be taken into consideration to improve performance.

The *Safety-State* () function estimates the criteria whether the agent can pass through the obstacles and the convergence of the algorithm. A safe and a distance movement characteristic of the node are considered in the proposed method, thus it can determine an appropriate path that avoids the collision with obstacles and moves to the destination quickly for the agent which is calculated by safe (n) represented in (3). This value is calculated using the minimum distance between the obstacles, the distance towards the target from the current focal point and the entire distance between Source and Target. To generate a smooth path, size of the agent is considered by considering the number of nodes equal to the size of an agent when testing the candidate successor nodes. This leads to the testing more than one node together to direct the possible direction of movement.

$$safe(n) = \begin{cases} d_m(n)e^{-\left(\frac{k_{min}(n)}{D}\right)^2}, k_{min}(n) > A_s, o(n) \neq 0 \\ 0, \quad otherwise \end{cases} \qquad (3)$$

A_s represents the agent size, $d_m(n)$ represents distance of the agent at the focal point f_n towards the target point (T) and $k_{min}$is the minimum distance of between the obstacles which has to be greater than the A_s.To determine that the agent leaves the obstacle boundary when the target becomes visible or when it is able to pass through the obstacles without collision, the convergence of the algorithm is important which is provided by(4).

$$d(F,T) - F \leq d_m(T) - \min(k_{min}, R) \qquad (4)$$

This equation provides the guarantee that the distance to the target from the current focal point F always decreases at least by $\min(k_{min}, R)$ between successive focal points. This ensures that algorithm always converges.

```
Safety-State(k_min,A_s){
        F=Free Space; R=Maximum Range of the Camera;
        if no obstacle
        then F = R;
elseif target not visible
        thend(F,T) − F ≤ d_m(T) − min(k_min, R)
        else  \\leave obstacle when the target is visible
                                                d(F,T) − F ≤ 0
        end
  if (A_s>k_min)
  then calculate safe(n)
        compute f(n) = g(n)+h(n);
  end if
end if
```

Figure 3: Pseudocode for Safety-State function

After the safe(n) is calculated the other 2 functions Process-State and Modify-Cost are calculated and modified according to g(n) and h(n) which determines the heuristic cost.

- $g(n)$ = OPEN list = D_s* maintains this list to store the path costs of already visited node. The value is taken from $c(X;\ Y)$ which is an arc cost function.
  The actual cost of the current path from the source node S to the current expanding node n is given by:

$$g_s(n) = g(n) + safe(n) \qquad (5)$$

The traditional D* algorithm in the path planning is chiefly focusing on calculating the shortest path. However, the shortest path becomes less practical and helpful in the complicated environment with many obstacles. The concern of this paper is how to arrive at the destination in the most safe and fastest way.

Therefore, g(n) (the actual cost of the current path from the start node S(0, 0) to the current expanding node n) is modified as per (5).

- $h(n)$ = $h(G;X)$ = D_s* maintains the sum of the arc costs from State X to Target T which is calculated using Euclidean distance.In this paper, the grip map is used and the distance relation between two nodes is described in Figure, where d denotes the direct distance of two nodes.

$\sqrt{2}\cdot d$	d	$\sqrt{2}\cdot d$
d	node	d
$\sqrt{2}\cdot d$	d	$\sqrt{2}\cdot d$

Figure 4: Distance relation between two nodes

The h (n) is the heuristic estimation of the minimum cost of a path from the current expanding node n to the target node T denoted by (6):

$$h(n) = d * \sqrt{(x_n - x_t)^2 + (y_n - y_t)^2} \quad (6)$$

Based on D_s* algorithm, the optimum path, is determined by tracing back through prev (T) from the start node T firstly and back to the start node S finally, where prev (n) =F_N, and $F_n$is the current focal point of the n^{th} step.

III. EXPERIMENTAL RESULTS

The evaluation criteria for the D_s* algorithm is based on qualitative analysis. The environment is depicted in terms of occupancy grids as a binary image. The 0's represent obstacles or blockage in path while 1's represent free space. With this the nodes and partial graph structures are generated which acts as an input to the D_s* algorithm. The D_s* algorithm is implemented using the ***ROBOTICS-9.8*** toolbox available in MATLAB.The proposed D_s* algorithm is shown in Figure 5, where S= (1, 4) is a start node, T= (6, 1) is a goal node.

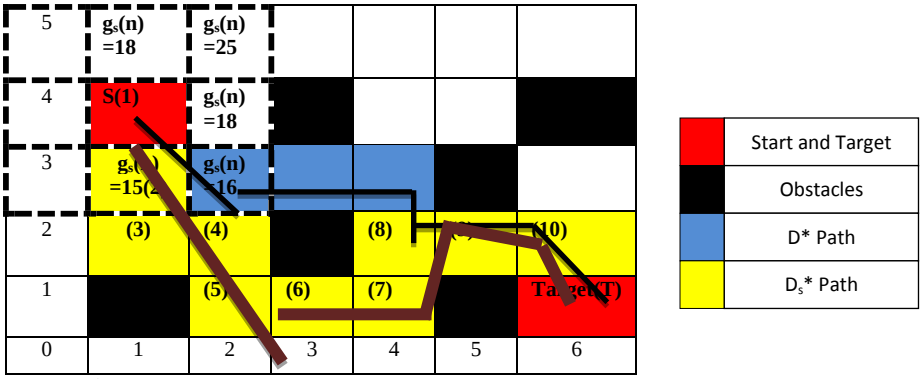

Figure 5: Example explanation of the proposed D_s* algorithm, where (1, 4) is a start node and (6, 1) has a lowest score.

As shown in Figure, there are ten searching steps to find the optimal path S=(1,4)-(1,3)-(1,2)-(2,1)-(3,1)-(4,1)-(4,2)-(5,2)-(6,2)-(6,1)=T. The D_s* algorithm uses 4-connectivity for calculation. The $g_s(n)$ determines the actual cost of the current node navigated and whichever step generates the minimum $g_s(n)$, that path is chosen for navigation.

*Outputs of D_s*path planning approach:*

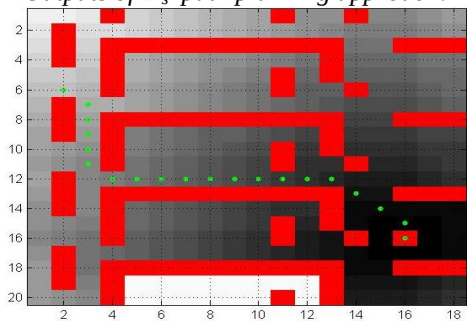

Figure 6: Path generated from D_s* path planning. S(2, 6) and T(16, 16), As = 0.5 units. Execution Time = 48ms.

In this experiment, the agent size considered is half the grid block. The D* takes 21 steps and 56ms to reach the target. The D_s* takes 26 steps but reaches the target quickly in 48 sec. This is due to the safe distance value which gets added to the distance function. The number of steps taken to reach the target is more due to the agent size constraint. This proves the efficiency of D_s* algorithm.

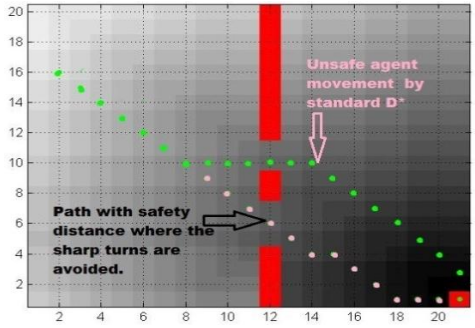

Figure 7: Example 1 - The path generated by both D* and D_s*

In this experiment, the agent size considered is one grid block. The D* takes 20 steps and 22ms to reach the target. The Ds*takes 21 steps but reaches the target quickly in 18 sec. This is due to the safe distance value which gets added to the distance function. The number of steps taken to reach the target is more due to the agent size constraint. This proves the efficiency of Ds* algorithm. When the agent reaches the wall or the boundary, it follows the standard wall avoidance algorithm to navigate towards the target.

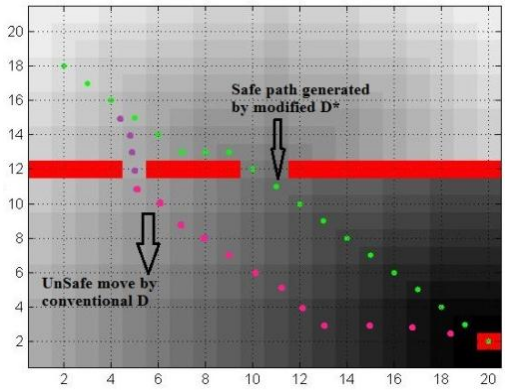

Figure 8: Example 2 - The path generated by both D* and D_s*

In this experiment, the agent size considered is one grid block. The D* takes 24 steps and 26 ms to reach the target. The D_s* takes 19 steps but reaches the target quickly in 28 sec. This is due to the safe distance value which gets added to the distance function. The number of steps taken to reach the target is more due to the agent size constraint. This proves the efficiency of D_s* algorithm. When the agent reaches the wall or the boundary, it follows the standard wall avoidance algorithm to navigate towards the target.

For execution of D_s*, the size of the agent considered is 1 unit rather than a point- mass agent. The navigation is done by first considering the size of an agent with the periphery of the obstacle encountered. If an agent can move between the obstacles, then safe (n) value is calculated based on(4).

Table I. COMPARITATIVE ANALYSIS

Algorithm	Occupancy Grid No.	Agent Size	Path Length (Nodes)	Execution Time (ms)
D*	Figure6	NA	21	56
	Figure7		20	22
	Figure 8		24	26
D_s*	Figure 6	0.5 unit	25	48
	Figure 7	1 units	20	18
	Figure 8	1 units	19	28

The path length depends on the $g_s(n)$ which indirectly depends on the safe (n). The accuracy of the algorithm depends on the distance estimation and accurate mapping of focal points. The execution time depends on the heuristic estimation of the path as explained in (6).

IV. CONCLUSIONS

The path generated by D* can lead to collision as it cannot generate safe paths and sub-optimal paths. Through this paper, we have proposed a modified D* algorithm, which guarantees planning of optimal path for mobile agents ensuring the safety and quick navigation. Application of cosine rule for estimating node distance is stabilized with the help of 3 different levels, in term of grid complexities. The variation in the path length is caused due to the safety distance parameter, which in some cases is greater than the path cost of D*. Finally, the paper concludes that a safety distance criterion is valid and modified D* can always guarantee safe path planning along with improvisation in performance at cases. Furthermore, the criteria can be verified on more complex environments with greater branching factor to validate efficiency and capabilities.

REFERENCES

[1] H. Zhang, J. Butzke, and M. Likhachev, "Combining global and local planning with guarantees on completeness", *In IEEE International Conference on Robotics and Automation*, 2012.

[2] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, Cambridge, MA: MIT Press, 2005.

[3] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for Graphs with Weighted Vertices", *IEEE Transaction on Computers*, vol.47, no.2, 1998.

[4] T. Goto, T. Kosaka, and H. Noborio, "On the heuristics of A* or a algorithm in ITS and robot path-planning," *2003 IEEWRSJ Intl, Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2003.

[5] M. Fu and B. Xue, "A path planning algorithm based on dynamic networks and restricted searching area", *IEEE International Conference on Automation and Logistics*, pp.1193-1197, 2007.

[6] I. Chabini and L. Shan, "Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks", *IEEE Trans. on Intelligent Transportation Systems*, vol.3, no.1, pp.60-74, 2002.

[7] A. Barto, S. Bradtke, and S. Singh, "Learning to Act Using Real-Time Dynamic Programming," Artificial Intelligence, vol. 72, pp.81–138, 1995.

[8] T. Ersson and X. Hu, "Path planning and navigation of mobile robots in unknown environments," in Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), 2001.

[9] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," Journal of Algorithms, vol. 21, pp. 267–305, 1996.

[10] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," in Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1995.

[11] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2002.

[12] D. Ferguson and A. Stentz, "The Delayed D* Algorithm for Efficient Path Replanning," in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2005.

[13] D. Ferguson and A. Stentz, "Multi-resolution Field D*," in Proceedings of the International Conference on Intelligent Autonomous Systems (IAS), 2006.

[14] A. Cherubini and F. Chaumette, "A redundancy-based approach for obstacle avoidance in mobile robot navigation", IEEE/RSJ Int. Conference on Intelligent Robots and Systems, 2010.

[15] N.M.Wardhana, H. Johan, andH. S. Seah, "Enhanced waypoint graph for surface and volumetric path planning in virtual worlds,"TheVisual Computer, vol. 29, no. 10, pp. 1051–1062, 2013.

[16] C.J. Harris and M. Stephens, "A combined corner and edge detector", *4th Alvey Vision Conference*, Manchester, 1988.

[17] D. Bo, X. Xiao-ming, C. Zi-xing, *Current Status and Future Development of Mobile Robot Path Planning Technology*, Control Engineering of China, 12(3):198-202, May 2005.

[18] J. Guo, L. Liu, Q. Liu, Y. Qu, "An Improvement of D* algorithm for Mobile Robot Path Planning in Partial Unknown Environment", *Second International Conference on Intelligent Computation Technology and Automation, 2009.*