Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

3D Field D*: Improved Path Planning and Replanning in Three Dimensions

Joseph Carsten*, Dave Ferguson, and Anthony Stentz
Carnegie Mellon University
Pittsburgh, PA
{jcarsten@alumni.cmu.edu, dif@cmu.edu, tony@cmu.edu}

***Abstract*— We present an interpolation-based planning and replanning algorithm that is able to produce direct, low-cost paths through three-dimensional environments. Our algorithm builds upon recent advances in 2D grid-based path planning and extends these techniques to 3D grids. It is often the case for robots navigating in full three-dimensional environments that moving in some directions is significantly more difficult than others (e.g. moving upwards is more expensive for most aerial vehicles). Thus, we also provide a facility to incorporate such characteristics into the planning process. Along with the derivation of the 3D interpolation function used by our planner, we present a number of results demonstrating its advantages and real-time capabilities.**

I. Introduction

Basic path planning consists of finding a sequence of state transitions that lead an agent from a given start state to a given goal state. The optimal path is the sequence of state transitions with the lowest combined cost of traversal. Path planning for a mobile robot is complicated by the fact that a robot usually starts with little or no knowledge about the terrain it will traverse. Instead, it is equipped with sensors that are able to perceive the nature of the environment in its vicinity. This information can be used to update its representation of the environment as it moves. Unfortunately, these updates may cause its current path to become invalid or suboptimal. It is therefore necessary to replan or repair its path in light of this new information. If the original information concerning the environment is grossly inaccurate or incomplete, this updating and replanning step must be completed at almost every stage of the robot's traverse. Therefore, it is important that this can be performed efficiently.

A variety of algorithms have been developed to accomplish this replanning (e.g. [1], [2], [3], [4], [5], [6], [7]). In particular, the D* family of algorithms have been widely-used for ground based mobile robot navigation [4], [5], [6]. These algorithms 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 new environmental information is received [4].

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 replanning algorithms have recently been developed [7], [8], [9]. 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).

In this paper, we build on this work and present an interpolation-based planning and replanning algorithm for generating low cost paths through three dimensional grids for aerial and underwater vehicles. Because such vehicles often find it much more difficult to move in some directions than others (e.g. moving upwards can be very expensive for aerial vehicles), we also provide a method for incorporating directional cost biases into the planning process. We begin by introducing the general D* and D* Lite algorithms. Next, we describe Field D*, an extension of these algorithms that uses interpolation to produce less costly paths through 2D grids. We go on to describe current research on 3D planning and then present a 3D version of Field D*. Finally, we provide example paths and a series of results demonstrating the advantages of our algorithm and its real-time capability.

II. The D* Lite Algorithm

D* and D* Lite are graph-based planning algorithms that are able to efficiently handle changes to the costs of arcs in the graph [4], [5]. We restrict our discussion to D* Lite because this algorithm is somewhat simpler than the original D* algorithm.

A. Generic Algorithm

Given a set of nodes S, a successor function $Succ(s)$ that specifies, for each node s, the set of nodes that are adjacent to s, and a cost function c, where $c(s, s')$ specifies the cost of transitioning from s to an adjacent node s', D* Lite plans a least-cost path from an initial node s_{start} to a desired goal node s_{goal}. In practice, it is useful to also have a function $Pred(s)$ that specifies the nodes to which s is adjacent. In addition, there may be several acceptable goal

*Joseph Carsten is now at the Jet Propulsion Laboratory in Pasadena, CA.

1-4244-0259-X/06/$20.00 ©2006 IEEE

nodes, in which case the plan returned will be the overall least-cost path to any of the goals.

D* Lite maintains an estimate of the cost, $g(s)$, of the optimal path from each state s to the goal, along with a one-step look ahead cost, $rhs(s)$, which is computed as

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ \min_{s' \in Succ(s)}(g(s') + c(s, s')) & \text{otherwise.} \end{cases}$$

Nodes s for which $g(s) = rhs(s)$ are called consistent nodes. If a node s is inconsistent, i.e., $g(s) \neq rhs(s)$, the current value of $g(s)$ may not represent the optimal cost to the goal. Search is performed from the goal node towards the start node. As in A*, a heuristic, $h(s, s')$, is used to guide the search. The heuristic estimates the cost of moving from s to s' and should satisfy

$$h(s, s') \leq c_{optimal}(s, s')$$
$$h(s, s'') \leq h(s, s') + h(s', s'').$$

A priority queue stores all inconsistent nodes. These nodes are sorted according to their key values $k_1(s)$ and $k_2(s)$:

$$\begin{aligned} k_1(s) &= min(g(s), rhs(s)) + h(s_{start}, s) \\ k_2(s) &= min(g(s), rhs(s)). \end{aligned}$$

Node s appears before node s' in the queue if $k_1(s) < k_1(s')$ or both $k_1(s) = k_1(s')$ and $k_2(s) < k_2(s')$. During the search, nodes are processed according to their ordering in the queue. If $g(s) > rhs(s)$ for the current node s at the top of the queue, then s is an overconsistent node. In this case, s is made consistent by setting $g(s)$ equal to $rhs(s)$. Then, the rhs values for all predecessor nodes of s are re-computed. If, on the other hand, $g(s) < rhs(s)$, then s is underconsistent. In this case, the value of $g(s)$ is set to ∞ and s is placed back on the queue as an overconsistent node (assuming $rhs(s) \neq \infty$). Once again, rhs values for the predecessors of s are re-computed. Any nodes that have been made inconsistent are added to the priority queue to be processed. Once the start node is consistent and there are no nodes left on the priority queue with a priority less than that of the start node, we are guaranteed to have found the optimal path.

Within this framework, coping with changes to arc costs is straightforward. If $c(s, s')$ changes, $rhs(s)$ is re-computed. If this causes s to become inconsistent, it is inserted into the priority queue. Once again, nodes on the queue are processed until the start node is consistent and there are no nodes left on the queue with priorities less than that of the start node. This incremental repair can be much more efficient than planning from scratch, especially if cost changes occur near the start position, as is often the case with sensor-equipped mobile robots.

B. Typical Implementation

In order to use D* or D* Lite for an actual path planning problem, we need to define the set of nodes S, the functions $Succ$ and $Pred$, and the cost function c. A simple way to define these elements is to partition the n-dimensional planning space into a uniform n-dimensional grid. The set of nodes S can then naturally be defined as the set of vertices of the grid, and $Succ(s) = Pred(s)$ can be defined to be the $3^n - 1$ closest vertices to node s. Each grid cell can be given a cost of traversal based on the cost of the corresponding area of the planning space, which can then be used to calculate the cost function c. Given these constructs, D*/D* Lite can then be used to plan paths.

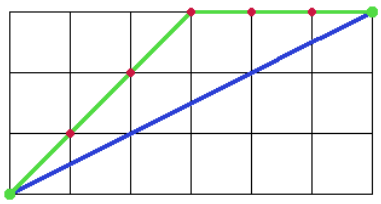


Fig. 1. Path planning in a uniform cost 2D environment from the lower-left corner to the upper-right corner. The optimal D* path is shown in green/above and the true optimal path is shown in blue/below.

Although the paths found using D*/D* Lite are optimal given the graph used for planning, in most cases they do not represent optimal paths given the original problem. Fig. 1 illustrates a typical situation encountered when using 2D uniform grids. In this case, the D* path is eight percent longer than a true optimal path. This is a direct result of how the problem has been defined. Using a uniform 2D grid to discretize the space, each node is adjacent to its eight neighbors. Therefore there are only eight distinct directions that can be used when planning paths and so, unless by chance some optimal path only requires these eight directions, the resulting paths will be suboptimal.

III. Field D*

The Field D* algorithm was devised to help alleviate problems created by using 2D grids to discretize the environment [7]. With Field D*, each node s is able to transition to any point on an adjacent grid cell *edge*, rather than just the endpoint nodes of these edges. This removes the heading constraints placed upon the path, allowing for a continuous range of headings and more direct, less-costly solutions.

If we knew the value $g(s_e)$ of every point s_e along these edges, then we could compute the true optimal value of node s simply by minimizing $c(s, s_e) + g(s_e)$, where $c(s, s_e)$ is the cost associated with an optimal local path from s to s_e. Unfortunately, there are an infinite number of such points s_e, and therefore computing $g(s_e)$ for each of them is not possible. Instead, Field D* uses linear interpolation to approximate $g(s_e)$. Given a point s_e residing on the edge between two grid nodes s_1 and s_2, its cost is assumed to be a linear combination of $g(s_1)$ and $g(s_2)$:

$$g(s_e) = y * g(s_2) + (1 - y)g(s_1), \quad (1)$$

where y is the distance from s_1 to s_e (assuming unit cells). See Fig. 2(right) for an illustration. Given this approximation, the optimal path from node s through the edge between nodes s_1 and s_2 can be computed in closed form – both the best node s_e on the edge and the best

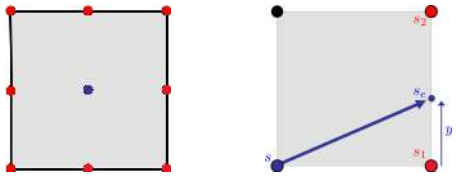


Fig. 2. (left) The adjacent edges (in bold) of a given 2D grid node (node in center). (right) Interpolation can be used to plan paths from a node s to any point s_e on an adjacent edge (here the edge between nodes s_1 and s_2 is being considered), rather than just the endpoints of these edges.

path to that node from s can be solved for concurrently [7]. The look ahead path cost $rhs(s)$ for node s can then be calculated as the cost associated with the least-cost path through any of the eight adjacent edges to s. In other words, the $Succ$ function used by Field D* comprises not a set of adjacent nodes to s, but rather a set of adjacent edges (see Fig. 2(left)).

This approach provides paths that are able to enter and exit cells at arbitrary positions, rather than just at the corners, and leads to much better paths through 2D grids. Further, this interpolation-based path cost calculation can be used to compute a path from any position within the grid, not just from the grid corners. See [7] for more details on Field D* and its application to 2D path planning.

IV. Planning in Three Dimensions

In three dimensions, the path planning problem is even more computationally expensive [10]. A number of approximation algorithms have been developed for generating suboptimal paths through continuous 3D space [11], [12], [13], [14], [15], but these algorithms typically do not incorporate regions of non-uniform cost in the environment. The existence of such areas, which are common in robotic path planning, makes the planning problem even more challenging. Thus, in robotics the problem is often simplified by using a (non-uniform cost) 3D grid to represent the environment and then extracting from this grid a graph to plan over, exactly as is done for the 2D case [16], [17]. Such an approach is fast and can encode the variations in traversal cost for different areas of the environment.

However, 3D grid-based adjacency graphs suffer the same limitations as their 2D counterparts. In particular, paths produced using these graphs will be suboptimal and will involve unnecessary turning. Ideally, we would like a way to produce more direct, less-costly paths through 3D grids without sacrificing the efficiency of standard 3D grid-based planning. To do this, we extend the Field D* algorithm to use interpolation in 3D. As we will see, this involves a more complex minimization problem that we tackle in the next section.

V. Field D* in Three Dimensions

To more accurately approximate true optimal paths through three dimensional grids, we need to overcome the discretization effects of standard 3D grid-based planning and instead allow for a continuous range of headings. As in the 2D case, this can be done by using interpolation to

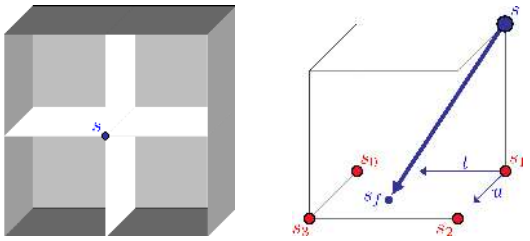


Fig. 3. (left) The adjacent faces (shaded) for a given node s – shown are the 12 adjacent faces for the four voxels that s resides on and that are into the page; a symmetric case exists for the four voxels coming out of the page. (right) Interpolation can be used to approximate the path cost of an arbitrary face point s_f.

provide path cost estimates for points not residing on the corners of the grid. However, rather than computing the path cost of a node s by looking at its adjacent edges and using interpolation to provide paths through arbitrary points on these edges, we look at the adjacent *faces* of s and plan paths through these faces. In the two-dimensional case each node had eight adjacent edges; in the three-dimensional case each node has twenty four adjacent faces (see Fig. 3(left)).

A. Using Interpolation in 3D

In order to calculate the least-cost path from a node s through an adjacent face f, we can use interpolation to approximate the cost of any point s_f on the face. Fig. 3 illustrates the planning scenario. Given the path costs of the nodes on the four corners of the face, $g(s_0), g(s_1), g(s_2)$, and $g(s_3)$, the path cost of any point s_f on the face can be approximated using linear interpolation by

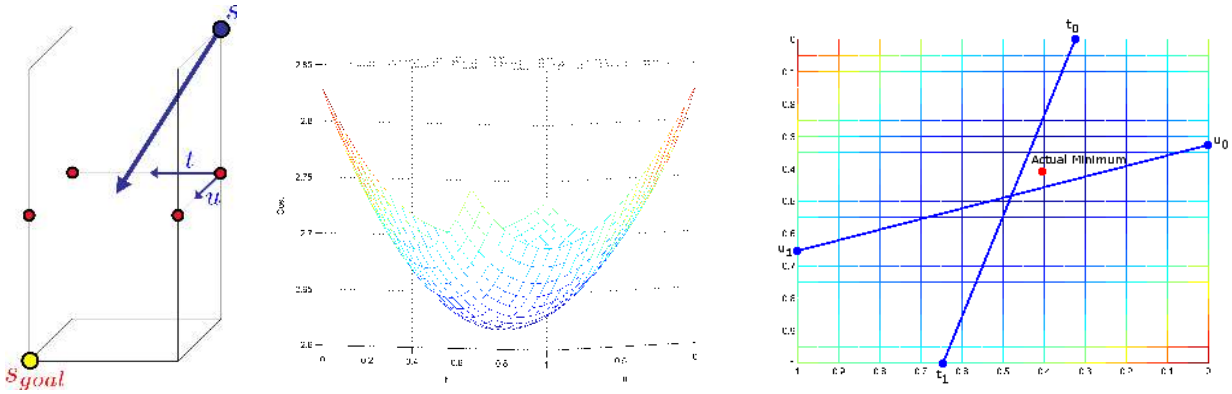
$$\begin{aligned} g(s_f) &= [(g(s_1) + (g(s_0) - g(s_1)) \cdot t] \cdot (1-u) \\ &+ [g(s_2) + (g(s_3) - g(s_2)) \cdot t] \cdot u, \end{aligned} \quad (2)$$

where t and u represent the position of point s_f in face coordinates (see Fig. 3(right) for an example). Given this approximation for the path cost of any point on face f and assuming a traversal cost of C for the voxel on which both f and s reside, we can compute the cost of a path from s through s_f, denoted $rhs_{s_f}(s)$:

$$\begin{aligned} rhs_{s_f}(s) &= C \cdot \sqrt{1 + t^2 + u^2} \\ &+ [(g(s_1) + (g(s_0) - g(s_1)) \cdot t] \cdot (1-u) \\ &+ [g(s_2) + (g(s_3) - g(s_2)) \cdot t] \cdot u. \end{aligned} \quad (3)$$

B. Path Cost Minimization

To compute the approximate least-cost path from s through an adjacent face f we need to minimize (3) with respect to both t and u. Unfortunately, this has no closed form solution. Of course it is possible to use numerical optimization to obtain a solution, but this is too slow to be of use for a real-time planning algorithm: this minimization needs to be performed tens to hundreds of millions of times for a moderately-sized planning problem. In order to achieve real-time performance, a result must be returned extremely efficiently. To accomplish this, we trade



(a) Planning Problem (b) Cost Surface (c) Approximate Solution

Fig. 4. A simple planning problem through two voxels of unit cost. (a) The start node s is in the upper right and the goal node s_{goal} is in the lower left. The cost surface is shown in (b) and a top-down view with the edge minima and our approximate solution (the intersection of the two lines) is shown in (c).

optimality for speed and use an approximate solution that is easily computed.

Fig. 4(a) shows a simple planning problem involving two voxels that can be used to illustrate our approximation technique. Fig 4(b) shows the cost of moving from node s through any point on the center face to the goal, assuming linearly-interpolated path costs for all points s_f on the face. Our goal is to find where the minimum of this surface resides: this represents the best point s_f on the face through which to transition.

In any minimization problem it is possible that the minimum lies on a boundary of the domain. During the minimization process, the boundaries must be checked for this condition. In our case, the boundaries are each of the four edges along the face. Finding the minimum along each edge is straightforward. In fact, it is nearly identical to the interpolation-based edge calculation for the two dimensional case discussed in Section III.

Once we have found the location of the minima along the edges, we use this information to help us find a possible minimum on the interior of the face. Connecting the minima along both pairs of opposite edges gives us two intersecting lines. We then test the point where these lines intersect to determine if it is of lower cost than any of the edge minima. Fig. 4(c) illustrates the situation for our example. The values of t_{int} and u_{int} at this intersection point can be calculated using (4) and (5) where t_0, t_1, u_0, and u_1 are each of the edge minima as shown in Fig. 4(c).

$$t_{int} = \frac{(t_1 - t_0) \cdot u_0 + t_0}{1 - (t_1 - t_0) \cdot (u_1 - u_0)} \quad (4)$$

$$u_{int} = (u_1 - u_0) \cdot t_{int} + u_0 \quad (5)$$

This method produces good approximations of the least-cost path given our linear interpolation assumption, and can be computed very efficiently. Given this technique, we can compute approximate least-cost paths from a node s through each of its adjacent faces f. From these paths, the one with minimum cost can be used as the approximate least-cost path from s, and its cost can be used to update $rhs(s)$. This new update can then be inserted into the D* Lite algorithm to provide an extension of Field D* to three-dimensional grids.

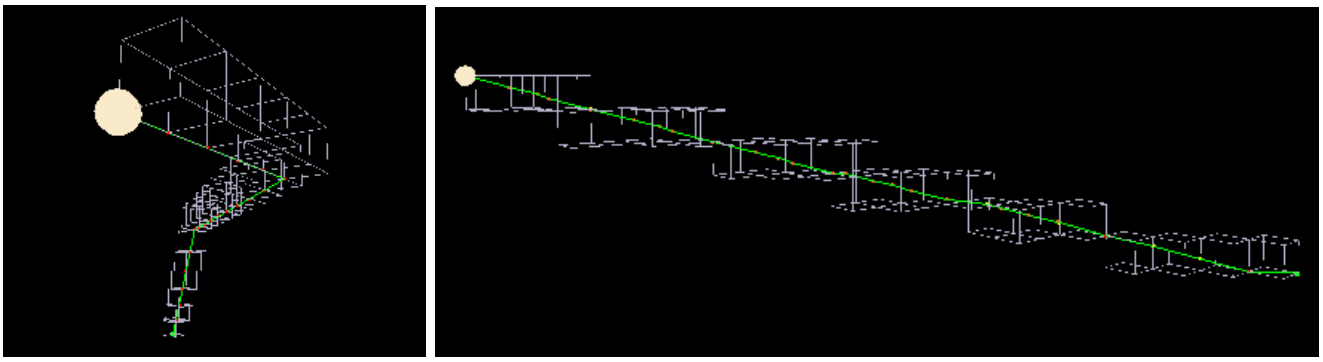
C. Incorporating Directional Transition Costs

In a variety of planning problems, traveling in one direction may be more costly than traveling in another. For example, it is generally more costly in terms of time and fuel for an aerial vehicle to travel upward than to move horizontally. It would be useful to be able to incorporate this information into the planning process.

One way to accomplish this would be to store multiple costs for each voxel. For example, one cost for upward travel through the voxel, one for downward, and one cost for everything else. The problem with this approach is that it requires three times the memory to store the cost field. This can be significant even in the case of a moderately-sized map. In addition, to support the most general case, six costs would need to be stored for every voxel.

Another option is to use global scale factors. For example, upward motion could be assigned to be twice as expensive as other directions of travel. If the base cost of moving through a voxel is ten, then a cost of twenty would be used for upward motion. Using global scale factors has several advantages over storing multiple costs for each voxel. Only six numbers need to be stored, one scale factor for each direction, regardless of map size. In addition, because the scale factors are global, they remain constant in every calculation that is done. This makes it easy to precompute some frequently used quantities involving these scale factors. Being able to precompute these values allows for better optimization and less overall computation time.

One disadvantage to the global scale factor approach is that it is less general than the first approach, in that it does not allow for arbitrary local cost variations. In the first approach, directional costs can be varied on a voxel by voxel basis. With global scale factors, the directional costs are fixed over the entire map. In most cases, the increased efficiency in terms of both memory usage and computation time is well worth the small loss in generality.



(a) Regular 3D Voxel Planner (b) 3D Field D* Planner

Fig. 5. Planning a path through an open environment from an initial location (large white sphere) to a goal location (small shaded cube). Shown are the paths produced by (a) a standard 3D grid-based planner, and (b) 3D Field D*. Also shown are the voxels the paths travel through. Notice that the path returned by the standard 3D grid-based planner is unable to travel in a straight line to the goal. In contrast, the Field D* path is able to produce almost the optimal straight line path, with only minor deviations due to its interpolation-based cost approximation.

In order to use global scale factors, the path cost calculation corresponding to Fig. 3 changes from that shown in (3) to

$$\begin{aligned} rhs_{s_f}(s) &= C \cdot \sqrt{c_z^2 + (c_x \cdot t)^2 + (c_y \cdot u)^2} \\ &+ [(g(s_1) + (g(s_0) - g(s_1)) \cdot t] \cdot (1 - u) \\ &+ [g(s_2) + (g(s_3) - g(s_2)) \cdot t] \cdot u. \end{aligned} \quad (6)$$

The parameters c_x, c_y, and c_z correspond to scale factors for moving left, out of the page, and downward, respectively, in relation to the coordinates shown in Fig. 3. Substituting this equation for (3) allows for directional scaling to be taken into account during planning.

VI. Results

Using the methods described, a three-dimensional version of Field D* Lite was implemented. We have found this algorithm to produce nice, straight paths that require much less unnecessary turning than regular 3D grid-based paths. Fig. 5 shows an example path produced using a regular 3D grid-based planner and our implementation of 3D Field D*. While the regular planner returns a path consisting of three line segments with sharp 45 degree turns connecting each segment, Field D* returns roughly a direct path to the goal location, with very minor deviations due to its interpolation-based approximation.

We also ran several experiments to provide an indication of the computation required by Field D*. All experiments were performed on a 1.9 GHz Pentium P4 PC with 512 MB of RAM and involved 3D grids of dimensions $150 \times 150 \times 150$. Table I shows results for expanding the entire cost field. In these tests, the goal was placed on the center of one side of the map and every reachable node was expanded. Voxel costs were stored as integer values ranging from free (255) to obstacle (65535). Beginning with a map consisting only of free cost voxels, maps were constructed by randomly changing the cost of a specified fraction of the voxels. This fraction is given in the first column of the tables as the 'Density' value. For entries marked 'Bin', non-free voxel costs were changed to obstacle cost, thus yielding a binary map of only free and obstacle voxels. For all other cases, selected voxel costs were changed to a random value between free and obstacle.

Density	rhs_{s_f}	rhs_{s_f} Time (s)		Total Time (s)		
	($\times 10^6$)	No SF	SF	No SF	SF	Regular
1	48.8	18.95	23.77	59.46	70.45	36.92
.5	64.1	26.91	31.47	66.37	76.71	31.04
Bin .5	53.1	23.79	26.86	60.01	69.05	26.54
.2	85.7	37.16	42.47	76.36	87.34	27.81
Bin .2	82.4	36.56	41.19	75.11	85.74	27.35

TABLE I

Complete cost field expansion for 150x150x150 grid.

Density	% Incorrect	Nodes Expanded		Total Time (s)	
		Initial	Replan	Initial	Replan
0	50	894000	62400	20.04	1.56
0	20	894000	25200	20.01	0.83
.5	50	836000	173000	15.29	4.51
.5	20	836000	27800	15.27	0.64

TABLE II

Robot navigation simulation for 150x150x150 grid.

Each entry in the table represents the average of twenty-five different maps. The 'rhs_{s_f}' column shows the number of times that an rhs update (Equation (6) when using scale factors, Equation (3) otherwise) was performed. The 'rhs_{s_f} Time' column shows the time taken for these computations. Most of the rest of the time is spent on insertions and deletions from the priority queue, and determining which faces to test in order to update the neighboring nodes of the top element on the queue. The final three columns show the total time taken for all computations. Testing was done with and without using scale factors (the 'SF' and 'No SF' entries, respectively), and the total time has also been reported for Regular D* Lite. When using scale factors, they were set to one for all directions. Therefore identical cost field expansions were generated in both cases. Overall, the algorithm is only slowed by about fifteen percent when scale factors are enabled, and requires roughly two to three times as much computation as regular D* Lite.

It is interesting to note that the total computation time required by Field D* increases when more of the map is free space. This increase is a side effect of an optimization in our implementation regarding how the rhs updates are performed. It is possible to put a minimum bound on the result of the $rhs_{s_f}(s)$ update calculation for a particular node s and face f. Moving the shortest possible distance through the appropriate voxel (one unit), to the lowest cost node on face f gives a lower bound on the value of $rhs_{s_f}(s)$. Therefore, if $rhs(s)$ is already lower than this bound, we do not need to do the computation. As the voxel costs in the map become more and more uniform, this minimum bound test eliminates less and less of the faces. As a result, more time is spent on planning through maps consisting mainly of free space.

Table II summarizes results for the simulation of a robot navigation task. In this test, a simulated robot was placed at the center of the environment and was equipped with an omnidirectional sensor with a range of seven units. The goal was placed at the center of the far right side of the environment. In the first two sets of experiments, the map was initially assumed to be completely free. In the second two sets of experiments, half of the voxels were initially assumed to be free and half were assumed to hold random costs. To begin each simulation, a path from the robot to the goal was planned using the robot's initial map. However, this initial map was erroneous: the actual environment had some fraction of the voxels holding different random costs from those stored in the robot's map. The robot then moved toward the goal and its map was updated when the true costs of the voxels in the environment were detected. At each update, the robot's path was replanned based on its new map information.

Results for both initial planning and all subsequent replanning are shown. Again, the results are an average of twenty-five different trials. The second column indicates the fraction of voxel costs that were incorrect in the initial map. Even when the robot begins with grossly inaccurate maps where up to fifty percent of its information is incorrect, it is still able to efficiently repair its paths during its traverse. In each of these tests the robot moved a minimum of 75 steps, which means the average replanning time for each path repair phase was only 0.06 seconds for the most complex set of experiments. This is fast enough to provide real-time performance for underwater or aerial vehicles operating at moderate speeds.

VII. Conclusion

We have presented a version of the Field D* algorithm capable of generating three-dimensional paths that are not restricted to a small set of headings. While a two-dimensional version of this algorithm exists and is currently employed by several robotic ground vehicles, until now no extension had been made to provide direct, interpolation-based paths for aerial or underwater vehicles. In this paper we have presented a three-dimensional version of the algorithm that uses an efficient approximation technique to provide real-time performance. The resulting approach produces less costly, more direct paths than classic 3D grid-based planners. We have also provided the capability to encode directional-dependent costs into the planning process.

VIII. Acknowledgements

This work was sponsored by the U.S. Army Research Laboratory, under contract "Robotics Collaborative Technology Alliance" (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements of the U.S. Government. Dave Ferguson is supported in part by a National Science Foundation Graduate Research Fellowship.

References

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

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] ——, "Field D*: An Interpolation-based Path Planner and Replanner," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2005.

[8] R. Philippsen and R. Siegwart, "An Interpolated Dynamic Navigation Function," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

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

[10] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proceedings of the 28th Annual IEEE Symposium on the Foundations of Computer Science*, 1987, pp. 49–60.

[11] C. Papadimitriou, "An algorithm for shortest-path motion in three dimensions," *Information Processing Letters*, vol. 20, pp. 259–263, 1985.

[12] J. Choi, J. Sellen, and C. Yap, "Precision-sensitive Euclidean shortest path in 3-space," in *Proceedings of the Annual ACM Symposium on Computational Geometry*, 1995, pp. 350–359.

[13] ——, "Approximate euclidean shortest paths in 3-space," *International Journal of Computational Geometry Applications*, vol. 7, no. 4, pp. 271–295, August 1997.

[14] S. Har-Peled, "Constructing approximate shortest path maps in three dimensions," in *Proceedings of the 14th Annual ACM Symposium on Computational Geometry*, 1998.

[15] J. Mitchell, *Handbook of Computational Geometry*. Elsevier Science, 2000, ch. Geometric Shortest Paths and Network Optimization, pp. 633–701.

[16] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida, "3-D path planning in a dynamic environment using an octree and an artificial potential field," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1995.

[17] P. Payeur, C. Gosselin, and D. Laurendeau, "Application of 3-D probabilistic occupancy models for potential field based collision free path planning," in *Proceedings of the IEEE Image and Multidimensional Digital Signal Processing Workshop*, 1998.