Field D*: An Interpolation-based Path Planner and Replanner

Dave Ferguson and Anthony Stentz

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania
{dif,tony}@cmu.edu

Summary. We present an interpolation-based planning and replanning algorithm for generating direct, low-cost paths through nonuniform cost grids. Most grid-based path planners use discrete state transitions that artificially constrain an agent's motion to a small set of possible headings (e.g. 0, $\frac{\pi}{4}$, $\frac{\pi}{2}$, etc). As a result, even 'optimal' grid-based planners produce unnatural, suboptimal paths. Our approach uses linear interpolation during planning to calculate accurate path cost estimates for arbitrary positions within each grid cell and to produce paths with a range of continuous headings. Consequently, it is particularly well suited to planning low-cost trajectories for mobile robots. In this paper, we introduce the algorithm and present a number of example applications and results.

1 Introduction

In mobile robot navigation, we are often provided with a grid-based representation of our environment and tasked with planning a path from some initial robot location to a desired goal location. Depending on the environment, the representation may be binary (each grid cell contains either an obstacle or free space) or may associate with each cell a cost reflecting the difficulty of traversing the respective area of the environment.

In robotics, it is common to improve efficiency by approximating this grid with a graph, where nodes are placed at the center of each grid cell and edges connect nodes within adjacent grid cells. Many algorithms exist for planning paths over such graphs. Dijkstra's algorithm computes paths from every node to a specified goal node [3]. A* uses a heuristic to focus the search from a particular start location towards the goal and thus produces a path from a single location to the goal very efficiently [5, 18]. D*, Incremental A*, and D* Lite are extensions of A* that incrementally repair solution paths when changes occur in the underlying graph [26, 7, 8, 9]. These incremental algorithms have been used extensively in robotics for mobile robot navigation in unknown or dynamic environments.

Pioneers | Automated E-Gator | Automated ATV | GDRS XUV

Fig. 1. Some robots that currently use Field D* for global path planning. These range from indoor planar robots (the Pioneers) to outdoor robots able to operate in harsh terrain (the XUV).

However, almost all of these approaches are limited by the small, discrete set of possible transitions they allow from each node in the graph. For instance, given a graph extracted from a uniform resolution 2D grid, a path planned in the manner described above restricts the agent's heading to increments of $\frac{\pi}{4}$. This results in paths that are suboptimal in length and difficult to traverse in practice. Further, even when these paths are used in conjunction with a local arc-based planner (e.g. as in the RANGER system [6, 25]), they can still cause the vehicle to execute expensive trajectories involving unnecessary turning.

In this paper we present Field D*, an interpolation-based planning and replanning algorithm that alleviates this problem. This algorithm extends D* and D* Lite to use linear interpolation to efficiently produce low-cost paths that eliminate unnecessary turning. The paths are optimal given a linear interpolation assumption and very effective in practice. This algorithm is currently being used by a wide range of fielded robotic systems (see Figure 1).

We begin by discussing the limitations of paths produced using classical grid-based planners and mention recent approaches that attempt to overcome some of these limitations. We then present an interpolation-based method for obtaining more accurate path cost approximations and show how this method can be incorporated into existing planning and replanning algorithms. We provide a number of example illustrations and applications of our approach and conclude with discussion and extensions.

2 Limitations of Classical 2D Path Planning

Consider a robotic ground vehicle navigating an outdoor environment. We can represent this environment as a uniform resolution 2D traversability grid, in which cells are given a cost per unit of traverse (traversal cost) reflecting the difficulty of navigating the respective area of the environment. If this traversability grid encodes the configuration space costs (i.e. the traversal costs have been expanded to reflect the physical dimensions of the vehicle), then planning a path for the robot translates to generating a trajectory through this grid for a single point. A common approach used in robotics for performing this planning is to combine an approximate *global* planner with an accurate

local planner [6, 25, 1, 23]. The global planner computes paths through the grid that ignore the kinematic and dynamic constraints of the vehicle. Then, the local planner takes into account the constraints of the vehicle and generates a set of feasible local trajectories that can be taken from its current position. To decide which of these trajectories to execute, the robot evaluates both the cost of each local trajectory and the cost of a global path from the end of each trajectory to the robot's desired goal location.

To formalize the global planning task, we need to define more precisely some concepts already introduced. First, each cell in the grid has assigned to it some real-valued traversal cost that is greater than zero. The cost of a line segment between two points within a cell is the Euclidean distance between the points multiplied by the traversal cost of the cell. The cost of any path within the grid is the sum of the costs of its line segments through each cell. Then, the global planning task (involving a uniform resolution grid) can be specified as follows.

The Global Planning Task: *Given a region in the plane partitioned into a uniform grid of square cells $\mathcal{T}$, an assignment of traversal costs $c : \mathcal{T} \rightarrow (0, +\infty]$ to each cell, and two points s_{start} and s_{goal} within the grid, find the path within the grid from s_{start} to s_{goal} with minimum cost.*

This task can be seen as a specific instance of the Weighted Region Problem [17], where the regions are uniform square tiles. A number of algorithms exist to solve this problem in the computational geometry literature (see [16] for a good survey). In particular, [17] and [21] present approaches based on Snell's law of refraction that compute optimal paths by simulating a series of light rays that propagate out from the start position and refract according to the different traversal costs of the regions encountered. These approaches are efficient for planning through environments containing a small number of homogenous-cost regions, but are computationally expensive when the number of such regions is very large, as in the case of a uniform grid with varying cell costs.

Because of the computational expense associated with planning optimal paths through grids, researchers in robotics have focussed on basic approximation algorithms that are extremely fast. The most popular such approach is to approximate the traversability grid as a discrete graph, then generate paths over the graph. A common way to do this is to assign a node to each cell center, with edges connecting the node to each adjacent cell center (node). The cost of each edge is a combination of the traversal costs of the two cells it transitions through and the length of the edge. Figure 2(a) shows this node and edge extraction process for one cell in a uniform resolution 2D grid.

We can then plan over this graph to generate paths from the robot's initial location to a desired goal location. As mentioned previously, a number of efficient algorithms exist for performing this planning, such as A* for initial planning and D* and its variants for replanning [5, 18, 26, 8]. Unfortunately, paths produced using this graph are restricted to headings of $\frac{\pi}{4}$ increments.

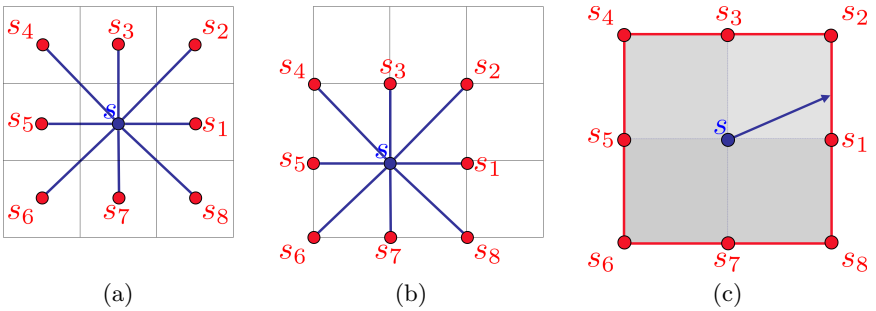

Fig. 2. (a) A standard 2D grid used for global path planning in which nodes reside at the centers of the grid cells. The arcs emanating from the center node represent all the possible actions that can be taken from this node. (b) A modified representation used by Field D*, in which nodes reside at the corners of grid cells. (c) The optimal path from node s must intersect one of the edges $\{\overrightarrow{s_1s_2}, \overrightarrow{s_2s_3}, \overrightarrow{s_3s_4}, \overrightarrow{s_4s_5}, \overrightarrow{s_5s_6}, \overrightarrow{s_6s_7}, \overrightarrow{s_7s_8}, \overrightarrow{s_8s_1}\}$.

This means that the final solution path may be suboptimal in path cost, involve unnecessary turning, or both.

For instance, consider a robot facing its goal position in a completely obstacle-free environment (see Figure 3). Obviously, the optimal path is a straight line between the robot and the goal. However, if the robot's initial heading is not a multiple of $\frac{\pi}{4}$, traditional grid-based planners would return a path that has the robot first turn to attain the nearest grid heading, move some distance along this heading, and then turn $\frac{\pi}{4}$ in the opposite direction of its initial turn and continue to the goal. Not only does this path have clearly suboptimal length, it contains possibly expensive or difficult turns that are purely artifacts of the limited representation. Such global paths, when coupled with the results of a local planner, cause the robot to behave suboptimally. Further, this limitation of traditional grid-based planners is not alleviated by increasing the resolution of the grid.

Sometimes it is possible to reduce the severity of this problem by post-processing the path. Usually, given a robot location s, one finds the furthest point p along the solution path for which a straight line path from s to p is collision-free, then replaces the original path to p with this straight line path. However, this does not always work, as illustrated by Figure 4. Indeed, for nonuniform cost environments such post-processing can often *increase* the cost of the path.

A more comprehensive post-processing approach is to take the result of the global planner and use it to seed a higher dimensional planner that incorporates the kinematic or dynamic constraints of the robot. Stachniss and Burgard [24] present an approach that takes the solution generated by the global planner and uses it to extract a local waypoint to use as the goal for a 5D trajectory planner. The search space of the 5D planner is limited to a small

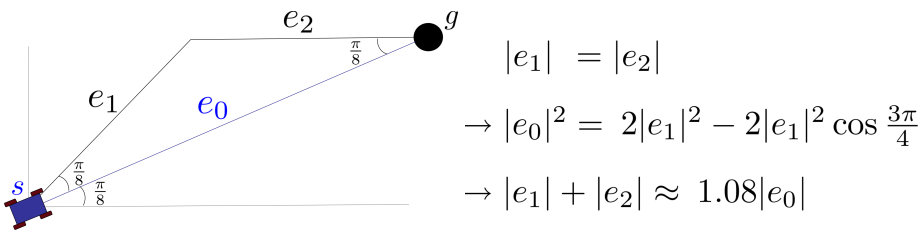

Fig. 3. A uniform resolution 2D grid-based path (e_1 plus e_2) between two grid nodes can be up to 8% longer than an optimal straight-line path (e_0). Here, the desired straight-line heading is $\frac{\pi}{8}$ and lies perfectly between the two nearest grid-based headings of 0 and $\frac{\pi}{4}$. This result is independent of the resolution of the grid.

area surrounding the global solution path. Likhachev et al. [15, 14] present an approach that uses the cost-to-goal value function of the global planner to focus an anytime global 4D trajectory planner. Their approach improves the quality of the global trajectory while deliberation time allows. However, these higher dimensional approaches can be much more computationally expensive than standard grid-based planners and are still influenced by the results of the initial grid-based solution.

Recently, robotics researchers have looked at more sophisticated methods of obtaining better paths through grids without sacrificing too much of the efficiency of the classic grid-based approach described above. Konolige [10] presents an interpolated planner that first uses classic grid-based planning to construct a cost-to-goal value function over the grid and then interpolates this result to produce a shorter path from the initial position to the goal. This method results in shorter, less-costly paths for agents to traverse but does not incorporate the reduced path cost into the planning process. Consequently, the resulting path is not necessarily as good as the path the algorithm would produce if interpolated costs were calculated during planning. Further, if we are computing paths from several locations (which is common when combining the global planner with a local planner) then this post-processing interpolation step can be expensive. Also, this approach provides no replanning functionality to update the solution when new information concerning the environment is received.

Philippsen and Siegwart [20] present an algorithm based on Fast Marching Methods [22] that computes a value function over the grid by growing a surface out from the goal to every region in the environment. The surface expands according to surface flow equations, and the value of each grid point is computed by combining the values of two neighboring grid points. This approach incorporates the interpolation step into the planning process, producing low-cost, interpolated paths. This technique has been shown to generate nice paths in indoor environments [19, 20]. However, the search is not focussed towards the robot location (such as in A*) and assumes that the transition cost from a particular grid node to each of its neighbors is constant. Consequently, it is

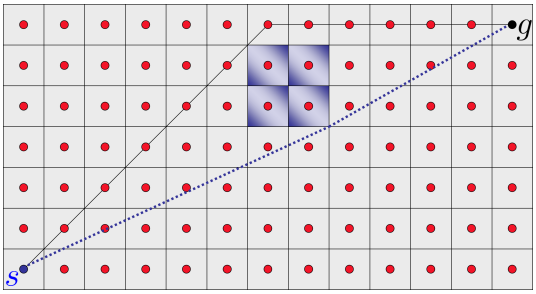

Fig. 4. 2D grid-based paths cannot always be shortened in a post-processing phase. Here, the grid-based path from s to g (top, in black) cannot be shortened because there are four obstacle cells (shaded). The optimal path is shown dashed.

not as applicable to navigation in outdoor environments, which are often best represented by large grids with widely-varying cell traversal costs.

The idea of using interpolation to produce better value functions for discrete samples over a continuous state space is not new. This approach has been used in dynamic programming for some time to compute the value of successors that are not in the set of samples [11, 12, 13]. However, as LaValle points out [13], this becomes difficult when the action space is also continuous, as solving for the value of a state now requires minimizing over an infinite set of successor states.

The approach we present here is an extension of the widely-used D* family of algorithms that uses linear interpolation to produce near-optimal paths that eliminate unnecessary turning. It relies upon an efficient, closed-form solution to the above minimization problem for 2D grids, which we introduce in the next section. This method produces much straighter, less-costly paths than classical grid-based planners without sacrificing real-time performance. As with D* and D* Lite, our approach focusses its search towards the most relevant areas of the state space during both initial planning and replanning. Further, it takes into account local variations in cell traversal costs and produces paths that are optimal given a linear interpolation assumption. As the resolution of the grid increases, the solutions returned by the algorithm improve, approaching true optimal paths.

3 Improving Cost Estimation through Interpolation

The key to our algorithm is a novel method for computing the path cost of each grid node s given the path costs of its neighboring nodes. By the path cost of a node we mean the cost of the cheapest path from the node to the goal. In classical grid-based planning this value is computed as

$$g(s) = \min_{s' \in nbrs(s)} (c(s, s') + g(s')), \tag{1}$$

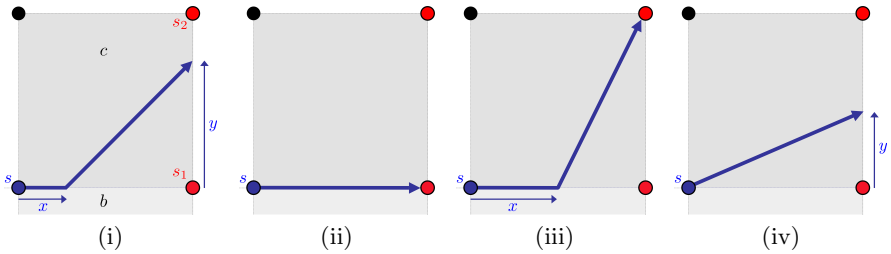

Fig. 5. Computing the path cost of node s using the path cost of two of its neighbors, s_1 and s_2, and the traversal costs c of the center cell and b of the bottom cell. Illustrations (ii) through (iv) show the possible optimal paths from s to edge $\overrightarrow{s_1 s_2}$.

where $nbrs(s)$ is the set of all neighboring nodes of s (see Figure 2), $c(s, s')$ is the cost of traversing the edge between s and s', and $g(s')$ is the path cost of node s'.

This calculation assumes that the only transitions possible from node s are straight-line trajectories to one of its neighboring nodes. This assumption results in the limitations of grid-based plans discussed earlier. However, consider relaxing this assumption and allowing a straight-line trajectory from node s to any point on the boundary of its grid cell. If we knew the value of every point s_b along this boundary, then we could compute the optimal value of node s simply by minimizing $c(s, s_b) + g(s_b)$, where $c(s, s_b)$ is computed as the distance between s and s_b multiplied by the traversal cost of the cell in which s resides. Unfortunately, there are an infinite number of such points s_b and so computing $g(s_b)$ for each of them is not possible.

It is possible, however, to provide an approximation to $g(s_b)$ for each boundary point s_b by using linear interpolation. To do this, we first modify the graph extraction process discussed earlier. Instead of assigning nodes to the centers of grid cells, we assign nodes to the *corners* of each grid cell, with edges connecting nodes that reside at corners of the same grid cell (see Figure 2(b)).

Given this modification, the traversal costs of any two equal-length segments of an edge will be the same. This differs from the original graph extraction process in which the first half of an edge was in one cell and the second half was in another cell, with the two cells possibly having different traversal costs. In the modified approach the cost of an edge that resides on the boundary of two grid cells is defined as the minimum of the traversal costs of each of the two cells.

We then treat the nodes in our graph as sample points of a continuous cost field. The optimal path from a node s must pass through an edge connecting two consecutive neighbors of s, for example $\overrightarrow{s_1 s_2}$ (see Figure 2(c)). The path cost of s is thus set to the minimum cost of a path through any of these edges, which are considered one at a time. To compute the path cost of node s using

ComputeCost(s, s_a, s_b)
01. if (s_a is a diagonal neighbor of s)
02. $\quad s_1 = s_b$; $s_2 = s_a$;
03. else
04. $\quad s_1 = s_a$; $s_2 = s_b$;
05. c is traversal cost of cell with corners s, s_1, s_2;
06. b is traversal cost of cell with corners s, s_1 but not s_2;
07. if ($\min(c, b) = \infty$)
08. $\quad v_s = \infty$;
09. else if ($g(s_1) \leq g(s_2)$)
10. $\quad v_s = \min(c, b) + g(s_1)$;
11. else
12. $\quad f = g(s_1) - g(s_2)$;
13. $\quad$ if ($f \leq b$)
14. $\quad\quad$ if ($c \leq f$)
15. $\quad\quad\quad v_s = c\sqrt{2} + g(s_2)$;
16. $\quad\quad$ else
17. $\quad\quad\quad y = \min(\frac{f}{\sqrt{c^2 - f^2}}, 1)$;
18. $\quad\quad\quad v_s = c\sqrt{1 + y^2} + f(1 - y) + g(s_2)$;
19. $\quad$ else
20. $\quad\quad$ if ($c \leq b$)
21. $\quad\quad\quad v_s = c\sqrt{2} + g(s_2)$;
22. $\quad\quad$ else
23. $\quad\quad\quad x = 1 - \min(\frac{b}{\sqrt{c^2 - b^2}}, 1)$;
24. $\quad\quad\quad v_s = c\sqrt{1 + (1 - x)^2} + bx + g(s_2)$;
25. return v_s;

Fig. 6. The Interpolation-based Path Cost Calculation

edge $\overrightarrow{s_1 s_2}$, we use the path costs of nodes s_1 and s_2 and the traversal costs c of the center cell and b of the bottom cell (see Figure 5).

To compute this cost efficiently, we assume the path cost of any point s_y residing on the edge between s_1 and s_2 is a linear combination of $g(s_1)$ and $g(s_2)$:

$$g(s_y) = yg(s_2) + (1 - y)g(s_1), \tag{2}$$

where y is the distance from s_1 to s_y (assuming unit cells). This assumption is not perfect: the path cost of s_y may not be a *linear* combination of $g(s_1)$ and $g(s_2)$, nor even a function of these path costs. However, this linear approximation works well in practice, and allows us to construct a closed form solution for the path cost of node s.

Given this approximation, the path cost of s given s_1, s_2, and cell costs c and b can be computed as

$$\min_{x,y}[bx + c\sqrt{(1 - x)^2 + y^2} + yg(s_2) + (1 - y)g(s_1)], \tag{3}$$

where $x \in [0, 1]$ is the distance traveled along the bottom edge from s before cutting across the center cell to reach the right edge a distance of $y \in [0, 1]$ from s_1 (see Figure 5(i)). Note that if both x and y are zero in the above equation the path taken is along the bottom edge but its cost is computed from the traversal cost of the center cell.

Let (x^*, y^*) be a pair of values for x and y that solve the above minimization. Because of our use of linear interpolation, at least one of these values

will be either zero or one. We formally prove this in an extended technical report version of this paper [4]. Intuitively, if it is less expensive to partially cut through the center cell than to traverse around the boundary, then it is least expensive to completely cut through the cell. Thus, if there is any component to the cheapest solution path from s that cuts through the center cell, it will be as large as possible, forcing $x^* = 0$ or $y^* = 1$. If there is no component of the path that cuts through the center cell, then $y^* = 0$.

Thus, the path will either travel along the entire bottom edge to s_1 (Figure 5(ii)), or will travel a distance x along the bottom edge then take a straight-line path directly to s_2 (Figure 5(iii)), or will take a straight-line path from s to some point s_y on the right edge (Figure 5(iv)). Which of these paths is cheapest depends on the relative sizes of c, b, and the difference f in path cost between s_1 and s_2: $f = g(s_1) - g(s_2)$. Specifically, if $f < 0$ then the optimal path from s travels straight to s_1 and will have a cost of $(\min(c, b) + g(s_1))$ (Figure 5(ii)). If $f = b$ then the cost of a path using some portion of the bottom edge (Figure 5(iii)) will be equivalent to the cost of a path using none of the bottom edge (Figure 5(iv)). We can solve for the value of y that minimizes the cost of the latter path as follows.

First, let $k = f = b$. The cost of a path from s through edge $\overrightarrow{s_1 s_2}$ is

$$c\sqrt{1 + y^2} + k(1 - y) + g(s_2). \tag{4}$$

Taking the derivative of this cost with respect to y and setting it equal to zero yields

$$y^* = \sqrt{\frac{k^2}{c^2 - k^2}}. \tag{5}$$

Whether the bottom edge or the right edge is used, we end up with the same calculations and path cost computations. So all that matters is which edge is cheaper. If $f < b$ then we use the right edge and compute the path cost as above (with $k = f$), and if $b < f$ we use the bottom edge and substitute $k = b$ and $y^* = 1 - x^*$ into the above equation. The resulting algorithm for computing the minimum-cost path from s through an edge between *any* two consecutive neighbors s_a and s_b is provided in Figure 6. Given the minimum-cost paths from s through each of its 8 neighboring edges, we can compute the path cost for s to be the cost of the cheapest of these paths. The corresponding path is optimal given our linear interpolation assumption.

4 Field D*

Once equipped with this interpolation-based path cost calculation for a given node in our graph, we can plug it into any of a number of current planning and replanning algorithms to produce low-cost paths. Figure 7 presents our simplest formulation of *Field D**, an incremental replanning algorithm that

```
key(s)
01. return [min(g(s), rhs(s)) + h(s_start, s); min(g(s), rhs(s))];

UpdateState(s)
02. if s was not visited before, g(s) = ∞;
03. if (s ≠ s_goal)
04.     rhs(s) = min_{(s',s'')∈connbrs(s)} ComputeCost(s, s', s'');
05. if (s ∈ OPEN) remove s from OPEN;
06. if (g(s) ≠ rhs(s)) insert s into OPEN with key(s);

ComputeShortestPath()
07. while (min_{s∈OPEN}(key(s)) ≺̇ key(s_start) OR rhs(s_start) ≠ g(s_start))
08.     remove state s with the minimum key from OPEN;
09.     if (g(s) > rhs(s))
10.         g(s) = rhs(s);
11.         for all s' ∈ nbrs(s) UpdateState(s');
12.     else
13.         g(s) = ∞;
14.         for all s' ∈ nbrs(s) ∪ {s} UpdateState(s');

Main()
15. g(s_start) = rhs(s_start) = ∞; g(s_goal) = ∞;
16. rhs(s_goal) = 0; OPEN = ∅;
17. insert s_goal into OPEN with key(s_goal);
18. forever
19.     ComputeShortestPath();
20.     Wait for changes in cell traversal costs;
21.     for all cells x with new traversal costs
22.         for each state s on a corner of x
23.             UpdateState(s);
```

Fig. 7. The Field D* Algorithm (basic D* Lite version).

incorporates these interpolated path costs. This version of Field D* is based on D* Lite[1].

In this figure, $connbrs(s)$ contains the set of consecutive neighbor pairs of node s: $connbrs(s) = \{(s_1,s_2), (s_2,s_3), (s_3,s_4), (s_4,s_5), (s_5,s_6), (s_6,s_7), (s_7,s_8), (s_8,s_1)\}$, where s_i is positioned as shown in Figure 2(c). Apart from this construction, notation follows the D* Lite algorithm: $g(s)$ is the current path cost of node s (its g-value), $rhs(s)$ is the one-step lookahead path cost for s (its rhs-value), $OPEN$ is a priority queue containing inconsistent nodes (i.e., nodes s for which $g(s) \neq rhs(s)$) in increasing order of key values (line 1), s_{start} is the initial agent node, and s_{goal} is the goal node. $h(s_{start}, s)$ is a heuristic estimate of the cost of a path from s_{start} to s. Because the key value of each node contains two quantities a lexicographic ordering is used: $\text{key}(s) \dot{<} \text{key}(s')$ iff the first element of key(s) is less than the first element of key(s') or the first element of key(s) equals the first element of key(s') and

[1] Differences between Field D* and D* Lite appear on lines 4 and 20 through 23. As opposed to the original, graph-based version of D* Lite, lines 20 - 22 tailor Field D* to grids. Also, because paths intersect edges and not just nodes, the heuristic value $h(s_{start}, s)$ must be small enough that when added to the cost of any edge incident on s it is still not greater than a minimum cost path from s_{start} to s.

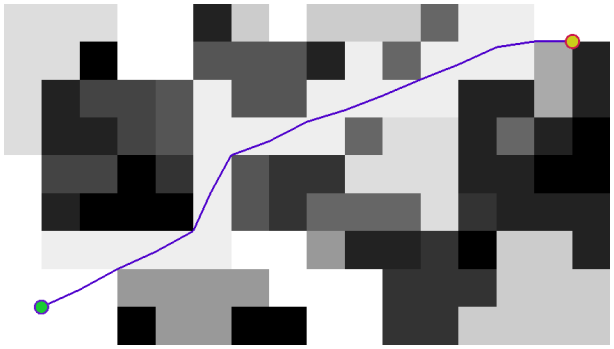

Fig. 8. A close-up of a path planned using Field D* showing individual grid cells. Notice that the path is not limited to entering and exiting cells at corner points.

the second element of key(s) is less than the second element of key(s'). For more details on the D* Lite algorithm and this terminology, see [8, 7]. Also, the termination and correctness of the Field D* algorithm follow directly from D* Lite and the analysis of the cost calculation provided in Section 3.

This is an unoptimized version of Field D*. In our extended technical report [4] we discuss a number of optimizations that significantly improve the overall efficiency of planning and replanning with this algorithm.

Once the cost of a path from the initial state to the goal has been calculated, the path is extracted by starting at the initial position and iteratively computing the cell boundary point to move to next. Because of our interpolation technique, it is possible to compute the path cost of *any* point inside a grid cell, not just the corners, which is useful for both extracting the path and getting back on track if execution is not perfect (which is usually the case for real robots).

Figures 8 and 9 illustrate paths produced by Field D* through three nonuniform cost environments. In each of these figures, darker areas represent regions that are more costly to traverse. Notice that, unlike paths produced using classical grid-based planners, the paths produced using Field D* are not restricted to a small set of headings. As a result, Field D* provides lower-cost paths through both uniform and nonuniform cost environments.

5 Results

The true test of an algorithm is its practical effectiveness. We have found Field D* to be extremely useful for a wide range of robotic systems navigating through terrain of varying degrees of difficulty (see Figure 1).

To provide a quantitative comparison of the performance of Field D* relative to D* Lite, we ran a number of replanning simulations in which we measured both the relative solution path costs and runtimes of the opti-

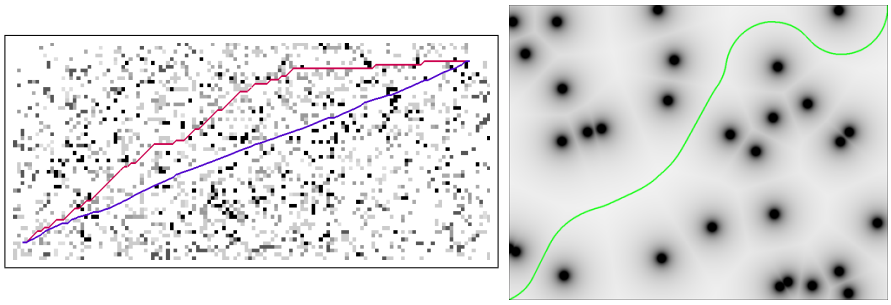

Fig. 9. (left) Paths produced by D* Lite (top) and Field D* (bottom) in a 150×60 nonuniform cost environment. **(right)** Field D* planning through a potential field of obstacles.

mized versions of the two approaches. We generated 100 different 1000 $\times$ 1000 nonuniform cost grid environments in which each grid cell was assigned an integer traversal cost between 1 (free space) and 16 (obstacle). With probability 0.5 this cost was set to 1, otherwise it was randomly selected. For each environment, the initial task was to plan a path from the lower left corner to a randomly selected goal on the right edge. After this initial path was planned, we randomly altered the traversal costs of cells close to the agent (10% of the cells in the environment were changed) and had each approach repair its solution path. This represents a significant change in the information held by the agent and results in a large amount of replanning.

During initial planning, Field D* generated solutions that were on average 96% as costly as those generated by D* Lite, and took 1.7 times as long to generate these solutions. During replanning, the results were similar: Field D* provided solutions on average 96% as costly and took 1.8 times as long. The average initial planning runtime for Field D* on a 1.5 GHz Powerbook G4 was 1.5s, and the average replanning runtime was 0.07s. In practice, the algorithm is able to provide real-time performance for fielded systems.

6 Discussion

Although the results presented above show that Field D* generally produces less costly paths than regular grid-based planning, this is not guaranteed. It is possible to construct pathological scenarios where the linear interpolation assumption is grossly incorrect (for instance, if there is an obstacle in the cell to the right of the center cell in Figure 5(i) and the optimal path for node s_2 travels above the obstacle and the optimal path for node s_1 travels below the obstacle). In such cases, the interpolated path cost of a point on an edge between two nodes may be either too low or too high. This in turn can affect the quality of the extracted solution path. However, such occurrences are very rare, and in none of our random test cases (nor any cases we have ever

encountered in practice) was the path returned by Field D* more expensive than the grid-based path returned by D* Lite. In general, even in carefully-constructed pathological scenarios the path generated by Field D* is very close in cost to the optimal solution path.

Moreover, it is the ability of Field D* to plan paths with a continuous range of headings, rather than simply its lower-cost solutions, that is its true advantage over regular grid-based planners. In both uniform and nonuniform cost environments, Field D* provides direct, sensible paths for our agents to traverse.

7 Conclusion

In this paper we presented Field D*, an extension of classical grid-based planners that uses linear interpolation to efficiently produce less costly, more natural paths through grids. We have found Field D* to be extremely useful for mobile robot path planning in both uniform and nonuniform cost environments.

We and others are currently extending the Field D* algorithm in a number of ways. Firstly, a 3D version of the Field D* algorithm has been developed for vehicles operating in the air or underwater [2]. We are also developing a version that interpolates over headings, not just path costs, to produce smoother paths when turning is expensive. Finally, we are also working on a version of the algorithm able to plan over nonuniform grids, for vehicles navigating through very large environments.

8 Acknowledgments

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 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. O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.
2. J. Carsten. 3D Field D*. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, 2005.
3. E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

4. D. Ferguson and A. Stentz. The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-uniform Cost Environments. Technical Report CMU-RI-TR-05-19, Carnegie Mellon School of Computer Science, 2005.
5. P. Hart, N. Nilsson, and B. Rafael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.
6. A. Kelly. *An Intelligent Predictive Control Approach to the High Speed Cross Country Autonomous Navigation Problem*. PhD thesis, Carnegie Mellon University, 1995.
7. S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2002.
8. 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.
9. S. Koenig and M. Likhachev. Incremental A*. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.
10. K. Konolige. A gradient method for realtime robot control. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2000.
11. R. Larson. A survey of dynamic programming computational procedures. *IEEE Transactions on Automatic Control*, pages 767–774, 1967.
12. R. Larson and J. Casti. *Principles of Dynamic Programming, Part 2*. Marcel Dekker, New York, 1982.
13. S. LaValle. *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006. To be published in 2006.
14. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.
15. M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*. MIT Press, 2003.
16. J. Mitchell. *Handbook of Computational Geometry*, chapter Geometric Shortest Paths and Network Optimization, pages 633–701. Elsevier Science, 2000.
17. J. Mitchell and C. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38:18–73, 1991.
18. N. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.
19. R. Philippsen. *Motion Planning and Obstacle Avoidance for Mobile Robots in Highly Cluttered Dynamic Environments*. PhD thesis, EPFL, Lausanne, Switzerland, 2004.
20. R. Philippsen and R. Siegwart. An Interpolated Dynamic Navigation Function. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.
21. N. Rowe and R. Richbourg. An efficient Snell's-law method for optimal-path planning across two-dimensional irregular homogeneous-cost regions. *International Journal of Robotics Research*, 9(6):48–66, 1990.
22. J. Sethian. A fast marching level set method for monotonically advancing fronts. *Applied Mathematics, Proceedings of the National Academy of Science*, 93:1591–1595, 1996.

23. S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr. Recent progress in local and global traversability for planetary rovers. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.
24. C. Stachniss and W. Burgard. An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002.
25. A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2):127–145, 1995.
26. Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.