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

Dave Ferguson and Anthony Stentz

Carnegie Mellon University
{dif,tony}@cmu.edu

Abstract. We present an interpolation-based planning and replanning algorithm for generating smooth paths through non-uniform 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 the 'optimal' grid 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 continuous range of headings. Consequently, it is particularly well suited to planning smooth, least-cost trajectories for mobile robots. In this paper, we present a number of applications and results, a comparison to related algorithms, and several implementations on real robotic systems.

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.

Many algorithms exist for planning paths on such grids. Dijkstra's algorithm computes paths from every grid location to a goal location [1]. 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 [3,10]. D*, Incremental A*, and D* Lite are extensions of A* that incrementally repair solution paths when changes occur in the underlying graph [13,5,4]. These incremental algorithms have been used extensively in robotics for mobile robot navigation in unknown or dynamic environments.

However, these approaches are limited by the small, discrete set of possible transitions they allow between grid cells. For instance, when planning on a 2D grid, it is common to plan from the center of each grid cell and only allow transitions to the centers of adjacent grid cells. This restricts the agent's heading to increments of $\frac{\pi}{4}$, which results in paths that are suboptimal in length and difficult to traverse in practice.

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 globally-smooth paths. The paths are optimal with respect to a linear interpolation assumption and very effective in

practice. This algorithm is currently in use in a number of fielded robotic systems in both indoor and outdoor environments.

2 Limitations of Classical 2D Path Planning

Consider a ground vehicle navigating an outdoor environment. We can represent this environment as a 2D traversability grid in which cells are given a cost of traversal reflecting the difficulty of navigating the respective area of the environment. With such a grid we can extract a graph for path planning quite easily: 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 connects and the length of the edge. Fig. 1(a) shows the node and edge extraction process for one cell in a uniform 2D grid.

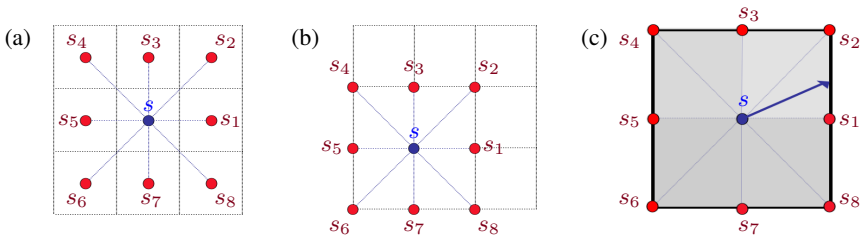


Fig. 1. (a) A standard 2D grid used for path planning, in which nodes reside at the center of each grid cell. The arcs emanating from the center node represent all the possible actions that can be taken from this node. (b) A modified representation used by Field D*, in which nodes reside at the corners of grid cells. (c) The optimal path from node s must intersect one of the edges $\{\overrightarrow{s_1s_2}, \overrightarrow{s_2s_3}, \overrightarrow{s_3s_4}, \overrightarrow{s_4s_5}, \overrightarrow{s_5s_6}, \overrightarrow{s_6s_7}, \overrightarrow{s_7s_8}, \overrightarrow{s_8s_1}\}$.

We can then plan over this graph to generate paths from the robot's initial location to a desired goal location, using any of the algorithms already mentioned. Unfortunately, paths produced using this graph are restricted to headings of $\frac{\pi}{4}$ increments. 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 open environment (see Fig. 2). 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 $\frac{\pi}{4}$, the path returned by traditional planners would require that 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.

Sometimes it is possible to alleviate 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 approach does not al-

ways work, as illustrated by Fig. 3. Indeed, for non-uniform cost environments such smoothing can often *increase* the cost of the path.

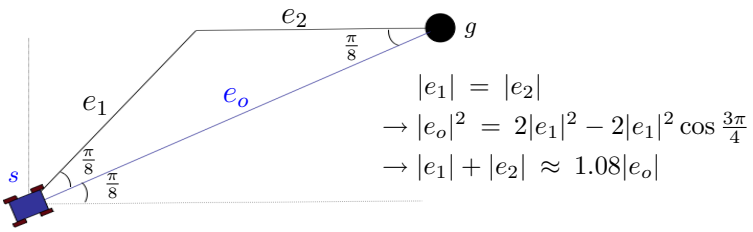


Fig. 2. A uniform 2D grid-based path (e_1 plus e_2) between two grid nodes can be up to 8% longer than an optimal straight-line path (e_0). Here, the desired straight-line heading is $\frac{\pi}{8}$ and lies perfectly between the two nearest grid-based headings of 0 and $\frac{\pi}{4}$. This result is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

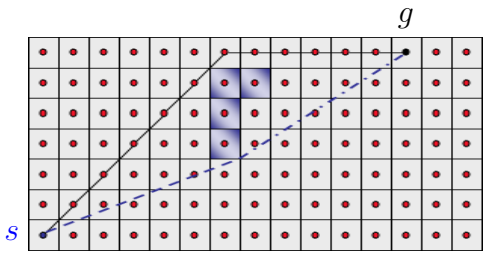


Fig. 3. 2D grid-based paths cannot always be smoothed in a post-processing phase. Here, the grid-based path from s to g (top, in black) cannot be smoothed because there are four obstacle cells (shaded). The optimal path is shown dashed (in blue).

Recently, researchers have looked at more sophisticated methods of obtaining smooth paths through grids. Konolige [6] presents an interpolated planner that first uses discrete grid planning to construct a cost-to-goal value function over the grid and then interpolates this result to produce a smooth path from the initial position to the goal. This method results in smoother paths for agents to traverse but does not incorporate the smoothed path cost into the planning process. Consequently, the resulting path is not necessarily as good as the path the algorithm would produce if continuous costs were calculated during planning. Further, this approach provides no replanning functionality.

Philippsen [11] presents an algorithm based on Fast Marching Methods [12] 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. Philippsen has shown that this approach generates nice paths in indoor environments [11]. 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 not as applicable to navigation in outdoor environments, which are often best represented by large, non-uniform cost grids.

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

Fig. 4. Some robots that currently use Field D* for path planning. These range from indoor planar robots (the Pioneers) to outdoor robots able to operate in harsh terrain (the XUV).

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 [7–9]. However, as LaValle points out [9], when the action space is also continuous, this becomes difficult, as solving for the value of a state now requires minimizing over an uncountably 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 globally smooth, low-cost paths. It relies upon an efficient, closed-form solution to the above minimization problem for 2D grids, which we introduce in Section 3. 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. It is very effective in practice and is currently employed as the path planner in a wide range of fielded robotic systems (see Fig. 4).

3 Improving Cost Estimation through Interpolation

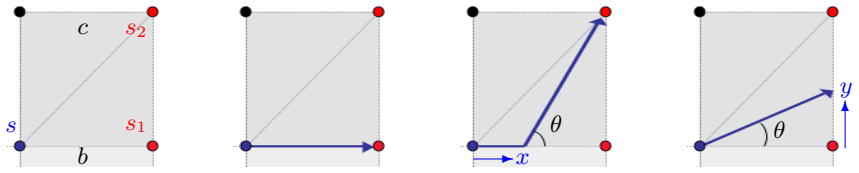
The key to our algorithm is a novel method of computing the path cost of each grid node s given the path costs of its neighboring nodes. By the path cost of a node we mean the cost of a path from the node to the goal. In classical grid-based planning this value is computed as

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

where $nbrs(s)$ is the set of all neighboring nodes of s (see Fig. 1), $c(s, s')$ 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



(i) (ii) (iii) (iv)

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.

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 Fig. 1(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 node s must pass through an edge connecting two consecutive neighbors of s, for example $\overrightarrow{s_1 s_2}$ (see Fig. 1(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 by one. To compute the path cost of s using edge $\overrightarrow{s_1 s_2}$, we use the path costs of nodes s_1 and s_2 and the traversal costs of the center and bottom cells (see Fig. 5).

In order 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 $g(s_2)$:

$$g(s_y) = yg(s_2) + (1 - y)g(s_1),$$

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

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

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 Fig. 5). 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 $(\bar{x}, \bar{y})$ be a pair of values for x and y that solve the above minimization. Because of the 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

ComputeCost(s, s_a, s_b)
01. if (s_a is a diagonal neighbor of s)
02. $s_1 = s_b$; $s_2 = s_a$;
03. else
04. $s_1 = s_a$; $s_2 = s_b$;
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. $v_s = \infty$;
09. else if $(g(s_1) \leq g(s_2))$
10. $v_s = \min(c, b) + g(s_1)$;
11. else
12. $f = g(s_1) - g(s_2)$;
13. if $(f \leq b)$
14. if $(c \leq f)$
15. $v_s = c\sqrt{2} + g(s_2)$;
16. else
17. $y = \min(\frac{f}{\sqrt{c^2 - f^2}}, 1)$;
18. $v_s = c\sqrt{1 + y^2} + f(1 - y) + g(s_2)$;
19. else
20. if $(c \leq b)$
21. $v_s = c\sqrt{2} + g(s_2)$;
22. else
23. $x = 1 - \min(\frac{b}{\sqrt{c^2 - b^2}}, 1)$;
24. $v_s = c\sqrt{1 + (1 - x)^2} + bx + g(s_2)$;
25. return v_s;

Fig. 6. The Interpolation-based Path Cost Calculation

[2]. Intuitively, if it is ever 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, the path will either travel along the entire bottom edge to s_1 (Fig. 5(ii)), or will travel a distance x along the bottom edge then take a straight-line path directly to s_2 (Fig. 5(iii)), or will take a straight-line path from s to some point s_y on the right edge (Fig. 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 goes straight to s_1 and will have a path cost of $\min(c, b) + g(s_1)$ (Fig. 5(ii)). If $f = b$ then the cost of a path using some portion of the bottom edge (Fig. 5(iii)) will be equivalent to the cost of a path using none of the bottom edge (Fig. 5(iv)). We can solve for the $\bar{y}$ of the latter path (equal to $1 - \bar{x}$ for the former path) that minimizes the path cost as follows. Firstly, let $k = f = b$. The path cost of s is

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

Taking the differential with respect to y and setting this equal to zero yields:

$$\bar{y} = \sqrt{\frac{k^2}{c^2 - k^2}}$$

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 $\bar{y} = 1 - \bar{x}$ into the above equation. The resulting algorithm for computing the minimum path cost of s given *any* two consecutive neighbors s_a and s_b is provided in Fig. 6.

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, $g(s) = \infty$;
03. if ($s \neq s_{goal}$)
04. $rhs(s) = min_{(s',s'') \in connbrs(s)}$ComputeCost($s$, s', s'');
05. if ($s \in$ *OPEN*) remove s from *OPEN*;
06. if ($g(s) \neq rhs(s)$) insert s into *OPEN* with key(s);

ComputeShortestPath()
07. while (min$_{s \in OPEN}$(key(s)) $\dot{<}$ key(s_{start}) OR $rhs(s_{start}) \neq g(s_{start})$)
08. remove state s with the minimum key from *OPEN*;
09. if ($g(s) > rhs(s)$)
10. $g(s) = rhs(s)$;
11. for all $s' \in nbrs(s)$ UpdateState(s');
12. else
13. $g(s) = \infty$;
14. for all $s' \in nbrs(s) \cup \{s\}$ UpdateState(s');

Main()
15. $g(s_{start}) = rhs(s_{start}) = \infty$; $g(s_{goal}) = \infty$;
16. $rhs(s_{goal}) = 0$; *OPEN* $= \emptyset$;
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).

4 Field D*

Once equipped with this interpolation-based cost calculation for a given node in our graph, we can plug it into a number of current planning and replanning algorithms to produce smooth paths. Fig. 7 presents our most recent formulation of *Field D**, an incremental replanning algorithm incorporating these interpolated path costs. This version of Field D* is based on D* Lite (with differences highlighted in red)[1].

In this figure, $connbrs(s)$ contains the set of consecutive neighbor pairs of state 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 Fig. 1(c). Apart from this construct, notation follows from the D* Lite algorithm: $g(s)$ is the current path cost of state s, $rhs(s)$ is the one-step lookahead path cost for s, *OPEN* is a priority queue containing inconsistent states (i.e., states s for which $g(s) \neq rhs(s)$) in increasing order of *key* values (line 01), s_{start} is the initial agent state, and s_{goal} is the goal state. $h(s_{start}, s)$ is a heuristic estimate of the cost of a path from s_{start} to s. Because the key value of each state contains two quantities, a lexicographic ordering is used, where key(s) $\dot{<}$ 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 the second element of

[1] As opposed to the original, graph-based version of D* Lite, lines {20 - 22} tailor Field D* to grids. Also, because paths may involve points on edges and not just cell corners, $h(s_{start}, s)$ must be small enough that when added to the cost of any edge emanating from s it is still not greater than a minimum cost path from s_{start} to s.

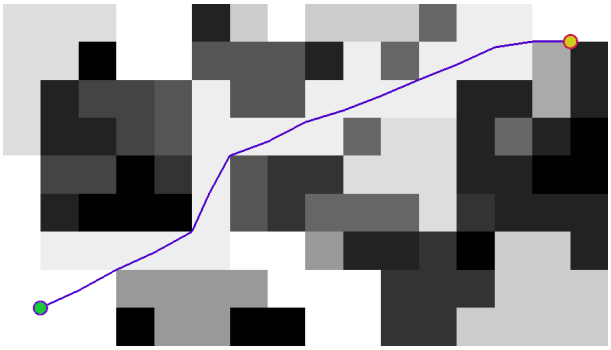


Fig. 8. A close-up of a path planned using Field D* showing individual grid cells. Notice that the path is not limited to entering and exiting cells at corner points.

key(s) is less than the second element of key(s'). For more details on the D* Lite algorithm and this terminology, see [4].

This is an unoptimized version of Field D*. In our extended technical report we discuss a number of optimizations that significantly improve the overall efficiency of planning and replanning with this algorithm [2].

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 Field D* applied to three non-uniform cost environments. In each of these figures, darker areas represent regions that are more costly to traverse. Unlike paths produced using classic grid-based planners, paths produced using Field D* are not restricted to a small set of headings. As a result, Field D* provides smoother, lower-cost paths through both uniform and non-uniform 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 Fig. 4).

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 optimized versions of the two approaches. We generated 100 different 1000 $\times$ 1000 non-uniform 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

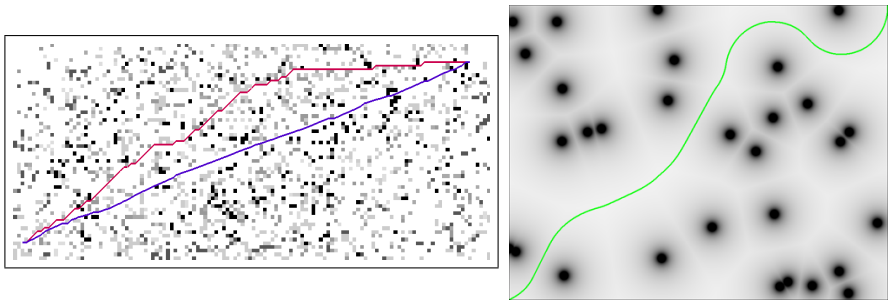


Fig. 9. (left) Paths produced by D* Lite (red/top) and Field D* (blue/bottom) in a 150×60 non-uniform cost environment. **(right)** Planning through a potential field of obstacles. At high grid resolutions, Field D* produces smooth curves through both uniform and non-uniform cost environments; this is not generally true of standard grid-based planners.

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 replanning runtime for Field D* on a 1.5 GHz Powerbook G4 was 0.07s. In practice, the algorithm is able to provide real-time performance on fielded systems.

6 Discussion and Conclusions

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 Fig. 5(i) and the optimal path for state s_2 travels above the obstacle and the optimal path for state s_1 travels below the obstacle). In such cases, the interpolated path cost of a point on an edge between two states 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 difference in smoothness between the paths returned by Field D* and D* Lite rather than their relative costs that is the true advantage of Field D*. In both uniform and non-uniform cost environments, Field D* provides smooth, sensible paths for our agents to traverse. Further, because its interpolation-based transition function is continuous, it has been used to successfully generate smooth, low cost paths in very low resolution grids when memory is limited.

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 non-uniform cost environments. We are currently looking at extending Field D* to interpolate over path headings, not just costs, in order to produce solutions that minimize the amount of turning required over the path. We are also developing a multi-resolution version of the algorithm able to plan over very large distances and a 3D version for air vehicles.

7 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. E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
2. 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.
3. 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.
4. 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.
5. S. Koenig and M. Likhachev. Incremental A*. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.
6. K. Konolige. A gradient method for realtime robot control. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2000.
7. R. Larson. A survey of dynamic programming computational procedures. *IEEE Transactions on Automatic Control*, pages 767–774, 1967.
8. R. Larson and J. Casti. *Principles of Dynamic Programming, Part 2*. Marcel Dekker, New York, 1982.
9. S. LaValle. *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/). To be published in 2006.
10. N. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.
11. R. Philippsen and R. Siegwart. An Interpolated Dynamic Navigation Function. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.
12. 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.
13. Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.