*Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*
*IOS Press, 2006*



# Multi-resolution Field D*

Dave Ferguson and Anthony Stentz
*The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA*
{*dif, tony*}*@cmu.edu*

**Abstract.** We present a multi-resolution path planner and replanner capable of efficiently generating paths across very large environments. Our approach extends recent work on interpolation-based planning to produce direct paths through non-uniform resolution grids. The resulting algorithm produces plans with costs almost exactly the same as those generated by the most effective uniform resolution grid-based approaches, while requiring only a fraction of their computation time and memory. In this paper we describe the algorithm and report results from a number of experiments involving both simulated and real field data.

## 1. Introduction

A common task in mobile robot navigation is to plan a path from an initial location to a desired goal location. Typically, this is done by storing the information about the environment in a discrete traversability cost map, where the cost of traversing each area depends on any number of factors, including safety risk, traversal time, or fuel expended, and then planning over this cost map.

In robotics, it is common to improve the efficiency of planning by approximating the cost map as a graph. For instance, given a cost map consisting of a 2D grid of cells of uniform size (or *resolution*), a graph is typically constructed by placing nodes at the centers or corners of the grid cells and inserting edges between each node and the nodes in its 8 adjacent cells. Several algorithms exist for planning paths over such graphs. Dijkstra's algorithm and A* are useful for computing initial paths to the goal from every location or a single location, respectively [2,6]. D* and D* Lite are very effective at repairing these paths when new information about the environment is received [9,5]. Combining uniform resolution grids with these incremental planning algorithms has been a very popular approach for mobile robot navigation in partially-known or dynamic environments.

However, such an approach has two significant limitations. Firstly, the small, discrete set of transitions modeled by the graph can reduce the quality of the resulting paths. In the case of a uniform resolution grid, each node in the extracted graph only has edges connecting it to its 8 closest neighboring nodes. This limits the heading of any solution path to increments of $\frac{\pi}{4}$. This can lead to suboptimal paths and undesirable behavior [3].

Secondly, uniform resolution grids can be very memory-intensive. This is because the entire environment must be represented at the highest resolution for

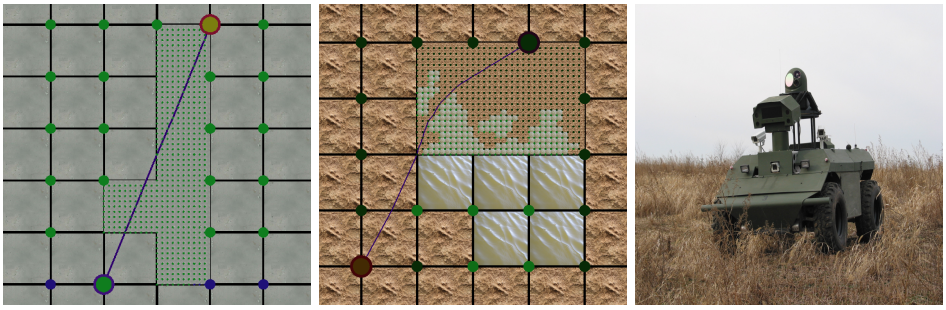

**Figure 1.** (*left, center*) Multi-resolution Field D* produces direct, low-cost paths (in blue/dark gray) through both high-resolution and low-resolution areas. (*right*) The GDRS XUV robot used for autonomous navigation of outdoor terrain. Multi-resolution Field D* was initially developed in order to extend the effective range of rugged outdoor vehicles such as this by one to two orders of magnitude.

which information is available. For instance, consider a robot navigating a large outdoor environment with a prior overhead map. The initial information contained in this map may be coarse. However, the robot may be equipped with onboard sensors that provide very accurate information about the area within some field of view of the robot. Using a uniform resolution grid-based approach, if *any* of the high-resolution information obtained from the robot's onboard sensors is to be used for planning, then the *entire environment* needs to be represented at a high-resolution, including the areas for which only the low-resolution prior map information is available. Storing and planning over this representation can require vast amounts of memory.

To overcome the first of these limitations, improvements to uniform resolution grid-based planners have been developed that use interpolation to produce paths not constrained to a small set of headings [7,3]. In particular, the Field D* algorithm extends the D* family of algorithms by using linear interpolation to derive the path cost of points not sampled in the grid [3]. This algorithm efficiently produces very low-cost paths with a range of continuous headings.

A number of techniques have been devised to address the second limitation. One popular approach is to use quadtrees rather than uniform resolution grids [8]. Quadtrees offer a compact representation by allowing large constant-cost regions of the environment to be modeled as single cells. They thus represent the environment using grids containing cells of varying sizes, known as non-uniform resolution grids or *multi-resolution* grids.

However, paths produced using quadtrees and traditional quadtree planning algorithms are again constrained to transitioning between the centers or corners of adjacent cells and can thus be grossly suboptimal. More recently, framed quadtrees have been used to alleviate this problem somewhat [1,10]. Framed quadtrees add cells of the highest resolution around the boundary of each quadtree region and allow transitions between these boundary cells through the quadtree region. As a result, the paths produced can be much less costly, but the computation and memory required can be large due to the overhead of the representation (and in pathological cases can become significantly more than is required by a full high-resolution representation). Also, segments of the path between adjacent

high-resolution cells suffer the same $\frac{\pi}{4}$ heading increment restrictions as classical uniform resolution grid approaches.

In this paper, we present an approach that combines the ideas of interpolation and non-uniform resolution grid representations to address both of the limitations mentioned above. The resulting approach provides very cost-effective paths for a fraction of the memory and, often, for a fraction of the *time* required by uniform resolution grid-based approaches. We begin by describing how linear interpolation can be used to provide direct, low-cost paths through non-uniform resolution grids. We then present our new algorithm, Multi-resolution Field D*, along with a number of example illustrations and results comparing it to its uniform resolution predecessor.

## 2. Interpolation-based Path Planning

Given a grid with nodes at the corners of every grid cell, classic grid-based planners restrict the available actions from each node to be direct paths to neighboring nodes. This restriction affects the quality and effectiveness of grid-based paths, as mentioned above.

### *2.1. Using interpolation to produce more accurate path cost estimates*

We can overcome the limitations of classical grid-based planning if we allow each node to transition to any point on any neighboring *edge*, rather than to just the small set of neighboring nodes. We define the neighboring edges of a node $s$ to be all edges that can be reached from $s$ via a straight-line path for which $s$ is *not* an endpoint (see Figure 2 (left)). The rationale here is that the optimal path from $s$ to the goal must pass through one of these neighboring edges, so if we knew the optimal paths from every point on any of these edges to the goal we could compute the optimal path for $s$. If $E$ is the set of neighboring edges of $s$ and $P_E$ is the set of all points on these edges, then the cost of the optimal path for $s$ is

$$g^*(s) = \min_{p \in P_E} (c(s,p) + g^*(p)), \tag{1}$$

where $c(s,p)$ is the cost of the cheapest path from $s$ to $p$ that does not travel through any other neighboring edges of $s$, and $g^*(p)$ is the cost of the optimal path from point $p$ to the goal.

Unfortunately, there are an infinite number of points $p$ on any one of the neighboring edges of $s$, so solving for the least-cost paths from each of these points is impossible. Fortunately, a method was presented in [3] that uses linear interpolation to approximate the path cost of any point on an edge using just the path costs of the two endpoints of the edge. In that work, this approximation was used to provide an efficient closed form solution to the above path cost calculation for any given corner node $s$ in a uniform resolution grid. Using this method, very accurate paths through uniform resolution grids can be efficiently computed.

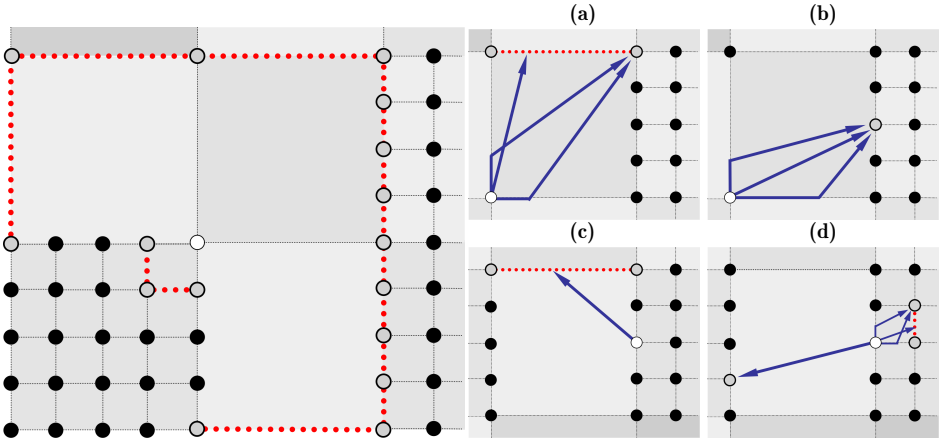


**Figure 2.** (left) The neighboring edges (dashed in red/gray, along with their endpoints in gray) from a given node (white) in a grid containing cells with two different resolutions: low-resolution and high-resolution. (a, b) Some of the possible paths to a neighboring edge/node from a low-resolution node. On the left are the possible optimal path types (in blue/dark gray) through the top low-resolution edge (dashed in red/gray) and its endpoint nodes (in gray). Linear interpolation is used to compute the path cost of any point along the top edge. On the right are the possible optimal path types (in blue/dark gray) to one neighboring high-resolution corner node (in gray). (c, d) Some of the possible paths (in blue/dark gray) from a high-resolution corner node (white) to a neighboring low-resolution edge (c) and to a high-resolution node (d, left) and edge (d, right).

### *2.2. Combining interpolation with multiple resolutions*

We can use the same basic idea to provide accurate path costs for nodes in non-uniform resolution grids. The main difference is that, while in the uniform resolution case each node $s$ has exactly 8 neighboring edges of uniform length, in the non-uniform resolution case a node may have many more neighboring edges with widely-varying lengths. However, linear interpolation can still be used to approximate the path costs of points along these edges, exactly as in the uniform resolution case.

As a concrete example of how we compute the path cost of a node in a multi-resolution grid, we now focus our attention on a grid containing cells of two different resolutions: high-resolution and low-resolution. This two-resolution case addresses the most common navigation scenario we are confronted with: a low-resolution prior map is available and the robot is equipped with high-resolution onboard sensors. Although we restrict our attention to this scenario, the approach is general and can be used with arbitrarily many different resolutions.

In a grid containing two different resolutions, each node can reside on the corner of a low-resolution cell, the corner of a high-resolution cell, and/or the edge of a low-resolution cell. Examples of each of these possibilities can be seen in Figure 2(b): the white node is the corner of a low-resolution cell and the gray node is the corner of a high-resolution cell *and* on the edge of a low-resolution cell. Let's look at each of these possibilities in turn.

Firstly, imagine we have a node that resides on the corner of a low-resolution cell. We can calculate the least-cost path from the node through this cell by

**ComputePathCost**$(s)$

```
1  v_s = ∞;
2  for each cell x upon which s resides
3    if x is a high-resolution cell
4      for each neighboring edge e of s that is on the boundary of x
5        v_s = min(v_s, min_{p∈P_e}(c(s,p) + g^i(p)));
6    else
7      for each neighboring edge e of s that is on the boundary of x
8        if e is a low-resolution edge
9          v_s = min(v_s, min_{p∈P_e}(c(s,p) + g^i(p)));
10       else
11         v_s = min(v_s, min_{p∈EP_e}(c(s,p) + g(p)));
12 return v_s;
```

**Figure 3. Computing the Path Cost of a Node $s$ in a Grid with Two Resolutions**

looking at all the points on the boundary of this cell and computing the minimum-cost path from the node through any of these points. We can approximate the cost of this path by using linear interpolation to provide the path cost of arbitrary boundary points, exactly as in uniform resolution Field D*. However, some of the boundary may be comprised of high-resolution nodes. In such a case, we can either use interpolation between adjacent high-resolution nodes and allow the path to transition to any point on an adjacent high-resolution edge, or we can restrict the path to transitioning to one of the high-resolution nodes. The former method provides more accurate approximations, but it is slightly more complicated and less efficient. Depending on the relative sizes of the high-resolution cells and the low-resolution cells, either of these approaches may be appropriate. For instance, if the high-resolution cells are much smaller than the low-resolution cells, then interpolating across the adjacent high-resolution edges when computing the path from a low-resolution node is not critical, as there will be a wide range of heading angles available just from direct paths to the adjacent high-resolution nodes. However, if the high-resolution cells are not significantly smaller than the low-resolution cells then this interpolation becomes important, as it allows much more freedom in the range of headings available to low-resolution nodes adjacent to high-resolution nodes. In Figure 2 we illustrate the latter, simpler approach, where interpolation is used to compute the path cost of points on neighboring, strictly low-resolution edges (e.g. the top edge in (a)), and paths are computed to each neighboring high-resolution node (e.g. the gray node on the right edge in (b)).

For nodes that reside on the corner of a high-resolution cell, we can use interpolation to approximate the cost of the cheapest path through the high-resolution cell (see the paths to the right edge in (d)). Finally, for nodes that reside on the edge of a low-resolution cell, we can use a similar approach as in our low-resolution corner case. Again, we look at the boundary of the low-resolution cell and use interpolation to compute the cost of points on strictly low-resolution edges (e.g. the top edge in (d)), and for each high-resolution edge we can choose between using interpolation to compute the cost of points along the edge, or restricting the path to travel through one of the endpoints of the edge. The latter approach is illustrated for computing a path through the left edge in (d).

**key**($s$)
1 return $[\min(g(s), rhs(s)) + h(s_{start}, s); \min(g(s), rhs(s))]$;

**UpdateNode**($s$)
2 if $s$ was not visited before, $g(s) = \infty$;
3 if ($s \neq s_{goal}$)
4 $rhs(s)$ = ComputePathCost($s$);
5 if ($s \in OPEN$) remove $s$ from $OPEN$;
6 if ($g(s) \neq rhs(s)$) insert $s$ into $OPEN$ with key($s$);

**ComputeShortestPath**()
7 while ($\min_{s \in OPEN}$(key($s$))$\dot{<}$ key($s_{start}$) OR $rhs(s_{start}) \neq g(s_{start})$)
8 remove node $s$ with the minimum key from $OPEN$;
9 if ($g(s) > rhs(s)$)
10 $g(s) = rhs(s)$;
11 for all $s' \in nbrs(s)$ UpdateNode($s'$);
12 else
13 $g(s) = \infty$;
14 for all $s' \in nbrs(s) \cup \{s\}$ UpdateNode($s'$);

**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 to grid or traversal costs;
21 for all new cells or cells with new traversal costs $x$
22 for each node $s$ on an edge or corner of $x$
23 UpdateNode($s$);

**Figure 4. The Multi-resolution Field D* Algorithm (basic version)**

Thus, for each node, we look at all the cells that it resides upon as either a corner or along an edge and compute the least-cost path through each of these cells using the above approximation technique. We then take the cheapest of all of these paths and use its cost as the path cost of the node.

Pseudocode of this technique is presented in Figure 3. In this figure, $P_e$ is the (infinite) set of all points on edge $e$, $EP_e$ is a set containing the two endpoints of edge $e$, $g^i(p)$ is an approximation of the path cost of point $p$ (calculated through using linear interpolation between the endpoints of the edge $p$ resides on), $c(s, p)$ is the cost of a minimum-cost path from $s$ to $p$, and $g(p)$ is the current path cost of corner point $p$. We say an edge $e$ is a 'low-resolution edge' (line 8) if both the cells on either side of $e$ are low-resolution. An efficient solution to the minimizations in lines 5 and 9 was presented in [3].

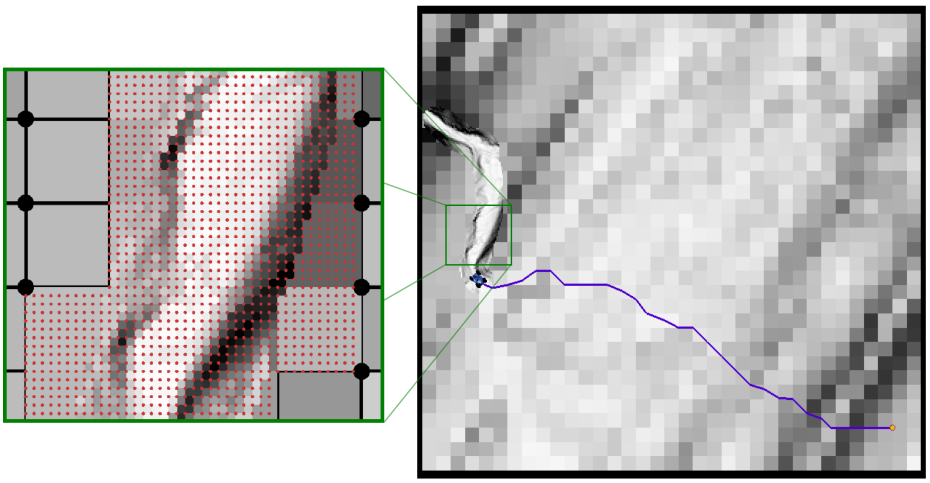

**Figure 5.** Multi-resolution Field D* used to guide an agent through a partially-known environment. On the left is a section of the path already traversed showing the high-resolution cells. This data was taken from Fort Indiantown Gap, Pennsylvania.

## 3. Multi-resolution Field D*

The path cost calculation discussed above enables us to plan direct paths through non-uniform resolution grids. We can couple this with any standard path planning algorithm, such as Dijkstra's, A*, or D*. Because our motivation for this work was robotic path planning in unknown or partially-known environments, we have used it to develop an incremental replanning algorithm which we call *Multi-resolution Field D**.

A basic version of the algorithm is presented in Figure 4. Here, the ComputeMinPathCost function (line 4) takes a node $s$ and computes the path cost for $s$ using the path costs of all of its neighboring nodes and interpolation across its neighboring edges. Apart from this, notation follows the D* Lite algorithm: $g(s)$ is the current path cost of node $s$, $rhs(s)$ is the one-step lookahead path cost for $s$, *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, where $\text{key}(s) \dot{<} \text{key}(s')$ iff the first element of $\text{key}(s)$ is less than the first element of $\text{key}(s')$ or the first element of $\text{key}(s)$ equals the first element of $\text{key}(s')$ and the second element of $\text{key}(s)$ is less than the second element of $\text{key}(s')$. For more details on the D* Lite algorithm and this terminology, see [5].

As with other members of the D* family of algorithms, significant optimizations can be made to this initial algorithm. In particular, several of the optimizations presented in [4] are applicable and were used in our implementation.

Because of its use of interpolation, Multi-resolution Field D* produces extremely direct, cost-effective paths through non-uniform resolution grids. Figures 1 and 5 show example paths planned using the algorithm.

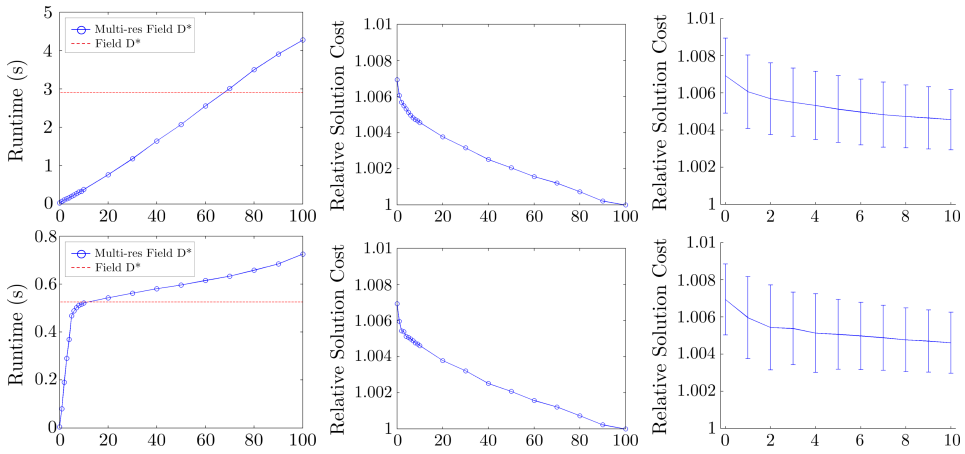


**Figure 6.** Computation time and solution cost as a function of how much of the environment is represented at a high-resolution. The $x$-axis of each graph depicts the percentage of the map modeled using high-resolution cells, ranging from 0 (all modeled at a low-resolution) to 100 (all modeled at a high-resolution). (*top*) Initial planning. A path was planned from one side to the other of 100 randomly generated environments and the results were averaged. (*bottom*) Replanning. 5% of each environment was randomly altered and the initial paths were repaired.

## 4. Results

Multi-resolution Field D* was originally developed in order to extend the effective range over which unmanned ground vehicles (such as the XUV vehicle in Figure 1) could operate by orders of magnitude. We have found the algorithm to be extremely effective at reducing both the memory and computational requirements of planning over large distances. To quantify its performance, we ran several experiments comparing Multi-resolution Field D* to uniform resolution Field D*. We used uniform resolution Field D* for comparison because it produces less costly paths than regular uniform grid-based planners and far better paths than regular non-uniform grid-based planners.

Our first set of experiments investigated both the quality of the solutions and the computation time required to produce these solutions as a function of the memory requirements of Multi-resolution Field D*. We began with a randomly generated $100 \times 100$ low-resolution environment with an agent at one side and a goal at the other. We then took some percent $p$ of the low-resolution cells (centered around the agent) and split each into a $10 \times 10$ block of high-resolution cells. We varied the value of $p$ from 0 up to 100. We then planned an initial path to the goal. Next, we randomly changed 5% of the cells around the agent and replanned a path to the goal. We focussed the change around the robot to simulate new information being gathered in its vicinity. The results from these experiments are presented in Figure 6. The $x$-axis of each graph represents how much of the environment was represented at a high-resolution. The left graphs show how the time required for planning changes as the percent of high-resolution cells increases, while the middle and right graphs show how the path cost changes. The $y$-values in the middle and right graphs are path costs relative to the path cost computed when 100% of the environment is represented at a high-resolution. The right

| | Field D* | Multi-res Field D* |
|---|---|---|
| Total Planning and Replanning Time (s) | 0.493 | 0.271 |
| Initial Planning Time (s) | 0.336 | 0.005 |
| Average Replanning Time (s) | 0.0007 | 0.0012 |
| Percent high-resolution cells | 100 | 13 |

**Figure 7.** Results for uniform resolution Field D* versus Multi-resolution Field D* on a simulated robot traverse through real data acquired at Fort Indiantown Gap. The robot began with a low-resolution map of the area and updated this map with a high-resolution onboard sensor as it traversed the environment. The map was 350 × 320 meters in size.

graph shows the standard error associated with the relative path costs for smaller percentages of high-resolution cells. As can be seen from these results, modeling the environment as mostly low-resolution produces paths that are only trivially more expensive than those produced using a full high-resolution representation, for a small fraction of the memory and computational requirements.

This first experiment shows the advantage of Multi-resolution Field D* as we reduce the percentage of high-resolution cells in our representation. However, because there is some overhead in the multi-resolution implementation, we also ran uniform resolution Field D* over a uniform, high-resolution grid to compare the runtime of this algorithm with our Multi-resolution version. The results of uniform resolution Field D* have been overlaid on our runtime graphs. Although uniform resolution Field D* is more efficient than Multi-resolution Field D* when 100% of the grid is composed of high-resolution cells, it is far less efficient than Multi-resolution Field D* when less of the grid is made up of high-resolution cells.

Our second experiment simulated the most common case for outdoor mobile robot navigation, where we have an agent with some low-resolution prior map and high-resolution onboard sensors. For this experiment, we took real data collected from Fort Indiantown Gap, Pennsylvania, and simulated a robotic traverse from one side of this 350 × 320 meter environment to the other. We blurred the data to create a low-resolution prior map (at 10 × 10 meter accuracy) that the robot updated with a simulated medium-resolution sensor (at $1 \times 1$ meter accuracy with a 10 meter range) as it traversed the environment.

The results from this experiment are shown in Figure 7. Again, Multi-resolution Field D* requires only a fraction of the memory of uniform resolution Field D*, and its runtime is very competitive. In fact, it is only in the replanning portion of this final experiment that Multi-resolution Field D* requires more computation time than uniform resolution Field D*, and this is only because the overhead of converting part of its map representation from low-resolution to high-resolution overshadows the trivial amount of processing required for replanning.

In general, our results have shown that Multi-resolution Field D* produces paths that are trivially more expensive than those returned by uniform resolution Field D*, and it provides these paths for a fraction of the memory. Further, in many cases it is also significantly more computationally efficient than uniform resolution Field D*.

## 5. Conclusions

We have presented Multi-resolution Field D*, an interpolation-based path planning algorithm able to plan direct, low-cost paths through non-uniform resolution grids. This algorithm addresses two of the most significant shortcomings of grid-based path planning: solution quality and memory requirements. By using a non-uniform resolution grid to represent the environment and linear interpolation to approximate the path costs of points not sampled on the grid, Multi-resolution Field D* is able to provide solutions of comparable quality to the most effective uniform resolution grid-based algorithms for a fraction of their memory and computational requirements. It is our belief that, by combining interpolation with non-uniform representations of the environment, we can 'have our cake and eat it too', with a planner that is extremely efficient in terms of both memory and computational requirements while still producing high quality paths.

## 6. Acknowledgements

This work was sponsored in part 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] D. Chen, R. Szczerba, and J. Uhran. Planning conditional shortest paths through an unknown environment: A framed-quadtree approach. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1995.

[2] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[3] D. Ferguson and A. Stentz. Field D*: An Interpolation-based Path Planner and Replanner. *International Symposium on Robotics Research (ISRR)*, 2005.

[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] S. Koenig and M. Likhachev. D* Lite. *National Conference on Artificial Intelligence (AAAI)*, 2002.

[6] N. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[7] R. Philippsen and R. Siegwart. An Interpolated Dynamic Navigation Function. *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[8] H. Samet. Neighbor Finding Techniques for Images Represented by Quadtrees. *Computer Graphics and Image Processing*, 18:37–57, 1982.

[9] Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning. *International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[10] A. Yahja, S. Singh, and A. Stentz. An efficient on-line path planner for outdoor mobile robots operating in vast environments. *Robotics and Autonomous Systems*, 33:129–143, 2000.