An Efficient On-line Path Planner for Mobile Robots Operating in Vast Environments

Alex Yahja, Sanjiv Singh and Anthony Stentz

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

Mobile robots operating in vast outdoor unstructured environments often have only incomplete maps and must deal with new objects found during traversal. Path planning in these environments must be incremental to accommodate new information and must use efficient representations. This paper reports recent results in path planning using an efficient data structure (framed quadtrees) and an optimal algorithm (D*) to incrementally replan optimal paths. In particular, we show the difference in performance when the robot starts with no information about the world versus when it starts with partial information about the world. Our results indicate that, as would be expected, starting with partial information is better than starting with no information. However, in many cases, the effect of partial information is performance that is almost as good as starting out with complete information about the world, while the computational cost incurred is significantly lower. Our system has been tested in simulation as well on an autonomous jeep, equipped with local obstacle avoidance capabilities and results from both simulation and real experimentation are discussed.

Keywords: optimal path planning; framed quadtrees; outdoor mobile robots; unstructured environments

1. Introduction

Path planning for a mobile robot is typically stated as getting from one place to another. The robot must successfully navigate around obstacles, reach its goal and do so efficiently. Outdoor environments pose special challenges over the structured world that is often found indoors. Not only must a robot avoid colliding with an obstacle such as a rock, it must also avoid falling into a pit or ravine and avoid travel on terrain that would cause it to tip over.

Vast areas have their own associated issues. Such areas typically have large open space where a robot might travel freely and are sparsely populated with obstacles. However, the range of obstacles that can interfere with the robot's passage is large- the robot must still avoid a rock as well as go around a mountain. Large areas are unlikely to be mapped at high resolution *a priori* and hence the robot must explore as it goes, incorporating newly discovered information into its database. Hence, the solution must be incremental by necessity.

Another challenge is dealing with a large amount of information and a complex model of the vehicle. Taken as a single problem, so much information must be processed to determine the next action that it is not possible for the robot to perform at any reasonable rate. We deal with this issue by using a layered approach to navigation. That is, we decompose navigation into two levels- local and global. The job of local planning is to avoid obstacles, reacting to sensory data as

quickly as possible while driving towards a subgoal [4][5]. A more deliberative process, operating at a coarser resolution of information is used to decide how best to select the subgoals such that the goal can be reached. This approach has been used successfully in the past in several systems at Carnegie Mellon [2][13].

Approaches to path planning for mobile robots can be broadly classified into two categories- those that use exact representations of the world (e.g. [6]), and those that use a discretized representation (e.g. [1][7]). The main advantage of discretization is that the computational complexity of path planning can be controlled by adjusting cell size. In contrast, the computational complexity of exact methods is a function of the number of obstacles and/or the number of obstacle facets, which we cannot normally control. Even with discretized worlds path planning for outdoor environments can be computationally expensive and on-line performance is typically achieved by use of specialized computing hardware as in [7]. By comparison the proposed method requires general purpose computing only. This is made possible by precomputing an optimal path off-line given whatever *a priori* map is available, and then efficiently modifying the path as new map information becomes available, on-line.

Methods that use uniform grid representations must allocate large amounts of memory for regions that may never be traversed or that may not contain any obstacles. Efficiency in map representation can be obtained by the use of quadtrees, but at a cost of optimality. Recently, a new data structure called a framed quadtree has been suggested as a means to overcome some of the issues related to the use of quadtrees [3]. We have used this data structure to extend an existing path planner that has in the past used uniform (regular) grid cells to represent terrain. This path planner, D* [12], has been shown to produce optimal paths in changing environments by incorporating knowledge of the environment as it is incrementally discovered. Coupling the two provides a method that is correct, resolution-complete and resolution-optimal. It also does this efficiently. The paths are always shorter, and in all but the most cluttered environments, it executes faster and uses less memory than when regular grids are used [15]. In general, the sparser or the more unknown the world, the greater advantage of using framed-quadtrees.

2. Optimal and Incremental Path Planning

Unstructured outdoor environments are often not only sparse but also at best have been mapped at a coarse resolution. If complete and accurate maps were available, it would be sufficient to use a standard search method such as A* [8] to produce a path. Imperfections in control, inertial sensing, and perception often introduce erroneous and changing information. Thus, a mobile robot must gather new information about the environment and efficiently replan new paths based on this new information. In these partially known environments, a good traverse can be achieved by replanning paths incorporating information as it becomes available. This approach, known as Best Information Planning [14], produces a path based on all available information and replans from the current position to the goal when new information becomes available. Best Information Planning is intuitively satisfying and has been shown to produce lower-cost traverses on average than other selected algorithms for unknown and partially-known environments. Also, Best Information Planning is able to make use of prior information to reduce the traversal cost.

It is possible to use A* to replan a new path every time it is needed, but this approach is computationally expensive. Our approach is to use D* (described in detail in [11][12]) that allows

replanning to occur in real-time. Incremental replanning makes it possible to greatly reduce computational cost, as it only updates the path locally, when possible, to obtain the globally optimal path. D* produces the same results as planning from scratch with A* for each new piece of information, but is much faster.

3. Efficient Representation of Space

While discretization of space allows for control over the complexity of path planning, it also provides a flexible representation for obstacles and cost maps and eases implementation of search methods. One method of cell decomposition is to tessellate space into equally sized cells each of which is connected to its eight neighbors. This method, however, has two drawbacks: resulting paths can be suboptimal and memory requirements are high. Quadtrees address the latter problem, while framed quadtrees address both problems, especially in sparse environments. Below we compare the representations using a simple example.

3.1 Regular Grids

Regular grids represent space inefficiently. Natural terrain is usually sparsely populated and is often not completely known in advance. Many equally sized cells are needed to encode empty areas, making search expensive since a very large number of cells must be searched. Moreover, regular grids allow only eight angles for direction, resulting in abrupt changes in path direction and an inability, in some cases, to generate a straight path through empty areas (Fig. 1a). It is possible to smooth such jagged paths, but there is no guarantee that the smoothed path will converge to the truly optimal path.

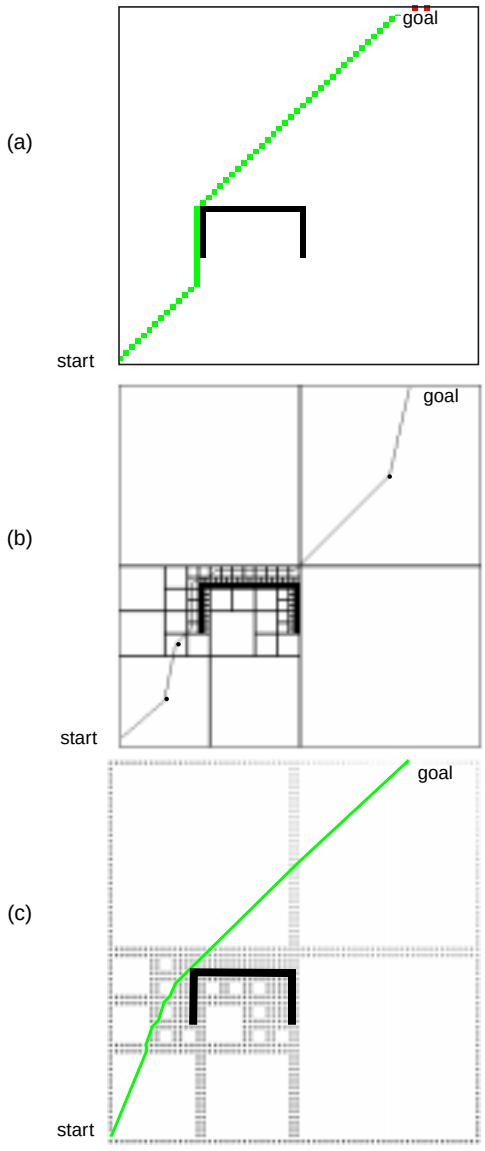


Fig. 1. An example of a path generated using (a) regular grid representation, (b) quadtree, (c) framed-quadtree. The black cul-de-sac is an obstacle.

3.2. Quadtrees

One way to reduce memory requirements is to use a quadtree instead of a regular grid. A quadtree [10] is based on the successive subdivision of a region into four equally sized quadrants. The quadrants are labelled NE (Northeast), NW (Northwest), SW (SouthWest), and SE (Southeast), respectively. The criterion for splitting a region into four smaller regions is that if a region still contains obstacles, it is split until either a subregion free of obstacles is encountered or the smallest cell is reached. In the latter case, the cell which is partially filled is marked as an obstacle cell. Figure 2 shows the quadtree subdivision of the map area and the corresponding quadtree data structure. The leaves (i.e., quadtree nodes without children) are called terminal

quadtree nodes. A quadtree with a single top-level node is called single-root quadtree. An array of connected single-root quadtrees is called a multiple-root quadtree.

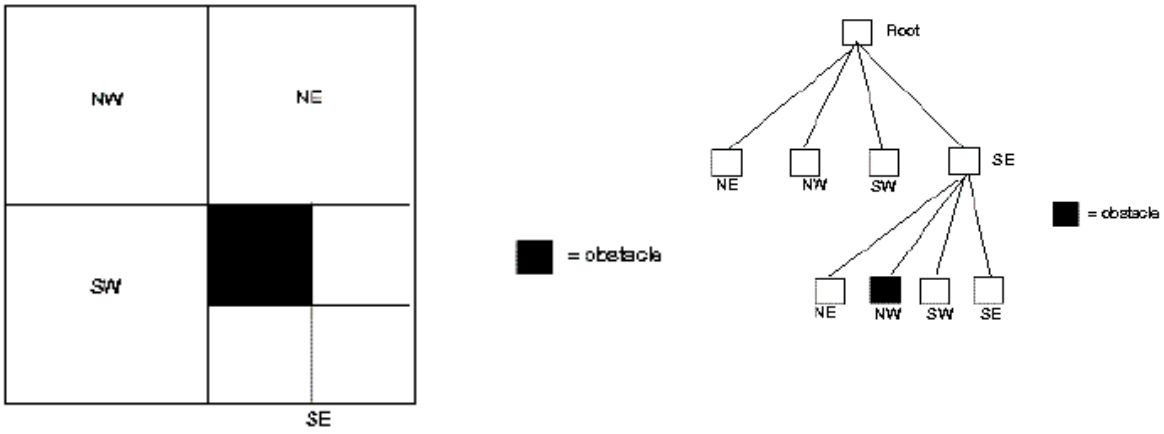


Fig. 2. Quadtree tessellation of a region due to a single obstacle and the corresponding quadtree data structure.

Quadtrees allow efficient partitioning of the environment since single cells can be used to encode large empty regions. However, paths generated using quadtrees are suboptimal because they are constrained to segments between the centers of the cells. Fig. 1b shows an example path generated using a quadtree.

3.3 Framed Quadtrees

To remedy the above problem, we have used a modified data structure in which cells of the highest resolution are added around the perimeter of each quadtree region. This augmented representation is called a framed quadtree [3]. A path generated using this representation is shown in Fig. 1c. The small grey rectangles around the cells are the border cells of each quadrant. The shade of gray of the border cells signifies the cost to the goal: the lighter the shade, the less the cost to the goal from that border cell.

This framed quadtree representation permits many more angles of direction, instead of just eight angles as in the case of regular grids. A path can be constructed between two border cells lying far away from each other. Most importantly, the paths generated more closely approximate optimal paths. The drawback of using framed quadtrees is that they require more memory than regular grids in uniformly, highly cluttered environments because of the overhead involved in the book-keeping.

4. Implementation of Framed-Quadtree D*

We have developed a system, called Frame-Quadtree D*, implemented by creating a frame-quadtree data structure and interfacing it to the D* path planning algorithm. The frame-quadtree data structure itself is created by building a quadtree data structure, connecting each neighboring terminal quadtree node, adding the border cells, and connecting each border cell to all its neighboring border cells.

4.1 Building Framed Quadtrees

After quadtree nodes are recursively built, neighbor pointers to each quadtree node are assigned. Samet's algorithm finds the neighboring terminal quadtree nodes by going up and down the quadtree structure guided by the nodes' size, quadrants, and directions [9]. Specifically, two procedures are utilized. The former finds quadtree neighbors whose quadtree areas intersect at a common side (also known as the side-adjacent neighbors). The latter finds quadtree neighbors whose areas intersect at a common corner (also known as the corner neighbors). To connect all terminal quadtree nodes with their neighbors, we commence from the smallest quadtree node proceeding to large ones until all quadtree nodes are properly connected to all their neighbors.

After allocating a numbered list of border cells around the perimeter of every quadtree node, border cell neighborhood pointers are assigned. If both border cells are inside the same quadtree node, they are simply connected. On the other hand, if neighboring border cells belong to different quadtree nodes, it is necessary to determine if the quadtree nodes are side-adjacent or corner neighbors. If they are side-adjacent or corner neighbors, we pick the smaller quadtree node as our reference and find at most 3 neighboring border cells adjacent to each border cell of this smaller quadtree node. Border cell neighborhood pointers of these border cells are then allocated to point to each of their neighboring border cells. These are called outer border cell neighborhood pointers. In addition to these structural pointers, each border cell has one specialized pointer, called a backpointer, for use in the D* algorithm.

4.2 Interfacing D* with Framed-Quadtrees

D* operates on a cost graph. Framed-quadtree border cells represent the nodes in the cost graph. Each border cell has a static cost (an obstacle cell has a prohibitive static cost while a free cell has a small static cost corresponding to the cost of traversing half the cell) and a heuristic cost corresponding to the estimated cost of traveling to the goal from that border cell. The link between two border cells corresponds to an edge in the cost graph.

The heuristic cost of traveling through this edge is calculated by:

$$\text{heuristic cost AB} = (\text{static cost of border cell A} + \text{static cost of border cell B}) * \text{straight line cell distance between A and B.}$$

where static cost is defined as the cost of traversing half of the cell size.

To determine the cost of traveling from point A inside an empty quadtree area, the same formula is used with the static cost associated with point A set equal to an empty cell cost. Before computing the heuristic cost of point A, however, we find the lowest cost border cell inside this empty quadtree node as cell B in the above formula. Along with the estimated path cost to the goal, the static cost affects the heuristic cost. We assign a heuristic cost of 0 to the goal cell and start propagating D* values from the goal cell initially. After this first propagation, D* only needs to visit cells locally as needed, that is, not all border cells have to be visited in order for D* to recompute the optimal path. As the robot senses its environment, it puts new cells onto the OPEN list of D*. The OPEN list contains cells queued for heuristic cost recomputing. D* then calculates the optimal path based on those cells' static and heuristic values [12].

The results of optimal path calculation by D* are adjusted heuristic values for the cells (thus, an adjusted cost graph) and adjusted backpointers. Backpointer of each border cell points to the

neighboring border cell having the least cost to the goal. Following backpointers from any cell, an optimal path is recovered from any cell.

4.3 Updating Border Cells based on New Information

As discussed above, if a new obstacle is detected inside an empty quadtree node, we split that node into 4 equally sized nodes. However, we need to split the node in a principled way, so that the consistency of D* values is maintained.

If A and B are the border cells in a quadtree node that is going to be split, the following is done:

1. Check if the neighborhood pointers between A and B are affected by the split. If they are not, then do nothing, else proceed to step 2.
2. Label the link AB “to be deleted”.
3. Place both A and B on the OPEN list with their current heuristic values. Do steps 1-3 for all border cell pairs in the quadtree node.
4. D* will pop up cells from the OPEN list for recomputation of heuristic costs (also known as expansion). When expanding a cell X with D*, if a cell Y has a backpointer to X through a “to be deleted” link, place Y back on the OPEN list with its heuristic cost set equal to the maximum cost MAXCOST and set Y's backpointer to NULL.
5. Delete all links connected to cell X that are labelled “to be deleted”.

The splitting of a quadtree node also causes the modification of framed-quadtree structures by re-executing the framed quadtree build-up procedures incrementally and locally. In this way, the framed-quadtree mimics the local and incremental nature of D*, giving us efficient computation. This is the beauty of our approach: it is locally adaptable, it changes the frame quadtree structures locally and applies the D* algorithm locally in response to some local environment change, but it still has a globally optimal path.

4.4 Modified Framed Quadtrees

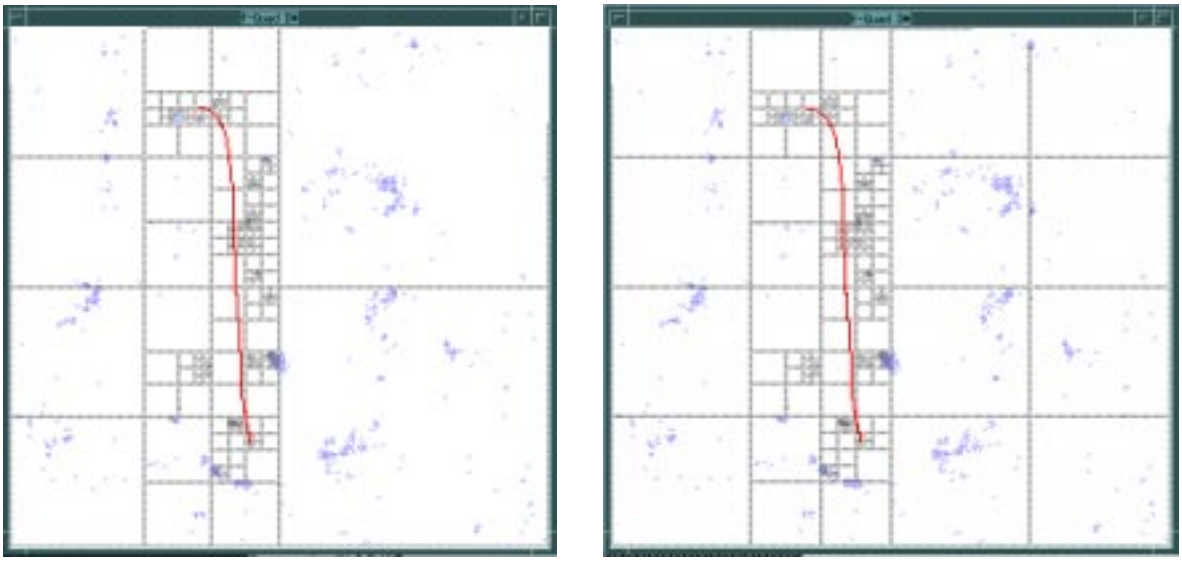
Framed quadtrees allows a border cell to have high number of connections to its neighbor inside a large quadtree node. The higher the number of border cells around the perimeter of a quadtree node, the higher the connectivity, defined as the maximum number of connections from a border cell to others.

For a very large world, however, the connectivity would be unnecessarily high, which would cause a slowdown in querying heuristic cost values inside large empty quadtree areas. There are several ways to address this:

- Limiting the area of a single-root framed quadtree and allocating multiple connected framed-quadtrees as needed. This leads to a Multiple-Root Framed Quadtree.
- Allocating bigger border cells first for huge empty quadtree nodes. As the quadtree nodes get nearer to the vehicle or get split into smaller nodes, the border cells get smaller, too. This leads to a Modified Single-Root Framed Quadtree. Ordinary framed quadtrees are denoted as Single-Root Framed-Quadtree.

For a 256 x 256 framed-quadtree node, the connectivity can be as high as 3 x 256 - 4 + 2 + 3 = 769. Figure 3 compares Multiple-Root and ordinary Single-Root Framed Quadtrees in binary-cost 256 x 256 worlds. All obstacles are unknown in advance. The gray multiple-size rectangles are

visited border cells, part of the framed-quadtree structure. Figure 3a shows an ordinary Single-Root Framed-Quadtree D* running in a 256 m x 256 m binary world. After the vehicle arrives at the goal, the connectivity is 385, as shown by a border cell in the largest remaining quadtree rectangle. Figure 3b shows a Multiple-Root Framed-Quadtree D* (4 roots) running in the same 256 m x 256 m binary world. The connectivity after vehicle arrives at the goal is 193.



(a) (b)

Fig. 3. (a) Ordinary Single-Root Framed-Quadtree D* allocates less quadtree nodes but has more connectivity. (b) Multiple-Root Framed-Quadtree D* uses more quadtree nodes but has less connectivity.

Table 1 shows the comparison of cells created by the above multiple-root and ordinary single-root framed quadtrees.

Table 1: Cell Count Comparison

	Ordinary Single-Root Framed-Quadtree D*	Multiple-Root Framed-Quadtree D*
Final connectivity	385	193
Quadtree nodes	552	693
Border cells	6,224	7,224
Border cell connections	381,762	380,256

As shown, multiple-root framed quadtrees use less cells and less connections than ordinary single-root framed quadtrees, and have less final connectivity.

5. Graph Complexity

As stated, border cells form the nodes in the D* cost graph. Let k be the maximum number of border cells along one dimension. For an empty world of $k \times k$ cells, the number of border cells is $4 \times (k-1)$ cells. The number of links between border cells is $(4(k-2)(3k-4+2+3) + 4(2k-3+2+5))/2 = 2(3k^2 - 3k +2)$. If this empty world is subdivided into 4 equally sized quadtree areas, the number of border cells becomes $4 \times (4 \times (k/2 - 1))$, which is an increase. The number of links, however, is $4 \times 2 \times (3 \times (k/2)^2 - 3 \times (k/2) + 2) = 2(3k^2 - 6k +8)$, which is a decrease if $k>2$ and is the same if $k=2$. For a world of $k \times k$ cells with every cell an obstacle, the number of border cells is k^2, the number of links is $8k^2/2$. Thus, along the subdivision process, the number of border cells will increase from $4k-4$ to k^2 cells, while the number of links will decrease from $(6k^2 - 6k +4)$ to $4k^2$.

Let n be the number of nodes in the graph (that is, $n = k \times k$) and b be the branching factor (which is upper-bounded by $4k-4$ for framed quadtrees). Note that the worst case complexity of D* heuristic cost evaluation is $O(b\ n\ log(n))$.

For k=256, the number of border cells for an empty 256 x 256 world is 1020, while the number of links is 391684. For a 256 x 256 world with every cell an obstacle, the number of border cells is 65536, while the number of links is 262144. Figure 4 below shows the typical connection patterns for a corner and a side border cell, respectively.

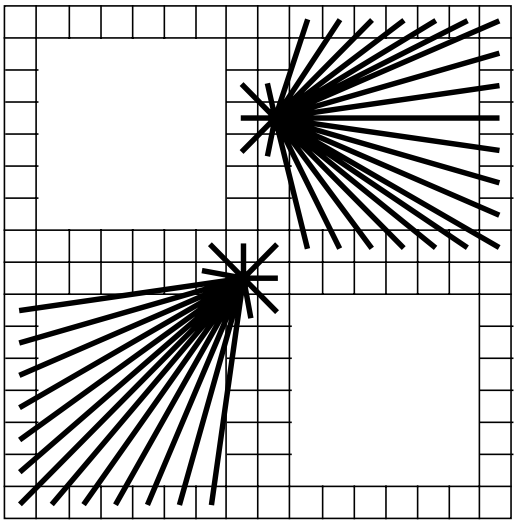


Fig. 4. Typical connection patterns for a corner border cell and a side border cell.

6. Simulation Results

We have conducted path planning experiments using simulated fractal terrains of varying complexity. The simulation environment is a 256 x 256 cell world with obstacles (each a 1 x 1 cell) distributed by a fractal terrain generator. The amount of clutter in the world is parameterized by a fractal gain.

We use two representations for obstacles. The first is a simplified representation in which the terrain has a binary cost depending on a fractal generator. Either the terrain is passable, in which

case the cost to move from one cell to another is the Euclidean distance, or, the terrain is impassable and the cost to move to a cell containing an obstacle is infinite. The second is a more realistic method that encodes the cost of moving from one cell to another as a function of a fractal, resulting in a continuous-cost map. In this case, the fractal generator directly produces the cost map; that is, it directly produces the cost of traversing from one cell to another. This cost map can be thought of as a derivative of an elevation map.

Below we show simulation results that empirically show the change in size and connectivity of the search graph. Also shown is the benefit of using framed quadtrees over regular grids and the benefit of using partial map information.

6.1 Size and Connectivity of the Search Graph

Figure 5 shows that the comparison of border cell count between completely unknown and fully known worlds, both of fractal 256 x 256 worlds. The number of obstacles grows with fractal density.

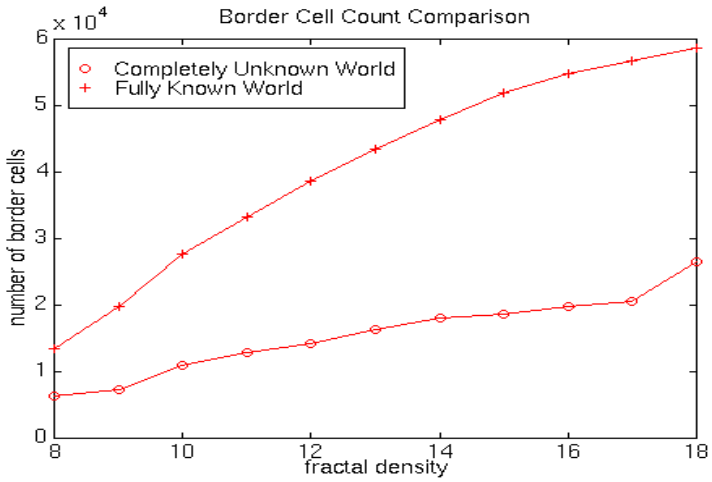


Fig. 5. Border cell count comparison.

As shown above, fully known worlds use more border cells than completely unknown worlds. Both, however, use an increasing number of border cells as the density of obstacles increases. Figure 6 below shows the comparison of border cell's link count for completely unknown and fully known worlds.

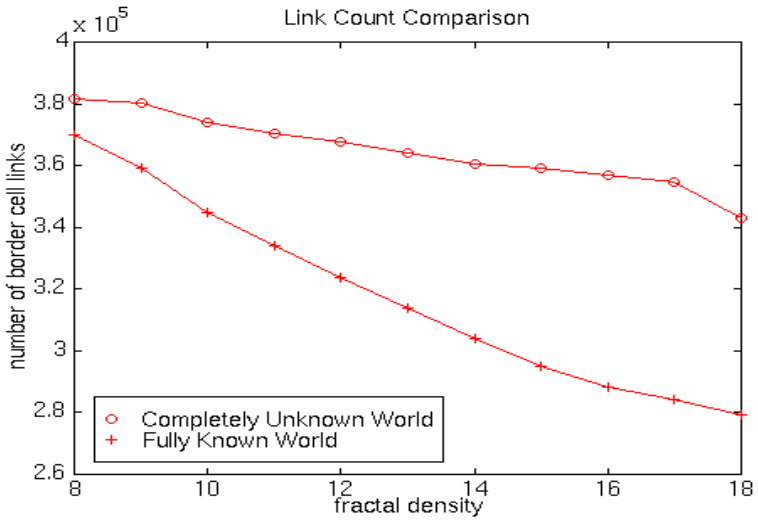


Fig. 6. Border cell's link count comparison.

As illustrated, fully known worlds create more border cell links than completely unknown worlds. Both, however, use a decreasing number of links as the density of obstacles increases. The numbers of border cells and links is between those of empty worlds and fully obstacle-filled worlds, as calculated previously.

6.2 Binary-Cost Worlds

An extensive set of simulations were conducted using regular grids and framed quadtrees. The results are available in [15]. As an illustration of the results, Figure 7 shows the difference in the traverse length when framed-quadtrees are used as opposed to regular grids. The path length of framed-quadtrees (right) is shorter and smoother than the one using regular-grids (left). For sparse environments like this one, Framed-Quadtree D* is also faster and uses less memory.

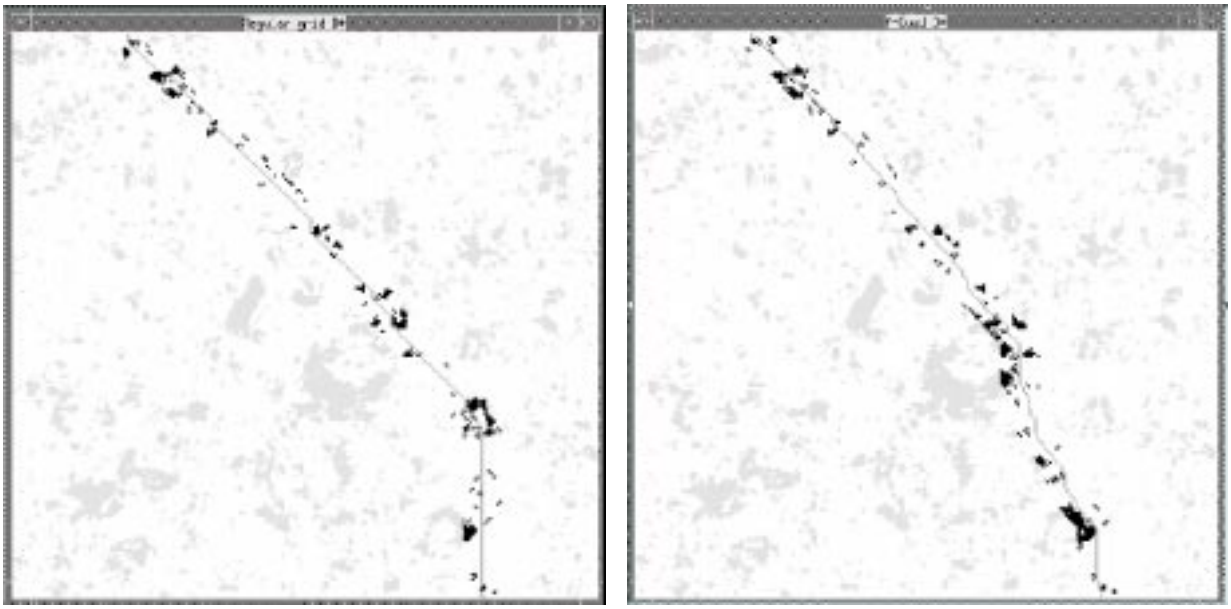
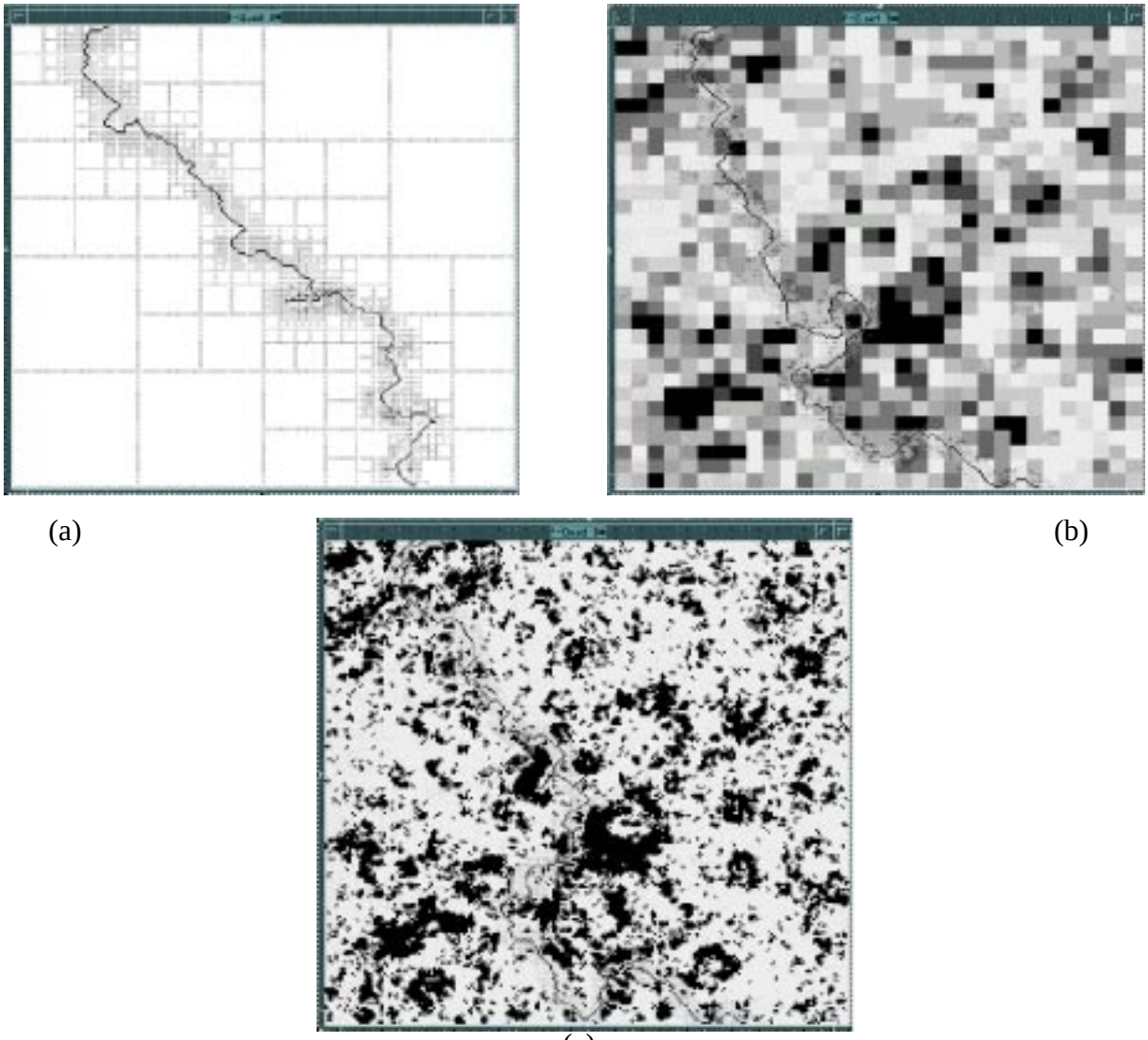


Fig. 7. Traverses generated in a binary fractal world using regular grids (left) and framed quadtrees (right). The lighter cells represent occupied areas that are unknown in advance. The dark cells represent the obstacles that are discovered by the vehicle's sensors. The traverse generated when framed quadtrees are used is shorter and more natural.

6.3 Continuous-Cost Worlds

For continuous cost worlds, we examine three cases: completely unknown, partially known, and fully known continuous cost worlds. In completely unknown worlds, we do not have any map information before the robot starts moving. In contrast, in fully known worlds, we have all map information *a priori*. In some cases, however, the terrain that the robot must traverse is known at a low resolution such as would be produced by an aerial flyover. A low resolution map (coarse map) contains partial information about the world.

For example, Figure 8 shows traverses in a world of which the robot (a) has no knowledge before it starts, (b) has a coarse map, which each cell's cost corresponding to the average of an 8 x 8 area, before it starts, and finally (c) has a complete map *a priori*. Fig. 8a also shows the structure of framed quadtrees generated by the end of the traverse. As can be seen, the path found when a coarse map is available resembles the best possible traverse that can be followed for the fully known world. As we show below, the cost of generating such traverses is less than the cost incurred when a full map is available.



(a) (b)

(c)

Fig. 8. A traverse in (a) a completely unknown continuous-cost fractal world, (b) a partially known (coarse information) continuous-cost fractal world, and (c) a fully known continuous-cost fractal world

We present results of 3000 trials below, comparing traversal cost, memory usage, and execution time. Terrain density is parameterized in ten steps of fractal gain. For each step in terrain density, we conducted 100 runs for each of the three cases (completely unknown, partially known and totally known).

6.3.1 Traversal Cost

Note that traversal cost is not exactly the same as the traversal length. Hence a meandering path might be picked even if a direct path is available, because the direct path happens to have a high cost path. Figure 9 shows traverses in the fully known world have the lowest cost. Conversely, traverses in a completely unknown world have the highest cost because the robot

often enters high cost areas and in some cases goes down dead ends before it backtracks. Traverses in the partially known world are in the between the two curves but it is interesting to note that even with 1/64th as much *a priori* information as in the fully known case, the traversal cost is not significantly higher.

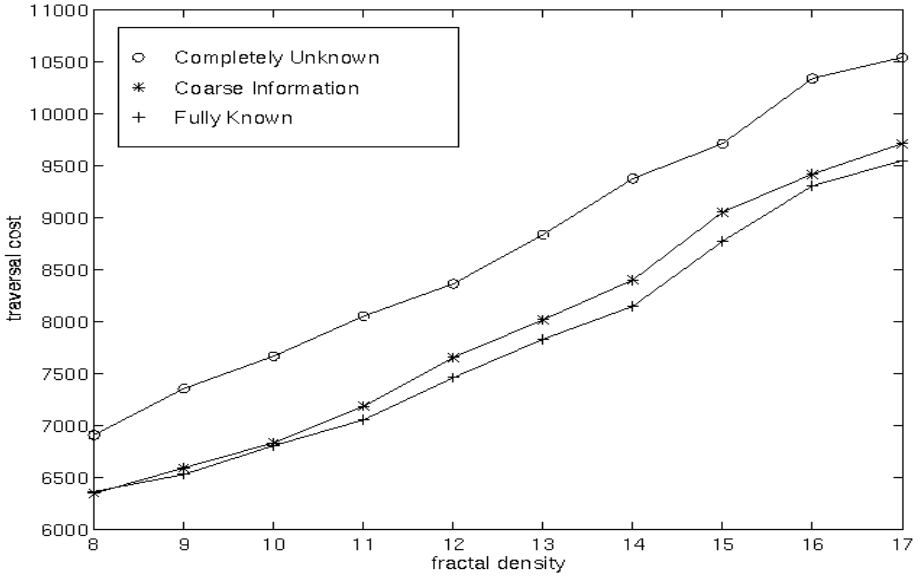


Fig. 9. Traversal cost comparison.

6.3.2 Memory Usage

Figure 10 shows that comparatively encoding the fully known world uses the most memory, while the completely unknown world uses the least. The coarsely known world is in between. As expected, the more information, the more memory is required by a framed-quadtree to represent that information.

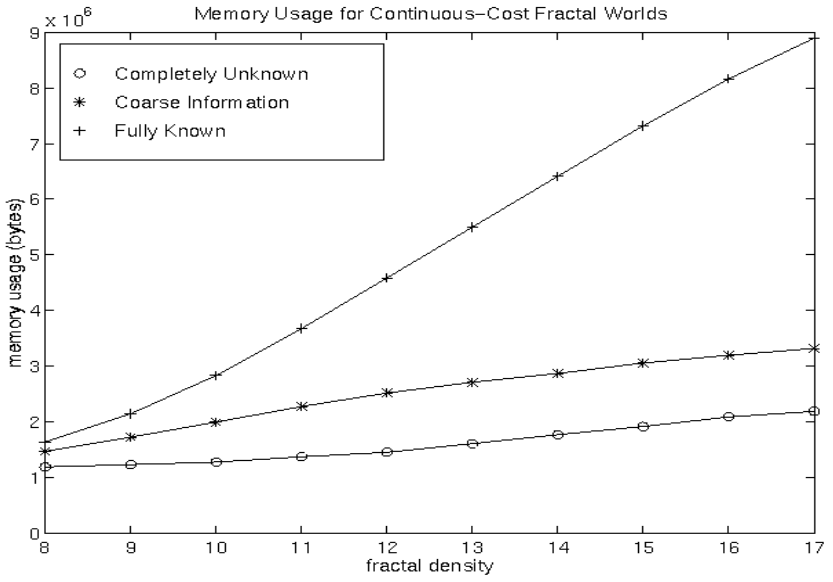


Fig. 10. Memory usage comparison.

6.3.3 Execution Time

Traverses in the fully known world execute faster for sparse worlds, but execute slower for denser worlds than the completely unknown world, due to the time needed to build up framed quadtree structure and to propagate initial D* values. However, traverses in the coarsely known world execute faster than both as fewer data structures are necessary than in the case of the fully known world, and its traverse is less likely to stumble into high cost areas than in the case of the completely unknown world. This is depicted in Figure 11.

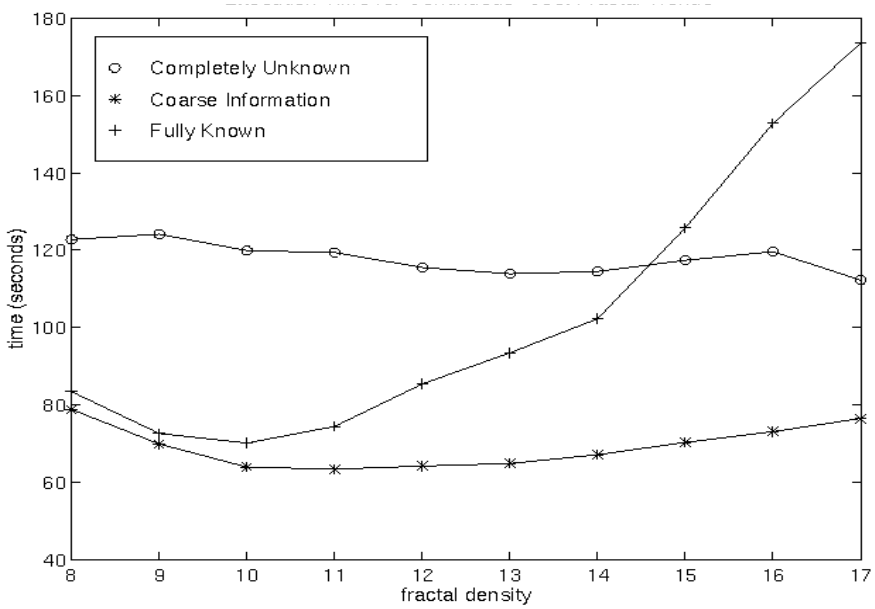


Fig. 11. Execution time comparison.

7.Test Results on an Autonomous Vehicle

Since our system is targeted for implementation on a car-like vehicle, it is necessary to determine realistic commands that can be issued to the vehicle. For example, it is not possible to execute a command that moves the vehicle laterally. At each control step, we hypothesize a small set (approximately 20) of arcs that the vehicle can execute in the next interval. The approach fans out steering arc curves over cells, querying the cells' heuristic costs, adding them up, and selecting the arc that has the lowest total cost. This sum corresponds to the best arc that reduces the traveling cost to the goal. Figure 12 shows that the vehicle should not go straight, as the sum of the heuristic costs of the cells along its straight or nearly-straight arcs, including the three shown in the figure, are prohibitive because they cross an obstacle cell.

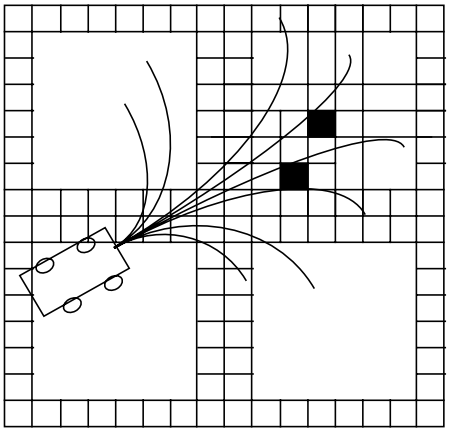


Fig. 12. The optimal arc command to be issued to the vehicle is determined by querying heuristic costs of cells lying along each arc. Black rectangles denote obstacle cells.

We have performed several tests on an automated military jeep (Figure 13). Our vehicle uses a vertical-baseline stereo system to generate range images. The resulting images are processed by the SMARTY local navigator [4], which handles local obstacle detection and avoidance. This obstacle map is fed to a global navigator running a path planning algorithm, such as framed-quadtree D*. Both the local and global navigators submit steering advice to an arbiter, which selects a steering command each time interval and passes it to the controller [13]. Figure 14 shows the system modules and data flow.

Fig. 13. The autonomous vehicle (HMMWV) used for our experiments. The vehicle is equipped with stereo vision, inertial guidance and GPS positioning.

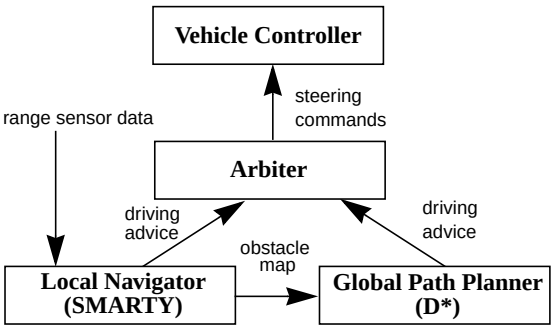


Fig. 14. Data flow in the implemented system.

Figure 15 shows a successful traverse of the vehicle that covered 200 meters in 6 minutes. During this traverse, the vehicle detected and avoided 80 obstacles.

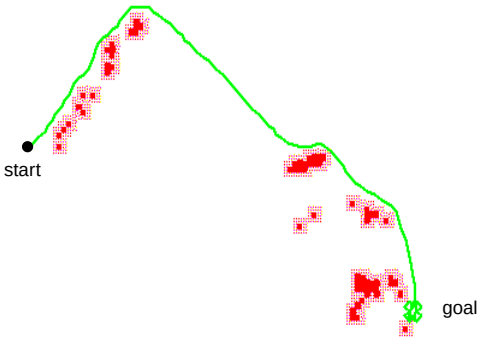


Fig. 15. Successful long traverse of the vehicle using framed-quadtree D* through a terrain with obstacles to the goal. The dark rectangles are obstacles detected and avoided during the traverse. The shaded areas surrounding the dark obstacles are potentially dangerous zones.

Figure 16 shows a close-up of the data structure produced after the above run. As expected, a large part of the environment that is not explored is represented by a small number of cells.

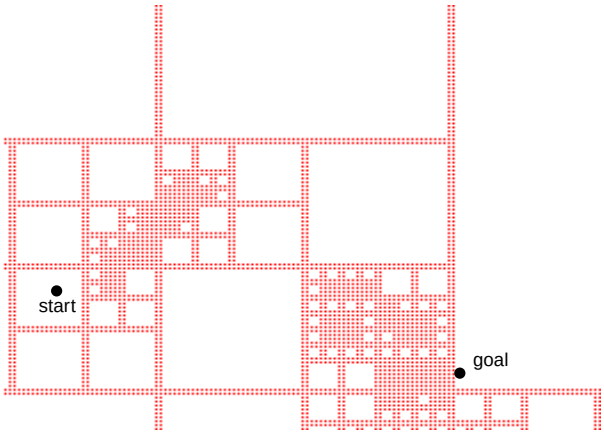


Fig. 16. A close up of the data structure produced from the execution of the path in Fig. 15.

8. Conclusion

Appropriate map representation allows us to do path planning more efficiently in vast unstructured environments than regular-grid representation would allow. Combining the optimal path planning algorithm with framed quadtree map representation has the benefit of optimal path planning while minimizing the memory requirements.

We have extended our method for global path planning suited to autonomous vehicles operating in vast unstructured environments. Our previous method combined the D* algorithm, and a binary framed-quadtree data structure. While a binary world is acceptable for robots operating indoors, it is desirable to use a representation that encodes the three dimensional nature of outdoor terrain. Our extended method can handle continuous-cost maps that are more representative of natural terrains.

The results from extensive simulation in continuous-cost worlds shows that coarse terrain information while not only practical can also result in reduced execution times while incurring only a small cost in optimality over the perfectly known world. In terms of memory usage, there is also a persuasive argument to use coarse information about the world.

As a comparison, note that the memory requirements remain constant, irrespective of the number of the objects in the world when a regular grid is used. Hence this method lends itself to the applications such as future planetary rovers which will have severe limitations in memory, but, nevertheless require relatively high performance.

Several extensions can be made to the system to further enhance its capability and performance:

- Summarizing cells that have been traversed and are far away from the vehicle by collapsing several framed-quadtree nodes will allow further reduction in memory requirements.
- Smart allocation of quadtree nodes and border cells by taking into consideration factors such as the distance of the areas to the vehicle, the importance of the areas, and the reliability of the map information about the areas.
- Extension to handle moving objects. Moving objects, given that their speed is within computational speed bound, can be handled by Framed-Quadtree D* by dividing and summarizing framed quadtree nodes as the obstacle moves along.

References

[1] J. Barraquand, et. al., Numerical potential field techniques for robot path planning, Technical Report STAN-CS-89-1285, Dept. of Computer Science, Stanford University, 1989.

[2] B. Brumitt and A. Stentz, Dynamic mission planning for multiple mobile robots, in: Proc. IEEE International Conference on Robotics and Automation, May 1996.

[3] D. Z. Chen, et. al., Planning conditional shortest paths through an unknown environment: a framed-quadtree approach, in: Proc. 1995 IEEE/RJS International Conference on Intelligent Robots and Systems, IROS 95, Vol. 3, pp. 33-38, August 1995.

[4] M. Hebert, SMARTY: Point-based range processing for autonomous driving, in: Intelligent Unmanned Ground Vehicle, M. Hebert, C. Thorpe, and A. Stentz, eds. (Kluwer Academic Publishers, Dordrecht, 1997).

[5] A. Kelly, An intelligent predictive control approach to the high speed cross country autonomous navigation problem, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, 1995.

[6] T. Lozano-Perez and M.A. Wesley, An algorithm for planning collision-free paths among polyhedral objects, in: Communications ACM 22, 10, October 1979.

[7] J. Lengyel, et. al., Real time robot motion planning using rasterizing computer graphics hardware, in: Proc. SIGGRAPH. 1990.

[8] S. Russell and P. Norvig, Artificial intelligence: a modern approach (Prentice Hall, Singapore, 1995).

[9] H. Samet, Neighbor finding techniques for images represented by quadtrees, Computer Graphics and Image Processing 18 (1982), 37-57.

[10] H. Samet, An overview of quadtrees, octrees, and related hierarchical data structures, NATO ASI Series, Vol. F40 (1988), 51-68.

[11] A. Stentz, The Focussed D* algorithm for real-time replanning, in: Proc. the International Joint Conference on Artificial Intelligence, August 1995.

[12] A. Stentz, Optimal and efficient path planning for partially-known environments, in: Proc. IEEE International Conference on Robotics and Automation, May 1994.

[13] A. Stentz and M. Hebert, A complete navigation system for goal acquisition in unknown environments, Autonomous Robots 2(2) (1995).

[14] A. Stentz, Best Information Planning for unknown, uncertain, and changing domains, in: AAAI-97 Workshop on On-line-Search.

[15] A. Yahja, A. Stentz, S. Singh and B. Brumitt, Framed-quadtree path planning for mobile robots operating in sparse environments, in: Proc. IEEE Conference on Robotics and Automation, Leuven, Belgium, May 1998.