A Beamlet-Based Graph Structure for Path Planning Using Multiscale Information

Yibiao Lu, Xiaoming Huo, *Senior Member, IEEE*, and Panagiotis Tsiotras, *Senior Member, IEEE*

***Abstract*—Path-planning problems are fundamental in many applications, such as transportation, artificial intelligence, control of autonomous vehicles, and many more. In this paper, we consider the deterministic path-planning problem, equivalently, the single-pair shortest path problem on a given grid-like graph structure. Current commonly used algorithms in this area include the A* algorithm, Dijkstra's algorithm, and their numerous variants. We propose an innovative beamlet-based graph structure for path planning that utilizes multiscale information of the environment. This information is collected via a bottom-up fusion algorithm. This new graph structure goes beyond "nearest-neighbor" connectivity, incorporating "long-distance" interactions between the nodes of the graph. Based on this new graph structure, we obtain a multiscale version of A*, which is advantageous when preprocessing is allowable and feasible. Compared to the benchmark A* algorithm, the use of multiscale information leads to an improvement in terms of computational complexity. Numerical experiments indicate an even more favorable behavior than the one predicted by the theoretical complexity analysis.**

***Index Terms*—A*, beamlet-like structure, bottom-up fusion algorithm, Dijkstra's algorithm, dynamic programming, path-planning.**

I. Introduction

A new beamlet-based graph structure is proposed for efficient path-planning within an environment full of obstacles. The main idea is to use multi-scale information, by utilizing techniques similar to those used in [1] for the purpose of statistical image processing. The theoretical analysis shows that the proposed multiscale version of the well-known A* algorithm based on this new graph structure has lower worst-case complexity than the standard A* algorithm applied to the nearest-neighbor graph. In numerical experiments, we found that the proposed multiscale structure significantly reduced the number of node expansions in several large-scale environment scenarios.

Manuscript received January 30, 2010; revised November 23, 2010 and May 11, 2011; accepted September 16, 2011. Date of publication March 23, 2012; date of current version April 19, 2012. This work was supported by the NSF through award CMMI-0856565. Recommended by Associate Editor F. Dabbene.

Y. Lu and X. Huo are with the H. Milton Stewart School of Industrial and System Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA (e-mail: ylv3@gatech.edu; huo@gatech.edu).

P. Tsiotras is with the Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA (e-mail: tsiotras@gatech.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2012.2191836

The main idea of the proposed multiscale graph structure can be summarized as follows. Consider a uniform n by n grid representing the world (or an image) assuming, without loss of generality, 4-nearest-neighbor connectivity. There are $O(n^2)$ vertices and $O(n^2)$ edges in the corresponding graph. In order to reduce the number of node expansions (the most time-consuming step in all graph search algorithms), the proposed graph structure first employs a recursive dyadic partitioning (to be defined later) to divide the environment into "blocks" of different sizes. The block sizes are determined by the relative importance of information within those blocks. The collection of all blocks of the same size defines the *information scale*. The preprocessing of information within each block is conducted via an innovative *Bottom-Up Fusion* algorithm, which "fuses" multiscale information from finer scales to coarser scales. Therefore, a properly designed search algorithm, defined on the preprocessed "blocks," can significantly reduce the number of vertices in the graph, while only slightly increasing the number of edges. As a result, path searching can be sped-up significantly, when preprocessing is feasible.

There are four major ingredients in the proposed multiscale approach, briefly summarized below (a full description is given in Section III). (a) A recursive dyadic partitioning that divides the entire grid world into hierarchically organized *d-squares*. These d-squares are of different size, depending on the scale. Only a subset of these d-squares is used during path-planning. The concept of *Path-Finding Reduced Recursive Dyadic Partition* (PFR-RDP) is introduced to describe the collection of d-squares used for path-planning. It can be shown that the PFR-RDP contains at most $O(\log n)$ d-squares. (b) For each d-square, all free boundary cells (i.e., vertices) are connected by edges, with the edge weights being equal to the lengths of the corresponding shortest paths. (c) A fusion algorithm, which efficiently computes the weights in (b) using the recursive relationship between the dyadic squares across different scales. A new graph (based on the PFR-RDP and the weights being computed in (c)) is thus obtained. We call this new graph the *beamlet graph*, owing to its similarity with the data structure introduced in [1] to encode efficiently all linear features in an image.[1] (d) Finally, the A* algorithm (or any other similar graph search algorithm) is run on the beamlet graph to identify the optimal path.

As it will be shown in the sequel, the proposed beamlet graph has $O(n)$ vertices and $O(n^2)$ edges. The worst-case complexity

[1]Note that the beamlet graph defined in the current paper is different than the beamlet graph introduced in [1]; we have still decided to use the term *beamlet graph*, because both take advantage of "long-distance" neighboring relations between the nodes of the graph. This slight abuse of terminology should not cause a confusion—the two names attach to different problems.

of running A* or Dijkstra's algorithm on the original 4-nearest-neighbor graph is $O(n^2 \log n)$, assuming a Fibonacci heap is used. The complexity on the newly designed beamlet graph is $O(n^2)$. The reduction by a factor of $\log n$ initially may not seem impressive; however, our numerical simulation results proved to be much more encouraging. In some cases, our approach demonstrates an increase in speed of the A* algorithm by one or two orders of magnitude. The numerical experiments show that by combining the new beamlet-based graph structure with the A* algorithm yields faster query time, adequate usage of memory and reasonable preprocessing time.

The major contributions of our work are summarized below. First, we introduce a new, beamlet-based graph structure, which uses the multiscale information from the environment in order to reduce the complexity of benchmark path-finding algorithms. Although tested extensively against the standard A* and Dijkstra algorithms, the proposed data structure is not tied to a specific graph search algorithm, and it can be easily incorporated into other existing approaches, such as A* with stronger heuristics (i.e., true distance heuristic, differential heuristic, etc [2], [3]), bidirectional search algorithms, etc [4]–[6]. Second, we provide a systematic approach, the *bottom-up fusion* algorithm, that allows us to "fuse" local information from finer scales into global information at coarser scales. This algorithm is of more general interest, as it can be employed whenever a multi-scale partition of the environment is available.

The rest of this paper is organized as follows. Section II gives a description of the problem formulation of the path-planning problem. Section III provides the details of the proposed beamlet-based graph structure, which contains the dyadic partition tree construction, and the bottom-up fusion information collection method. The proposed multiscale version of A* (m-A*) applies A* on the resulting beamlet graph. The order of complexity of the m-A* is analyzed in Section IV. Section V compares m-A* with the benchmark A* algorithm via numerical experiments under different scenarios. Since the performance of A* strongly depends on the heuristic used, we also compare m-A* and A* in terms of a much stronger heuristic than the standard L_1 distance, namely, the true distance heuristic (TDH) [2], [3]. As shown in Section V-B, m-A* outperforms A* even when a much stronger heuristic is used; furthermore, the performance gap increases with the size of the problem data. Section VI offers a brief overview of the theory behind multiscale beamlet analysis, along with a comparison of our approach with related work from the path-planning literature. Section VII summarizes the results of this paper and provides some suggestions for possible future extensions.

II. Problem Formulation

A deterministic path-planning problem consists of a graph $G = (V, E)$, where V is the set of vertices (i.e., the possible vehicle locations) and E is the set of edges, representing transitions between these vertices. The weight of each edge represents the cost of transitioning between the two corresponding vertex (viz. node) locations. Planning a path from an initial vertex to an end vertex can be cast as a single-pair, shortest path problem on this graph. In the deterministic (respectively, dynamic) path-planning problem, the environment does not (respectively, does) change over time.

We consider path-planning problems in a deterministic 2-D environment. Without loss of generality, we assume that the information about the environment is given via an n by n square image, where n is dyadic: $n = 2^J$ and J is a positive integer. Note that such an image-based formulation is well-adopted in the path-planning literature. Often, the image is called the *grid-world* [7]. The proposed method does not require the image to be square. However, assuming a squared image simplifies our algorithmic description. Therefore, henceforth, we will assume squared images. We will also assume that the image contains two types of pixels: black pixels (representing non-traversable *obstacles*) and white pixels (representing traversable *free cells*). The path-planning problem is to find the shortest path between a given pair of *source* and *destination* pixels.

Two popular shortest-path search algorithms in a deterministic setting are Dijkstra's algorithm [8] and the A* algorithm [9]. Both algorithms give the optimal path, and can be considered as special implementations of dynamic programming [10]. A* operates essentially the same way as Dijkstra's algorithm, except for the fact that it uses a heuristic estimate to guide the search towards the most promising states. The use of heuristics in A* potentially reduces the computational time.

To apply either search algorithm, one needs to first construct the search graph. In this graph each free cell is defined to be a vertex (viz. node), and, correspondingly, it is connected only to its free four nearest-neighbors (*four-nearest-neighbor connectivity assumption*). However, both Dijkstra's and A* algorithms have the tendency to be slow as the space to be searched increases. Below we propose a *beamlet-based graph structure* that takes advantage of the sparse information induced by the quadtree decomposition in a hierarchy of dyadic squares, thus improving the performance of the standard A* algorithm when it is applied on the beamlet graph. As it will be shown in Section IV, such a strategy can reduce significantly the order of computational complexity, in the worst-case. It should be pointed out that although four-nearest-neighbor connectivity is assumed throughout the paper for simplicity, our theorems (supported by the numerical experiments) show that the same results also hold were eight-nearest neighbor connectivity had been used, instead.

The main objective of the proposed new graph structure is to construct a smaller size graph on which the computational complexity of searching for the shortest path is efficiently reduced. The intuition for the ensuing problem size reduction is given as follows. The direct implementation of Dijkstra's or A* algorithm searches through all free cells in the environment. This can be overwhelmingly redundant: if, for instance, the origin and destination vertices are in the upper-left and bottom-right quadrants, respectively, it is not necessary to scan through all the free vertices in the upper-right and bottom-left quadrants. Instead, one only needs to consider the boundary white pixels of these two quadrants. An illustration of this idea can be seen in Fig. 1. Armed with this intuition, in the sequel we develop a new dynamic programming algorithm for path planning in a cluttered environment, which takes advantage of preprocessed information organized in a multiscale fashion, analogous to the

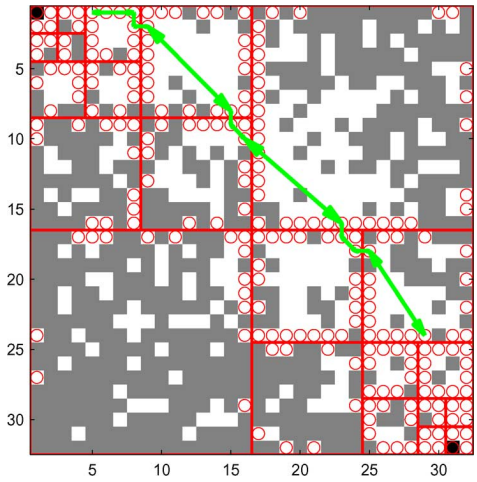


Fig. 1. Free boundary cells attached to each dyadic square are the only ones that need to be considered in the proposed approach. Note that not all free boundary cells need to be expanded when running Dijkstra's or the A* algorithm on the beamlet graph. This figure also illustrates the vertices in the beamlet graph, defined in Section III-B.

quadratic tree structure that has been used in beamlet analysis [1]. The algorithm is explained in detail in the following section.

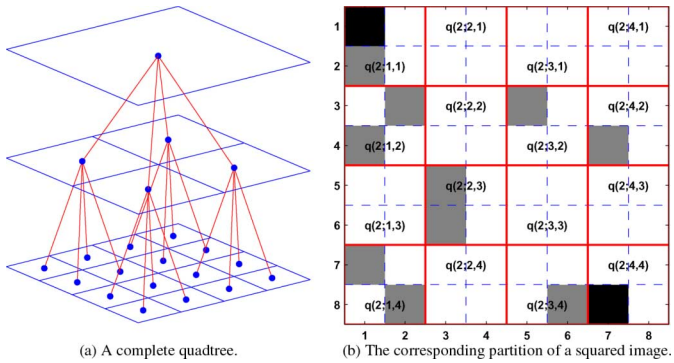
III. Multiscale Path Planning Strategy With Preprocessed Information

Each free cell is a vertex in the nearest neighbor graph and is connected to the free cells among its four nearest-neighbors. Each edge has unit weight. We describe our approach in four steps. In Section III-A, we describe the *recursive dyadic partition* and the *path-finding reduced recursive dyadic partition*. These serve as the starting points of our approach. We then describe a new type of connectivity (Section III-B), motivated by the beamlet structure introduced in [11]–[13]. We call the new data structure the *beamlet graph*, owing to its similarity with the connectivity relations arising in the beamlet graph structure of [12]. To compute the edge weights needed to create the beamlet graph, a bottom-up fusion algorithm is given in Section III-C. The proposed multiscale A* algorithm, essentially runs A* (or Dijkstra's) algorithm on the aforementioned beamlet graph. This is explained in detail in Section III-D.

A. Recursive Dyadic Partitioning of the Environment

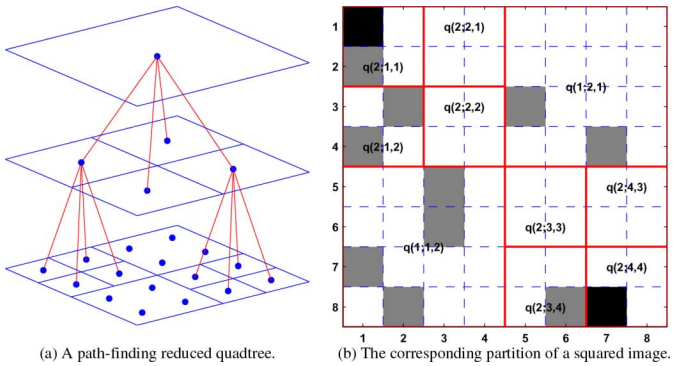
We describe two types of recursive dyadic partitioning (RDP). The first one is the complete version of RDP. We then introduce the *Path-Finding Reduced* RDP (PFR-RDP), which will play an important role in defining the beamlet graph—the graph structure that we will rely on.

The complete recursive dyadic partition can be described in a top-down approach: a squared image is subdivided into smaller d-squares, repeating the partitioning until the finest resolution of the image is reached. Let s $(1 \leq s \leq J)$ denote the *scale*. A dyadic square (referred to as a d-*square* from this point on) at scale s, indexed by a, b $(1 \leq a, b \leq 2^s)$ will be denoted by $q(s; a, b)$. We have $q(s; a, b) = \{(i, j) : 2^{J-s}(a-1)+1 \leq i \leq 2^{J-s}a, 2^{J-s}(b-1)+1 \leq j \leq 2^{J-s}b\}$. The d-square $q(s; a, b)$ at scale s can be partitioned into four d-squares at scale $s+1$; i.e., we have $q(s; a, b) = q(s+1; 2a-1, 2b-1) \cup q(s+1; 2a-1, 2b) \cup q(s+1; 2a, 2b-1) \cup q(s+1; 2a, 2b)$. Accordingly, we say that the d-square $q(s; a, b)$ has four children. The family of d-squares at all scales forms a quadtree. The correspondence between the recursive dyadic partitioning and the quadtree is illustrated in Fig. 2.



(a) A complete quadtree. (b) The corresponding partition of a squared image.

Fig. 2. (a) A complete recursive dyadic partition with the corresponding quadtree; (b) Complete recursive dyadic partition on a simple 8 × 8 image. The black cells are the source and destination and the gray cells are obstacles. The d-squares shown in the figure all come from the third (bottom) layer of partition in (a). (a) A complete quadtree. (b) The corresponding partition of a squared image.



(a) A path-finding reduced quadtree. (b) The corresponding partition of a squared image.

Fig. 3. (a) A partial recursive dyadic partition and the corresponding PFR-RDP; (b) A partial recursive dyadic partition on a simple 8 × 8 image. The black cells denote the source and destination. The gray cells are obstacles. The d-squares shown in figure are from the third (bottom) layer in (a). (a) A path-finding reduced quadtree. (b) The corresponding partition of a squared image.

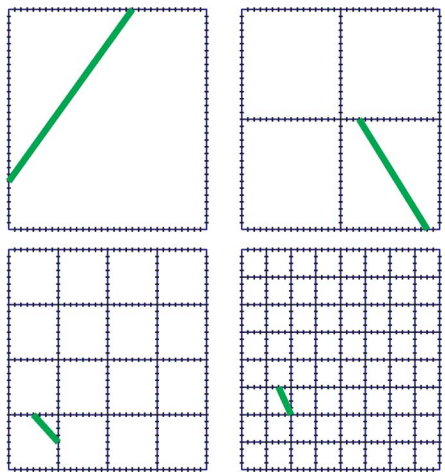


Fig. 4. Illustrations of the original beamlets. The successive subdivision of the sides of the squares provides a hierarchical data structure that efficiently encodes the distance between any two points at the boundaries of the squares.

In the path-planning problem, for a given pair of start and destination cells, only part of the complete RDP is needed. This is the Path-Finding Reduced RDP (PFR-RDP). The PFR-RDP is generated as follows. The image is initially subdivided into four equal, smaller d-squares. If either the source or the destination lies in a smaller d-square, the dyadic subdivision of this d-square will continue, unless the finest resolution has been reached. If a d-square contains neither the origin nor the destination, no further partitioning is done to this d-square. The resulting partition is a partial recursive dyadic partition—not all d-squares are partitioned to the finest resolution—and corresponds to a partial quadtree. Fig. 3 shows an example of a PFR-RDP along with the corresponding PFR-quadtree. The pseudo-code for the PFR-RDP is given in Algorithm 1.

Algorithm 1 PFR-RDP (Path-finding reduced recursive dyadic partitioning)

1: Set the largest scale to $J = \log_2 n$, where the image size is n by n.

2: Initialize the list $dptree = [1, 1, 1]$—the d-square at the coarsest level.

3: **for** $s = 1 : J - 1$ **do**

4: For d-square at scale s in $dptree$

5: **if** v_s (source) or v_e (destination) is in this d-square

6: In $dptree$, remove the line corresponding to this d-square;

7: Partition into four equal, smaller d-squares, and insert them as new lines in $dptree$

8: **end if**

9: **end for**

$dptree$

B. Beamlet-Like Connectivity

Recall that in the nearest neighbor graph, only the four nearest neighbors of a cell are connected by edges. Here we introduce another type of connectivity that takes advantage of connections between faraway cells. We will see that such a connectivity, together with the aforementioned PFR-RDP, can reduce the computational complexity of the search algorithm. For each fixed d-square, we only consider the free cells on its boundary. A pair of free cells on the boundary of the d-square are said to be connected by an edge if and only if there exists a feasible path between the two within this d-square. Note that such a definition is similar to the concept of beamlets introduced in [12]. We refer to Section VI-A for the background theory on beamlet analysis. Representative original beamlets in [12] are displayed in Fig. 4. Fig. 5 shows several "beamlets" in the context of the current paper attached to a 8×8 dyadic square. The green lines[2]

[2]Please refer to the electronic version of the paper for the color versions of the figures.

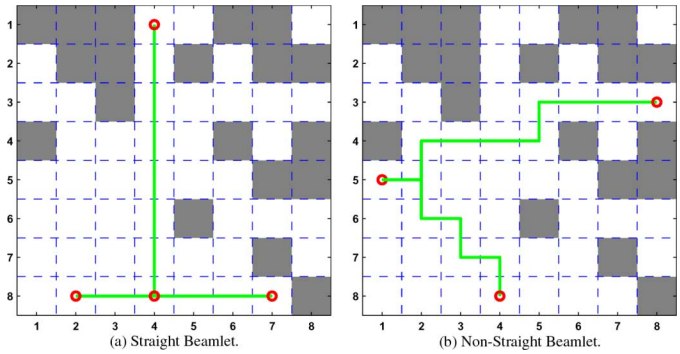


Fig. 5. (a) Three straight beamlets in a 8×8 image; the red circles are the end points of each beamlet; (b) Two beamlets connecting the cells (1,5)–(4,8) and (1,5)–(8,3); note that beamlets in the shortest-path problem may not be straight lines. (a) Straight beamlet. (b) Non-straight beamlet.

show the corresponding shortest paths (i.e., the optimal beamlets) between two pairs of boundary free cells. Notice that a beamlet in the shortest-path problem may not be a straight line (Fig. 5(b)). The beamlet graph is now defined as follows. First, the PFR-RDP of the given n by n image is obtained. All the free cells on the boundaries of each d-square in the PFR-RDP are defined to be the vertices in the beamlet graph. Two vertices in the beamlet graph are connected under two conditions: (a) they are nearest neighbors; or (b) they belong to the same d-square in the PFR-RDP and there is a beamlet (i.e., a feasible path inside the d-square) connecting them. Within each d-square, the weight of an edge is the length of the shortest path connecting the two vertices. An example of a PFR-RDP partition is shown in Fig. 1. In Fig. 1 the red grid shows the partial dyadic partition corresponding to the PFR-RDP. The red circles are the free boundary cells, i.e., the vertices in the beamlet graph.

To compute the weights of the edges within all d-squares in a PFR-RDP we make use of the following bottom-up fusion algorithm.

C. Bottom-Up Fusion Algorithm

The proposed multiscale path planning strategy requires the availability of the shortest path distances between any pair of free boundary cells for each d-square. When $s = J$ or $J - 1$, there are one or four pixels in the d-square, respectively. Hence, it is straightforward to compute these distances. For the general case, recall that a d-square $q(s; a, b)$ can be partitioned into four smaller d-squares at scale $s + 1$. If we already know the inter-distances between the free boundary cells within each of the smaller d-squares, and by considering the connectivity of the free boundary cells that belong to neighboring d-squares, we can treat all free boundary cells of the four d-squares at scale $s + 1$ as vertices in a "fused" graph, and run Johnson's algorithm [14] to compute all shortest paths. The distances between free cells from neighboring d-squares can be computed directly. Since there are no more than $n2^{2-s}$ of these cells, the search algorithm can be run efficiently. Fig. 6 shows how this "fusion" of shortest distances is conducted recursively within an 8×8 d-square. During the first step, the inter-distances between the free boundary cells of the four d-squares on the finer layer are computed using Johnson's algorithm. The green arrows show some of these distances. The dashed lines in the second layer,

corresponding to the solid grid partition, show the fusion of the inter-distances. The dashed lines at the center of Fig. 6 indicate the final step of the fusion algorithm which yields the inter-distances between the pair of black cells. The pseudo-code for this bottom-up fusion algorithm is provided in Algorithm 2.

Algorithm 2 BottomUpFusion (For each d-square)

1: Read the parameters of each d-square: s (scale), a, b (location);

2: **if** $s = \log n - 1$ **then**

3: Compute the free boundary cells as vertices (Trivial case: only four cells in the d-square)

4: Calculate the four nearest neighbor connectivity (edges) within each d-square

5: Run Johnson's algorithm on the resulting graph to obtain all pairs of shortest paths: $cgraph$ and $pathList$.

6: **end if**

7: **if** $s > 1$ **then**

$$graph1, path1 = \text{BottomUpFusion}(s+1, 2a-1, 2b-1)$$
$$graph2, path2 = \text{BottomUpFusion}(s+1, 2a, 2b-1)$$
$$graph3, path3 = \text{BottomUpFusion}(s+1, 2a-1, 2b)$$
$$graph4, path4 = \text{BottomUpFusion}(s+1, 2a, 2b)$$

12: Merge $graph1, \ldots, graph4$ into $Graph$ by adding the connected edges between neighboring d-squares

13: Run Johnson's algorithm on $Graph$ and get $cgraph$ and $tmpPathList$

14: Insert the missing parts of paths in $tmpPathList$ from $path1, \ldots, path4$ to obtain $pathList$

15: **return** $cgraph, pathlist$ (i.e., the beamlet graph)

16: **end if**

Overall, we have a bottom-up fusion algorithm that computes the inter-distances between the free boundary cells of all d-squares. In other words, we have found the shortest paths between all boundary free cells. These are exactly the edge weights in the beamlet graph.

D. Multiscale A* Algorithm on the Beamlet Graph

In the previous section, we described how we can obtain a beamlet-based graph structure. From this point on, we denote the beamlet graph as $BG = (V, E)$, where V denotes the vertices, i.e., the free boundary cells of all the d-squares in the PFR-RDP, and E denotes the edges representing the shortest distance paths between pairs of free boundary cells.

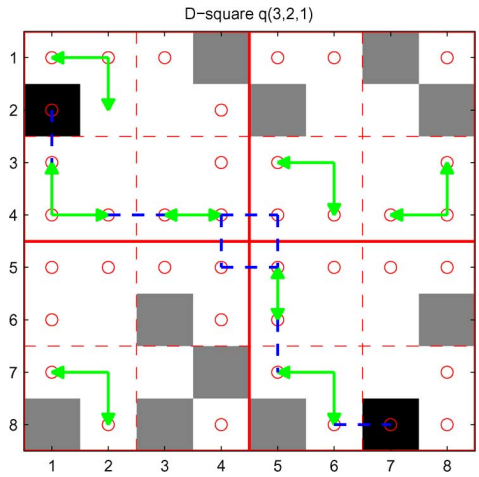


Fig. 6. Bottom-up fusion in the d-square q(3,2,1) of Fig. 1. The information fusion is conducted in a complete dyadic partition. Notice that the solid red grid stands for the partition corresponding to the second layer of the associated quadtree, and the dashed red grid corresponds to the partition with respect to the third layer. The green arrows indicate the inter-distances between free boundary cells. The blue lines show the fusion process.

By using this new graph structure, we implement a multiscale version of the A* algorithm, henceforth called m-A* for short. The main steps of m-A* mimic the standard A* algorithm. For convenience, we repeat the main steps of the algorithm below. m-A* plans a path from the source vertex $v_s \in V$ to the destination vertex $v_e \in V$. To do this, the algorithm stores an estimate $g(v)$ of the path length from v_s to each vertex v. The algorithm also keeps an estimate of the path length from v to the destination denoted by $f(v) = g(v) + h(v, v_e)$. Initially, we set $g(v) = \infty$ for all vertices in V. The algorithm begins by setting $g(v_s) = 0$ and then places this vertex in a priority queue, known as the *OPEN* list. Each element v in this queue is ordered according to its f-value, that is, the sum of its current path length from v_s, stored in $g(v)$, and a heuristic estimate of the path length to the destination, $h(v, v_e)$. The vertex having the minimum f-value is pushed to the front of the priority queue. To be admissible, the heuristic $h(v, v_e)$ should underestimate the cost of the optimal path from v to v_e [15]. In our implementation, the heuristic estimate used was the usual L_1 distance. It should be emphasized, however, that this choice can be made without loss of generality. Other stronger heuristics and/or the use of a bi-directional search could have been used, with the results essentially remaining the same (see Section V-B).

The algorithm pops the vertex v at the front of the queue and updates the g-values of all vertices in V representing obstacle-free cells that are reachable from this vertex through a direct edge (denoted as $Succ(v)$): if the value of $g(v)$ plus the weight of the edge between v and its neighboring cell v' in the beamlet graph (denoted as $c(v, v')$) is less than the current g-value of the vertex v', then the g-value of v' is set to this new, lower value. If the g-value of a neighboring vertex v' changes, it is placed in the *OPEN* list. The algorithm continues exploring all vertices in the queue until it arrives at the destination vertex. At this stage, if the heuristic is admissible, then the path length from v_s to v_e is guaranteed to be optimal. The complete algorithm is given in Algorithm 3.

Algorithm 3 Multiscale A*

1: Initialize img, v_s, v_e

2: Conduct $PFR-RDP$ and obtain $dptree$

3: Run the *Bottom-Up Fusion* algorithm on each d-square in $dptree$ and get beamlet graph

4: **for** all $v \in V$ **do**

5: $g(v) = \infty$

6: **end for**

7: $h(v_s, v_e) = heuristicEstimate(v_s, v_e)$

8: $g(v_s) = 0$; $f(v_s) = h(v_s, v_e)$

9: $CLOSE := \emptyset$

10: $OPEN := v_s$ with value $g(v_s) + h(v_s, v_e)$

11: **while** $OPEN \neq \emptyset$ **do**

12: $v = \arg\min_{u \in OPEN}(g(u) + h(u, v_e))$

13: remove state v from $OPEN$, add it into $CLOSE$

14: **if** $v = v_e$ **then**

15: **return** construct optimal path

16: **end if**

17: **for** all $v' \in Succ(v)$

18: **if** $v' \in CLOSE$ **then**

19: continue

20: **end if**

21: **if** $v' \in OPEN$ **then**

22: **if** $g(v') > g(v) + c(v, v')$ **then**

23: $g(v') = g(v) + g(v, v')$

24: update v' with value $g(v') + h(v', v_e)$

25: **end if**

26: **else**

27: insert v' into $OPEN$ with value $f(v') = g(v') + h(v', v_e)$

28: **end if**

29: **end for**

30: **end while**

31: **return** failure

IV. Complexity Analysis

In this section we provide a detailed analysis for the computational complexity of m-A*. First, we discuss the bottom-up fusion part, and then move on to the overall complexity of m-A*.

A. Complexity of Information Fusion Part

We derive an upper bound for the algorithmic complexity of the bottom-up fusion algorithm, i.e., we consider the worst case. Let $T(m)$ denote the amount of computations required to find the inter-distances between all pairs of free boundary cells within an $m \times m$ d-square. A recursive relationship dividing a cell of dimensions $2m \times 2m$ to four equal $m \times m$ squares can be written as follows:

$$T(2m) = 4T(m) + f(2m) \tag{1}$$

where $f(2m)$ denotes the effort for solving the all-pair shortest path problem during the information fusion step. Fig. 7 shows an example of the fusion step within a d-square under the worst case scenario (no obstacles). Both connections of inter-distances between the same d-square (represented in (1) by $T(m)$), and between neighboring d-squares (represented in (1) by $f(2m)$) are shown in this figure.

A key step during the fusion process is the solution of the all-pair shortest path problem on the given graph. For the graph shown in Fig. 7, for instance, one needs to consider all boundary pixels of the smaller d-squares as vertices in the beamlet graph. Also note that there are at most $4(m-1)$ boundary pixels per smaller d-square. It follows that $|V| \leq 4 \times 4(m-1)$. The edges in the beamlet graph belong into two categories:

(a) Edges connecting the free boundary cells within each smaller d-square.

(b) Edges connecting the nearest-neighbor pixels between neighboring d-squares.

The green arrows in Fig. 7 show some of these edges. In each d-square, there are at most $\binom{4m-4}{2}$ edges; there are at most $4m$ edges connecting free cells that are nearest neighbors not in the same d-square. Hence, we have $|E| \leq 4\binom{4m-4}{2} + 4m < 32m^2$.

Lemma 1: For the bottom-up fusion algorithm, we have $f(m) = O(m^3)$, where $f(m)$ as in (1).

Proof: Johnson's Algorithm [14] is the standard algorithm for solving the all-pair shortest path problem. If one assumes that an intermediate step of Johnson's algorithm (an implementation of the Dijkstra's algorithm) is done via Fibonacci heap, then the overall complexity is $O(|V|^2 \log(|V|) + |V||E|) = O(m^2 \log(m) + m^3) = O(m^3)$. This is exactly the complexity of $f(m)$.

To evaluate $T(m)$, we will need the following Master Theorem [16, Sect. 4.3].

Theorem 2 (Master Theorem): Consider the recurrent relation of an algorithm of the form

$$T(m) = aT\left(\frac{m}{b}\right) + f(m), \quad a \geq 1, b > 1$$

where m is the size of the entire problem, a is the number of subproblems in the recursion, m/b is the size of each subproblem (it is assumed that all subproblems are of the same size). Let $f(m)$ be the cost of the work done outside the recursive calls, which includes the cost of dividing the problem, and the cost of merging the solutions to these subproblems.

Suppose the following two conditions are satisfied:

(a) $f(m) = O(m^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and

(b) $af(m/b) \leq cf(m)$ for some constant $c < 1$ and sufficiently large m.

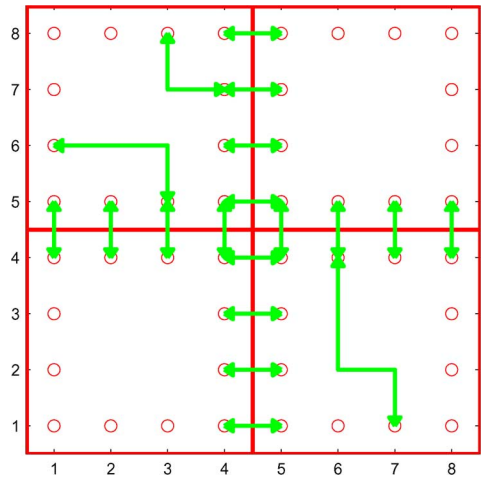


Fig. 7. Illustration for the complexity analysis of the fusion algorithm. The three right-angle arrows provide examples of inter-distances between free boundary cells within two smaller scale d-squares. Along the common boundary of neighboring smaller scale d-squares, the straight arrow lines show the fusion of the inter-distances among these four d-squares. Note that for the purposes of complexity analysis, this figure illustrates the worst case scenario (i.e., no obstacles exist and hence all boundary cells at both levels are free).

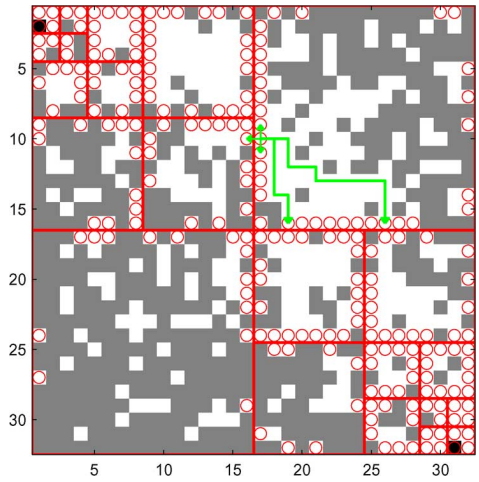


Fig. 8. Illustration of the beamlet graph for a 32×32 image. The PFR-RDP is shown by the red grid lines. Free cells are marked by circles: these are the free boundary cells of the d-squares in the PFR-RDP. If two free cells belonging to different d-squares are nearest neighbors, they are also connected. All free cells within the same d-square are considered connected, as long as a feasible path exists.

Then $T(m) = O(f(m))$.

As a direct consequence of Theorem 2 we therefore have the following result.

Corollary 3: For the $T(m)$ in (1), we have $T(m) = O(f(m)) = O(m^3)$.

Proof: Take $a = 4$ and $b = 2$, and apply Theorem 2.

B. Complexity of Searching

We now consider the order of complexity for running A* (or Dijkstra's) algorithm on the beamlet graph (BG), and the 4-nearest-neighbor graph (NNG), respectively. We consider the beamlet graph first. In order to better illustrate the idea, consider the PFR-RDP and the corresponding beamlet graph for the scenario shown in Fig. 8. Recall that the vertices in the beamlet graph are the free boundary cells in all the d-squares in the PFR-RDP. For an n by n image, there are two scale-1 d-squares, and six scale-s d-squares when $s \geq 2$. For a d-square at scale s, there are at most $n2^{2-s}$ free boundary pixels. Hence, the upper bound of the total number of vertices in the beamlet graph is: $2 \times 2n + 6 \times n + 6 \times (n/2) + 6 \times (n/4) + \cdots = 4n + 6n + 3n + (3/2)n + \cdots \leq 16n$. On the other hand, within a d-square at scale s, the number of the edges is at most $\binom{n2^{2-s}}{2}$. At scale s, the number of connected free cells belonging to different d-squares is at most $n2^{2-s}$. Hence, the upper bound of the number of edges is $2\binom{2n}{2} + 2n + 6\binom{n}{2} + 2 \times (n/2) + 6\binom{n/2}{2} + 2 \times (n/4) = 4n^2 + 2n + 3n^2 + n + (3/4)n^2 + \ldots \approx 8n^2$. Recall that the complexity of running Dijkstra's algorithm with Fibonacci heap is $O(|E| + |V| \log |V|)$ [16]. We therefore have the following theorem.

Theorem 4: The complexity of running Dijkstra's algorithm on the beamlet graph is $O(n^2)$.

Proof: The previous calculation gives the following estimates for the number of vertices and edges in the beamlet graph: $|V| = 16n$ and $|E| \approx 8n^2$, respectively. Using the known complexity bounds of Dijkstra's algorithm [8], the upper bound of the complexity is $O(n^2 + n \log n) = O(n^2)$.

Since A* and Dijkstra's algorithm have identical worst-case complexity [15], the following is immediate.

Corollary 5: The complexity of running A* algorithm on the beamlet graph is $O(n^2)$.

For comparison, in the nearest-neighbor graph we have $|V| = n^2$ and $|E| = 4n^2$. One can thus easily establish the following result.

Theorem 6: If A* or the Dijkstra's algorithm is run on the nearest-neighbor graph, the worst-case complexity is $O(4n^2 + n^2 \log n^2) = O(n^2 \log n)$.

The proof of the theorem is evident and is therefore skipped. Note, from Theorem 4 and Theorem 6, that running a search algorithm on the beamlet graph yields a reduction by a factor $\log n$ when compared to the same algorithm being run on the nearest neighbor graph.

C. Memory Usage

Since the beamlet-based graph structure involves preprocessing, the memory usage for storing the precomputed information needs to be estimated. Recall that the main step during processing is the bottom-up fusion algorithm, which is a recursive algorithm. In each recursion, only the free boundary cells from the four d-squares at the immediately finer scale are given as input to the all-pairs shortest path algorithm.

During the path-finding step, recall that in our complexity analysis we showed that, for an $n \times n$ image, the number of vertices in the beamlet graph has an upper bound of $16n$ and the number of edges has an upper bound of $8n^2$. Hence, the storage overhead for the beamlet graph is of order $O(n^2)$. Since we also need to store all shortest paths, the additional memory required is

$$\sum_{s=1}^{\log n} 6\binom{n2^{2-s}}{2} 6n2^{-s} \asymp O(n^3). \tag{2}$$

TABLE I
OVERALL COMPLEXITY COMPARISON

Complexity	No Preprocessing	m-A*	All-pairs Computed
Preprocessing	0	$O(n^3)$	$O(n^4 \log n)$
Query Time	$O(n^2 \log n)$	$O(n^2)$	$O(\log n)$

To see this, recall that for a scale s d-square, the number of edges is less than or equal to $\binom{n2^{2-s}}{2}$ and there are 6 (or less) d-squares at each scale s. It therefore requires no more than $6n2^{-s}$ vertices to record the shortest path between any two free boundary cells within a scale s d-square. By summing over all scales, (2) results. In summary, the total storage overhead is, asymptotically, $O(n^3)$. The memory overhead for the nearest neighbor graph is easily computed to be $O(n^2)$, because in the nearest neighbor graph both vertices and edges are of order $O(n^2)$ and one only needs to consider traversability between neighboring cells.

D. Preprocessing Time

The proposed multiscale strategy combines the bottom-up fusion algorithm of Section III-C with the search algorithm on the reduced-size beamlet graph. Recall that in each d-square of the PFR-RDP, an all-pair shortest path algorithm is solved. As seen already, the computational complexity of this preprocessing step is $O(n^3)$. Depending on the resolution used in the RDP, two extremes for the single-source path planning problem arise:

- Running a search algorithm (with Fibonacci Heap) on the entire environment (on the nearest neighbor graph), making no use of multiscale information. The overall complexity of this method is $O(n^2 \log n)$.
- Running an all-pairs shortest path finding algorithm (such as Johnson's algorithm) on the entire environment as the preprocessing step, which has complexity $O(n^4 \log n)$, and then extract the shortest distance for the given source and destination. The second step just involves a binary search and has complexity $O(\log n)$. The total complexity of this option is therefore $O(n^4 \log n)$.

The proposed m-A* algorithm falls in-between these two extremes (see also Table I). In terms of applications, this flexibility can prove to be very useful. For instance, for many vehicles with embedded autonomous capabilities (e.g., ground robots, small UAVs) on-board processing of the available data can quickly become an implementation bottleneck during path-planning and execution. Making use of off-line (pre-computed) information is often helpful in practice to overcome this limitation. The proposed approach can be seen in this context as a compromise between off-line pre-processing and on-line search. Of course, in cases where the multiscale information of the environment is provided directly to the planner by a suitable sensor, making use of the proposed multiscale graph structure will outperform traditional search algorithms by orders of magnitude, as demonstrated by both the previous complexity analysis and by our numerical studies.

V. NUMERICAL STUDIES

In this section, we provide numerical experiments to compare the performance from running the A* algorithm on the standard nearest neighbor graph and the proposed multiscale A* on the new beamlet graph. The comparison is based on the number of vertex expansions during the search. This is a more accurate criterion than, say, running time, since the query time heavily depends on the computer hardware used in each case. In these numerical experiments, the bottom-up fusion algorithm and the A* algorithm with Fibonacci heap were implemented in Matlab 2009(a) and C++, respectively, on an Intel Core2 Duo CPU 2.26 Ghz, with 1.89 GB of RAM, running Windows XP.

TABLE II
MULTISCALE A* ALGORITHM COMPARED TO TRADITIONAL A*. EXAMPLE I

ImageSize	64×64		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	644 (19)	152 (10)	4.24 (1.90)
Exp2	654 (15)	164 (11)	3.99 (1.36)
Exp3	630 (24)	155 (15)	4.06 (1.60)
Exp4	622 (25)	174 (15)	3.56 (1.71)
Exp5	656 (21)	149 (12)	4.41 (1.75)
ImageSize	128×128		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	1299 (207)	185 (97)	7.02 (2.13)
Exp2	1337 (301)	207 (87)	6.46 (3.46)
Exp3	1316 (198)	204 (96)	6.45 (2.06)
Exp4	1443 (62)	201 (57)	7.18 (1.09)
Exp5	1303 (203)	202 (95)	6.46 (2.13)

A. Comparison Based on L_1 Heuristic

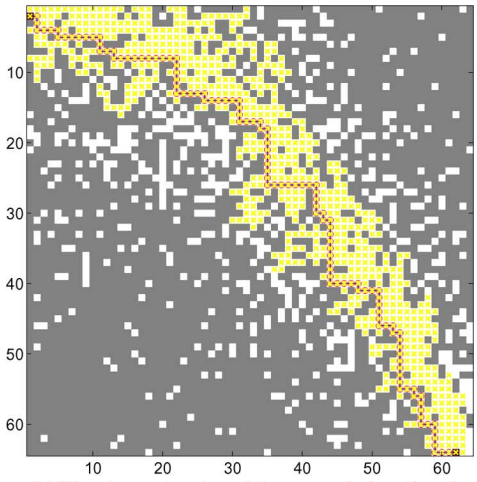
In this section we provide the results from numerical simulations using three distinct cases of an obstacle-filled environment. In the first simulation, the probability of a certain cell (at location (x, y), where $1 \leq x, y \leq n$,) to be a free cell is assigned to be $p(x, y) = \exp(-\gamma|y - x^2/n|)$, where the constant γ will be specified later. The intuition of this model is that cells near the curve $y = x^2/n$ have higher probability to be free than cells far away from the same curve. This gridworld simulates the situation when there is a main "corridor" in the environment.

Table II contains the simulation results when $\gamma = 1/15$. The numbers in the first two columns indicate the number of vertex expansions in each case. The values in parentheses are the corresponding running times in milliseconds. The last column shows the ratios between the number of the expanded nodes used by the two algorithms. Fig. 9 shows the comparison of the shortest path from the nearest-neighbor graph and the beamlet graph, respectively. The yellow crosses show the corresponding expanded vertices in both cases. As shown in this simulation, the m-A* algorithm clearly outperforms A* in terms of the number of vertex expansions.

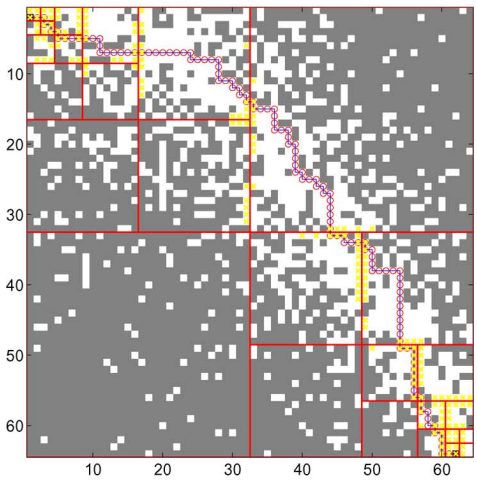
The second simulation represents a much more difficult situation for path planning, and it involves a heavily cluttered environment (Fig. 10). The presence of a large number of randomly distributed small obstacles will typically require a high number of vertex expansions for all search algorithms applied on the nearest neighbor graph.

A comparison of the vertex expansions (query time) for both A* and m-A* is given in Table III. The table shows again the advantage of m-A* over the traditional A* algorithm in terms of the number of vertex expansions.

The proposed beamlet-based graph structure was also tested on a real-world environment. Specifically, Fig. 11 shows the elevation map of a certain area in the US. In the figure, gray areas are obstacles. The results of the shortest paths and the node

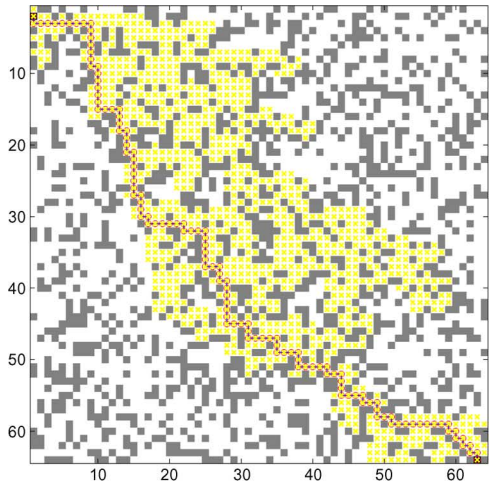


(a) The shortest path and the expanded vertices in the nearest neighbor graph.

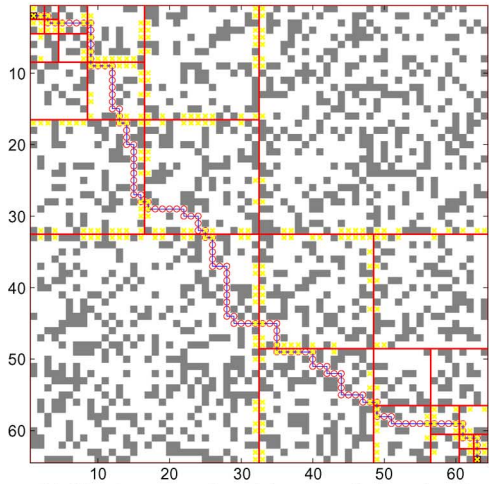


(b) The shortest path and the expanded vertices in the beamlet graph.

Fig. 9. Example I: (a) The shortest path of NNG in a 64×64 image; the yellow crosses denote the expanded vertices during the search; (b) The shortest-path and the expanded vertices in the beamlet graph for the same image.



(a) The shortest path and the expanded vertices in the nearest neighbor graph.



(b) The shortest path and the expanded vertices in the beamlet graph.

Fig. 10. Example II: (a) The shortest path in the NNG for a 64×64 image; the yellow crosses denote the expanded vertices during the search; (b) The shortest-path and the expanded vertices in the beamlet graph for the same image.

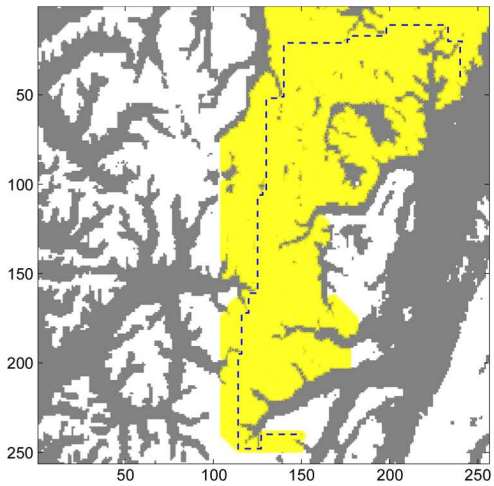
expansions obtained using the nearest neighbor graph and the beamlet graph are shown in the same figure. For this scenario, the number of node expansions in the nearest neighbor graph and the beamlet graph are 16083 and 1043, respectively, which shows that by using the proposed graph structure, the speed of the shortest path-finding benchmark algorithm can be improved by an order of magnitude.

B. Comparison Using Stronger Heuristics

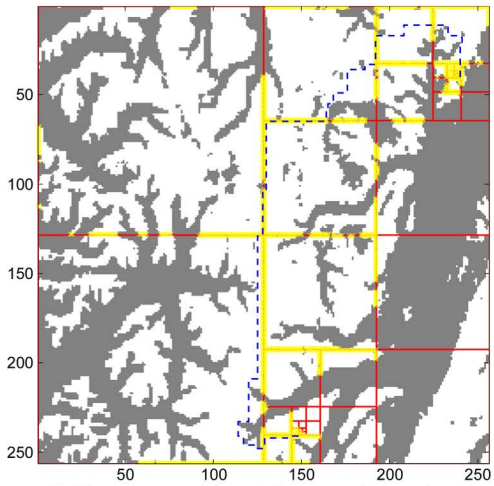
The beamlet graph can be viewed as a data structure that stores a more accurate heuristic than the basic L_1 heuristic. Thus, the superiority of m-A* over A* is the greatest when the baseline heuristic is the weakest. The situation may be quite different if the baseline heuristic is very accurate. So the question of whether m-A* can still outperform A* with a more accurate heuristic is relevant. Consequently, we also tested m-A* with a very accurate heuristic, namely, a true distance heuristic (TDH) with differential distance heuristic. In order to ensure a fair comparison, the number of landmarks were of the same order as the nodes in the beamlet graph, that is, of order $k = O(\sqrt{n^2}) = O(n)$. To this end, we randomly chose k rows out of n rows in the image. By symmetry, we also chose the corresponding k columns. All the free nodes in these k rows and k columns are considered as landmarks. This is similar to the way landmarks are chosen in Fig. 1(b) of [3].

TABLE III
MULTISCALE A* ALGORITHM COMPARED TO TRADITIONAL A*. EXAMPLE II

ImageSize	64×64		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	1931 (307)	345 (260)	5.60 (1.18)
Exp2	2025 (130)	366 (87)	5.53 (1.49)
Exp3	2396 (234)	383 (159)	6.26 (1.47)
Exp4	2223 (267)	351 (46)	6.33 (5.80)
Exp5	2017 (181)	346 (143)	5.83 (1.27)
ImageSize	128×128		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	7254 (164)	666 (140)	10.89 (1.71)
Exp2	7755 (320)	814 (143)	9.53 (2.24)
Exp3	6712 (217)	742 (198)	9.05 (1.10)
Exp4	7755 (320)	814 (143)	9.52 (2.24)
Exp5	7588 (299)	770 (281)	9.85 (1.06)

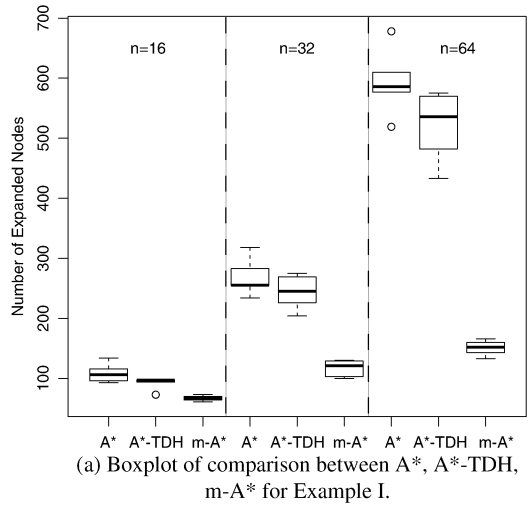


(a) The shortest path and the expanded vertices in the nearest neighbor graph.

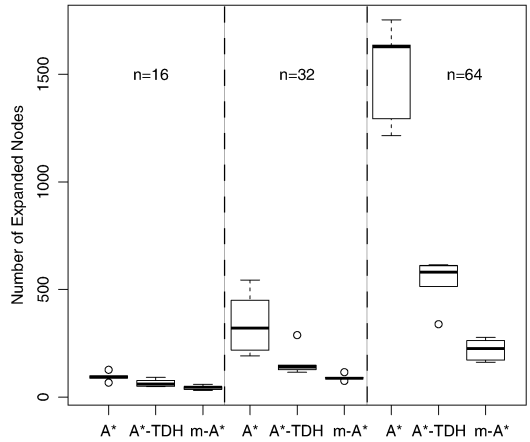


(b) The shortest path and the expanded vertices in the beamlet graph.

Fig. 11. Example III: (a) The shortest path in the NNG for a 256×256 image; the yellow crosses denote the expanded vertices during the search; the blue dashed line shows the shortest path. (b) The shortest-path and the expanded vertices in the beamlet graph for the same image.



(a) Boxplot of comparison between A*, A*-TDH, m-A* for Example I.



(b) Boxplot of comparison between A*, A*-TDH, m-A* for Example II.

Fig. 12. (a) Summary of numerical comparison between A* with L_1 heuristic, A* with TDH, and m-A* for Example I. Here n denotes the size of the grid-world. Each comparison group is derived based on numerical results from five randomly generated gridworlds.; (b) Summary of numerical comparison between A* with L_1 heuristic, A* with TDH, and m-A* for Example II. The notation is the same as in Example I.

The results from the comparison between the standard A* (with L_1 distance heuristic), the A* with a true distance heuristic (A*-TDH) and the m-A* are shown in Fig. 12. For each image size of 16×16, 32×32 and 64×64, we randomly constructed five different gridworlds and run A*, A*-TDH, m-A* on each one respectively. As shown in these figures, m-A* gives the smallest number of node expansion among the three algorithms even when the stronger TDH heuristic is used. Furthermore, it is shown that the gain from m-A* is more significant as the image size increases.

VI. Discussion and Related Prior Work

A. Beamlets as a Predecessor

The algorithm proposed in this paper is rooted in the beamlet theory—a multiscale methodology for linear and curvilinear features in 2-D. Beamlets provide a framework for multiscale analysis, in which line segments play a role analogous to the role played by points in wavelet analysis. They add two crucial elements missing from wavelet processing, however: orientation and elongation information [1]. Beamlets are proven to achieve optimal asymptotic performance in feature detection problems [17]. They have been used to design efficient coding algorithms for images made of curves [18], [19]. Beamlets are numerically more efficient than traditional curve processing algorithms, such as JBIG2 [20], because they make use of their inherent multiscale structure in an innovative and efficient manner. The PFR-RDP used in this article is similar to the beamlet-decorated recursive dyadic partitioning of [1], depicted in Fig. 13.

B. Related Work

Recent work on the shortest path finding problem bears some similarities with our methodology and thus a comparison is warranted. From the vast amount of the existing literature, we distinguish four algorithms that use multi-scale ideas to speed up the search during execution: state abstraction strategies (AS) [21]–[23], hierarchical A* (h-A*) [24], contraction hierarchies (CH) [25], and true distance heuristics (TDH) [2], [3].

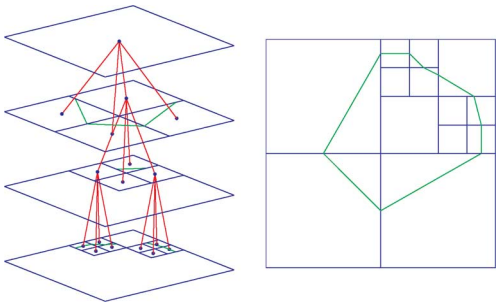


Fig. 13. To each path in the beamlet graph in [1] corresponds a polygonal curve in the plane.

Recall that the beamlet graph is induced from the nearest neighbor graph, and it has a reduced number of vertices. It takes advantage of "long-distance" interactions between vertices, which correspond to cells that may be very far apart in the original map. This is similar—in spirit—to the state abstraction strategy that has been intensively analyzed in [21], [22] and originally proposed in [23]. The strategy consists of building heuristic functions in an abstraction space. Actions in the abstraction space are then refined to actions in the environment by the A* algorithm, thereby substantially accelerating searching. Despite the apparent similarity between m-A* and the algorithms in [21] and [22] in the use of abstraction ideas, there are nonetheless distinct differences between the two approaches. First, their objectives are quite different. The objective of [21] is real-time execution and adaptive graph exploration; the goal is to determine the few next moves locally and quickly. Our objective, instead, is to find the shortest path globally. As a result, m-A* always guarantees an optimal solution, while the AS in [21] only ensures sub-optimality. On the other hand, AS works with general graphs, and not only with gridworld graphs, which are the focus of the present paper. Most closely related to m-A* is probably the sector abstractions (SA) of [22]. Sector abstractions are inspired from the hierarchical path finding A* (HPA*) algorithm of [26] and are also limited to grid-based maps. The key idea is to divide the map into clusters (corresponding to equally-sized squares) and generate a new graph by using the information of the free cells belonging to the boundary of these clusters. This bears a strong resemblance with m-A*. However, the beamlet-like connectivity is absent in [22], [26] and the vertices of the abstract graph are just a subset of the original gridded map. Furthermore, m-A* adopts a "bottom-up" fusion algorithm to organize pre-computed information based on the self-similarity across multiple resolution scales. This is missing from HPA* and SA.

The hierarchical A* (h-A*) algorithm of [24] builds a hierarchical abstraction of the state space based on the "clustering" of the "max-degree" vertex in the nearest neighbor graph. The process is repeated until all states (i.e., vertices) have been assigned to some abstract state. The heuristic used in the original graph is obtained via a shortest path search in the abstracted state space graph. Redundant expansion of nodes is avoided, since once the shortest path search from one vertex to the goal in the higher abstraction graph is completed, all the heuristics of the subsequent vertices in the path to the goal vertex, and its children on the next abstraction level are obtained without much difficulty (the latter is called "h*-caching" in [24]). The proposed beamlet-based graph also induces a hierarchical structure on the original graph, which is obtained, however, via path-finding on the PFR-RDP. The bottom-up fusion algorithm combines local information at the finer (lower) levels into global information at the upper (coarser) levels, while avoiding redundant calculations at each level, since only free boundary cells are involved at each step. Another difference between m-A* and h-A* is that for the gridworld graphs we consider, there is no difference between the degrees of the vertices in the graph (that is, all vertices have the same degree owing to the use of a topological graph), which means that the PFR-RDP is a more appropriate tool for building the hierarchical information structure.

In [25] the concept of *contraction hierarchies* (*CH*) is proposed for path planning for large-scale road networks. In [27] it is shown that hierarchical abstractions and CH have similar overhead and performance. Contraction hierarchies is a particularly powerful technique for road networks. There are two basic steps involved in CH: (a) an ordering of all nodes, (b) the contraction, i.e., the removal of certain nodes based on the ordering, and the insertion of shortcut edges, so that the shortest paths are preserved when the contracted nodes are removed from the graph. CH works with any node ordering; however, the ordering has a huge influence on preprocessing and query performance. Although several alternatives exist, a bidirectional search is typically used on the contracted graph to carry out the search [4]. Although both CH and m-A* introduce hierarchies, the former is specifically designed for roadmaps, whereas the beamlet graph used in m-A* is more suitable for gridworlds. The induced PFR-RDP in m-A* takes advantage of the topological information in the gridworld, which does not exist in the CH framework. The ordering of vertices in CH is critical and quite complex, involving many considerations. The PFR-RDP used in m-A*, on the other hand, is intuitive and easy to implement. However, the PFR-RDP only makes sense when a gridworld formulation is adopted. Despite the similarities between CH and the beamlet graph (BG), neither of them is a special case of the other. For an $n \times n$ image the CH has n^2 levels, while the BG has $\log n$ scales. Therefore BG is a lot simpler. Furthermore, the bottom-up fusion algorithm of m-A* is also very different from the contraction steps used in CH. By design, it is expected that the bottom-up fusion algorithm should be much faster. In fact, as shown in Section IV-A, the complexity of the bottom-up fusion algorithm is $O(n^3)$. No order of complexity is known for the CH, due to the many possibilities in its realization. Note however that CH would run contraction steps n^2 times. It is thus expected that, in the worst case, the contraction steps in CH will be more than $O(n^3)$ (given that there are n^2 nodes).

Other recent path-planning algorithms focus on the use of "optimal" distance heuristics to guide the search during the A* and thus speed up the graph search algorithm. Although the all-pair distance would be the perfect heuristic for searching the shortest path, the calculation and memory required would be unrealistic. The *true distance heuristic* (*TDH*) of [2], [3] reduces the memory requirements by using only a subset of the all-pairs-shortest-path information. To achieve this, the method predetermines the distance between k "landmark" nodes (where

$k \ll n^2$)—also called canonical states in [2], [3]—on which the all-pair shortest paths algorithm runs. This enables a more precise estimate of the distance to the destination. The complexity of TDH can be easily verified to be $O(kN) = O(n^3)$, where $N = n^2$ is the number of vertices in the graph with corresponding memory requirements $O(kn^2) = O(n^3)$, if $k = O(\sqrt{n^2})$ (cf. Table 1 in [2]). That is, both TDH and the proposed algorithm share the same order of magnitude of computation and storage complexity. However, the success of the TDH depends crucially on the appropriate selection of the k canonical states. In fact, as mentioned in [2] "the best number and location of canonical states is an open problem." The boundary free cells obtained systematically via the PFR-RDP in m-A* can be viewed as canonical states. In that sense, the proposed PFR-RDP provides (at least) a partial answer to the previous question. Most importantly, our beamlet graph is built on multiscale information, which is not evident in the TDH approach.

It is important to mention that the proposed m-A* is not tied to any particular heuristic; any suitable heuristic from the literature will do. Instead, m-A* takes advantage of a simpler graph to perform the search—the beamlet graph. Therefore, although for the sake of simplicity, in the current implementation a unidirectional search along with the L_1-distance heuristic was used, a bidirectional search could have been used instead without much difficulty, with all the conclusions remaining essentially the same. Note that many of the most recent work in advance path-planning [4], [25], [28] all employ bidirectional search during their execution of the A* algorithm.

Since the original submission of the paper, new related work has been published including [29], [30], both of which incorporate similar aspects with the proposed graph search algorithm. The method in [30] identifies rectangular empty areas and prunes all interior nodes, leaving only the ones at the perimeter, similarly to the boundary cells of PFR-RDP in m-A*. The authors in [29] partition the original graph into disjoint subgraphs with few border nodes (the so-called "portals"). True distances between all pairs of portals are stored and used as admissible heuristics. This idea is similar to the "beamlet" connectivity between the boundary nodes of PFR-RDP used in m-A*.

VII. Conclusion

We have introduced an innovative beamlet-based graph structure that facilitates path-finding in gridworlds with obstacles. The main idea is the construction of a graph structure that encodes efficiently long-distance interactions between vertices that go beyond the four-neighbor connectivity relations of the underlying nearest-neighbor topological graph. This is achieved by iterative subdivisions of the entire environment that give rise to a hierarchy of graphs, along with an innovative bottom-up fusion algorithm that combines local information from the lower (finer) scales to obtain global information at the upper (coarser) scales. Both the theoretical complexity analysis as well as the numerical examples demonstrate that the proposed multiscale A* (m-A*) algorithm provides significant improvements over the original A* algorithm applied on the original nearest neighbor graph.

The proposed graph structure can be viewed as the foundation on which several extensions of existing graph search algorithms can be based on. For instance, the multiscale graph structure in an incremental search setting would lead to multiscale versions of LPA* and/or D* algorithms [31]–[34], allowing to handle dynamic changes in the environment. For some initial results towards this direction, see [35]. In case of motion-planning, beamlets can be used to efficiently and naturally incorporate curvature restrictions on the resulting paths. This is crucial for mobile agents with limited turning capability (e.g., fixed-wing UAVs, unmanned ground vehicles, etc). Also, the generalization of the proposed 2-D multiscale strategy to 3-D (or higher dimensions) is straightforward, albeit computationally more involved. Finally, the recursive nature of the bottom-up fusion algorithm invites the possibility for its parallelization. The potential of the parallel execution of the bottom-up fusion algorithm would tremendously improve the overall performance of m-A*.

Acknowledgment

The authors would also like to thank O. Arslan for his help with the implementation of A* with True Distance Heuristic (TDH).

References

[1] D. Donoho and X. Huo, "Beamlets and multiscale image analysis," *Multiscale and Multiresolution Methods*, vol. 20, pp. 149–196, Spring, 2002.

[2] A. Felner, N. Sturtevant, and J. Schaeffer, "Abstraction-based heuristics with true distance computations," in *Proc. 8th Symp. Abstraction, Reformulatoin, Approx,*, 2009, pp. 74–81.

[3] N. Sturtevant, A. Felner, M. Barer, J. Schaeffer, and N. Burch, "Memory-based heuristics for explicit state spaces," in *Proc. Int. Joint Conf. Artif. Intell.*, Pasadena, CA, Jul. 11–17, 2009, pp. 609–614.

[4] A. Goldberg and C. Harrelson, "Computing the shortest path: A* search meets graph theory," in *Proc. 16th ACM-SIAM Symp. Discrete Algorithms*, 2005, pp. 156–165.

[5] D. de Champeaux and L. Sint, "An improved bidirectional heuristic search algorithm," *J. ACM*, vol. 24, no. 2, pp. 177–191, 1977.

[6] I. Pohl, "Bi-directional Search," in *Machine Intelligence*. Edinburgh, U.K.: Edinburgh Univ. Press, 1971, vol. 6, pp. 127–140.

[7] D. Ferguson, M. Likhachev, and T. Stentz, "A guide to heuristic-based path planning," in *Proc. Int. Workshop Planning Under Uncertainty Auton. Syst., Int. Conf. Autom. Planning Scheduling (ICAPS)*, Jun. 2005, [CD ROM].

[8] E. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[9] P. Hart, N. Nilsson, and B. Rafael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[10] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[11] X. Huo, "Sparse Image Representation via Combined Transforms," Ph.D. dissertation, Dept. Statistics, Stanford Univ., Stanford, CA, Aug. 1999.

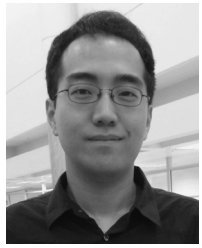
[12] D. Donoho and X. Huo, "Beamlets pyramids: A new form of multiresolution analysis, suited for extracting lines, curves and objects from very noise image data," in *Proc. SPIE*, Jul. 2000, vol. 4119, no. 1, pp. 434–444.

[13] D. Donoho and X. Huo, "Applications of beamlets to detection and extraction of lines, curves, and objects in very noisy images," in *Proc. Nonlin. Signal Image Processing*, Jun. 2001, [CD ROM].

[14] D. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[15] P. E. Hart, N. J. Nillsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern. SSC4*, vol. 4, no. 2, pp. 100–107, 1968.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press and McGraw-Hill, 2001.
[17] E. Arias-Castro, D. Donoho, and X. Huo, "Near-optimal detection of geometric objects by fast multiscale methods," *IEEE Trans. Inform. Theory*, vol. 51, no. 7, pp. 2402–2425, Jul. 2005.
[18] X. Huo and J. Chen, "JBEAM: Multiscale curve coding via beamlets," *IEEE Trans. Image Processing*, vol. 14, no. 11, pp. 1665–1677, Nov. 2005.
[19] D. Donoho and X. Huo, "Beamlab and reproducible research," *Int. J. Wavelets, Multiresolution Inform. Processing*, vol. 2, no. 4, pp. 391–414, 2004.
[20] P. G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. J. Rucklidge, "The emerging JBIG2 standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 7, pp. 838–848, Jul. 1998.
[21] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau, "Graph abstraction in real-time heuristic search," *J. Artif. Intell. Res.*, vol. 30, pp. 51–100, 2007.
[22] N. Sturtevant and R. Jansen, "An analysis of map-based abstraction and refinement," in *Proc. 7th Int. Conf. Abstraction, Reform., Approx.*, 2007, pp. 344–358.
[23] N. Sturtevant and M. Buro, "Partial pathfinding using map abstraction and refinement," in *Proc. Nat. Conf. Artif. Intell.*, 2005, vol. 20, no. 3, p. 1392.
[24] B. Larsen, E. Burns, W. Ruml, and R. Holte, "Searching without a heuristic: Efficient use of abstraction," in *Proc. 24th AAAI Conf. Artif. Intell. (AAAI-10)*, 2010, pp. 114–120.
[25] R. Geisberger, "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks," Ph.D. dissertation, Institut für Theoretische Informatik, Universität Karlsruhe, , 2008.
[26] A. Botea, M. Muller, and J. Schaeffer, "Near-optimal hierarchical pathfinding," *J. Game Development*, vol. 1, pp. 1–30, 2004.
[27] N. Sturtevant and R. Geisberger, "A comparison of high-level approaches for speeding up pathfinding," in *Proc. 6th Artif. Intell. Interactive Digital Entertainment Conf.*, Palo Alto, CA, Oct. 11–13, 2010, pp. 76–82.
[28] A. Goldberg, H. Kaplan, and R. Werneck, "Reach for A*: Efficient point-to-point shortest path algorithms," in *Proc. Workshop Algorithm Eng. Exper.*, 2006, pp. 129–143.
[29] M. Goldenberg, A. Felner, N. Sturtevant, and J. Schaeffer, "Portal-based true-distance heuristics for path finding," in *Proc. 3rd Annu. Symp. Combinatorial Search*, 2010, pp. 39–45.
[30] D. Harabor and A. Botea, "Breaking path symmetries on 4-connected grid maps," in *Proc. 6th Annu. Int. AIIDE Conf.*, Palo Alto, CA, Oct. 11–13, 2010, [CD ROM].
[31] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artif. Intell. J.*, vol. 155, no. 1–2, pp. 93–146, 2004.
[32] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, "Incremental heuristic search in artificial intelligence," *Artif. Intell. Mag.*, vol. 25, no. 2, pp. 99–112, 2004.
[33] S. Koenig and M. Likhachev, " D* lite," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2002, pp. 476–483.
[34] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," *Int. J. Robot. Autom.*, vol. 10, no. 3, pp. 89–100, 1995.
[35] Y. Lu, X. Huo, O. Arslan, and P. Tsiotras, "An incremental, multiscale search algorithm for dynamic path planning with low worst case complexity," *IEEE Trans. Man, Syst. Cybern. B*, vol. 41, no. 6, pp. 1556–1570, Dec. 2011.



Yibiao Lu received the B.S. degree in applied mathematics from the University of Science and Technology of China, Hefei, in 2008 and is currently pursuing the Ph.D. degree at the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta.

His research interests are applied statistics and machine learning.

Xiaoming Huo (SM'04) received the B.S. degree in mathematics from the University of Science and Technology, Hefei, China, in 1993 and the M.S. degree in electrical engineering and the Ph.D. degree in statistics from Stanford University, Stanford, CA, in 1997 and 1999, respectively.

Since August 2006, he has been an Associate Professor with the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta. He represented China in the 30th International Mathematical Olympiad (IMO), which was held in Braunschweig, Germany, in 1989, and received a golden prize. His research interests include statistics and multiscale methodology. He has made numerous contributions on topics such as sparse representation, wavelets, and statistical problems in detectability.

Dr. Huo was a Fellow of IPAM in September 2004. He received the Georgia Tech Sigma Xi Young Faculty Award in 2005. His work has led to an interview by Emerging Research Fronts in June 2006 in the field of Mathematics—every two months, one paper is selected.

Panagiotis Tsiotras (SM'02) received B.S. degree in mechanical engineering from the National Technical University of Athens, Greece in 1986, the M.S. degree in aerospace engineering from Virginia Polytechnic Institute and State University, Blacksburg, in 1987, and the M.S. degree in mathematics and the Ph.D. degree in aeronautics and astronautics from Purdue University, West Lafayette, IN, in 1992 and 1993, respectively.

He is a Professor in the School of Aerospace Engineering, Georgia Institute of Technology. His research interests are in optimal and nonlinear control and vehicle autonomy.

Dr. Tsiotras received the NSF CAREER Award. He is a Fellow of the AIAA.