Multiresolution On-Line Path Planning for Small Unmanned Aerial Vehicles

Dongwon Jung and Panagiotis Tsiotras

Abstract— **In this article we propose a new online multiresolution path planning algorithm for a small unmanned air vehicle (UAV) with limited on-board computational resources. The proposed approach assumes that the UAV has detailed information of the environment only in the vicinity of its current position. Information about far away obstacles is also available, albeit with less accuracy. The proposed algorithm uses an integer arithmetic implementation of the fast lifting wavelet transform (FLWT) to get a multiresolution cell decomposition of the environment, whose dimension is commensurate to the on-board computational resources. A topological graph representation of the multiresolution cell decomposition is constructed efficiently, directly from the approximation and detail wavelet coefficients. Hardware-in-the-loop simulation (HILS) results validate the applicability of the algorithm on a small UAV autopilot. Comparisons with the standard $\mathcal{D}^*$-lite algorithm are also presented.**

I. Introduction

Autonomous operation of UAVs requires both trajectory design (planning) and trajectory tracking (control) tasks to be completely automated. Given the short response time scales of modern aerial vehicles, on-board, real-time path planning is particulary challenging for small UAVs, which may not have the on-board computational capabilities (e.g. CPU and memory) to implement some of the sophisticated path planning algorithms developed in the literature.

In a typical mission of a UAV, various sensors (e.g., cameras, radars, laser scanners, satellite imagery) having different range and resolution characteristics are employed to collect information about the environment the vehicle operates in. A computationally efficient path planning algorithm, specifically adopted for on-line implementation, should therefore choose the expedient information from all these sensors, and utilize the on-board computational resources on the part of the path (spatial and temporal) that needs it most. In a nutshell, a computationally efficient algorithm suitable for *on-line* implementation should be characterized by a combination of short term tactics (reaction to unforeseen threats) with long-term strategy (planning towards the ultimate goal).

Multiresolution analysis is widely used in practice to mitigate the computational overhead in numerically costly applications, for example, computer graphics, using progressive, view-dependent, meshes [4]. The application of multiresolution methods to path planning problems is relatively recent. In [1], [13] multiresolution, hierarchical algorithms were used to alleviate the computational burden associated with path planning over a complex, unstructured, or partially known environment. The typical approach in this context involves the use of quadtrees [12], [16]. One drawback of quadtree-based decompositions is that a fine resolution is used close to the boundaries of all obstacles, regardless of their distance from the agent. This tends to waste computational resources.

Recently, [15] proposed an efficient hierarchical path planning algorithm for autonomous agents navigating in a partially known environment $\mathcal{W}$ using an adaptive, discrete, cell-based approximation of $\mathcal{W}$. A high resolution representation of $\mathcal{W}$ is used close to the current position of the agent (leading to a local solution with great accuracy), while a low resolution representation is used far away from the vehicle (thus incorporating the ultimate goal objective). It should be noted that this path planning algorithm is distinguished from the earlier wavelet-based motion planning algorithm in Ref. [13], in that it always incorporates high resolution representation nearby the agent, whereas the algorithm in Ref. [13] incorporates several stages of refinement to compute a path at a certain level of accuracy.

In this article, we present a multiresolution hierarchical path planning algorithm, which is an extension of the algorithm developed in Ref. [15]. We use the fast lifting wavelet transform (FLWT), to construct the associated cell connectivity relationship directly from the wavelet coefficients. This eliminates the need for quadtree decompositions, as in Ref. [15]. The FLWT allows us to use integer arithmetic that results in faster speeds during hardware implementation with a small-size microcontroller.

II. A Multiresolution Decomposition of $\mathcal{W}$

A. The 2D wavelet transform

Let a function f over the domain $\mathcal{W} = [0,1] \times [0,1]$ be discretized using a (fine) dyadic grid of dimension $2^N \times 2^N$. Its discrete 2D wavelet transform at scale $J \leq N$ is given by

$$f(x,y) = \sum_{k,\ell=0}^{2^J-1} a_{J,k,\ell}\Phi_{J,k,\ell}(x,y) + \sum_{i=1}^{3}\sum_{j=J}^{N-1}\sum_{k,\ell=0}^{2^j-1} d^i_{j,k,\ell}\Psi^i_{j,k,\ell}(x,y), \tag{1}$$

where $a_{J,k,\ell}$ are the approximation coefficients and $d^i_{j,k,\ell}$ are the detail coefficients, and $\Phi_{J,k,\ell}$ and $\Psi^i_{j,k,\ell}(x,y)$ are the scaling function and wavelets constructed by tensor products of the corresponding 1D functions (see [2]). In this paper we use the Haar wavelet system for reasons that will become

D. Jung is a Post-doctoral Fellow at the School of Aerospace, Georgia Institute of Technology, Atlanta, GA, 30332 USA, E-mail: dongwon.jung@gatech.edu

P. Tsiotras is a Professor at the School of Aerospace, Georgia Institute of Technology, Atlanta, GA, 30332 USA E-mail: tsiotras@gatech.edu

apparent shortly. The Haar family has compact support on the interval $[0, 1]$. It can be shown that in the 2D case the scaling function $\Phi_{j,k,\ell}$ and the wavelet function $\Psi^i_{j,k,\ell}$ $(i = 1, 2, 3)$ have compact support on the rectangle $c^j_{k,\ell} \triangleq [k/2^j, (k+1)/2^j] \times [\ell/2^j, (\ell+1)/2^j]$. Subsequently, we may associate the two-dimensional scaling function $\Phi_{j,k,\ell}$ and the wavelet functions $\Psi^i_{j,k,\ell}$ with the rectangular cell $c^j_{k,\ell}$.

We use the fast lifting wavelet transform (FLWT) as the main mathematical tool. The FLWT offers a fast computation of the wavelet transform, in conjunction with the use of integer arithmetic to further reduce the computational cost. In addition, the FLWT computations are done *in-place*, allowing the immediate inverse transform, thus saving computational memory. This makes the FLWT especially suitable for processing using small micro-controllers.

B. Wavelet decomposition of the risk measure

Assume now that we are given a function $\mathsf{rm} : \mathcal{W} \mapsto \mathcal{M}$ that represents the "risk measure" [15] at the location $\mathsf{x} = (x, y)$, where $\mathcal{M}$ is a collection of m integer distinct risk measure levels. We may think of $\mathsf{rm}(\mathsf{x})$ as an indication of the proximity of the agent to the obstacle space, or the probability that the agent belongs to the obstacle space.

We construct approximations of $\mathcal{W}$ at distinct levels of resolution $J_{\min} \le j \le J_{\max}$, at ranges r_j from the current location of the agent $\mathsf{x}_0 = (x_0, y_0)$, in the sense that resolution j is used for all points inside the neighborhood

$$\mathcal{N}(\mathsf{x}_0, r_j) \triangleq \{\mathsf{x} \in \mathcal{W} : \|\mathsf{x} - \mathsf{x}_0\|_\infty \le r_j\}, \tag{2}$$

where $r_{J_{\max}} \le r_j \le r_{J_{\min}}$. That is, the finer resolution $J_{\max}$ is used for points close to the current location, and coarser resolutions are used elsewhere, according to the distance from the current location. Figure 1 illustrates this situation. The choice of $J_{\max}$ is determined by the requirement that at this level all cells can be resolved into either free or obstacle cells. The choice of $J_{\min}$ as well as the window span r_j are dictated by the on-board computational resources.

Let now $\mathcal{I}(j) \triangleq \{0, 1, 3, \cdots, 2^j - 1\}$ and let

$$\mathcal{K}(j) \quad \triangleq \{k \in \mathcal{I}(j) \mid I_{j,k} \cap [x_0 - r_j, x_0 + r_j] \neq \varnothing\}, \tag{3a}$$
$$\mathcal{L}(j) \quad \triangleq \{\ell \in \mathcal{I}(j) \mid I_{j,\ell} \cap [y_0 - r_j, y_0 + r_j] \neq \varnothing\}. \tag{3b}$$

where $I_{j,k} \triangleq [k/2^j, (k+1)/2^j]$ is the dyadic interval of length $1/2^j$. Then the wavelet decomposition of rm, given by

$$\begin{aligned}\mathsf{rm}(x, y) = &\sum_{k,\ell \in \mathcal{I}(J_{\min})} a_{J_{\min},k,\ell} \Phi_{J_{\min},k,\ell}(x, y) \\ &+ \sum_{i=1}^{3} \sum_{j=J_{\min}}^{J_{\max}-1} \sum_{\substack{k \in \mathcal{K}(j) \\ \ell \in \mathcal{L}(j)}} d^i_{j,k,\ell} \Psi^i_{j,k,\ell}(x, y),\end{aligned} \tag{4}$$

induces, with a slight abuse of notation, the following multiresolution cell decomposition on $\mathcal{W}$

$$\mathcal{C}_d = \Delta C_d^{J_{\min}} \oplus \cdots \oplus \Delta C_d^{J_{\max}}, \tag{5}$$

where, ΔC^j_d is a union of cells $c^j_{k,\ell}$ of dimension $1/2^j \times 1/2^j$.

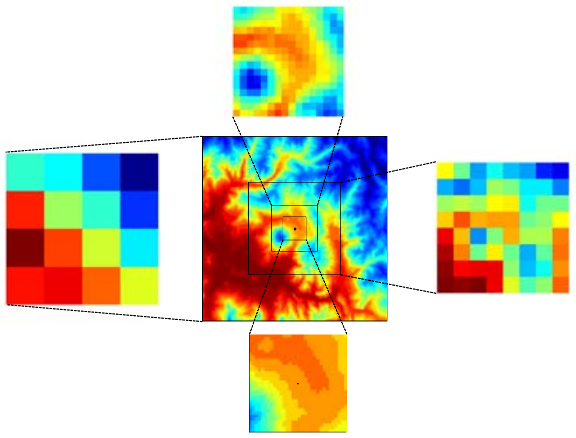


Fig. 1. Multiresolution representation of the environment according to the distance from the current location of the agent.

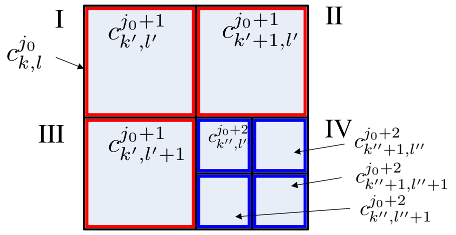


Fig. 2. Multiresolution cell subdivision across different levels.

III. Multiresolution Graph Connectivity

A. Computation of Adjacency List from the FLWT

We assign a topological graph $\mathcal{G} = (V, E)$ to the cell decomposition $\mathcal{C}_d$ in (5) as follows. The nodes of $\mathcal{G}$ represent the cells $c^j_{k,\ell}$ in $\mathcal{C}_d$ and the edges represent the connectivity relationship between those nodes. In this section we show that the connectivity of the graph $\mathcal{G}$ can be constructed directly from the wavelet coefficients. Equivalently, we compute the adjacency list of $\mathcal{G}$ directly from wavelet coefficients obtained from the FLWT.

Since the scaling function $\Phi_{j,k,\ell}$ and the wavelet functions $\Psi^i_{j,k,\ell}$ of the 2D Haar wavelet are associated with square cells $c^j_{k,\ell}$, the corresponding approximation and nonzero detail coefficients encode the necessary information regarding the cell geometry (size and location). Recall that the approximation coefficients are the average values of the risk measure value over the cells, and the detail coefficients determine the size of each cell. To this end, consider a cell $c^{j_0}_{k,\ell}$ at level j_0, whose dimension is $1/2^{j_0} \times 1/2^{j_0}$ and is located at (k, ℓ). A cell will be called *independent* if it is associated with a non-zero approximation coefficient $a_{j_0,k,\ell}$, while the corresponding detail coefficients $d^i_{j,k,\ell}$ $(i = 1, 2, 3)$ at level $j_0 \le j \le J_{\max}$ are all zero. Otherwise, the cell is marked as a *parent* cell, and is subdivided into four *leaf* cells at level $j_0 + 1$. If a leaf cell cannot be subdivided further, it is classified as an independent cell. In Fig. 2, the top-most parent cell $c^{j_0}_{k,\ell}$ is subdivided into three independent cells at level $j_0 + 1$ with each non-zero approximation coefficient in the quadrant I, II, and III (all zero detail coefficients). For quadrant IV, the cell is further subdivided into four independent leaf cells at level $j_0 + 2$.

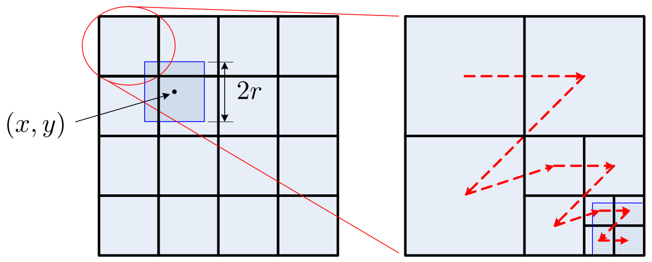


Fig. 3. Recursive raster scan method for identifying independent cells.

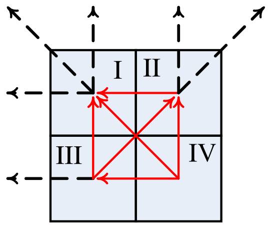


Fig. 4. Basic connectivity properties with respect to the location of the leaf cell.

Assume now that we are given the Haar wavelet transform of the risk measure function rm up to the level $J_{\min}$. The coarsest level of the cell dimension is set to $J_{\min}$. In Fig. 3 the initial coarse grid is drawn on the left. The agent is located at $\mathsf{x} = (x, y)$ and the high resolution horizon is given by r. Recalling expressions (3), we distinguish cells at distinct resolution levels, by starting from a coarse cell $c^{j_0}_{k,\ell}$, and by determining if the cell either partially intersects or totally belongs to the set $\mathcal{N}(\mathsf{x}, r)$. The cell $c^{j_0}_{k,\ell}$ is easily ascertained to satisfy this property by choosing the indices such that $(k,\ell) \in (\mathcal{K}(j_0), \mathcal{L}(j_0))$. If the cell needs to be subdivided into higher resolution cells, the inverse fast lifting wavelet transform is first performed on the current cell (local reconstruction) in order to recover the four approximation coefficients at level $j_0 + 1$ and the corresponding detail coefficients. We then adopt the raster scan method(zigzag search: I→II→III→IV) to examine each cell inside the parent cell overlapping with $\mathcal{N}(\mathsf{x}, r)$. This procedure is recursively repeated until the maximum resolution level $J_{\max}$ is reached. Figure 3 illustrates the recursive raster scan search. Once a cell is recognized as independent, we assign a node in the graph $\mathcal{G}$ with the node cost being the approximation coefficient representing the average risk measure over the cell. In addition, the detail coefficients associated with the current cell are all set to zero; this will provide the necessary connectivity information between the cells later on.

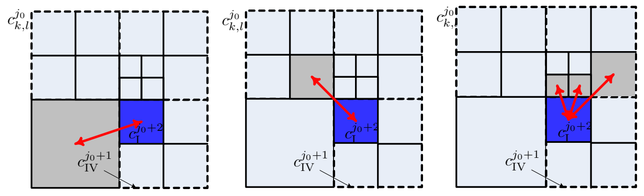
After a cell has been identified as an independent cell, we search the adjacent cells in order to establish the adjacency relationship with the current cell. Recall that two cells c_i and c_j are adjacent if their boundaries have common points. For our case of square cells, this implies that two cells are adjacent only along the following eight directions: Left, top, right, bottom, and the four diagonal directions. Following the recursive raster search for cell identification, the adjacency search requires establishing links between two cells that have been identified as independent cells. Recalling that the raster search progresses from left to right and from top to down (zigzag progress) as illustrated in Fig. 3, we confine the adjacency search to the following directions: Left, top-left, top, and top-right from the current cell. By doing this, we render half of the links (for eight cell connectivity) to be connected from the current cell, and the remaining links with the current cell will be connected as the recursive raster scan progresses to the next cells. In addition, because we deal with cells of different dimensions, it is required to devise a generic method to find the adjacency relationship between the cells.

Figure 4 illustrates the basic search direction of each leaf cell inside a parent cell. The dashed arrow points towards an external search region, that is, an adjacent cell could be found beyond the parent cell, whereas the solid arrow points towards an internal search region that belongs to the parent cell. In each search, we implicitly assume that the level of adjacent cells may vary from that of the parent cell to $J_{\max}$ (external connection), or from that of the current cell to $J_{\max}$ (internal connection).

A leaf cell inherits the search region from its parent cells, whose search direction turns out to be one of the solid arrows in Fig. 4. Figure 5(a) shows this inheritance property. In Fig. 5(a) the current cell is chosen to be $c^{j_0+2}_{\mathrm{I}}$. This cell is a leaf cell of the parent cell $c^{j_0+1}_{\mathrm{IV}}$, which further becomes a leaf cell of the top-most parent cell $c^{j_0}_{k,l}$. The cell $c^{j_0+1}_{\mathrm{IV}}$ is located on the fourth quadrant inside the top-most parent cell $c^{j_0}_{k,l}$ so that the search region for $c^{j_0+1}_{\mathrm{IV}}$ ends up with the internal searches at the level $j_0 + 1$, whose adjacency search property is inherited to the cell $c^{j_0+2}_{\mathrm{I}}$ for left, top-left, and top direction searches. Having ascertained the basic search directions, we refine the adjacent search looking for opposite cells which must be independent and adjacent to the current cell. Because the opposite cells of the current cell could have different dimensions, as depicted in Fig. 5(a), we establish links by examining the associated detail coefficients of the opposite cells. Along the left search direction of $c^{j_0+2}_{\mathrm{I}}$, as illustrated in Fig. 5(a), one finds that only one independent cell at level $j_0 + 1$ is linked to $c^{j_0+2}_{\mathrm{I}}$.

The adjacency search algorithm refines its search to the higher levels if the opposite cell is not an independent cell, that is, if it is comprised of finer cells. This refinement subsequently forces a search of cells of the finer dimension (level) which are neighboring to the current cell. Subsequently, the detail coefficients of the opposite cells are examined in order to find the next finer cell that is adjacent to the current cell. For the top-left search direction of $c^{j_0+2}_{\mathrm{I}}$, as illustrated in Fig. 5(b), the search process initially examines the cell $c^{j_0+1}_{\mathrm{I}}$ located at the top-left corner of the current cell through the corresponding detail coefficient. Provided that the detail coefficient associated with the cell $c^{j_0+1}_{\mathrm{I}}$ takes a non-zero value, the cell is not an independent cell. Subsequently, the cell $c^{j_0+1}_{\mathrm{I}}$ is subdivided and the search process repeats at level j_0+2 when the opposite cell to the current cell becomes an independent adjacent cell. In Fig. 5(b), since there exists no other independent cells along the top-left direction except the shaded one, a bidirectional link is established between the current and the opposite cells.

Similarly, for the top search direction, two cells at level $j_0 + 3$ and one at level $j_0 + 2$ are found to be independent and adjacent to the current cell. The bidirectional links are accordingly connected from the current cell $c^{j_0+2}_{\mathrm{I}}$ to those



(a) Search left (b) Search top-left (c) Search top

Fig. 5. Adjacency search algorithm with the recursive refinement.

adjacent cells. Figure 5(c) depicts this situation.

B. Cost assignment

We associate each node $v \in \mathcal{G}$ to a cell $c^j_{k,\ell} \in \mathcal{C}_d$. Moreover, since $\mathcal{G}$ is a topological graph, we may associate each node v with some point $\mathsf{x} \in c^j_{k,\ell}$. Without loss of generality, we choose the center of the cell denoted by $\mathsf{cell}_{\mathcal{G}}(v)$. If $\mathsf{x} \in c^j_{k,\ell}$ we will write $v = \mathsf{node}_{\mathcal{G}}(\mathsf{x})$. To each directed edge (u, v) of $\mathcal{G}$ we assign an edge cost, given as

$$\mathcal{J}(u, v) = \mathsf{rm}(\mathsf{cell}_{\mathcal{G}}(v)) + \alpha\|\mathsf{cell}_{\mathcal{G}}(u) - \mathsf{cell}_{\mathcal{G}}(v)\|_2, \quad (6)$$

where $\alpha \geq 0$ is a weight constant. The first term in (6) is proportional to the probability that the target node is close to an obstacle, while the second term penalizes the (Euclidean) distance between $\mathsf{cell}_{\mathcal{G}}(u)$ and $\mathsf{cell}_{\mathcal{G}}(v)$. The estimate

$$h(v) = \|\mathsf{cell}_{\mathcal{G}}(v) - \mathsf{cell}_{\mathcal{G}}(v_f)\|_\infty, \quad (7)$$

where $v_f = \mathsf{node}_{\mathcal{G}}(\mathsf{x}_f)$ is also used for the distance of the current node to the goal node. Given the graph $\mathcal{G}$ along with the associated edge and node costs, we then invoke the $\mathcal{A}^*$ algorithm [11] to find a path that minimizes the cumulative cost from the start to goal nodes (or determine that such a path does not exist).

IV. Multiresolution Path Planning

A. Multiresolution path planning algorithm

Starting from $\mathsf{x}(t_0) = \mathsf{x}_0$ at time $t = t_0$, the proposed multiresolution path planning algorithm constructs at each time t_i $(i = 0, 1, \ldots)$ a multiresolution decomposition $\mathcal{C}_d(t_i)$ of $\mathcal{W}$ around the current location of the vehicle, $\mathsf{x}(t_i)$, along with the corresponding graph $\mathcal{G}(t_i)$. Repeated execution of $\mathcal{A}^*$ at each iteration step yields a sequence of nodes $\{\mathsf{node}_{\mathcal{G}(t_0)}(\mathsf{x}(t_0)), \mathsf{node}_{\mathcal{G}(t_1)}(\mathsf{x}(t_1)), \cdots, \mathsf{node}_{\mathcal{G}(t_f)}(\mathsf{x}(t_f))\}$.

The process terminates at some time t_f when $\|\mathsf{x}(t_f) - \mathsf{x}_f\|_\infty < 1/2^{J_{\max}}$. At the last step the agent moves from $\mathsf{x}(t_f)$ to x_f. The details of the algorithm, along with a pseudo-code implementation can be found in [5].

B. $\mathcal{D}^*$-lite path planning algorithm

$\mathcal{D}^*$ is a popular algorithm for solving path planning problems in unknown or partially known environments, originally proposed by Stentz [14]. The $\mathcal{D}^*$-lite algorithm [10], which implements the same planning strategy as $\mathcal{D}^*$, adopts an efficient incremental search to reduce the time required to replan.

Below we employ the $\mathcal{D}^*$-lite algorithm, and compare the results with those obtained using the proposed multiresolution algorithm. We discuss the benefits and shortfalls between the two approaches. In order to be able to do a sensible comparison, we assume that the agent is equipped only with a proximity sensor that senses the environment close to its current location with high accuracy. In other words, we deliberately incorporate a unknown environment with incremental updates in order to constructively invoke the $\mathcal{D}^*$-lite algorithm. The reason for this is attributed by the fact that with an assumption of prior information of the unchanging environment, the $\mathcal{D}^*$-lite is nothing but $\mathcal{A}^*$, which may be inappropriate to compare with the proposed algorithm.

Let $\mathcal{S}(i)$ be the known region up to $t = t_i$ using a sensor with the range r_J, defined by

$$\mathcal{S}(i) = \mathcal{S}(i-1) \cup \mathcal{N}(\mathsf{x}_i, r_J) \quad (8)$$

where x_i is the current location of the agent at $t = t_i$ and $\mathcal{N}(\mathsf{x}_i, r_J)$ represents the effective sensory region at that moment. In Eq. (8) it is assumed that the agent navigates an initially unknown environment while updating the map from the collected information. In order to take this process into consideration for replanning, we assign a conditional cost to each edge (u, v), which depends on the relative location of the edges to $\mathcal{S}$ as follows,

$$\mathcal{J}(u, v) = \begin{cases} \mathsf{rm}(\mathsf{cell}_{\mathcal{G}}(v)) + \|\mathsf{cell}_{\mathcal{G}}(u) - \mathsf{cell}_{\mathcal{G}}(v)\|_2, & u, v \in \mathcal{S}, \\ \|\mathsf{cell}_{\mathcal{G}}(u) - \mathsf{cell}_{\mathcal{G}}(v)\|_2, & u, v \in \mathcal{W} \setminus \mathcal{S}. \end{cases} \quad (9)$$

For the edges outside $\mathcal{S}$ we simply impose the traversal cost between nodes owing to the uniform size of cells. If this is the case, the $\mathcal{A}^*$ search algorithm simply computes a shortest path from an initial node to the goal node which might pass through obstacles outside $\mathcal{S}$. Nevertheless, whenever the map is updated using contingent information from the sensor, we accordingly update the corresponding edge costs by appending the obstacle cost to each edge as given in (9).

V. Hardware in-the-loop Simulation Results

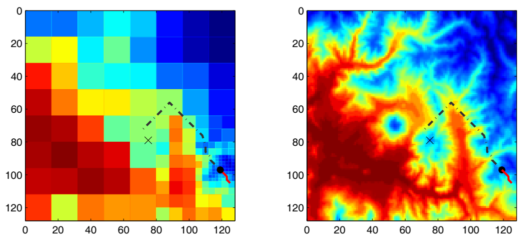
A. Hardware overview

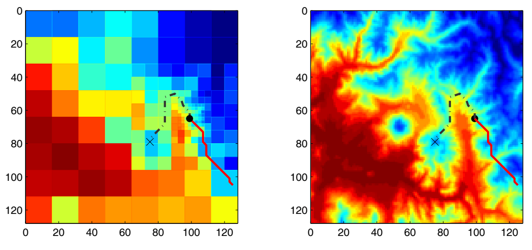
A UAV autopilot platform has been developed to implement the multiresolution, wavelet-based path planning algorithm described above. The on-board autopilot is based on the Rabbit RCM-3400 micro-controller (30 MHz with 512 KB RAM) and provides data acquisition, processing, and communication with the ground station, in addition to all UAV control functions. Further details about the UAV platform, autopilot and HILS set-up can be found in [6] and [7].

B. Simulation results using the proposed algorithm

In this section we present simulation results of the proposed algorithm for a non-trivial scenario. The environment $\mathcal{W}$ is an actual topographic (elevation) map of a US state, shown in Fig. 6. The environment is assumed to be square of dimension 128×128 units. Taking into account the available memory of the micro-controller, we choose the fine level as $J_{\max} = 6$ and the coarse level as $J_{\min} = 3$. This makes the total number of nodes in the graph not to exceed the maximum count of 256 that corresponds to the maximum allowable variable size of the micro-controller. The ranges from the current location at distinct levels of resolution are selected as $(r_6, r_5, r_4) = (8, 15, 30)$.

The results from the multiresolution path planning algorithm are shown in Fig. 6. Specifically, Fig. 6 shows the evolution of the path at different time steps as the agent moves to the final destination. In each figure, solid lines represent the actual path followed by the agent, while dashed-dot line represent the best proposed path to the destination at each instant. As seen in Fig. 6(c), the actual path followed by the agent differs from the one predicted in either Figs. 6(a) or 6(b). This is due to the fact that at time t_5 and t_{21} the agent does not have complete information for the future positions up to the high-resolution, confident level. However, as the agent gets closer to the obstacle as shown in Fig. 6(b), it recognizes the presence of obstacles and redirects itself to avoid any obstacles. Note that the path is represented by a finite sequence of square cells, hence in order to actually control the UAV to track the path, the proposed algorithm needs to be implemented in conjunction with the path generation [9], path following [8] algorithms.


(a) $t = t_5$


(b) $t = t_{21}$

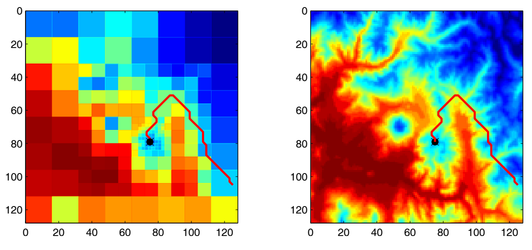

(c) $t = t_f$

Fig. 6. Path evolution and replanning. Figures on the left show the multiresolution approximation of the environment with respect to the current location of the agent.

C. Simulation results for the $\mathcal{D}^*$-lite algorithm

In this section we present simulation results of the $\mathcal{D}^*$-lite path planning algorithm for the same environment used in the previous section. We adopt a uniform cell decomposition of cell size $J_{\max} = 6$, which is the same as the finest level of the previous section. The range of the proximity sensor is chosen to be $r_6 = 7$, thus resulting in the high resolution window by 7 by 7 square grids.

The result from the $\mathcal{D}^*$-lite algorithm is shown in Fig. 7. The solid line represents the actual path followed by the agent, which shows that the $\mathcal{D}^*$-lite algorithm effectively replans the entire path circumventing the obstacles, reaching the final destination. It should be noted that we deliberately implement the $\mathcal{D}^*$-lite algorithm using an unknown environment. By the incremental updates of the path whenever the agent detects the obstacles, the $\mathcal{D}^*$-lite algorithm shows reactive behavior to circumvent the obstacles. This may be improved by incorporating coarse information (a priori known) outside $\mathcal{S}$, similarly to the proposed algorithm. However, since the $\mathcal{D}^*$-lite algorithm solves the path over a uniform grid, it turns out that extra computational overhead is necessary to compute such coarse information. In this paper, and for proper comparison, we avoid this extra computation between the proposed algorithm and the $\mathcal{D}^*$-lite algorithm.

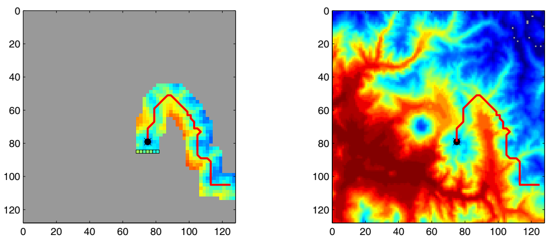


Fig. 7. Path evolution and replanning using the $\mathcal{D}^*$-lite algorithm.

VI. Comparison

The proposed multiresolution path planning algorithm was written in C code and implemented on the on-board autopilot described earlier. Because the micro-controller used in the autopilot has limited computational resources, the code was written by giving special attention not only to the accuracy of the results, but also to the expected computational speed during implementation.

We compared the computational costs between the proposed multiresolution path planning algorithm and the $\mathcal{D}^*$-lite algorithm for several cases. Because it was not possible to implement $\mathcal{D}^*$-lite on the autopilot due to memory limitations, all simulations were carried out using an IBM-PC (Pentium M 2.0 GHz, 1 GB RAM), based on codes written in C for implementing both path planning algorithms. The proposed wavelet-based multiresolution path planning algorithm accomplishes the objective of reaching the goal using a fewer number of iterations than the $\mathcal{D}^*$-lite algorithm, as shown in Table I. This is due to the fact that the proposed algorithm effectively manages the information at coarse resolutions so as to compute a preferred path early on. The $\mathcal{D}^*$-lite algorithm, on the other hand, typically requires more iterations to reach the goal than the proposed algorithm, since the agent is required to explore the environment and to replan the path along the movement of the agent.

The $\mathcal{D}^*$-lite algorithm is typically faster, although the performance of the proposed algorithm can be improved by using, say, four-connectivity instead of eight-connectivity during the adjacency search algorithm. This will possibly halve the computation time with little performance degradation. Additional improvements can be expected by judiciously reusing prior information at each iteration. We are

TABLE I

THE COMPUTATIONAL COST COMPARISON BETWEEN THE MULTIRESOLUTION PATH PLANNING V/S THE $\mathcal{D}^*$-LITE.

Items	Scenario I		Scenario II		Scenario III		Scenario IV		Scenario V	
	$\mathcal{D}^*$lite	Wavelet	$\mathcal{D}^*$lite	Wavelet	$\mathcal{D}^*$lite	Wavelet	$\mathcal{D}^*$lite	Wavelet	$\mathcal{D}^*$lite	Wavelet
# iteration	35	31	93	50	61	40	123	52	44	43
# nodes in $\mathcal{G}$	4096	201	4096	194	4096	192	4096	185	4096	194
Data processing [msec]	2.03	2.03	2.03	2.03	2.03	2.03	2.03	2.03	2.03	2.03
Adjacency search [msec]	-	0.977	-	0.987	-	0.969	-	0.94	-	0.958
$\mathcal{A}^*$ search [msec]	-	0.1	-	0.125	-	0.094	-	0.138	-	0.066
Init. $\mathcal{D}^*$-lite search [msec]	1.87	-	2.03	-	2.03	-	1.87	-	2.03	-
$\mathcal{D}^*$-lite update [msec]	4.1	-	23.8	-	11.3	-	43.9	-	12.7	-
Total Comp. time [msec]	5.97	33.387	25.83	55.6	13.33	42.53	45.77	56.056	14.73	44.032
Computational cost (%)	17.8	100	46.46	100	31.35	100	81.65	100	33.45	100
Memory cost (%)	2037.8	100	2111.3	100	2133.3	100	2214.1	100	2111.3	100

currently working on the latter idea; for some preliminary results, see [3]. On the other hand, the proposed algorithm requires much less memory as shown in Table I, when compared to $\mathcal{D}^*$-lite. In fact, for our problem, it was not possible to implement $\mathcal{D}^*$-lite on the microprocessor due to memory limitations. For on-line, on-board path planning, the proposed algorithm has therefore an advantage in terms of scalability according to the available on-board computational resources.

VII. CONCLUSIONS

Autonomous path planning for small UAVs imposes severe restrictions on control algorithm development, stemming from the limitations imposed by the on-board hardware, and the requirement for on-line implementation. In this work we have proposed a method to overcome this problem by using a new hierarchical, multiresolution path planning scheme. The algorithm computes at each step a multiresolution representation of the environment using the wavelet transform. The idea is to employ high resolution close to the agent (where is needed most), and a coarse resolution at large distances from the current location of the agent. The connectivity relationship of the resulting cell decomposition can be computed directly from the nonzero detail coefficients of the wavelet transform. The algorithm is scalable and can be tailored to the available computational resources of the agent.

ACKNOWLEDGMENT

Partial support for this work was provided by NSF award CMS-0510259.

REFERENCES

[1] S. Behnke, "Local Multiresolution Path Planning," in *RoboCup 2003: Robot Soccer World Cup VII*, vol. 3020 of *Lecture Notes in Computer Science*, pp. 332–343, Berlin: Springer, 2004.

[2] C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms*, Upper Saddle River, New Jersey: Prentice Hall, 1998.

[3] R. Cowlagi and P. Tsiotras, "Multiresolution Path Planning with Wavelets: A Local Replanning Approach," in *American Control Conference*, (Seattle, WA), 2008.

[4] P. S. Heckbert and M. Garland, "Multiresolution Modeling for Fast Rendering," in *Proceedings of Graphics Interface*, (Banff, Canada), pp. 43–50, May 1994.

[5] D. Jung, *Hierarchical Path Planning and Control of a Small Fixed-Wing UAV: Theory and Experimental Validation*, Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, Dec. 2007.

[6] D. Jung, E. J. Levy, D. Zhou, R. Fink, J. Moshe, A. Earl, and P. Tsiotras, "Design and Development of a Low-Cost Test-Bed for Undergraduate Education in UAVs," in *Proceedings of the 44th IEEE Conference on Decision and Control*, (Seville, Spain), pp. 2739–2744, Dec. 2005.

[7] D. Jung and P. Tsiotras, "Modelling and Hardware-in-the-loop Simulation for a Small Unmanned Aerial Vehicle," in *AIAA Infotech at Aerospace*, (Rohnert Park, CA), May 2007. AIAA Paper 07-2763.

[8] D. Jung and P. Tsiotras, "Bank-To-Turn Control for a Small UAV using Backstepping and Parameter Adaptation," in *International Federation of Automatic Control (IFAC) World Congress*, (Seoul, Korea), July 2008. to appear.

[9] D. Jung and P. Tsiotras, "On-line Path Generation for Small Unmanned Aerial Vehicles Using B-Spline Path Templates," in *AIAA Guidance, Navigation and Control Conference*, 2008. submiited.

[10] S. Koenig and M. Likhachev, "$\mathcal{D}^*$ Lite," in *Proceedings of the National Conference of Artificial Intelligence*, pp. 476–483, 2002.

[11] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[12] H. Noborio, T. Naniwa, and S. Arimoto, "A Quadtree-Based Path-Planning Algorithm for a Mobile Robot," *Journal of Robotic Systems*, Vol. 7, No. 4, pp. 555–574, 1990.

[13] D. K. Pai and L.-M. Reissell, "Multiresolution Rough Terrain Motion Planning," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 1, pp. 19–33, 1998.

[14] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3310–3317, May 1994.

[15] P. Tsiotras and E. Bakolas, "A Hierarchical On-Line Path Planning Scheme using Wavelets," in *Proceedings of the European Control Conference*, (Kos, Greece), July 2007.

[16] J. Vörös, "Low-cost Implementation of Distance Maps for Path Planning using Matrix Quadtrees and Octrees," *Robotics and Computer Integrated Manufacturing*, Vol. 17, pp. 447–459, 2001.