Data Structure for Efficient Processing in 3-D

Jean-François Lalonde, Nicolas Vandapel and Martial Hebert
Carnegie Mellon University
{jlalonde,vandapel,hebert}@ri.cmu.edu

***Abstract*— Autonomous navigation in natural environment requires three-dimensional (3-D) scene representation and interpretation. High density laser-based sensing is commonly used to capture the geometry of the scene, producing large amount of 3-D points with variable spatial density. We proposed a terrain classification method using such data. The approach relies on the computation of local features in 3-D using a support volume and belongs, as such, to a larger class of computational problems where range searches are necessary. This operation on traditional data structure is very expensive and, in this paper, we present an approach to address this issue. The method relies on reusing already computed data as the terrain classification process progresses over the environment representation. We present results that show significant speed improvement using ladar data collected in various environments with a ground mobile robot.**

I. Introduction

Autonomous ground robot navigation in outdoor natural environment, specifically in the presence of vegetation, requires advanced three-dimensional perception capabilities [13], [2]. Cross-country navigation, like in a forest or meadow, introduces new challenges compared to road following or desert/planetary environment traverses. In the latter case, elevation maps built from 3-D data suffice to represent and reason about the robot surroundings. In the former case, however, a full 3-D internal representation is necessary to accommodate the complexity of the scene that contains porous material (vegetation), thin structures (branches, wires) and overhanging structures (tree canopy). Figure 1 represents an example of scene considered[1]. Note the point density variation and the presence of the structures mentioned above.

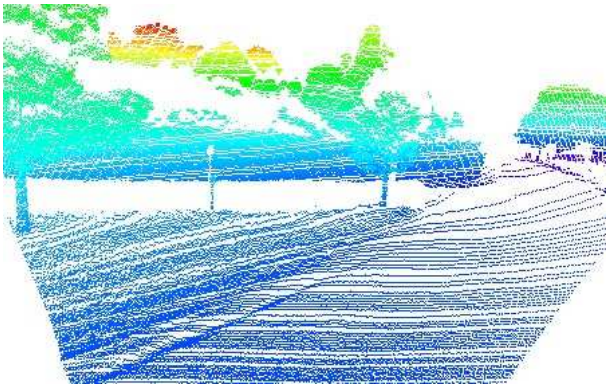


Fig. 1. Example of terrain considered. The color of each point is based on the elevation, with red (blue) representing high (low) elevation.

[1]The figures in this paper are best viewed in color

Recent advances in sensor design have enabled the use of laser radars that provide tens of thousands of 3-D points per second, even a hundred thousand, with centimeter range resolution. The problem now is how to handle such a large point cloud and how to design the data flow from the sensor to the environment interpretation. One critical aspect is to be able to perform quickly basic operations such as insertion, access, and range search. Traditional optimal tree-based data structures are ill-suited for dynamic data sets because of numerous insertions produced by a moving robot in an outdoor environment.

In this paper, we present a new approach for handling 3-D data for efficient on-board processing. The data processing we are concerned with are kernel-based methods, where for a given point, some operations are performed using a support volume. The core of the approach is to minimize computation by re-using pre-computed intermediate results. The approach is demonstrated with data from a ground mobile robot, the Demo III XUV, for ladar-based terrain classification [14].

The rest of the paper is divided into four sections where we present: the state of the art in data structures, specifically for robot navigation; our approach, with a complexity and memory analysis; results from static ground robot, and the conclusion.

II. State of the art

In this section we review recent work on 3-D data structure for ground mobile robot. We also look at traditional data structures.

A. Robotics

Three-dimensional data have been used for a long time for outdoor robot navigation, initially from stereo camera then from laser radar. If the terrain is unobstructed, one common approach is to create a 2D grid of the terrain with the terrain classification results. The data processing can take place in the sensor reference frame (range image) and the results are then back-projected in that 2D grid. An alternative is to create an intermediate digital elevation map by gridding the 3-D data into a 2D-1/2 map and then doing some processing, convolving a robot model with the terrain for example [13]. In both cases however, the data processing is not performed in 3-D.

If the terrain contains vegetation such as trees, grass or bushes, the previous approach is not sufficient. A full 3-D representation is necessary to represent the environment and to produce a better and higher level of scene interpretation. One such approach has been demonstrated for vegetation detection and ground surface recovery in [6], [2]. In both cases, a

dense 3-D grid representation of the environment is maintained around the robot and scrolled as it moves. A ray tracing algorithm updates each voxel by counting the number of times it has been traversed by or stopped a laser ray. Such statistics are then used to determine if the voxel is likely to be the load bearing surface or vegetation. The data processing requires data insertion and retrieval, but no range search.

Similarly, 3-D occupancy grid approaches create a voxelized 3-D model by performing insertion and access but are not optimized for range search [9].

Sometimes operations in 3-D can be reduced to 2-D operations as shown in [11] for natural environment navigation. Unfortunately, in general, we cannot follow such an approach.

B. Data structures

Traditional pre-computed tree-based data structures (Kd-tree, range tree) are efficient for performing range search. Unfortunately, their performance degrades rapidly as additional data are inserted after construction [10]. Lersch in [7], presents a data structure for structural segmentation of 3-D point-cloud data, called the windowed priority queue. The approach focuses on the indexing of the data for fast retrieval. The computation performed is similar to the one used in our work [14] but it is performed off-line. Approximate search is sometimes proposed, but we did not consider it in order to maintain the necessary classification rate.

Gao proposes an interesting work on efficient proximity search in 3-D for kinetic data [4]. The author extends Voronoi diagram and Delaunay triangulation to an environment made of 3-D voxels. A simple example is provided. It is not clear how we can efficiently scale this approach to handle the point density and grid resolution in our context.

Machine learning and statistical methods require efficient data structures for nearest neighbor search, range search, regression or kernel operations [8], [5]. But most of the attention is focused on high dimensional data set rather than dynamic data set.

III. Nature of the computation

A. Terrain classification

Using 3-D ladar data as input, we perform point-wise classification to detect vegetation, thin structures and solid surfaces. The method relies on the use of the scatter matrix to extract features via principal component analysis. For each point, the approach computes the scatter matrix within a support volume and then extracts its principal component (eigenvalues). A linear combination of the components and the associated principal directions define the features. A model of the features distribution is learned off-line, prior to the mission, from labeled data. As the robot traverses a terrain, data are accumulated, features computed and maximum likelihood classification performed on-line. Figure 2 presents an example of such terrain classification. For additional details please see [14].

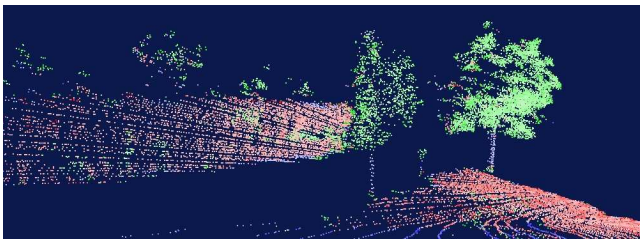


Fig. 2. Example of terrain classification. In red/green/blue:surfaces, scatter volumes and linear structures.

B. On-board robot implementation

In order to handle a hundred thousand points per second, we previously implemented a data structure that reduces dramatically the amount of data to handle without compromising the features computation. The environment is represented as a set of sparse voxels that contain intermediate results used to compute the scatter matrix. It is straightforward to see that the scatter matrix can be decomposed in terms of sums, sums of squared and sums of cross products of 3-D point coordinates. These results are simply nine numbers. For each new point falling into a voxel, the nine numbers are updated. To compute the saliency features for a given point, range search is performed and the neighboring voxels results are summed up together to compute the scatter matrix. The voxels are accessed via a hash-map. Thus, with such a data structure, we significantly compress the data while keeping all information, regardless of the performed computation.

The computations we perform belong to a general class of processing that requires the retrieval and the use of the data within a support volume around a point of interest. We introduce here some notations

- $PCD = \{p_1, p_2, ..., p_m\}$: point cloud data, a set of m 3-D points;
- $p_k = (x_k, y_k, z_k)$: a 3-D point in PCD;
- $N(p_i, s)$: the set of points around a given point p_i, within a volume of radius or scale s, such that $\| p_i - p_k \|_\infty \leq s$ with $i \neq k$;
- $F(N(p_i, s))$: a function over the neighborhood points.

In our case $F = \sum_{p \in N(p_i,s)} (p - \bar{p})^T (p - \bar{p})$. The work presented in [12] belongs to the same class of computation, but the objective is to extract simultaneously multiple planar structures using projection-based regression. The function F takes the form

$$F(N(p_i, s)) = \frac{1}{|N(p_i, s)| h_\theta} \sum_{i=1}^{|N(p_i,s)|} \kappa(\frac{y_i^T \theta - p_i}{h_\theta})$$

with κ a kernel function scaled to the bandwidth h_θ. We refer the reader to [12], [1] for additional details.

This approach has been tested extensively on-board a ground robot. The processing time on the current hardware allows operation at slow speed (1-2 m/s) with a hundred thousand input points per second, depending on the complexity of the terrain. The motivation behind our recent work is

to increase the processing speed to handle higher terrain representation resolution and faster robot navigation speed.

IV. Approach

A. General principle

Real-time area-based correlation stereo algorithms reuse previously-computed data to achieve greater execution speed. For example, in [3], Faugeras et al. decompose the zero mean normalized correlation score into partial sums, and add/remove only columns contribution as the epipolar line is scanned. A similar approach is used to handle change of line by removing/adding line contribution at the image borders. Since the nature of the computation needed for feature extraction is analogous to correlation (i.e. it can be decomposed into partial sums), we apply that principle to a voxel representation in 3-D.

However, there are two fundamental differences between the 2-D and 3-D cases that justify the need for a novel approach. First, in stereo, correlation is performed along the epipolar line, resulting in a unidirectional scanning. However, many different strategies exist to scan the 3-D space (obtained by permuting the order of axes). Figure 3 illustrates two examples. Second and most importantly, full 3-D data is usually very sparse, that is, a large number of voxels are empty. This contrasts with images in which each pixel contain information that can easily be retrieved in a subsequent step.

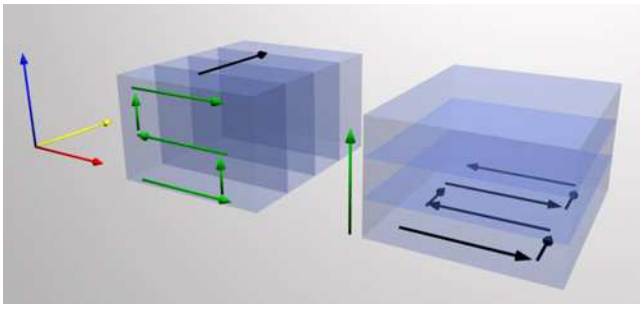


Fig. 3. Two possible scanning strategies in 3-D

First, we introduce some notation. Given a volume V subdivided into voxels, let:

- $n = n_x \times n_y \times n_z$ the total number of voxels in the volume;
- v be the number of occupied (non-empty) voxels in the volume;
- $k = (2r+1)$ the neighborhood size (eg. for range search), where r is the radius, in number of voxels

We derive the following equations to compute the number of voxels that need to be visited for range search computation. We do not take into account the differences that may arise at the boundaries of the volume: for a large volume, they are assumed to be negligible. We also assume that k is the same in each dimension.

B. Brute force approach

1) Description: The implementation presented in Section III-B, relies on a brute force approach [14] that visit the whole k^3 neighborhood each time: a hash map is used as a data structure and occupied voxels are scanned in random order.

2) Analysis: The total number of visited voxels is simply

$$t_{brute} = vk^3 \tag{1}$$

This method takes advantage of the sparseness of the data: only occupied voxels are visited. However, on the dense regions, much of the computations are repeated many times because neighborhoods overlap.

C. Naive approach

1) Description: A naive approach would be to act as if the volume was densely populated, that is, to scan the whole volume in an ordered way while executing neighborhood computation for every voxel, even the empty ones. This is the direct translation of the 2-D approach using 3-D data.

An appropriate data structure is the dense voxel representation, in which memory space is reserved for each voxel of the volume at the beginning. The drawback of this choice is that memory usage is mostly inefficient, thus limiting the volume of interest. It is, however, much faster since it does not require hash key computation, and allows ordered traversal of the volume.

2) Analysis: Since it re-uses previously-computed data and recomputes neighborhood slices each time, the number of visited voxels at each step is $2k^2$. Since it is done over the whole volume, $n = v$, and the total number of visited voxel is

$$t_{naive} = 2nk^2 \tag{2}$$

This method does not take advantage of the sparseness of the data, and must scan the whole volume each time. The condition for t_{naive} to be less than t_{brute} is :

$$t_{naive} < t_{brute} : 2nk^2 < vk^3$$

$$\frac{v}{n} > \frac{2}{k}$$

If $k = 9$, then $v/n > 0.23$. At least 23% of the voxels in the volume must be valid for the naive method to be faster than the brute force. As our experimental results show (Section V), the v/n ratio tends to be very low, typically under 2%, justifying the need for a better approach.

D. Memory-based approach

1) Description: This method takes advantage of the dense regions in the volume. The principle is the same as in Section IV-C, but the computations are done only on the occupied voxels. Therefore, this algorithm will need to find the *previous occupied voxel* and see if it is close enough, that is, if the distance between the two is less than $k/2$. This concept

is related to the volume traversal order because it directly depends on the scanning direction.

To help formalize the problem, we follow a two-step procedure: first we assume a constant cost independent of the number of visited voxels so we can derive easily a lower bound on the $\frac{v}{n}$ ratio, second we determine the expression for this cost.

2) Simplified lower bound estimation: Since this approach requires the voxels to be scanned in predetermined order, let d be the distance (in number of voxels) between the current voxel and the previous occupied voxel in the volume, along the scanning direction. Let X be the discrete random variable representing d. The distribution of X depends only on the data.

For the sake of simplicity, we first assume that, for a given voxel v_p, the range search computation will require visiting αk^2 voxels if $d < \frac{k}{2}$, and k^3 otherwise, with constant α. Moreover, since every voxel in the whole volume must be analyzed, a cost of n must be added. t_{memory} is computed as the expected cost:

$$\begin{aligned} t_{memory} &= v\left(\alpha k^2 P[X < \frac{k}{2}] + k^3 P[X \geq \frac{k}{2}]\right) + n \\ &= v\left(\alpha k^2 P[X < \frac{k}{2}] + k^3 (1 - P[X < \frac{k}{2}])\right) + n \\ &= v\left((\alpha k^2 - k^3) P[X < \frac{k}{2}] + k^3\right) + n \end{aligned}$$

The condition for t_{memory} to require less operations than t_{brute} becomes:

$$v\left((\alpha k^2 - k^3) P[X < \frac{k}{2}] + k^3\right) + n < vk^3$$

$$\frac{v}{n} > \frac{1}{(k^3 - \alpha k^2)\, P[X < \frac{k}{2}]}$$

For example, if half of the occupied voxels are located within less than $\frac{k}{2}$ distance from the previous occupied voxel (that is, $P[X < \frac{k}{2}] = 0.5$), the condition becomes

$$\frac{v}{n} > \frac{1}{0.5k^3 - \alpha k^2}$$

If $k = 9$ and $\alpha = 1$ then $\frac{v}{n} > 0.00352$. In that case, at least 0.3% of the voxels in the volume must be valid for this method to visit fewer voxels than the first.

3) Lower bound estimation: Now, we relax the previous assumption and consider the general case where the number of visited voxels depends on d. Figure 4 illustrates this problem in 2-D. Since $d = 2$, the two rightmost columns are added to the neighborhood of the voxel at $(2, 2)$, and the two leftmost columns are subtracted. In total, four columns (instead of five) must be computed.

We define $\bar{d}$ as the expected value of d over those voxels for which $d < \frac{k}{2}$.

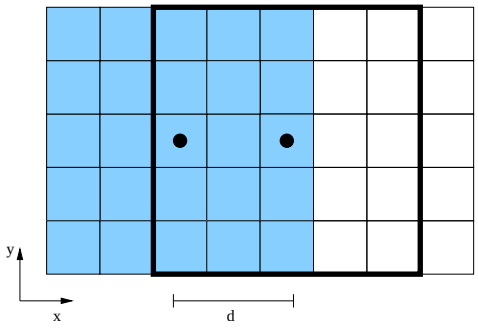


Fig. 4. Illustration of data reuse with sparse 2-D data with $k = 5$. Each square is a voxel, and dots indicate occupied voxels.

$$\bar{d} = \sum_{d=1}^{\frac{k}{2}-1} d \frac{P(X = d)}{P(X < \frac{k}{2})}$$

t_{memory}, in the general case, becomes

$$t_{memory} = v\left((2\bar{d}k^2 - k^3) P[X < \frac{k}{2}] + k^3\right) + n \quad (3)$$

In the worst case, $P[X < \frac{k}{2}] = 0$, Equation 3 becomes $t_{memory} = vk^3 + n$, which is equivalent to $t_{memory} = t_{brute} + n$. The only difference with the brute force method is the need to visit each voxel in the volume. The condition for t_{memory} to require less operations than t_{brute} becomes:

$$\frac{v}{n} > \frac{1}{\left(k^3 - 2\bar{d}k^2\right) P[X < \frac{k}{2}]} \quad (4)$$

As shown in Section V-B, Equation 4 is a lower limit on which we can guarantee faster execution.

E. Algorithm and data structure

The proposed algorithm is illustrated in pseudo-code by Algorithm 1. In this example, the scanning order is specified as input by the user, and the algorithm automatically determines in what direction it should look for re-usable data.

The data structure previously used in Section IV-C.1 is insufficient for the needs of Algorithm 1. We therefore propose a variant of the dense voxel representation that maintains an array of pointers to previously visited occupied voxels in memory. Figure 5 illustrates the principle in 2-D, but it is easily generalizable to higher dimensions. The 5×4 grid is the original dense voxel representation, and the two additional arrays are pointers to previously-visited, occupied voxels in memory. In this example, the scanning order is x then y, and the current voxel v_c is filled in blue, whereas the previously visited voxels are shown in a lighter shade of blue. The occupied voxels are dotted. The algorithm has access to the nearest previously visited occupied voxel just by looking at the cells in red, which correspond to the (x, y) position of v_c. v_p (position $(2, 0)$ in this example), can then easily be retrieved.

Algorithm 1 General scanning algorithm

Require: V the voxelized volume and its boundaries

for every occupied voxel in V **do**
  $v_c \leftarrow V(x, y, z)$, the current occupied voxel
  Retrieve d, the distance to the closest occupied voxel that has already been visited, and dir, the direction associated with it
  if $d \geq \frac{k}{2}$ **then**
    $n_c \leftarrow$ the whole k^3 neighborhood. {There's no sufficiently close occupied voxel}
  else
    Retrieve v_p, the previous occupied voxel located at distance d in direction dir of v_c
    $n_p \leftarrow$ stored neighborhood computation of v_p
    $s_p \leftarrow$ the d rightmost (along dir) slices of v_p
    $s_c \leftarrow$ the d leftmost (along dir) slices of v_c
    $n_c \leftarrow n_p - s_p + s_c$
  end if
end for

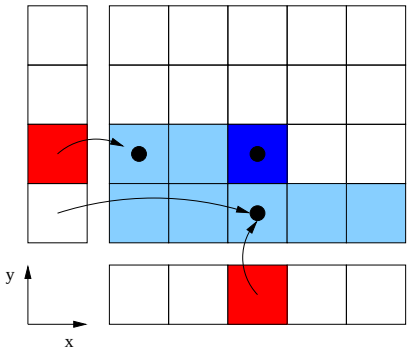


Fig. 5. 2-D example of the proposed data structure. Occupied voxels are dotted, and the interest voxel is blue. The previously visited voxels are drawn in a lighter shade of blue. The cells in red point to the previous valid result along their respective dimension.

V. Results

A. Implementation and data collection

We implemented a templatized version of Algorithm 1 and the corresponding data structure in C++ for Linux OS. In order for it to be efficient, we paid attention to basic rules such as not to perform costly run-time operations (polymorphism) and to access the data only using references. The code runs on a regular PC (Intel Xeon, 2.8 GHz, 1.5 GB RAM).

The data used in this paper were collected using the GDRS eXperimental Unmanned Vehicle [6]. This car-sized autonomous vehicle is equipped with a high-speed rugged range sensor that produces more than 100,000 3-D points per second with cm range resolution. The laser is mounted on a turret scanning the ground surface. The field tests were conducted at Fort Indiantown Gap in Pennsylvania in December 2004. Various terrain types were traversed including unstructured roads, forest and meadows.

B. Validation of theoretical results

To validate the derivation of Section IV-D, a set of synthetic random data is generated over a volume of interest with various uniform point density. We compute the speedup obtained by comparing the brute force method of Section IV-B with the optimal method of Section IV-D.3. The values of $\bar{d}$ and $P[X < \frac{k}{2}]$ are also computed to determine the lower bound predicted by Equation 4. Figure 6 shows the results obtained experimentally.

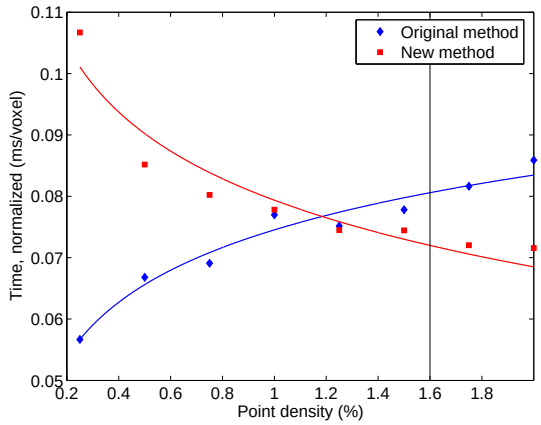


Fig. 6. Validation of theoretical results. The vertical line indicates the lower bound predicted by Equation 4.

These results emphasize the fact that if Equation 4 is satisfied, the new method is guaranteed to be faster than the first. Under this limit, there is no guarantee because the analysis is only considering the average over the whole volume and not taking into account the local clustering of the data.

C. Creation and insertion

The creation of the data structure is done only once at the initialization of the process. For a 200x200x30 grid, it takes 720 ms to create the 1.2 million voxels. Moreover, with 10 cm grid voxels, we can insert nearly three million points per second. In our application, each voxel stores nine 64-bit values for saliency computation. Moreover, each element in the additional arrays introduced in Section IV-E are 32-bit pointers, and results are contained within three 64-bit values. The total memory usage, for the flat ground example shown in Table I is:

$$\begin{aligned} memory &= 200 \times 200 \times 30 \times 9 \times 64 \\ &\quad + 200 \times 200 \times 32 + 2 \times 200 \times 30 \times 32 \\ &\quad + 59,275 \times 3 \times 64 \\ &\quad \approx 84MB \end{aligned}$$

Similarly we obtain 112 MB for the forest and 126 MB for the tall grass environment. Those memory requirements are well within current computers memory capabilities.

TABLE I

STATISTICS FOR THE DIFFERENT TERRAINS WITH 10 CM VOXELS.

Terrain	Size n (in cells)	Raw data	Occupied voxels v	$\frac{v}{n}$	$\bar{d}$	$P[X < \frac{k}{2}]$
Flat ground	200x200x30	2.0 million	59,275	0.049	1.0263	0.9917
Forest	160x250x40	1.7 million	112,001	0.070	1.0519	0.9923
Tall grass	200x300x30	1.2 million	117,756	0.065	1.0678	0.9856

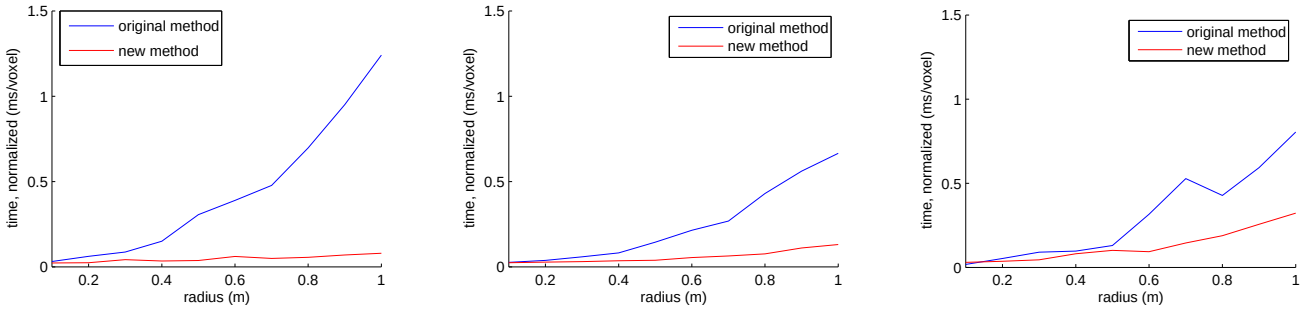


Fig. 7. Point density influence for the tall grass data set. (a) With raw data (117,000 occupied voxels). (b) With data sub-sampled 10 times (55,000 occupied voxels). (c) With data sub-sampled 100 times (9500 valid voxels).

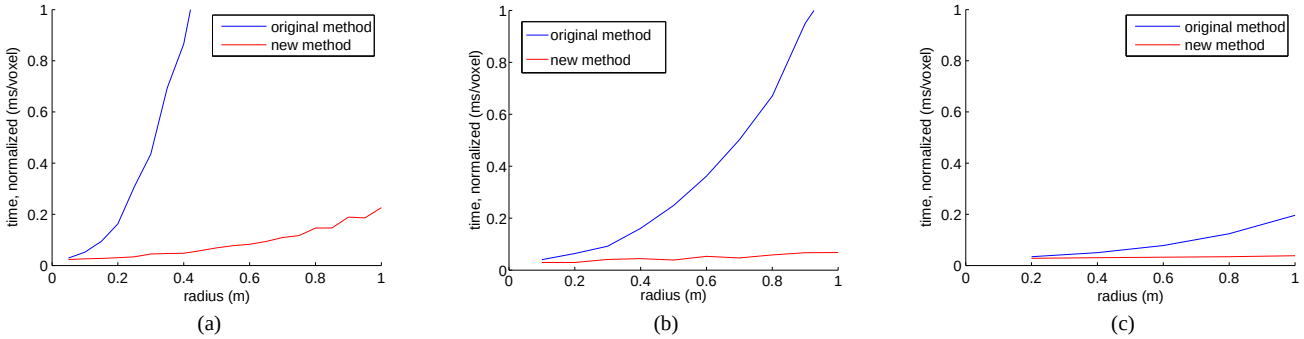


Fig. 8. Voxel size influence for the flat terrain data set at full density. (a) 5 cm voxel size. (b) 10 cm voxel size. (c) 20 cm voxel size.

D. Comparison for different terrain types

In this section, we compare and analyze the performance of our approach for different types of terrain: bare ground (Figure 9-(a-c)), highly cluttered forest (Figure 9-(d-f)) and an open space with vegetation cover (Figure 9-(g-i)). The bare ground scene includes a gravel trail bordered by a jersey barrier. A concertina wire is laid across the trail. The forest scene is made of large tree trunks scattered over a rough terrain covered with short grass and debris. The last terrain is a side slope covered by dense, dry, waist-high grass, with some large poles present. The statistics relative to each data set are presented in Table I. The initial results seem to show that the type of terrain does not influence $\bar{d}$ and $P[X < \frac{k}{2}]$. We think that the very high point density of each data set may have leveled the results.

The center column of Figure 9 illustrates the histograms of the distribution of X for different strategies. The strategies illustrated represent the direction in which the previous occupied voxel is searched (top-right: *x*, bottom-left: *y*, bottom-right: *z*, top-left: best of the three). As expected, the optimal strategy always shows the highest peak at 1, whereas the *z* strategy always gives the lowest. This is because the ground plane is roughly aligned with the *xy*-plane, so adjacent occupied voxel are more likely to be on that plane than along the *z* direction.

The speed improvement over the previous method is illustrated by the third column of Figure 9. The blue curve indicates the performance (in ms per occupied voxel) of the previous method and the red curve shows the performance of the new method. The speedup is substantial, and increases with the radius r used for range search. For example, at the current voxel size used on-board of the robot of r=0.4 m, the new method is 4.6 times faster on the flat ground example, 5.5 times on the forest example, and 3.6 times on the tall grass, which results in an average speedup of approximately 4.5 over the three examples.

E. Parameters influence

In this section, we analyze the influence of two important parameters: the point density, which depends on the sensor used, and the voxel size that is passed as input to the system.

1) Point density: Intuitively, denser data means a larger number of occupied voxels, which in turn implies a higher probability of overlapping neighborhoods. This is confirmed

by experimental results obtained by artificially varying point density by sub-sampling the original data set 10 and 100 times. Timing results for the tall grass example are shown in Figure 7. We observe that the new method performs faster with denser data. In addition, Table II shows relevant statistics for those three examples. We can see that $P[X < \frac{k}{2}]$ increase and $\bar{d}$ decrease with higher point density, which confirms our intuition.

On the other hand, it is interesting to note that the previous method runs *slower* with denser data. This is explained by the fact that, for each voxel, the neighborhood is likely to contain more points than with sparser data. Therefore, the number of visited voxel per occupied voxel is higher, hence the increase in computation time.

TABLE II

RESULTS STATISTICS FOR THE POINT DENSITY INFLUENCE.

Sub-sampling	Raw points	Occupied voxels v	$\bar{d}$	$P[X < \frac{k}{2}]$
0 (raw data)	1,251,402	117,756	1.06	0.9856
10	114,161	54,808	1.2	0.9617
100	10,390	9,469	1.97	0.6926

2) Voxel size: We observe that, for a smaller voxel size, the number of voxels must be greater to keep the same range search radius r. For example, if $r = 4$ with voxel size of 10 cm, then $r = 8$ with voxel size of 5 cm, so 8 times more voxels must be visited than before. More generally, if v_{size} is the voxel size and n_{neigh} the number of neighbors of a voxel, then with v_{size}/k we get $k^3 n_{neigh}$ neighbors. Moreover, smaller voxel size increases the number of holes in the data, which in turn increases $\bar{d}$ and decreases $P[X < \frac{k}{2}]$, as shown in Table III. Figure 8 shows timing results obtained by running the previous and new method on the same full resolution data set and varying only the voxel size. The running time is indeed much slower with a voxel size of 5 cm versus 10 cm. Interestingly, the difference is not as obvious when comparing 10 and 20 cm.

These results show the important compromise relative to this parameter. An increasingly high voxel size will result in faster performance, but also in a loss of precision in scene details. Indeed, much of the high frequency content of the scene will be lost. On the other hand, if the voxel size is too low, the details will be preserved, but the running time will be much slower. The best parameter (10 cm in our case) is found by running benchmark tests on typical examples.

VI. CONCLUSION

In this paper we present a method inspired by dense stereo-correlation in order to improve the computation speed of 3-D ladar data analysis for terrain classification. It should be noted that any computation based on a support volume and divisible into elementary sums can take advantage of this work. The approach relies on the reuse of computed data. The approach is validated on ladar data obtained in various environments using the Demo III XUV.

For the three typical data set analyzed, we observed significant improvement in execution speed without noticeable differences between the various terrain types studied. We achieve on average a 4.5 fold speedup with a voxel size of 0.1 m and a range search radius of 0.4 m. Those parameters are the ones used in the field tests reported in [14].

The current implementation is based on a static data structure and a uniform weighting scheme of the contribution of the different volume elements. We are working on the implementation of a scrolling version that will allow us to test it live on-board the ground vehicle. Also, we are investigating a different weighting scheme to extend this work to generic kernel-like computation.

ACKNOWLEDGMENTS

Prepared through collaborative participation in the Robotics consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-209912. The authors would like to thank General Dynamics Robotic Systems for its support.

REFERENCES

[1] H. Chen and P. Meer. Robust computer vision through kernel density estimation. In *European Conference on Computer Vision*, 2002.

[2] A. Kelly et al. Toward reliable off-road autonomous vehicle operating in challenging environments. In *International Symposium on Experimental Robotics*, 2004.

[3] O. Faugeras et al. Real-time correlation-based stereo : algorithm, implementations and applications. Technical Report RR-2013, INRIA, 1993.

[4] J. Gao and R. Gupta. Efficient proximity search for 3-d cuboids. In *Computational Science and Its Applications*, volume 2669 of *Lecture Notes in Computer Science*, 2003.

[5] A. Gray and A. Moore. Data structures for fast statistics. Tutorial presented at the International Conference on Machine Learning, 2004.

[6] A. Lacaze, K. Murphy, and M. DelGiorno. Autonomous mobility for the demo iii experimental unmanned veh icles. In *Proceedings of the AUVSI Conference*, 2002.

[7] J. Lersch, B. Webb, and K. West. Structural-surface extraction from 3-d laser-radar point clouds. In *Laser Radar Technology and Applications IX*, volume 5412. SPIE, 2004.

[8] T. Liu, A. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Neural Information Processing Systems*, 2004.

[9] H. Moravec. Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical Report CMU-RI-TR-96-34, Carnegie Mellon Univeristy, 1996.

[10] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[11] A. Talukder, R. manduchi, A. Rankin, and L. Matthies. Fast and reliable obstacle detection and segmentation for cross-country navigation. In *IEEE Intelligent Vehicle Symposium*, 2002.

[12] Ranjith Unnikrishnan and Martial Hebert. Robust extraction of multiple structures from non-uniformly sampled data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

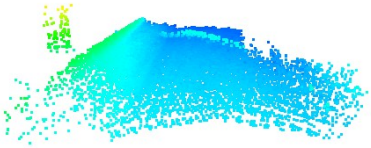
[13] N. Vandapel, R. Donamukkala, and M. Hebert. Unmanned ground vehicle navigation using aerial ladar data. *To appear in the International Journal of Robotics Research*, 2005.

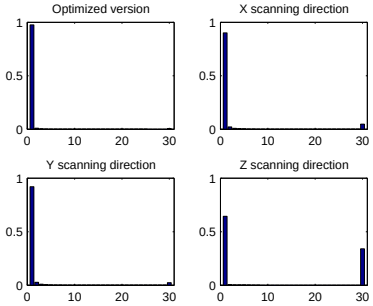
[14] N. Vandapel, D. Huber, A. Kapuria, and M. Hebert. Natural terrain classification using 3-d ladar data. In *IEEE International Conference on Robotics and Automation*, April 2004.

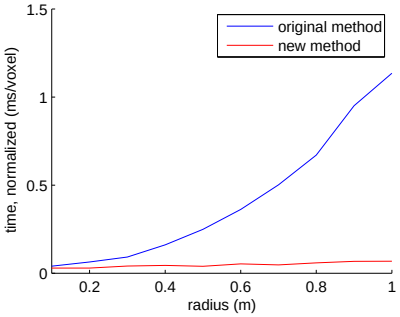
TABLE III

STATISTICS FOR THE VOXEL SIZE INFLUENCE. 2,046,123 RAW DATA POINTS

Voxel size	Size n (in cells)	Occupied voxels v	$\bar{d}$	$P[X < \frac{k}{2}]$
5 cm	400x400x60	359,327	1.1063	0.993
10 cm	200x200x30	59,275	1.0263	0.991
20 cm	100x100x15	14,485	1.00	0.986

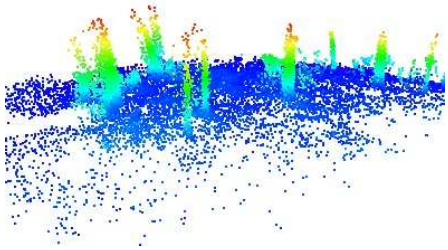


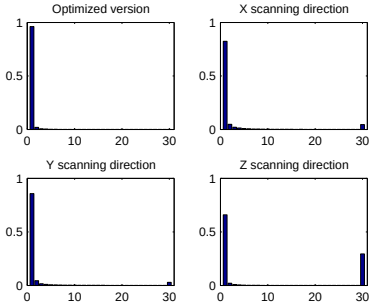


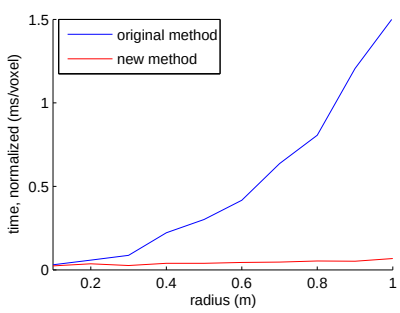


Flat ground statistics: $\frac{v}{n}$ = 0.049 ; $\bar{d}$ = 1.0263 ; $P[X < \frac{k}{2}]$ = 0.9917

(a) (b) (c)

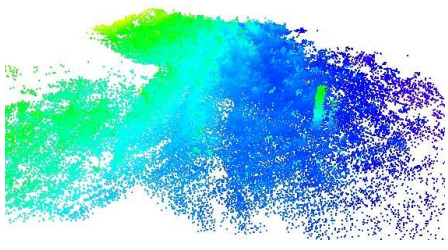


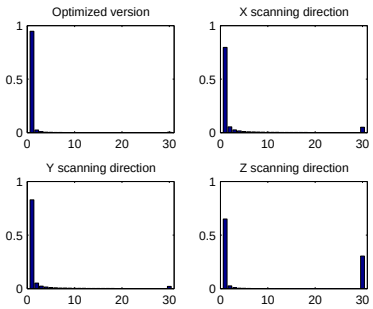


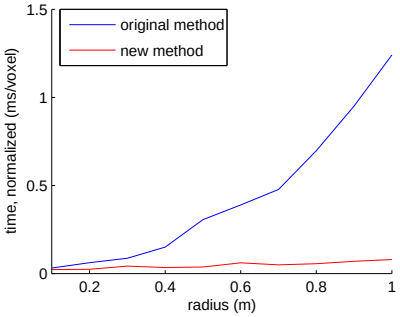


Forest statistics: $\frac{v}{n}$ = 0.070 ; $\bar{d}$ = 1.0519 ; $P[X < \frac{k}{2}]$=0.9923

(d) (e) (f)







Tall grass statistics: $\frac{v}{n}$ = 0.065 ; $\bar{d}$ = 1.0678 ; $P[X < \frac{k}{2}]$ = 0.9856

(g) (h) (i)

Fig. 9. Terrain influence. Each line corresponds to a terrain or environment type: flat, forest and tall grass. The first column contains a snapshot of a 3-D model of the scene, the elevation is color coded. The second column contains histograms of distribution of distances between current voxel and previous occupied voxel, for different scanning directions. The *x* (*y*) axis is the distance in number of voxel (the number of voxels). The rightmost peak represents infinite distance, that is, there is no previous occupied voxel in that direction. It is positioned at an arbitrary distance in the graph. The last column shows a comparison of speed of execution of the original versus the new method, with voxel size of 0.1 m.