VECIMS 2004 - IEEE International Conference on
Virtual Environments, Human-Computer Interfaces, and Measurement Systems
Boston, MA, USA, 12-14 July 2004

An Optimized Computational Technique for Free Space Localization in 3-D Virtual Representations of Complex Environments

P. Payeur

Vision, Imaging, Video and Autonomous Systems Research Laboratory
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada, K1N 6N5
ppayeur@site.uottawa.ca

Abstract - *Modeling 3-D objects as octrees has demonstrated considerable advantages which led to numerous applications in robotics where free space localization is critical. Efficient neighbor finding techniques in tree structures are required for such models to be used properly, especially for path planning and collision avoidance. In this paper, an optimized neighbor finding approach is presented that is based on a recursive addressing scheme which precludes any backtracking into the tree structure while preserving model compactness. Neighboring cell addresses are computed directly given a displacement direction in 3-D space and the address of the starting cell. Neighboring rule sets that have been previously derived for a quadtree representation are now extended to octrees. Given the algebraic rules that are defined, computation of a neighboring cell address in an octree comes down to basic arithmetic operations with carry. The algorithm complexity is kept low in order to provide good performances.*

Keywords - *3-D modeling, neighbor finding, occupancy maps, virtual navigation.*

I. INTRODUCTION

Modeling the environment in which an autonomous system must operate is a critical issue in robotic applications. The availability and fast access to information about the cluttering of space is primordial for solving many problems in this field. Octrees demonstrate important advantages to satisfy these requirements in robotic telemanipulation because of their compactness and their capability to represent multiresolution models which is critical to ensure performances in 3-D space representations.

Octrees offer a compact way to encode the same information as the classical 3-D occupancy grids that result from a recursive subdivision of space along each of their axes. When each of the resulting space cell is tagged with the occupancy state of space, this provides a detailed representation of space cluttering. Such models can contain a combination of different levels of resolution depending on the uniformity of space state.

An important aspect is that an octree structure does not contain any direct information about the Cartesian position of a given cell. All cells are defined in relationship with the origin and the size of the mother cell that corresponds to the entire volume that is modeled. The traversal path from the mother cell to a cell of interest in the tree structure is then the only way to locate a given volume.

Several applications of octrees require a means of computing 3-D spatial displacements based on the connectivity between cells contained in the model as if it were encoded in a Cartesian grid. Split-and-merge segmentation in 3-D computer vision, properties estimation in biomedical imaging [1] and collision-free path planning for robotic manipulators [2] are all examples of such applications. In the later case, a neighbor finding technique is needed to identify a free path among a set of objects. As the entire structure of a robot needs to be moved without colliding with the environment, the number of cells to validate is significantly increased in comparison with the problem of mobile robot path planning that often summarizes to the displacement of a point. An efficient neighbor finding technique is therefore essential to quickly locate free space in 3-D volumes. Unfortunately, in a tree structure, neighboring leaves no longer coincide with geometrically adjacent cells in 3-D space, and steps from a leaf to its neighbors must then be defined without referring to Cartesian coordinates.

Samet brought a significant contribution on how to build and use tree-based structures [3,4,5]. But the majority of this work is concerned with 2-D space. The classical neighbor finding approach proposed by Samet is based on the search of a common ancestor cell [6,7]. This technique has been revisited by Besançon [8] who designed a neighbor finding algorithm relying on the encoding structure introduced by Ballard and Brown [9]. The idea consists in backtracking in the tree starting from the initial cell until a common ancestor of this cell and its neighbor of interest is reached. From this point, the algorithm goes down the tree and reaches the desired neighbor cell. Logical functions are introduced to verify the neighboring status between two given cells and the algorithm is designed to minimize backtracking. Nevertheless, it tends to create long traversals of the tree before a valid ancestor can be found, especially in three-dimensional models. In addition, neighbor finding between corner neighbors appears to be more difficult to process than between face neighbors using this approach.

Other strategies that exploit the model size rather than the tree structure can also be found. Klinger and Rhodes [10] propose such an algorithm to identify a sister cell in a given direction in a quadtree by means of a sequence of primitive displacements. This approach is based on some complex primitives that lead to an important computational load. Other researchers propose to add some data inside of each cell of the tree in order to keep the identity of every neighbor cells. Hunter and Steiglitz [11] call these supplementary informations *ropes* which are inserted in the model during its building. These ropes allow to directly access the neighbor cell in a given direction without the need for a neighbor search while the model is in use. On the other hand, extra memory space is required to store the ropes, and the neighbor finding process complexity is not reduced since all neighbors must be computed for each cell while the model is built. As only a small percentage of the ropes are actually used in a given application, computing time is wasted in preprocessing many ropes that will never be used.

Schrack [12] proposes a sophisticated approach to perform elementary operations such as addition and subtraction on data encoded in the linear quadtree defined by Gargantini [13]. These operations are used to identify neighbors of the same size in a given direction by encoding geometrical data in the cells. With this information available, the identity of a neighbor cell in a given direction is easily computed. In retrospect, such an approach is equivalent to encoding the model as a Cartesian grid rather than a tree structure, compromising their inherent advantages. Binary trees (bintrees) have also been proposed to provide models which are approximately 25% more compact than classical quadtrees and octrees [14,15]. Unfortunately, bintrees use a simplified structure that limits their application range.

In this paper, a neighbor finding approach that does not imply any backtracking in the octree structure nor the encoding of any supplementary data in the model is presented. This method has been initially developed and validated for 2-D space [16] and is now extended to three-dimensional models. The algorithm uses an addressing scheme which identifies each cell. Starting from a given address, the addresses of neighbor cells are computed using simple algebra and taking into account the displacement direction and the address of the original cell.

The following sections define the selected addressing scheme and detail the proposed address-based neighbor identifier before applying it to free space identification in a generic 3-D space representation. Finally the complexity and performance of the approach are analyzed.

II. CELLS ADDRESSING

The basis of the proposed navigation algorithm in octree models is the addressing scheme that associates a numerical value with each branch and each leaf of the tree structure. Various classical types of addressing have been considered during the development. For instance, Shu and Kankanhalli's addressing method [1] relies on the subdivision level, L, and on the relative coordinates *(x, y)* of a given cell with respect to the origin as shown in figure 1a. Here the concept is presented on a 2-D multiresolution grid for clarity. The address associated to each cell is of the form *(L, x, y)*, where the mother cell receives address (0, 0, 0). This type of addressing allows to build a tree for which it is not required to use pointers that connect leaves and branches to each other. It results in a simple structure but requires a more complex set of logical rules to process the addresses. Especially, a complete update of all addresses is required if the subdivision level of a part of the model is changed, making this encoding inadequate for dynamic modeling.

Samet [4] proposes a continuous addressing scheme as shown in figure 1b. The address grows with the number of

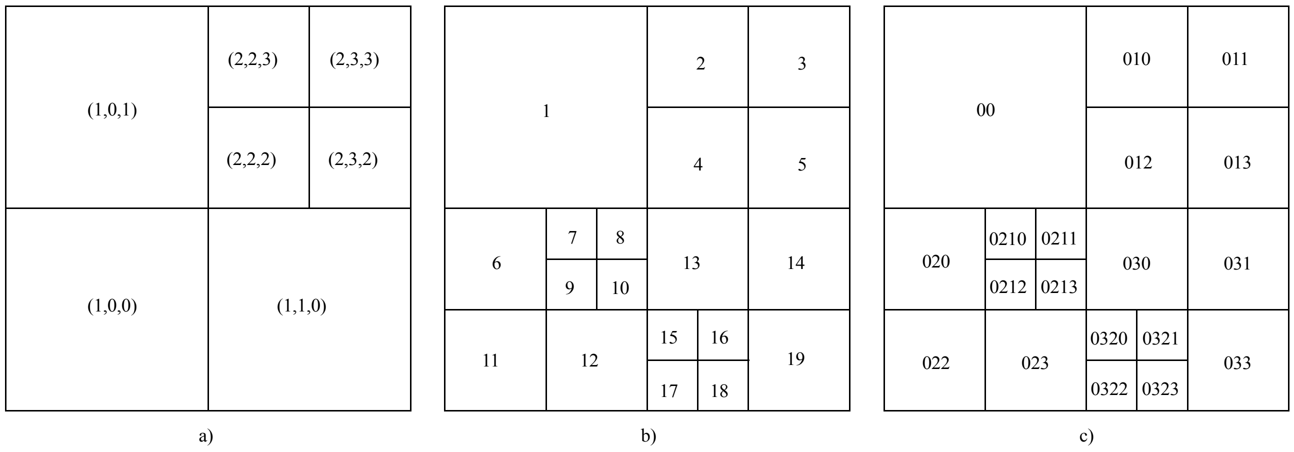

Fig. 1. Various addressing schemes: a) Shu and Kankanhalli, b) Samet, c) Major *et al.*

independent cells in the grid. In spite of the relative simplicity of this addressing approach, it appears to be difficult to define a standard numbering strategy since the subdivision level of each region of space is not known *a priori* in a model in evolution like those found in telerobotics. Major *et al*. [17] proposes a basic addressing method as shown in figure 1c. Cells are incrementally numbered from 0 to 3 following a scanning pattern of the four children (for a quadtree) resulting from the subdivision of a mother cell. The number of digits varies with the resolution level as one digit is associated with each level starting from the lowest one (the entire model). As in the example of figure 1c for a 2-D map, the first mother cell is tagged 0 while its four children are respectively tagged 00, 01, 02 and 03 in a predefined order. In accordance with this pattern, the four children of the second child (01) are respectively tagged 010, 011, 012 and 013.

This addressing follows a logical numbering scheme which depends on the resolution level of each cell. For this reason, it remains perfectly independent from the subdivision state of a given cell and allows easy addressing of multiresolution grids. Local addresses can also be easily updated following the need to increase the resolution of some areas of the model or to merge groups of cells exhibiting similar characteristics. This scheme then provides the required flexibility and adaptability for complex environments representation.

Moreover, it is easily extended to 3-D space where subdividing a cell results in eight children instead of four. For instance, the eight children of the main 3-D mother cell would receive respectively the addresses: 00, 01, 02, 03, 04, 05, 06 and 07.

The definition of the scanning pattern is not a critical issue. The ordering of addresses can vary provided that the same scanning pattern is respected at each resolution level and that the same numbering structure is preserved. The repetitiveness characteristic is essential in order to take advantage of the addressing scheme to identify neighboring cells in space. The addressing scheme that has been adopted in the present work is a 3-D extension of Major *et al*. with the first cell located at the bottom left corner on the back side of the main mother cell as shown in figure 2a.

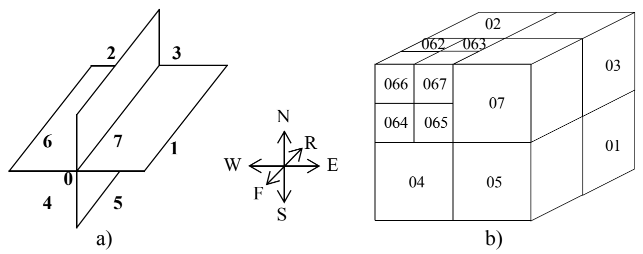

Fig. 2. a) Incremental addressing scheme assigned to b) a 3-D space occupancy representation.

III. ARITHMETIC FOR NEIGHBORS IDENTIFICATION

The proposed neighbor finding technique results from the logical behavior of the selected addressing scheme. The addressing structure leads to a set of rules which determine the address change when one moves from a given cell to its neighbor for each possible direction. These rules are similar to elementary arithmetic operations with carry on the next left digit which composes the address.

Considering only the back layer of the tridimensional Cartesian grid shown in figure 2b to illustrate the rule definition process in a simple case, their are eight possible directions of displacement: North (N), South (S), East (E), West (W), North-East (NE), North-West (NW), South-East (SE) and South-West (SW), as the third dimension is neglected [16]. Therefore, neighbors can be grouped into two categories: *edge neighbors* respectively located in the North, South, East and West directions; and *vertex neighbors* located respectively in the North-East, North-West, South-East and South-West directions.

According to their relative position in the tree structure, s*ister cells* are defined as cells that belong to the same mother cell. *Cousin cells* are defined as cells whose mother cells are sister cells. From there, neighboring rules are built by comparing successively only the digits of the cells addresses that correspond to a same level of resolution, e.g. the digits which occupy the same position in the address expression. The computation is thus performed one level at a time starting at the highest resolution level (the rightmost digit) until the lowest resolution level is reached (the leftmost digit).

A. Neighboring Between Sister Cells

Based on the addressing scheme of figure 2, for a given level of resolution (sister cells), address computation rules for edge neighbors can be established as follows through a close examination of address variations for each direction: the digit changes from 0 to 1 or from 2 to 3 in the East direction, from 0 to 2 or from 1 to 3 in the North direction, from 1 to 0 or from 3 to 2 in the West direction, and finally from 3 to 1 or from 2 to 0 in the South direction. Similarly, for vertex neighboring between sister cells, the rules can be defined as follows: a digit is changed from 2 to 1 in the South-East direction, from 1 to 2 in the North-West direction, from 0 to 3 in the North-East direction and from 3 to 0 in the South-West direction.

B. Neighboring Between Cousin and Distant Cells

An important phenomenon that occurs in the addresses when a displacement from a given cell to its neighbor implies a change in their parent cells is that the digit associated to the parent's resolution level is also affected. Neighboring rules must then be applied successively to each digit starting from the rightmost one until the affected parent's resolution level is reached. Moving from a parent to another is similar to the application of a carry on the immediate left digit in elementary arithmetic.

Processing a carry, to which a direction is associated here, follows the same neighboring rules as those applied to the previous digit. That is, the set of rules between sister cells is applied with eventually another carry being applied on the next left digit until common parents are reached. Because of the recursive structure of the tree, this situation must occur. In the worst case, neighboring rule set processing continues until the leftmost digit is reached. But this occurs only when the delimitations between each of the four children of the global mother cell are traversed. In the large majority of situations, carries on left digits are limited to higher levels of resolution (few rightmost digits).

Even though a given number of carries might be required to find the address of a neighbor cell, it is important to note that no backtracking in the tree structure is required. Only arithmetic manipulations on the address digits are necessary. This significantly speeds up processing as the model stored in memory doesn't need to be accessed and no validation of the actual neighboring between cells needs to be performed. Computation of the other cell's address with the proposed computational technique guaranties that a valid neighbor is reached in the desired direction.

The application of the neighboring rule set is driven by two simple principles. First, processing an address always begins with the rightmost digit (the highest resolution level) and successively proceeds to the left as carries are generated. Second, processing is based on the initial value of the digit in the starting cell's address and on the moving direction. These two informations alone determine the rule to apply in order to identify the appropriate new value for the considered digit. Furthermore, these rules are easily encoded in a lockup table to speed up processing.

IV. FREE SPACE LOCALIZATION IN 3-D STRUCTURES

The definition and the application of a neighboring rule set in 3-D structures (octrees) follows the same principle as for the 2-D case. The main difference is that the number of rules increases as a consequence of the additional number of neighbors that goes from 8 to 26 possible directions as shown in figure 3.

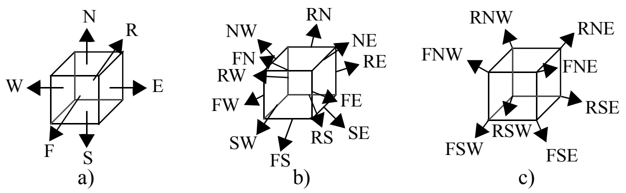

Fig. 3. Possible directions of displacement in 3-D space: a) face, b) edge, and c) vertex neighbors.

Neighboring relationships can be divided into three categories depending of the type of intersection occurring between cells in a given direction. *Face neighbors* occur in the North (N), South (S), East (E), West (W), Front (F) and Rear (R) directions. *Edge neighbors* occur in the North-East (NE), North-West (NW), South-East (SE), South-West (SW), Front-East (FE), Front-West (FW), Read-East (RE), Rear-West (RW), Front-North (FN), Front-South (FS), Rear-North (RN) and Rear-South (RS) directions. Finally, *vertex neighbors* occur in the Front-North-East (FNE), Front-North-West (FNW), Front-South-East (FSE), Front-South-West (FSW), Rear-North-East (RNE), Rear-North-West (RNW), Rear-South-East (RSE) and Rear-South-West (RSW) directions.

Neighboring rules are defined based on a close inspection of addresses behavior when a displacement in each possible direction occurs and for each of the eight possible children that result from the subdivision of a mother cell in an octree. The principle of a carry applied to the immediate left digit is also preserved. For compactness, the resulting set of rules is presented as lookup tables 1, 2 and 3 for each of the three types of neighbors.

The numbers in the left columns correspond to the initial value of a given address digit while the header rows represent moving directions in accordance with the 3-D compass-card depicted in figure 2. Alphanumeric codes in the lockup tables define the required operation to compute the new address for each digit value and displacement direction. When only a number (0 to 7) appears, this means that the initial value of the digit must be replaced by this number and no carry is generated (sister cell neighboring). The addition of a carry mark (+ X) indicates that a carry in the X direction must also be applied on the digit located immediately to the left of the digit presently considered in the cell address.

For example, if one moves in the Front-North-West (FNW) direction starting from the octree cell tagged 0742, this consists in a vertex-type neighboring relationship. When table 3 is consulted, it reveals that for a digit value equal to 2 and the FNW direction, the initial digit value must be replaced by 5 and that a carry in the North-West direction is applied to the following left digit, e.g. 4. This carry implies an edge-type neighboring relationship. Therefore, line labelled 4 in table 2 is consulted for the NW direction and indicates that the digit value of 4 must be replaced by a value of 7 and that a new carry is generated in the West direction and applied to the following left digit, e.g. 7 in the original address. As a final step, a face neighboring is requested by this last carry. Table 1 is then used to determine that, for an original digit value of 7 with a displacement in the West direction, the value is replaced by a 6 and no more carry is generated thus completing the processing of the address. The neighbor cell address from 0742 in the Front-North-West direction is finally 0675. This can be verified geometrically by a close inspection of figure 2.

Table 1. Face neighboring rules in an octree structure.

	N	S	E	W	F	R
0	2	2 + S	1	1 + W	4	4 + R
1	3	3 + S	0 + E	0	5	5 + R
2	0 + N	0	3	3 + W	6	6 + R
3	1 + N	1	2 + E	2	7	7 + R
4	6	6 + S	5	5 + W	0 + F	0
5	7	7 + S	4 + E	4	1 + F	1
6	4 + N	4	7	7 + W	2 + F	2
7	5 + N	5	6 + E	6	3 + F	3

Table 2. Edge neighboring rules in an octree structure.

	NW	NE	SW	SE	FN	RN	FS	RS	FE	FW	RE	RW
0	3+W	3	3+SW	3+S	6	6+F	6+S	6+RS	5	5+W	5+R	5+RW
1	2	2+E	2+S	2+SE	7	7+R	7+S	7+RS	4+E	4	4+RE	4+R
2	1+NW	1+N	1+W	1	4+N	4+RN	4	4+R	7	7+W	7+R	7+RW
3	0+N	0+NE	0	0+E	5+N	5+RN	5	5+R	6+E	6	6+RE	6+R
4	7+W	7	7+SW	7+S	2+F	2	2+FS	2+S	1+F	1+FW	1	1+W
5	6	6+E	6+S	6+SE	3+F	3	3+FS	3+S	0+FE	0+F	0+E	0
6	5+NW	5+N	5+W	5	0+FN	0+N	0+F	0	3+F	3+FW	3	3+W
7	4+N	4+NE	4	4+E	1+FN	1+N	1+F	1	2+FE	2+F	2+E	2

Table 3. Vertex neighboring rules in an octree structure.

	FNE	FNW	FSE	FSW	RNE	RNW	RSE	RSW
0	7	7+W	7+S	7+SW	7+R	7+RW	7+RS	7+RSW
1	6+E	6	6+SE	6+S	6+RE	6+R	6+RSE	6+RS
2	5+N	5+NW	5	5+W	5+RN	5+RNW	5+R	5+RW
3	4+NE	4+N	4+E	4	4+RNE	4+RN	4+RE	4+R
4	3+F	3+FW	3+FS	3+FSW	3	3+W	3+S	3+SW
5	2+FE	2+F	2+FSE	2+FS	2+E	2	2+SE	2+S
6	1+FN	1+FNW	1+F	1+FW	1+N	1+NW	1	1+W
7	0+FNE	0+FN	0+FE	0+F	0+NE	0+N	0+E	0

Using this augmented set of rules for computing the addresses of neighboring cells in an octree offers an efficient way to directly identify paths of empty space connected with a given starting point (a given cell) no matter the complexity of the scene encoded in the 3-D virtual representation. Since such a process must be repeated a very large number of times in a path planning operation with collision avoidance in order to successfully connect volumes of space through which all members of a robot manipulator can safely circulate, the proposed technique significantly contributes to reduce the computational workload introduced by the determination of a proper sequence of movements in virtual representations of complex environments. Some generalization has also been made to efficiently handle multiresolution models and to deal with border effects in the model using the proposed approach. Even though they are not detailed here due to space limitations, experimentation demonstrated their efficiency and generality.

V. ALGORITHM COMPLEXITY

The complexity of the proposed approach simply depends on the number of digits to be processed for finding the neighbor cell address in a given direction. In other words, it depends on the number of accesses that are made to the lookup tables containing the rule sets. The geometric location of neighboring cells in the grid thus slightly influences the complexity. As shown above, sister neighbors are rapidly identified since no carry is generated. Only one access to a lookup table is sufficient to compute the neighbor cell address completely. In the case of cousin cells, two digits must be changed and, for more distant neighboring relationships, the number of accesses grows proportionally to the gap between terminal leaves of common parent branches in the tree structure.

Therefore, the maximum complexity of the proposed neighbor finding algorithm is $O(p)$ where p represents the number of resolution levels in the octree that is equal to the number of digits in the addresses for the highest resolution. This algorithm is much simpler in comparison with a backtracking algorithm that searches for a common parent. As the later needs to scan all potential children and check for a neighborhood relationship in the proper direction by accessing the model, such an approach has a maximum complexity of $O((2^3)^p)$ for a 3-D model.

Moreover, manipulating the addresses by only accessing lookup tables is faster than accessing the octree virtual representation each time neighboring needs to be verified. Implementation of the proposed arithmetic is also limited to the encoding of 3 lookup tables. Experimentation with the proposed approach for empty space localization in robot path planning demonstrated a drastic improvement in computation time in comparison with a classical backtracking approach.

VI. CONCLUSION

In this paper, an algorithm allowing a fast neighbor cell search in 3-D virtual representations of complex scenes encoded as octree models has been presented. The approach based on a hierarchical addressing scheme of cells, provides a reliable and efficient strategy for identification of free space in robotic guidance tasks without the lost of compactness characterizing multiresolution octree models.

Moreover, the performance of the algorithm is totally independent of the complexity of the model, it is able to take full advantage of multiresolution encoding, and no supplementary information needs to be added in the model as widely suggested in the literature.

These characteristics allow to keep a low complexity in comparison with backtracking neighbor finding algorithms which search the tree until a common parent is found. Even though the algorithm has been developed in the context of collision-free path planning for telemanipulators, numerous applications can be envisioned especially in the fields of computer vision and robotics.

ACKNOWLEDGMENT

The author wishes to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) that made this research work possible.

REFERENCES

[1] R. Shu, M. S. Kankanhalli, "Efficient Linear Octree Generation from Voxels", *Image and Vision Computing*, vol. 12, no. 5, pp. 297-303, June 1994.

[2] M.C. Martin, H.P. Moravec, *Robot Evidence Grids*, Technical Report CMU-RI-TR-96-06, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PN, 1996.

[3] H. Samet, "Region Representation: Quadtrees from Boundary Codes", in *Communications of the ACM*, vol. 23, no. 3, pp. 163-170, March 1980.

[4] H. Samet, *The Design and Analysis of Spatial Data Structures*, Reading, MA, Addison-Wesley, 1990.

[5] H. Samet, *Applications of Spatial Data Structures*, Reading, MA, Addison-Wesley, 1990.

[6] H. Samet, "Neighbor Finding Techniques for Images Represented by Quadtrees", in *Computer Vision Graphics and Image Processing*, vol. 18, pp. 37-57, 1982.

[7] H. Samet, "Neighbor Finding in Images Represented by Octrees", in *Computer Vision Graphics and Image Processing*, vol. 46, pp. 367-386, 1989.

[8] J. E. Besançon, *Vision par Ordinateur en Deux et Trois Dimensions*, Paris, Eyrolles, pp. 341-353, 1988.

[9] D. H. Ballard, C. M. Brown, *Computer Vision*, Englewood Cliffs, NJ, Prentice-Hall Inc., 1982.

[10] A. Klinger, M. L. Rhodes, "Organization and Access of Image Data by Areas", in *IEEE Transactions on Pattern Analysis and Machine Intelligence I*, pp. 50-60, January 1979.

[11] G. M. Hunter, K. Steiglitz, "Operations on Images Using Quad Trees", in *IEEE Transactions on Pattern Analysis and Machine Intelligence I*, pp. 145-163, April 1979.

[12] G. Schrack, "Finding Neighbors of Equal Size in Linear Quadtrees and Octrees in Constant Time", in *Computer Vision Graphics and Image Processing*, vol. 55, no. 3, pp. 231-239, May 1992.

[13] I. Gargantini, "Linear Octtrees for Fast Processing of Three-Dimensional Objects", in *Computer Graphics and Image Processing*, vol. 20, pp. 365-374, 1982.

[14] C. Y. Huang, K. L. Chung, "Fast Operations on Binary Images Using Interpolation-Based Bintrees", in *Pattern Recognition,* vol. 28, no. 3, pp. 409-420, March 1995.

[15] C. Y. Huang, K. L. Chung, "Faster Neighbor Finding on Images Represented by Bincodes", in *Pattern Recognition*, vol. 29, no. 9, pp. 1507-1518, September 1996.

[16] P. Payeur, "Efficient Neighbor Identification in Multiresolution kD-Trees for Mobile Robot Navigation", in *Proceedings of the IASTED International Conference on Robotics and Automation*, Clearwater, FL, Nov. 2001.

[17] F. Major, J. Malenfant, N. F. Stewart, "Distance Between Objects Represented by Octtrees Defined in Different Coordinate Systems", *Computer & Graphics*, vol. 13, no. 4, pp. 497-503, 1989.