PERGAMON

Robotics and Computer Integrated Manufacturing 17 (2001) 447–459

Robotics and Computer Integrated Manufacturing

www.elsevier.com/locate/rcim

Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees

Jozef Vörös*

Faculty of Electrical Engineering and Information Technology, Slovak Technical University, Ilkovicova 3, 812 19 Bratislava, Slovak Republic

Accepted 29 June 2001

Abstract

Hierarchical tree representations are used for computing and representing distance maps in homogeneous or weighted 2D and 3D environments. It is shown that the matrix forms of quadtree and octree descriptions are proper for storing not only image or spatial data but also distance data. Distance maps are used in collision-free path planning in environments with static obstacles. The resulting sequences of free-region quadrants and octants enable to compute distance-optimized paths to the chosen resolution level. More illustrative examples are included and analyzed for 2D and 3D workspaces. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Robotics; Quadtrees; Octrees; Distance map; Path planning

1. Introduction

From implementation point of view, path-planning algorithms can be broadly divided into two categories [1]. In the first category are the methods, which make trivial (if any) changes to the image representation before planning a path (regular grid search, vertex graph methods). The methods in the second category make elaborate representation changes to convert to a representation, which is easier to analyze before planning collision-free paths (transform methods, potential fields, Voronoi method). Though the two categories by no means exhaust the existing methods, they point out that what may be needed is a compromise between these two categories, i.e., appropriate pre-processing of the image representations without significant data extension, while preserving the original geometric descriptions.

Distance maps (also known as distance transforms) belong to the second category and are defined as the minimum cost to reach a particular location from a starting position. They are of considerable interest in connected 2D and 3D domains and have been used in robotics for a variety of path-planning applications, e.g., planning minimum-length paths through mobile robot domains (where walls and fixed furnishings impose constraints on the domain) and controlling robot arm motion by planning in 3D space [2]. Implementations of distance maps are generally performed in grid-based representations and their main disadvantage is the large amount of memory and the costly computation of the distance map through large areas of free space.

Significant memory saving can be achieved when quadtrees or octrees are used for distance maps. In these cases, if there are large areas of free space (or obstacles), then these areas can be represented by a few large blocks in the corresponding quadtree or octree and can be dealt with, as units by the preprocessing and path-planning algorithms. In the previous work, more approaches have been developed for collision detection and path planning, based on the hierarchical tree representations of the robot's environment, e.g., a quadtree-based multiresolution approach using preprocessing [1] or an environment exploration algorithm [3], using path graphs [4], direction-oriented planning [5], octree-based distance maps [6,7], or framed quadtrees and octrees [8,9]. In these applications, the tree representations served only as the input for the algorithms used in the spirit of the first category approaches mentioned above. In all cases, extra data were used for storing the distances.

In this paper, a simple and memory-saving implementation of tree-based distance map and subsequent path planning in 2D and 3D homogeneous or weighted environments is presented. Both tasks are based on

*Tel.: +421-35-6410-572.
E-mail address: jvoros@elf.stuba.sk (J. Vörös).

recently proposed strategies for repetitive neighbor finding in quadtrees [10] and octrees [11] and simplify the distance map approach twofold. First, the implementation is based on the so-called matrix form of quadtree and octree description, which is less memory and execution time extensive than the standard pointer-oriented ones [3,6,12]. Second, the distance values are stored into the matrix entries without changing the nature of original scene description and no extra records are required, compared to the previous approaches [3,6]. Moreover, the neighbor-finding methods enable to find all smaller, larger or equal-sized neighbors for any node. It means that the distance map can be performed for the whole scene (the map concerns only free leaf nodes).

The quadtree/octree distance maps for given goal points are used for finding distance-optimized paths (to the chosen resolution level) from any start point. The resulting path is determined by a sequence of adjacent quadtree or octree nodes (generally of varying sizes) connecting the start and goal points. These nodes enable to compute the coordinates of path segments optimizing the path length. The path planning is fast and complete to the resolution, i.e., it always finds a path if there is one and reports failure otherwise. The changes in quadtrees and octrees due to the distance maps do not influence the corresponding scene representations and do not extend the memory requirements, as they are performed within the given matrix descriptions. Hence, the proposed approach is a good compromise between the above-mentioned two categories.

In Section 2, a brief overview of the matrix representations for quadtrees and octrees is described. In Section 3, a new method of quadtree- and octree-based distance-map computing and storing is presented. This is applied to path planning in 2D and 3D environments in Sections 4 and 5, where more illustrative examples are included.

2. Matrix tree representations

2.1. Matrix quadtree

A quadtree representation of a binary image is a tree whose leaves represent square areas or quadrants of the image. They are labeled with the color of the corresponding area, i.e., BLACK, WHITE, or GRAY (both black and white). Quadtrees can be stored as lists of all nodes' records [13] or locational codes (linear quadtrees) [14].

The matrix quadtree R corresponding to a $(2^N \times 2^N)$-dimensional binary image is given as an $(L \times 4)$-dimensional matrix of all the GRAY (non-terminal) node descriptions [15]

$$R = [P_1, P_2, P_3, \ldots, P_L]^{\mathrm{T}},$$

where L is the number of GRAY nodes. A GRAY node at level k of quadtree representing a $(2^k \times 2^k)$-dimensional region is described by an ordered quadruple

$$P_i = (Q_{i1}, Q_{i2}, Q_{i3}, Q_{i4}), \tag{2}$$

where Q_{im}, $m = 1, \ldots, 4$, denote its four children at level $k-1$. If Q_{im} is a non-terminal node, then

$$Q_{im} = r \quad \text{for a GRAY quadrant}, \tag{3}$$

where $r = 2, 3, \ldots$, refers to the position (row) in vector (1) where the quadruple P_r corresponding to this GRAY quadrant is described. If Q_{im} is a terminal or leaf node, its value can be one of the two distinct non-positive integers chosen for the corresponding color

$$Q_{im} = \begin{cases} -c & \text{for BLACK quadrants,} \\ 0 & \text{for WHITE quadrants.} \end{cases} \tag{4}$$

where $c > 0$ is any integer number. The matrix quadtree given by (1) is a minimum and top–down description of a regular quadtree. Compared to the standard pointer-oriented quadtree description [13], requiring six entries for each node, in this case, only four integers are needed, and only for the non-terminal nodes. Hence, the memory saving is notable (more than 33%).

An important feature of matrix-quadtree description is that the substitutions for terminal nodes given by (4) can significantly extend the capacity of tree representations. Besides their main role, to give information on the shapes and locations of 2D objects, they enable to store further relevant information on areas considered, namely, different negative integers in (4) can be used for distinguishing among more objects in a scene represented by one tree, for assigning colors or labels for connected areas. For path-planning purposes, proper c can represent the weight factor of a given block in heterogeneous environment with weighted regions [9], e.g.,

$$Q_{im} = \begin{cases} -w_j & \text{for } j\text{th weighted regions,} \\ -1 & \text{for free regions,} \\ 0 & \text{for forbidden regions,} \end{cases} \tag{5}$$

where $w_j > 1$ is the corresponding weight factor [16]. A further possibility is to store distance data related to the nodes, i.e., using the following substitutions:

$$Q_{im} = \begin{cases} -d_{im} & \text{for free quadrant,} \\ 0 & \text{for forbidden quadrant,} \end{cases} \tag{6}$$

a distance value d_{im} can be associated with each free quadrant and directly stored to the corresponding entry of the matrix quadtree. The use of substitution by (5) and (6) does not change the structure of given tree.

For example, the simple scene in Fig. 1 is represented by the quadtree in Fig. 2, which is described by the

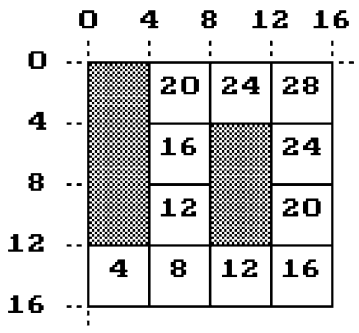

Fig. 1. Scene with distance data.

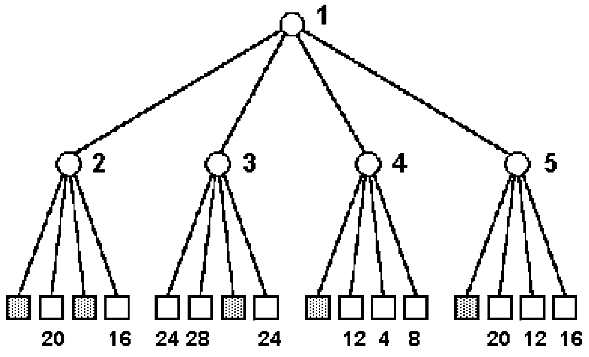

Fig. 2. Quadtree graph.

Table 1
Matrix quadtree

P_i/Q_{ij}	0	1	2	3
1	2	3	4	5
2	0	−1	0	−1
3	−1	−1	0	−1
4	0	−1	−1	−1
5	0	−1	−1	−1

matrix in Table 1. Now the distances of the free quadrants from the bottom-left corner quadrant are stored to the matrix quadtree in Table 2, as negative integers.

Memory saving by this form of distance-map representation is evident. While in the previous approaches [3,12], each node of quadtree (i.e., 21) requires a record containing seven fields (five pointers to the parent and four children nodes, one field for node type and one field for distance value) (i.e., $21 \times 7 = 147$ fields), in the present approach, each non-terminal node (i.e., five) requires only four integers (i.e., $5 \times 4 = 20$ integers). Evidently, the memory saving is enormous in this case.

Table 2
Quadtree distance map

P_i/Q_{ij}	0	1	2	3
1	2	3	4	5
2	0	−20	0	−16
3	−24	−28	0	−24
4	0	−12	−4	−8
5	0	−20	−12	−16

2.2. Matrix octree

The octree representation of 3D object space can be defined as an octree whose eight leaves represent octants of this space and are labeled as FULL, VOID or GRAY (both full and void) in accordance with labeling of corresponding volume [13,17,18]. Each node of octree is associated with a cube of object space resulting from successive subdivisions into eight octants and the root represents the entire space. Octrees can be stored as lists of node records or locational codes (linear octrees) [19].

The matrix octree corresponding to a $(2^N \times 2^N \times 2^N)$-dimensional object space is given as an $(M \times 8)$-dimensional matrix of all the GRAY node descriptions [20]

$$V = [F_1, F_2, F_3, \ldots, F_M]^{\mathrm{T}}, \tag{7}$$

where M is the number of GRAY nodes. A GRAY node at level k of octree associated with $(2^k \times 2^k \times 2^k)$-dimensional volume is written as an ordered eight-tuple

$$F_i = (S_{i1}, S_{i2}, S_{i3}, \ldots, S_{i8}), \tag{8}$$

where F_i denotes the parent node and S_{im}, $m = 1, \ldots, 8$, denote its eight children at the $(k-1)$ level. Equally, as in the case of quadtree, if S_{im} is a non-terminal node, then

$$S_{im} = r, \quad \text{for GRAY octant}, \tag{9}$$

where $r = 2, 3, \ldots$, refers to the position (row) in vector (7) where the record F_r corresponding to this GRAY octant is described. If S_{im} is a terminal node, we substitute the corresponding label assignment into (8), e.g.,

$$S_{im} = \begin{cases} -v & \text{for full octant,} \\ 0 & \text{for forbidden octant,} \end{cases} \tag{10}$$

where $v > 0$. The matrix octree is again a minimum and top–down description of a regular octree and, compared to the standard pointer-oriented octree description [13], requiring 10 entries for each node, in this case, only eight integers are needed and only for the non-terminal nodes.

Analogously, as by matrix quadtrees, different negative integers v in (10) can be used for storing distance data and instead of (10), the following substitutions can be considered:

$$S_{im} = \begin{cases} -d_{im} & \text{for free octant,} \\ 0 & \text{for forbidden octant,} \end{cases} \qquad (11)$$

where d_{im} is the distance value associated with the given free octant.

3. Distance maps

A number of distance-map algorithms, more or less complex and accurate, have been developed. Computational implementations of these distance maps invariably involve discretized or grid-based representation of the planar and spatial domains, over which the distance map is defined. As the presence of forbidden areas (obstacles) is generally considered, the essence of these methods for computing distance map is to approximate the Euclidean distances between remote pairs of pixels/voxels by propagating distance information locally and adding increments that represent distances between neighboring pixels/voxels [21–23]. The distance map is represented in an array with each pixel/voxel containing an integer, standing for the distance between it and the given (goal) pixel/voxel. A main drawback of such discretized representations, however, is the large amount of memory required to store them. Larger spaces and/or finer resolution, of course, increase this usage exponentially.

A more memory-efficient method than 2D/3D binary array is the hierarchical tree-data structure, where large areas of domains can be represented by a few large blocks in the corresponding quadtree/octree. Thus, applying the distance map to quadtrees/octrees instead of grids, will significantly improve the storage efficiency and the execution time. In the following, the distance map of the set of free-space blocks in 2D or 3D is fully oriented to path planning. The aim is to determine, for each block of the free space, the minimum distance in the chosen metrics, between that block and the goal block. Hence, the distance is considered as a path through free blocks given by local distance increments. Using matrix quadtree and octree representations with substitutions according to (6) and (11), respectively, enables to store the distance values to the matrix entries.

3.1. Quadtree distance map

The distance map for a quadtree can be defined as a function that yields for each free quadrant in the tree, the distance or the path length (in the chosen metric) to the quadrant containing the given (reference or goal) point G. In this case, the distance map is computed rather at squares of different sizes than at pixels [6,13,24]. Therefore, the definition of distance is oriented to the nodes of the quadtree and to the type of neighborhood chosen.

In this paper, the neighbors in four principal directions will only be considered, i.e., the pairs of edge neighbors in the horizontal and vertical directions. The distance between two neighboring nodes of quadtree can be defined in different ways. In the simplest case, the distance corresponds to the size of neighboring node (given by its level), and the distance map calculates the sum of neighboring nodes' sizes from the reference/goal node (quadrant) to the respective one. Hence, the global distances in the scene are approximated by propagating local distances, i.e., distances between neighboring quadrants.

The quadtree distance map uses an artificial wave starting in a given quadrant G, to efficiently search the quadtree, in order to find and assign the shortest paths to other quadrants. The wave propagates outward from G. The boundary of the propagating wave at any point in time is called the wavefront. It encloses the area that has been searched by the wave at a particular point in time. When the wave assigns a distance value to a particular quadrant, that quadrant is said to be covered by the wave. This process is continued until the distance wave flows through the whole free space.

The distance-map computation algorithm requires examining all the neighbors of each free node, hence, its efficiency strongly depends on the neighbor-finding algorithm used. The strategy for repetitive neighbor finding [10] can accomplish, effectively, the distance map of quadtree, because generating a neighbor can be carried out in $O(N)$ time, where N is the number of quadtree levels. As the strategy is based on matrix quadtrees, marking the free nodes with their distances from the node G, i.e., updating matrix entries, is a trivial operation performed with the quadtree traversal during neighbor finding. The distances are put into the corresponding matrix entries as integers with minus sign. If all the free quadrants are covered, the transformed quadtree can be used to find the shortest path from any free quadrant to the quadrant G.

To illustrate the process of quadtree distance map, assume that the reference node is the uppermost one on the left of the scene in Fig. 3. It is reasonable to use the reference node size (8 pixels) as the starting value of subsequent mapping. Then, for its two-edge neighbors, the values are computed as increments, i.e., $8+8$, and so on. Naturally, the incrementation is performed only once for every node. The distance-wave front flows around obstacles. The distance values after three steps of mapping procedure are written into the blocks in Fig. 3, the uncovered nodes are assigned a value 1.

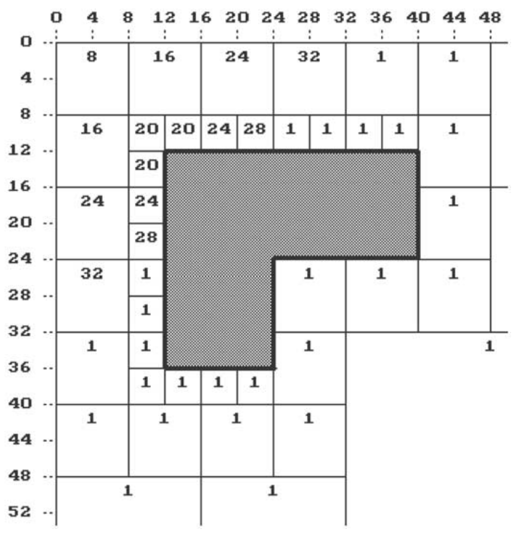

Fig. 3. Distance wave propagation in 2D.

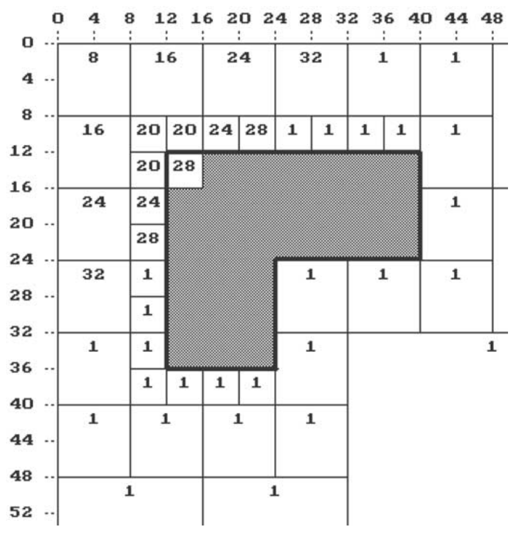

Fig. 4. Distance wave propagation in weighted area.

In the case of weighted distance map, the distance increments for blocks of weighted obstacles are multiplied by the corresponding weight factor, representing the amount of the difficulty or loss required to travel through that particular block (such as sand, grass, etc.), as compared to traveling over a flat, smooth surface block (corresponding to a minimum weight). This is shown in Fig. 4, where the first quadrant of a weighted region with factor two, has been reached and the distance wave front has penetrated the weighted node. Therefore, this node gains the value with penalized increment, i.e., $20+2\times4=28$.

Finally, note that the distance map based on the matrix quadtree and using the repetitive neighbor-finding strategy [10] is very quick and the overall execution time depends on the number of free nodes. It requires no additional data fields and the changes because of nodes' marking are reversible. Hence, clearing (simple relabeling of all negative entries in the matrix to -1) and re-computing the map for another goal point is comfortable.

3.2. Octree distance map

The distance map for an octree can be defined as a function that yields for each free octant in the tree, the minimum distance (in the chosen metric) to the octant containing the given goal point G. In this case, the distance map is computed rather at cubes of different sizes, than at voxels, and the definition of distance is oriented to the nodes of octree and to the type of neighborhood chosen.

For the sake of simplicity, the neighbors in six principal directions only will be considered, i.e., the pairs of side neighbors in the horizontal, vertical and front–back directions. The distance between two neighboring nodes of octree corresponds to the size of neighboring node, hence, the distance map sums the sizes of neighboring nodes from the goal node (octant) to the respective one. The global distances in the scene are again approximated by propagating local distances, i.e., distances between neighboring octants.

The octree distance map uses an artificial spatial wave starting in a given octant G to efficiently search the octree in order to find the shortest paths to other octants. The distance-map computation algorithm requires to examine all the neighbors of each free octant, hence, again its efficiency strongly depends on the neighbor finding algorithm used. Recently proposed repetitive neighbor-finding strategy in octrees [11] can be used to accomplish, effectively, the distance map of octree by marking every free node with its distance from the goal node. The matrix octree with substitution by (11) enables direct storing of these extra data as they can be put into the corresponding matrix entries as integers with minus sign. The execution time is again $O(N)$, where N is the number of octree levels.

An illustrative example of spatial distance-wave propagation after one step is shown in Fig. 5, where the goal point is in the leftmost bottom octant with the starting distance value 16. The neighboring free nodes are assigned the distance values 24 and 32. The whole distance map of the scene is stored to the matrix octree in Table 3 and requires $8\times6=48$ integers. The corresponding pointer based description of octree distance map for the same scene requires much more data [6].

It is worthwhile to note that the distances by this approximation can differ more significantly from the real ones than in the quadtrees. However, it is

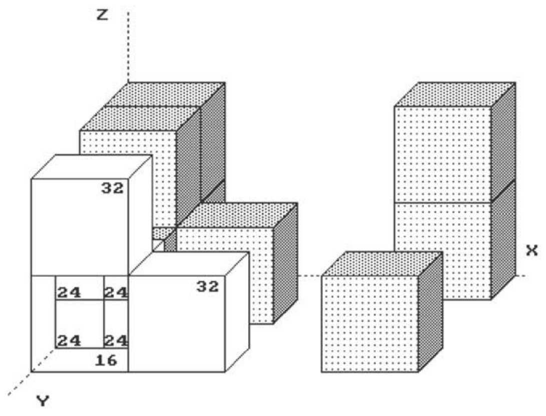

Fig. 5. Distance wave propagation in 3D.

Table 3
Octree distance map

F_i/S_{ij}	0	1	2	3	4	5	6	7
1	2	3	4	5	−96	−120	−64	−96
2	0	−104	0	0	0	−88	0	−72
3	−88	0	−72	−88	−104	0	−88	−104
4	6	−40	−16	−32	−40	−56	−32	−48
5	−56	−72	−48	0	−72	−88	−64	−80
6	0	0	−24	−24	0	0	−24	−24

subordinate, because they serve only as virtual values by path planning.

Equally as in the 2D case, the distance map based on the matrix octree and using the repetitive neighbor-finding strategy [11] is very quick and the execution time depends on the number of free octants. No extra data fields are required and the changes in octree are reversible. Hence re-computing the map for another goal point is very easy.

4. Path planning in 2D

Discretized distance maps have been used in robotics for path planning and efficient collision-detection application in static environment. The above described low-cost implementation of distance maps can serve as the basis for effective finding of optimal sequences of free nodes which can be used for computing optimized path increments through the corresponding squares/blocks.

4.1. General approach

Let the robot's 2D environment (workspace) be represented by a matrix quadtree. Given the goal point in the workspace, first we determine the corresponding quadtree leaf node G, representing the region of the space containing this point. For the goal node G, the distance map of workspace is performed.

After the distance map of quadtree, a path between any start point S and G consisting of the non-obstacle nodes of the quadtree can be planned. For any starting point S within the environment representing the initial position of the mobile robot, the path to the goal is traced by walking downhill, via the steepest descent approach. Beginning with node S, the transformed quadtree representation is used for finding an optimal non-obstacle leaf node adjacent to the node being processed, i.e., the neighbor node with the least distance value is chosen as optimal. For neighbor finding, the same method is again used, as for the distance map with the execution time of $O(N)$. The distance optimal node is included into the path list. The process continues with searching an optimal neighbor for this node and is repeated up to reaching the neighbor node identical with the node G. The result is a list of quadtree nodes that determines a sequence of adjacent free squares (ordinarily of varying sizes). This free space includes at least one collision-free path between the start and goal nodes. If there is no downhill path, then it can be concluded that there is no path from the start node to the goal node.

A number of strategies can be used to generate a path through the free-space quadrants. Computing optimized path through these squares can be based on the neighbor-finding method used, namely, a modified variant of node encoding was proposed for the neighbor finding in [10], where the locational codes are represented by a variable number of quaternary digits. This method of addressing nodes indicates the size of quadrants relative to the image dimensions implicitly, by the number of digits. The quaternary digits of locational code can be represented in binary form. A node at the nth level of quadtree, say A, is then given as

$$A = \begin{pmatrix} x_0 y_0 \\ x_1 y_1 \\ . \\ x_n y_n \end{pmatrix} \tag{12}$$

where x_i, y_i are 0 or 1, and $0 \leqslant n < N$. This form enables to compute the coordinates (x_A, y_A) of the corresponding square (upper corner on the left) and its dimension d_A, as follows:

$$x_A = \sum_{i=0}^{n} x_i \times 2^{N-i-1}, \tag{13}$$

$$y_A = \sum_{i=0}^{n} y_i \times 2^{N-i-1}, \tag{14}$$

$$d_A = 2^{N-n-1}. \tag{15}$$

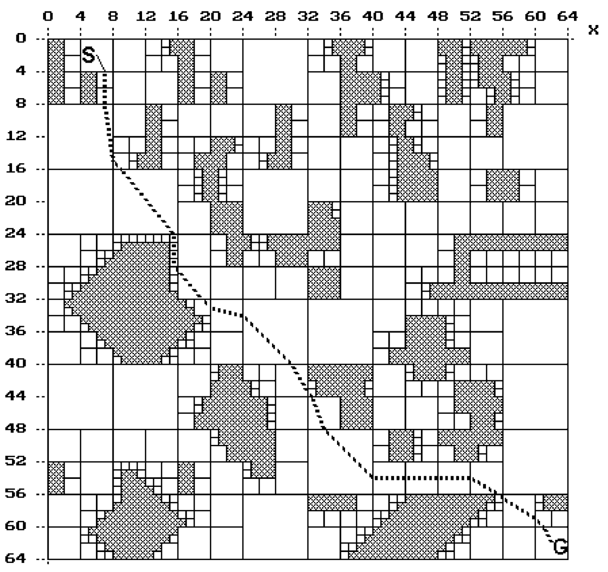
Fig. 6. Path planning in 2D scene.

A reasonable strategy would be to steer through the middle of the intersection of the entry and exit boundary edges of quadrants. Such a strategy generates relatively "safe" paths. The optimized path through a free node is the line connecting the center of common side with the previous node and the center of the common side with the following node. The center points can be easily computed using relations (13)–(15). It is worthwhile to note that these relations can be used also reversely to find the node of quadtree corresponding to a point given by its coordinates.

To illustrate the feasibility of the proposed path-planning approach, an example is shown in Fig. 6, where the robot workspace is full of obstacles. A binary array is shown, with start and goal points marked, along with an indication of the path determined by the algorithm. The path length given by the distance map is $L_\mathrm{d} = 96$ units, while the optimized path length is $L_\mathrm{op} = 90,9697$ units. The scene is identical with that of [1,4], for comparison, and its distance-map representation requires only $262 \times 4 = 1048$ integers.

4.2. Extended quadtrees

It is well known that standard quadtree-based approaches cannot guarantee finding shortest paths because of the nature of the quadtree itself. In some cases, the quadtree data structure may be sensitive to obstacle placement (position and dimensions), where non-optimal solutions may be generated when an obstacle is located on or near the boundary of a large free quadrant. Previous suggestions for fixing this problem require repositioning of the original quadtree [12], or using an elaborate approach based on the so-called framed quadtrees [9].

Very simple solving of the above problem is based on the so called extended quadtrees [25]. The k-extended quadtree is defined as the quadtree, where the nodes of interest (black or white) are not in the minimum form (i.e., given as maximal blocks), but correspond to the full grid of cells of required k-level dimensions. Proper application of k-extended quadtrees or locally extended ones can lead to significant improvements.

Such a pathological case of quadtree-based path planning using distance map is shown in Fig. 7. The scene is described by (15×4) matrix of integers and its distance map is in Table 4. The existence of two large free blocks in the neighborhood of given obstacle has

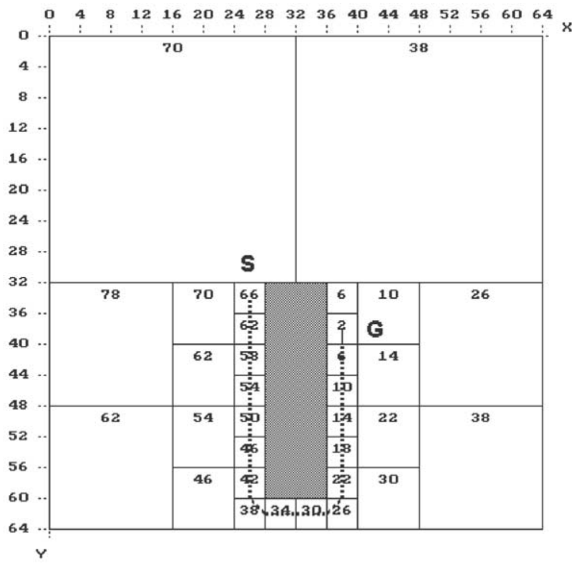
Fig. 7. Path planning using original quadtree.

Table 4
Distance map for original quadtree

P_i/Q_{ij}	0	1	2	3
1	−70	−38	2	3
2	−78	4	−62	5
3	6	−26	7	−38
4	−70	8	−62	9
5	−54	10	−46	11
6	12	−10	13	−14
7	14	−22	15	−30
8	−66	0	−62	0
9	−58	0	−54	0
10	−50	0	−46	0
11	−42	0	−38	−34
12	0	−6	0	−2
13	0	−6	0	−10
14	0	−14	0	−18
15	0	−22	−30	−26

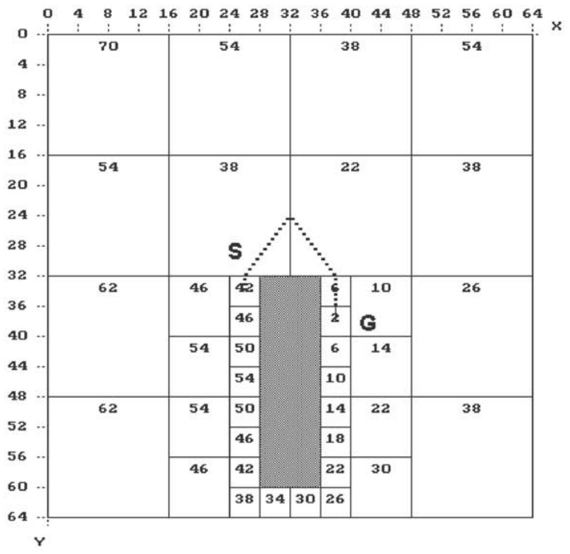

Fig. 8. Path planning using 4-extended quadtree.

Table 5
Distance map for original quadtree

P_i/Q_{ij}	0	1	2	3
1	16	17	2	3
2	−62	4	−62	5
3	6	−26	7	−38
4	−46	8	−54	9
5	−54	10	−46	11
6	12	−10	13	−14
7	14	−22	15	−30
8	−42	0	−46	0
9	−50	0	−54	0
10	−50	0	−46	0
11	−42	0	−38	−34
12	0	−6	0	−2
13	0	−6	0	−10
14	0	−14	0	−18
15	0	−22	−30	−26
16	−70	−54	−54	−38
17	−38	−54	−22	−38

caused significant error in path planning, where the approximate path length $L_d = 64$ and the optimized one $L_{op} = 61,6568$. However, simple splitting of these two free blocks, i.e., using extended quadtree, can fix this problem as is shown in Fig. 8, at the cost of adding two rows to the scene matrix description (i.e., 2×4 more integers). The distance map of 4-extended quadtree is in Table 5. The sequence of neighboring free nodes determining the path is

$$P_{dec} = \{(2110); (03); (12); (3001); (3003)\}$$

and the corresponding binary equivalents in the form of (12) are

$$P_{bin} = \{(01,10,10,00)^T; (00,11)^T; (10,01)^T; (11,00,00,10)^T; (11,00,00,11)^T\}.$$

Using the relations (13)–(15) gives the following block coordinates and dimensions:

$$P_{coor} = \{[(24,32),4]; [(16,16),16]; [(32,16),16]; [(36,32),4]; [(36,36),4]\}.$$

Now $L_d = 40$ while the computed optimized path length $L_{op} = 28,0000$.

Evidently, the extended matrix has eliminated the above problem and further extensions can lead to even shorter paths. Path in 3-extended quadtree (described by $29 \times 4 = 116$ integers, $L_d = 24$, $L_{op} = 22,422$) is shown in Fig. 9 and in 2-extended quadtree ($85 \times 4 = 340$ integers, $L_d = 24$, $L_{op} = 21,6568$) is shown in Fig. 10. A path identical with framed quadtree approach is shown in Fig. 11, where the quadtree is globally 2-extended and locally 1-extended ($175 \times 4 = 700$, $L_d = 22$, $L_{op} = 18,1623$).

Extended quadtrees may generally shorten the path lengths at the expense of required memory space for quadtree description. Such an improvement can be demonstrated on the scene in Figs. 12 and 13. In the first case, the original minimum matrix quadtree was used for the path planning requiring $166 \times 4 = 664$ integers

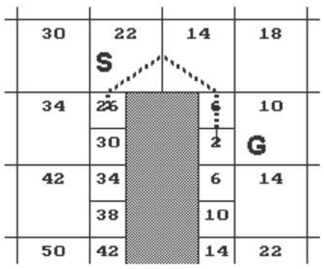

Fig. 9. Path in 3-extended quadtree.

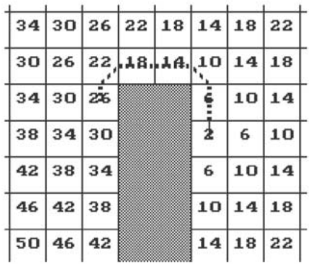

Fig. 10. Path in 2-extended quadtree.

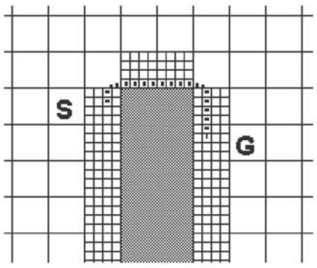

Fig. 11. Path in globally 2-extended and locally 1-extended quadtree.

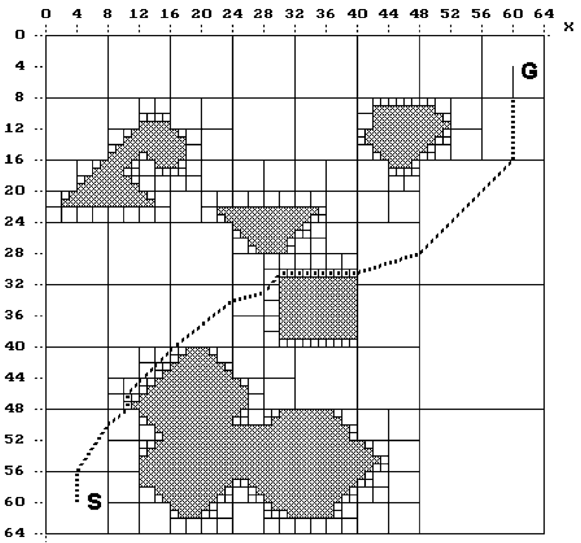

Fig. 12. Path planning using original quadtree.

($L_\mathrm{d} = 93$, $L_\mathrm{op} = 89,1948$). In the second case, 3-extended quadtree was used, requiring $206 \times 4 = 824$ integers and giving $L_\mathrm{d} = 92$, $L_\mathrm{op} = 83,6789$. The increase of memory space for distance map in extended quadtree is 24%, the path is about 6% shorter.

4.3. Weighted regions

The technique in this case is similar to that of the homogeneous free space described above. The robot environment is partitioned into a set of regions, each of which is associated with a given weight factor. A path through a weighted region assumes a cost that is determined by the defined distance of the path in that region and that region's weight factor. Given the goal point in the workspace, first we determine the corresponding quadtree leaf node G, representing the region of the space containing this point. For this goal node G, the weighted distance map of workspace is performed in the way described in Section 3. Then, from any starting point in the environment, the shortest path to the goal can be traced by following the path of steepest descent. Note that in the case of weighted path planning, the start and/or goal points can be positioned also in the weighted regions.

The computer simulation results of solving the path planning in weighted robot workspace are presented for the 2D scene according to Fig. 14. If the region A is associated with the weight factor two, while the other obstacles are impenetrable, the weighted distance optimal path passes through this region as is shown by the dotted line between the start point S and the goal point G. However, if the weight factor of the region A is three or higher, the resulting path will bypass this region as it is shown in Fig. 15.

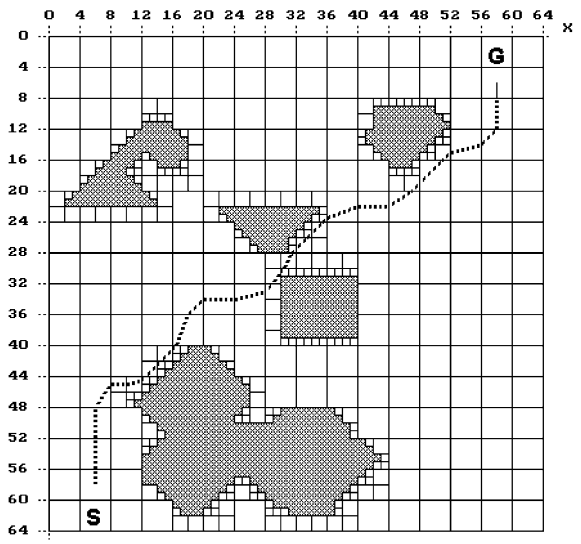

Fig. 13. Path planning using 3-extended quadtree.

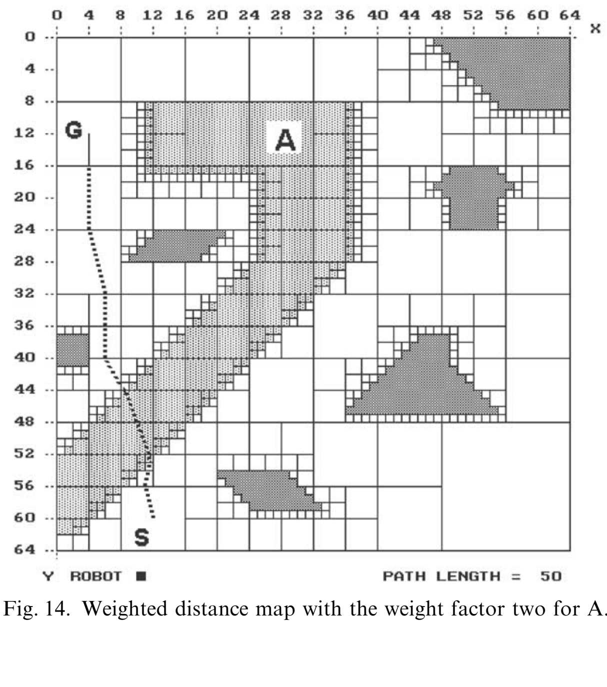

Fig. 14. Weighted distance map with the weight factor two for A.

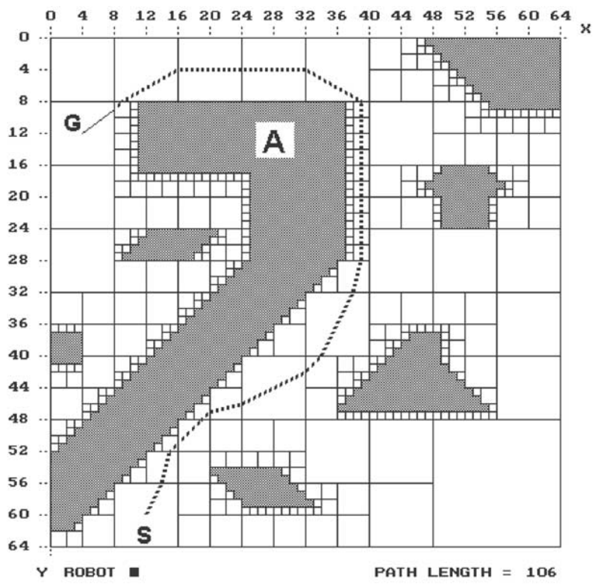

Fig. 15. Weighted distance map with the weight factor three for A.

5. Path planning in 3D

Spatial path planning for a robot or a robot arm in 3D workspace is a complex problem because more factors must be considered, e.g., dimensions, interactions between links, the degrees of freedom, etc. In many applications, the problem may be decomposed into one or more single-point cases. Then, the above low-cost implementation of distance map in 3D can be used for effective spatial path planning.

The problem of single-point path planning can be stated similarly as in the case of 2D. Let the robot's 3D environment or workspace be represented by a matrix octree. According to the dimensions of the robot (or a part of robot arm being considered), we minorize the octree extending the obstacles and shrinking the free space accordingly. Now, we can operate with the robot as with a point with three degrees of freedom, i.e., translations along the x, y, and z axes.

For the given goal node G, the distance map of workspace is performed and the data are stored into the matrix octree. Then, for the given start node, the sequence of neighboring free octants (ordinarily of varying sizes) is generated by following the steepest descent path. This free space includes at least one collision-free spatial path between the start and goal nodes.

If desired, a path with optimized length through these cubes can be computed. Again, a modified variant of node encoding can be considered, where the locational codes are represented by a variable number of base eight digits, i.e., n digits for a node at the n-th level of octree. The codes are equivalent to a depth-first traversal. If the base eight digits of locational code are represented in the binary form, then a node at the n-th level of octree, say B, is given as

$$B = \begin{pmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{pmatrix}, \tag{16}$$

where x_i, y_i, z_i are 0 or 1, and $0 \leqslant n < N$. This form provides the coordinates (x_B, y_B, z_B) of the given cube and its dimension d_B, as

$$x_B = \sum_{i=0}^{n} x_i \times 2^{N-i-1}, \tag{17}$$

$$y_B = \sum_{i=0}^{n} y_i \times 2^{N-i-1}, \tag{18}$$

$$z_B = \sum_{i=0}^{n} z_i \times 2^{N-i-1}, \tag{19}$$

$$d_B = 2^{N-n-1}. \tag{20}$$

and is used for the neighbor finding in [11]. The free nodes in the sequence generated by path planning are in this form, hence, the relations (17)–(20) can be used for the generation of paths with optimized length.

An example of spatial path planning using octree distance map is shown in Fig. 16. A very simple 3D scene is shown with start and goal points marked, along with an indication of the path. The octree graph of the 3D scene with the distance data is shown in Fig. 17 and the corresponding matrix representation in Table 6, where the negative entries correspond to the distances of the nodes from the goal node G. The spatial location of the free blocks is shown in Fig. 18. The memory saving is evident. The distance map of the scene stored to the matrix in Table 6 requires $8 \times 7 = 56$ integers, while the corresponding pointer-based description for each node requires a record with 11 fields (nine pointers to the parent and eight children nodes, one field for node type

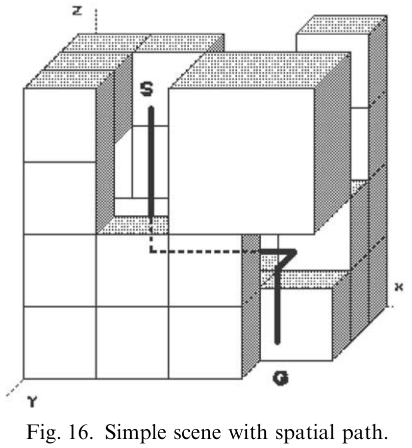

Fig. 16. Simple scene with spatial path.

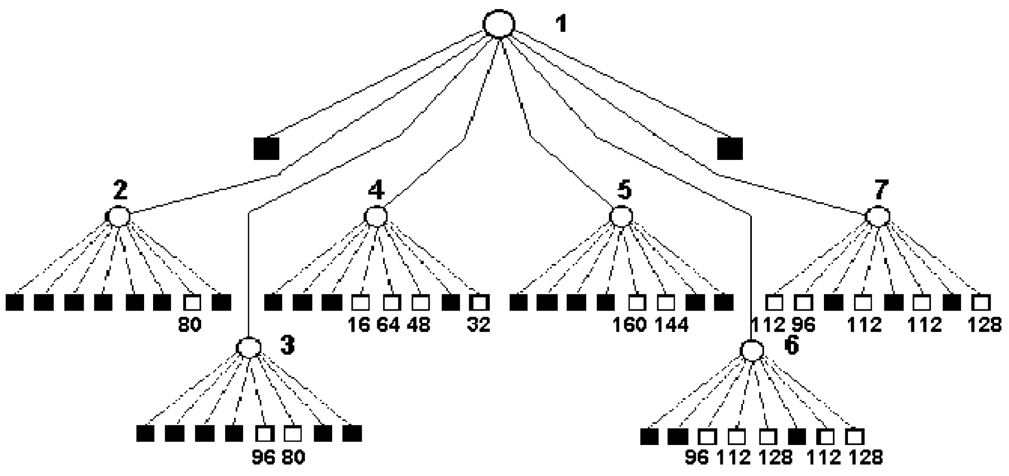

Fig. 17. Scene octree graph with distance data.

Table 6
Distance map for 4-extended quadtree

F_i/S_{ij}	0	1	2	3	4	5	6	7
1	0	2	3	4	5	6	7	0
2	0	0	0	0	0	0	−80	0
3	0	0	0	0	−96	−80	0	0
4	0	0	0	−16	−64	−48	0	−32
5	0	0	0	0	−160	−144	0	0
6	0	0	−96	−112	−128	0	−112	−128
7	−112	−96	0	−112	0	−112	0	−128

and one field for distance value), giving $11 \times 57 = 627$ fields.

Note that the path is shown as a line connecting the centers of adjacent octants and the sequence of neighboring octree nodes determining the path in this example is

$$P_{\text{dec}} = \{(6,5);\ (6,1);\ (2,5);\ (3,4);\ (3,5);\ (3,7);\ (3,3)\}.$$

These nodes can be used for exact localization of the given cubes and computing of the points, where the optimized path should enter and leave the cube. According to (16), the corresponding binary equivalents are

$$\begin{aligned}P_{\text{bin}} = \{ & (011,101)^{\mathrm{T}}; (011,100)^{\mathrm{T}}; (010,101)^{\mathrm{T}};\\ & (110,001)^{\mathrm{T}}; (110,101)^{\mathrm{T}};\\ & (110,111)^{\mathrm{T}}; (110,110)^{\mathrm{T}}\}\end{aligned}$$

and using (17)–(20), the following coordinates and dimensions are obtained:

$$\begin{aligned}P_{\text{coor}} = \{ & [(16,32,48),16]; [(16,32,32),16]; [(16,32,16),16];\\ & [(32,32,16),16]; [(48,32,16),16]; [(48,48,16),16];\\ & [(48,48,0),16]\}.\end{aligned}$$

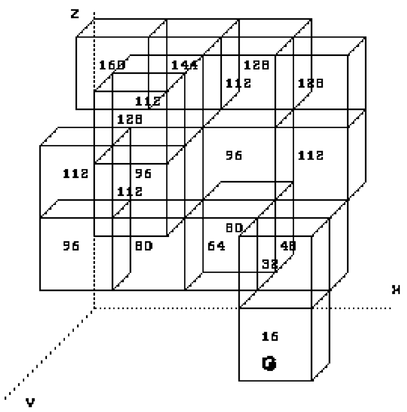

Fig. 18. Free blocks with distances.

Now, these values can be used for computing a path with optimized length.

Finally, a more complex example of spatial path planning using distance map is shown in Fig. 19, where the workspace is full of obstacles. The generated path goes through openings in more objects. The memory requirements for the scene and distance map storing are $273 \times 8 = 2184$ integers.

The similarity between quadtrees and octrees implies that all the approaches proposed for path planning in 2D can be simply extended for 3D cases. It means that the concept of extended trees and the weighted regions can be applied even in octrees, if required.

6. Conclusion

A low-cost implementation technique has been presented for using distance maps in 2D and 3D path planning. The proposed hierarchical tree-oriented distance map approach requires less computational effort, as always the largest area (quadrant, octant) of the free space is investigated, and less memory than a pixel or

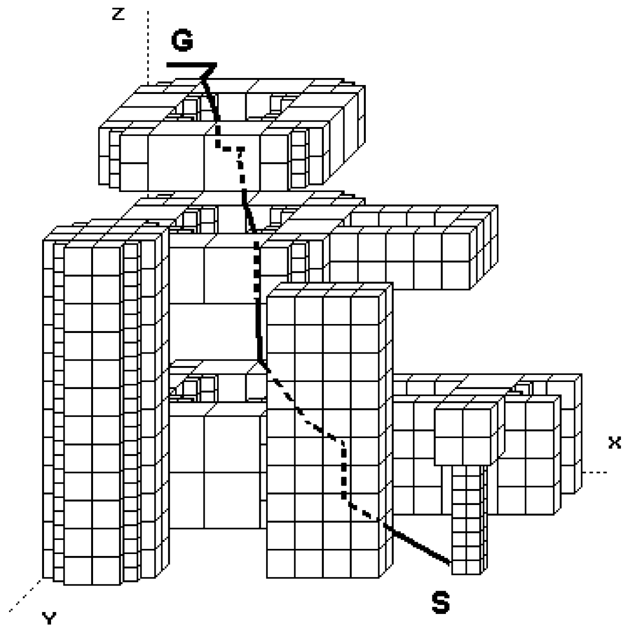

Fig. 19. Complex 3D scene with distance map path planning.

voxel-based distance map approach. Moreover, using matrix forms of quadtree and octree representations for 2D and 3D scenes requires even less memory space than the standard pointer-based forms because always only non-terminal nodes are described using the least possible number of data.

By definition, additional data related to given regions can be directly stored to the matrix quadtree and octree representations, hence they are appropriate not only for computing but also for storing the planar and spatial distance data. Consequently, contrary to the other known approaches, no extra fields for storing distance maps are needed. This is an important factor especially in the case of mobile autonomous robots.

Generation of distance maps and path planning in 2D and 3D robot environments is effectively performed using the recently proposed strategies for repetitive neighbor finding in quadtrees and octrees. The resulting sequences of free region quadrants and octants enable to compute distance-optimized paths to the chosen resolution level. Not only scenes with homogeneous free space but also scenes with weighted regions have been considered and the potential of using extended trees has been presented to overcome some inherent problems of tree-oriented path planning. More examples are included and analyzed for 2D and 3D robot workspaces to illustrate the efficiency of proposed methods.

Finally note that the matrix form of regular quadtree and octree descriptions can be used in other tree-based path-planning approaches [6,9,27], where they may lead to significant reduction of required memory space and execution time. For example, the distance map approach can be combined with the potential field method to keep the paths away from obstacles [26], while profiting from the proposed low-cost implementation.

Acknowledgements

The author gratefully acknowledges financial support from the Slovak Scientific Grant Agency (VEGA).

References

[1] Kambhampati S, Davis LS. Multiresolution path planning for mobile robots. IEEE J Robotics Automat 1986;2(3):135–45.
[2] Latombe JC. Robot motion planning. Boston: Kluwer Academic Publishers, 1991.
[3] Zelinsky A. A mobile robot exploration algorithm. IEEE Trans Robotics Automat 1992;8(6):707–17.
[4] Noborio H, Naniwa T, Arimoto S. A quadtree-based path-planning algorithm for a mobile robot. J Robotic Systems 1990;7(4):555–74.
[5] Vörös J. Simple path-planning algorithm for mobile robots using quadtrees. In: Kopacek P, editor. IFAC Workshop on Human-Oriented Design of Advanced Robotics Systems, 1995. p. 197–202.
[6] Jung D, Gupta KK. Octree-based hierarchical distance maps for collision detection. J Robotic Systems 1997;14:789–806.
[7] Vörös J. Trajectory planning in 3D workspaces represented by octrees. In: Villa A, Kopacek P, editors. Modelling, management and control. IFAC Workshop on Manufacturing Systems, 1997. p. 331–6.
[8] Chen DZ, Szczerba RJ, Uhran JJ. A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment. IEEE Trans Robotics Automat 1997;13(5):668–81.
[9] Szczerba RJ, Chen DZ, Uhran JJ. Planning shortest paths among 2D and 3D weighted regions using framed-subspaces. Int J Robotics Res 1998;17:531–46.
[10] Vörös J. A strategy for repetitive neighbor finding in images represented by quadtrees. Pattern Recognition Lett 1997;18(10):955–62.
[11] Vörös J. A strategy for repetitive neighbor finding in octree representations. Image Vision Comput 2000;18(14):1085–91.
[12] Samet H. Distance transform for images represented by quadtrees, IEEE Trans. Pattern Anal Mach Intell 1982;4(3):298–303.
[13] Samet H. The design and analysis of spatial data structures. Reading: Addison Wesley, 1990.
[14] Gargantini I. An effective way to represent quadtrees. Commun ACM 1982;25(12):905–10.
[15] Vörös J. Top down generation of quadtree representation using color-change code. Comput Artificial Intell 1994;13(1):91–103.
[16] Vörös J. Mobile robot path planning among weighted regions using quadtree representations. In: Pichler F, Moreno-Diaz R, Kopacek P, editors. Lecture notes in computer science, Vol. 17. Berlin, Heidelberg, New York: Springer, 2000. p. 239–49.
[17] Jackins CL, Tanimoto SL. Oct-trees and their use in representing three-dimensional objects. Comput Graphics Image Process 1980;14:249–70.
[18] Meagher D. Geometric modeling using octree encoding. Comput Graphics Image Process 1980;19:129–47.
[19] Gargantini I. Linear octtrees for fast processing of three-dimensional objects. Comput Graphics Image Process 1982;20:365–74.

[20] Vörös J. Approximate representations of multi-object scenes in robotics. In: Boromisza T, editor. IFAC Symposium on Intelligent Components and Instruments for Control Applications, 1994. p. 223–8.

[21] Borgefors G. Distance transformations in digital images. Comput Vision Graphics Image Process 1986;34:344–71.

[22] Borgefors G, Hartmann T, Tanimoto ST. Parallel distance transforms on pyramid machines: theory and implementation. Signal Process 1990;21(1):61–86.

[23] Piper J, Granum E. Computing distance transformations in convex and non-convex domains. Pattern Recognition 1987;20(6):599–615.

[24] Shaffer CA, Stout QF. Linear time distance transforms for quadtrees. CVGIP: Image Understanding 1991;54(2):215–23.

[25] Vörös J. Using extended quadtrees in robot path planning. Proceedings of the Seventh Workshop on Robotics in Alpe–Adria–Danube Region, 1998. p. 451–6.

[26] Zelinsky A. Using path transforms to guide the search for findpath in 2D. Int J Robotics Res 1994;13(4):315.

[27] Yahja A, Singh S, Stentz A. An efficient on-line path planner for outdoor mobile robots. Robotics Autonomous Systems 2000;32(2–3):129–43.