JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 1355-1366 (2006)

Efficient and Complete Coverage of 2D Environments by Connectivity Graphs for Motion Planning Algorithms*

LEENA LULU AND ASHRAF ELNAGAR+
Intelligent Systems and Infrastructure Group
Department of Computer Science
University of Sharjah
P O Box 27272, Sharjah, UAE
+*E-mail: ashraf@ieee.org*

In this paper, we use an efficient art gallery-based algorithm for placing a small number of guards to inspect a 2D environment. The gallery models a 2D workspace for an autonomous robot. The proposed algorithm efficiently computes a near-optimal (small) number of guards in $O(n \log n)$ time and requires a linear storage complexity. An additional set of connection nodes are computed to form the connectivity graph, which contains all guards. This graph has far less number of vertices when compared to similar data structures used in conventional visibility-based or probabilistic-based motion planning algorithms. The resulting placement of guards and connectors can then be used as control points along the path of an autonomous mobile robot for navigation or inspection tasks. The proposed algorithm is not only offering a better performance in terms of computational cost but also ease of implementation.

Keywords: simple polygons with holes, connectivity graph, visibility graph, free path, collision avoidance, motion planning

1. INTRODUCTION

This paper describes a novel approach to path planning that is designed for autonomous mobile robots in global environments. The approach is based on the well-known Art Gallery problem [1-4], where a new robust and fast algorithm for placing guards in simple polygons with holes is developed, [5]. The polygon models a 2D workspace for an autonomous robot and the holes represent the obstacles in this workspace. The method seeks to efficiently construct a roadmap that contains the smaller possible number of nodes. Experimental results demonstrate that the proposed algorithm is fast and outperforms comparable popular path planning algorithms. Our proposed algorithm can be used in a variety of applications such as 3D model construction of a workspace, navigation and inspection tasks, medical surgery and virtual reality exploration systems. In ideal cases, the constructed data structure (roadmap) contains fewer than 1/3 times the number of nodes in similar data structures in conventional visibility graph or probabilistic-based motion planning algorithms.

The goal of the robot motion planning problem is to design autonomous robots that

Received July 8, 2004; revised October 20, 2004; accepted January 13, 2005.
Communicated by Chin-Teng Lin.

* A preliminary version "Guarding polygons with holes for robot motion planning applications," has been appeared in Proceedings of the International IEEE Conference on Systems, Man and Cybernetics, 2004, pp. 923-928.

can be asked to travel to a specific location without specifying how to go there. Such robots should have sufficient information to decide how to reach its destination. This can be achieved by setting up the work space ($\mathcal{W}$-space), which is the real world where the robot actually moves around. The problem can then be mapped into another space called the configuration space ($\mathcal{C}$-space) [6]. The configuration space is the parameter space of the robot. A robot in the $\mathcal{W}$-space is mapped to a point in the $\mathcal{C}$-space and any point in the $\mathcal{C}$-space corresponds to some placement (configuration) of the actual robot in the $\mathcal{W}$-space. In most cases, the $\mathcal{C}$-space has higher dimensionality than the actual $\mathcal{W}$-space. Nevertheless, planning in the $\mathcal{C}$-space is easier than in the $\mathcal{W}$-space, because the $\mathcal{C}$-space representation simplifies the planning problem by mapping the complex robot geometry into a single point in $\mathcal{C}$-space. Consequently, the problem can then be stated as follows: Given an initial configuration and a goal configuration, generate all collision-free paths between them and then choose the shortest one.

We present a new approach to solving the robot motion planning problem. We restrict our work to static and global 2D environments. In a static environment, no object is allowed to move except the robot. Global planning requires the availability of complete knowledge of the environment. The proposed algorithm uses the 3-coloring principle [7-9] to color the vertices of specific sub-divisions of the exterior polygon called the *y*-monotone pieces. Using triangulation, each *y*-monotone piece is subdivided into convex polygons (i.e., triangles). Our algorithm runs in $O(n \log n)$ for computing n_g guards and their placements that will lead to covering the entire workspace. Such a number is sufficient and sometimes necessary to cover a polygon with holes and it does not exceed a certain upper tight bound (i.e., $\left\lfloor \frac{n+h}{3} \right\rfloor + 1$ guards).

The computation of guards' placement ensures the complete coverage of the whole workspace. However, extra guard-nodes (called connectors) are sometimes necessary to increase the connectivity of the workspace. A roadmap is then constructed of all guards and connectors. Such connectors would bridge between the disjoint guard-islands and hence resulting in a one connected component with full connectivity for navigating the 2D workspace. Connection nodes are selected from a set of potential connectors at random, which produces expected better results when compared with a similar deterministic approach, [10]. The algorithm is robust and efficient as it improves both time and storage complexities when compared with existing algorithms. In addition, it is easy to implement when compared to other available algorithms for solving the problem.

The rest of the paper is organized as follows: Section 2 describes the proposed guard placement algorithm in detail. Then, we present the connectivity graph structure and how it is computed to connect the 2D workspace in section 3. The computational complexity of the proposed complete algorithm is analyzed in section 4. Simulation results are presented in section 5, and finally, we conclude the work in section 6.

2. GUARDS PLACEMENT

We follow the general steps used to solve the art gallery problem for simple polygons without holes based on Fisk's algorithm [11] but we modify some parts to account for the holes as required. The basic modules for placing the guards in a simple polygon *P*

start with an arbitrary triangulation of the polygon, $\mathcal{T}_{\mathcal{P}}$. The triangulation process is carried out after decomposing the polygon into so-called *y*-monotone pieces. The next step is to color the resulting graph of the triangulated polygon $\mathcal{G}(\mathcal{T}_{\mathcal{P}})$ with three distinct colors using the 3-coloring principle. The 3-coloring principle is applied on all triangular subdivisions where each vertex is assigned one of three different colors. Colors are assigned in such a way that no two adjacent vertices in the graph have the same color. Therefore, three different distributions of colors are produced, one for each color. Note that in the presence of holes, the resulting dual graph for the polygon triangulation $\mathcal{G}(\mathcal{T}_{\mathcal{P}})$ is not a tree as it is the case in hole-free polygons. This is because the triangles that surround a hole will cause a cycle in the graph. The existence of such cycles would lead, while coloring, to visiting a vertex more than once. The complete general algorithm phases for capturing the coverage and connectivity of any 2D workspace are depicted in Fig. 1. In the sequel, we discuss each phase in detail.

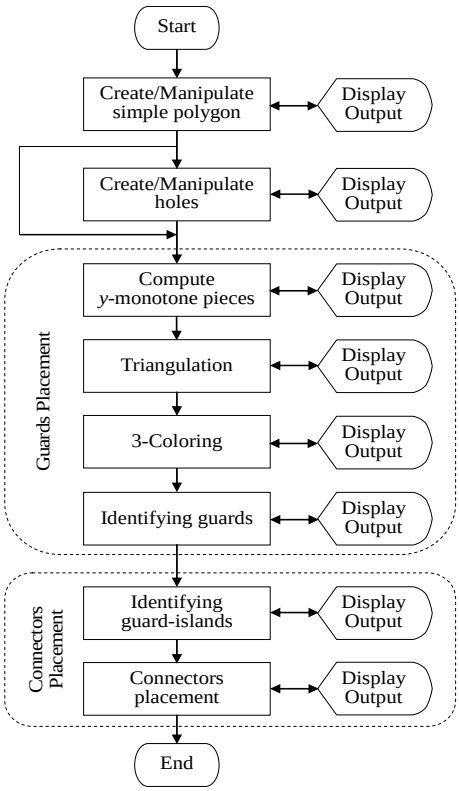

Fig. 1. Block diagram of the proposed complete algorithm.

2.1 Partitioning and Triangulation

Partitioning a simple polygon with holes that models the workspace into *y*-monotone pieces is obtained while traversing the polygon's exterior boundary. A simple poly-

gon P is called monotone with respect to a line l if the intersection of the polygon with any line l' perpendicular to l is either a line segment, a point, or empty, [10]. For example, a polygon is said to be y-monotone if it is a monotone with respect to the y-axis. This decomposition is advantageous because it simplifies the triangulation process, which takes place after computing the y-monotone pieces. The triangulation algorithm may be used for each piece to result in the triangulation of the whole planar subdivision. Many algorithms in computational geometry incorporate polygon triangulation, for example, see [12-14].

2.2 Coloring and Guard Placement

In the 3-coloring phase, the vertices of a triangulated polygon P without holes are assigned three different colors for each triangle. This process guarantees the coloring of each vertex in a triangle with a distinct color. To verify this fact, we examine what is called the dual graph $\mathcal{G}(\mathcal{T}_\mathcal{P})$, which is a structure where each triangle is represented by a node and each shared diagonal between two triangles is an arc. Notice that this structure is a tree in the case of simple polygons without holes. The holes will create cycles in the resulting dual graph $\mathcal{G}(\mathcal{T}_\mathcal{P})$, which could lead to problems in the 3-coloring phase, since some of the triangles will be visited more than once during the traversal of the dual graph.

Our proposed algorithm modifies the 3-coloring principle to handle the problem of cycles in the dual graph of the triangulated polygon $\mathcal{G}(\mathcal{T}_\mathcal{P})$. The algorithm handles polygons without holes too. The idea is based on the use of the y-monotone pieces generated in the first step and applying the 3-coloring principle on each y-monotone, independently. Ultimately, all the y-monotone pieces of the polygon will be visited during the traversal of the graph, since every y-monotone shares at least one diagonal with the other y-monotones. That is, if we maintain a graph of the y-monotone pieces of the polygon, the resulting graph is connected. If the shared diagonal between two triangles in two different y-monotones has a hole-vertex as one of its endpoints, this means that the triangles are part of a cycle in the dual graph of the triangulated polygon.

We allow recoloring of a vertex, if necessary, whenever it is revisited. Hence, the coloring procedure will continue to assign colors to other triangles' vertices of the current y-monotone. At last, when the coloring phase is completed, we count the occurrences of the colors used taking into account the recoloring of each vertex. The three sets of potential guards $\{C_0, C_1, C_2\}$ are sorted based on the following criteria:

(1) Minimize guard placement on boundary vertices.
(2) Maximize guard vertices that belong to more than one set of C_i (i.e., vertices that are recolored.

This is achieved by comparison. The guards of the best set (say C_0) are placed on the corresponding vertices. Allowing vertices on the boundary of the workspace may produce sort of undesired paths. Minimizing such vertices would lead to smoother paths. On the other hand, the recolored vertices can be considered as key nodes as they provide commonality between the three different color classes. Such nodes would act as implicit connection nodes between the guard-islands as will be discussed in section 3.

Note that the resulting n_g placements of guards entirely inspect the workspace and maintain a full coverage even when there are very narrow passages. A workspace is said to be covered by the guards set if any point in the workspace is seen from at least one guard. Fig. 2 shows a complete example of guards' placement on a polygon with holes.

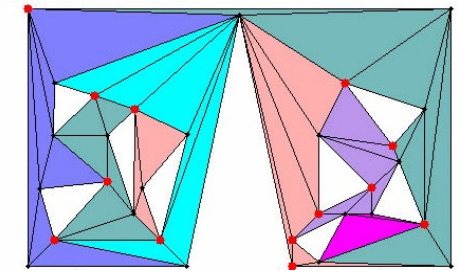

Fig. 2. An example of guards' placement of a workspace with eight holes including a very narrow short passage. 13 guards are sufficient to observe the entire workspace. The *y*-monotones are displayed with different colors and the guards are shown in red.

3. ROADMAP CONSTRUCTION

The proposed guards' placement algorithm efficiently computes a near-optimal number of guards that cover the whole workspace. The constructed set of guards constitutes a coverage of the workspace if the union of their visibility domains covers this workspace. Note that allowing recoloring the vertices in the 3-coloring process, of the guards' placement phase, increases the intersection of the visibility domains between the guards and hence maximizes the connectivity between them. The derived graph of *visible* guards can then be used as control points defining a navigational path. A collision-free path between the start and goal configurations of a robot exists if and only if both lie in the same island, where an island represents a disjoint set of visible guards to each other.

However, the full coverage of the 2D environment does not, in general, entail the connectivity of the environment. That is, there may exist some guards that do not see each other and therefore, the workspace is decomposed into a number of disjoint sets of guard-islands. In this case, connection nodes are needed to join such disjoint guard-islands. This leads to building a graph capturing the connectivity of the workspace, which will result in one island (roadmap).

3.1 Identifying Guard-Islands

In order to accomplish one connected component of the entire workspace, guards are first grouped into a set of disjoint components. The guards that 'see' each other are gathered into one island. Determining the connected components for the resulting graph of guards can be obtained in linear time using an efficient rooted-trees representation. Each node in the rooted-tree contains one guard, and each tree corresponds to one set of guards. The basic operations supported by this data structure include: MakeSet(g), which creates a new set with a single member g, Union(s_1, s_2), which merges two sets into one.

The two sets are assumed to be disjoint prior to the operation. FindSet(g) returns a pointer to a unique id of the set containing g. Two heuristics have been introduced to greatly improve the running time of the operations on the disjoint guards sets. Namely, "union by rank" and "path compression", which provide the asymptotically fastest disjoint-set data structure known, [15]. The union-by-rank heuristic maintains an upper bound on the height of each node (i.e., rank). During a Union operation, the root with the smaller rank shall point to the root of the other set. The second heuristic is used, during FindSet method, to link directly each encountered node along the visited path to the root.

3.2 Adding Connectors

After identifying the guard-islands, the connectivity graph can then be constructed in an incremental manner. At each elementary iteration, the algorithm selects a potential connector (c) at random and attempts to connect some pairs of guards belonging to different islands. This is obtained by processing all the current islands while finding guards that are visible from c. This potential connector (c) is added to the graph as a connection node only if it is visible to at least two guards belonging to two (or more) islands. Such islands are then merged into a larger one including the newly added connector as the bridging node. This process ends when all the guard-islands are merged into one, achieving the full connectivity of the workspace.

The question that might arise is: how do we choose a connector? Recall, in the 3-coloring phase of the guard placement algorithm, three different distributions of colors are produced. One of which is selected as the guards' set for covering the whole workspace. We choose one of the remaining two sets (say C_1 and C_2) as the potential set of connectors such that the recoloring of vertices are kept to minimum. This is because each recoloring of a vertex may exist in the already selected guards' set (C_0) leading to unnecessary testing for connectivity via this vertex.

Once we determine the potential set of connectors (say C_1), we proceed to select a possible connector based on a criteria explained next. A greedy approach would involve processing all connectors (n of them): ($c_i \in C_1$, $\forall i = 1, \ldots, n$) and choose the minimum number of connectors required to connect all islands. Although this approach optimizes the connectivity, it incurs a high computational cost. Therefore, we propose to use a randomized approach instead. The idea is to pick one connector c_i from C_1 at random and then proceed to compute its visibility and bridging capability. Note that, in rare cases, the roadmap construction may require the use of the second remaining set of potential connectors (C_2) as the first set may be not sufficient to capture the full connectivity of the workspace.

It is worth mentioning that randomization has attracted the interest of many researchers as a powerful technique for solving many complex problems such as inspection and planning tasks. It has been applied successfully in many applications (for example, [10]). Next, we present the pseudocode for capturing the connectivity of the workspace.

Algorithm RoadmapConstruction(I)
Input: Guard-islands: $I = \{G_1, G_2, \ldots, G_m\}$
Output: The roadmap $G(V, E)$ is constructed
1. Let C_i be the list of 'potential' connectors

2. **while** *cardinality*(G) > 1
3. Select a connector c from C_i at random
4. $G_{vis} \leftarrow \phi$
5. *guards* $\leftarrow \phi$
6. **for** all islands G_i
7. **for** all visible $g_k \in G_i$ from c
8. Select g such that $| c - g_k |$ is minimum $\forall g_k$
9. *guards* $\leftarrow g$
10. $G_{vis} \leftarrow G_i$
11. **if** *guards* $\neq \phi$ /* c is a connection node */
12. Add c to G
13. **for** all visible g_k of *guards*
14. Create edge (c, g_k)
15. Merge islands in G_{vis}

After capturing the full coverage and connectivity of the graph G, it is now possible to consider a start q_{start} and a goal q_{goal} configurations of a robot such that there exists a sequence of collision-free paths between them along G. Then, there is a guard node g_1 and a guard node g_2 in G such as q_{start} is visible from g_1 and q_{goal} is visible from g_2. If such a path exists, the solution of the problem results in a path constituted by a finite connected sequence of guards and connectors computed by the proposed algorithm.

4. COMPLEXITY ANALYSIS

For the guard placement phase, a simple polygonal workspace with n vertices and h holes can be covered with a near-optimal number of guards (n_g) in $O(n \log n)$ time and requires only an $O(n)$ storage. We first decompose the polygonal workspace P with n vertices and h holes into y-monotone polygons, which takes only $O(n \log n)$ time complexity using a plane sweep algorithm, [16]. The triangulation of each of the monotone pieces is done in a linear time based on [14]. The coloring process of the vertices of the polygon takes $O(n + c)$ running time, where c represents the number of color replacements that is $< \left\lfloor \frac{h}{3} \right\rfloor$, which is less than n. Thus, the amount of storage used by the algorithm is clearly linear. The polygon is initially stored in a special data structure called the doubly-connected edge list. Every vertex/edge is stored at most once. The subdivisions induced by P, i.e. the y-monotones, and the added diagonals in the triangulation phase are also stored in the doubly-connected edge list. As a result, the total running time of this phase is $O(n \log n)$ and uses $O(n)$ storage. The main results of the algorithm are detailed and proved in [5].

Similar approaches that solve this problem have reported higher complexities. Namely, Bjorling-Shachs and Souvaine [17] reported $O(n^2)$ computational cost whereas Hoffmann, Kaufmann and Kriegel [18] reported $O(n^3 \log n)$. Next, we analyze the computational complexity of the connectivity graph. A graph is first constructed containing the derived guards from the guards' placement phase. Initially, each node in the graph corresponds to a guard and each edge corresponds to a straight line-segment joining the

guards that are visible to each other. The population of the graph with the guards' set takes $O(n_g^2 \log n)$, where n_g is the number of guards and n is the total number of vertices in the whole workspace.

Guards are then divided into a number of connected components using a fast and efficient disjoint-set data structure that is linear in the total number of operations performed. This running time is further enhanced by using the two heuristics: union by rank and path compression. This leads to the worst-case of $O(m\ \alpha(n))$ time complexity for m disjoint-set operations on n elements (i.e., guards), where $\alpha(n)$ is a very slowly growing function, [19].

Capturing the connectivity of the workspace, exploits a randomized approach in which the running time depends on the specific random order made by the algorithm. In our case, the choice of a connector is determined by a permutation of all potential connectors. As a result, the expected running time of the algorithm would be $O(n_c n_g \log n_g)$, where n_c is the number of connectors added to the graph in order to capture the connectivity of the workspace, and n_g is the number of guards used to maintain its coverage.

5. SIMULATION RESULTS

The proposed algorithm has been extensively tested. Figs. 2, 3 and 4 present a complete example of the roadmap construction using the proposed algorithm. Fig. 2 depicts a near-optimal guard placement on a workspace cluttered with eight obstacles (holes). The algorithm starts by decomposing the environment into *y*-monotone pieces, which are shown in different colors. Next, these *y*-monotones are triangulated for coloring. The guards are placed at the color class the satisfies our proposed criteria (see section 2.2), which guarantees full coverage of the workspace.

In Fig. 3, three distinct guard-islands have been identified (for best viewing it is colored in different colors). Connection nodes are necessary to capture the connectivity of the environment. Note that this example is considered one of the limitations of the visibility-based probabilistic approach, [20]. The intersection domain between the left and right sides of the environment is very small and the probability to pick (randomly) a connection node above the tip of the dent is very low. However, our algorithm successfully captured the full connectivity through two connection nodes (displayed in blue) as shown in Fig. 4.

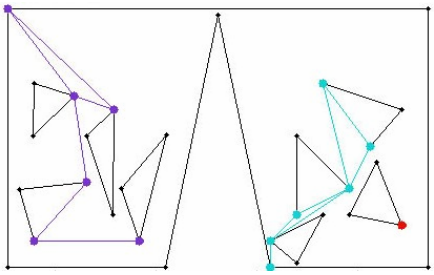

Fig. 3. The computed guard-islands for the workspace of Fig. 2. The three computed islands are displayed: First island on the left side connects all holes in its region; second island on the right side connects three holes with gray-color line segments; and the last island is the rightmost hole only.

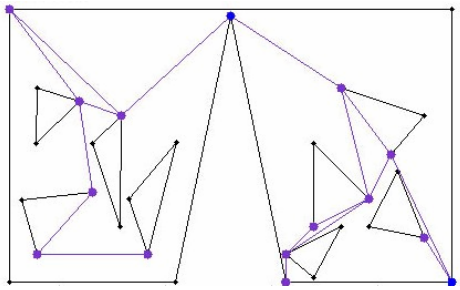

Fig. 4. The resulting connected roadmap for the same workspace of Fig. 2 after adding two connectors on the boundary of the 2D workspace. Namely, bottom right vertex to connect the islands on the right hand side and the vertex at the tip of the dent to bridge the two sides.

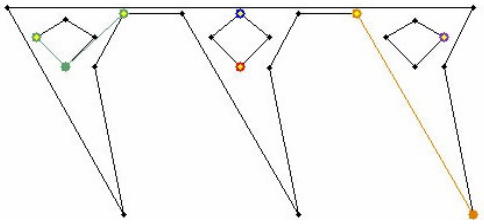

Fig. 5. Five isolated islands produced for the computed eight guards. Guard-islands are displayed in different colors.

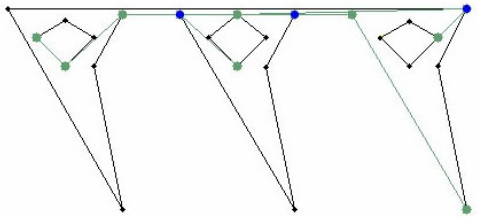

Fig. 6. The connected roadmap for the workspace of Fig. 5. Notice that three connectors are added to capture the complete connectivity of the roadmap.

Fig. 5 demonstrates another complex workspace, where nine guards are necessary to fully observe the whole region. The guards are divided into five distinct islands, based on their visibility domains, presented in different colors. The complete connectivity of the roadmap is illustrated in Fig. 6, where three connectors were added to result in one connected roadmap.

Figs. 8 and 9 illustrate the difficult case to place guards in narrow passages as it is problematic with PRM approaches. The computed guards placements provide full coverage to the whole workspace (Fig. 7). Fig. 8 depicts three isolated islands that need to be connected in order to construct the roadmap to capture its connectivity. Our proposed algorithm produced a complete connectivity for the roadmap as shown in Fig. 9.

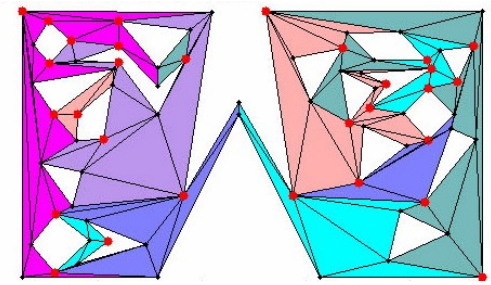

Fig. 7. Placement of guards.

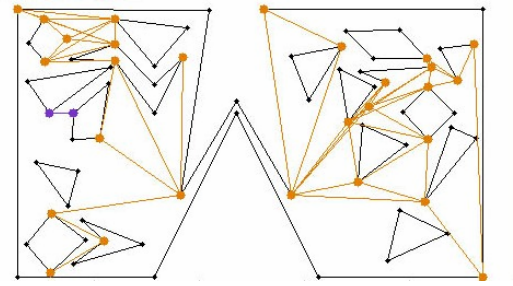

Fig. 8. A 2D workspace with a long concave narrow passage. Three islands are shown.

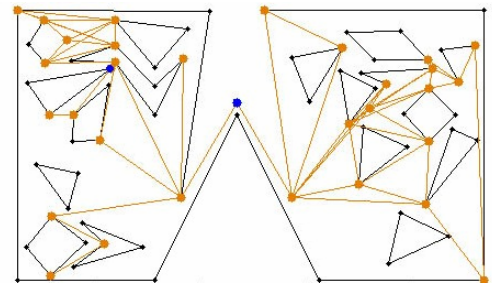

Fig. 9. The algorithm successfully captures the connectivity of the 2D workspace in the presence of long narrow passages, which are problematic for probabilistic-based approaches.

6. CONCLUSIONS

In this paper, a novel method for deciding the placement of guards necessary to observe a 2D workspace has been studied. An algorithm for placing a near-optimal set of guards that runs in $O(n \log n)$ time and uses $O(n)$ storage, where n is the total number of vertices has been presented. A modified 3-coloring algorithm is used to assign colors to the vertices of the y-monotones after applying the triangulation. The algorithm is robust and efficient. When compared to existing similar algorithms, it offers not only a better computational cost but also ease of implementation.

The algorithm proves to work very satisfactory in large workspaces, since it guarantees a complete coverage of the workspace with a small number of guards, independ-

ently of its configuration and complexity. Compared to PRM approaches, the algorithm can handle very narrow passages and difficult regions. The derived set of guards' placements in addition to the connection nodes form the connectivity graph, which is searched for a free path. This graph has far less number of vertices when compared to similar data structures used in conventional visibility-based or probabilistic-based motion planning algorithms.

REFERENCES

1. D. Haussler and E. Welzl, "Recent results on illumination problems," *Discrete Computational Geometry*, Vol. 2, 1987, pp. 127-151.
2. T. Shermer, "Recent results in art galleries," in *Proceedings of the IEEE*, Vol. 80, 1992, pp. 1384-1399.
3. L. Szabó, "Recent results on illumination problems," *Intuitive Geometry, Bolyai Society Mathematical Studies*, Vol. 6, 1972, pp. 207-221.
4. J. Urrutia, "Art gallery and illumination problems," in J. R. Sack and J. Urrutia (eds.), *Handbook of Computational Geometry*, Elsevier Science Publishers, North Holland, Amsterdam, 2000, pp. 973-1026,
5. A. Elnagar and L. Lulu, "A robust and fast algorithm for guarding simple polygons," to appear in *the 16th International Canadian Conference on Computational Geometry*, 2004.
6. T. Lozano-Pérez, "Spatial planning: a configuration space approach," *IEEE Transactions on Computers*, Vol. 32, 1983, pp. 108-120.
7. D. Matula, G. Marble, and J. Isaacson, "Graph coloring algorithms," in R. Read (ed.), *Graph Theory and Computing*, Academic Press, New York, 1972, pp. 109-122.
8. D. Johnson, "Worst case behavior of graph coloring algorithms," in *Proceedings of the Southeastern Conference on Combinatorics, Graph Theory and Computin*g, 1974, pp. 513-527.
9. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill Book Company, Boston, 2001.
10. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkoph, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 2000.
11. S. Fisk, "A short proof of Chvatal watchman theorem," *Journal of Combinatorial Theory Series B*, Vol. 24, 1978, pp. 374.
12. B. Chazelle and J. Incerpi, "Triangulating a simple polygon in linear time," in *Proceedings of the 31st Annual IEEE Symposiums on Foundations of Computer Science*, 1990, pp. 220-230.
13. A. Fournier and D. Montuno, "Triangulating simple polygons and equivalent problems," *ACM Transactions on Graphics*, Vol. 3, 1984, pp. 153-174.
14. M. Garey, D. Johonson, F. Preparata, and R. Tarjan, "Triangulating a simple polygon," *Information Processing Letters*, Vol. 7, 1978, pp. 175-179.
15. R. Tarjan and J v. Leeuwen, "Worst-case analysis of set union algorithms," *Journal of the ACM*, Vol. 31, 1984, pp. 245-281.
16. D. Lee and F. Preparata, "Location of a point in a planar subdivision and its applications," *SIAM Journal of Computing*, Vol. 6, 1977, pp. 594-606.

17. I. Bjorling-Sachs and D. Souvaine, "An efficient algorithm for guard placement in polygons with holes," *Discrete and Computational Geometry*, Vol. 13, 1995, pp. 77-109.
18. F. Hoffmann, M. Kaufmann, and K. Kriegel, "The art gallery theorem for polygons with holes," in *Proceedings of the 32nd Annual IEEE Symposium of Foundations of Computer Science*, 1991, pp. 39-84.
19. R. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM*, Vol. 22, 1975, pp. 215-225.
20. C. Nissoux, T. Simeon, and J. P. Laumond, "Visibility based probabilistic roadmaps," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3, 1999, pp. 1316-13210.

Leena Lulu received her B.Sc. and M.Sc. degrees in Computer Science from the University of Sharjah, UAE in 2002 and 2004, respectively. She spent the summer of 2002 as a research assistant before joining the Department of Computer Science at the University of Sharjah as a visiting lecturer. Her research interests are in the areas of robot motion planning and computational geometry where she has published 10 papers.

Ashraf Elnagar received his Ph.D. in Computer Science from the University of Alberta, Canada, in 1993. He spent 2 years as a lecturer and NSERC research fellow in the University of Alberta and Simon Fraser University, respectively. Then he moved to the Gulf region where he worked in the Department of Computer Science at Sultan Qaboos University, Oman, from 95 to 98. In September 1998, he joined the Department of Computer Science at the University of Sharjah, UAE, where he chaired the department between 98 and 03. He is currently an Associate Professor. His research interests include AI, robot motion planning and pattern recognition. He is a senior member of the IEEE society and has more than 60 publications in these fields. He has won several prestigious research awards, among which is the 1999 Shoman's Best Young Researcher Award in the fields of Mathematics, Computer Science and Statistics in the Arab World.