Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea • May 21-26, 2001

A Recursive Algorithm of Obstacles Clustering for Reducing Complexity of Collision Detection in 2D Environment

Jin–Liang Chen, Jing–Sin Liu and Wan-Chi Lee
Institute of Information Science 20
Academia Sinica
Nankang, Taipei 115, Taiwan, R.O.C.
Email: liu@iis.sinica.edu.tw

ABSTRACT

In applications of industrial robots, the robot manipulator must traverse a pre-specified Cartesian curve (path) with its hand tip while links of the robot safely move among obstacles. In order to reduce the costs of collision detection, the number of collision checks can be reduced by enclosing a few obstacles (a cluster) with a larger (artificial) bounding volume, e.g. by their convex hull, without cutting the specified curve. In this paper, an efficient and convergent recursive algorithm for refining an initial randomly generated set of clusters is proposed to tackle the problem of clustering convex polygonal obstacles in 2D robot's scene. Simulation results show that the proposed algorithm acquires less number of clusters and computationally more efficient. In addition, the algorithm can be easily applied to dynamic environment based on the idea of *seeds* in clusters.

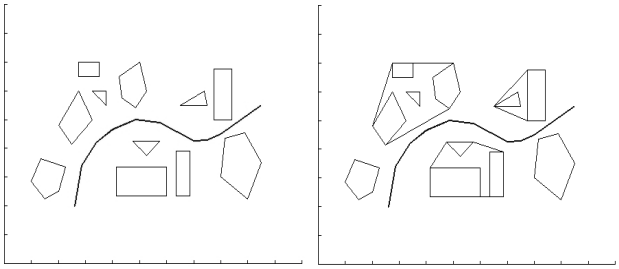
1. INTRODUCTION

Collision detection and avoidance for mobile robots or articulated robots along a prescribed trajectory is a basic step toward motion planning. The problem has been studied in many literatures [4-9] during the last two decades. In applications of industrial robots, the (mobile) robot manipulator must traverse a pre-specified Cartesian curve with its hand tip while links of the robot safely move among obstacles. As such, the complexity increases with the dimension of robot's configuration space and the number of obstacles in robot's scene (or workspace) [10]. Furthermore, the problem is more complicated if the workspace is dynamic, i.e., obstacles may be intersected, deleted, or moved. In order to reduce the costs of collision detection, attempts are 1) to simplify the shapes of objects to facilitate collision check of complex geometry, such as polyhedrons, spheres, ellipsoids or planar convex polygons; and 2) to reduce the number of collision checks by enclosing a few obstacles with a larger (artificial) bounding volume, e.g. by their convex hull [1].

Our work focuses on how to systematically cluster the real obstacles into fewer but larger artificial obstacles, which cannot cross (cut) the pre-specified curve, in a 2D environment. In other words, the less number of artificial obstacles shall result in the more efficient computation of collision detection and not affect the robot moving in its workspace. In [1], a heuristic algorithm having complexity $O(n^2m)$ was proposed, where n is the number of convex polygonal obstacles and m is the total number of vertices. The algorithm first determines an input sequence of obstacles to form clusters. Such sequence is intuitive, namely, to combine the closest pair of obstacles in turn if such combination doesn't cut the specified curve. As such, $\binom{n}{2}$ pairs of minimum Euclidean distances [3] between all obstacles are necessarily computed firstly. Then, these lengths of distances are necessarily sorted in decreasing sequence for clustering.

Although the criterion of minimum distance is first imposed in their algorithm, significance of this criterion was not explicitly explained in [1]. It should be noted that the geometric criterion in fact seems insignificant because the major concern is only that *the specified curve cannot be cut*. Therefore, we propose three viewpoints for tackling the problem, namely, 1) the number of clusters k should be as less as possible, 2) k is determined on the specified curve and independent of any criterion, and 3) assigning obstacles into a set of clusters that number has been estimated may depend on certain of criteria, such as minimum distance if similarity of obstacles' location in workspace is significant for each cluster.

In this paper, a more efficient and convergent recursive algorithm is proposed to tackle the problem of clustering convex polygonal obstacles in 2D robot's scene. The algorithm has *approximately* linear complexity of $O(nkm)$. Noted that the number of clusters k in general is greatly less than the number of obstacles n. And, in fact k is determined on the specified curve not on the density of obstacles (i.e., n). Namely, k is *almost* constant



(a) (b)

Figure 1, (a) A robot's scene is with eleven convex polygonal obstacles cluttered around a specified curve (trajectory of robot). (b) The clustering result via using the algorithm in [1], in which five clusters are obtained.

while the curve is fixed. The algorithm proceeds as follows. Firstly, the number of clusters k can be estimated after scanning all obstacles twice. Two procedures are proposed to identify the *seeds* of obstacles in clusters that determine existence of these clusters. Where, a cluster could be deleted if there are no seed within it. On the contrary, non-seeds in a cluster indicate that the obstacles can also be grouped into the other clusters, which are called *floating* obstacles. Secondly, procedures are developed to reassign the floating obstacles into clusters based on certain of criteria, such as minimum Euclidean distance. Finally, simulation results show that clustering of obstacles is very efficient in computational time and less number of clusters is obtained by using the proposed recursive algorithm.

2. DEFINITIONS OF THE PROBLEM

Suppose a 2-D robot's scene in which a Cartesian path (curve) has been specified for robot end-effector to follow and many obstacles modeled by convex polygons cluttered without crossing this curve. For instance, Fig.1a shows eleven convex polygonal obstacles cluttered around a specified curve. The goal of the proposed algorithm is to simplify the scene by clustering the set of convex polygons into fewer but larger artificial convex polygonal obstacles that cannot cross (or cut) the specified curve. For example, Fig.1b shows the clustering result via using the algorithm in [1], in which the eleven obstacles are reduced into five artificial obstacles (i.e. clusters). However, it is expected to obtain less number of clusters and do such clustering more efficiently.

Assume a set of convex polygonal obstacles $\mathbf{H}=\{P_i, i=1 \text{ to } n\}$ is to be grouped into clusters $\{C_j, j=1 \text{ to } k\}$. A cluster $C_j=\{P_a \mid P_a \in C_j\}$ is represented by an artificial convex hull [2] enclosing the set $\{P_a\}$. Define the notation $P_b \subset C_j$ to mean that P_b can be grouped with C_j. Namely, $P_b \subset C_j$ implies that the convex hull of P_b and C_j cannot cut the specified curve, otherwise it is denoted by $P_b \not\subset C_j$. Note that $P_a \in C_j$ implies $P_a \subset C_j$, on the contrary $P_b \subset C_j$ doesn't imply $P_b \in C_j$. To proceed, two definitions are used to describe the relationship between two clusters C_x and C_y (or two polygons P_a and P_b):

1) $C_x \subset C_y$ if and only if $P_a \subset P_b$, $\forall\ P_a \in C_x$ and $\forall\ P_b \in C_y$, otherwise, $C_x \not\subset C_y$,
2) $C_x \subset C_y$ ($C_x \not\subset C_y$) if and only if $C_y \subset C_x$ ($C_y \not\subset C_x$).

3.THE RECURSIVE ALGORITHM

The proposed algorithm consists of four procedures (Procedure I-IV) to group n polygons into k clusters.

3.1 ESTIMATION OF THE NUMBER OF CLUSTERS

In Procedure I, clusters are constructed, one by one, around the specified curve until all polygons have been scanned once. Initially, the number of clusters is set to be null, and then polygons are inputted by a random sequence. Let the first input polygon be within the *first* cluster. Then, the *next* cluster can be created when the current input polygon cannot be grouped into any of clusters created previously. Such procedure is written in the following pseudocode.

INITIALIZATION:
A Cartesian piecewise linear curve is specified;
Let $\mathbf{H}=\{P_i, i=1 \text{ to } n\}$ be the set of convex polygons;
Let C_j denote the cluster j;
Let k denote the number of clusters, and initially, $k \leftarrow 0$;

PROCEDURE I:
While H is not empty **do**
  Select and **remove** a P_i at random from the set **H**;
  Let flag $\leftarrow$0;
  For $j \leftarrow 1$ **to** k **do**
    Compute a convex hull CH to enclose P_i and C_j;
    If CH doesn't cut the specified curve **then**
      $C_j \leftarrow$ CH;
      Label $P_i \in C_j$;
      flag $\leftarrow$1;
      Break this for-loop;
    Endif
  Endfor
  /* *To create a new cluster* C_{k+1} *when* $P_i \not\subset C_j$, j=1 to k ***/**
  If flag = 0 **then**
    $k \leftarrow k+1$;
    $C_k \leftarrow P_i$;
  Endif
Endwhile

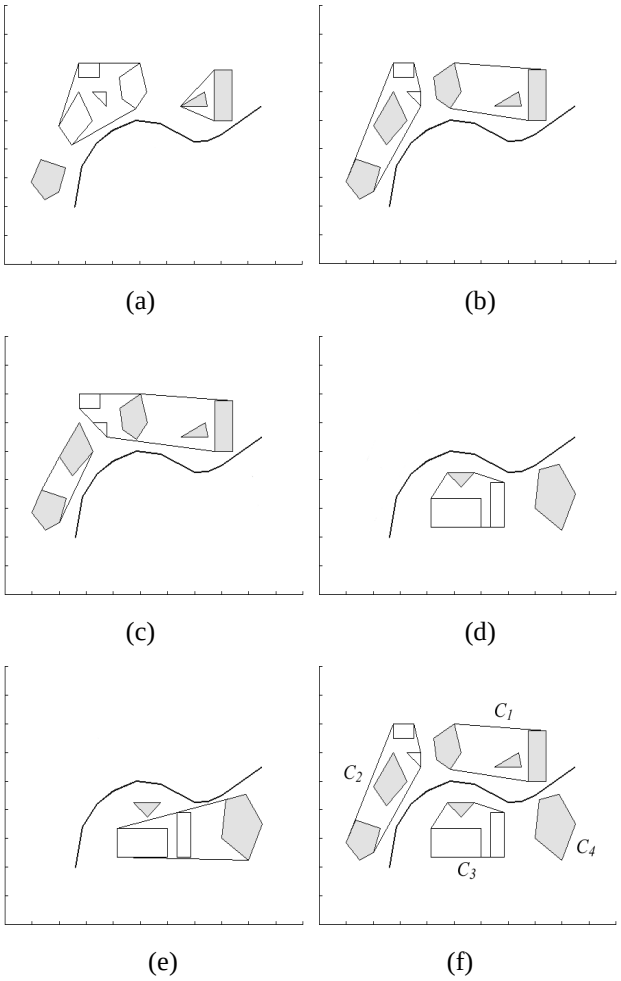


Figure 2, At most six sorts of possible clustering results can be obtained for the scene in Fig.1a, which can be composed of (a) to (e). For example, (f) shows the result composed of Fig.2b and Fig.2d. Seeds are shaded.

Procedure I can efficiently be used to obtain a set of initial clusters by that the specified curve cannot be cut. In fact, the output clusters by using the procedure would not be unique while different input sequences of polygons were used. For instance, at most six sorts of possible clusters can be obtained for the scene shown in Fig.1a. First, we separate this scene into two sets of polygons, one above and one below the specified curve. Fig.3a to Fig.3c and Fig.3d to Fig.3e show all possible clusters for the two sets of polygons, respectively. Although the output clusters might be different, such sets of clusters are usable at all because the specified curve cannot be cut still. Subsequently, in order to incorporate with certain of criteria and further reduce the number of clusters, Procedure II-IV in the following are proposed. Before that, two properties are shown for the clusters obtained via using Procedure I.

PROPERTY 1: $P_b \not\subset C_x$, where $P_b \in C_y$ and $x<y$.
Proof: By Procedure I, C_y is created due to $P_b \not\subset C_x$, $\forall$ $P_b \in C_y$ and $\forall$ $x<y$.

PROPERTY 2: $C_x \not\subset C_y$, $\forall$ $x \neq y$.
Proof: By Definition 2 and Property 1, $P_b \not\subset C_x$ implies $C_y \not\subset C_x$. Thus, we have $C_x \not\subset C_y$.

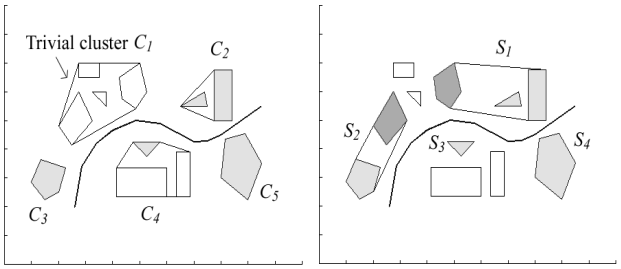
3.2 DELETION OF TRIVIAL CLUSTERS

Based on Property 1, Procedure II is proposed in order to examine whether $P_a \subset C_y$ or not, where $P_a \in C_x$ and $y>x$. After Procedure II is used, we can know which polygons have the alternative choice to be reassigned into other clusters. Such polygons are called *floating*. On the contrary, a polygon $P_i \in C_j$ is called a *seed* of C_j if P_i cannot be reassigned, i.e., $P_i \not\subset C_x$, $\forall$ $x \neq j$. For instance, the shaded polygonal obstacles in Fig.2 are seeds in clusters.

PROCEDURE II:
For $i \leftarrow 1$ **to** n **do**
  Let y indicate the index $P_i \in C_y$;
  For $j \leftarrow y+1$ **to** k **do**
    Compute a convex hull CH to enclose P_i and C_j;
    If CH doesn't intersect the specified curve **then**
      Record $P_i \subset C_j$;
    Endif
  Endfor
Endfor

Normally, a cluster C_j may include some seeds that can determine the existence of C_j. That is, at least the seeds would stay in C_j even if all floating $P_a \in C_j$ were removed from C_j and reassigned into the other clusters. However, a cluster may include no seed, which is called a *trivial cluster*. For instance, Fig.3a shows that a trivial cluster consists of (four) floating polygons. Any trivial cluster should be deleted because the number of clusters should be as less as possible. Assume a cluster C_t is trivial and $\{P_a,P_b\} \in C_t$. Therefore, there must exist a set of other clusters $\{C_x,C_y\}$ that $P_a \subset C_x$ and $P_b \subset C_y$, where $P_a \not\subset C_y$ and $P_b \not\subset C_x$, due to the fact of $C_t \not\subset C_x$ or C_y and Definition 1. In other words, the existence of a trivial cluster C_t is possible while $|C_t| \geq 2$ and $k \geq 3$ (i.e. at least C_t, C_x and C_y).

Deleting a trivial cluster C_t in theory involves two effects: 1) the cluster C_t disappears and 2) all $P_a \in C_t$ must be reassigned into the other clusters. Both of the effects may result in reduction of reassignment choice of the floating polygons $P_i \notin C_t$. Firstly, after a trivial cluster C_t is deleted, $P_i \subset C_x$ should be reexamined, where some $P_a \in C_t$ are removed into C_x. Secondly, the seeds within C_j, for $j=$



(a) (b)

Figure 3, (a) shows five clusters consisting of five seeds and a trivial cluster. (b) shows four embryo seeds S_j, j=1 to 4, and four floating obstacles after the trivial cluster is deleted. Darker shaded polygons are new seeds.

1 to k, should be updated because some floating polygons P_i may become seeds. Then, the *embryo* clusters S_j={all the *seeds* within C_j}, for j=1 to k, can be enclosed by their convex hulls before reassignment of floating polygons. For instance, after the trivial cluster in Fig.3a is deleted, Fig.3b shows four embryo clusters and four floating obstacles. In addition, any subset $\{P_a\}\in C_t$ can be totally assigned into another cluster C_x if $\forall P_a \subset C_x$. That is because the subset $\{P_a\}$ is originally a valid cluster by Definition 1. In the following, Procedure III is used to delete trivial clusters.

PROCEDURE III:
For $t \leftarrow 1$ **to** k **do**
  If there is no *seed-polygon* within C_t **then**
    Remove each $P_a \in C_t$ from C_t and **reassign** P_a into a cluster C_x, where $P_a \subset C_x$, via certain of criteria;
    Update whether $P_i \subset C_x$ still or not, for i=1 to n;
    Update seeds within C_j, j=1 to k;
  Endif
Endfor
Delete all empty clusters and **update** k;
For $j \leftarrow 1$ **to** k **do**
  Compute a convex hull S_j to enclose all the *seeds* in C_j;
Endfor

3.3 REASSIGNMENT OF FLOATING OBSTACLES

Finally, each floating P_i should be assigned to a certain S_j, which is achieved based on certain of criteria. For example, Fig.2f shows the final clustering result after the four floating polygons in Fig.3b are reassigned via using the geometric criterion of minimum Euclidean distance [3] between each P_i and a certain S_j.

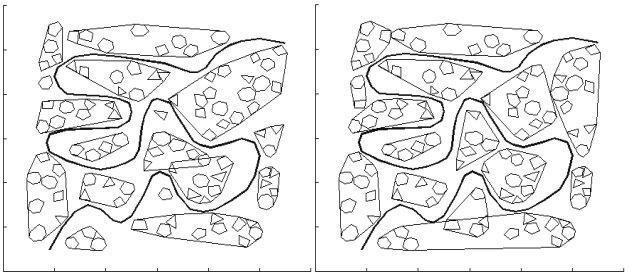
Although the case in Fig.3b is simple, in theory reassignment of the floating polygons may be more complicated. Assume a set of polygons G_j should be assigned to S_j, in which polygons may come from different clusters, i.e., G_j is not originally a valid cluster. In other words, the set G_j may be separated into sub-clusters D_s, s=1…h, by path of the specified curve. And, the set G_j may not be totally assigned to S_j although each polygon in G_j can singly be grouped with S_j. Reasoned as above, only one of the sub-clusters D_s of G_j can be assigned to S_j. In this case, the proposed algorithm can be *recursively* called to cluster polygons in G_j into D_s, s=1…h, *as less number h as possible*. Where, Procedure IV is proposed to achieve the recursion of the recursive algorithm (Procedure I-IV).

PROCEDURE IV:
Initialization
  Let ***F***={all of *floating* polygons};
  Define G_j be the set of *floating* polygons that should be grouped into S_j, j=1 to k;
While there exists any floating polygon within ***F*** **do**
  /* *To produce* G_j, j=1 to k. ***/**
  Let G_j be empty, j=1 to k;
  For each *floating* polygon P_i in **F** **do**
    Group P_i into a proper G_j via certain of criteria if $P_i \subset C_j$, i.e., $G_j \leftarrow P_i \cup G_j$;
  Endfor
  /* *To assign floating polygons in* G_j *into* S_j, j=1 to k. ***/**
  For each non-empty G_j **do**
    Call *the* ***recursive algorithm*** (Procedure I-IV) with **H**=G_j as the input, and then **output** sub-clusters D_s of G_j, s= 1 to h;
    Select a certain D_s via certain of criteria to be assigned to S_j, i.e., $S_j \leftarrow D_s \cup S_j$;
    Remove all $P_i \in D_s$ from ***F***;
    Set $P_i \not\subset C_j$ **if** $P_i \not\subset D_s$, for each floating P_i;
  Endfor
Endwhile

3.4 CONVERGENCE OF THE RECURSIVE ALGORITHM

In this subsection, we prove that the recursive algorithm can always be terminated in finite time. To begin with, Procedure I-II can be completed by twice scanning the set $\{P_i, i=1\ldots n\}$. Then, Procedure III can be finished in (k–2) times for the worst case because the procedure would be terminated at least while k<3. In other words, the convergence of the recursive algorithm is only determined on Procedure IV. We only need to prove that the *depth* of recursively calling the algorithm is finite and the floating obstacles would inevitably be grouped into one of clusters.

Firstly, Procedure IV can be finished while k=1 (cluster) because all obstacles are seeds in this case. Namely, the floating obstacles may exist only when $k \geq 2$.



(a) (b)

Figure 4, Clustering results: (a) 17 clusters via using the algorithm in [1] and (b) 15 clusters via using the proposed recursive algorithm for a randomly generated 100 convex polygons around a specified curve.

Assume k=2 clusters (i.e. two seeds must exist at least) and then the number of floating obstacles is the maximum, i.e., $n-2$. Reasoned as above, the maximum number of obstacles inputted to the recursively called algorithm is $n-2$. Therefore, the depth of recursively calling the algorithm is $\lfloor n/2 \rfloor$ times in worst case. Secondly, assume $P_a \in C_j$ and $P_b \in G_j$, and then any floating P_a can at least be grouped into S_j since $P_b \in G_j$ implies $P_b \subset C_j$ and $P_b \subset P_a$.

3.5 APPLIED TO DYNAMIC ENVIRONMENT

Our algorithm can easily be applied to dynamic environment in which obstacles may be inserted, deleted, or moved by virtue of these designed procedures and the definitions of seed and floating obstacles in clusters. If a set of new obstacles is inserted among the existing clusters, Procedure I would be applied to group them with clusters rapidly. Then, Procedure II and IV could be applied to determine whether some of them are seeds and reassign them, respectively.

On the other hand, a cluster may become trivial and should be deleted (that Procedure III would be used) while all seeds within it have been deleted or removed. Otherwise, a cluster remains a cluster with changing its artificial shape. Furthermore, obstacles may move within workspace, which is equivalent to delete then insert them.

3.6 COMPLEXITY OF THE ALGORITHM

To check if convex polygonal obstacles could be grouped with clusters, computation of the convex hulls needs O(m) times [2] in worst case. Here, computing intersection between the convex hull and the specified curve is assumed in constant time [1]. In Procedure I, grouping each polygon at most spends O(km) times of computing convex hulls. Therefore, O(nkm) computations are totally required. Similarly, O(nkm) times of detecting seeds in clusters should be spent in Procedure II. To proceed, in Procedure III the relationship between certain polygons and clusters should be updated in O(nkm) times after deleting a trivial cluster. Due to the facts that trivial clusters are indeed rare and k is almost constant, the complexity is O(nkm) in practice. Finally, reassignment of floating obstacles should also spend O(nkm) times. We conclude that the complexity of the recursive algorithm is O(nkm). In Section 4, simulation results verify the *approximately* linear time complexity of the recursive algorithm.

4. SIMULATION

In order to demonstrate the efficiency of the proposed algorithm, Fig.4a and Fig.4b show the simulation results of the algorithm in [1] with 17 clusters and of our algorithm with 15 clusters, respectively, for a randomly generated robot's scene with 100 convex polygonal obstacles around a prescribed curve. Note that clustering does not exclude the situation of overlapping among artificial obstacles. Such overlaps in fact are insignificant because the specified curve is still free of those artificial obstacles. The geometric criterion of minimum Euclidean distance between floating polygons and embryo clusters is used while reassignment of polygons (Procedure IV).

In addition, experiments are performed on the similar scenes randomly generated with various numbers n and m around the same curve in order to compare computational times. Fig.5 and Fig.6 show comparisons of computational times by fixing m/n=5(k=5~19), m=500(k=13~17), respectively. From Fig.5 and Fig.6, computational times using the algorithm in [1] grow exponentially, i.e., O(n^2m). On the contrary, complexity of the recursive algorithm is *approximately* linear O(nkm), as the example of fixing m=500 in Fig.6 shows. Normally, increasing the number of obstacles n might *slightly* increase the number of clusters k because additional obstacles may be randomly placed on where they can't be grouped into any of clusters and then necessarily create new clusters. The slightly increased number k results in a little rise in computational time of the recursive algorithm, for example, a little rise of the computational time occurs between 140 and 150 of n in Fig.5 or Fig.6 due to k is increased by one. In general, the number of clusters k affects the computational complexity very limitedly because its variation is very small.

On the other hand, because the load of computing convex hulls in Procedure IV is heavy, computational time of the recursive algorithm shall be larger than that of [1] when $m/n > 7$ as Fig.6. In such case, we suggest passing the use of Procedure IV in clustering, and then the computational time should be considerably saved.

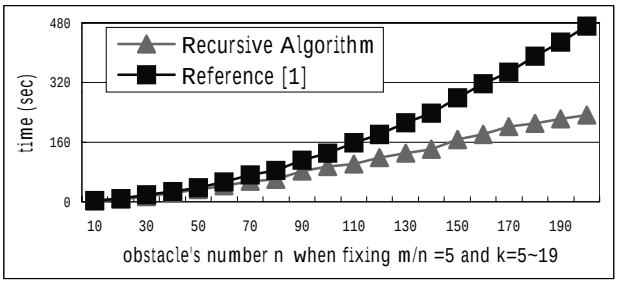


Figure 5, Comparison of computational time for varying *n* from 10 to 200 and fixing *m*/n=5 and *k*=5~19.

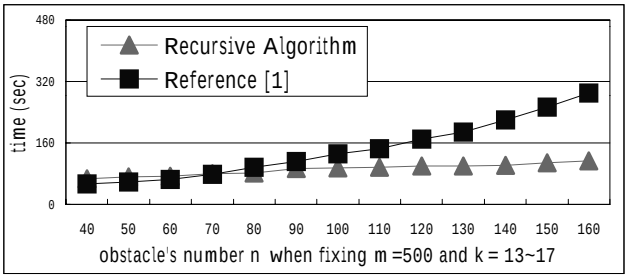


Figure 6, Comparison of computational time for varying *n* from 40 to 160 and fixing *m*=500 and *k*=13~17.

n	Average *k*	*n*	Average *k*	*n*	Average *k*
40	14.3/13.7	80	15.3/14.0	120	16.0/15.3
50	13.7/12.7	90	15.3/15.3	130	17.3/15.7
60	13.7/13.0	100	17.0/15.7	140	16.0/16.0
70	14.3/14.3	110	16.3/16.3	150	17.7/16.7

Table, the average *k* ([1] / the recursive algorithm) of three simulations with respect to *n* varied from 10 to 200.

Finally, the resulted numbers of clusters averaged from three simulations are compared in Table by the form ([1] / the recursive algorithm). The number of clusters by our algorithm is obviously less than that by [1], because trivial clusters can be effectively deleted in clustering by the use of Procedure III. Comparing the two algorithms, the proposed algorithm comes up with fewer clusters much more efficiently when the total numbers of obstacles and vertices are large.

5. CONCLUSIONS AND DISCUSSIONS

The recursive algorithm for efficiently clustering obstacles in the robot's scene into fewer artificial obstacles (clusters) has been proposed for reducing the complexity of robotic collision detection. By identifying *seeds* of clusters, the algorithm can further reduce the number of clusters *k* by deleting trivial clusters and incorporate with certain of criteria for reassigning non-seeds, for example, the smaller size of clusters or the larger free space for collision avoidance. Compared to the complexity $O(n^2m)$ of algorithm in [1], complexity of the recursive algorithm is approximately linear, i.e., $O(nkm)$. Note that *k* is greatly less than the number of obstacles *n* and almost constant while the specified Cartesian curve (path) is fixed.

Although the use of the recursive algorithm is restricted to 2-dimensional scene, Procedure I for estimating the number of clusters can be still applied to 3-dimensional scene. In order to extend the algorithm to 3-dimensional scene, our future work is to redesign Procedure III and IV.

REFERENCES

[1] A. C. Nearchou, N. A. Aspragathos, and D. P. Sofotassios, "Reducing the complexity of robot's scene for faster collision detection", *Journal of Intelligent and Robotic Systems*, vol.26, no.1, pp.79-89, 1999.

[2] J. V. Leeuwen, "The handbook of theoretical computer science", *Elsevier Science Publishers*, 1990.

[3] F. Chin, C. A. Wang, "Optimal algorithm for the intersection and the minimum distance problems between planar polygons", *IEEE Transaction On Computers*, vol.32, no.12, pp.1203-1207, 1983.

[4] J. C. Latombe, "Robot motion planning", *Kluwer Academic Publishers*, 1991.

[5] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles", *Comm. ACM*, vol.22, no.10, pp. 560-570, 1979.

[6] T. Lozano-Perez, "Spatial planning: A configuration space approach", *IEEE Trans. On Computers*, vol.32, no.2, pp.108-120, 1983.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", *International J. Robot. Res.*, vol.5, no.1, pp. 90-98, 1986.

[8] A. C. Nearchou and N. A. Aspragathos, "Collision-detection continuous path control of manipulators using genetic algorithm", *J. Systems Engrg.*, vol.6, pp.20-32, 1996.

[9] K. P. Valavanis, T. Hebert, R. Kolluru, and N. Tsourveloudis, "Mobile robot navigation in 2-D dynamic environments using electrostatic potential field", *IEEE Trans. On Syst. Man and Cyber.*, vol.30, no.2, pp.187-196, 2000.

[10] J. T. Schwartz and C. K. Yap, "Advances in robotics: Algorithmic and geometric aspects of robotics", *Lawrence Erlbaum Associates, Inc.*, vol.1, pp.95-143, 1987.