# A PROBABLISTIC APPROACH TO COLLISION-FREE ROBOT PATH PLANNING[1]

Sungtaeg Jun and Kang G. Shin

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

## ABSTRACT

One of the major problems with collision-free path planning for robots is the amount and complexity of computation required. The severity of this problem often leads to the design of unintelligent robots which simply follow pre-planned paths. The reason for this computational difficulty lies in that all obstacles in the workspace are treated uniformly regardless of their size and shape.

To remedy the above problem, we have developed a new method that takes into consideration the size and shape of the obstacles as well as the distance from the robot's current position to the obstacles. First, the workspace is divided into a finite number of cubes. An obstacle is represented by the set of cubes it occupies. Second, the probability of each cube becoming a deadend is calculated under the assumption that both the starting and destination points are randomly located in the workspace with a known distribution. Third, these probabilities are used to determine an optimal robot path that minimizes some cost associated with the path. Finally, several illustrative examples are presented. Our method is intended to be used for 3D problems and simple enough to be implemented for on-line robot path planning.

## 1. INTRODUCTION

The expectation of productivity increase and product quality improvement with industrial robots has been outgrowing the advances in robot technology. It is important to develop intelligent robots that can perform sophisticated tasks, rather than simply following precalculated trajectories. One of the major problems with the development of such intelligent robots is on-line obstacle detection and avoidance. There are several proposed algorithms for this problem, but few of them are acceptable in practice due mainly to their computational complexity.

The Configuration Space Approach (CSA) was used first in [1] and furthered in [2]. The CSA applies *growing* transformation on obstacles until the robot shrinks to a point. This transformation has simplified the search for collision-free regions considerably. The CSA only identifies safe regions in the workspace and does not generate any path for a given origin-destination pair. Lozano-Perez used the V-graph method to find a path [2]. The problem with the V-graph method is that it always generates a path very close to obstacles. One way to avoid this problem is to establish *guarded areas* in the vicinity of the obstacles. However, this solution presents another problem: use of large guarded areas may eliminate those paths that would be safe if smaller guarded areas were used. On the other hand, the use of small guarded areas would increase the danger of collision with obstacles.

O'Dunlaing and Yap [3] proposed an algorithm based on the Voronoi diagram. It usually finds the safest path by following the middle line between obstacles but has the following drawbacks. First, the safest path is not always the most desirable path; for example, it could be very long [4]. Second, the complexity of this algorithm grows rapidly as the number of obstacles in the workspace increases. The second drawback becomes more serious when this algorithm is to be used for 3D workspaces.

Khatib [5] proposed an approach using the *artificial field*. In this approach, the obstacles generate repelling forces while the destination generates attracting forces. The robot follows the line with the most attracting forces. The problem with this approach, however, is that search usually ends up with *local maxima*. Gilbert and Johnson [6] used the distance function to generate paths as well as trajectories. Though use of the distance function is quite elegant for off-line path planning, it takes too much time to be used for on-line robot path planning.

Kambhampati and Davis [7] proposed an algorithm using the $A^*$ search for mobile robot path planning. As a heuristic cost function, they used the sum of traveling distance and accumulated clearance from the nearest obstacle. During the course of search, they used the Euclidean distance between the destination and the current best node as the cost-to-go. Since this measure is a lower bound of the actual cost, the search is guaranteed to find the optimal solution for their heuristic cost function. However, the Euclidean distance estimator is the lower bound of traveling distance only and does not include the accumulated clearance. Therefore, almost every region has to be searched (i.e., an exhaustive search) until the destination is reached. To alleviate the need of an exhaustive search, they proposed to use a *pruned quadtree* that limits access to certain regions. This solution suffers from the same drawbacks found in the approach based on the guarded regions.

The computational complexity associated with conventional algorithms is due mainly to their inability to differentiate the various sizes, shapes, and orientations of obstacles. Figure 1 shows three pairs of cases which may occur during the search for a robot path. Conventional search methods using the *distance-to-go* and/or the *distance-from-obstacle* treat all these cases indiscriminately, although case (a) requires a special care to avoid the obstacle when compared with case (b). In fact, it is not easy to differentiate all possible cases. Instead of trying to classify all the

[1]The work reported in this paper was supported in part by the Airforce Office of Scientific Research under contract No. F33615-85-C-5105.

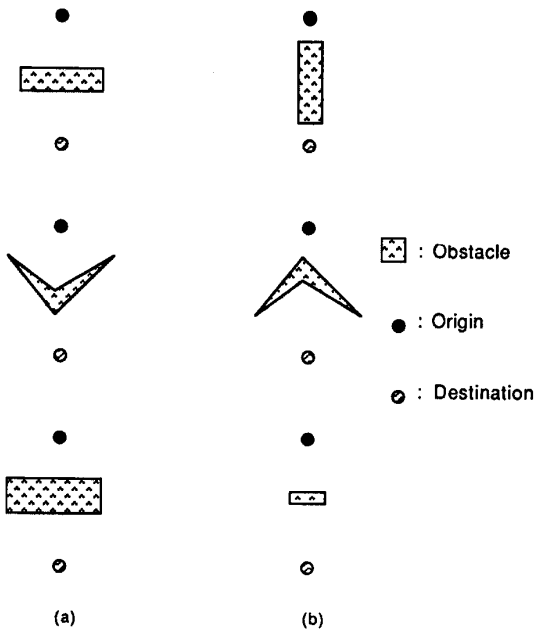



Figure 1. Various possible situations during the search.

factors that affect the search decision, we shall develop a single measure that subsumes all possible cases.

Almost every search algorithm uses the *expand–and–select* paradigm to reach a goal node from a starting node. When a node is expanded, the search tests whether it is a goal or not. If it is, the search will end, and if it is not, the search will choose another node from those nodes already generated and the procedure continues. Depending on the search strategy used, it may or may not choose one of the nodes generated most recently. When the next node chosen is not one of the nodes generated most recently, the most recent node expansion may become useless. In certain cases, the chosen node may not have any successor, thus reaching a *deadend*.[2]

The search will generate more nodes and, thus, wastes more time as the number of deadends encountered in the search increases. To reduce the search time, the number of deadends encountered during the search must be minimized. The measure used in our path planning algorithm is the probability of meeting a deadend. Clearly, the nodes in Fig. 1(a) have a higher probability of meeting deadends than those in Fig. 1(b). In general, the nodes in the vicinity of a larger obstacle have higher probabilities of meeting deadends than those near a smaller obstacle. (This holds for both convex and concave obstacles.) We shall present a search algorithm based on these probabilities to find an optimal robot path that minimizes the path cost.

The paper is organized as follows. Section 2 describes how to determine the probabilities of meeting deadends. Section 3 presents a search algorithm that utilizes these probabilities and an example workspace. In Section 4, our search algorithm is simulated for this example workspace and compared with the $A^*$ algorithm, and the paper concludes with Section 5. Although our algorithm is developed for robot path planning in 3D, only 2D examples are presented for the ease of understanding.

[2]A deadend is referred to as a node that does not have any children.

## 2. PROBABILITIES OF MEETING DEADENDS

It is assumed that there are a set of obstacles and a robot in the workspace. The goal of a robot path planner is to determine a curve or a set of points for the robot to follow from a starting point or an origin to a destination without colliding with any obstacle in the workspace. There are two sources of the difficulty associated with robot path planning: (i) an infinite number of paths exist for each given origin-destination pair, and (ii) it is in general difficult to represent an obstacle of arbitrary shape in the workspace One way of circumventing these sources of difficulty is to divide the workspace into a finite number of cells. Such a division not only reduces the infinite number of possible paths to a finite number of paths, but also allows an obstacle to be represented by the set of cells it occupies.

Following the above approach, let the workspace be divided into $m \times n \times l$ identical cubes, each of which is represented by its *center point*. As mentioned above, an obstacle in the workspace is then represented by the set of cubes (i.e., points) it occupies. Without loss of generality, we can assume that a robot is shrunk to a point by expanding obstacles. A robot moves towards the destination by advancing one point each time. The goal of the robot path planner is to find a sequence of neighboring points or cubes from the origin to the destination while minimizing certain cost associated with the path. Here, we define the cost of a path as the total number of points or cubes in the path, i.e., actual path length. It should be noted that the cost of a path will increase if the path runs sideways or backward instead of moving toward the destination. The main strategy of our algorithm is to minimize such encounters.

For a given destination, a point is said to be a *deadend* when (i) all of its neighbors are occupied by obstacles, or (ii) moving to any of its unoccupied neighbors increases[3] the path cost. A point is said to *meet a deadend* when its neighbor which is chosen to move to is a deadend, or its chosen neighbor meets a deadend.

Unlike conventional methods,[4] no prior knowledge on the origin-destination is assumed. Instead, each origin-destination pair will be given to the path planner at the beginning of its real execution. However, the probability distribution of locating the origin-destination pair is assumed to be given prior to the execution. This probability distribution can in general be arbitrary, i.e., it does not have to be any well-known distribution.

For the clarity of presentation, it is necessary to introduce the following symbols.

$V$: The set of all points or cubes in the workspace.

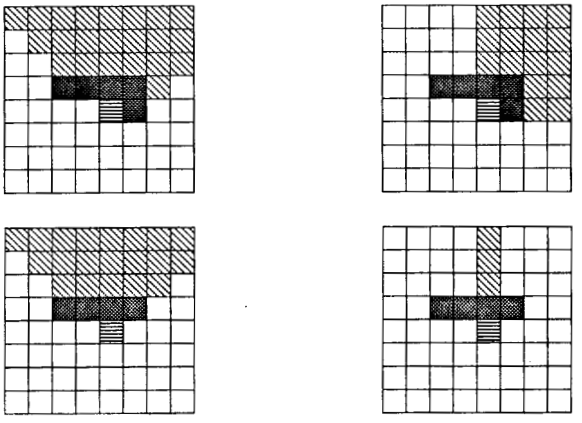
$v_{ijk}$: The $(i,j,k)$-th point in the workspace.

$N(v_{ijk})$: The set of points that are neighbors of $v_{ijk}$, i.e., the robot can move to one of these points from $v_{ijk}$ without passing through any other point in between.

$\hat{V}_{ijk}$: The set of destinations for which $v_{ijk}$ becomes a deadend.

[3]That is, taking sideways instead of moving towards the destination.

[4]They are concerned with the development of an algorithm for finding a path when the origin-destination pair is given. If a new pair is given, then the entire algorithm must repeat.



(a) Diagonal move allowed. (b) Diagonal move is not allowed.

Points that are not obstructed.

Current position.

Points that are occupied by obstacle.

Destinations that should be detoured from current position.

Figure 2. Examples of obtaining $\hat{V}_{ij}$.

$C_{ijk}$: The event that $v_{ijk}$ becomes a deadend.

$D_{ijk}$: The event that the search meets a deadend if $v_{ijk}$ belongs to the chosen path.

$R_{ijk}^{lmn}$: The event that $v_{ijk}$ precedes $v_{lmn}$ on a path. These two points are not necessarily neighbors to each other on the path.

$d_{ijk}$: The probability that $v_{ijk}$ becomes a destination.

$p_{ijk}$: The probability that $v_{ijk}$ becomes a deadend.

$q_{ijk}$: The probability that the search will meet a deadend later if the path passes through $v_{ijk}$.

Obviously, only the points that are adjacent to obstacles could be deadends. The average probability that a point $v_{ijk}$ becomes a deadend is $p_{ijk} = \sum_{v_{lmn} \in \hat{V}_{ijk}} d_{lmn}$. For example, if destinations are uniformly distributed throughout the workspace, then $d_{ijk} = \frac{1}{|V| - K}$ for all destinations in $\hat{V}_{ijk}$, where $K$ is the number of points that are occupied by the obstacles and $|A|$ is the cardinality of the set $A$. Then, the average probability of $v_{ijk}$ becoming a deadend is $p_{ijk} = \frac{|\hat{V}_{ijk}|}{|V| - K}$. Note that $\hat{V}_{ijk}$ depends not only on the definition of $N(v_{ijk})$, but also on the path cost. Generally, the neighbor of a point is defined as the set of all points that are horizontally, vertically, and diagonally adjacent to that point. The neighbor of a point can also be restricted to only those points that are vertically or horizontally adjacent, but not diagonally, to the point. With the two different definitions of neighbor, the examples of Figure 2 show the set of destinations that require a backtrack from the current position when the number of points in a path is used as the path cost.

The probability that a path containing $v_{ijk}$ meets a deadend can be calculated by $P[D_{ijk}] = P[\bigcup_{v_{lmn} \in V} (R_{ijk}^{lmn} \cap C_{lmn})]$. Since a robot path consists of a sequence of successive neighboring points, we get

$$P[D_{ijk}] = P[C_{ijk} \cup \bigcup_{v_{lmn} \in V} \{(\bigcup_{v_{opq} \in N(v_{ijk})} R_{ijk}^{opq} \cap R_{opq}^{lmn}) \cap C_{lmn}\}].$$

The event that a point becomes a deadend and the event that a point leads subsequently to a deadend are mutually exclusive. Thus, the above equation becomes:

$$\begin{aligned} P[D_{ijk}] &= P[C_{ijk}] + P[\bigcup_{v_{lmn} \in V} \{(\bigcup_{v_{opq} \in N(v_{ijk})} R_{ijk}^{opq} \cap R_{opq}^{lmn}) \cap C_{lmn}\}] \\ &= p_{ijk} + P[\bigcup_{v_{lmn} \in V} \{(\bigcup_{v_{opq} \in N(v_{ijk})} R_{ijk}^{opq} \cap R_{opq}^{lmn}) \cap C_{lmn}\}] \\ &= p_{ijk} + P[\bigcup_{v_{opq} \in N(v_{ijk})} \{(\bigcup_{v_{lmn} \in V} R_{ijk}^{opq} \cap R_{opq}^{lmn}) \cap C_{lmn}\}] \\ &= p_{ijk} + P[\bigcup_{v_{opq} \in N(v_{ijk})} \{R_{ijk}^{opq} \cap (\bigcup_{v_{lmn} \in V} (R_{opq}^{lmn} \cap C_{lmn}))\}] \\ &= p_{ijk} + P[\bigcup_{v_{opq} \in N(v_{ijk})} (R_{ijk}^{opq} \cap D_{opq})]. \end{aligned} \tag{2.1}$$

Since the search algorithm always chooses only one neighbor each time, the events $R_{ijk}^{opq}$'s for all $v_{opq} \in N(v_{ijk})$ are mutually exclusive. Thus, Eq. (2.1) becomes:

$$\begin{aligned} P[D_{ijk}] &= p_{ijk} + \sum_{v_{opq} \in N(v_{ijk})} P[R_{ijk}^{opq} \cap D_{opq}] \\ &= p_{ijk} + \sum_{v_{opq} \in N(v_{ijk})} P[R_{ijk}^{opq} \mid D_{opq}] P[D_{opq}]. \end{aligned} \tag{2.2}$$

It should be noted that $R_{ijk}^{opq}$ depends on the search strategy used. For example, in a blind search, $R_{ijk}^{opq}$ is independent of $D_{opq}$. When $R_{ijk}^{opq}$ is independent of $D_{opq}$, $R_{ijk}^{opq}$ can be obtained by the same method used to calculate $p_{ijk}$.

Most logical search strategies utilizing the probabilities of meeting deadends must choose a point that is least likely to lead to a deadend as long as such a choice does not increase the path cost. However, it is difficult to derive $P[R_{ijk}^{opq} \mid D_{opq}]$ in Eq. (2.2) because $D_{opq}$ is unknown and depends on $R_{ijk}^{opq}$. Moreover, the path cost depends on the probabilities of meeting deadends. To overcome these difficulties, it is necessary to introduce a new event, denoted by $D_{ijk}^n$, that $v_{ijk}$ will meet a deadend in $n$ steps. Then, Eq. (2.2) can be rewritten as:

$$P[D_{ijk}^{n+1}] = \begin{cases} p_{ijk} & \text{if } n = 1 \\ p_{ijk} + \sum_{v_{opq} \in N(v_{ijk})} P[R_{ijk}^{opq} \mid D_{opq}^n] P[D_{opq}^n] & \text{if } n \geq 2. \end{cases} \tag{2.3}$$

By using mathematical induction and Eq. (2.3), $P[D_{ijk}^n]$ can be calculated for any $n \geq 2$. Since $D_{ijk} = \lim_{n \to \infty} D_{ijk}^n$, we can derive $P[D_{ijk}] = \lim_{n \to \infty} P[D_{ijk}^n]$ by applying Eq. (2.3) recursively.

In what follows, we shall develop a search algorithm to determine an optimal robot path between an arbitrary pair of points based on the probability of each point being a deadend in the workspace.

## 3. THE ROBOT PATH PLANNER

The robot path planner consists of two phases. In the first phase, the path planner transforms the workspace information into the probabilities of meeting deadends which are computed off-line as discussed in the previous section. 2D examples of the output of the first phase using the number of points in a path as the path cost are shown in Fig. 3. In these examples, the workspace consists of 32×32 points and the neighbor of a point in the workspace is defined as the set of all points that are horizontally, vertically, and diagonally adjacent to the point. Since robot motions are usually performed through rotary joints, diagonal movements are more natural than strict horizontal and/or vertical movements in many cases. The destinations used in these examples are assumed to be distributed uniformly[5] over the entire workspace. 108 points in Fig. 3a are occupied by the obstacles while 144 points in Fig. 3b are occupied by the obstacles, i.e., more obstacles in Fig. 3b than in Fig. 3a. The obstacle data is shown on the left and $P[D_{ijk}]$ at the right of the figures.

Upon receiving the workspace information, the first phase of the path planner transforms the obstacle data into the probabilities of meeting deadends using the procedure **P1** below.

**Procedure P1**

1. Compute $p_{ijk}$ for all $v_{ijk} \in V$ using $p_{ijk} = \dfrac{|\hat{V}_{ijk}|}{|V| - K}$.
2. Initialize $P^*[D_{ijk}] := p_{ijk}$, $error := 0.001$, and $max\_error := \infty$.
3. Repeat 3a-c until $max\_error < error$.
   - 3a. For all $v_{ijk} \in V$, compute $P^*[R_{ijk}^{opq} \mid D_{opq}]$ using $P^*[D_{opq}]$, and $P[D_{ijk}] := p_{ijk} + \sum_{v_{opq} \in N(v_{ijk})} P^*[R_{ijk}^{opq} \mid D_{opq}] P^*[D_{opq}]$.
   - 3b. Set $max\_error := \min(max\_error, \max_{v_{ijk} \in V}(|P(D_{ijk}) - P^*[D_{ijk}]|))$.
   - 3c. Set $P^*[D_{ijk}] := P[D_{ijk}]$.

The second phase of the path planner converts an origin-destination pair to a collision-free optimal path. The hill-climbing method is chosen as our search strategy. The main advantage of the hill-climbing method is that it may converge to a good solution very quickly. However, it may waste a great deal of time when a successor that leads to a local maximum or a deadend is chosen. Since our algorithm is designed to choose a successor that is least likely to lead to a deadend, this disadvantage of the hill climbing method is minimized.

Although our method chooses a point that will least likely lead to a deadend, it may still lead to a deadend. Furthermore, there are some origin-destination pairs that will always lead to deadends due to a large obstacle between them. One way of dealing with this problem is to backtrack to an earlier point and choose some other direction from there on. Decisions to be made in the backtracking are: the number of points to backtrack and the existence of a path without taking sideways. The former is important because a smaller number of backtrack steps will often lead to the same deadend while a larger number of backtrack steps will waste search time and may also lead to another deadend that the original path has already avoided. The latter presents a more critical problem than the former. The existence of a path can be verified only after all combinations of points are tested. According to the simulation results obtained thus far, our method is shown to avoid deadends very well when there exists a path connecting the origin and the destination without taking sideways. That is, if the first attempt to find a path for a given origin-destination pair meets a deadend, the second attempt usually ends up with going to a deadend again.

To remedy the above situation, we adopted the following backtracking policy. For a partially constructed path $P_k = v_0 v_1 \cdots v_k$, where $v_0$ is the origin, $v_k$ the point that becomes a deadend, and $q_i$ is the probability of $v_i$ leading to a deadend.[6]

Step 1: Find the first $v_i'$ such that $v_i'$ is a sibling of $v_i$ and $q_i' < q_{i-1}$, for $i = k-1, k-2, \cdots, 1$. Set the path to $P_i := v_0 v_1 \cdots v_{i-1} v_i'$.

Step 2: If no such point is found, set the path $P_i := v_0 v_1 \cdots v_i v_i^* \hat{v}$ where $v_i^*$ and $\hat{v}$ are the first pair of points such that $\hat{v}$ is a child of $v_i^*$ and $\dfrac{1 - q_i^*}{\hat{q} - q_i^*}$ is less than the projected distance[7] between the destination and $v_i^*$ for $i = k-1, k-2, \cdots, 0$.

It should be noted that path cost will increase when the backtrack point is not found in Step 1.

Another key issue associated with the efficiency of any search method is the search direction. Backward search is usually more efficient than forward search when there are more initial states (i.e., origins) than final states (i.e., goals). Note that this usual selection of search direction is not directly applicable to robot path planning, since one and only one origin-destination pair needs to be considered each time in robot path planning. However, a simple modification in accordance with the following observation will make the usual selection applicable to robot path planning: there are more ways to reach a point when the probability of the point meeting a deadend is small than when this probability is large. In other words, backward search is more efficient than forward search when the destination has a higher probability of meeting a deadend than the origin. Our experiments show that on the average, 30% of deadends can be avoided by exchanging[8] the origin and the destination based on their probabilities of meeting deadends.

The procedure **P2** described below constructs two paths each time: FPATH starting from the origin and BPATH from the destination. Out of these two paths, a path which is less probable to meet a deadend will be expanded first. The search will continue to reduce the differences between the two paths, FPATH and BPATH, until the two paths intersect.

**Procedure P2**

1. Initialize the origin with FORWARD and the destination with BACKWARD. Set FPATH and BPATH to null sets.
2. Repeat 2a-d until $FPATH \cap BPATH \neq \emptyset$:
   - 2a. Select a node which has a lower probability of meeting a deadend from FORWARD or BACKWARD. Call that node CURRENT.

[5] As mentioned earlier, any other distributions can be assumed.

[6] For notational convenience, we used a single, instead of triple, subscript to represent a point or cube in the workspace.

[7] That is, the minimum number of points between $v_i^*$ and the destination in the absence of obstacles.

[8] Switching between backward search and forward search.

2b. Insert CURRENT to the corresponding set, FPATH or BPATH.

2c. Choose a successor of CURRENT from its neighbor. A node which minimizes the path cost is selected as the successor. If there is more than one node with the minimal path cost, a node which has the lowest probability of meeting a deadend is chosen as the successor. If there is no successor for CURRENT, backtrack to an earlier node.

2d. Call the successor FORWARD or BACKWARD, depending on its predecessor.

## 4. RESULTS

The workspace model used in our simulation consists of 32×32 points. The simulation is performed using the obstacle data in Fig. 3a and Fig. 3b. Origin-destination pairs are generated by a uniform random number generator. The entire simulation is performed on the Sun 3/50 running a FranzLISP interpreter. Hence, the processing time shown in Table 1 corresponds to the CPU time of the Sun 3/50.

We also generated paths using the $A^*$ algorithm for the purpose of comparison. The heuristic used in the $A^*$ algorithm is $f(p) = g(p) + h(p)$, where $g(p)$ is the number of points from the origin to the node $p$ and $h(p)$ represents the estimated cost from $p$ to the destination. The estimated cost from $p$ to the destination $q$ is obtained as $h(p) = D(p, q) - C(p)$, where $D(p, q)$ represents the minimal number of points between $p$ and $q$ and $C(p)$ is the minimum clearance between obstacles and $p$. The reasons we have chosen this heuristic estimator are: (i) $D(p, q)$ is the lower bound of the actual path cost, and (ii) the $A^*$ search will choose a point that is not likely to be wasted, i.e., a point that is far from obstacles. Since $h(p)$ is an under-estimate of the actual cost, the $A^*$ algorithm guarantees the optimal solution that maximizes the clearance.

To ensure the parity between the two simulations, the random number generator produces exactly the same sequence of origin-destination pairs for both our method and the $A^*$ algorithm. The simulation results for both methods are tabulated in Table 1. The paths generated by our method require slightly higher costs ($\simeq$ 1%) than those generated by the $A^*$ algorithm. However, the total processing time taken heavily favors our method over the $A^*$ algorithm. Furthermore, the average time taken to find each path in our method is small enough to be used for on-line path planning.

| Workspace | Fig. 3a | | Fig. 3b | |
|---|---|---|---|---|
| Algorithm | A1 | A2 | A1 | A2 |
| Avg. path length | 15.97 | 15.81 | 17.14 | 16.37 |
| Avg. # of node expansions | 17.27 | 57.45 | 20.39 | 78.47 |
| Avg. # of backtracking | 0.21 | NA | 3.21 | NA |
| Avg. CPU time | 4.78 | 53.86 | 6.92 | 86.73 |

A1 : Simulation using our algorithm.
A2 : Simulation using the $A^*$ algorithm.
(Each data is based on 100 origin-destination pairs.)

Table 1. Simulation results using the example in Figs. 4a and 4b.

## 5. CONCLUSION

In this paper, we have developed an objective measure of describing the effects of obstacles on collision-free robot path planning. This measure is then used for a search method similar to the hill climbing method. The hill climbing method generates solutions very fast if it does not encounter deadends. Although it is not possible to avoid deadends completely during the search, we can minimize the probability of encountering deadends based on the measure developed here. The quality of the path obtained from our method was shown to be comparable to that from the $A^*$ algorithm. More importantly, the computational cost is reduced substantially (to less than 1/10) when compared with the $A^*$ search.

The main advantage of our algorithm will become more vivid if it is implemented for a 3D environment. The complexity of conventional algorithms increases dramatically when it is to be implemented for 3D problems. For example, the $A^*$ algorithm will suffer greatly from the computational complexity when every node generates 26 successors in case of 3D, rather than 8 successors in case of 2D. Since more number of successors imply more number of ways of avoiding deadends, our algorithm will work far better when there are more successors to choose.

## REFERENCES


[1] S. M. Udupa, "Collision Detection and Avoidance in Computer Controller Manipulators," *Proc. 5th Int'l Joint Conf. Artificial Intelligence*, 1977.

[2] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," *IEEE Trans. Comp.*, vol. C-32, no. 2, pp. 108-119, Feburary 1983.

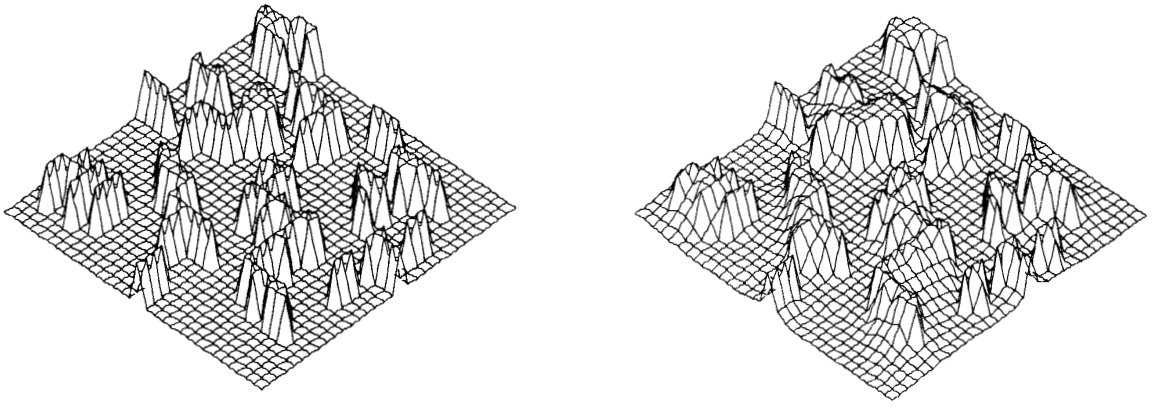
[3] C. O'Dunlaing and C. K. Yap, "The Voronoi Diagram method of motion-planning: I. The case of a disc.," *J. Algorithm*, vol. 6, pp. 104-111, 1985.

[4] S. - H. Suh and K. G. Shin, "Robot Path Planning with a Weighted Distance-Safety," *Proc. 26-th Conf. on Decision and Control*, pp. 634-641, December 1987 (An improved version will also appear in *IEEE J. Robotics and Automation*).

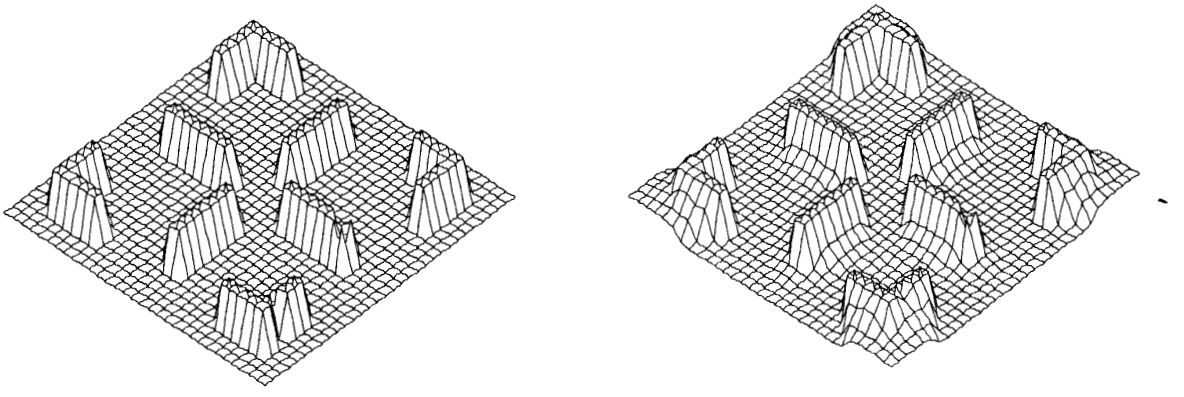
[5] O. Khatib, "Real-Time Obstacle Avoidance For Manipulators and Mobile Robots," *IEEE Int. Conf. on Robotics and Automation*, pp. 500-505, March 1985.

[6] E. G. Gilbert and D. W. Johnson, "Distance Functions: Their Application to Robot Path Planning in the Presence of Obstacles," *IEEE Journal Robot. Automat.*, vol. RA-1, pp. 21-30, March 1985.

[7] S. Kambhampati and L. S. Davis, "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 135-145, sept. 1986.



(a)



(b)

Fig 3. Maps of sample workspaces and probabilities of becoming and meeting deadends