# Optimal Path Planning for Mobile Robot Navigation

**Article** *in* IEEE/ASME Transactions on Mechatronics · September 2008

DOI: 10.1109/TMECH.2008.2000822 · Source: IEEE Xplore

CITATIONS
112

READS
1,520

**3 authors:**

Gene Eu Jan
National Taipei University
**107** PUBLICATIONS **696** CITATIONS

SEE PROFILE

Ki Yin Chang
National Taiwan Ocean University
**21** PUBLICATIONS **414** CITATIONS

SEE PROFILE

Ian Parberry
University of North Texas
**99** PUBLICATIONS **1,980** CITATIONS

SEE PROFILE

# Optimal Path Planning for Mobile Robot Navigation

Gene Eu Jan, Ki Yin Chang, and Ian Parberry

***Abstract*—Some optimal path planning algorithms for navigating mobile rectangular robot among obstacles and weighted regions are presented. The approach is based on a higher geometry maze routing algorithm. Starting from a top view of a workspace with obstacles, the so-called free workspace is first obtained by virtually expanding the obstacles in the image. After that, an 8-geomerty maze routing algorithm is applied to obtain an optimal collision-free path with linear time and space complexities. The proposed methods cannot only search an optimal path among various terrains but also find an optimal path for the 2-D piano mover's problem with 3 DOF. Furthermore, the algorithm can be easily extended to the dynamic collision avoidance problem among multiple autonomous robots or path planning in the 3-D space.**

***Index Terms*—$\lambda$-*geometry* maze router, path planning, piano mover's problem.**

## I. Introduction

THE MOTION planning problem is to determine a path of a mobile robot from its source (initial) position to a destination (goal) position through a workspace populated with obstacles. The obstacles may be either stationary or moving. The desired path is an optimal one with no collisions between the robot and the obstacles. If the robot has constant speed and instantaneous acceleration, the planning problem is reduced to finding an optimal collision-free path. Let **M** represent a moving rigid robot, the path planning problem for **M** can be formulated as follows. Given the geometric descriptions of a moving robot and obstacles in a bounded region, a source (expressed by $S$) and a destination (expressed by $D$) of the moving robot **M** determine that a continuous collision-free path $LL_{\rm path}$ from $S$ to $D$ exists or not, and if so, plan such a path [1]. But, there are many different aspects of the path planning problem for a mobile robot including optimal path planning among rectangular obstacles, optimal path finding among weighted regions, path planning to traverse narrow passages (also called piano mover's problem), etc. This paper proposes a higher geometry maze routing algorithm that has fewer constraints and can be applied to the previous aspects in the path planning problem.

Most of the methods for solving the path planning problem based on the environment map for mobile robots are described by Latombe [2], such as road map, cell decomposition, and potential field methods. Additionally, several recent approaches have been proposed to solve this problem. For example, Jiang *et al.* [3] used a visibility graph without generating a complex configuration space to find a shortest collision-free path for a point. The point is evaluated to find whether it can be used as a reference to build up a feasible path for the mobile robot. Szczerba *et al.* [4] computed a shortest path between two distinct locations through a 2-D or 3-D environment consisting of weighted regions. Their cell decomposition approach was based on new frame subspace data structures. Hu *et al.* [5] applied an efficient network flow formulation to optimally solve the robust path planning problem. Laumond *et al.* [6] presented an efficient motion planner for a car-like robot based upon recursive subdivision of a collision-free path. Furthermore, Barraquand *et al.* [7] used bitmap arrays to construct potential field numerically for robot path planning. They are in the class of artificial potential field methods. There are also some other concepts based on exploration of robot distance maps [8], transformation of dynamic digital distance maps [9], or extensions of the popular A$^*$ search [10], [11]. Recently, Wu and Zhou [12] found the shortest time routes, while both deadlock and blocking in automated guided vehicles (ACVs) systems are avoided. It first found the shortest routes based on a guide path, and then, performed rerouting, whenever necessary, to avoid deadlock and blocking, according to a deadlock avoidance [13]. Yuan and Yang [14] proposed a novel approach of automated multi-robot nanoassembly planning also. Their approach applied an improved self-organizing map to coordinate assembly tasks of nanorobots while generating optimized motion paths at run time with a modified shunting neural network. In this paper, the proposed algorithms use cell map with cost function and higher geometry wave propagation to search the optimal path for a known environment (an image covers whole or part of environment). Among all path planning algorithms, two approaches [15], [16] have been proposed to solve the robot path planning problems among arbitrary shape obstacles in the image plane that have the same scheme as our proposed method. Their main ideas are derived from mathematical morphology or topology. Our algorithm simplifies the problem into a set of geometric systems. Thus, three advantages are gained. First, the algorithm is able to search the optimal path among weighted regions of various terrains in the same workspace. Second, our algorithm can intelligently rotate a mobile robot to pass through narrow passages by the optimal path planning algorithm with linear time complexity, which their approaches fail to accomplish. Furthermore, the algorithm can be easily extended to multiple robots in one workspace since the cost function, the arriving time, of each cell in the desired path has been determined. The collision detection and avoidance for these robots can be achieved based on the simulation of multiple robots with their configuration domains

Manuscript received October 31, 2005; revised April 3, 2008. Published August 13, 2008 (projected). Recommended by Technical Editor M. Meng.

G. E. Jan is with the Institute of Electrical Engineering, National Taipei University, Taipei 104, Taiwan, R.O.C. (e-mail: gejan@mail.ntpu.edu.tw).

K. Y. Chang is with the Department of Merchant Marine, National Taiwan Ocean University, Keelung 20224, Taiwan, R.O.C. (e-mail: b0170@mail.ntou.edu.tw).

I. Parberry is with the Department of Computer Science, University of North Texas, Denton, TX 76203-6886 USA (e-mail: ian@cs.unt.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMECH.2008.2000822

and space marking of the collision area [17]. Thus, dynamic collision avoidance for multiple autonomous robots is feasible with no extra work.

Besides the robot motion, there are some other applications of path searching algorithms, e.g., the routing of a very large scale integration (VLSI) design problem [18], the path searching in electronic chart display information systems (ECDIS) [19], and the road map routing problem [20]. Most of these algorithms are based on Lee's routing algorithm [21] and its variations [22] in the grid plane. The popularity of Lee's algorithm lies in its simplicity, easy implementation, and the guarantee to find a shortest path if one exists. The procedure of Lee's algorithm can be visualized as a *wave propagation* process. Two lists $L$ and $L_1$ are defined to keep track of the cells on the wavefront (also called frontier cells) and their neighboring cells, respectively. Inserting the source cell into list $L$ initializes the search. After all the neighboring cells in $L$ are included in $L_1$, the list $L_1$ is processed such that an expanded wavefront is found. Then, any cell in $L$ is deleted if its entire neighboring cells have been processed (updated their arrival time values) and $L$ is updated by this new wavefront. The search is terminated if the destination cell is reached.

In the early years, Lee's algorithm was explored to route the fractal robot in the grid space as well as to plan their motion paths. But the algorithm allows only Manhattan paths, thus its applications are limited. This paper proposes a new pixel-based approach that improves Lee's rectilinear (four-directional) routing to higher geometry routing and the result has the same time and space complexities of $O(N)$ as Lee's algorithm, where $N$ is the number of pixels (cells) in the workspace. Our algorithm solves this problem by using suitable data structures to perform uniform wave propagation, the correctness of which has been proven [23]. Thus, our maze routing algorithm works in 4-geometry, 8-geometry, or higher geometry and benefits Lee's two advantages that are obstacle independency and the guarantee to obtain the optimal path if one exists. Compared with the 4-geometry [9] and the 8-geometry [8] searching algorithms, our maze routing algorithm significantly improves the time complexity to $O(N)$ and further extends the work to weighted regions and piano mover's problems with linear time and space complexities. Several similar uniform wave propagation methods arise in the field of pattern recognition [24] and computer-aided design [25]. However, these methods are not suitable for applying in the image plane. The application of our algorithm is naturally favorable for a pixel-based plane such as the raster plane since the algorithm is developed in a grid plane. The algorithm is similar to the breadth-first search algorithm until the time of arrival of the planar cells is determined or the given condition is reached. The computation of the optimal path finding is not as extensive as the graph scheme since the problem is independent of the shapes of the obstacles and no preprocessing effort is required to construct a suitable search structure. Therefore, the proposed optimal path planning algorithms generate collision-free paths among weighted regions based on the 8-*geometry* maze routing algorithm that is applied to virtual obstacles expansion method as well.

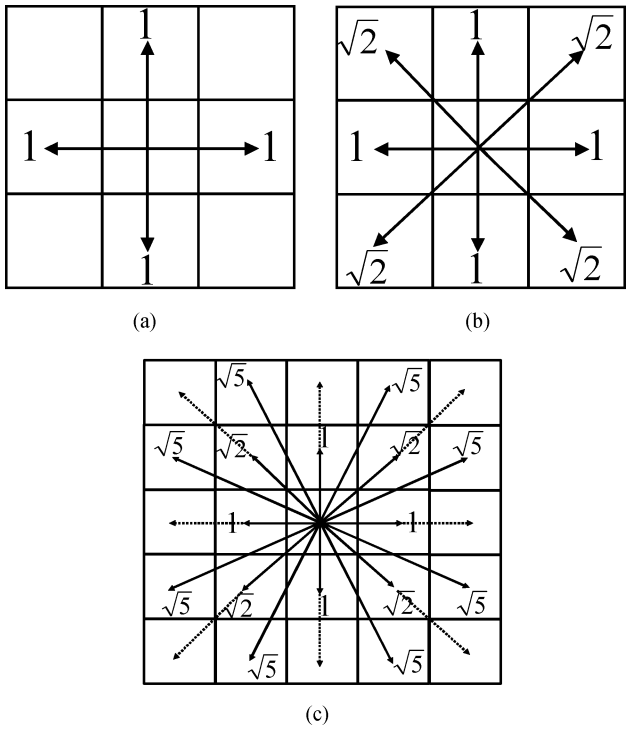



Fig. 1. Cell connection styles [26]. (a) 2-*geometry* neighborhood. (b) 4-*geometry* neighborhood. (c) 8-*geometry* neighborhood.

## II. Higher Geometry Maze Routing Algorithm

### A. λ-Geometry

For an $m \times n$ rectangular grid of $N$ cells, a *cell map* has a finite set of values and indicates which cells constitute obstacles at the very least. Each cell represents a pixel in the raster plane. Thus, there should not be any significant distortion between the original geometry map and the raster chart because the size and the shape of the cells are exactly the same as the pixels. The λ-*geometry* allows edges with the angles of $i\pi/\lambda$, for all $i$, $\lambda = 2,\ 4,\ 8$ and $\infty$ corresponding to rectilinear, 45°, 22.5°, and Euclidean geometries, respectively [26]. For a 2-*geometry* neighborhood, we are only interested in the cells above, below, to the left, and to the right of a cell, i.e., all of the cells that are distance 1 unit from the center cell, as shown in Fig. 1(a). For a 4-*geometry* neighborhood, each cell has eight neighbors that have distances of 1 or$\sqrt{2}$ units from the center cell; thus, movements along the diagonal directions are possible, as shown in Fig. 1(b). Note that Lee's algorithm can also be applied to the situation where every cell has eight neighbors. But, in order to work properly for Lee's algorithm, all eight neighboring cells must be equidistant from the center cell.

In the 8-*geometry* neighborhood, each cell has 24 related neighbors. The distance of the 24 cell-connected neighborhoods to the center cell is shown in Fig. 1(c). There are 16 solid-line reachable neighbors required to compute for each move. However, the eight dashed-line reachable neighbors do not need to be computed since they can be extended by the next computation and will be computed in the next move. Note that each angle in the 8-geometry is not exactly 22.5°, half of 45°, because

the angle is divided in the grid plane, not in a circular plane. Comparing the 4-*geometry* neighborhood with the 8-*geometry* neighborhood, the latter has twice the selective directions as the 4-*geometry* neighborhood and the obtained path would be smoother. For the 16-*geometry* neighborhood, the running time for computation and condition statements is about twice that of the 8-*geometry*. However, the 16-*geometry* routing has twice the selective directions for searching an optimal path as that of the 8-*geometry*. Thus, for the 32-*geometry* and other higher geometry maze routers, it is merely required to increase the number of circular buckets and conditional statements. The running time is $O(\lambda N)$ for the $\lambda$-*geometry* algorithm. In this paper, the proposed methods are based on the 8-*geometry* maze router to reduce the required computation. However, the proposed optimal path planning works in the $\lambda$-*geometry*.

Lee's algorithm fails for higher geometry if different distances occur. A straightforward attempt to improve Lee's algorithm by using equal cost wavefronts causes a substantial increase in time complexity [18], [19]. However, our algorithm works in a general context and uses a different data structure from that of Lee's algorithm. But in the case of 2-geometry, the two algorithms are the same.

### B. Required Data Structures and the 8-Geometry Maze Router

The required data structures for the proposed method are mainly a cell map, buckets, and linked lists.

1) *Cell Map*: In the formatting of the cell structure of a raster plane, the number of data fields is flexible to meet the requirements of the problem. In an $m \times n$ grid plane, any cell $C_{i,j}$ has four parameters $F/O_{i,j}, AT_{i,j}, Vis_{i,j}$, and $Dir_{i,j}$ where $0 \le i \le m-1,\ 0 \le j \le n-1$. The first parameter *F/O* (*free workspace* or *obstacle*) distinguishes whether a cell is in the obstacle regions, the value is infinity, or in the free workspace where the value is 1 for flat, smooth surface. If the parameter *F/O* of any cell has a finite value between 1 and infinity, we are working on the optimal path among weighted regions. The second parameter $AT$ (time of arrival) stores the time needed to travel from the source cell to the current cell and its initial value is infinity. The third parameter *Vis* (*Visited*) distinguishes whether the cell has been visited or not and its initial Boolean value is *false*. The fourth parameter *Dir* keeps track of which direction the previous cell moves to and causes the minimum $AT$ value. There are total 16 different directions for each single move and its initial value is 0000. The cells' initial values are illustrated in Fig. 2, where the white, black, and gray cells represent free workspace, obstacles, and weighted regions, respectively.
2) *Buckets and Linked Lists*: The buckets $\beta$ consist of a sequence of initially empty queues with continuous integers in the range 0 to $AT_{\max}$, where $AT_{\max}$ is the maximum $AT$ value and obviously less than $N$. Each queue is called a bucket $LL_l$ that stores a series of cell indices, where $l$ is both the index and the header value of bucket $LL_l$ and $l \le AT_{i,j} < l+1$ for the $AT_{i,j}$ values of the cells in the bucket $LL_l$. For an $m \times n$ grid of cells, the first step in the algorithm is to input a source cell and a destination cell. According to the algorithm, the $AT$ value of the source cell is 0 and the indices of the source cell $S$ will be inserted into the first bucket $LL_0$. The remaining cells that spread from $S$ will be inserted into their corresponding buckets according to their $AT$ values. For a single terrain problem, the $AT$ value of any cell would be increased at most by $\sqrt{5}$ from its neighboring cell. The number of decimal digits in the irrational numbers $\sqrt{2}$ and $\sqrt{5}$ is determined by the total number of cells on the grid plane. Therefore, the number of buckets can be reduced to four ($LL_0, LL_1, LL_2$, and $LL_3$) for the purpose of recycling since the updated cells would be inserted into one of the next three buckets. In addition, a temporary list $TL$ is applied to the algorithm to store the indices $(i, j)$ of visited cells until all of the cells in the current bucket have updated their neighboring cells. This data structure enables us to perform uniform propagation for arbitrary cost functions.

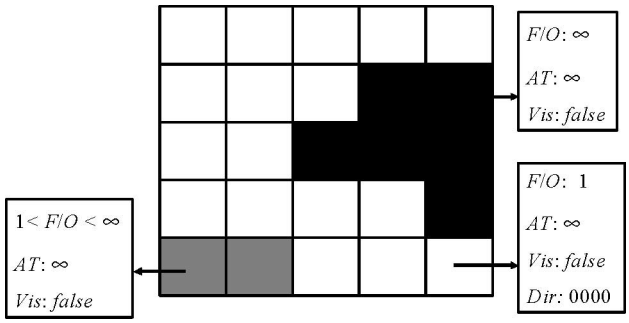



Fig. 2. Cell map.

As a whole, there are mainly three different aspects between our higher geometry algorithm and the other 2-*geometry* algorithms. 1) Buckets are introduced to process the propagation of several nonequal (single-step) cost functions to a uniform propagation without a series of sorting process. 2) The *Vis* parameter is introduced to make sure that each cell is inserted into the $TL$ exactly once. Due to these two aspects, the algorithm has a linear time complexity. 3) The parameter *F/O* is introduced for weighted regions problem.

For the convenience of understanding the concept of our algorithm, the 8-*geometry* maze routing algorithm is described as a function with one terrain. This is also for the simplicity of our presentation since the algorithm is applied to both the virtual obstacles expansion method and the optimal path planning algorithm.

*Algorithm 1: The 8-geometry maze routing algorithm*
Initialization:

For each cell $C_{i,j}(F/O_{i,j}, AT_{i,j}, Vis_{i,j}, Dir_{i,j})$ in an $m \times n$ grid plane, the initial $F/O_{i,j}$ value is 1 if cell $C_{i,j}$ is in the smooth free space or $\infty$ if it is in the obstacle, $AT_{i,j} = \infty$ and $Vis_{i,j} =$ *false* for all cells, where $0 \le i \le m-1,\ 0 \le j \le n-1$. The initial value of *index* is 0.

Input the coordinates of the source cell $S$ and the destination cell $D$. If the source cell $F/O_{i,j} = 1$, then update $AT_{i,j} = 0$.

Step 1:
- Step 1.1: Insert the source cell $S$ into the $TL$ and update the source cell's *Vis*$_{i,j}$ to *true*.
- Step 1.2: Insert the indices of the source cell into the $LL_0$.

Step 2: For each cell in the $LL_{\text{index}}$, update the $AT_{i,j}$ values of its neighboring cells.
- Step 2.1: If the destination cell $D$ is removed from the $LL_{\text{index}}$, then break.
- Step 2.2: Remove the indices of the first cell $C_{i,j}$ from the front end of the $LL_{\text{index}}$.
- Step 2.3: Update the time of arrival $AT_{i,j}$, of the 8 *geometry* neighbors of cell $C_{i,j}$. For each $C_{i,j}$'s neighbor $C_{i',j'}$, if the cell's $F/O_{i',j'} = 1$ then
  - Case 1: $|i'-i|^2 + |j'-j|^2 = 1$ call Update_*AT&Vis*$((i',\ j'), AT_{i,j}+1,$ *Dir*$_{i',j'})$
  - Case 2: $|i'-i|^2 + |j'-j|^2 = 2$ call Update_*AT&Vis*$((i',\ j'),\quad AT_{i,j}+\sqrt{2},$ *Dir*$_{i',j'})$
  - Case 3: $|i'-i|^2 + |j'-j|^2 = 5$ call Update_*AT&Vis*$((i', j'), AT_{i,j}+\sqrt{5},$ *Dir*$_{i',j'})$
- Step 2.4: If $LL_{\text{index}}$ is not empty, then go to step 2.

Step 3: Insert the cells' indices of the $TL$ into their corresponding buckets.
- Step 3.1: For all the indices in the $TL$, move indices ($i$, $j$) from $TL$ into $LL_{\lfloor AT_{i,j} \rfloor \bmod 4}$.
- Step 3.2: If the $TL$ is empty, then update the *index* value by *index* = *index* $+1) \mod 4$.

Step 4: If two consecutive buckets are not empty, then go to step 2.

END {The 8-geometry maze routing algorithm}

The function Update_*AT&Vis*$((i,\ j),$ *new*_$AT_{i,j}$, *Dir*$_{i,j})$ updates the values of $AT_{i,j}$ and *Dir*$_{i,j}$ if *new*_$AT_{i,j}$ is smaller, inserts the indices of $C_{i,j}$ into the bucket $TL$ if *Vis*$_{i,j}$ is *false*, and then, updates *Vis*$_{i,j}$ to *true*. Once the 8-*geometry* maze routing algorithm is executed, the $AT_{i,j}$ values of all the cells between the source cell and the destination cell are uniformly propagated and updated to their minimum values.

## III. Optimal Path Routing Algorithm Among Weighted Regions

It is well known that the wave propagation scheme has a natural capability to find a weighted optimal path. In this maze routing algorithm, the parameter *F/O* is introduced for weighted regions in which the speed of the object for those cells in the specific regions is divided by the value of its parameter *F/O*. The algorithm uses uniform wave propagation to process the movement of the wavefront, the boundary of the propagating wave, from the source position, refracting when it encounters regions with varying costs, until it reaches the destination. The $AT$ value of the cell in the weighted regions is equal to the value of parameter *F/O* times the original $AT$ value of traveling in the free space. The uniform wave propagation for weighted regions can be achieved by increasing a certain number of buckets as temporary storage spaces for $AT$ values. For instance, if a weighted region with *F/O* value of 2, then the algorithm requires five circular buckets for the 8-geometry router. Hence, there is no need for time-consuming data structures like heaps or Fibonacci heaps [27], [28] used in the search procedure of Dijkstra's single-source shortest path algorithm [29]. The Fibonacci heaps are very slow in practice and often useless as far as run time improvements are concerned. The optimal path is also called the shortest path if there is only one terrain (weighted region) with a weight of 1 in the grid plane and the speed of robot is constant.

In the weighted regions, the path planning for an autonomous robot traveling over various terrains may include obstacles that are impenetrable (corresponding to a weight of $\infty$), or include other regions that just slow the movement down. Such regions are assigned a particular weight value to each of them. The weight of a region represents the amount of effort required to travel through that particular region, as compared to traveling with a flat, smooth surface (corresponding to a weight of 1). The higher values for the weighted regions represent the slower traveling speeds or the higher $AT$ values over the regions. The proposed algorithm, the optimal path routing among weighted regions, has been implemented and will be introduced as follows.

*Algorithm 2: The optimal path routing among weighted regions*

Initialization:

For each cell $C_{i,j}(F/O_{i,j}, AT_{i,j},$ *Vis*$_{i,j},$ *Dir*$_{i,j})$ in an $m \times n$ grid plane, the initial $F/O_{i,j}$ value is 1 if cell $C_{i,j}$ is in the free space or $\infty$ if it is in the obstacle. If the parameter *F/O* of any cell has a finite value in between 1 and $\infty$, the cell belongs to one of the weighted regions. $AT_{i,j} = \infty$ and $Vis_{i,j} = false$ for all cells, $0 \le i \le m-1$, $0 \le j \le n-1$. The initial values of *index* $F/O_{i,j}$ and *MaxF/O* are zeros, where *MaxF/O* is the maximum value of $F/O_{i,j}$.

Step 1: Input the coordinates of the source cell $S$ and the destination cell $D$. If $F/O_{i,j}$ of the source cell is not equal to $\infty$, then update $AT_{i,j} = 0$, else return the error message "The source cell is in the obstacle" and terminate. If $F/O_{i,j}$ of the destination cell is equal to $\infty$, then return the error message "The destination cell is in the obstacle region" and terminate.

Step 2: Searching the *MaxF/O* value for all cells.

For $i = 0$ to $m-1$

For $j = 0$ to $n-1$

If *new*_$F/O_{i,j} >$ *MaxF/O*, then *MaxF/O* = *new*_$F/O_{i,j}$

End {For}

End {For}

Step 3: Compute the number of *Buckets* required (*Max_Cir*).
- Step 3.1: Acquire the *MaxF/O* value.
- Step 3.2: Update the *Max_Cir* value.

$$\textit{Max\_Cir} = \frac{(\textit{MaxF/O} \times 22) + ((\textit{MaxF/O} \times 22)\%10 + 9)}{10} + 1$$

Step 4: Update the time of arrival between the source cell and the remaining cells.

Step 4.1:

Step 4.1.1: Insert the source cell $S$ into the $TL$ and update the source cell's $Vis_{i,j}$ to *true*.

Step 4.1.2: Insert the indices of the source cell into the $LL_0$.

Step 4.2: For each cell in the $LL_{\text{index}}$, update the $AT_{i,j}$ values of its neighboring cells.

Step 4.2.1: If the destination cell $D$ is removed from the $LL_{\text{index}}$, then break.

Step 4.2.2: Remove the indices of the first cell $C_{i,j}$ from the front end of the $LL_{\text{index}}$ and update this cell's $Vis_{i,j}$ to *true*.

Step 4.2.3: Update the time of arrival $AT_{i,j}$, of the 8-*geometry* neighbors of cell $C_{i,j}$. For each $C_{i,j}$'s neighbor $C_{i',j'}$, if the cell's $F/O_{i',j'} \neq \infty$ then

Case 1: $|i'-i|^2 + |j'-j|^2 = 1$ call Update_*AT&Vis*_with_*WR*$((i',\ j'),\ AT_{i,j} + (1 \times F/O_{i',j'}))$

Case 2: $|i'-i|^2 + |j'-j|^2 = 2$ call Update_*AT&Vis*_with_*WR* $((i',\ j'),\ AT_{ij} + (\sqrt{2} \times F/O_{i',j'}))$

Case 3: $|i'-i|^2 + |j'-j|^2 = 5$ call Update_*AT&Vis*_with_*WR* $((i',\ j'),\ AT_{ij} + (\sqrt{5} \times F/O_{i',j'}))$

Step 4.2.4: If $LL_{\text{index}}$ is not empty, then go to step 4.2.

Step 5: Insert the cells' indices of the $TL$ into their corresponding buckets.

Step 5.1: For all the indices in the $TL$, move indices $(i,\ j)$ from $TL$ into $LL_{\lfloor AT_{i,j} \rfloor \bmod \text{Max\_Cir}}$.

Step 5.2: If the $TL$ is empty, then update the *index* value by $index = (index + 1) \mod Max\_Cir$.

Step 6: If two consecutive buckets are not empty, then go to step 4.2.

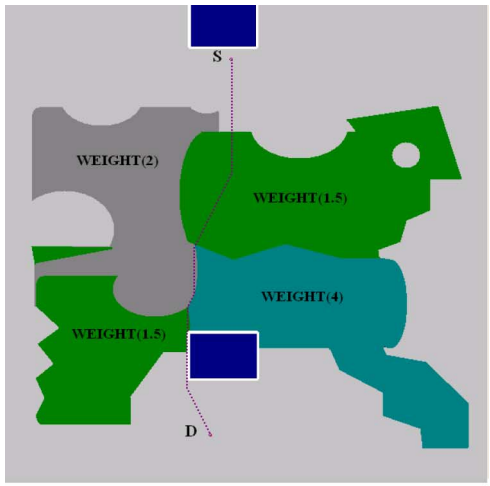
Step 7: Backtracking

If the $AT_{i,j}$ value of the destination cell is infinity, return the error message: "There is no existing path." Otherwise, track the optimal path from the destination cell by selecting one of the 8-*geometry* neighbors with the smallest time of arrival value and repeating the selection step by step until the source cell is reached. Reverse them to obtain the optimal path $LL_{\text{path}}$.

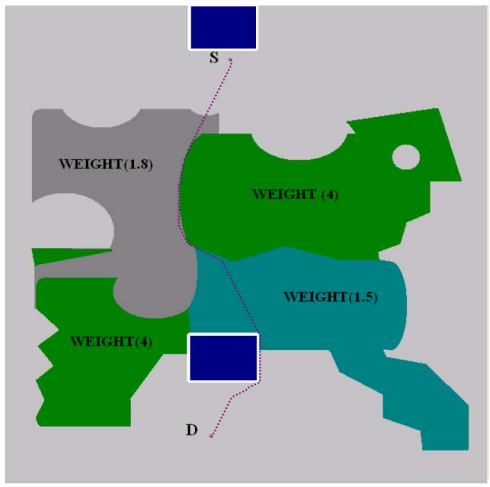
END {The optimal path routing among weighted regions}

The function Update_*AT&Vis*_with_*WR*$((i',\ j'), new\_AT_{i,j})$ updates the $AT_{i,j}$ value if $new\_AT_{i,j}$ is smaller, inserts the indices of $C_{i,j}$ into the bucket $TL$ if $Vis_{i,j}$ is *false*, and then, updates $Vis_{i,j}$ to *true*. Once the 8-*geometry* maze routing algorithm is executed, the $AT_{i,j}$ values of all the cells between the source cell and the destination cell are uniformly propagated and updated to their minimum values.

The dashed paths representing the desired optimal paths over varying terrains (varying weight values) are shown in Fig. 3(a) and (b). In this figure, it is assumed that the source is expressed by $S$, the destination is expressed by $D$, and the black, white, and gray areas represent the obstacles, virtual obstacles, and weighted regions (the number on it is the weight value), respectively.




(a)

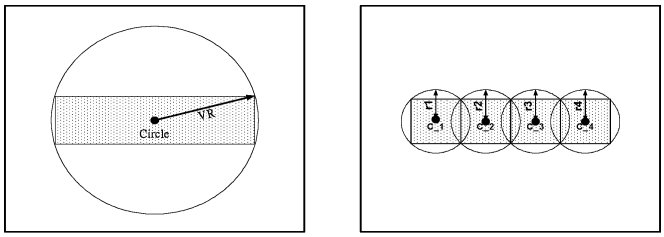



(b)

Fig. 3. Varying environmental terrains, assigned a different weight for each region, can result in different optimal paths.

## IV. Optimal Path Planning for Mobile Robot Navigation

The path planning problem involves several steps. First of all, take a top view image of the workspace or an image from the workspace arrangement. Secondly, the smallest circumscribing circle for the robot is obtained and the center of the circle will be treated as a single cell object to simulate the robot motion. This allows us to use a robot of any shape. After that, the obstacles are virtually expanded by a radius $VR$, by the virtual obstacles expansion method. Finally, by the higher geometry maze routing algorithm, one can easily obtain the optimal path for the robot motion. Both virtual obstacles expansion and path planning algorithm are based on the 8-*geometry* maze routing algorithm. To be more precise, a multiple circles model is generated to decrease the difference of area between the single circle model and robot configuration for dealing with some special-shaped robots by rotation scheme, which is known as a piano mover's problem.




(a) (b)

Fig. 4. (a) Single circle model and (b) multiple circles model for a robot configuration.

### A. Initial Approach

Once the image of the workspace is taken, an initial process of the robot configuration and the obstacles is required to obtain the free workspace. The configuration of a rigid robot contains many cells on a raster plane. We will focus on a specific cell and its corresponding cells (checkpoints) if the robot is treated as a single cell object. Thus, the intensive computation of the solid object is avoided. Given a robot of any shape in the workspace, we can circumscribe a smallest circle around the entire robot configuration. This is called the single circle model. The conventional method stores the robot image into a linked list and computes the mean values of the $x$ and $y$ coordinates as the approximated center of the circle. This center cell of the robot can be simply treated as a single cell object for the robot motion and the radius of the circle is exactly the width of the virtual obstacles $VR$, which will be applied to the virtual obstacles expansion.

When comparing the area of the single circle model with a rectangular robot, the difference of area is significant. More practically, a multiple circles model robot is generated for dealing with some rod-like robots. For the multiple circles model, the centers of the circles for each block are defined as the checkpoints. The maximum value of these smallest circumscribing radii is defined as the thickness of the virtual obstacles. The movement of a single cell object of this model is still obtained by the single circle model. For example, four circles cover a robot configuration as shown in Fig. 4(b). The centers of the four circles are considered as the checkpoints to verify whether the robot configuration interferes with the obstacles or not. For instance, the robot is able to move from cell $C_{i,j}$ to cell $C_{i',j'}$ that means all four coordinates of these circles do not interfere with all of the obstacles (including the virtual obstacles). Comparing Fig. 4(a) with (b), the multiple circles model is more precise and practical.

The single circle and multiple circles models for a rectangular robot configuration are presented as follows.

*Function 1: Single circle model*

Initially, set $a \leftarrow 0$, $b \leftarrow 0$ and $t \leftarrow 0$, where $2^a$ represents the number of cut blocks and $b$ is the sequence number of the cut blocks.

Step 1: Insert the cells of robot into linked list of robot $LR(a, b)$.
  For $i = 0$ to $m - 1$
    For $j = 0$ to $n - 1$
      If $R_{i,j}(t) =$ *true*, which means $C_{i,j}$ belongs to robot domain, then insert the $C_{i,j}$ into the $LR(a, b)$.
    End {For}
  End {For}

Step 2: Searching the center $C^o_{i,j}$ of single circle model for the robot configuration.
  Step 2.1: Set *count* $\leftarrow 0$, where *count* represents a counter that stores the number of cells removed from $LR(a, b)$.
  Step 2.2: Summation of indices in $X$ and $Y$ coordinates.
    Step 2.2.1: Remove a $C_{i,j}$ from $LR(a, b)$.
    Step 2.2.2: If *count* $= 0$, then
      $\left|C^o_{i,j}\right|_X = |C_{i,j}|_X$ and $\left|C^o_{i,j}\right|_Y = |C_{i,j}|_Y$.
      Otherwise,
      $\left|C^o_{i,j}\right|_X = \left|C^o_{i,j}\right|_X + |C_{i,j}|_X$ and $\left|C^o_{i,j}\right|_Y = \left|C^o_{i,j}\right|_Y + |C_{i,j}|_Y$
    Step 2.2.3: Update the value of the parameter *count*.
      count = count + 1
    Step 2.2.4: If $LR(a, b)$ is not empty, then go to step 2.2.
  Step 2.3: Obtain the $C^o_{i,j}$.
    $\left|C^o_{i,j}\right|_X = \left|C^o_{i,j}\right|_X /count$ and $\left|C^o_{i,j}\right|_Y = \left|C^o_{i,j}\right|_Y /count$

Step 3: Decide the width of the virtual obstacles $VR$.
  Step 3.1: Set $r \leftarrow 0$, where $r$ is the radius of the circle.
  Step 3.2: Remove a $C_{i,j}$ from $LR(a, b)$.
  Step 3.3: If $r < Dist(C_{i,j}, C^o_{i,j})$, then $r \leftarrow Dist(C_{i,j}, C^o_{i,j})$.
  Step 3.4: If $LR(a, b)$ is not empty, then go to step 3.2.
  Step 3.5: If $s = 1$, then $VR \leftarrow r$, where $s$ is the number of checkpoints.

END {The function of single circle model}

*Function 2: Multiple circles model*

Initially, set $a \leftarrow 0$, $b \leftarrow 0$ and $t \leftarrow 0$.

Step 1: Insert the cells of robot into the $LR(a, b)$.
  For $i = 0$ to $m - 1$
    For $j = 0$ to $n - 1$
      If $R_{i,j}(t) =$ *true*, then insert the $C_{i,j}$ into the $LR(a, b)$.
    End {For}
  End {For}

Step 2: Determine the number of the cut blocks for the robot.
  Step 2.1: Obtain the $C^o_{i,j}$ by the step 2 of single circle model function.
  Step 2.2: Remove all the $C_{i,j}$ from $LR(a, b)$ into the *Temp*.
  Step 2.3: Decide the $LR(a, b)$ for storing the cells of the cut blocks, where $a = a + 1$ and $b = 2 * b$.
  Step 2.4: Cutting process.
    Step 2.4.1: Remove all the $C_{i,j}$ from *Temp* into the corresponding $LR(a, b)$.
      Case 1: If $W \geq L$ and $|C_{i,j}|_X \geq \left|C^o_{i,j}\right|_X$, then move the $C_{i,j}$ into the $LR(a, b)$, where $W$ and $L$ represent the width and length of robot, respectively.
      Case 2: If $W < L$ and $|C_{i,j}|_Y \geq \left|C^o_{i,j}\right|_Y$, then move the $C_{i,j}$ into the $LR(a, b)$.

Case 3: If $W \geq L$ and $|C_{i,j}|_X < |C^o_{i,j}|_X$, then move the $C_{i,j}$ into the $LR(a, b+1)$.

Case 4: If $W < L$ and $|C_{i,j}|_Y < |C^o_{i,j}|_Y$, then move the $C_{i,j}$ into the $LR(a, b+1)$.

Step 2.4.2: If $a \leq (\log_2 q) - 1$, then follow the sequence number of $LR(a, b)$ to repeat step 2. Otherwise, go to step 3.

Step 3: Obtain $C^o_{i,j}$ and $VR$.

Step 3.1: Determine the $C^o_{i,j}$ by the step 2 of single circle model function.

Step 3.2: Obtain the $VR$ value by the step 3 of single circle model function.

Step 3.3: Insert $C^o_{i,j}$ into the $LC$.

Step 3.4: If $VR < r$, then $VR \leftarrow r$.

END {The function of multiple circles model}

After that, the obstacles are virtually expanded according to the thickness of $VR$. In this virtual obstacles expansion method [30], all the boundary cells of obstacles with at least one of the cell's neighbor in the free space are defined as the frontier cells. The virtual obstacles can be expanded by treating each frontier cell, the boundary cell of obstacles, as a special case of the source (initial) cell with limited propagation (expansion) radius. More precisely, insert the indices of all frontier cells into list $TL$ and then initialize the 8-geometry maze routing algorithm until the $AT$ values of the wavefront cells are greater than or equal to $VR$. Then, those cells whose $AT$ values are less than $VR$ are defined as virtual obstacles. The computation of virtual obstacles expansion is quite efficient since the virtual obstacles is obtained by applying the 8-geometry maze router only once for the entire frontier cells. The difference between the virtual obstacles and real obstacles is that the real obstacles have $AT_{i,j} = \infty$ and the virtual obstacles have $\lfloor AT_{i,j} \rfloor \leq VR$, where $\lfloor AT_{i,j} \rfloor$ means the floor of the $AT_{i,j}$ value. But, both of them have the same $F/O_{i,j}$ value that is $\infty$ (infinity).

The initial approach can be summarized as follows.

1) Determine the width and the length of robot configuration, the speed of the robot, and the coordinates of the source cell $S$ and the destination cell $D$. Also, obtain the radius of the smallest circle surrounding the robot.
2) Decide to work on the single circle or multiple circles model.
3) Execute the single circle or multiple circles model functions based on (2). The multiple circles model simplifies a robot as a single cell object with several checkpoints.
4) Execute virtual obstacles expansion method to obtain the virtual obstacles.

The following Sections IV-B and C are discussed in one terrain with the weight of 1. If the various weighted regions are included, the algorithm is working on the optimal path planning among weighted regions problem.

### B. Optimal Path Planning Without Rotation Scheme

In this section, the circle models, virtual obstacles expansion method, and the maze routing algorithm will be applied to the optimal path planning algorithm without rotation scheme. The primary scheme of this algorithm is first presented, and then, the detailed method of each step is described.

*Function 3: Virtual obstacles expansion*

Initially, set speed = 1 cell/unit time, therefore the $AT$ value has the same unit as virtual radius $VR$.

Step 1: Obtain the frontier cells from the obstacles.

For each frontier cell $(C_{i,j})_{\text{frontier}}$ insert the $(C_{i,j})_{\text{frontier}}$ into a linked list $VL$ if its $F/O_{i,j}$ value is *infinity* and at least one of its 8 neighbors $C_{i',j'}$ is in the free space.

Step 2: For each frontier cell $(C_{i,j})_{\text{frontier}}$ in the linked list $VL$, update the $AT_{i,j}$ values of its neighboring cells.

Step 2.1: Call the 8-geometry maze routing algorithm. This algorithm is modified to fit the following condition. If any cell $C_{i,j}$ in the current bucket that $\lfloor AT_{i,j} \rfloor \geq VR$, then break the function. Meanwhile, set the Boolean value of the $Vis_{i,j}$ back to *false* for all the propagated cells $C_{i,j}$ in the free workspace.

Step 2.2: Iterations

If $VL$ is not empty, then go to step 2.

Step 3: For each cell $C_{i,j}$, if $\lfloor AT_{i,j} \rfloor \leq VR$ then set $F/O_{i,j}$ value to $\infty$, else if $VR < AT_{i,j} < \infty$ then reinitialize $AT_{i,j} = \infty$.

END {The function of virtual obstacles expansion}

*Algorithm 3: The optimal path planning without rotation scheme*

Initialization:

For each cell, $C_{i,j}(F/O_{i,j}, AT_{i,j}, Vis_{i,j}, Dir_{i,j})$, in an $m \times n$ raster plane, the initial values are defined as shown in Fig. 2, where $0 \leq i \leq m-1$, $0 \leq j \leq n-1$.

Step 1: Determine the circle model.

Step 2: Call virtual obstacles expansion function.

Step 3: Call the *algorithm of the optimal path routing among weighted regions* to obtain the time of arrival ($AT$) between the source cell and the destination cell.

Step 4: Trace back

If the $AT_{i,j}$ value of the destination cell is infinity, return the error message: "There is no existing path." Otherwise, trace a shortest path from the *Dir* of the destination cell until the source cell is reached. Reverse them to obtain the optimal path $LL_{\text{path}}$.

END {The optimal path planning without rotation scheme}

The running time of the single circle model for both the virtual obstacles expansion method and path searching algorithm is only a small fraction of a second for a modest PC in a $400 \times 300$ raster plane. The time complexity for the multiple circles model with $s$ checkpoints is increased to $O(sN)$.

### C. Optimal Path Planning With Rotation Scheme

In this section, the robot motion will be much closer to a real object movement if the rotation scheme is applied. But to simplify the path planning problem, most of the algorithms only treat the robot configuration as a point, which is same as the single circle model and is depicted in Fig. 5(a). Thus, for some special-shaped robot configuration, the path generated by the single circle model may not be the optimal one if the robot

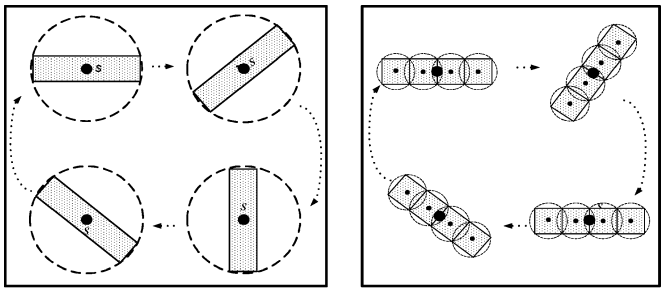


Fig. 5. Illustrations of robot rotation using (a) the single circle model and (b) the multiple circles model.

is only treated as a point. For instance, when a robot with long and skinny configuration is to traverse narrow passages, the approach may consider it impassable since the diameter of the circle is larger than the width of entrance. Therefore, the multiple circles model with rotation scheme is introduced, which is much more practical, and is shown in Fig. 5(b), where the rotation center of robot is treated as a single cell. This makes the long axis of the rod-like robot penetrate the passage by rotating it with a proper angle. Note that the center of rotation can be defined anywhere inside the robot object to emulate the situation in reality. The proposed method searches the optimal path in free workspace based on the interference detection between a single cell object, and its corresponding checkpoints if necessary, with all of the obstacles (including virtual obstacles).

Two rotation concepts are implemented in this scheme. First, while the robot is rotating, the long axis of the configuration, or moving axis, always keeps parallel to the path, which emulates the most of the real object movement as shown in Figs. 6(d) and 7. Second, for a multiple circles model with rotation scheme, an extra Boolean parameter $P(angle)_{i,j}$ is added in the cells data structure (the 8-*geometry* maze routing has 16 different selective directions, but the angle of rotation will be restricted for nonholonomic robots). Thus, once the direction is changed between $Dir_{i,j}$ and $Dir_{i',j'}$, we must verify all of their checkpoints to avoid the collisions between robot configuration and the obstacles. This means the robot will not interfere with the obstacles if it moves from the cell $C_{i,j}$ to $C_{i',j'}$, where the $C_{i',j'}$ is propagated by the maze router. The maximum chordal deviation for the rotation of the robot should be considered as the tolerance in the virtual obstacles since their thickness is the radius of the smallest circumscribing circle $VR$. The proposed algorithm, the optimal path planning with rotation scheme, has been implemented and will be introduced as follows.

*Function 4: Obstacles-avoidance during rotation*

DO

Step 1: Compute the $Vect$ value between $C_{i,j}$ and the cell $C_{i,j}^{LC}$, of linked list of checkpoints $LC$.

Step 1.1: Remove the cell $C_{i,j}^{LC}$ from the $LC$.

Step 1.2: Compute the $Vect$ value.

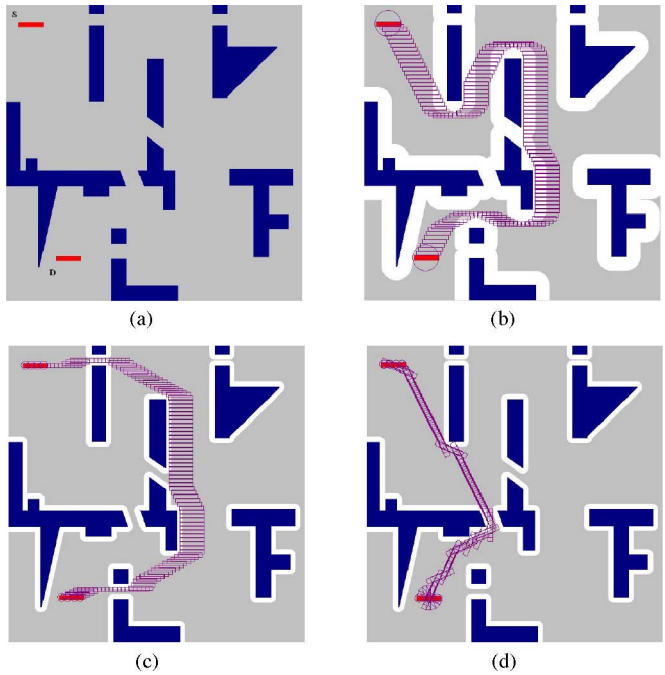


Fig. 6. Illustration of the path planning problem (a) and the optimal path for a mobile robot using the single circle model (b), the multiple circles model (c), and the multiple circles model with rotation scheme (d).

$|Vect|_X = \left|C_{i,j}^{LC}\right|_X - |C_{i,j}|_X$ and
$|Vect|_Y = \left|C_{i,j}^{LC}\right|_Y - |C_{i,j}|_Y$

Step 1.3: Insert the $Vect$ into the linked list of vector $LV(0)$.

Step 1.4: If $LC$ is not empty, then go to step 1.

Step 2: Compute the $Vect$ value of the 8-*geometry* neighbors of cell $C_{i,j}$.

Step 2.1: Set integer $angle = 0$

Step 2.2: Remove the $Vect$ from the $LV(0)$.

Step 2.3: Counterclockwise rotation $(\pi/8) * angle$

Step 2.4: If $W \geq L$, then insert the $Vect$ into the $LV(angle)$. Otherwise, into the $LV$ $((angle + 4) \bmod 16)$.

Step 2.5: If $LV(0)$ is not empty, then go to step 2.2.

Step 2.6: If $angle < 15$, then $angle = angle + 1$ and go to step 2.2.

Step 3: Compute the $P(angle)_{i,j}$ of the cell with $F/O_{i,j} = true$.

Step 3.1: Set $angle = 0$

Step 3.2: Remove the $Vect$ from the $LV$ $angle$).

Step 3.3: If the cell $F/O_{i,j} = true$, then $|C_{i,j}|_X = |C_{i,j}|_X + |Vect|_X$ and $|C_{i,j}|_Y = |C_{i,j}|_Y + |Vect|_Y$

Step 3.4: If the cell $F/O_{i,j} = false$, then $P(angle)_{i,j} = false$ and go to step 3.6.

Step 3.5: If the $LV(angle)$ is not empty, then go to step 3.2.

Step 3.6: If $angle < 15$, then $angle = angle + 1$ and go to step 3.2.

END {The function of obstacles-avoidance during rotation}

*Algorithm 4: The optimal path planning with rotation scheme*

Initialization:

For each cell, $C_{i,j}(F/O_{i,j}, AT_{i,j}, Vis_{i,j}, Dir_{i,j})$, in an $m \times n$ raster plane, the initial values are defined as shown in Fig. 2,

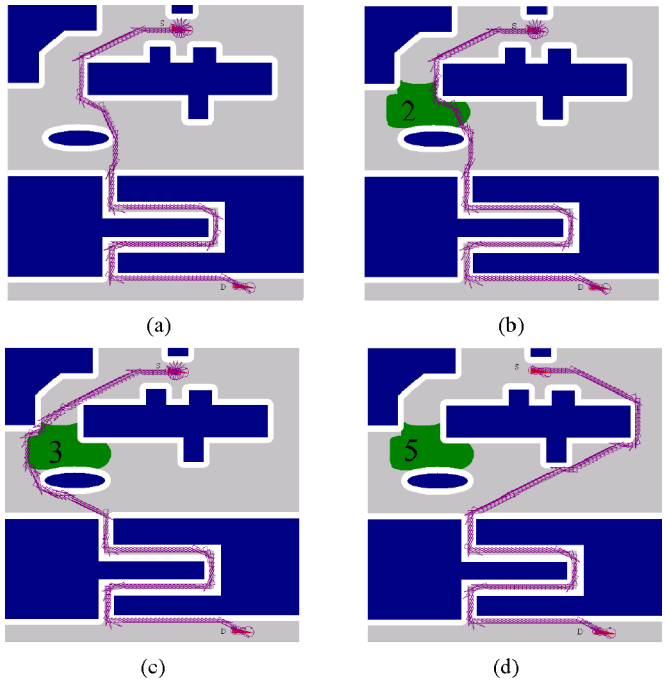



Fig. 7. Illustration of the path planning problem for the optimal paths to pass through narrow passages for a mobile robot using the multiple circles model with rotation scheme among single terrain (a) and varying terrains with different weights of 2, 3, and 5 to obtain different optimal paths (b), (c), and (d), respectively.

where $0 \le i \le m-1$, $0 \le j \le n-1$. The initial parameter $P(angle)_{i,j} = false$ and value of *index* is 0.

Step 1: Initialize the multiple circles model.

Step 2: Call virtual obstacles expansion function.

Step 3: Call the function of obstacles-avoidance during rotation.

Step 4: Call the *algorithm of the optimal path routing among weighted regions* to obtain the time of arrival ($AT$) between the source cell and the destination cell.

Step 5: Trace back

If the $AT_{i,j}$ value of the destination cell is infinity, return the error message: "There is no existing path". Otherwise, trace a shortest path from the *Dir* of the destination cell until the source cell is reached. Reverse them to obtain the optimal path $LL_{\text{path}}$.

END {The optimal path planning with rotation scheme}

## V. Examples of the Proposed Algorithms

This paper has implemented these approaches using the single circle and multiple circles models in the image plane, as shown in Figs. 6 and 7. In these image planes, the rectangular object represents robot configuration. Fig. 6(a) shows the source positions $S$ and the destination positions $D$ for the robot. Fig. 6(b) and (c) shows the expanded virtual obstacles and the obtained optimal paths by using the single circle and multiple circles models without rotation (only translation) scheme, respectively. The simulation results, Fig. 6(b) and (c), show that the multiple circles model has smaller virtual obstacles and the resulting path is faster than the single circle model (the path can move closer to the boundary of the obstacles). Thus, the multiple circles model passes two shortcut passages and resulting path is faster than the single circle model. For the multiple circles model with rotation scheme, Fig. 6(d) shows that the robot is able to traverse inclined passages and the resulting path is the fastest one. Furthermore, some other examples using multiple circles model with rotation scheme among single terrain and varying terrains are demonstrated. Fig. 7(a) shows the result that the multiple circles model with rotation scheme is applied among single terrain. Fig. 7(b), (c), and (d) shows the results of the optimal path planning with rotation scheme among the terrain with the different weights of 2, 3, and 5, respectively. The various environmental terrains (with different weights) result in different traveling speeds. The simulation results show that our algorithms are able to search a 2-D with 3 DOF (2-D 3 DOF) piano mover's problem among weighted regions with linear time and space complexities. Thus, a mobile robot can be intelligently rotated to pass through narrow passages with linear time and space complexities. It shows that our approach outperforms existing image approaches since most methods cannot obtain the optimal path in a map with complex concave obstacles.

## VI. Conclusion

Some optimal path planning algorithms in an image workspace or grid plane among obstacles and weighted regions have been presented. In this approach, the single cell object travels along the optimal path in the grid plane without any collision. For the multiple circles model, the checkpoints of the single cell object are ensured to be collision free with all of the obstacles. As a whole, the algorithms are very efficient comparing with other methods since no preprocessing to construct a suitable search graph is required. The concepts involved in the algorithms are simple and can be implemented in the image plane or grid plane. The proposed methods cannot only search an optimal path among weighted regions, but also intelligently rotate the robot configuration to pass through narrow passages with linear time and space complexities, which other image methods fail to accomplish.

In the future, the proposed algorithms will be extended in volume to solve the path planning problems for 3-D robot motions, which will be more practical in reality.

## References

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

[2] J. C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[3] K. Jiang, L. D. Seneviratne, and S.W. E. Earles, "Shortest path based path planning algorithm for nonholonomic mobile robots," *J. Intell. Robot. Syst., Theory Appl.*, vol. 24, pp. 347–366, Apr. 1999.

[4] R. J. Szczerba, D. Z. Chen, and J. J. Uhran, "Planning shortest paths among 2D and 3D weighted regions using framed-subspaces," *Int. J. Robot. Res.*, vol. 17, pp. 531–546, May 1998.

[5] T. C. Hu, A. B. Kahng, and G. Robins, "Optimal robust path planning in general environments," *IEEE Trans. Robot. Autom.*, vol. 9, no. 6, pp. 775–784, Dec. 1993.

[6] J. P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. Robot. Autom.*, vol. 10, no. 5, pp. 577–593, Oct. 1994.

[7] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 2, pp. 224–241, Mar./Apr. 1992.
[8] T. E. Boult, "Dynamic digital distance maps in two dimensions," *IEEE Trans. Robot. Autom.*, vol. 6, no. 5, pp. 590–597, Oct. 1990.
[9] A. Zelinsky, "A mobile robot exploration algorithm,," *IEEE Trans. Robot. Autom.*, vol. 8, no. 6, pp. 707–717, Dec. 1992.
[10] K. Subbarao and S. D. Larry, "Multiresolution path planning for mobile robots," *IEEE J. Robot. Autom.*, vol. RA-2, no. 3, pp. 135–145, Sep. 1986.
[11] C. Alexopoulos and P. M. Griffin, "Path planning for a mobile robot," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 2, pp. 318–322, Mar./Apr. 1992.
[12] N. Q. Wu and M. C. Zhou, "Shortest routing of bidirectional automated guided vehicles avoiding deadlock and blocking," *IEEE/ASME Trans. Mechatronics*, vol. 12, no. 1, pp. 63–72, Feb. 2007.
[13] N. Q. Wu and M. C. Zhou, "Deadlock modeling and control of automated guided vehicles systems," *IEEE/ASME Trans. Mechatronics*, vol. 9, no. 1, pp. 50–57, Mar. 2004.
[14] X. Yuan and S. X. Yang, "Multirobot-based nanoassembly planning with automated path generation," *IEEE/ASME Trans. Mechatronics*, vol. 12, no. 3, pp. 352–356, Jun. 2007.
[15] P. L. Lin and S. Chang, "A shortest path algorithm for a nonrotating object among obstacles of arbitrary shapes," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 825–833, May 1993.
[16] J. L. Dí-az de León S. and J. H. Sossa A., "Automatic path planning for a mobile robot among obstacles of arbitrary shape," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 467–472, Jun. 1998.
[17] K. Y. Chang, G. E. Jan, and I. Parberry, "A method for searching optimal routes with collision avoidance on raster charts," *J. Navigat.*, vol. 56, no. 3, pp. 371–384, 2003.
[18] Y. L. Lin, Y. C. Hsu, and F. S. Tsai, "Hybrid routing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 9, no. 2, pp. 151–157, Feb. 1990.
[19] G. E. Jan, M. B. Lin, and Y. Y. Chen, "Computerized shortest path searching for vessels," *J. Marine Sci. Technol.*, vol. 5, pp. 95–99, Jun. 1997.
[20] J. Fawcett and P. Robinson, "Adaptive routing for road traffic," *IEEE Comput. Graph. Appl.*, vol. 20, no. 3, pp. 46–53, May/Jun. 2000.
[21] C. Y. Lee, "An algorithm for path connection and its applications," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346–365, Sep. 1961.
[22] J. H. Hoel, "Some variations of Lee's algorithm," *IEEE Trans. Comput.*, vol. C-25, no. 1, pp. 19–24, Jan. 1976.
[23] G. E. Jan, K.-Y. Chang, S. Gao, and I. Parberry, "A 4-geometry maze routing algorithm and its application on multi-terminal nets," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 116–135, 2005.
[24] R. Kimmel, A. Amir, and A. M. Bruckstein, "Finding shortest paths on surfaces using level sets propagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 6, pp. 635–640, Jun. 1995.
[25] Z. Xing and R. Kao, "Shortest path search using tiles and piecewise linear cost propagation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 2, pp. 145–158, Feb. 2002.
[26] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. Boston, MA: Kulwer, 1999.
[27] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *J. ACM*, vol. 37, pp. 213–223, Apr. 1990.
[28] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian, "Faster shortest-path algorithms for planar graphs," *J. Comput. Syst. Sci.*, vol. 55, pp. 3–23, Aug. 1997.
[29] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, pp. 269–271, 1959.
[30] G. E. Jan, K. Y. Chang, and I. Parberry, "A new maze routing approach for path planning of a mobile robot," in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, Jul. 20–24, 2003, vol. 1, pp. 552–557.

**Gene Eu Jan** received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 1982, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Maryland, College Park, in 1988 and 1992, respectively.

During 1991, he was a Visiting Assistant Professor in the Department of Electrical and Computer Engineering, California State University, Fresno. From 1993 to 2004, he was an Associate Professor in the Departments of Computer Science, and Navigation, National Taiwan Ocean University, Keelung, Taiwan. Since 2004, he has been a Professor in the Department of Computer Science and Institute of Electrical Engineering, National Taipei University, Taipei, where he is also the Dean of the College of Electrical Engineering and Computer Science. His current research interests include parallel computer systems, interconnection networks, motion planning, electronic design automation (EDA), and very large scale integration (VLSI) systems design.

**Ki Yin Chang** received the Ph.D. degree in mechanical engineering from Michigan State University, Lansing, in 1991.

During 2003, he was a Visiting Associate Professor in computer science and engineering at the University of North Texas, Denton. He is currently a Professor and the Chairman of the Merchant Marine Department, National Taiwan Ocean University, Keelung, Taiwan, R.O.C. His current research interests include navigational search and interception systems, motion planning, vessel traffic service (VTS) systems/automatic identification systems (AISs), marine geographic information systems (GISs), and collision avoidance decision systems.

**Ian Parberry** received the Ph.D. degree in computer science from the University of Warwick, Coventry, U.K., in 1984.

He is a Professor of computer science and engineering at the University of North Texas (UNT), Denton, where he also directs the Laboratory for Recreational Computing (LARC). He is the author or coauthor of six books and more than 70 articles on a wide range of computing subjects including algorithms, complexity theory, parallel computing, neural networks, and entertainment computing. Recently, he has been engaged in work on procedural generation of game assets, visibility determination, cloud generations and rendering, and audio games. His current research interests include procedural clutter generation, general processing on graphics processing unit (GPGPU), and computer vision.