NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont eté dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

A FLUID MODEL FOR ROBOT PATH PLANNING IN A TIME INVARIANT ENVIRONMENT

ZIXI LI

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 1994

ISBN 0-612-10871-6

Abstract

A Fluid Model for Robot Path Planning in a Time Invariant Environment

Zixi Li

Intelligent robot motion planning has been one of the main trends in robotic research for more than three decades, and colorful variety of techniques are proposed. Among them, numerical techniques have obtained particular attention, because in many cases, they are able to achieve real time performance. In this thesis, a new numerical technique for robot motion planning is presented. It is initialized from the theories of ideal fluid in fluid mechanics. The path planning process is just like a small particle in the fluid finding its way out, driven by the velocity potentials along the endless flow. In a predefined space, usually an enclosed space littered with obstacles, velocity potentials are computed by Poisson's equation. Because the simulation is constrained by a harmonic function, no spontaneous creation of local minima will occur, which plague some other potential field methods.

Various iteration techniques are experimented with the simulation in order to choose the best suitable method. The natural flowing path is determined by a depth-first algorithm and other heuristic algorithms are proposed to further improve the path. A heuristic bitmap technique for collision avoidance in two-dimensions (3 DOF) is also suggested. This model contains high potential for parallel processing. Experiments on a vector machine shed light on this approach. Although, the technique described in this thesis deals with global path planning in a 2-D environment, it can be extended to n-dimensional domains.

Acknowledgments

I would like to specially thank my thesis supervisor, Dr. T. D. Bui, for his many valuable instructions and encouragements along the process of this work. In fact, this project was initialized by him and without his support, both theoretically and financially, it would not have succeeded. I would also like to thank Dr. L. Tao, for his help in our experiments on a vector supercomputer. I acknowledge Mr. Quanlin Gu for his help in computation techniques, Dr. Xinming Yu for his support in collecting references in this research field. Thanks are also given to Paul Gill, Michael Assels and William Wong, who helped me to solve many technical problems in computation and in using the computer facilities of our department.

I cherish memories to my beloved daughter, Yao, and this work is dedicated to her. Her love to her father is forever the strength to urge him forward. I also thank my wife Denglian for her unselfish support and understanding, and my two lovely children, Qin and Gang.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robot motion planning has been attracting a lot of research for three decades and many planning methods have been proposed from various scientific fields such as differential geometry, topology, graphics and mechanics, etc. In fact, it is a colorful combination of varieties of sciences and technologies and it is called an important subset of *Spatial Reasoning* (see [Lat91]).

Motion planning is by no means a simple problem. J.F. Canny has proved in [Can88] that dynamic motion planning for a point robot in the plane with a bounded velocity modulus and arbitrarily many obstacles is NP-hard, even when the moving obstacles are convex polygon moving without rotation at constant linear velocity. This conclusion discloses the difficulties we face in the robot path planning, and the reason why it encourages wide research interests on this field.

In this introduction, we first give a historic review of this literature, and then we briefly discuss the most important approaches and methods.

1.1 Classification of Motion Planning Problems

Classically, path planning is divided into two problems. One is the general mover's problem, which determines if there exists a path between two predefined locations (e.g., A and B). Any collision-free path joining A and B at its two extremities is a solution to the problem. The other is the shortest (or optimal) path problem. It does

not only determine the existence of paths, but also tries to find the shortest one when a set of paths are proved to be present. Nilsson initiated the trend of the research on the shortest path problem in computational geometry by proposing *Visibility graph* (see [Nil69]). It is the earliest path planning method in the Roadmap Approach. (A more detailed explanation will be given in the following section.)

Generally, in the process of path planning, most of current algorithms consist of two phases. Phase one solves the general mover's problem and phase two solves the shortest path problem. Before starting the planning, the coordinates of objects (e.g., the obstacles, robots) in the working domain are assumed to be known. All the spatial information in the domain such as edges and vertices of the objects, the hierarchical relations among the free regions and so on are stored in a data base. Then the data base is searched by different techniques such as graph search, hierarchical tree search, or potential guided search, etc. to plan the path.

Very often, people do not only stress the path with shortest Euclidean distance. They try to find a path with good qualities. Sometimes, the term, "good" or "ideal" path is used. It emphasizes a path with good attributes such as less danger of collision with obstacles, more flexibility in navigation and a reasonably short distance, etc.

Other classifications of motion planning problems still exist (see [HA92]). According to the availability of information of obstacles, there are *static* and *dynamic* problems. In a static problem, all the information about obstacles is known a *priori*, and the motion of the robot is designed from the given information. In dynamic planning, only partial information about the obstacles is available. e.g., the visible parts of the obstacles. The robot plans a path based on the available information. The information updating and path planning are interchanged continuously until the robot reaches its goal. Most of the papers in motion planning have dealt with the static case.

Based on whether there exist moving obstacles or not, the motion planning can be *time-varying* or *time-invariant*. Or according to whether there are inherent restrictions on the motion of robots, e.g., the bounds on robot's velocity and acceleration, the constraint on the curvature of robot's paths and so on, motion planning is either *constrained* or *unconstraint*.

Motion planning is *conformable* if the shape of robots can be changed, e.g., a multiple linked robot. Otherwise it is *nonconformable*. Also according to the number of robots involved in path planning, there is *single* or *multiple robots* problem.

In a word, all those classifications of the problems indicate the complexity people are facing in this field. And it will become more and more intriguing as industries require solutions in higher dimensional spaces and with higher degrees of freedom.

1.2 Configuration Space

Generally, robots have their own activity boundaries. We call all those areas inside the boundaries the working space. In other words, a working space (also called world space or physical space) is where the robot's activities are bounded. It is very difficult to plan the robot's motion directly in the working space, because the robot may have different shapes and have the capability of rotations. An effective idea is to create another space, in which the robot is presented as a point only. Then the robot's translation and rotation problem, the global path planning problem and the collision avoidance problem can be solved simultaneously. This space is called the *configuration space*. However, the configuration space has to be consistent with the working space in view of the robot path availability. Therefore, a transformation function is needed. Let $\mathcal{W}$ be the working space, and Ω be its configuration space. Also we assume only rigid objects are the concern of our research. Then we have the transformation function

$$\mathcal{T} : \mathcal{W} \rightarrow \Omega. \tag{1}$$

Here $\mathcal{T}$ is decided by several attributes like the dimension of the working space, the shape of the robots and the obstacles. Fig. 1 shows an example of the transformation from a 2-D working space with a convex-polygon-shaped robot and obstacles to its configuration space, using the algorithm proposed by [LP83]. In this example, the robot has only two degrees of freedom (2 DOFs). The verification of this transformation is shown in Fig. 2, in which the typical contacts of the robot with the obstacle is specified. The free space for the robot in Ω is denoted C_{free} or $\mathcal{F}$ later in the thesis, and the space occupied by the obstacles is denoted $C_{obstacle}$ or $\mathcal{B}$. However, the

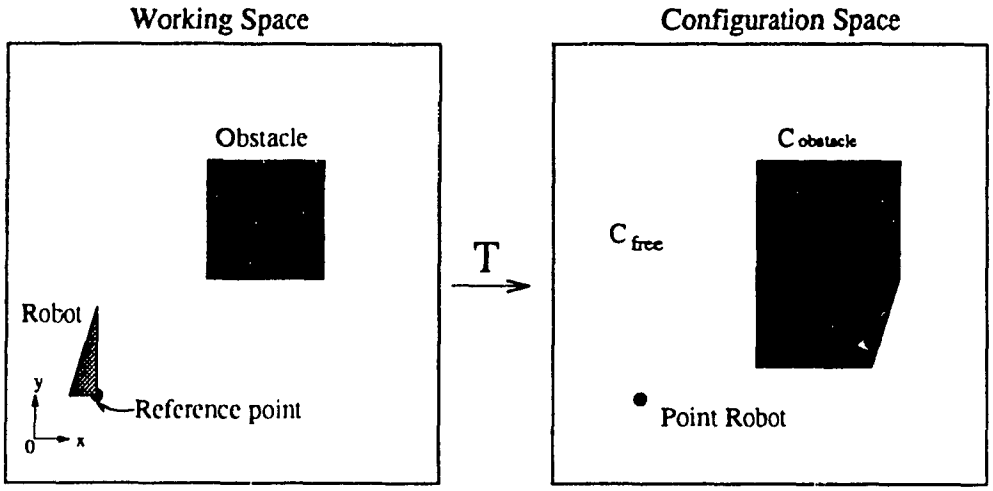


Figure 1: An example transformation by $\mathcal{T}$ for a triangular robot with 2 DOFs (no rotation).

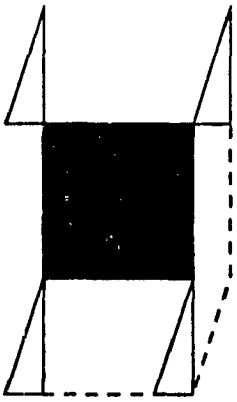


Figure 2: The typical positions of contact between the robot and the obstacle.

transformation function $\mathcal{T}$ is more complicated or even impossible for high degrees of freedom.

1.3 Main Approaches to Motion Planning

Up to now, hundreds of path planning algorithms are available for reference. They may be divided into three categories (see [Lat91]):

1. Roadmap methods.
2. Cell decomposition methods (either exact or approximate).
3. Potential fields.

This section follows the above division and discusses in detail the theories, applications, merits and deficiencies of each method.

1.3.1 Roadmap Approach

The basic idea of roadmap methods is to explore the connectivity of C_{free} in Ω through analysis of its geometrical and topological features such as the coordinates of the edges, vertices of the robot and the obstacles, etc. Then a network of one-dimensional curve representing the connectivity of C_{free} is extracted. The network is called *Roadmap*, or *Skeleton*. Once constructed, the roadmap is used as a set of standard paths. Path planning is therefore reduced to connecting the initial and goal configuration in the map and the map is searched for a path.

The earliest technique of this approach is *Visibility graph* proposed by Nilsson in 1969 (see [Nil69]). It leads to the research for finding the shortest path in computational geometry. This method connects every pair of vertices in C_{free} by a straight segment if it does not traverse the interior of a $C_{obstacle}$ (see Fig. 3). For a naive method, i.e., connecting every pair of vertices available (see [LPW79]), the time complexity is $O(n^3)$, where n is the total number of vertices of the obstacles. The performance of this method can be improved to $O(c^2 + n\log n)$ by constructing only the reduced visibility graph (see [Roy88]), where c is the number of disjoint convex

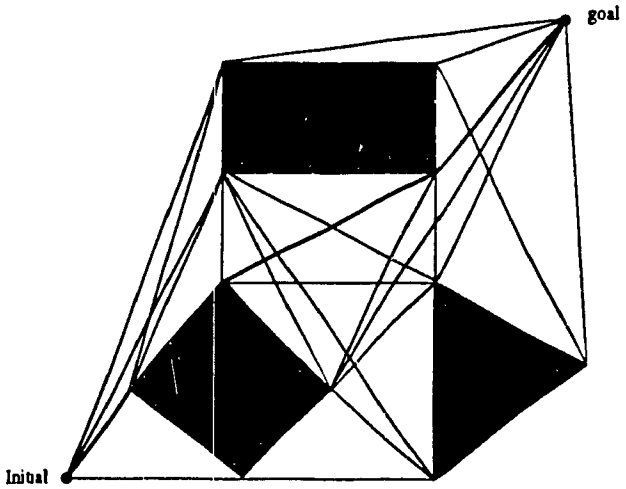


Figure 3: An example of visibility graph.

polygons. The generalized polygon technique can transform those non-polygons into polygons in order to apply this method. Also this method can be improved to $O(n^2)$ (see [Ede87]).

Another technique, *Retraction*, is the classical one in topology (see [K.84]). Let ρ be the retraction function, then

$$\rho : C_{free} \rightarrow R, \tag{2}$$

where $R(\subset C_{free})$ is a network of one dimensional curves. This is called *Preserved retraction*. [OY82] proposed a retraction of C_{free} onto its Voronoi diagram and hence maximized the clearance between the robot and the obstacles. Let $\beta = \partial C_{free}$, for any $q \in C_{free}$, let

$$\text{clearance}(q) = min_{p \in \beta} ||q - p||, \tag{3}$$

where $||q - p||$ is the Euclidean distance between q and p, and let

$$\text{near}(q) = \{p \in \beta / ||q - p|| = \text{clearance}(q)\}. \tag{4}$$

The Voronoi diagram of C_{free} is the set:

$$Vor(C_{free}) = \{q \in C_{free} / card(\text{near}(q)) > 1\} \tag{5}$$

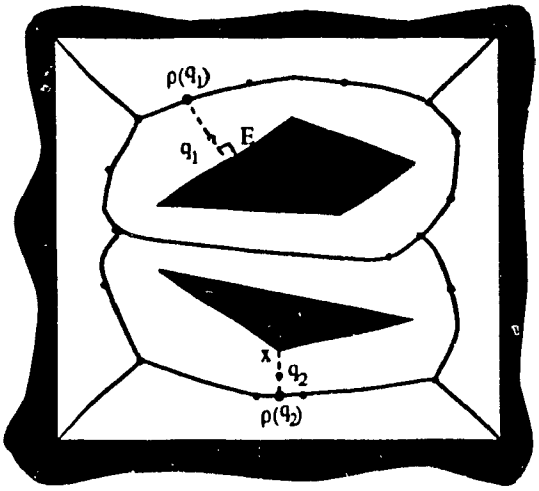


Figure 4: An example of Voronoi graph. Any configuration will be projected on the graph (e.g., $\rho(q_1)$ and $\rho(q_2)$ are the images of q_1 and q_2) and the graph is searched to find a path.

where $card(E)$ denotes the cardinality of the set E. Fig. 4 shows the result of O'Dunlaing's algorithm (the solid curves) and the image projections of q_1 and q_2 on the graph (the dotted lines). This algorithm is applicable only when $C = \mathbf{R}^2$. It can be computed in $O(n^2)$ time since the total number of arcs in $Vor(C_{free})$ is $O(n)$, where n is the sum of all the vertices and edges.

A general roadmap method called *Silhouette method* is proposed in [Can88]. It is the only known complete path planning algorithm which runs in single exponential time in Ω's dimension. This method sweeps out a plane along arbitrary direction in Ω and extract the extremal parts called *critical points* which are traced out to form the silhouette curves. Then the roadmap can be represented as a graph whose links are algebraic curve segments and whose nodes are the end points of these segments.

1.3.2 Cell Decomposition

Cell Decomposition method first decomposes C_{free} into a collection of non-overlapping regions called *cells*. The technique by which the union of the cells is exactly equal to

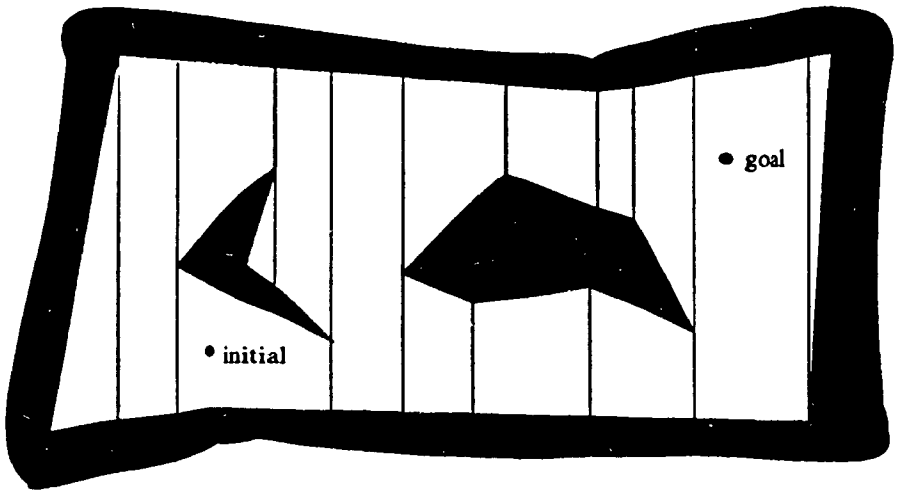


Figure 5: An example of trapezoidal decomposition.

C_{free} is called the exact cell decomposition, otherwise the approximate cell decomposition. The approximate cell decomposition requires that the cells have a simple prespecified shape such as a rectangloid. However, both decompositions construct a connectivity graph for C_{free} and search the graph to find a sequence of cells connecting q_{init} and q_{goal}. A path is then extracted from such a sequence.

Exact Cell Decomposition

The simplest but non-optimal method in exact cell decomposition is *Trapezoidal decomposition*, which is used only for convex polygonal configuration space. Each cell of the decomposition is either a trapezoid or a triangle. Two cells are adjacent if and only if their boundaries share a segment (see Fig. 5). This algorithm tries to reduce the number of the cells. Schwartz and Sharir (see [SS83]) solve translation and rotation of a polygonal robot $\mathcal{A}$ by modeling it as a line segment (called a ladder), decomposing the set of positions of $\mathcal{A}$ into 2D noncritical regions and representing the adjacency relation among the cells in a connectivity graph. This algorithm requires $O(n^5)$. Cylindrical algebraic decomposition, based on Colins decomposition, is also proposed by Schwartz and Sharir.

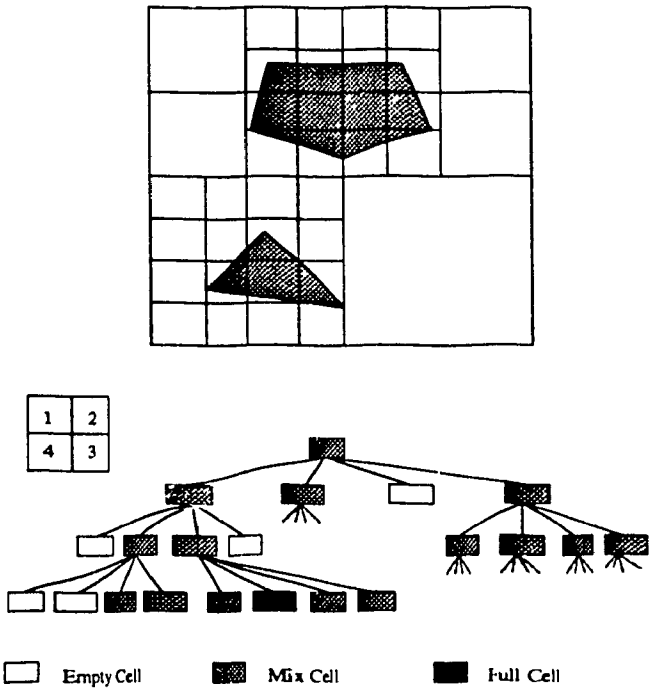


Figure 6: The hierarchical relations of the decomposed cells represented by a quadtree.

Approximate Cell Decomposition

A noticeable recent development on approximate cell decomposition is made by David Zhu and Jean-Claude Latombe (see [ZL91]). They adopted the hierarchical decomposition used in computer graphics, and divided the workspace into three kinds of regions: *free, occupied and mixed.* The mixed regions are subdivided into smaller regions again until it is distinct. The hierarchical relations between the free regions in the subdivision process are recorded and then the connected free regions to the goal are traced to find the path. Fig. 6 shows hierarchical relations of the cells by a quadtree.

1.3.3 Potential Field

Potential field is sometimes regarded as a local method in path planning. It consists of searching a grid placed onto the robot's configuration space (see [Don84]). Heuristic techniques from the partial information about the configuration space are used to

guide the search. A widely used heuristic technique guides the search for a path along the flow of the negated gradient vector field generated by an artificial potential field. Particularly, it treats the robot in configuration space as a particle under the influence of an artificial potential field. This method is first used in on-line collision avoidance (see [Kha80]). It seeks real time efficiency but does not always guarantee a solution. Most planning methods based on the potential field approach have a strong empirical flavor. They are usually incomplete. However, they are increasingly popular for implementing practical motion planners. Also the simple potential-guided path planning techniques do not assume any specific potential function.

A general potential field approach constructs artificial forces based on either the distance to the goal or the distance to the boundary of the objects; and heuristically searches the local paths guided by a combination of the forces. The artificial force $F(q)$ is composed of two forces: the attractive force $\vec{F}_{att}$ and the repulsive force $\vec{F}_{rep}$, where

$$\vec{F}_{att}(q) = -\vec{\nabla}U_{att}(q) = -\xi(q - q_{goal}) \tag{6}$$

in the case of a parabolic well, and

$$\vec{F}_{rep}(q) = -\vec{\nabla}U_{rep}(q) = \begin{cases} \eta(\frac{1}{\rho(q)} - \frac{1}{\rho_0})\frac{1}{\rho^2(q)}\vec{\nabla}\rho(q) & \text{if } \rho(q) \leq \rho_0, \\ 0 & \text{if } \rho(q) > \rho_0, \end{cases} \tag{7}$$

where U is a differential potential function and $U{:}C_{free} \rightarrow \mathbf{R}$, and $\vec{U}(q)$ denotes the gradient vector of U at the configuration q. ρ_0 is called the distance of influence of the C-obstacles.

The *depth-first planning* and the *best-first planning* are the two simple potential-guided planning techniques. The former simply follows the steepest descent of the potential function until the goal configuration is attained. For this technique, the strategy to escape from the local minima is under much concern. The latter embedded the configuration space with a fine regular grid GC. It consists of iteratively constructing a tree T whose nodes are configurations in GC. The root is q_{init}. At every iteration, the algorithm examines the neighbors of the leaf of T that has the smallest potential value. If their values are less than some large threshold, then join them to T as successors of the leaf being considered. Each node in T has a pointer

towards its parent. The algorithm terminates when q_{goal} has been attained (success) or all the accessible free subset of GC from q_{init} has been explored (failure). If success, a path is generated by tracing the pointer from q_{goal} to q_{init} (see [BL89]).

This approach offers an efficient way to solve collision avoidance problem. However, it may face serious local minima which often make the global path planning to fail. Many efforts are made to find a local minimum free environment for robots or seek some strategies to escape from local minima. There exist some escaping techniques but generally they are very expensive considering the time and the lower qualities of the path they produced.

1.3.4 Numerical Potential Methods

On the contrary, numerical potential methods can effectively avoid the trouble of local minima. Various numerical methods now exist in the literature. They often try to generate a local minimum-free environment. An early example of the potential field method is the charge distribution model proposed by Khatib [Kha85]. Some of the deficiencies of this method [KB91] are avoided by more recently reported potential field models satisfying Laplace's equation together with Dirichlet [CJ90] or Neumann boundary conditions [TB91]. These methods are ideally suited for implementation on massively parallel distributed processor systems, either digital [CJ90] or analogue [TB91]. A wave propagation strategy to optimal path planning in metric configuration space is first treated in Dorst and Trovato [DT88].

Simpler numerical potential methods have been proposed in [BL89], which prove to be efficient and powerful in dealing with many DOFs in robot path planning (see [BL89]). For example, the W-potentials, which are computed by a simple technique called *wavefront expansion* (see [Lat91]), is obtained by recursively assigning successive cardinal numbers from the goal configuration (q_{goal}) to unoccupied neighboring regions until all those connected free regions are labeled. However the resulted collision-free path tends to graze the boundary of C-obstacles, which restricts the movement of the robot. To avoid this, an improved W-potential method is proposed, in which a skeleton is extracted from free space and the path is found through tracing this skeleton using wavefront expansion. This skeleton is actually similar to a *Voronoi*

diagram in the workspace. This method maximizes the distance from the C-obstacles and gives the robot more freedom and less danger of collision. Of course, the resulted path is no longer optimal or even short. It is good for a workspace with densely and evenly distributed obstacles, however, it does not generate economical paths in other cases. Also it is too rigid in the choice of path planning space (i.e., always in the middle curve of the free channels bounded by the obstacles) and therefore it is unsuitable for planning multiple moving objects.

Connolly and Burns initiated the use of Laplace's Equation as a navigation function in path planning (see [CJ90]). They propose a global method which computes the solutions to Laplace's equation in arbitrary n-dimensional domain, and results in a weak form of what Rimon and Kodischek define as navigation functions (see [RK88]). The solutions of Laplace's equation are composed of a system of harmonic functions. Harmonic functions satisfy the min-max principle. Therefore, spontaneous creation of local minima within the region is impossible if Laplace's equation is imposed as a constraint on the function used.

What is a navigation function? A navigation function generally has to satisfy four properties (see [RK88]): (1) Analyticity, (2) Polar, (3) Admissibility, and (4) Morse. Every harmonic function ϕ defined on a compact region $\bar{\Omega} = \partial\Omega \cup \Omega$ satisfies three of the four properties for navigation functions. This may be seen as follows:

1. Analyticity: Every harmonic function is analytic [BS85].

2. Polar: Select a point q_d to be the goal point, constrain $\phi(q_d) = 0$ and set all obstacle boundary points p to some constant $\phi(p) = c$. Since all harmonic functions satisfy the min-max principle, ϕ is polar. In other words, q_d will be the point at which ϕ attains its minimum value on $\bar{\Omega}$. Hence, all streamlines of ϕ lead to q_d.

3. Admissibility: if we simply set the constant $c = 1$ above, then ϕ will be admissible in the sense of [RK88]. This is a simple normalization of ϕ.

4. The fourth property is that a navigation function is Morse (i.e., there are no degenerate critical points) [Mil70]. Every critical point of ϕ in Ω must be an isolated saddle point, from which streamlines may easily be found.

In [CJ90], Dirichlet boundary condition is used. If the starting point is known, another harmonic function can also be established. Then the harmonic functions can be combined using superposition techniques. One such function, which represents a point source (whose potential is infinite at the source), is $log(r(x,y))$, where $r(x,y)$ is the Euclidean distance from the source (x_0, y_0). The gradient for this function represents the vector field which would drive an effector away from the obstacles:

$$log(r(x,y)) = log(\sqrt{(x-x_0)^2 + (y-y_0)^2}), \tag{8}$$

$$\frac{\partial}{\partial x} log(r(x,y)) = \frac{x-x_0}{r(x,y)}, \tag{9}$$

$$\frac{\partial}{\partial y} log(r(x,y)) = \frac{y-y_0}{r(x,y)}. \tag{10}$$

Thus, the gradient for this function at a given point is always a unit vector in a direction away from the source point. Note also that the second partial derivatives with respect to x and y vanish everywhere, so Laplace's equation is satisfied. However, the superposition of harmonic functions presents problems. There is no guarantee for collision avoidance in complex or dynamic environments. The potential in the neighborhood of a given obstacle is a function not only of that obstacle's potential, but also of every other obstacle's (or goal's) potentials. By changing the configurations or strengths (which can easily happen in dynamic environments) the path of the robot can be led arbitrarily close to the obstacle. Therefore, the only structure that can be safely modeled this way is a point itself since the potential goes to infinity at the point source, and thus superposition of fields associated with sources at a finite distance cannot affect this.

1.4 About the theses

In fact, our research is directly developed from [CJ90] and [BL89]; Poisson's equation is employed to constrain the generation of the potential function over the region of the configuration space. This function is harmonic except at the two sets of points: sources and sinks. The generation of the potentials in the specified region is an analogue of ideal fluid flowing in that region. The idea in wavefront expansion (see

[BL89]) is further developed in our fluid model which has the same effect in finding a global collision-free path, and also, more importantly, it creates a beneficial environment for path planning, path correction and multiple robot navigation.

Chapter 1 acts as an introduction to the path planning literature. Chapter 2 states the theories of ideal fluid, the fluid simulation process and its mathematical representations. Correct results of fluid simulation is the base of the path planning and navigation algorithms. Chapter 3 explains and proves the generic properties of the domain simulated by the techniques stated in Chapter 2, and discusses the domain's structural features. Several definitions (or basic structures) related to the mechanism of a path in the fluid model are defined. Also, dynamic characteristics of the simulated ideal fluid in a closed domain are analyzed, which conclude that the existing forces can distort the path planning. Chapter 4 explains the basic path planning algorithm, SFM, its merits and deficiencies, and provides two techniques in path improvement. Chapter 5 proposes a bitmap technique for collision avoidance and chapter 6 compares the experiments on a variety of techniques for solving Poisson's equation. In this chapter, advanced computation techniques such as SOR and MAM are explained. Chapter 7 records the experiments on a high-powered parallel machine, and indicates the promising future of this method. Finally, Chapter 8 proposes two highly efficient path planning techniques, which result from modifications to the navigation function stated in Chapter 2.

Chapter 2

Simulation of Ideal Fluid

In Fluid mechanics, the term, *ideal fluid*, is used to indicate the behavior of a real fluid away from the boundaries. It is assumed that the fluid is incompressible and invisid. Its flow is therefore irrotational. These features enable generalization of physical problems and establishment of mathematical models for solving them.

2.1 Basics in Ideal Fluid

This section contains a brief introduction to the theory of ideal fluid which is closely related to our application. The volume flow rates denoted by ψ and the velocity potentials denoted by ϕ are two important elements in both the simulation of ideal fluid and the path planning. The flow pattern formed by the two types of flow lines, along which ψ and ϕ are constant respectively, acts as important references for our heuristic planning algorithms. Also our attention is confined to two-dimensional steady flow only.

2.1.1 Condition for Continuity of Ideal Flow

Suppose in an ideal fluid, there exists an open region S without interior sources. From the conservation law, the flow rate into S must be equal to the volume flow rate out of it (see [Mas90], Chapt. 10). Assume the fluid has a unit thickness, we have (see

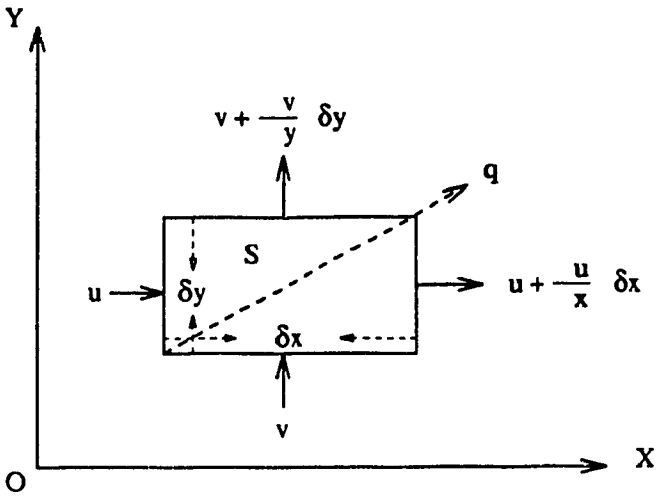


Figure 7: The continuity of flow of ideal fluid through region S, where $\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$.

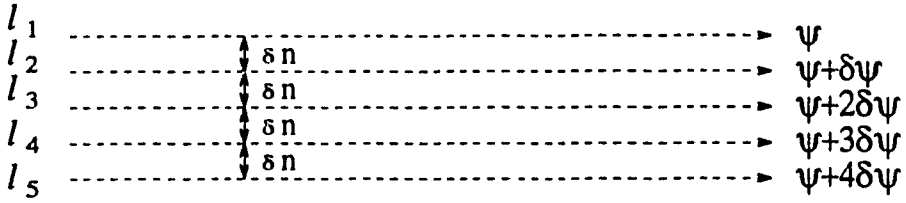


Figure 8: A straight and parallel flow pattern, where l_i are the streamlines and ψ is the volume flow rate constant on streamlines.

Fig.7)

$$u\delta y + v\delta x = (u + \frac{\partial u}{\partial x}\delta x)\delta y + (v + \frac{\partial v}{\partial y}\delta y)\delta x$$

$$= u\delta y + v\delta x + \delta x\delta y(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}). \tag{11}$$

where u, v are the components of q, the velocity. However, Eq. 11 is valid if and only if

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{12}$$

This is the condition for continuity of ideal fluid.

2.1.2 Stream Function ψ

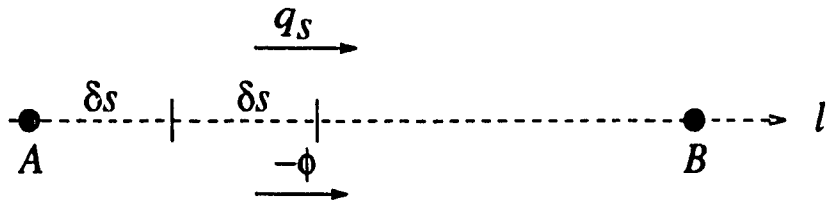


Figure 9: The relation between the velocity q along s and the velocity potential ϕ along s.

Conventionally, we use ψ to represent the volume flow rate, and the streamline shows the path of certain layer of fluid, along which ψ is constant. Therefore, the average velocity q

$$q = \frac{\partial \psi}{\partial n}, \tag{13}$$

where n is a vector normal to streamlines (see Fig. 8). If ψ is constant, the shorter the distance (δn) between streamlines, the faster the speed.

2.1.3 Velocity Potential ϕ

When the flow is irrotational, we can define the velocity potential as (see Fig. 9),

$$-\phi = \int_A^B q_s ds, \tag{14}$$

where q_s is the component of q along δs from A to B. The minus sign shows ϕ decreases in the direction of flow. Take differentiation of both sides of Eq. 14, we get

$$\delta\phi = -q_s \delta s, \tag{15}$$

thus

$$q_s = -\frac{\partial \phi}{\partial s}, \text{ as } \delta s \to 0, \tag{16}$$

It is clear, if δs is perpendicular to a streamline, then $q_s = 0$ and $\delta\phi = 0$. Therefore, ϕ is constant. We call the line with constant ϕ *equipotential line.*

From Eq. 16, we have

$$u = -\frac{\partial \phi}{\partial x}, \qquad v = -\frac{\partial \phi}{\partial y}, \tag{17}$$

Substitute Eq. 17 into Eq. 12, the condition of flow continuity, we get

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0, \tag{18}$$

or simply

$$\Delta \phi = 0, \tag{19}$$

where Δ is the second-order differential operator. Equation 18 is the two dimensional formula of Laplace's equation. Laplace's equation finds many applications in physics. Here, it can be used to approximate the flow pattern of a fluid by calculating the fluid velocity potentials at different small regions in its flow path.

2.1.4 Flow Net

For any two-dimensional irrational flow of an ideal fluid, two series of lines may be drawn. One is the streamlines, along which ψ is constant. The other is the equipotential lines along which ϕ is constant. Assume q along the streamline s is constant, then from Eq. 15, if $\delta\phi = 0$, $\delta s = 0$. The only line on which $\delta s = 0$ is the line perpendicular to s. Therefore, equipotential lines are perpendicular to their corresponding streamlines. Hence the lines of constant ψ and lines of constant ϕ together form a grid of quadrilaterals with 90 degrees of corners. This grid is called *flow net*. It provides a simple yet valuable indication of the flow pattern (see Fig. 10). If $\delta s = \delta n \rightarrow 0$, the quadrilaterals become complete squares.

From Newton' second law, the fluid flows from higher altitude to lower altitude out of the force of gravity or, in other words, of the velocity potential. Along with the decrement of altitude, it gains its horizontal movement. If we ignore the vertical movement of fluid, the horizontal one represents a path in two dimensional environment. Assume there are points A and B located in the passway of the fluid, a particle leaving off A and later reaching B actually draws a path between A and B. Tracing particles' movements in the fluid is the fundamental aspect of path planning in our fluid model.

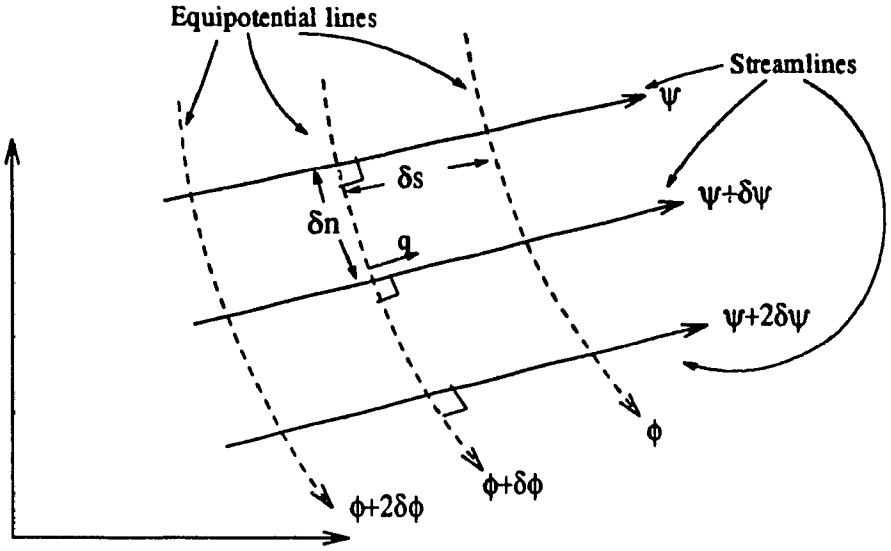


Figure 10: Elements of a flow net. The streamlines and the equipotential lines form a grid of quadrilaterals having 90 degree of corners.

2.2 Poisson's Equation and Velocity Potentials

2.2.1 Initialization of Ω

Conservation Law

From Section 1.1.3, velocity potentials, ϕ, in an ideal fluid satisfies Laplace's equation. Or in other words, Laplace's equation can be employed to simulate the ideal fluid through the computation of its velocity potentials in its flow domain. Let Ω be a closed domain for the flow, which is also referred to as the configuration space for the robot path planning. It is assumed that no fluid can flow in or out of the boundaries of Ω. Interior sources and inner boundaries (which, in our case, stand for the areas occupied by the obstacles in Ω) may be initialized in it. There are two kinds of interior sources, one (denoted by S^+) having the positive volume flow rate (i.e., the sources), the other (denoted by S^-) having the negative volume flow rate (i.e., the sink). In order to satisfy the conservation law in Ω, we define

$$R(S^+) = -R(S^-), \tag{20}$$

where $R(P)$ represents the volume flow rate at node P. Hence, we have (see [Roy88], [McC89])

$$\int_{\Omega} \hat{R} dV = 0, \tag{21}$$

where

$$\hat{R} = \sum_{i=1}^{n} R(S_i^+) + \sum_{j=1}^{m} R(S_j^-), \text{ where } n = 1, 2, \ldots \text{ and } m = 1, 2, \ldots \tag{22}$$

and V stands for the volume of the flow. Eq. 21 is the compatibility condition which states that the net flow rate in Ω must be zero. Only when this condition is satisfied, there exists a solution for Ω. Therefore, Eq. 21 imposes the constraint on the initialization of the sources and sinks.

Discretization of Ω

A classical way to seek numerical solutions in a domain is first to divide the domain into many subdomains. For our method, a grid is embedded in Ω, which has the size $M \times N$. A node is located on each vertex of those small rectangular regions, or to say, Ω is discretized and represented by a number of regular elements in its domain. For consistency, S^+, S^-, the vertices of objects in Ω, etc. are always defined on the grid nodes, and so are the points on which the values of ϕ are calculated and compared. However, the values of ϕ on the none-node locations can be obtained through proper interpolation functions.

2.2.2 Poisson and von Neumann Boundary Conditions

Theoretically, m and n in Eq. 22 may be different provided it satisfies Eq. 21. In robot path planning, however, we consider only a one-to-one problem, i.e., from the initial position of a robot, plan the path to its expected destination. In any instant, the robot's initial position and expected goal position are unique. Therefore, for simplicity, we define only two interior sources in Ω, one is the source, S^+, which represents the initial position of a robot. The other is the sink, S^-, the destination.

ϕ in an open ideal flow without sources and sinks satisfies Laplace's equation (see [Mas90]). For Ω defined in 1.2.1, ϕ satisfies Poisson's equation

$$\Delta\phi = -f, \tag{23}$$

or

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}=-f \text{ or } \frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}=-f, \tag{24}$$

and

$$f(x,y)=\begin{cases} -b & \text{if } (x,y)=S^+, \\ +b & \text{if } (x,y)=S^-, \\ 0 & \text{otherwise,} \end{cases} \tag{25}$$

where b is a positive constant and it can be conveniently assigned to be 1. It is clear that ϕ in Ω still satisfies Eq. 19, except at S^+ and S^-. The points at S^+ and S^- are called *singular points*. They require special consideration. Since no fluid is in or out of the boundary of Ω (denoted by $\partial\Omega$) or the boundary of obstacles, the Neumann boundary condition is the best choice for boundaries. Then we have

$$\frac{\partial\phi}{\partial n}=0 \text{ on } \partial\Omega, \tag{26}$$

where n is the normal vector to the boundaries. Eq. 26 indicates that the velocity along the normal vector at the boundary is zero. This property prevents the simulated fluid from flowing directly to the boundaries and excludes the interior occupied areas (the space occupied by obstacles) from computation. This technique leads to a very good solution to the collision problem encountered by many path planning algorithms.

2.2.3 Results of Simulation

This section gives several examples of fluid simulation. All the figures in this section are generated based on computer simulations, i.e., the computation of velocity potentials according to the parameterization of Ω using Eqs. 24-26. Fig. 11 shows the flow map of an ideal fluid in Ω without obstacles. The arrows show the main flow directions. The directions of arrows are determined by the Steepest Falling Method (SFM), a path planning algorithm stated in Section 5. The different sizes of the arrows indicate the magnitudes of the potential gradients. The bigger the size of an arrow, the larger the gradient. In Fig. 12, there are three obstacles in Ω. From both figures, we can notice at least two properties of the flow:

1. No arrow directs to either the boundary of Ω or the boundary of obstacles.

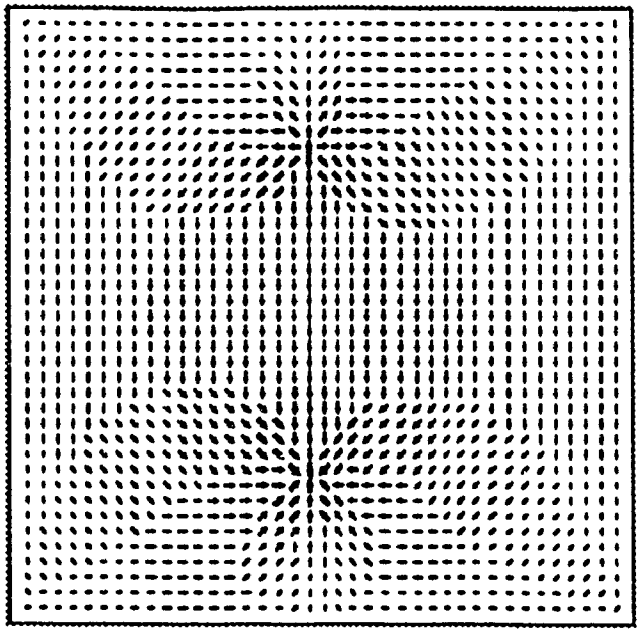


Figure 11: A sample of the flow map generated by the algorithms in 2.2.2. There are eight directions of an arrow and five sizes which show the different levels of the gradients of velocity potentials.

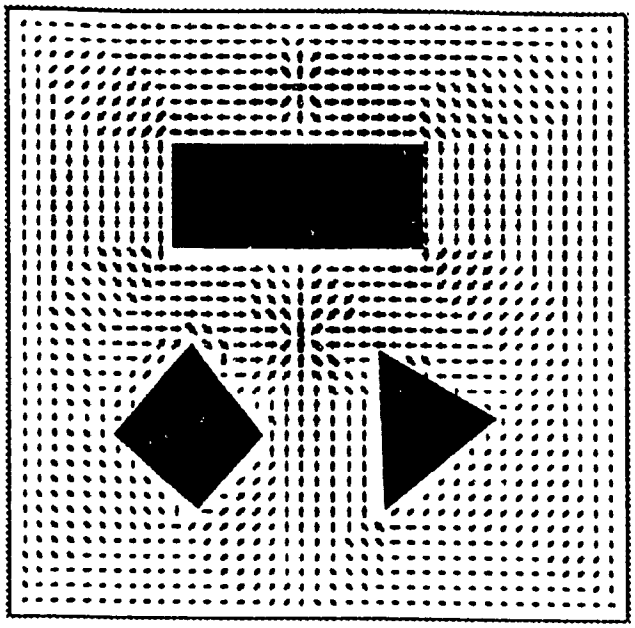


Figure 12: Another sample of the flow map generated by the algorithms in 2.2.2.

2. Starting from any arrow location in Ω, one can always reach the sink (S^-) by tracing the arrow heads successively.

3. The consecutive arrows indicate streamlines (i.e., paths) in Ω. One noticeable feature is that the streamlines, i.e., the available paths, are often parallel to each other, provided that they are not close to the boundaries, the source or the sink.

These properties prove the correctness of the simulation. Fig. 13 shows the equipotential contours of the flow in Fig. 11. They form cycles around S^+ and S^-. Taking any small region and combined with Fig. 11, it is not hard to find that the contour lines are perpendicular to the direction of the streamlines. Therefore, the two figures indicate a flow net in Ω. Similarly for Fig. 12 and Fig. 14.

Figure 13: A sample of the equipotential contours. The brightest spot is S^+ and the darkest spot is S^-. The value of velocity potentials decreases as the darkness of the shades increases.

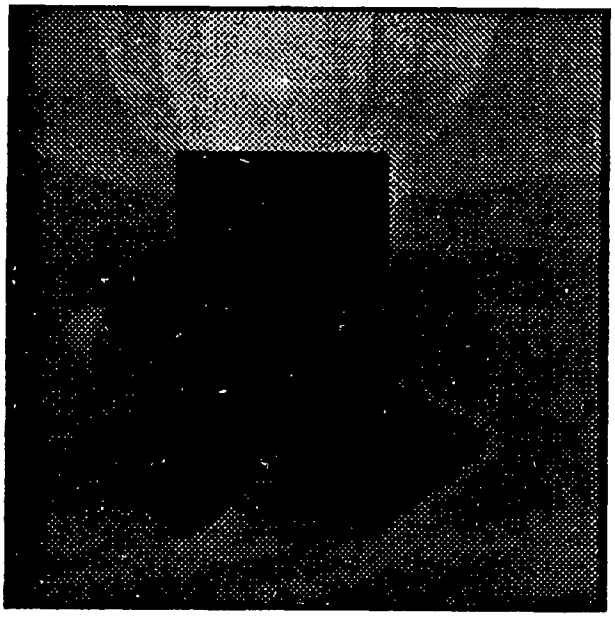


Figure 14: Another sample of the equipotential contours.

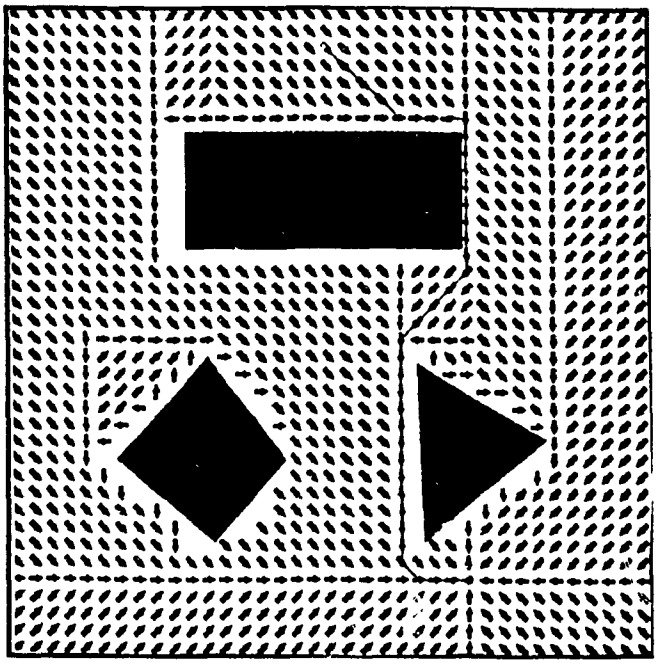


Figure 15: A sample of the flow map generated by wavefront algorithm.

2.2.4 Comparisons with Simple Potential Algorithm

The flow maps shown in Fig. 15 is generated by a simple potential algorithm called wavefront expansion or *W-potentials* which we mentioned in Chapter 1. It is comparable to Fig. 12. Also Fig. 16 and Fig. 17 are provided to show the differences in 3-D meshes generated by our fluid model and *W-potentials*. The flow directions in both maps are determined by SFM. Several things should be pointed out:

- Tracing the arrow from S^+, both maps can reach S^- without a danger of collision.
- Fig. 15 finds the shortest path but touches the boundary.
- Fig. 12 finds a short path which is properly away from the boundary.
- In Fig. 15, all streamlines converge to the shortest path, that is the valleys shown in Fig. 17. This makes it difficult for multiple robot navigation.

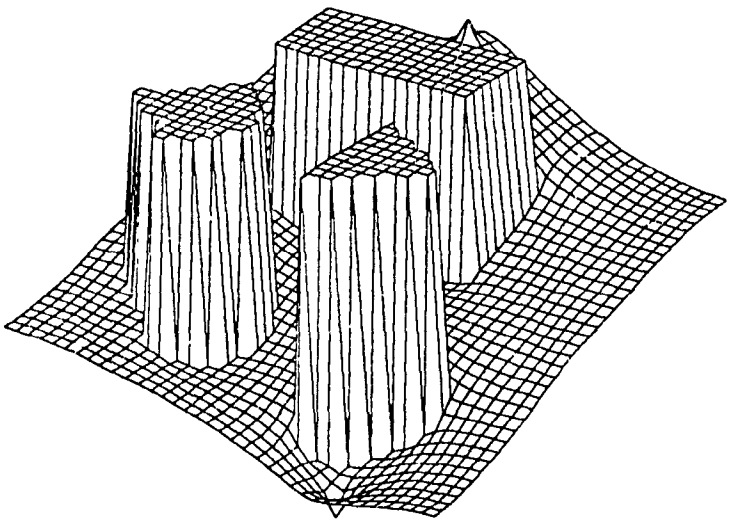


Figure 16: A 3-D mesh of computed potentials using fluid simulation.

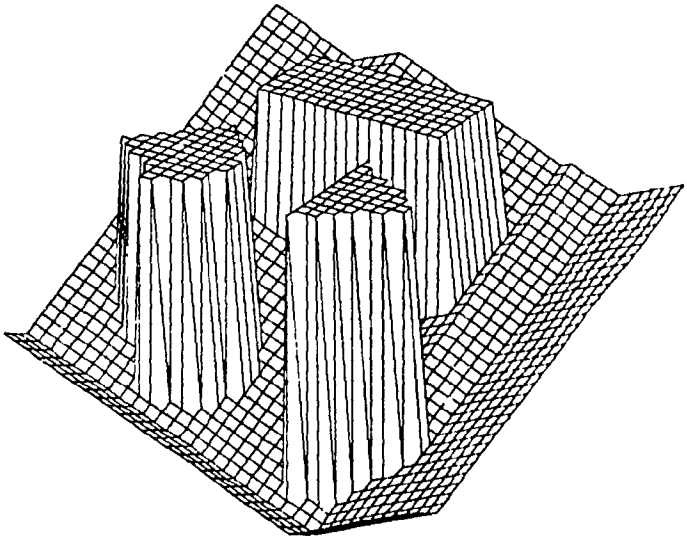


Figure 17: A 3-D mesh of computed potentials using W-potentials.

- In Fig. 15, all streamlines converge to S^-, the sink. The smooth declining surfaces (no valleys) in Fig. 16 imply the flexibility of path adjustments. The only place where arbitration is needed for solving contention is the small area around S^-.
- In Fig. 12, parallel property of streamlines occurs.
- In Fig. 15, no parallel property.
- In Fig. 12, different sizes of the arrows show different gradients of the velocity potentials generated. It is useful in flow direction detection, speed control, etc.
- In Fig. 15, all arrows are of the same size.

Clearly, from the comparison above, the ideal fluid model is superior to the simple cardinal fluid model. The former is more flexible. Each streamline is exactly a collision-free path, we can choose the best suitable path among them. The parallel feature of path lines makes the collision-avoidance easier in multiple robot navigation. Also a solution in Ω may be reused again and again provided the destination is not changed. The drawback of the former algorithm is that it takes more time in the generation of the potentials.

Chapter 3

Generic and Structural Properties

In Chapter 2, the basic theories of fluid simulation on a predefined domain are stated. Let Ω be such a domain. So Ω actually defines an environment for the designated robots. For constraining robots' activity in the domain, and making the experiment more realistic, very often different shapes of 'obstacles' are created in it. Robots can only move freely in those regions where the imaginary fluid can access. Those regions occupied by 'obstacles' are like high beaches at sea, the fluid passes along them but never goes over them. Only those regions in Ω accessible by fluid need to be computed. Therefore, in view of the fluid simulation, two spaces in Ω are distinct. One is the computational space and the other is the non-computational space. Computational space is where velocity potentials in the fluid are computed by the algorithms stated in 1.2.1 and 1.2.2. It is also called *Free Space* in robotic terminology, and denoted here by $\mathcal{F}$ (or C_{free}). Clearly, it is the area where robots' activity are bounded. Non-computational space is also called *Occupied Space*, denoted here by $\mathcal{B}$ (or $C_{obstacle}$). Now we define the shared boundary of $\mathcal{F}$ and $\mathcal{B}$ as the *inner boundary*, and the shared boundary of $\mathcal{F}$ and $\partial\Omega$ the *outer boundary*. They are both denoted by $\partial\mathcal{F}$. Also, $\mathcal{F}$ and $\mathcal{B}$ are subsets of Ω. Then formally, it can be described as follows:

$$\mathcal{F} \cup \mathcal{B} = \Omega \text{ and } \mathcal{F} \cap \mathcal{B} = \emptyset. \tag{27}$$

No doubt, due to the initialization in 1.2.1, the source, $S^+ \in \mathcal{F}$ and the sink, $S^- \in \mathcal{F}$.

3.1 Generic Properties

In this section, some interesting properties of $\mathcal{F}$ in the simulated domain, Ω, are analyzed. These properties reflect the important advantages by which the robot path planning can be easily and successfully conducted. Generically, it has the following properties:

Property 1. No *local minimum* exists in $\mathcal{F}$. In our context, we can define local minimum as follows: *Local minimum is any point other than S^-, the sink, that has the lowest scalar value of ϕ in its surrounding regions.*

Proof. First of all, the simulation function - Poisson's equation is a variation of Laplace's equation. The system of equations representing $\mathcal{F}$ in Ω is harmonic except for a few points which are designated as sources and sinks. Here, sources are actually the global maxima and sinks are the global minima. Because any other regions in $\mathcal{F}$ satisfy a harmonic equation, any critical points in those regions must be saddle points, since local extrema of the function are not possible there. This can be easily seen through analyzing the harmonic functions (see [CJ90]). Here is a two-dimensional version of Laplace's equation,

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0, \tag{28}$$

Consider the two curves which are the x and y cross-sections of ϕ at some point p_0. If the second derivatives of ϕ are not zero at p_0, the two curves in question must have second derivatives of opposite sign. Assuming that $\phi \in C^2$, this implies that one curve must be concave upward, and the other must be concave downward. Thus we have that either ϕ is plannar at p_0, or that there is a direction outward from p_0 in which ϕ decreases, and another in which ϕ increases. Therefore, in any region in $\mathcal{F}$ where the above harmonic equation holds, local extrema of ϕ cannot exist. Because the Neumann boundary condition is imposed on the computation, local minima cannot occur on the boundaries (i.e., $\partial\mathcal{F}$) either. Therefore, all critical points on $\partial\mathcal{F}$ are saddled points. More formally, if a function ϕ satisfies Poisson's equation on some region $\Omega \subset R^n$, then ϕ attains its minimum and maximum values only on those designated points, i.e., the sources and sinks. Therefore, if the robot reaches a saddle point, and it is not near the goal, then there must be a way out. This exit from the

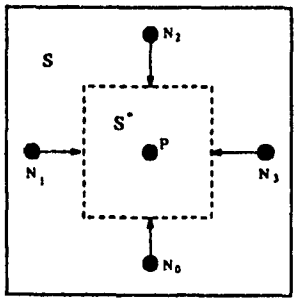


Figure 18: A local minimum centered at P

critical point may be found by performing a search in the neighborhood of the critical point.

Also, this property can be easily proved from fluid mechanics. Assume P, $P(i,j) \neq S^-$ is a local minimum. Consider a small region $S \in \mathcal{F}$ around P, where $\phi(P)$ is smaller than $\phi(N_i)$, $i = 0, 1, \ldots$ (see Fig. 18). Because P is the local minimum, from fluid mechanics, there exist flows from N_i to P. Let $S^* \subseteq S$ and $P \in S^*$, then the fluid goes into S^* from $N_0, N_1, \ldots$ However, neither fluid can flow out of the boundary of S^* for P holds the lowest potential, nor it can flow out from P for $P \neq S^-$. Therefore, the assumption contradicts the conservation law in S^*.

Also the proof can be done mathematically by contradiction. Suppose that there exists a local minimum at the node P represented by (i_0, j_0) and $P \neq S^-$ such that

$$\phi(i_0, j_0) < \phi(i, j), \; i = i_0 \pm 1 \text{ and } j = j_0 \pm 1. \tag{29}$$

i.e., The inequality "<" holds for all the four neighboring nodes: $(i_0 + 1, j_0), (i_0 - 1, j_0), (i_0, j_0 + 1)$ and $(i_0, j_0 - 1)$. From Eq. 24 and Eq. 26, We have the difference equation on (i_0, j_0),

$$a_{i_0,j_0}\phi(i_0,j_0)+b_{i_0,j_0}\phi(i_0+1,j_0)+c_{i_0,j_0}\phi(i_0-1,j_0)+d_{i_0,j_0}\phi(i_0,j_0+1)+e_{i_0,j_0}\phi(i_0,j_0-1)=g, \tag{30}$$

when

$$a_{i_0,j_0} - b_{i_0,j_0} - c_{i_0,j_0} - d_{i_0,j_0} - e_{i_0,j_0} = 0. \tag{31}$$

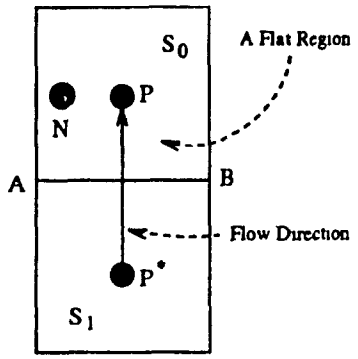


Figure 19: A flat region S_0 cannot exist if there is flow from P^* to P.

Then $g < 0$. This result contradicts the initialization in Eq. 25, in which, all right-hand sides of the system of difference equations are zero except at S^+ and S^-. Furthermore, for the difference equation on S^+, the right-hand side is greater than zero due to Eq. 25, therefore, the only node which satisfies Eq. 29 is S^-, but it is the global minimum defined in Eq. 25 and $P \neq S^-$ by the assumption. □

Let Property 1 hold. Then the following properties are its natural results, and they are easily proved.

Property 2. No *flat* regions exist in $\mathcal{F}$. In our context, a flat region is a region where all the values of ϕ are equal. Or in other words, the velocity potentials between any two points in that region are zero.

Proof. Let $S_0 \subseteq \mathcal{F}$ is a flat region and P, N are arbitrary points in S_0, then $\phi(P) = \phi(N)$. Assume S_1 is not a flat region and is connected to S_0 by $\overline{AB}$, and P^* is an arbitrary point in S_1 (see Fig. 19). If $\phi(P^*) \neq \phi(P)$, which is the usual case when S_1 is not a flat region, from Eq. 16, there must be flow between P and P^*. However, when the flow crosses $\overline{AB}$ into S_0, S_0 is no longer a flat region. Therefore, the assumption is incorrect and $\phi(P^*) = \phi(P)$. Then S_1 has to be a flat region so as to keep S_0 *flat*. In the same manner, we can prove that all regions directly or indirectly connected to S_0 through the fluid are flat regions, then we have

$$\bigcup_{i=0}^{n} S_i = \mathcal{F}. \tag{32}$$

However, $\mathcal{F}$ is not flat because of the initialization of Ω at Section 1. □

Property 3. No circular flow exists in $\mathcal{F}$.

Proof. To maintain a circular flow, a concentric force must be present, which can be measured by circulation (Γ). From [Mas90],

$$\zeta = \frac{\Gamma}{\text{Area}} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}, \tag{33}$$

where ζ is the vorticity. From Eq.16, we can get the components of q_s:

$$u = -\frac{\partial \phi}{\partial x}, \qquad v = -\frac{\partial \phi}{\partial y}, \tag{34}$$

Substitute Eq. 34 into Eq.33, we get

$$\zeta = \frac{\partial\left(-\frac{\partial \phi}{\partial y}\right)}{\partial x} - \frac{\partial\left(-\frac{\partial \phi}{\partial x}\right)}{\partial y} = -\frac{\partial^2 \phi}{\partial x \partial y} + \frac{\partial^2 \phi}{\partial x \partial y} = 0. \tag{35}$$

Eq. 35 indicates that ζ must be zero. When $\zeta = 0$, however, $\Gamma = 0$, hence no concentric force occurs. This is the property of ideal fluid. □

Because of the above three properties of the fluid model, solving the general mover's problem for a point object becomes very easy. This will be explained by first establishing a few theorems. Hence we have

Theorem 1. *If a solution for ϕ (see Eq. 24 and 26) exists in Ω, there exists a path (τ) in $\mathcal{F}$ from S^+ to S^- by simply following the decrement of the values of ϕ computed at the discrete points in Ω.*

Proof. The proof is trivial. Let Property 1, 2 and 3 hold in Ω. From Eq. 26, the velocity vector q perpendicular to $\partial\mathcal{F}$ is zero, i.e., no fluid flows onto the boundary. As we defined in Chapter 2, S^+ is the only maximum and S^- is the only minimum in Ω. All other points in $\mathcal{F}$ are saddle points. The continuity of the solutions of ϕ in $\mathcal{F}$ of Ω (see Figure 2.10) ensures a path can be found by only tracing the decrement of ϕ from S^+ successively in the neighboring points just like particles in the fluid enforced by the velocity potentials, and finally reaching S^-. □

Now we have proved three generic properties of the fluid model which is used as a navigation function in path planning in a configuration space. Also this leads to the establishment of Theorem 1. However, One thing should be mentioned here. Because the solution of Poisson's equation requires iteration techniques (for example,

the finite difference method in our work), it terminates when there is no change in ϕ on any grid node from one iteration to the next, or when the change is smaller than ϵ, a predefined threshold. Even if 64-bit floating point representations are used, this is still not sufficient to avoid flat regions in the simulated fluid area. This may cause failure to the path planning. In that case, Connolly's remedy to this problem can be used (see [CJ90]). Basically, when a flat region is the found, i.e., the numerical values of ϕ are equal, then shift all the values to the left by n bits, compare again, until differences are found.

For practical use of robot path planning and to achieve high efficiency, the solution of the system here may not necessarily indicate the exact solution of an equilibrium state of idea fluid. In our case, the iterative computation is accompanied by a path finder. Whenever a path is found, the computation terminates.

Generally, the destination of a robot bounded in Ω is assigned as the sink, S^-, and the original position of the robot can be any other points in $\mathcal{F}$. Based on Theorem 1, it is often convenient to initialize the position of the robot at S^+, and its destination at S^-.

3.2 Structural Properties

In order to understand how a robot path planning is processed, a close look at the general structure of Ω is necessary. In this section, definitions are given to basic structure of Ω, types of paths and measure of paths, and most importantly, a path configuration condition is discussed, which results in Property 5.

3.2.1 Grid Structure in Ω

From 2.2.1, Ω is embedded in a grid structure with the size of $M \times N$. Typically, M, N are powers of 2. Therefore, Ω is divided into $(M-1) \times (N-1)$ rectangles. A node is defined on the crossing of grid lines. It may be shared by up to four rectangles. There are $M \times N$ nodes in Ω. A rectangle is the smallest area element in Ω. It is called a *Unit Region.*

Definition 1. *Unit Region.* A unit region $U \subseteq \Omega$ is a rectangular area bounded by four nodes and four edges with the length in x-direction equal to h_x and in y-direction equal to h_y, where

$$h_x = \frac{1}{M-1} \text{ and } h_y = \frac{1}{N-1}. \tag{36}$$

Here the size of a square-shaped Ω is assumed to be 1 and h_x and h_y are the intervals in x and y directions respectively.

From the definition, all the unit regions are equal. If $h_x = h_y$, all the unit regions are squares.

3.2.2 Elementary Steps in U

Since U is the basic area element in Ω, a movement crossing a U is regarded as an action. The distance which an action covers is called an *elementary step*. Or more exactly, we have the following definition:

Definition 2. *Elementary Step.* An elementary step is a distance crossing a unit region U by starting and ending at different nodes in U.

From the definition, there are three kinds of elementary steps in U.

- *Horizontal elementary step* e_x, which is equal to h_x.
- *Vertical elementary step* e_y, which is equal to h_y.
- *Diagonal elementary step* e_d, which is equal to $\sqrt{h_x^2 + h_y^2}$.

Fig. 20 shows a unit region U in Ω and the elemental steps available in U. The arrows show the bi-direction in the movement.

Clearly, the three different elementary steps are not equal in length. For the convenience of measuring the length of a path, some metric should be defined. So we have

Definition 3. *Unit Length.* Unit length e is a length equal to the shortest of the three different elementary steps in a U. That is,

$$e = min(e_x, e_y, e_d). \tag{37}$$

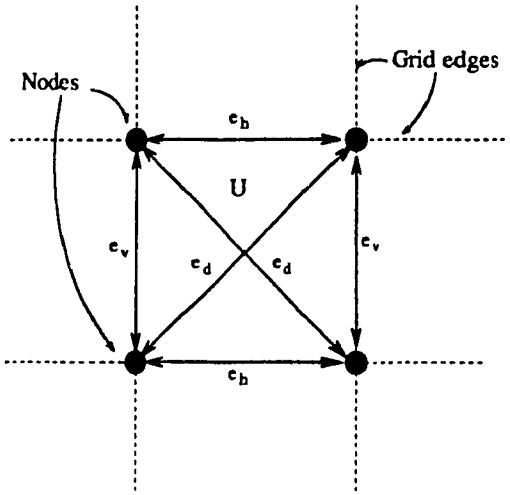


Figure 20: A unit region and its elementary steps

or generally,

$$e = min(e_x, e_y), \tag{38}$$

for $e_d = \sqrt{e_x^2 + e_y^2}$ is always greater than e_x or e_y provided none of e_x and e_y is zero.

Let L be the function returning the measurement of the length of a path in terms of e. For example, if $h_x = h_y$, then $e_x = e_y = e$, and

$$L(e_x) = L(e_y) = 1, \text{ and } L(e_d) = \sqrt{2}. \tag{39}$$

Here, L may not return an integer value because diagonal elementary steps are valid in Ω. However since e is known from the initialization of Ω, the exact length of a path is easy to be calculated. This property makes it reliable using the result of L for comparing the lengths of paths.

3.2.3 A Node and its Nearest Neighborhood (NN)

Let P be a node not on $\partial\mathcal{F}$ and S be its nearest neighborhood as shown in Fig. 21. S is composed of four unit regions, U_1, U_2, U_3 and U_4, and it is the regular area for a local path planning. P is located at the center of S and is often called the central node. It is the starting point for the next elementary step. There are eight neighboring nodes

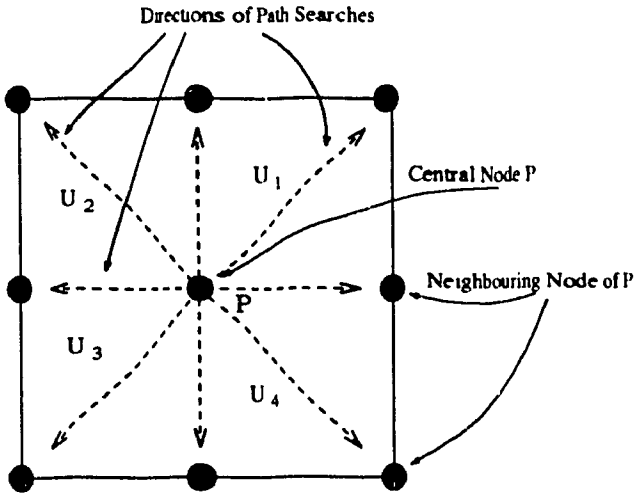


Figure 21: The nearest neighborhood of a node P: four unit regions and eight path search directions.

to P. Therefore, starting from P, there exist eight options in directions to go to its neighboring nodes by taking either a horizontal, vertical or diagonal elementary step.

3.2.4 Global Path and Local Path

Now, we can define the paths.

Definition 4. *The path.* A path τ is a route between any two distinct nodes in $\mathcal{F}$ and is a concatenation of elementary steps. i.e.,

$$\tau = e_1 \oplus e_2 \oplus ... \oplus e_n, \tag{40}$$

where $n \in \mathbf{N}$ and $n \neq 0$.

Definition 5. *The global path.* A path starting at S^+ and ending at S^- is called the *global path* (denoted by τ_{gl}). i.e.,

$$\tau_{gl} = e_1 \oplus e_2 \oplus ... \oplus e_n, \tag{41}$$

where $e_1 = \tau(S^+, P)$ and $e_n = \tau(Q, S^-)$. P is a node adjacent to S^+ and Q is a node adjacent to S^-.

Definition 6. *The local path.* A *local path* (denoted by τ_{loc}) is a path between any two distinct nodes, and

$$\tau_{loc} \subseteq \tau_{gl}. \tag{42}$$

However, a local path is chosen before the exact global path is known, Eq. 42 is valid if and only if the global path is successfully found.

Let Υ denote a set of paths whose two extremities are joined at the same nodes. We denote

$$\Upsilon = (\tau_1, \tau_2, ..., \tau_K), \text{ where } K = 0, 1, ... \tag{43}$$

Also Υ_{gl} denotes a set of global paths $\tau_{gl,i}$ starting from S^+ and ending at S^-. i.e.,

$$\Upsilon_{gl} = (\tau_{gl,1}, \tau_{gl,2}, ..., \tau_{gl,K}). \tag{44}$$

3.2.5 Path Configuration Condition

Due to the kinematic constraints of fluid, the following property is necessarily imposed on τ.

Property 4. Let $\phi(q)$ be the function returning the value of ϕ at q which is a configuration in $\mathcal{F}$ and usually refers to a node in Ω. Also let $q(0)$ and $q(N)$ be the coordinates of the starting node and ending node of $\tau(\in \Upsilon)$ and $q(m)$ be the joining node between two consecutive elemental paths, $0 < m < N-1$. We have the *path configuration condition*,

$$\phi(q(0)) > \phi(q(i)) > \phi(q(j)) > \phi(q(N)). \tag{45}$$

where, $i, j \in m$ and $i < j$. This condition indicates that any valid path must be a route in which the values of ϕ on its successive nodes is always decreasing. It is the path that a particle in the fluid can possibly travel. This condition can be easily verified by Eq. 14 and Eq. 15.

By application of this condition, we have the following advantages:

1. The characteristics of an ideal fluid under the force of gravity is correctly represented.

2. Each node can be used only once in the path. Therefore, there exists an upperbound for the number of elementary steps composed in a path which is $(M \times N - 1)$. The average length of a path is far below this upperbound.

3. Circular paths (i.e., paths starting and ending at the same node) are eliminated and the path searching process can always terminate no matter if it is successful to find the destination or not.

4. As a result, the amount of redundant path searches in the path search phase are greatly reduced, and therefore, the performance of the path searching algorithm is substantially improved.

This condition also keeps the good paths at higher priorities. For example, suppose the maximum directions of the paths at a point is L. There exist at most $L - 1$ directions at P provided $P \neq S^+$. However, for the smoothness of equipotential lines in the flow, the actual directions at each point on the path are much less. On average, only $L/2$ possible directions. Thus we have the maximum possible paths from a starting point is $(L/2)^N$, N is the number of possible steps needed to reach S^-. This is a great reduction compared to L^N.

Now we can have

Property 5. *Let Property 4 hold, then along any two paths in Υ, regardless of the number of elementary steps involved, the velocity potentials will change by the same amount.*

Or formally, let paths τ_i and $\tau_{i'}$ in Υ and $i \neq i'$, we have the total gradient G between the two ending points

$$G = \sum_{j=0}^{n-1} g_{i,j} = \sum_{j'=0}^{m-1} g_{i',j'}, \tag{46}$$

where n is the number of grid nodes connected in τ_i and m is that in τ_j, j and j' denote the jth and j'th elementary step in τ_i and τ_j respectively, and $g_{i,j}$ denotes the gradient of ϕ at step j in τ_i, i.e.,

$$g_{i,j} = |\phi(q_i(j+1)) - \phi(q_i(j))|. \tag{47}$$

The same formula as Eq. 47 for $g_{i'j'}$.

From Property 5, it is convenient to define the shortest path.

Definition 7. *The Shortest Path.* A path τ is the shortest path in Υ if and only if it has the greatest average gradient (g_{av}) among all the paths in Υ, where

$$g_{av}(\tau) = \frac{\sum g}{L(\tau)} = \frac{|\phi(q(0)) - \phi(q(N))|}{L(\tau)}. \tag{48}$$

Proof. Eq. 48 can be rewritten as

$$\sum g = g_{av}(\tau) L(\tau). \tag{49}$$

From Property 5, $\sum g$ is constant when $q(0)$ and $q(N)$ are defined. Therefore, if $g_{av}(\tau)$ returns the largest value in Υ, $L(\tau)$ must be the smallest. Then τ is composed of the least number of e's, i.e., it is the shortest.

3.3 Dynamics in Ω

In the real world, the movement of a particle in a fluid is caused by the gravity. The amount of potential force that a particle possesses is determined by its mass and altitude. Along with the declining of the altitude, the potential force decreases if the mass does not vary. In our fluid model, this force is represented integrally by two forces. Since we assume that the source of fluid (S^+) is the starting point of a robot and the sink (S^-) is its destination, the path planning is actually to search a route from S^+ to S^-, which is the same as the route taken by a particle of fluid. For ideal fluid, its density is constant and therefore, ignorable. Assume the only force exerted on the fluid is the gravity. This force can be interpreted as the velocity potential in the fluid model. Since the flow in Ω is presumed to be static, the velocity potential for a particle in all the regions of the fluid is static, too. Due to the initialization in Section 2.2.2, the system of equations for the simulation of ideal fluid is homogeneous except for the two singular points, S^+ and S^- (Eqs. 24 and 25). Therefore, values are added to the system only from the right-hand sides of the equations for S^+ and S^- and they are later propagated through the system. In this way, the solutions to Eqs. 24 and 25 simulate the ideal fluid flowing continuously into and out of Ω until

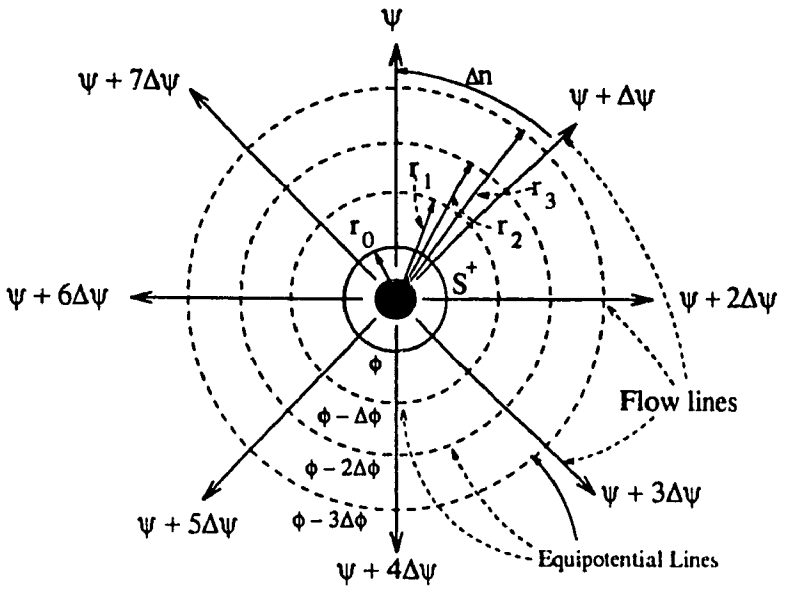


Figure 22: A radiative flow pattern at S^+

an equilibrium state in Ω is achieved. Particles of fluid come in from S^+ and go out of S^-. Hence, S^+ and S^- can be regarded as the sources of two opposite forces (like those in a magnetic field). One (S^+) is positive (or *repulsive*) and the other (S^-) is negative (or *attractive*) (see Fig. 11). The potential values obtained through fluid simulation actually represent the combination of the two forces.

In this section, the forces and flow patterns at S^+ and S^- will be analyzed. It is helpful for understanding the behavior of the fluid in a closed domain. This behavior sometimes brings irregular results for SFM (see Chapter 7). Based on this analysis, we can establish our strategies for planning better paths.

3.3.1 Radiativity at S^+

From 2.2.1, $R(S^+)$ is constant, i.e., a steady flow out of S^+ is independent of time. Assume S^+ is in an open area. The flow pattern at the small area around S^+ presents a full radiative shape (see Fig. 22). In Fig. 22, ψ is constant along the flow lines (i.e., the streamlines) and ϕ decreases along them in proportion to the increase of the distance (r) from S^+ (see Section 2.1.3). Hence the equipotential lines form concentric cycles

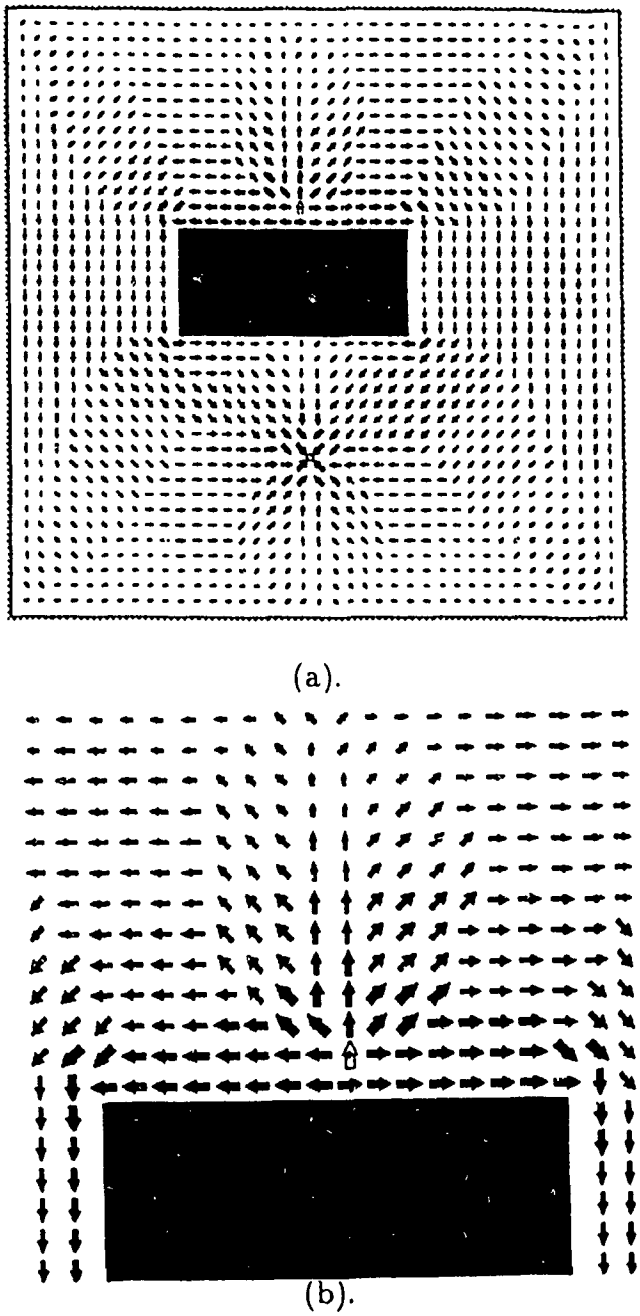
around S^+ and the velocities of the flow in all directions from S^+ are equal. However, if there is another force affecting the flow, for example, an attractive force from S^-, the equipotential curves are no longer circular. The direction in which the attractive force is larger (i.e., closer to S^-) now contains higher speed. It is reflected by the larger potential gradients in the same distance. This is how SFM works (see Fig. 32). While SFM is looking for the largest potential gradients, it actually seeks the best way to go to the origin of the attractive force.

When S^+ is defined close to the boundary, at least one side of the flow is hindered by the boundary. Because $R(S^+)$ is constant, the volume of the flow out of S^+ remains the same. Then the flow has to become more concentrated and the distance between two adjacent flow lines becomes closer, hence the average velocity is larger (Eq. 13). Also the velocity at different streamlines are no longer the same. In the direction opposite and perpendicular to the boundary, the speed is always the fastest. Fig. 23a shows a flow map of such a situation and Fig. 23b enlarges the region around the source, from which the force and the main flow direction is clearly seen. In this case, the repulsive force in one direction of S^+ is so large that the attractive force from S^- is not very affective. Hence, the main direction of flow from S^+ may not be the best direction for a path to its destination.

Fig.24 shows the results of SFM for different initial locations of S^+. Part a shows how horizontal moves of S^+ can change the natural path, and Part b gives the results of vertical moves. Paths in a(1) and b(1) are almost optimal. However, paths in a(2), a(3), b(2), b(3) are not ideal. They have the common problem: close to the boundary. We call this *locality sensibility* of S^+. It greatly affects the result of SFM. To deal with the locality sensibility, SFM should be improved. It will be discussed in detail in Chapter 7.

3.3.2 Concentricity at S^-

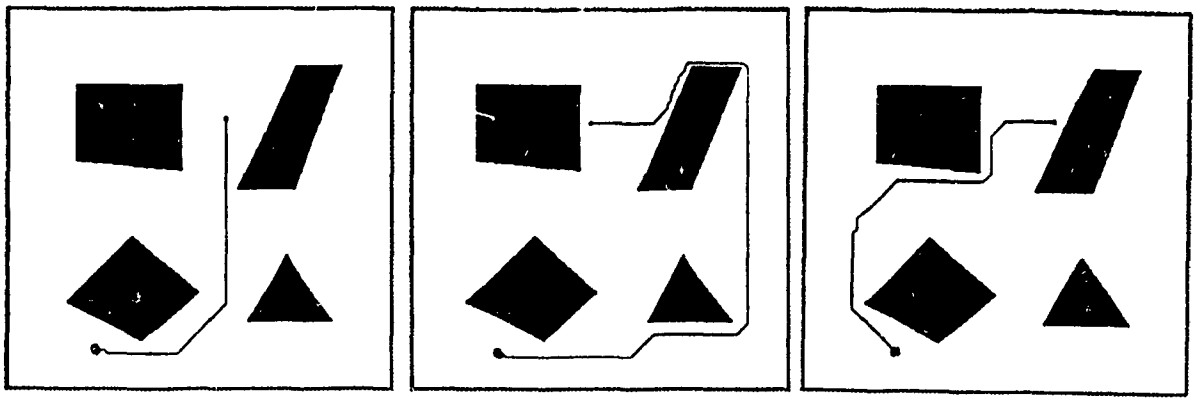
On the contrary, the flow pattern at S^- is concentric (see Fig. 25). The flow direction is outwards, hence ψ is negative and ϕ is also negative. Equipotential lines form the same kinds of cycles as those around S^+, however, the value is decreasing along the decrease of the radius r. S^-, in fact, can be regarded as a source from which a



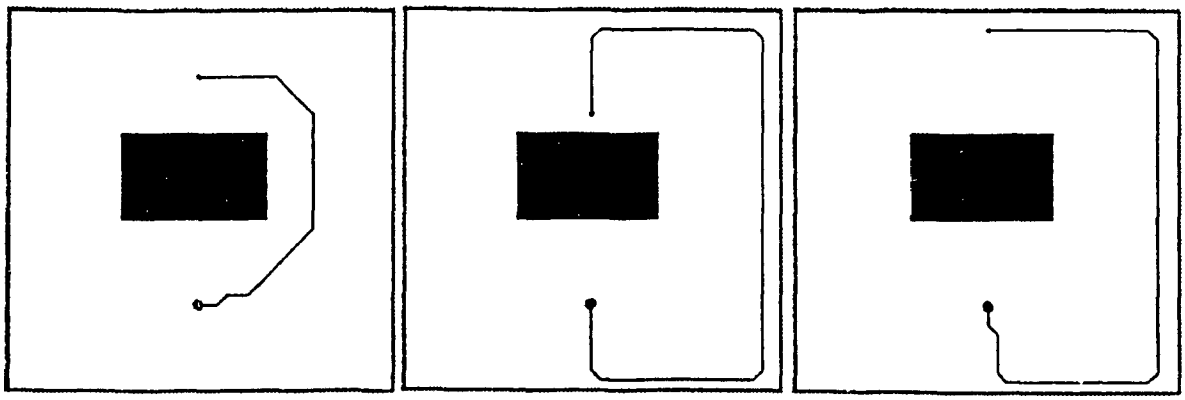
(a).

(b).

Figure 23: When the source is close to the boundary, the fastest flow direction is perpendicular to the boundary. (a) shows a flow map in Ω and (b) enlarges the region close to S^+.



a(1). a(2). a(3).



b(1). b(2). b(3).

Figure 24: Examples of natural paths. Notice the horizontal and vertical shift of the initial position may result in quite different paths.

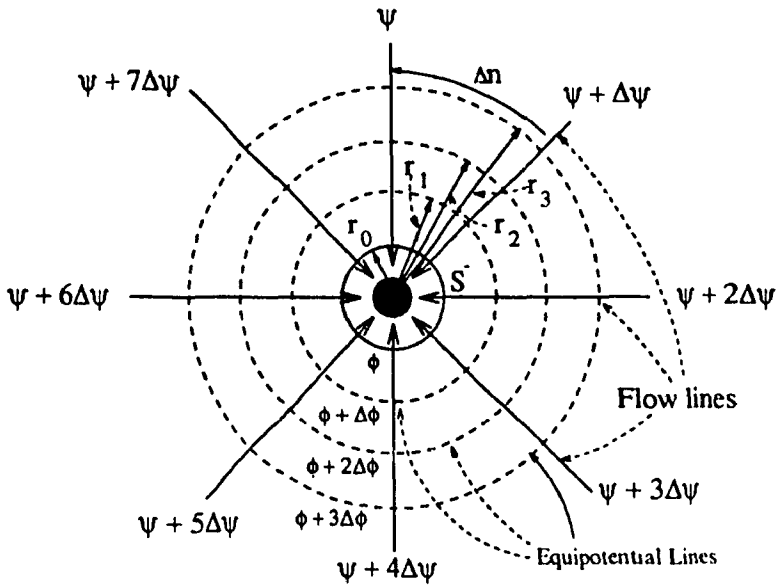


Figure 25: A concentric flow pattern at S^-

negative fluid "flows" out. Therefore, it possesses a force, a negative one compared with that from S^+. Judging from this view, the dynamics of S^-, both at the open area and by the boundary, is similar to that of S^+.

3.3.3 ϕ, the Combined Force

After fluid simulation, ϕ represents the value of the combination of the two forces from S^+ and S^- when both forces are present in Ω. Now let us discuss some features of ϕ in an obstacle-free environment.

From [Mas90],

$$\psi = K\frac{\partial \phi}{\partial \theta}, \tag{50}$$

where $K = \ln(r/r_0)$, and r_0 is an arbitrary small radius of either S^+ or S^-, where $q_r \to \pm\infty$ (see Fig. 22). Integrating both sides, we get

$$\phi = \frac{1}{2\pi K}\int \psi d\theta. \tag{51}$$

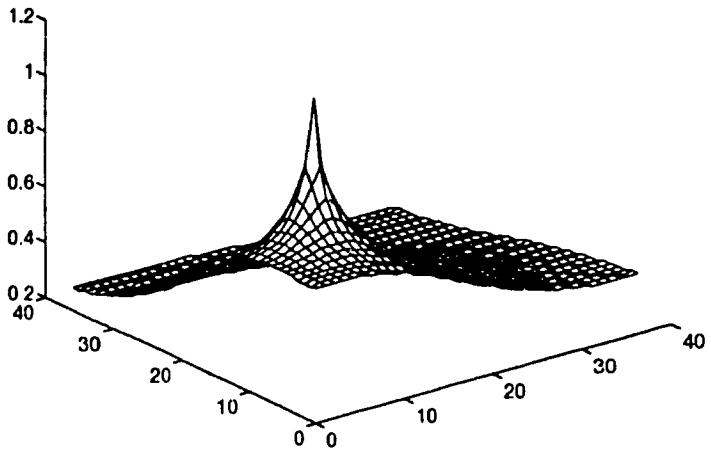


Figure 26: The distribution of ϕ when ψ is only positive. The tip of the upside-down parabolic well is S^+. ϕ tends to zero when the distance to S^+ goes to infinity.

Here, $\int \psi d\theta = R(S^+)$, which is constant, and as $r \to r_0$,

$$K = \ln\left(\frac{r}{r_0}\right) = \ln(1) = 0. \tag{52}$$

Clearly from Eq. 51, when r_0 is the radius of a very small centric point of S^+, $\int \psi d\theta$ is positive, then

$$\lim_{r \to r_0} \phi = \lim_{r \to r_0} \frac{1}{2\pi K} \int \psi d\theta = +\infty. \tag{53}$$

On the other hand, when r_0 is the radius of S^-, $\int \psi d\theta$ is negative, then

$$\lim_{r \to r_0} \phi = \lim_{r \to r_0} \frac{1}{2\pi K} \int \psi d\theta = -\infty. \tag{54}$$

Also in both cases, as $r \to \infty$, $\phi \to 0$. The following figures of 3D meshes generated from the values of velocity potentials show the distribution of ϕ in Ω in three different cases: (1). Only S^+, i.e., ψ is positive (Fig. 26). (2). Only S^-, i.e., ψ is negative (Fig. 27). (3). Both are present (Fig. 28). Of course, in the first two cases, the flow can never be static.

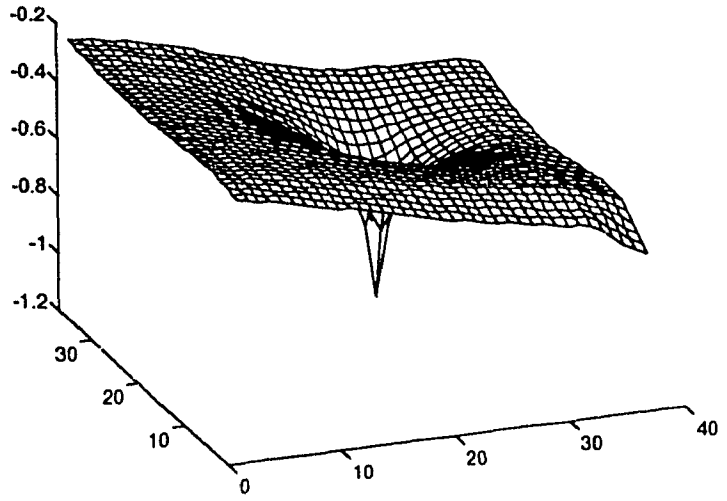


Figure 27: The distribution of ϕ when ψ is only negative.The tip of the parabolic well is S^-. ϕ tends to zero when the distance to S^- goes to infinity.

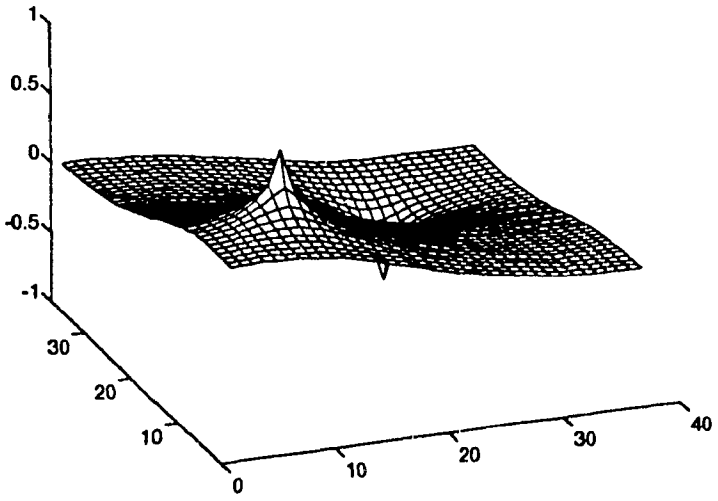


Figure 28: There are the two parabolic wells in opposite direction. ϕ is the combination of the two forces from S^+ and S^-.

Chapter 4

Path Generation

4.1 Steepest Falling Method (SFM)

By the path configuration condition defined in Eq. 45, the number of searches for the shortest path is greatly reduced. However, searching every valid path still needs extensive work and it is not economical. On the other hand, the shortest path may not be a good one, because it often goes along the boundaries, which brings unnecessary difficulties for robot navigation (see [Lat91]). Therefore, in [BL89], one of their path searching approaches is actually trying to maximize the distance between the path of the robot and the boundary so as to reduce the danger of collision. It is justifiable to put collision avoidance at the first place of path planning. In our fluid model, the solution of collision avoidance is integrated in the fluid simulation in Ω. A collision-free path can be obtained by simply following the natural flow of fluid, i.e., following the main streamline of the flow. Based on Theorem 1 and 2, a simple algorithm in a local path finding is proposed, which is similar to the *fast descending method* in PDE (see [Van83]) and to the *depth first planning* in [Lat91].

4.1.1 Algorithm and its Efficiency

Because ϕ is precomputed at all the nodes in $\mathcal{F}$ and on $\partial\mathcal{F}$, the algorithm of SFM becomes very simple. Let S be a nearest neighborhood (NN) centered at P and

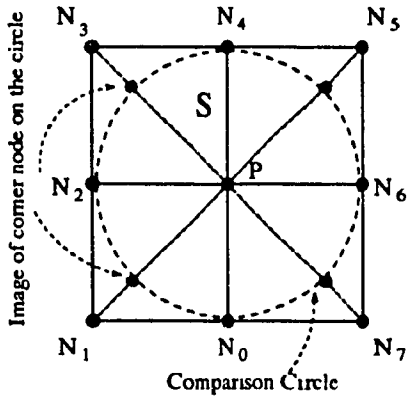


Figure 29: A nearest neighborhood of a node P for the local path planning.

$\mathcal{N}_1$, $\mathcal{N}_2$, ..., $\mathcal{N}_n$ are neighbors of P (see Fig. 29). The local path from P is found by connecting P to the node on ∂S with which the greatest potential gradient is obtained. Or in another word, by taking the direction in which the steepest falling of the fluid occurs (see Fig. 30). Hence, this technique is called *the Steepest Falling Method* or SFM.

Let N_{next} be the node of selection, then

$$\mathcal{N}_{next} \leftarrow D(max_{i=1}^{n}(g(P,\mathcal{N}_i)), \tag{55}$$

where D is a function returning the local path to the next node in S, and

$$g(P,\mathcal{N}_i) = \xi * (\phi(P(x,y)) - \phi(N_i(x',y')), \tag{56}$$

and

$$\xi = \begin{cases} 1 & \text{if } x' = x \text{ or } y' = y, \\ \frac{\sqrt{2}}{2}, & \text{otherwise.} \end{cases} \tag{57}$$

Here, the distance factor ξ ensures the comparison is taken in the same length (as the radius in a circle).

This algorithm is composed of three steps as shown in Fig. 31. The termination condition is $\mathcal{N}_{next} = S^-$. If there is flow in Ω, which is the precondition for invoking SFM, this condition can always be reached, i.e., the algorithm can always terminate.

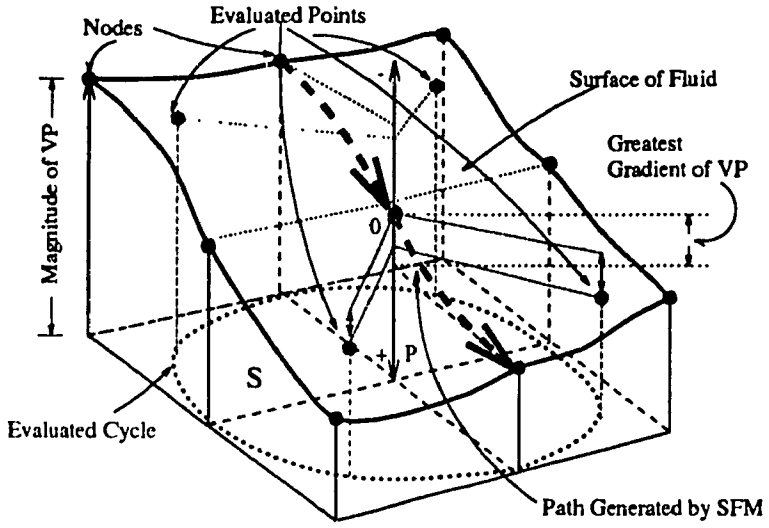


Figure 30: A 3D sample of SFM working on local path planning in the nearest neighborhood of a node P. Here. VP means velocity potential.

1. *Starting from P, obtain the vector of potential gradients, $g(P,\mathcal{N}_0), g(P,\mathcal{N}_1), ..., g(P,\mathcal{N}_j)$, j is the maximum number of available nodes in NN of P (the default is 8).*

2. *Compare the gradients obtained from 1 and select $\mathcal{N}_{next}$ such that*

$$g(P,\mathcal{N}_{next}) = max\,(g(P,\mathcal{N}_0), g(P,\mathcal{N}_1), ..., g(P,\mathcal{N}_j))\,.$$

3. *If $\mathcal{N}_{next} = S^-$, terminate the process, else $P = \mathcal{N}_{next}$, return to 1.*

Figure 31: The algorithm of SFM.

From the initialization of Ω in 2.2, the maximum number of elemental steps one path can contain is $M * N - 1$ because of no repeated traversal of the same node in a path constrained by Eq. 45. Let $\mathcal{K}$ be the total nodes joined in a path τ. For constructing τ, SFM invokes $\mathcal{K} - 1$ times of gradient calculations and comparisons (see Step 1 and Step 2 in Fig. 31). However, the time used for each invocation is constant because the number of nodes in a NN is fixed. Let t be the time used for each invocation. SFM's time efficiency (T) increases only linearly with $\mathcal{K}$, i.e.,

$$T = t * \mathcal{K}. \tag{58}$$

In the worst case, $\mathcal{K} = M * N - 1$. Hence we have the upper bound for T,

$$T = t * (M * N - 1). \tag{59}$$

In practice, the actual number of nodes joined by a path is far below the maximum because of the characteristics of ideal fluid.

4.1.2 Paths Generated by SFM

Because the path generated by SFM represents the natural movement of a particle in ideal fluid, we call it a *natural path*. Or formally, we have the definition:

Definition 8. *Natural Path.* A natural path τ_{nat} is a subset of τ in which every elementary step is generated by SFM, or in other words, it always takes the direction of the main streamline (which contains the steepest flow) in a concerned region. In our later explanation, the term, *streamline*, exactly means the natural path.

When a solution in Ω is obtained, the natural path τ_{nat} from S^+ to S^- is defined and unique in Ω (refer to Fig. 11 and Fig. 12). The paths shown in Fig. 32 are natural paths. In each figure of Fig. 32, there are two black points. The smaller one stands for S^+, or the starting point of a point robot, and the bigger one for S^-, or the destination of the robot. The solid line connecting the two points shows the generated path by SFM. Obviously, all the paths are or close to the shortest (optimal) one in their own environments. Also note that the lines are not smoothed in order to accurately trace the functionality of the algorithm on the grid of Ω.

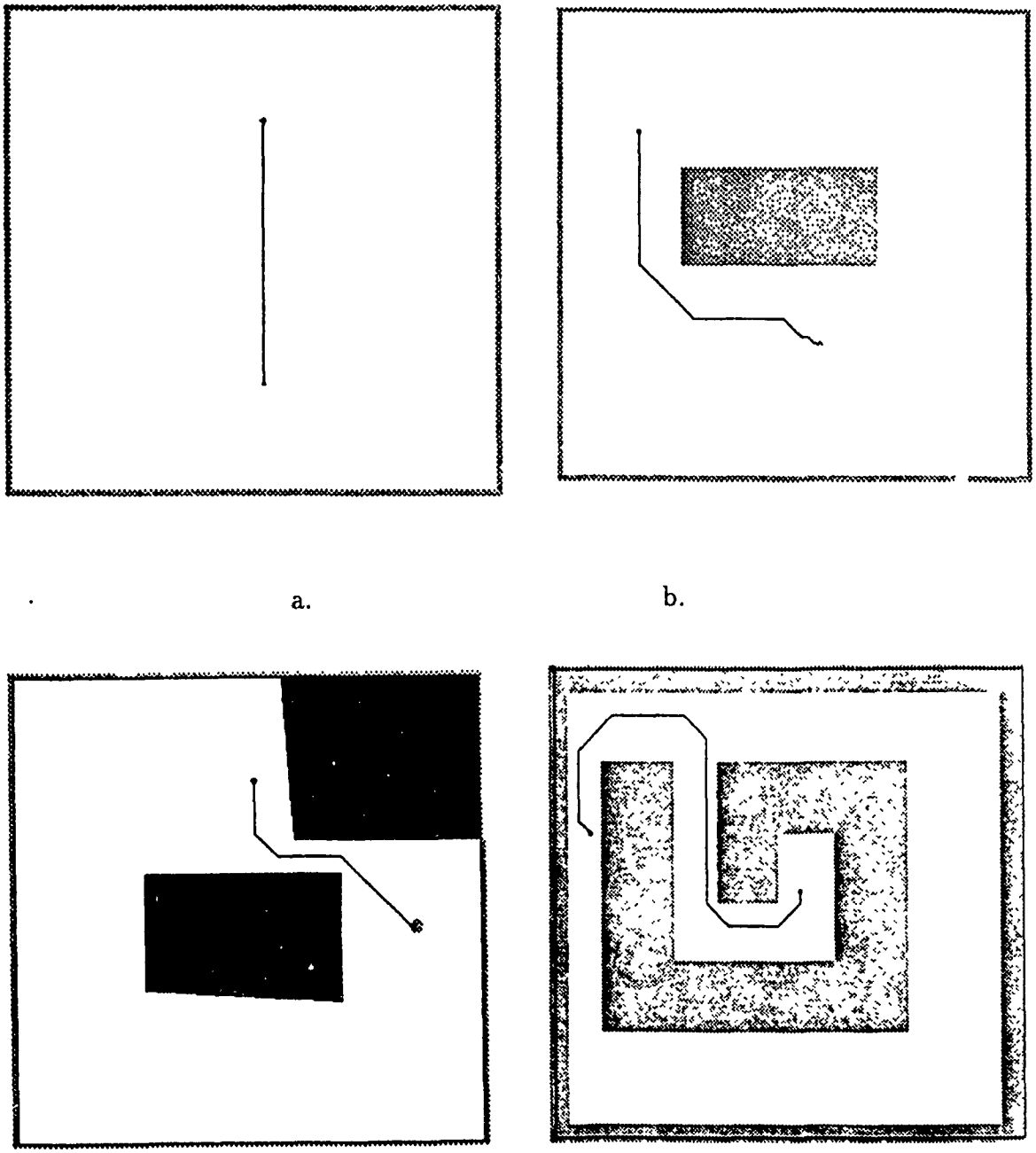


Figure 32: Examples of the path generated by SFM in different environments.

In general, natural paths are good (here "good" indicates short and proper). However, because of the locality sensibility of S^+, which is the initial position of a robot, the global path by SFM may not always be satisfactory (see Section 6.1). The forcefulness of the repulsive dominates the region around S^+, which makes the attractive force from S^- ineffective. Therefore, the local path planning at regions close to S^+ is dependent largely on the main direction of the repulsive, which may not be consistent to that of the attractive. We know that the velocity potential (ϕ) computed at a node represents the combination of the two forces but we do not know exactly what percentages they possess respectively. From Figs. 26 and 27, we can see that the repulsive force reduces drastically along the increase of the distance from S^+, and on the other hand, the attractive force becomes larger at the regions closer to S^-. This fact indicates that if we choose some nodes farther from S^+ for comparison, the result of SFM would be more reliable to show the shorter distance from S^-. Therefore, SFM can be supplemented by extending the concerned distance from S^+ or any other center point so as to compare potential gradients in a larger area. This technique is called *Area Expansion.*

In the following sections, two techniques of path improvement based on area expansion are proposed. The first is simple, fast and more heuristic. It adds direction correction technique to SFM in order to adjust the direction and shorten the path. This technique first takes two best candidate directions, compares the velocity potentials on the nodes in those directions until a significant difference is achieved, then chooses the winner. The second is more exact and reliable to find a best path but needs more time in execution. Because of the radiativity of fluid at S^+, we can find an alternative path by just moving the initial point of the robot a little away from S^+ and generate a natural path from it, joining the point to S^+ with a shortest line (which may not be a straight line when encountered by obstacles). This moving can be completed through image projection of the robot. Clearly, the projection is multi-directional. Hence, we can get a set of quite different natural paths starting from those nodes around S^+. When r increases, i.e., the concerned region around S^+ expands, the number of available nodes also increases. One noticeable result is that the set shows a full coverage of the paths in all related directions from S^+. This

technique is still valid in any other regions in $\mathcal{F}$ except those close to S^-. The difference is that, at the source, the searching area may be circular, i.e., in 360 degrees while at other places, it is not. This is because of the path configuration condition (Eq. 45). Theoretically, we can get as many points as we like on the boundary of a region. However, since Ω is embedded in a grid and only those grid nodes are under our consideration. In the region around S^+, for example, a NN, there are only eight boundary nodes. If we obtain a path starting from each of the nodes, we have a set which contains the possible paths in eight main directions from S^+, and the interval of selection is 45 degrees. Obviously, the best path selected from the set has very high reliability. This technique is probabilistically complete when the circle can be expanded endlessly, i.e., the interval for search tends to be infinitesimal. if one direction of the flow is hindered by the boundary, the others must have larger momentum (or velocity) than usual, provided R remains constant. If at a node P on τ, we have two natural subpaths τ_1 and τ_2, which lead to $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively, and $\bar{\phi}(\mathcal{N}_1) < \bar{\phi}(\mathcal{N}_2)$ i.e., $\bar{g}(P,\mathcal{N}_1) > \bar{g}(P,\mathcal{N}_2)$. SFM takes $\mathcal{N}_1$ as its successive step. However, there is no guarantee that the average gradient decrement of velocity potentials in τ_1 is greater than that in τ_2, because $\mathcal{N}_1$ is chosen locally, disregarding the global information. Therefore, techniques of path improvement are necessary to guarantee the achievement of a better path.

4.2 Flow Direction Correction (FDC)

The result from Eq. 55 to 57 may be improved by adding global direction reference to SFM. However, it is expensive to keep track of the distances and directions geometrically from S^- for each movement. According to Eq. 14, ϕ decreases along the streamlines proportionally with the increase of δs provided q_s is constant. However, q_s is not constant because of the spatial variety of flow channels in Ω. Hence the equi-potential lines around the sources occur as irregular curves (see Fig. 12). SFM plans the local path based on the numerical information in a NN. When the gradient difference g is too small (e.g., between the first largest and the second largest), it may not be significant in decision-making. However, it is an important signal to re-assess

the flow direction. We have added to SFM a heuristic algorithm called *Flow Direction Correction* (FDC), which is simply to stretch out the two or more possible candidate nodes in their own directions, and compare their velocity gradients again until certain condition is satisfied. This algorithm is effective because of the continuity of velocity potentials in Ω by Poisson's equation. The stretching-out technique allows the planning in a larger area in which the decline of ϕ towards S^- can be more accurately sensed out. By keeping track of the decreasing ratio of the gradients, we can predict the right direction which leads to a much better global path.

4.2.1 Algorithm

Let g_1 and g_2 be the first two greatest gradients obtained between P and its neighbors $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively and $g_1 > g_2$, $\mathcal{N}_1$ will be chosen provided

$$g_1 - g_2 \geq \epsilon, \text{ where } \epsilon = \frac{|\phi(P)|}{\mu M}. \tag{60}$$

ϵ is a threshold of significant gradient difference and is related to the size of the grid in Ω and the value of the potential computed at P. Clearly, ϵ varies only with the variation of ϕ because when Ω is defined, M is a constant. μ is an adjustable coefficient. Increasing μ, reduces ϵ and hence decreases the frequency of invoking FDC. Empirically, μ is chosen from 1 to 5.

If Eq. 60 is not satisfied, FDC is invoked. Starting from $\mathcal{N}_i$, $i = 1, 2$ in above case, one step is taken in the same direction as it comes from $P(i,j)$,

$$\mathcal{N}_{next}(x',y') \leftarrow \mathcal{N}(x,y), \tag{61}$$

where

$$x' = \begin{cases} x \pm 1 & \text{if } x \neq i, \\ x & \text{if } x = i, \end{cases} \quad y' = \begin{cases} y \pm 1 & \text{if } y \neq j, \\ y & \text{if } y = j. \end{cases} \tag{62}$$

Eq. 61 and 62 are repeated until the condition in Eq. 60 is satisfied. However, the maximum number of stretches, d, is controlled to prevent missing S^-. Naturally, d should be proportional to M, the size of grid in Ω, and is defined as $\lfloor M/16 \rfloor$. The scenario of FDC is shown in Fig. 33.

1. *Starting from P, find $\mathcal{N}_i$ and $\mathcal{N}_j$ by SFM where*

$$g(P,\mathcal{N}_i) > g(P,\mathcal{N}_j) > max^{n}_{k=0 \wedge k \neq i \wedge k \neq j}(g(P,\mathcal{N}_k)), \qquad (63)$$

where $n+1$ is the maximum available nodes in NN of P.

2. *If $|g(P,\mathcal{N}_i) - g(P,\mathcal{N}_j)| \geq \epsilon$, then*

$$\mathcal{N}_{next} = D(max(g(P,\mathcal{N}_i), g(P,\mathcal{N}_j)), \qquad (64)$$

go to 4, else assign $d = 1$ (only once) and go to 3.

3 *Stretch $\mathcal{N}_i$ to $\mathcal{N}_i^d$ and $\mathcal{N}_j$ to $\mathcal{N}_j^d$ by Eq. 61 and 62, and increase d by one, return to 2.*

4. *If $\mathcal{N}_{next} = S^-$, then terminate, else $P = \mathcal{N}_{next}$, return to 1.*

Figure 33: The algorithm of SFM with FDC.

4.2.2 Results of Experiments

We have used FDC to deal with many close-to-boundary cases and the results are satisfactory. The improved ratio compared to SFM will be explained in 7.3. Here are several examples of our experiments. Fig. 34 shows paths found by SFM with FDC in a close-to-boundary environment. Fig. 35 gives the flow map of Fig. 34. From the map, FDC technique can be seen from those points where the path is switched away from SFM. Figs. 36 and 37 give two comparable examples with the cases shown in Fig. 24. The other two figures (Fig. 38 and 39) show paths generated by FDC in a more complicated situation, and the paths are smoothed by B-splines. Finally, Fig. 40 shows one goal position with three global paths generated. It proves the computed function can be reused to a very satisfactory results.

4.2.3 Efficiency of the Algorithm

The parameter d shown in Fig. 33 is limited to a very small number. For example, for a Ω of 128x128, d should be limited to 8. If the condition of Eq. 60 is still not satisfied

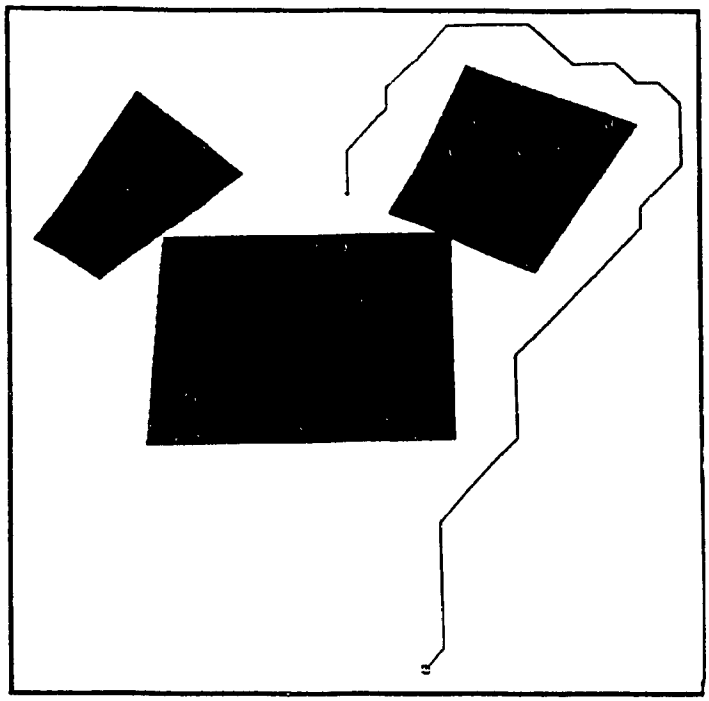


Figure 34: Examples of paths generated by SFM with FDC (1)

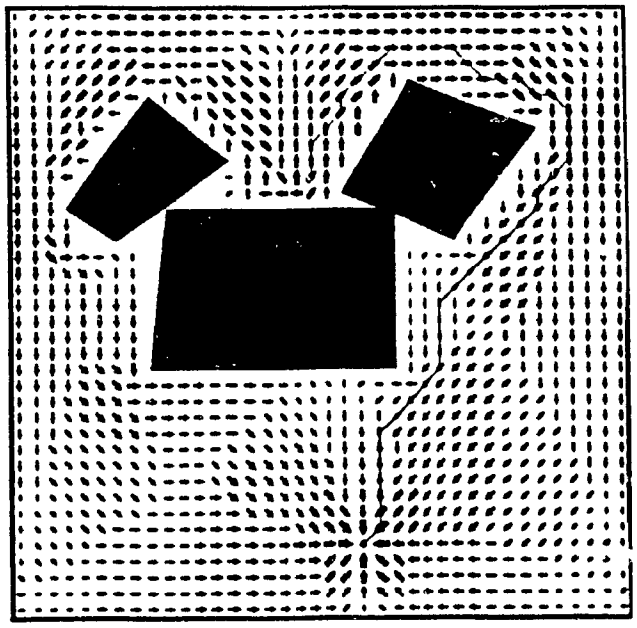


Figure 35: The flow map in the above figure with the generated path. The switching points of the path is clearly shown.

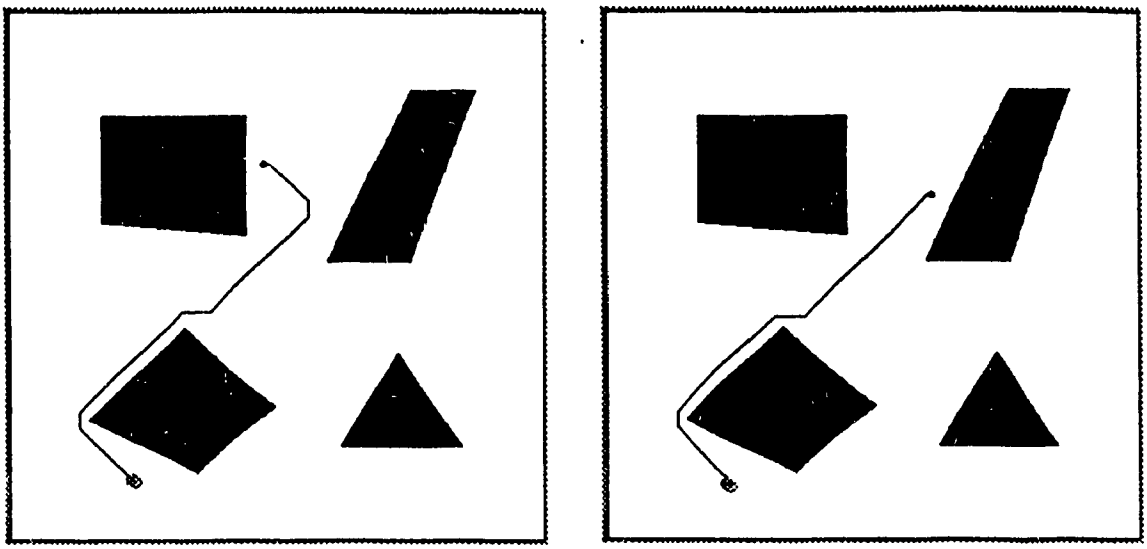


Figure 36: Examples of paths generated by SFM with FDC (2)

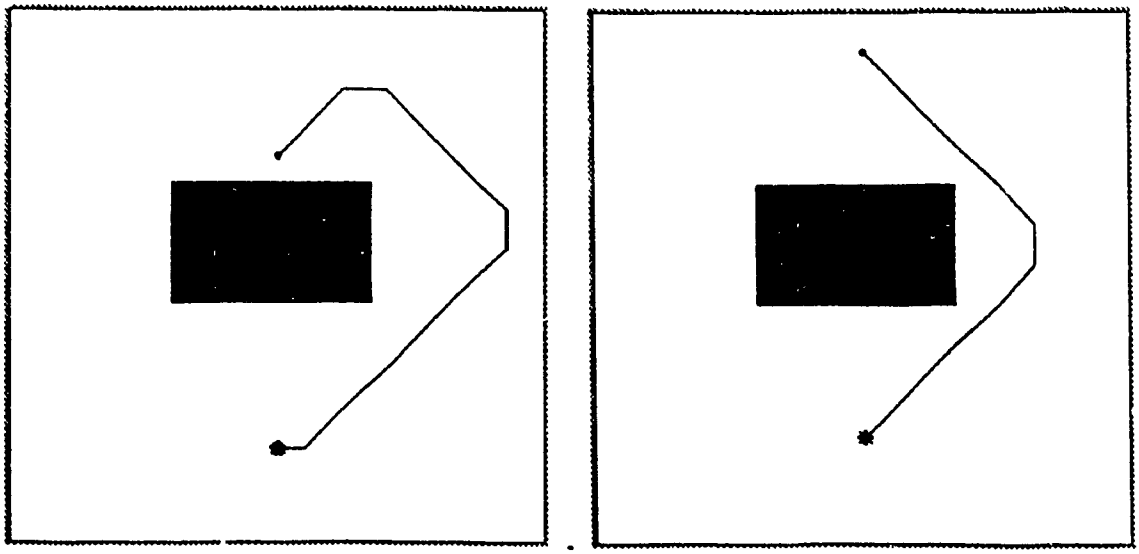


Figure 37: Examples of paths generated by SFM with FDC (3)

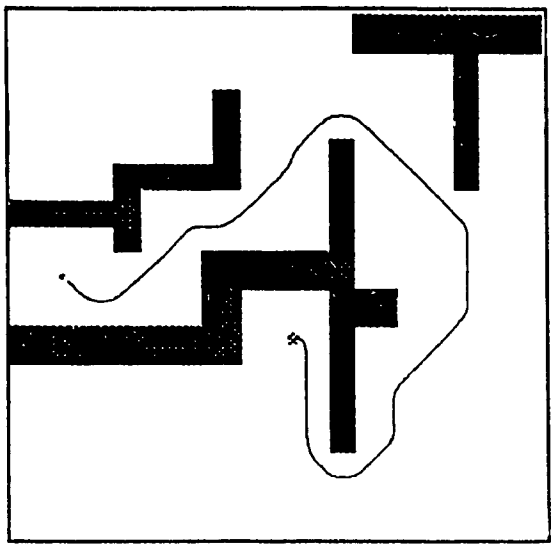


Figure 38: Example of paths generated by SFM with FDC (4). Simulation takes 8.54 seconds and path planning takes less than 0.01 second.

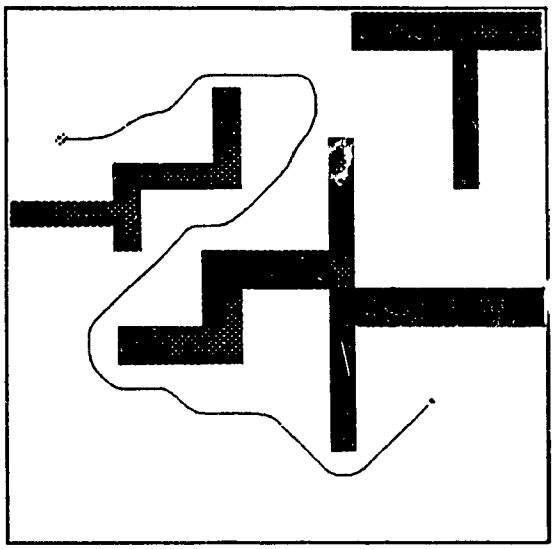


Figure 39: Example of paths generated by SFM with FDC (5). Simulation takes 10.07 seconds and path planning takes less than 0.01 second.

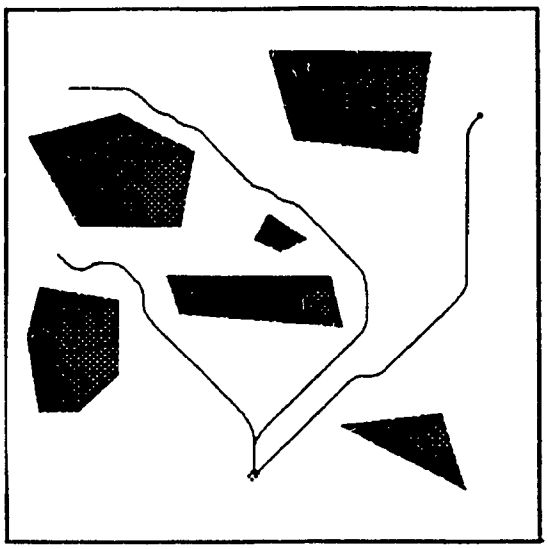


Figure 40: Examples of paths generated by SFM with FDC (6). Three paths are generated on the same simulation result.

when $d \geq 8$, SFM is resumed. This may lead to a longer path, but no control for stretching could result in missing the destination, S^-. On the other hand, controlled stretching limits the time spent on FDC so that the combined algorithm generates satisfactory paths while its performance does not degenerate.

Let t_{fdc} be the time spent on one comparison, then $d * t_{fdc}$ is the approximate time for one invocation of FDC. In the worst case, FDC is invoked at each step, and the time complexity is:

$$T = 2 * (\mathcal{K} - 1) * (d\, t_{fdc}) = 2d\, t_{fdc}\, \mathcal{K} - 2d\, t_{fdc} \approx O(\mathcal{K}). \tag{65}$$

$\mathcal{K}$ in Eq. 65 is the number of nodes joined by a path, bounded by $M * N$, and is the only varying factor.

4.3 Area Expansion and Image Projection (AEIP)

Another heuristic algorithm for path improvement called *Area Expansion and Image Projection* (AEIP) is proposed in this section. This algorithm first projects a configuration onto the boundary of a region around the configuration, and then from the

projected images, generates their natural paths separately and compares those paths so as to find the best one.

This algorithm takes the advantage of the finiteness of Υ in Ω, computes the preference ratio of the selected paths from Υ and chooses the path with the highest preference ratio. It provides an ensured way to find the best path in Υ.

4.3.1 Finiteness of Paths in Ω

No doubt, only when Υ is a set of finite elements, can we examine all the elements in it and find the best one.

Property 6. Υ is a finite set in Ω.

Proof. From 4.1, Ω is bounded and embedded in a grid of $M \times M$. If M is not infinite, so is the number of nodes (N) in Ω. In our method, the simulation of fluid is done by computing the values of the velocity potentials on those nodes and the path planning algorithms are also based on comparisons of the values on them. According to Eq. 45, ϕ along the path must be decreasing, and all the paths will converge to S^-, the destination, from the generic feature of Ω each path contains only a finite number of elementary steps. Finally, the number of the paths in Υ is decided by the radiativity of S^+ (see 6.1.1). It is clear that for Ω bounded and embedded in a grid, the number of streamlines radiated from S^+ is finite, and bounded by a factor of M. □

The finiteness of Υ indicates that it is possible, in the worst case, to compare all the subset of Υ in order to find the best path. In the following path improvement algorithm, SFM is always used as the basic path searching technique, or in other words, the comparisons are based on the natural paths generated by SFM.

4.3.2 Image Projection

First, we have the following definition for image:

Definition 9. *Image.* An image (I) is a copy of configuration $q(\in \mathcal{F})$ which is obtained through a parallel projection onto a boundary node of an expanded area $E(\in \mathcal{F})$ around q.

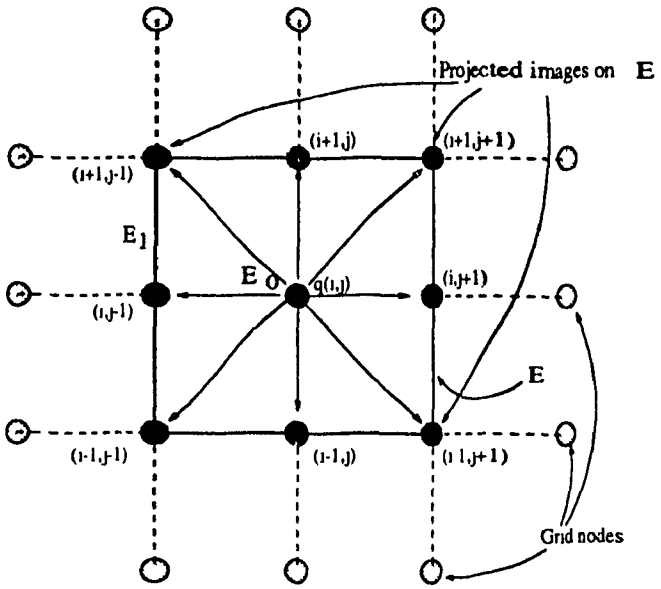


Figure 41: The preliminary area expansion and the eight directions of image projection of q. Here q is a point.

The boundary of E is denoted by ∂E. If q is a point, then the image of projection is also a point. In our context, ∂E always coincides with the horizontal or vertical grid lines in Ω.

Fig. 41 shows the basic method of projection of a configuration $q(i,j)$ in eight directions. The center for expansion is defined as E_0. Hence in E_0, there is only one node. E_1 is the first expansion around q. There are eight possible images of q on ∂E_1. They are, in clockwise, $I(i-1,j), I(i-1,j-1), I(i,j-1)$, $I(i+1,j-1), I(i+1,j), I(i+1,j+1)$, $I(i,j+1), I(i-1,j+1)$. A configuration can have multiple layers of images on the boundaries of successive area expansions $E_1, E_2, ...$, which may be generated recursively. For example, in Fig. 42, there are eight images $I_0, ..., I_7$ on ∂E_1, and total 16 images on ∂E_2. In the projection, only the images on the outer boundaries of the expansion are counted and the duplicates are discarded. Clearly, the image projection is one to many. For example, in Fig. 42, $I(2)$ on ∂E_1 has sub-images $I(1,4)$, $I(1,5)$, $I(2,6)$ on ∂E_2.

AEIP is composed of two phases. Phase one is for starting direction determination from the initial position of a robot, i.e., S^+. Phase two is for the on-path processing

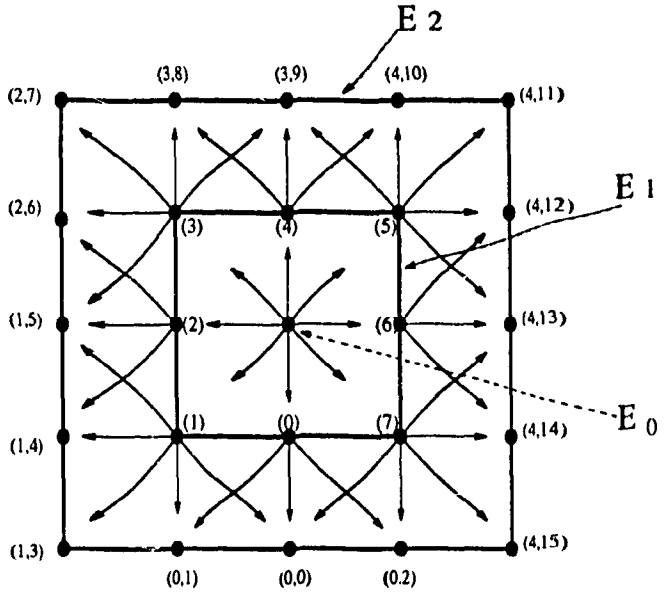


Figure 42: The images on ∂E_1 and ∂E_2. The arrows show the recursive image projection for the first two expansions. The first number in the bracket is the number of the previous expansion, and the second is the order of images generated in the current level.

until the goal (S^-) is reached.

4.3.3 AEIP at the Source

Originally, S^+ is defined as a line source which appears as a point in two-dimensional domain Ω and the initial position of a robot is assigned to it. Hence S^+ is always the starting point of a global path. Because of the radiative feature at S^+, in which direction the first step is going to take is crucial for path planning and therefore needs special consideration. On the other hand, this feature enables comparisons of the available natural paths in all directions by simply expanding the circle around S^+.

Starting from those images of S^+ on E_i, $i = 1, 2, ...$ separately, we can get, by SFM, a number of natural paths joining S^- at their ends. By comparing those paths using the length function L, we can choose one which returns the smallest value. If the result in E_i is not satisfied, we can do expansion again and examine more images in E_{i+1}.

We have this relation between the successive expanded areas. as well as far from the boundaries of obstacles if it is available.

To ensure the success of this technique, the following condition should be satisfied.

Projection Condition 1. Any expanded region, E_n, must be a subset of $\mathcal{F}$, and hence all projected images are in $\mathcal{F}$, or possibly on $\partial\mathcal{F}$.

This condition puts constraint on the expansion activity and ensure the algorithm to work in a valid environment. From this condition, E is only a finite set of expansions with $n <= M - 1$. And

$$E_i \subseteq E_{i+1}. \tag{66}$$

Therefore,

$$\Upsilon_{E_i} \subseteq \Upsilon_{E_{i+1}}. \tag{67}$$

Eq. 67 indicates that the paths found in E_i can also be found in E_{i+1}. In a special and rare case, an image on ∂E_i is exactly S^-, then the optimal path is found.

Let E_n be the nth expansion from S^+, $E_n \in \mathcal{F}$, then we have a vector of images I_n on ∂E_n. If the projection is complete, i.e., no boundary has been encountered

since the expansion, the number of images in I_n is $8 * n$. By increasing the number of layers in E, we increase the images of S^+, and therefore obtain more natural paths for comparison.

At E_n, a global path, $\tau_n^i (\in \Upsilon_{E_n})$ is composed of two segments. $\tau(S^+, I_n^i)$ is a shortest path established during the expansion process and $\tau(I_n^i, S^-)$ is a natural path. Therefore, the length of τ_n^i is,

$$L(\tau_n^i) = L(\tau(S^+, I_n^i)) + L(\tau(I_n^i, S^-)). \tag{68}$$

And the shortest path in E_n,

$$Short(E_n) = min_{E_n}(L(\tau_1), L(\tau_2), ..., L(\tau_k)), \tag{69}$$

where $k = 8 * n$. The expansion will be terminated when

$$Short(E_n) \geq Short(E_{n-1}) - L(e). \tag{70}$$

Here, $L(e) = 1$ (see Chapter 3), because we assume that each area expansion is to expand to E_n from E_{n-1} by stretching the distance from S^+ with the length of an elemental step. This requirement guarantees all the images will fall on the grid nodes. One special case for E_n is when $n = 0$. Then $E_0 = S^+$ and $Short(E_0) = L(\tau(S^+, S^-))$, which is the natural global path. If Eq. 70 is true, it is no need to go further since E_{n-1} already contains the shortest path.

In order to achieve high flexibility for robot navigation, the favorable path should not touch the boundaries of $\mathcal{B}$ or, at least, the total length of those segments in the path which touch $\partial\mathcal{B}$ should be reduced to as short as possible. For this reason, the boundary-touching distance is under concern in the path selection. In practice, a ratio of preference (Rat) for each path is calculated. Let e_i be any of elementary steps in τ_n^i which contact $\partial\mathcal{B}$ directly, then we have the following formula,

$$Rat_n^i = 0.5 * \frac{Short(E_n)}{L(\tau_n^i)} + \left(0.5 - \frac{\sum_{\tau_n^i} L(e_i)}{2L(\tau_n^i)}\right). \tag{71}$$

and the preferred path in E_n

$$\tau_{pref} \leftarrow min_{i=0}^{8n-1}(Rat_n^i). \tag{72}$$

1. *Obtain the natural path* $\tau(S^+, S^-)$ *in* E_0 *by SFM.*
2. *Assign* $n = 1$ *(only once), project* S^+ *on* ∂E_n *and obtain a set of images* $I_0, I_1, ..., I_{8n-1}$.
3. *Obtain all the natural paths* $\tau_0, ..., \tau_{8n-1}$ *for the images on* E_n, *and compute* $Short(E_n)$.
4. *Compare* $Short(E_n)$ *with* $Short(E_{n-1})$ *by Eq. 70. If it is true, calculate* τ_{pref} *on* E_n *and terminate. Otherwise, increase* n *by 1 and go back to Step 2.*

Figure 43: The algorithm of AEIP at the source.

Clearly, Eq. 71 favorites a path with 0 or fewer e_b's.

This algorithm is shown in Fig. 43. Fig. 44 shows the natural paths starting from the images on E_2.

4.3.4 AEIP on the Path

The algorithm for the source can also be employed for on-path improvement with the following extra condition:

Projection Condition 2. The image of Q will be projected to a node P on $\partial E_n (\subset \mathcal{F})$ in nth area expansion, provided $\phi(P) < \phi(Q)$.

In Projection Condition 2, Q is a configuration anywhere in $\mathcal{F}$ other than at S^+ and S^-. This condition is consistent with the path configuration condition stated in Chapter 3. ∂E_n has the same meaning as that in 4.2.2, but the images only occupy part of its nodes. This condition reduces the number of images projected from Q. Also n is controlled as in 4.2.2.

Assume after the execution of the algorithm in Section 4.2.2, Q is chosen as the favored point, from which a best natural path can be produced. Now the subpath $\tau_s(S^+, Q)$ is decided, which is the shortest from S^+ to Q. The AEIP algorithm on the path is invoked for further path planning from Q. There are still four steps in this algorithm which are stated in Fig. 45.

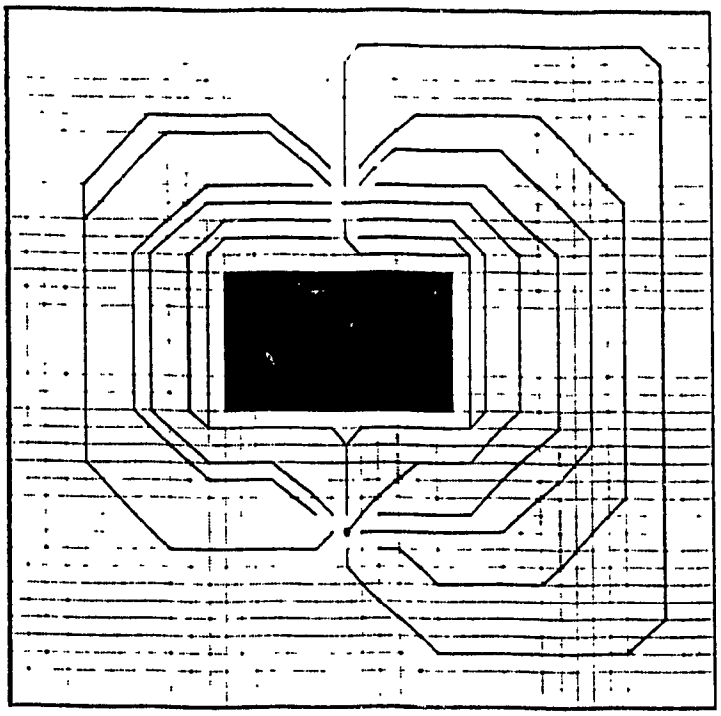


Figure 44: Natural paths from the projected images of S^+ on E_2.

1. *Calculate* $L(\tau(Q, S^-))$, *or* $Short(E_0^Q)$. *Assign* $n = 1$.
2. *Project* Q *on* ∂E_n^Q *and obtain a set of images* $I_0^Q, I_1^Q, ..., I_{8n-1}^Q$.
3. *Obtain all the natural paths* $\tau_0^Q, ..., \tau_{8n-1}^Q$ *for the images on* E_n^Q, *and compute* $Short(E_n^Q)$.
4. *Compare* $Short(E_n^Q)$ *with* $Short(E_{n-1}^Q)$ *by Eq. 70. If it is false, increase* n *by 1 and go back to Step 2. Otherwise, calculate* τ_{pref} *on* ∂E_n^Q *and identify* I_{pref} *from which* τ_{pref} *is obtained. Connect* Q *to* I_{pref} *with a shortest path. If* $I_{pref} = S^-$, *terminates, else let* $Q = I_{pref}$, *go back to 1.*

Figure 45: The algorithm of AEIP on the path

In Fig. 46, two examples of paths generated by AEIP are given. Compared with Fig. 24 and Figs. 36- 37, they are the best paths with the necessary distance away from the boundaries.

4.3.5 Time Complexity of AEIP

If no bound on AEIP, the number of paths it generates is exponential in $\mathcal{K}$, the average number of nodes joined by a path. For example, assume that the average number of images projected at S^+ is α and the average number of images projected from each node on the path is γ. Because each image indicates one path alternative, the total number of paths $\mathcal{P}$ is roughly, with possible duplicates of the paths,

$$\mathcal{P} = \alpha + \alpha * \gamma + \alpha * \gamma^2 + ... + \alpha * \gamma^{\mathcal{K}-1} = \alpha(\gamma^0 + \gamma^1 + \gamma^2 + ... + \gamma^{\mathcal{K}-1})$$

$$= \alpha(\sum_{i=0}^{\mathcal{K}-1} \gamma^i) \approx O(\alpha * \gamma^{\mathcal{K}-1}). \tag{73}$$

However, this algorithm is to be executed heuristically. In each cycle, only one path with the highest priority is selected, all others are excluded from further processing. Therefore, the performance of this algorithm is desirable. It will be analyzed in the following text.

4.3.6 Efficiency of AEIP at Source

Because S^+ holds the maximum value of ϕ in Ω (see Chapter 2), the projection can be performed from it in all the directions. Hence in the nth expansion, the maximum number of images (β) on E_n is $8n$. However, n is bounded by $M-1$ and when S^+ is located at the center of Ω and $n = \frac{M-1}{2}$, a configuration has the maximum number of images on the outmost expanded cycle, which is equal to $4M-4$. In the worst case, for an uncontrolled projection, the total number of images ever projected in the process

$$\beta_{total} = \sum_{n=1}^{\frac{M}{2}} 8n = M^2 + 2M = O(M^2). \tag{74}$$

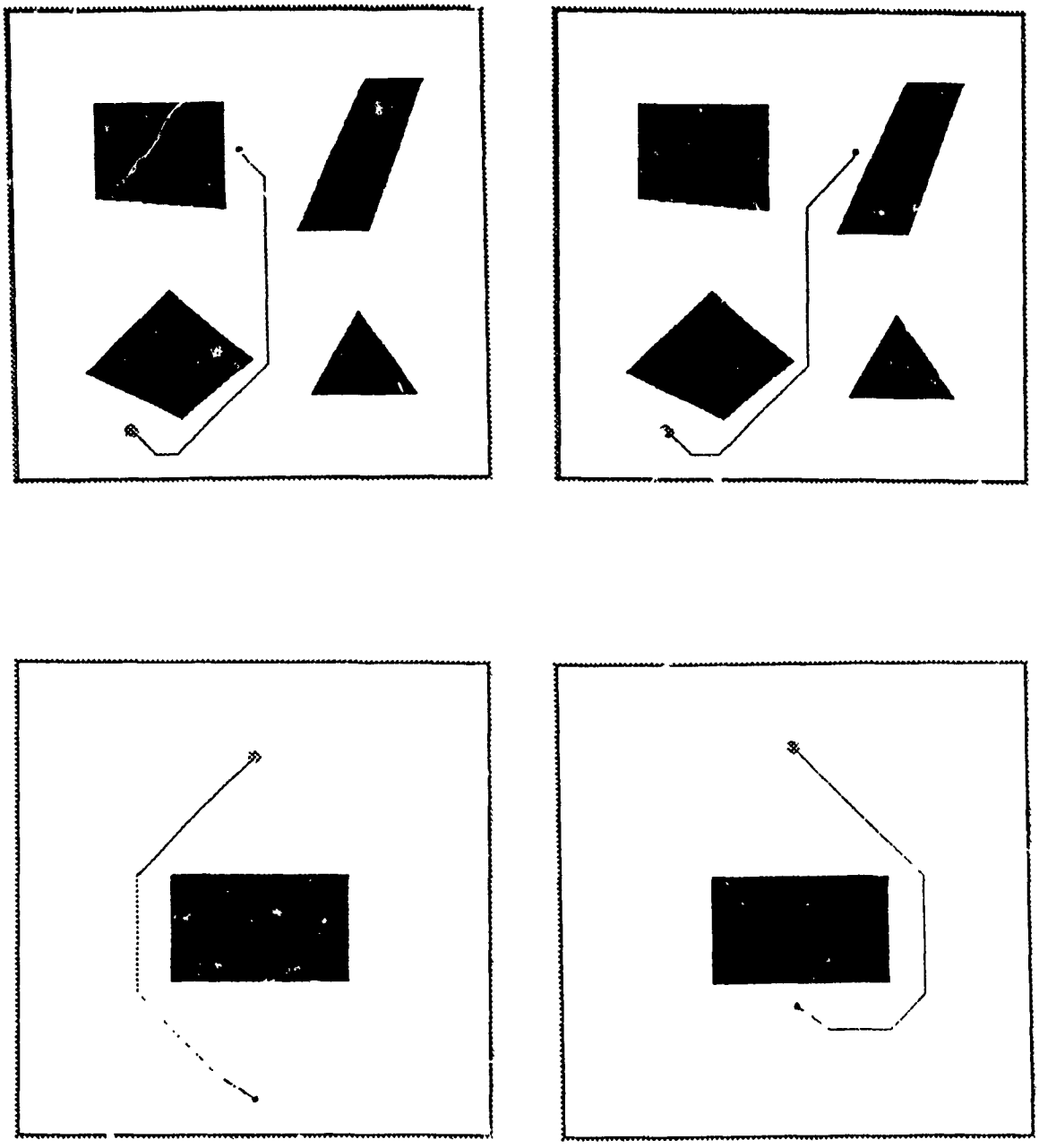


Figure 46: Samples of paths generated by AEIP.

Eq. 74 implies the upper bound of the number of images in the projection at the source is quadratic to M.

However, since AEIP at the source is used to select the correct direction to start with, one or two expansions are sufficient to obtain a good solution. When $n = 1$, there are eight paths going from S^+ in eight directions for selection. When $n = 2$, there are sixteen paths in sixteen directions. In practice, AEIP at the source is controlled by $n <= 2$. Therefore, $\beta_{total} \leq 24$ and $\mathcal{P} = 24$. The time complexity in phase one (T_1) related to the number of paths processed is

$$T_1 = \beta_{total} * \mathcal{P} * T = 24 * (24t\mathcal{K}) = 576t\mathcal{K}, \tag{75}$$

where t is the time for each iteration in SFM (see Eq. 58) and is constant.

4.3.7 Efficiency of AEIP on the Path

From Eq. 74, the coefficient for a full image projection is constant and equal to 8. Let c be the coefficient. In the second phase of AEIP, the center for expansion is any node on the way of a path other than S^+, then c is a variable. When P is close to S^+, S^-, or in narrow channels, c tends to be in a bigger range ($1 \leq c \leq 7$). However in general, the equipotential lines in a NN centered at P can be approximated by straight lines, then the average of $c \leq 4$ (see Fig.47). This reduction of c is significant when n is controlled in a small range. If $c = 1$, there is no need to invoke AEIP. Assume AEIP is executed in each elementary step, and $n = 2$ and $c = 4$, so $\beta_{total} = \sum_{n=1}^{2} c*n = 12$. Then the total number of paths processed is

$$\mathcal{P} = \beta_{total} * (\mathcal{K} - 1) = 12\mathcal{K} - 12, \tag{76}$$

where $\mathcal{K}$ is the total number of nodes traversed in a path. Let T be the time needed for each natural path generation by SFM as in Eq. 58. Then the time complexity of phase two (T_2) is

$$T_2 = \mathcal{P} * T = (12\mathcal{K} - 12) * t * \mathcal{K} = 12t\,\mathcal{K}^2 - 12t\,\mathcal{K} \approx O(\mathcal{K}^2). \tag{77}$$

Hence, in the worst case, this algorithm is quadratic with respect to the average number of nodes joined by a path (see Eq. 59). This is a drawback compared to

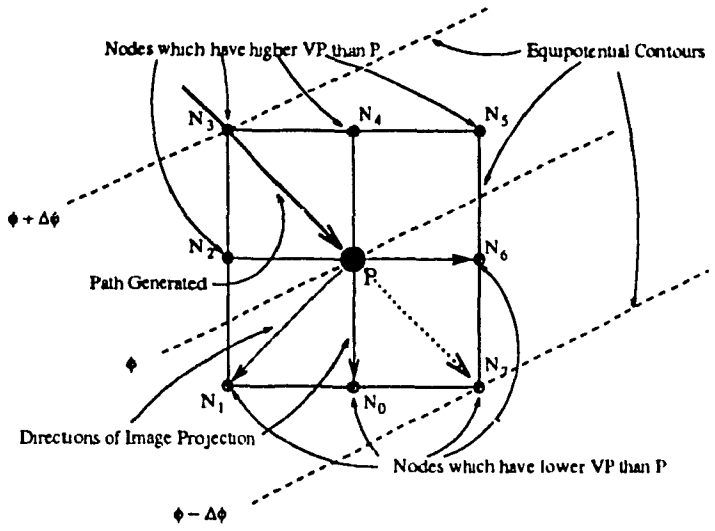


Figure 47: A neighborhood of P. P is a node on the path. In the example, only N_0, N_1, N_6, N_7 are valid node for image projection.

FDC, however, the actual number of nodes contained in a path is much smaller than M^2 for the characteristics of fluid, therefore, the resulted performance is reasonably fast.

4.4 Concluding Remarks

SFM is a simple but basic planning algorithm in our method. The paths it generates follow the route of ideal fluid and in general, are short. Because of the presence of dynamic forces in the flow, the natural routes of the fluid sometimes are not satisfactory. FDC and AEIP are designed to improve the paths. FDC imposes a threshold to the amount of the potential difference obtained by SFM from two adjacent points. It tries to find where the greater negative impact is and adjusts its flow direction so as to achieve a better path. AEIP is proposed based on the radiativity of flow of ideal fluid at the source. AEIP uses the technique of image projection, expands the searching area and achieves a vector of paths for comparison. This vector of paths covers all the directions from the point in consideration. The search is nearly complete, therefore,

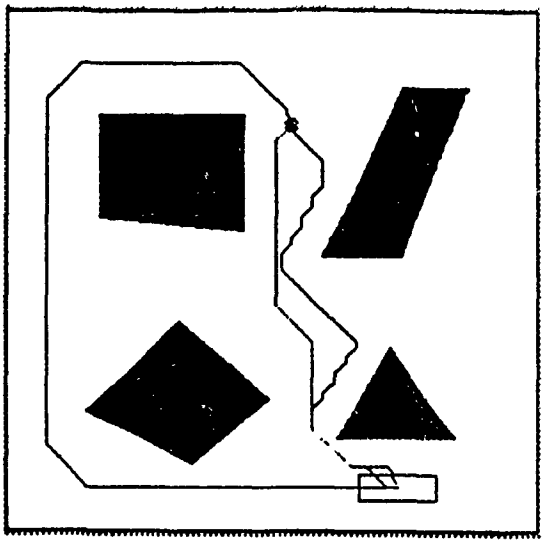


Figure 48: The comparison of the paths generated by the three algorithms. The one around the outer boundary is by SFM and the one in the middle is by AEIP. The zigzag one is by FDC.

Table 1: The comparisons among three algorithms on path planning.

Algorithm	Length(e)	Bound(e)	Improved(%)
SFM	186.72	32.48	-
FDC	139.61	0	25
AEIP	127.04	0	32

a better result is ensured. Fig. 48 shows an example of the different paths planned by SFM, FDC and AEIP. In Table 1, the average results from about 50 different samples are recorded. The improvement of the path is calculated by the following equation:

$$\text{Improved} = \left(1 - \frac{\text{Length by FDC or AEIP}}{\text{Length by SFM}}\right) * 100. \tag{78}$$

Also notice in the table that the elementary steps along the boundary of the obstacles are zero in the paths generated by FDC and AEIP, which suggests a better quality.

4.5 Comparison with W-Potentials

[BL89] proposed a few numerical potential field techniques, among them there is W-potentials which are generated by wave propagation. Its speed is claimed to be very fast and they are capable of solving as high as 31 DOF problems. In order to compare our algorithm in some aspects, we have experimented this potential function, and achieve some very interesting results.

For potential field construction, it is based on a very simple algorithm they called wavefront to generate W-potentials (i.e., potentials in the working space). Actually, this is a well-known simple algorithm which recursively assign cardinal numbers to unoccupied neighbors in a grid structure until all available spaces are exhausted. Generally, the increment of potential values in each closest level of neighborhood is regular, typically by 1 (see Fig. 17 in Chapter 2). This method is used repetitively to generate a Voronoi-like diagram first, then the potentials on this diagram and on all other available spaces in W (see Fig. 49). This is called *Improved W-potentials*. A path based on W-potentials is found by searching the diagram following the deepest descending algorithm.

W-potentials are then transformed into C-potentials (i.e., potentials in C-space). Usually, several control points are chosen from the points on the robot configuration to their corresponding goal positions. Then many paths are generated using the above technique. An arbitration function is used to integrate those paths into a global one to reduce the contention of those control points. However, no matter what kind of arbitration algorithm is used, local minima occur. The algorithm then has to adopt an escaping technique (see [BL89].

For finding a "good" path for non-point robot, C-potentials are calculated which integrate the paths of several control points chosen from the points on the robot configuration to their corresponding goal positions. An arbitration function is used to reduce the contention of the control points. Of course, local minima occur in C-potentials. According to this article, the quality of a potential function is not simply determined by the number of local minima but by the depth of those local minima. Because the deeper the local minima, the harder for robots to escape.

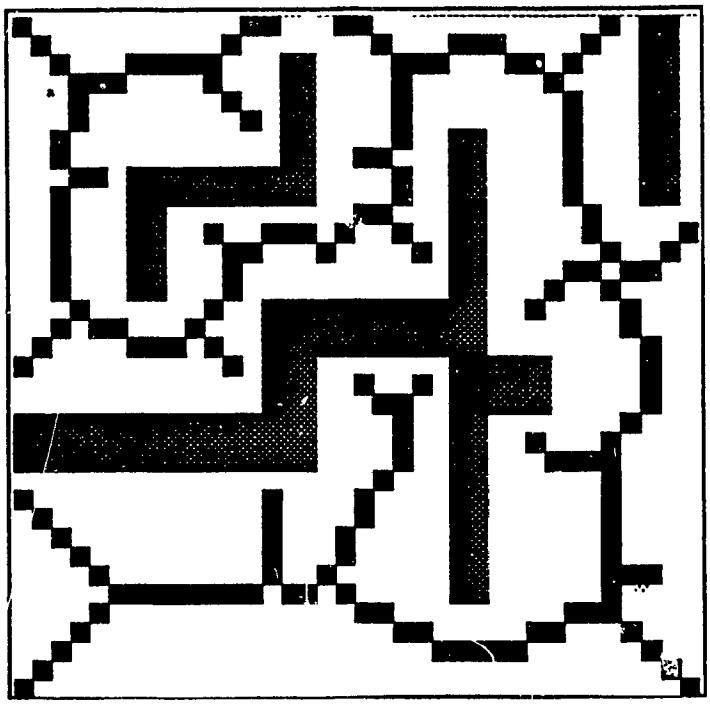


Figure 49: The Voronoi-like diagram generated by wavefront technique.

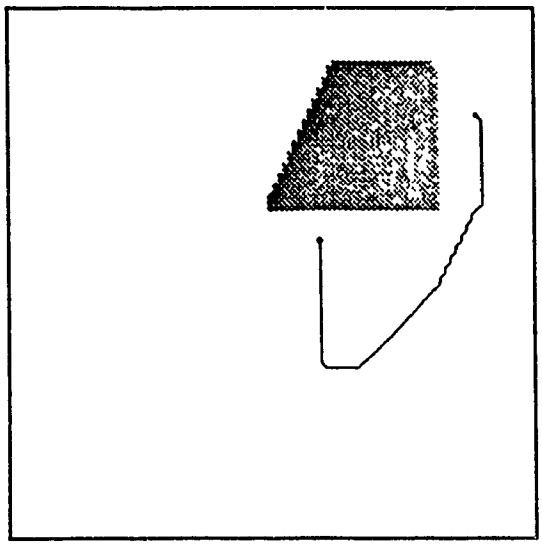


Figure 50: The global path generated by W-potential method (1).

Because the simple wavefront algorithm is used for generating W-potentials, this algorithm gains speed in the precomputation of W-space. According to our experiments, for an object with five control points, generating the W-potential for the whole work space (128X128) only takes 0.35 second. For our model, in order to achieve a steady flow simulation in W-space of the same configuration, 3.01 seconds and 140 iterations are needed. Also, the speed of generating W-potential tends to be constant. It changes slightly with the changes of the size of the work space and the number of obstacles. However, our model's results vary because, in solving the differential equation for a work domain, different configurations may require different number of iterations to achieve the desired error tolerance.

However, because the final path generated by W-potentials is based on the Voronoi like diagram, in some cases, there exit limitations. Fig. 50 and Fig. 51 give examples of the paths generated by the improved W-potential method. They are clearly not short paths. The results indicate that this algorithm works fine in an environment with evenly distributed obstacles, However, in other cases, it may not. By using our fluid model, this can be avoided. Fig. 52 and Fig. 52 show the results in the same working space as in Fig. 50 and Fig. 51.

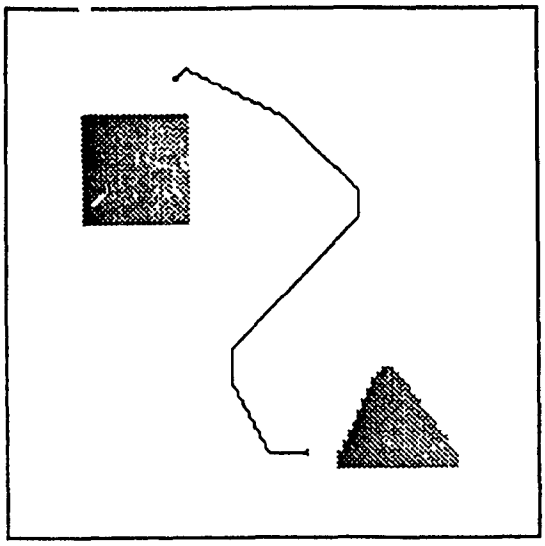


Figure 51: The global path generated by W-potential method (2).

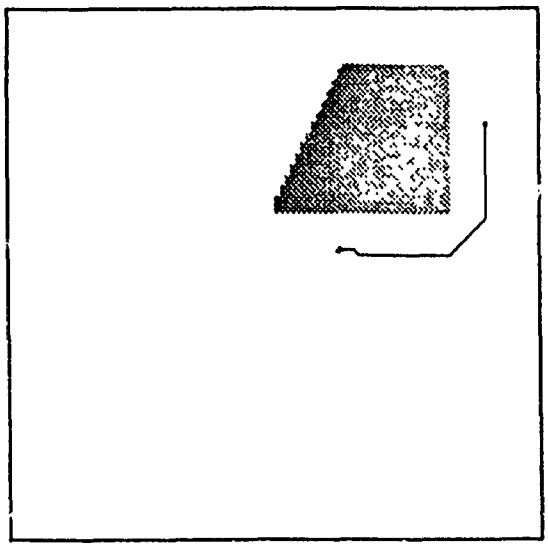


Figure 52: The global path generated by SFM (1).

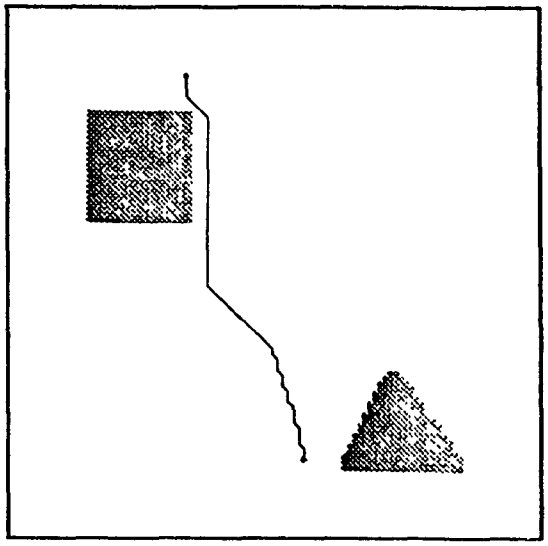


Figure 53: The global path generated by SFM (2).

Chapter 5

Technique on Collision Avoidance

5.1 Introduction

In an obstacle-present environment, robot motion is constrained by the boundary of the obstacles. It is the only constraint when we assume that the robot has no kinematic constraints for itself. Collision avoidance is a crucial issue in robot motion planning. In simple cases, like planning for point robots, or convex polygons with fixed orientation, the best way is to integrate the strategies of path planning and collision avoidance into one in a configuration space through Eq. 1. In the configuration space, the robot is represented as a point only, and the shape of obstacles ($\mathcal{B}$) are calculated by considering the shape of the original robot and its orientation. Lozano-Pérez gave an efficient algorithm for computing $\mathcal{B}$ in an environment in which all objects are convex polygons (see [LP83]).

However, for a robot with free translation and rotation, the collision avoidance becomes much harder. For example, in the configuration space of a robot with two translations and one rotation (3 DOFs), $\mathcal{B}$ is a 3 dimensional volume bounded by patches of C-surfaces (see [BT83]). The dimension in Ω is increased by one when one more DOF is added to the robot.

The alternative to keep the dimension of the configuration space the same as its physical space is first to plan a path for a reference point p in the robot (generally, p is the center or the center of gravity of the robot) and then to conservatively approximate

the range of free orientation of a robot at a given configuration on the path. Many planning methods have their own techniques for collision avoidance in this manner. For example, the *enclosed rectangle* in freeway method (see [Bro83]), *critical curves and non-critical regions* in exact cell decomposition method (see [SS83]) and so on.

Another technique is first to choose those extremities of the shape of a robot as reference points, and then plan a path for each reference point. An arbitration function is needed to solve the contention between those reference points so as to keep the physical integration of the robot in its motion. This technique is good for conformable robots.

When planning for multiple jointed robot or car-like robot, we are dealing with a robot with constraints not only on obstacles but also on its own kinematic movement. Different techniques are used to deal with both holonomic and nonholonomic constraints of a robot (see [Lat91], Chapter 9).

The basic approach of our current research is to define a new numerical method for motion planning in 2 dimensional environment. The robot is assumed to be rigid with free translations and possibly one rotation (i.e., 3 DOFs). Because the predefined domain (environment) is embedded in a grid structure, we find the bitmap technique is the most efficient and the easiest way for collision avoidance.

5.2 Bitmap for Obstacle Detection

Among all the techniques, bitmap is, perhaps, the best one to comply with the grid structure of Ω. We adopted the bitmap approach to solve the collision avoidance problem when rotation is allowed to a robot. This method is confined to solving a problem of 3 DOFs, i.e., an object having two degrees of translations and one degree of rotation.

5.2.1 Formation of the Bitmap

Assume Ω is a square and $h = h_x = h_y$ and $\mathcal{A}$ is a rectangular rigid robot and a free-flying object in 3 DOFs. Its width is equal to $2ah$ and its length $2bh$ ($a, b \geq 0$

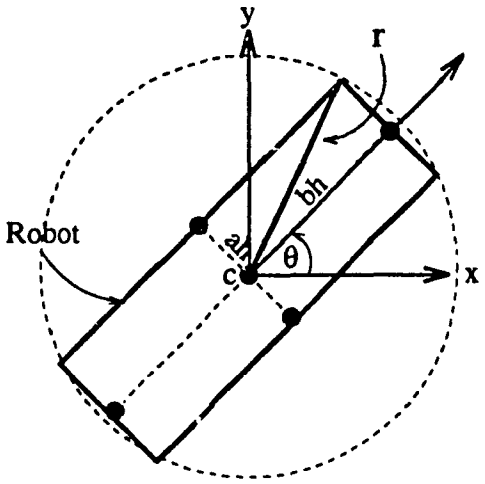


Figure 54: A rectangular robot represented by (x, y, θ).

and smaller than some predefined limit of the size of the robot). Let $c(x, y)$ be the center of the robot. The configuration of $\mathcal{A}$ in Ω is denoted by $\mathcal{A}(x, y, \theta)$. x, y are the domain coordinates in Ω and θ is the angle between the x-axis of Ω and the moving direction of $\mathcal{A}$, which coincides with the spine line of $\mathcal{A}$ across c (see Fig. 54). To satisfy a free rotation of the robot around c, the circular area around c with a radius r equal to $\sqrt{(ah)^2 + (bh)^2}$ must be free of obstacles (i.e., $\in \mathcal{F}$). After the initialization of Ω, each node contains information showing whether the region it represents is occupied or not. All the information is stored in an information base of Ω. Since the coordinates of c is known, we can start from the correspondent node of c and extract information about those nodes which are within the range of r. Let $\mathcal{N}(x', y')$ be a node in Ω. It occurs in the bitmap, provided

$$\sqrt{(x' - x)^2 + (y' - x)^2} \leq r. \tag{79}$$

Each node appears as a bit in the bitmap at its correspondent location. If the node is occupied, a bit is assigned to 1 in the bitmap, else to 0. After the extraction, a circular bitmap is constructed with only 1's and 0's as in Fig. 55. Now the collision detection can be done through scanning the extracted bitmap.

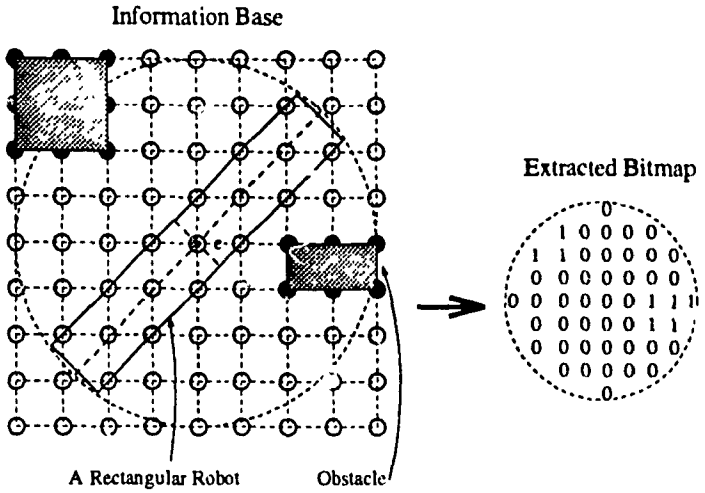


Figure 55: The bitmap technique for obstacle detection.

5.2.2 Bitmap Scanning and Bit Arrays

Let $\mathcal{M}$ denote the bitmap. It is conveniently subdivided into four quadrants, Q_I, Q_{II}, Q_{III} and Q_{IV} as in Fig. 56, since it is in the shape of a cycle. The scanning will be done for each quadrant separately in row and in column. For each quadrant, two bit arrays are defined. One is for the rows and the other for the columns. Along the scanning process, the bit values are "OR"ed and finally stored in bit arrays. Therefore, each bit in the array represents the union of the bits in that row or column. Examples of the scanning and the resulted bit arrays in Q_{II} and Q_{IV} for the sample bitmap in Fig. 55 are shown in Fig. 57. Clearly, the bit arrays have the same length, because there are as many rows as there are columns. The number of bits in each row or each column in $\mathcal{M}$ may vary. However, it can be calculated through Eq. 79. Therefore, only one array (e.g., $length[n]$, n is the size of bit arrays) is needed to store the exact number of bits in each row or column. The maximum of bits in both row and column is $\lfloor\sqrt{(ah)^2+(bh)^2}\rfloor$. The order of the array always starts from the center, i.e., the first bit shows the row or the column where c is on. The first bit is the most significant bit and the $(\lfloor\sqrt{(ah)^2+(bh)^2}\rfloor)$th bit is the least. The bitmap in each region will be processed in the same manner and its two bit arrays will be obtained which represent the results of horizontal and vertical scanning and union operation,

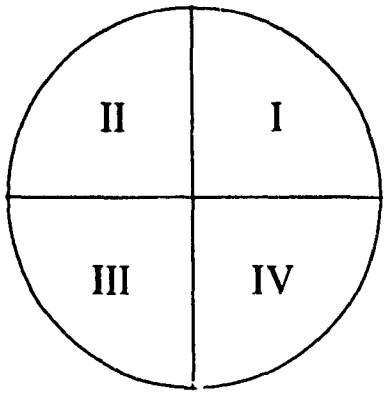


Figure 56: The subdivision of the bitmap.

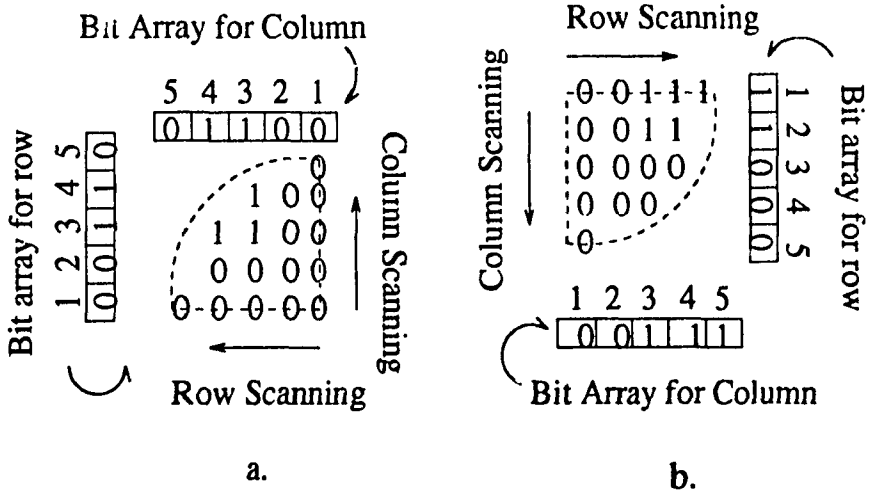


Figure 57: The scanning of the bitmap is always to start from the center of the circle. The bit values in the scanning process are ORed and the results are stored in the row or column bit arrays, respectively.

respectively. After scanning, eight bit arrays are obtained. For example, in Fig. 55, if we show in ascending order for quadrants, put the row bit array first and the column bit arrays second in a quadrant, the values of those bit arrays are 10000 00111 00110 00110 00000 00000 11000 00111.

5.2.3 Cases Analysis and Union of Quadrants

Assume that the configuration of a robot A is defined by $A(x, y, \theta)$ and $A \in \mathcal{F}$. The value of θ refers to the x-axis in Ω. Occupied regions (i.e., either R_0 or R_1) can be decided by θ.

For simplicity, assume $a = 0$, i.e., $\mathcal{A}$ has zero width (a straight line). The loss of accuracy by the assumption can be compensated, for example, through expansion of the obstacles in the configuration space with a parameter related to the original width of $\mathcal{A}$. Because of the center of $\mathcal{A}$ resides at the center of $\mathcal{M}$. There are four cases for the position of A:

1. Reside in Q_I and Q_{III},
2. Reside in Q_{II} and Q_{IV},
3. Reside on the boundary of Q_I and Q_{II} or Q_{III} and Q_{IV} (vertical),
4. Reside on the boundary of Q_I and Q_{IV} or Q_{II} and Q_{III} (horizontal),

Obviously, the two diagonal quadrants are always used together. Hence, they can be regarded as one part. Let R_0 be the union of Q_I and Q_{III} and R_1 of Q_{II} and Q_{IV} (see Fig. 58). Then the number of cases is reduced to three.

1. Reside in R_0,
2. Reside in R_1,
3. Reside on the boundary of R_0 and R_1.

Here, we actually have two questions to answer. While $\mathcal{A}$ is in R_i, $i = 0, 1$, the question is whether it can freely rotate in R_i or out of R_i. If not, what is the size of

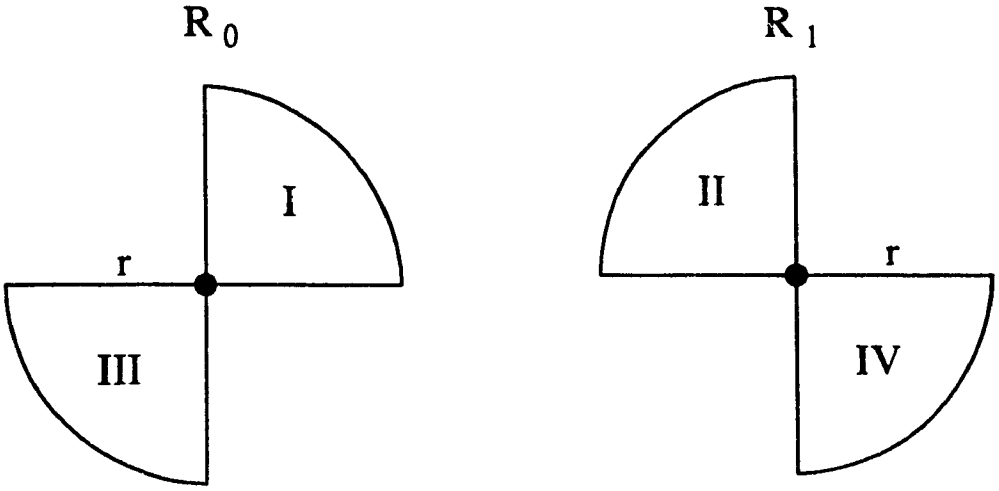


Figure 58: The two diagonal quadrants are combined together to decide the rotation. Therefore, $\mathcal{M}$ is divided into two regions: R_0 and R_1.

the angle to rotate without collision? While $\mathcal{A}$ is not in R_i, the question is whether $\mathcal{A}$ can rotate into $R_{|i-1|}$ and how far it can go. The second question needs to be solved only when the answer to the first is positive. The two problems are often required to be solved together so as to decide the maximum allowable rotation of $\mathcal{A}$ in both clockwise and counter-clockwise.

To further simplify the calculation, the two quadrants in R_i can be combined as if one quadrant has been turned 180 degrees around c. The technique is as follows.

Let Q' be the union of Q_1 and Q_3, i.e.,

$$Q' = Q_1 \cup Q_3. \tag{80}$$

The union operation can be either row-wise or column-wise. For example, we choose the row-wise union operation. Then

$$Q_1 = r_1^1 + r_1^2 + ... + r_1^n \text{ and } Q_3 = r_3^1 + r_3^2 + ... + r_3^n, \tag{81}$$

where r_m^n represents the bits in nth row of Q_m and n is the total number of row bit arrays in Q_1 or Q_3. Therefore,

$$Q' = r_1^1 \cup r_3^1 + r_1^2 \cup r_3^2 + ... + r_1^n \cup r_3^n. \tag{82}$$

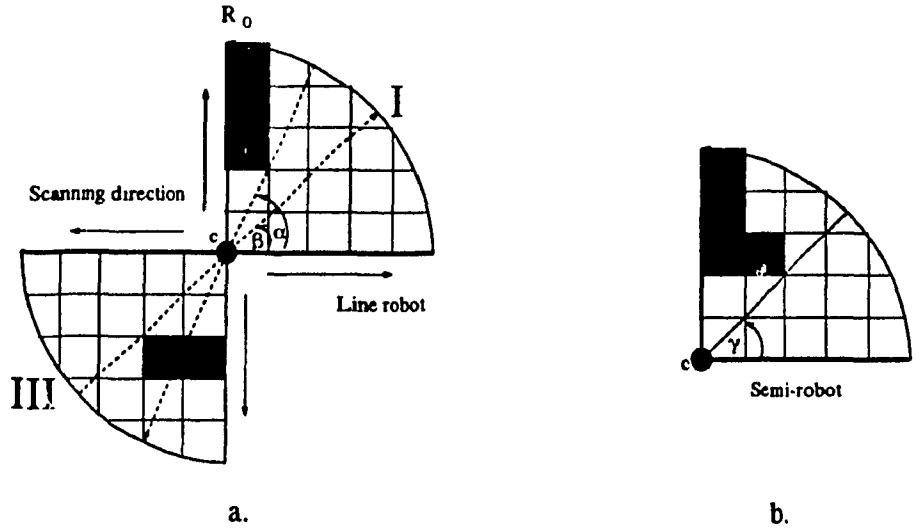


Figure 59: The union of the two diagonal quadrants. a). The original configuration in R_0. b). The combined quadrant, in which the maximum allowable rotation angle remains unchanged, i.e., $\gamma = \beta$.

The union operation should be done in the same order of the correspondent bit arrays. As we stated before, the order of the bit arrays is always from the center of $\mathcal{M}$ to the outside-circle. The advantage is shown in Fig. 59. In Fig. 59a, two angles (α and β) are calculated in Q_I and Q_{III} respectively. After the union operation, only one angle (γ) needs to be calculated. Clearly $\gamma = \beta$, which is the maximum allowable angle to rotate into R_0 counter-clockwise. This can be verified since the union of the row or column bit arrays of Q_1 and Q_3 is exactly equal to the row or column bit array of Q'. For example, in Fig. 59a, the row and column bit arrays for Q_1 contain 00111 and 10000 respectively, and for Q_3, they are 00100 and 11000. Then we have the union of the bit arrays of Q_1 and Q_3:

$$\begin{array}{llll} \textit{row} & 00111 & \textit{column} & 10000 \\ & 00100 & & 11000 \\ & ------ & & ------ \\ & 00111 & & 11000. \end{array} \tag{83}$$

Clearly, the result is the same as the row and column bit arrays for Q' (see Fig. 59b).

Now, for each region, i.e., R_0 or R_1, we can have only two bit arrays, one for the row and the other for the column.

5.2.4 Maximum Allowable Rotation into R_i

First, let us answer the second question raised in Section 5.2.3, because this problem can be solved only through analyzing the combined bit arrays of R_i. We have the following steps to process the bit arrays:

1. If the value of either row or column bit array for R_i is zero (i.e., all are zero bits in the array), R_i is totally in $\mathcal{F}$. Then, no further collision detection is needed.

2. The first bit in both bit arrays is called *entry bit*. As to R_0, if the first bit of its row bit array is 1, it cannot be entered counter-clockwise, and if the first bit of its column bit array is 1, it cannot be entered clockwise. The same as to R_1, but in reverse direction. For example, for R_0 in Fig. 55, the row bit array is 10000. Then this region cannot be entered counter-clockwise. However, its column bit array is 00111. It can be entered clockwise.

3. If the entry bit for R_i is zero, what is the maximum angle allowed for entering R_i without collision? This can be easily calculated through two ways.

   (a) **Tangent Method.** In Fig. 60a, if the lengths of $\overline{st}$ and $\overline{ct}$ are known,

   $$\alpha = arctan\left(\frac{\overline{st}}{\overline{ct}}\right) = arctan\left(\frac{uh}{vh}\right) = arctan\left(\frac{u}{v}\right), \qquad (84)$$

   where u, v are positive integers. In fact, u and v can be obtained through counting the bit arrays. The counting always starts from the entry bit. Clearly, the entry bit for a bit array must be zero so that the counting can be applied. There are two methods of counting, denoted by $m1$ and $m2$. For $m1$, counting starts from the entry bit until a 1 is encountered, then record the count in $c1$. For $m2$, when counting meets a 1, go on counting until a 0 or the last bit is encountered, put the count in $c2$. If encountered by a 0, go on counting the another part of the bit array. If all zeros, report

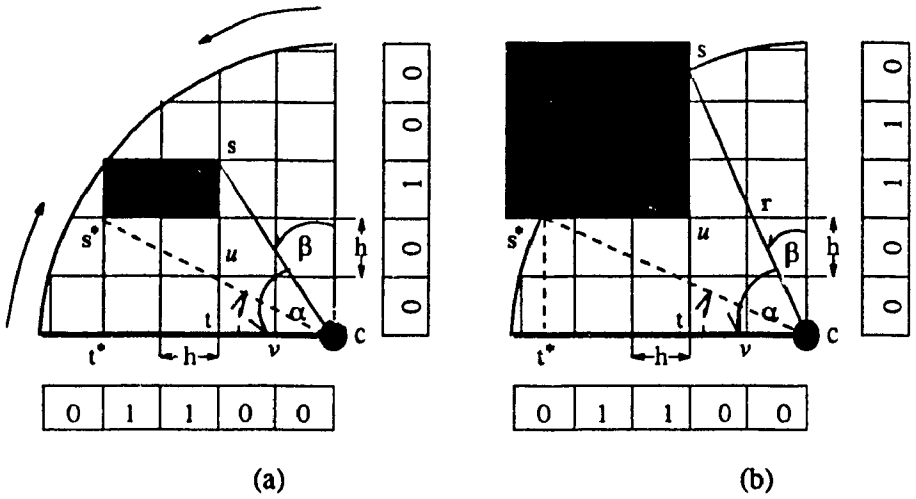


Figure 60: The tangent method and cosine method in calculation of α.

Table 2: The technique of calculating u and v in R_0 and R_1.

Entry	Clockwise				AntiClockwise			
Region	R_0		R_1		R_0		R_1	
	Array	Method	Array	Method	Array	Method	Array	Method
u	Col.	$m1$	Row	$m1$	Col.	$m2$	Row	$m2$
v	Row	$m2$	Col.	$m2$	Row	$m1$	Col.	$m1$

$c2$, else put the count after the last 1 in the array into $c2$. In different regions, the method applied may be different. Table 2 shows the detail of the counting method for different regions in different directions.

For example, in Fig. 60a, the column bit array is 00110 and the row bit array is 00100. This region is allowed to enter from both sides. Now we consider entering the region counter-clockwise. Counting the column bit array using $m1$, we get $v = 2$, and counting the row bit array using $m2$, we get $u = 3$. Now we can apply the tangent method to calculate α.

$$\alpha = arctan(\frac{u}{v}) = arctan(\frac{3}{2}) \approx 56.3. \tag{85}$$

Since α is known, the maximum counter-clockwise rotation angle β can be easily calculated according to what quadrant is under consideration.

(b) **Cosine Method.** Cosine method is used for calculation when the value n obtained by $m2$ is equal to or greater than $length[n]$. For example in Fig. 60b, $u = 4$ using $m2$ and $length[4] = 3$, so the cosine method is used. Then

$$\alpha = arccos\left(\frac{\overline{cs}}{\overline{ct}}\right), \tag{86}$$

where

$$\overline{cs} = r = \sqrt{(ah)^2 + (bh)^2} = \sqrt{(bh)^2} = bh = 5h, \tag{87}$$

since $a = 0$ and $b = 5$ for the line robot. Let $d = \frac{r}{h}$, then

$$\alpha = arccos(\frac{c}{d}) = arccos(\frac{2}{5}) \approx 66.4 degrees, \tag{88}$$

where c is either u or v obtained by $m1$.

5.2.5 Maximum Allowable Rotation in R_i

From Section 5.2.4, we know whether a region can be rotated into or not, and if it can, how far (or what angle) it can be rotated. However, in the quadrants which $\mathcal{A}$ occupies, if they are not entirely free, where are the bounds of its rotation? They cannot be calculated simply by using bit operations as we have done in Section 5.2.4. A new strategy of calculating the maximum rotation angle is explained below.

Let a bitmap $\mathcal{M}$ be extracted from Ω as stated in Section 5.2.1. Let R_i be in $\mathcal{M}$ where the line robot $\mathcal{A}(x, y, \theta)$ is positioned with its center point $c(x, y)$ at the center of $\mathcal{M}$ and an angle θ to the x-axis of Ω. If R_i is entirely in $\mathcal{F}$, only the entry checking to $R_{|i-1|}$ is necessary. Assume part of R_i is in $\mathcal{B}$. Then we have a different strategy.

1. Scan Q', the combined quadrant of R_i, row by row (or column by column) and locate the *critical points*. Here critical points mean where the bit value changes from 0 to 1 or from 1 to 0. If by row, scanning starts from the y-axis which crosses the center of $\mathcal{M}$. If by column, starts from x-axis. Always scan the row or column which is closer to the center first. The scanning is done in a loop. When the bit is 1, store the coordinates (in row and column) in $CritP$, then ignore the continuous 1's until a 0 is encountered or the end of the bit array. If 0

is met and not the end of the array, another coordinates (in row-1 and column) is stored in $CritP$. The scanning continues until all the bits array in Q' are passed. Then a set of critical points is obtained. The algorithm for row-wise scanning is shown below:

```
scan(Q,CritP, length, n, Cn)
   bool  Q[][];     2 dimensional arrays in 1's and 0's
   point *CritP;    store the coordinates of obstacles
   int   length[];  the number of bits in each row
   int   n, *Cn;    n is the number of rows or columns and
                    Cn is the number of elements in CritP
 { int i,j;  bool inbound = FALSE, temp;  *Cn = 0;
    for (i = 1; i < n; i++) {
        for (j = 0; j < length[i]; j++) {
           switch (Q[i][j]) {
              0:  temp = FALSE;
                  if (inbound != temp) {
                     CritP.r = i-1; CritP.c = j;
                     CritP++; *Cn++; inbound = FALSE;
                    }
                  break;
              1:  temp = TRUE;
                  if (inbound != temp) {
                     CritP.r = i; CritP.c = j;
                     CritP++; *Cn++; inbound = TRUE;
                    }
                  break;
             }
        }
    }
}
```

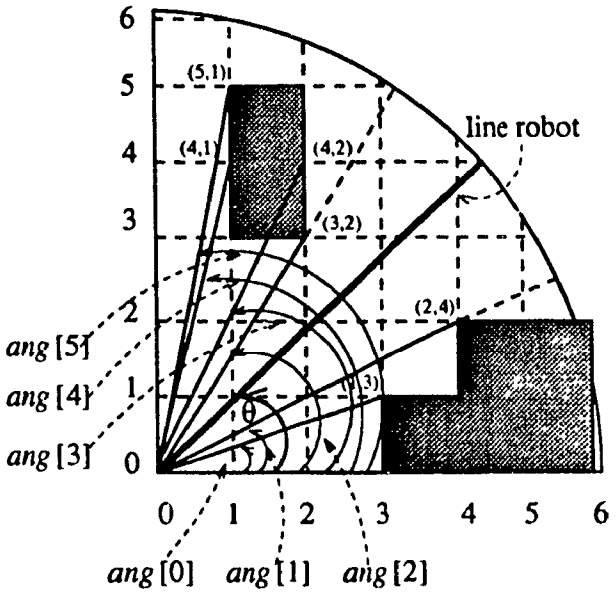


Figure 61: The calculation of maximum allowable rotation angles on Q' of R_0. The allowable rotation area is $ang[2] - ang[1]$.

2. Calculate the angles for each starting or ending point for the obstacles using

$$ang[i] = arctan(\frac{CritP[i].r}{CritP[i].c}). \tag{89}$$

Then sort $ang[]$ in ascending order. Finally, search $ang[]$ for the location of θ and get $ang[k]$ and $ang[l]$ which are closely adjacent to θ in two sides such that

$$ang[k] < \theta < ang[l]. \tag{90}$$

Clearly, $ang[k]$ and $ang[l]$ are the two bounds for the rotation of $\mathcal{A}$ around its center in R_i (see Fig. 61), i.e., the maximum counter-clockwise rotation is $ang[l]-\theta$ and the maximum clockwise rotation is $ang[k]-\theta$. The total allowable rotation angle is $ang[l] - ang[k]$. If θ is smaller or larger than any element in $ang[]$, one bound is on the x or y-axis in Q_i.

5.2.6 Concluding Remarks

The bitmap algorithm is an efficient technique for collision avoidance in a domain embedded in grids. In fact, this algorithm is valid for any shape of rigid objects

provided they can be enclosed in a circle. The possible change for different shapes is the change of the way $\mathcal{M}$ is divided. For example, for an L-shaped robot, $\mathcal{M}$ can be divided into two parts according to whether it is occupied by the robot. So the adjacent two quadrants now are combined into one region. The actual calculations (e.g., the tangent method or cosine method) may have to be changed too. If the two sides of the robot coincide with x and y-axis simultaneously, the division of the regions can be arbitrary. In practice, this technique is used after fluid simulation and a global path is found. In each step, $\mathcal{M}$ is created and tested. If no collision is found, go to the next step on the path. If it is, some simple escaping techniques are used such as backing up one or two steps, adjusting its moving directions, or reducing rotation angles, etc. $\mathcal{M}$ is regenerated, and the same cycle goes again until the goal configuration is reached.

The global path generated acts as important references for the moving directions of the non-point robot. It is used to guide the robot to its goal and the robot tends to converge its movements to the closest path line. Problem of this technique is that it may fail in some situation where the areas between obstacles are narrow and composed of continuous turns. Ideally, these techniques should be used with AEIP (see Chapter 4). When a collision danger is found on a global path and there is no way to escape it, the path will be abandoned and another global path generated by AEIP will be tried, because AEIP keep records of all the paths found. This process goes on until a collision-free path for the non-point robot is found or to declare failure when all the global paths are tested without success.

Fig. 62 gives some samples of paths for rectangular and triangular robots generated with the bitmap collision detection. Fig. 63 shows a path for an L-shaped robot.

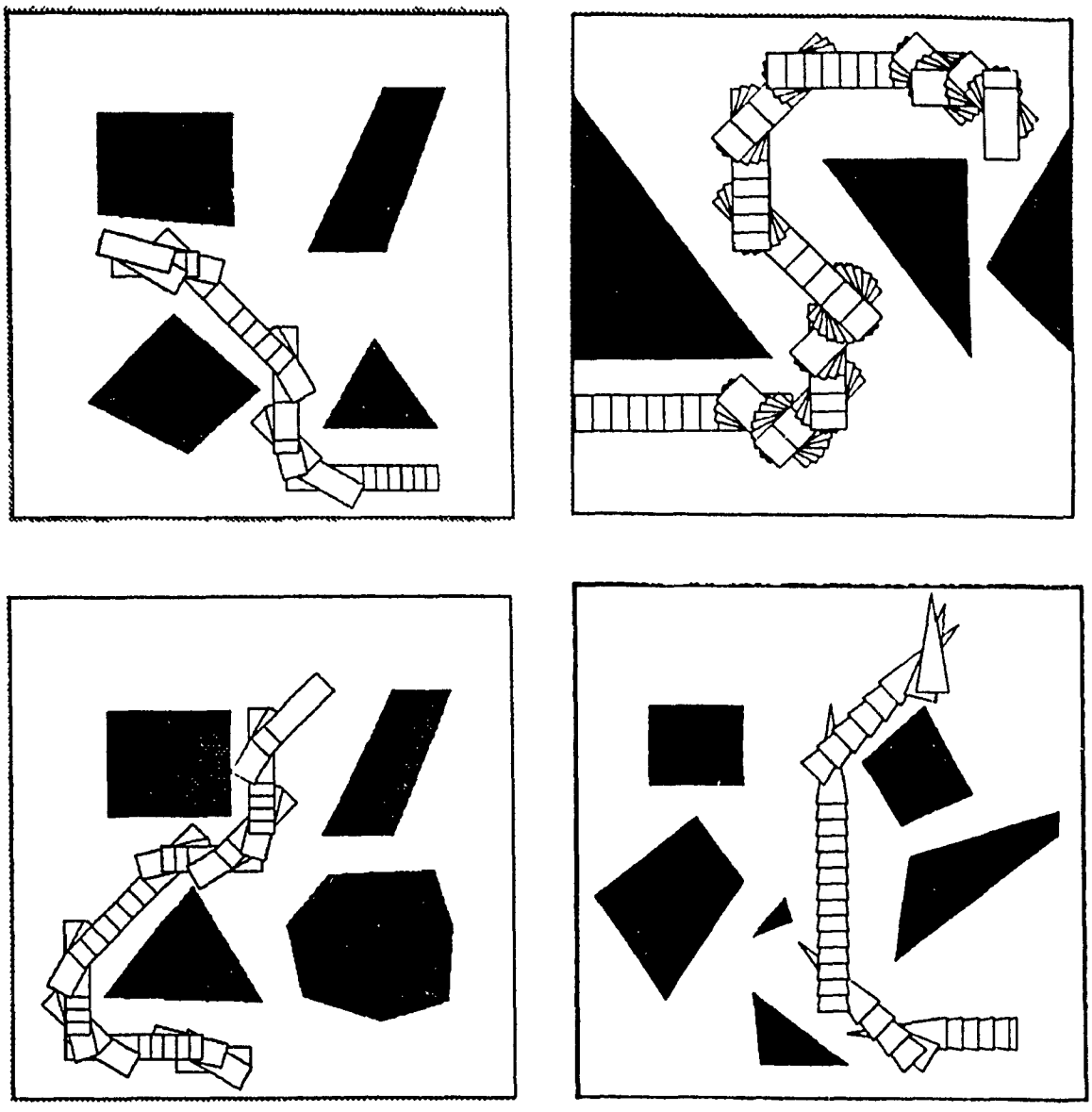


Figure 62: Samples of paths for rigid robots with 3 DOFs in different environment. The collision detection is done by the bitmap algorithm.

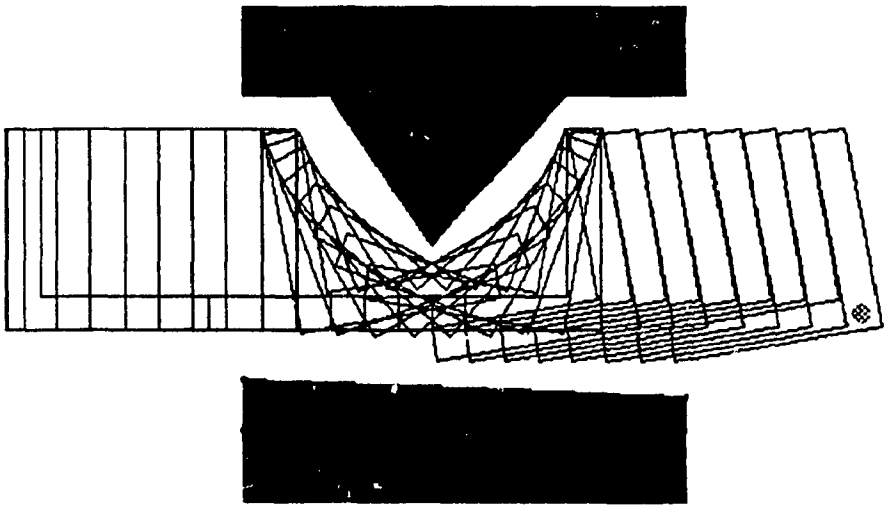


Figure 63: A path for an L-shaped robot generated by the fluid model and with the bitmap algorithm to deal with collision avoidance.

Chapter 6

Technique in Solving Poisson's Equation

Solving the path planning problem is at least NP hard. For real time application, the speed is essential. In our method, the simulation of fluid involves solving Eqs. 21 and 26, and its time complexity increases quadratically with the increase of the number of unknowns in its representative matrix (i.e., $O(n^2)$, where $n = M \times N$). However, there are advanced computation techniques which can substantially raise the convergence rate of iterations and lead to much better performance (up to $O(n \log n)$). In this section, different kinds of iteration techniques for the fluid simulation are briefed and the results of experiments are explained. By using advanced techniques, this model will be virtually made possible for real time application.

6.1 Formulation of Difference Equations

6.1.1 Denotations

To solve Poisson's equation in Ω, we adopted the *Finite Difference Method.* From Section 2.2, the bounded working domain Ω is embedded in a grid. Let the size of the grid be M in x-direction and N in y-direction. Then there are $(M-1) \times (N-1)$

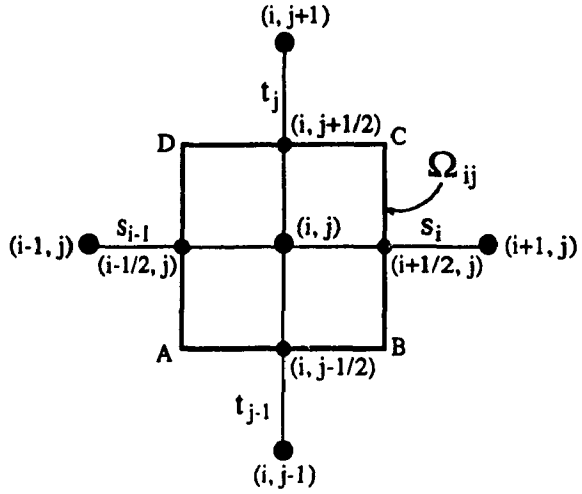


Figure 64: The interior difference node (i, j).

rectangles (denoted by $\square$) in Ω, i.e.,

$$\square_{ij} \subseteq \Omega \text{ and } \Omega = \bigcup_{ij} \square_{ij}, \tag{91}$$

where $0 \leq i \leq M-1$ and $0 \leq j \leq N-1$ and $\square_{ij}$ denotes a unit rectangle whose lower-left corner is at the node (i, j). In our context, a node is enclosed in brackets with two parameters showing the order of spacings in x and y directions. i.e.,

$$(i,j) = \{x, y \mid x = x_i,\ y = y_j\}, \tag{92}$$

Also the difference solution for the potentials is denoted by

$$\phi_{ij} = \phi(i,j) = \phi(x_i, y_j), \tag{93}$$

and the mesh spacings

$$s_i = x_{i+1} - x_i, \quad t_j = y_{j+1} - y_j, \tag{94}$$

Now let us establish the difference equations on the interior nodes (i, j) (see Fig. 64). A subdomain $\Omega_{ij} \in \Omega$ such that

$$\Omega_{ij} = \left\{ \begin{array}{l} (x, y), \quad x_{i-1/2} \leq x \leq x_{i+1/2}, \\ \qquad\quad y_{j-1/2} \leq y \leq y_{j+1/2}. \end{array} \right\} \tag{95}$$

where

$$x_{i+1/2} = (x_i + x_{i+1})/2, \quad y_{j+1/2} = (y_i + y_{j+1})/2. \tag{96}$$

and

$$x_{i-1/2} = (x_{i-1} + x_i)/2, \quad y_{j-1/2} = (y_{j-1} + y_i)/2. \tag{97}$$

Also denote the middle nodes

$$(i+1/2, j) = \left\{x, y \mid x = x_{i+1/2}, y = y_j\right\}, \tag{98}$$

$$(i, j+1/2) = \left\{x, y \mid x = x_i, y = y_{j+1/2}\right\}, \tag{99}$$

$$(i-1/2, j) = \left\{x, y \mid x = x_{i-1/2}, y = y_j\right\}, \tag{100}$$

and

$$(i, j-1/2) = \left\{x, y \mid x = x_i, y = y_{j-1/2}\right\}. \tag{101}$$

6.1.2 The five-point Formula

Integrating the two sides of Eq. 23 on Ω_{ij} yields (see [Way59])

$$\int\int_{\Omega_{ij}} \Delta\phi d\Omega = -\int\int_{\Omega_{ij}} f d\Omega. \tag{102}$$

From Green's formula, we obtain (see Fig. 64)

$$\int\int_{\Omega_{ij}} \Delta\phi d\Omega = \int_{\partial\Omega_{ij}} \frac{\partial\phi}{\partial n} dl \tag{103}$$

$$= \int_{\overline{AB}} \frac{\partial\phi}{\partial n} dl + \int_{\overline{BC}} \frac{\partial\phi}{\partial n} dl + \int_{\overline{CD}} \frac{\partial\phi}{\partial n} dl + \int_{\overline{DA}} \frac{\partial\phi}{\partial n} dl. \tag{104}$$

The integral on AB can be approximately by

$$\int_{\overline{AB}} \frac{\partial\phi}{\partial n} dl \approx \frac{\partial\phi}{\partial n}(i-1/2, j) \approx \frac{s_i + s_{i-1}}{2} \frac{\phi_{i,j-1} - \phi_{ij}}{t_{j-1}} \tag{105}$$

Similarly, we obtain

$$\int_{\overline{BC}} \frac{\partial\phi}{\partial n} dl \approx \frac{t_j + t_{j-1}}{2} \frac{\phi_{i+1,j} - \phi_{ij}}{s_{j-1}} \tag{106}$$

$$\int_{\overline{CD}} \frac{\partial\phi}{\partial n} dl \approx \frac{s_i + s_{i-1}}{2} \frac{\phi_{i,j+1} - \phi_{ij}}{t_{j-1}} \tag{107}$$

$$\int_{\overline{DA}} \frac{\partial \phi}{\partial n} dl \approx \frac{t_j + t_{j-1}}{2} \frac{\phi_{i-1,j} - \phi_{ij}}{s_{j-1}} \tag{108}$$

According to Eq. 25, f is a constant. Hence when (i,j) is neither S^+ nor S^-, then

$$\int\int_{\Omega_{ij}} f d\Omega = 0. \tag{109}$$

Then the difference equation from Eq. 105-108 can be written as

$$\frac{t_j + t_{j-1}}{2} \left(\frac{\phi_{i+1,j} - \phi_{ij}}{s_i} + \frac{\phi_{i-1,j} - \phi_{ij}}{s_{i-1}} \right) + \frac{s_i + s_{i-1}}{2} \left(\frac{\phi_{i,j+1} - \phi_{ij}}{t_j} + \frac{\phi_{i,j-1} - \phi_{ij}}{t_{j-1}} \right) = 0. \tag{110}$$

Also when $(i,j) = S^+$, we have

$$\int\int_{\Omega_{ij}} f d\Omega = -c, \tag{111}$$

and when $(i,j) = S^-$,

$$\int\int_{\Omega_{ij}} f d\Omega = c. \tag{112}$$

Consequently, the difference equations on S^+ and S^- are nonhomogeneous:

$$\frac{t_j + t_{j-1}}{2} \left(\frac{\phi_{i+1,j} - \phi_{ij}}{s_i} + \frac{\phi_{i-1,j} - \phi_{ij}}{s_{i-1}} \right) + \frac{s_i + s_{i-1}}{2} \left(\frac{\phi_{i,j+1} - \phi_{ij}}{t_j} + \frac{\phi_{i,j-1} - \phi_{ij}}{t_{j-1}} \right) = \pm c. \tag{113}$$

Here, c is a constant and typically is assigned to 1.

Next we discuss the difference equations when a node (i,j) is on the boundary as illustrated in Fig. 65. Here, the boundary is defined on grid edges, either horizontal or vertical. Let $\partial\Omega_{ij}$ be $ABECDA(\subset \Omega)$. We have

$$\int\int_{\Omega_{ij}} \Delta\phi dl = \oint_{\partial\Omega_{ij}} \frac{\partial \phi}{\partial n} dl$$

$$= \int_{\overline{AB}} \frac{\partial \phi}{\partial n} dl + \int_{\overline{BE}} \frac{\partial \phi}{\partial n} dl + \int_{\overline{EC}} \frac{\partial \phi}{\partial n} dl + \int_{\overline{CD}} \frac{\partial \phi}{\partial n} dl + \int_{\overline{DA}} \frac{\partial \phi}{\partial n} dl, \tag{114}$$

where

$$\int_{\overline{AB}} \frac{\partial \phi}{\partial n} dl \approx \frac{s_i}{2} \frac{\phi_{i,j-1} - \phi_{ij}}{t_{j-1}}. \tag{115}$$

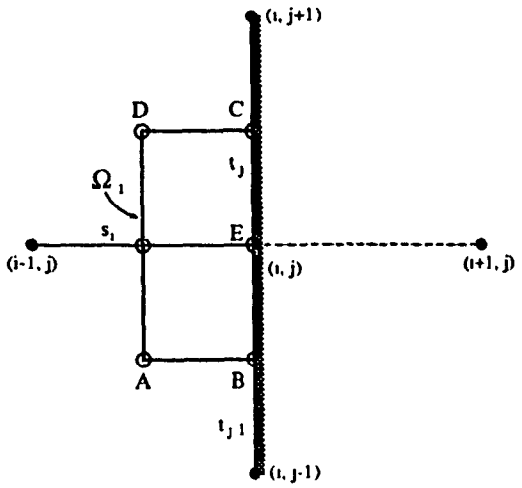


Figure 65: The boundary difference node (i, j).

$$\int_{\overline{CD}} \frac{\partial \phi}{\partial n} dl \approx \frac{s_i}{2} \frac{\phi_{i,j+1} - \phi_{i,j}}{t_j}. \tag{116}$$

From the Neumann condition (Eq. 26),

$$\int_{\overline{BE}} \frac{\partial \phi}{\partial n} dl = \int_{\overline{EC}} \frac{\partial \phi}{\partial n} dl = 0. \tag{117}$$

Also the integral $\int_{\overline{DA}} \frac{\partial \phi}{\partial n} dl$ is given in Eq. 108. Therefore, we obtain the difference equation on the boundary node (i, j) when (i, j) is neither S^+ nor S^-

$$\frac{t_j + t_{j-1}}{2} \frac{\phi_{i-1,j} - \phi_{i,j}}{s_{i-1}} + \frac{s_i}{2} \left(\frac{\phi_{i,j+1} - \phi_{i,j}}{t_j} + \frac{\phi_{i,j-1} - \phi_{i,j}}{t_{j-1}} \right) = 0. \tag{118}$$

When S^+ or S^- are on the boundary node (i, j), then

$$\frac{t_j + t_{j-1}}{2} \frac{\phi_{i-1,j} - \phi_{i,j}}{s_{i-1}} + \frac{s_i}{2} \left(\frac{\phi_{i,j+1} - \phi_{i,j}}{t_j} + \frac{\phi_{i,j-1} - \phi_{i,j}}{t_{j-1}} \right) = \pm 1. \tag{119}$$

For the equi-distance grid structure, i.e.,

$$s_i = t_i = h_x = h_y \quad \forall (i, j), \tag{120}$$

we can simplify Eq. 110, Eq. 113, Eq. 118 and Eq. 119 as

$$4\phi_{i,j} - (\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}) = 0, \tag{121}$$

$$4\phi_{ij} - (\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}) = \pm 1, \tag{122}$$

and

$$2\phi_{ij} - (\phi_{i-1,j} + \frac{1}{2}\phi_{i,j+1} + \frac{1}{2}\phi_{i,j-1}) = 0, \tag{123}$$

$$2\phi_{ij} - (\phi_{i-1,j} + \frac{1}{2}\phi_{i,j+1} + \frac{1}{2}\phi_{i,j-1}) = \pm 1. \tag{124}$$

The above techniques using box integration method can be found from Varga [Var62], and are developed into the finite volume element method in [McC89].

6.2 Stopping Condition for Relaxation Methods

Iteration methods are generally employed to approximate the solutions of Eq. 23. On the other hand, fluid simulation is to obtain a stable flow pattern in Ω so that it can be used as the guidance of path planning and motion control. Therefore, any solution will do if and only if it represents well a stable situation. It is found by Huang [Hua64] that the difference of any two sets of solutions for Eq. 23 is constant. By taking the constant off, we actually achieve the same result. Therefore, it is very important to choose a proper stopping condition for the iteration. By this condition, the correct solution (i.e., one in the set of the solutions) is obtained while the time spent on computation is as short as possible. For example, we take the residual r as the stopping condition for relaxation (see [Way59]). At the $(k+1)$th iteration,

$$r^{(k)} = ||\phi^{(k+1)} - \phi^{(k)}||_\infty = max_{1\le i\le n}^{(k)} |\phi_i^{(k+1)} - \phi_i^{(k)}|, \tag{125}$$

where $\phi^{(k)}$ is the solution of ϕ at the kth iteration and n is the total number of elements. A small threshold μ will be chosen so that when

$$r^{(k)} \le \mu, \tag{126}$$

the computation terminates.

Also we can define r as the ratio of the current residual ($r^{(k)}$) and the initial residual (r_{init}),

$$r = \frac{r^{(k)}}{r_{init}}, \tag{127}$$

and check the stopping condition similarly to Eq. 126. In general, by using Eqs. 125-126, μ chosen as 5×10^{-3} to 5×10^{-4} is sufficient to achieve a proper solution in Ω with a grid of not more than 256×256, and by using Eq. 127, μ is between 1×10^{-4} to 1×10^{-5}.

Below, we shall discuss several relaxation methods and their experimental results in the fluid simulation.

6.3 Jacobi's Method

Jacobi's method is a basic method for solving Eq. 23. Let us write the interior and boundary difference equations in Section 8.2 in a general form

$$a_{i,j}\phi_{i,j} + b_{i,j}\phi_{i+1,j} + c_{i,j}\phi_{i-1,j} + d_{i,j}\phi_{i,j+1} + e_{i,j}\phi_{i,j-1} = b_{i,j}, \tag{128}$$

where the coefficients and constants satisfy

$$b_{i,j} = 0, 1 \text{ or } -1, \tag{129}$$

$$a_{i,j} > 0, \quad b_{i,j} \leq 0, \quad c_{i,j} \leq 0, \quad d_{i,j} \leq 0, \quad e_{i,j} \leq 0, \tag{130}$$

and

$$a_{i,j} = -(b_{i,j} + c_{i,j} + d_{i,j} + e_{i,j}). \tag{131}$$

Then Jacobi's method is defined by

$$\phi_{i,j}^{(k+1)} = -\frac{1}{a_{i,j}}\left[b_{i,j}\phi_{i+1,j}^{(k)} + c_{i,j}\phi_{i-1,j}^{(k)} + d_{i,j}\phi_{i,j+1}^{(k)} + e_{i,j}\phi_{i,j-1}^{(k)} - b_{i,j}\right], \tag{132}$$

where $\phi_{ij}^{(0)}$ are the initial guess, e.g.,

$$\phi_{i,j}^{(0)} = 0 \quad \forall(i,j). \tag{133}$$

6.4 Gauss-Seidel Method

Gauss-Seidel's method is not only faster in convergence than Newton's method, but also requires less storage space. It operates on the same matrix, therefore, no copies are needed.

For the rate of convergence, Gauss-Seidel method is superior to Newton method. From Hagement and Young, Newton has $1/2\pi^2h^2$ and GS has only $2\pi h$. From Hagemen and Young [HY81], the formula for Gauss-Seidel is

$$\phi_{i,i}^{(k+1)} = \frac{1}{a_{i,i}}[b_i - a_{i,1}\phi_1^{(k+1)} - a_{i,2}\phi_2^{(k+1)} - \ldots - a_{i,i-1}\phi_{i,i-1}^{(k+1)} - a_{i,i+1}\phi_{i+1}^{(k)} - \ldots - a_{i,n}\phi_n^{(k)}]. \tag{134}$$

It can be rewritten as

$$A_{i,i}\Phi_i^{(k+1)} = -\sum_{j=1}^{i-1} A_{i,j}\Phi_j^{(k+1)} - \sum_{j=i+1}^{q} A_{i,j}\Phi_j^{(k)} + F_i, \tag{135}$$

Or in matrix form,

$$(D - C_L)\phi^{(k+1)} = C_U\phi^{(k)} + b, \tag{136}$$

where D is the diagonal matrix of A, and C_L and C_U are the lower triangular and upper triangular matrices. Eq. 136 can be also rewritten as

$$\phi^{(k+1)} = \mathcal{L}\phi^k + d, \tag{137}$$

where

$$\mathcal{L} = (I - L)^{-1}U,\ d = (I - L)^{-1}D^{-1}b, \tag{138}$$

and where

$$L \equiv D^{-1}C_L,\ U \equiv D^{-1}C_U. \tag{139}$$

$\mathcal{L}$ is the iteration matrix.

6.5 Successive Overrelaxation Method

To achieve a faster convergence rate, we combined Gauss-Seidel method with the successive over-relaxation method (SOR). Then Eq. 135 can be written as

$$A_{i,i}\Phi_i^{(k+1)} = \omega\left\{-\sum_{j=1}^{i-1} A_{i,j}\Phi_j^{(k+1)} - \sum_{j=i+1}^{q} A_{i,j}\Phi_j^{(k)} + F_i\right\} + (1-\omega)A_{i,i}\Phi_i^{(k)}, \tag{140}$$

where ω is the relaxation factor. When $\omega > 1$, Eq. 140 is the over-relaxation, while when $\omega < 1$, it is the underrelaxation. If $\omega = 1$, Eq. 140 reduces to Gauss-Seidel. To accelerate the convergence rate, we choose $w > 1$.

Eq.140 can also be expressed in matrix form:

$$D\phi^{k+1} = \omega(C_L\phi^{(k+1)} + C_U\phi^{(k)} + b) + (1-\omega)D\phi^{(k)}. \tag{141}$$

Or rewritten as

$$\phi^{(k+1)} = \mathcal{L}_\omega\phi^{(k)} + d_\omega^{(F)}, \tag{142}$$

where $\mathcal{L}$ is the iteration matrix for SOR and

$$\mathcal{L}_\omega = (I-\omega L)^{-1}(\omega L) + (1-\omega)I), \tag{143}$$

and

$$d_\omega^{(F)} = (I-\omega L)^{-1}\omega D^{-1}b. \tag{144}$$

The relaxation factor

$$\omega = \frac{2}{1-\sqrt{1-M(B)^2}}. \tag{145}$$

where $M(B) = Max_{1\le i\le n}\lambda_i$, and λ is the eigenvalues of $\mathcal{L}_\omega$. $M(B)$ here is the spectral radius of $\mathcal{L}_\omega$. From [HY81],

$$M(B) = \frac{(\lambda+\omega-1)}{\omega\sqrt{\lambda}} < 1. \tag{146}$$

Therefore, we can get the optimal relaxation factor by (see [Li80])

$$\omega_{opt} \approx \omega_{opt}^{(k)} = \frac{2}{1+\sqrt{1-\left(\frac{\lambda_1^{(k)}+\omega^{(k)}-1}{\omega^{(k)}\sqrt{\lambda_1^{(k)}}}\right)^2}}. \tag{147}$$

6.6 Multigrid Adaptive Method (MAM)

The time complexity of fluid simulation can be further decreased by using advanced techniques such as the multigrid method. This method processes the computation on a multigrid structure, and runs the iterations at each level either sequentially or parallelly. Fig. 66 shows a four level multigrid. As we stated in Section 2, Ω is solved by Poisson's equation (Eq. 24) with Neumann boundary conditions (Eq. 26). Therefore, for each level of multigrids, both the equation and the condition must be

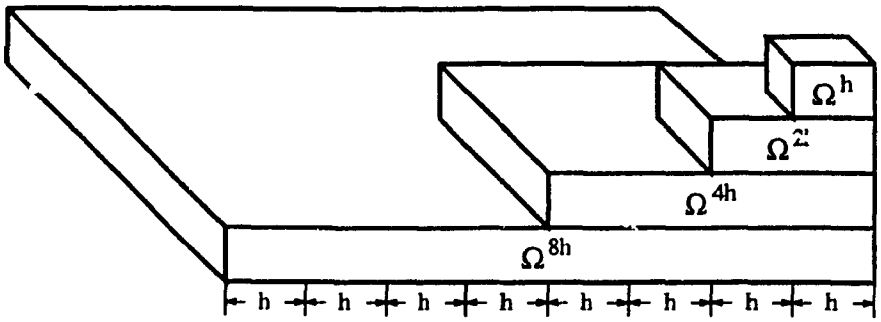


Figure 66: A multigrid structure with four levels.

satisfied. For simplicity, we assume the sources (i.e., S^+ and S^-) are on the shared nodes of multigrids, i.e., on the nodes of the coarsest grid.

From [McC89], the maximum number of levels on multigrids can be $n+1$, for

$$n = \log M. \tag{148}$$

Obviously, it is not applicable to use $n+1$ levels in Ω, because the coarsest level is only a point. For the presence of obstacles, the computable regions in Ω generally have an irregular shape. Some narrow channels appear in a finer grid may totally disappear in a coarser grid! (See Fig. 67). This phenomenon causes more relaxations on the finer grid so as to achieve the required accuracy. Hence the performance of the multigrid method is greatly degenerated. To prevent this from happening, the number of levels of coarser grids should be limited. For the choice of the degree of coarseness in the multigrid method, it is better to consider the maximum width of the predefined robots. Let four levels of multigrid is chosen. Then if M^h, the size of the finest grid, is 256, we have the sizes for other grids as follows,

$$M^{2h} = 128,\ M^{4h} = 64,\ M^{8h} = 32. \tag{149}$$

We adopted the following strategies for MAM:

1. Four levels of multigrids are used, which are denoted by Ω^h, Ω^{2h}, Ω^{4h} and Ω^{8h}.

2. S^+ and S^- are assigned on the coarsest grid so that they appear in all the grids.

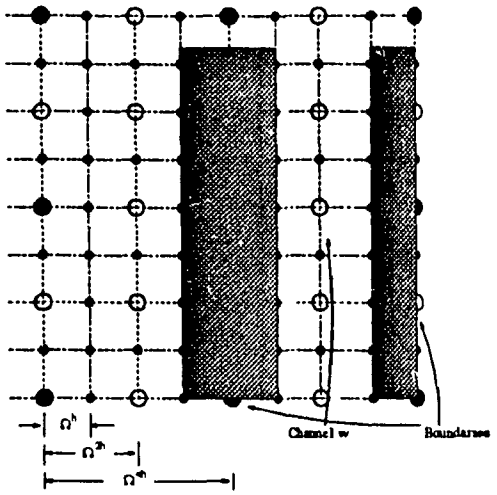


Figure 67: Part of the three levels of multigrid with boundaries. The big black nodes are those on Ω^{4h}, the big blank nodes on Ω^{2h} and the small dark nodes are on the finest grid Ω^h. Notice, channel w does not appear in Ω^{4h}.

3. The irregular inner boundaries are preprocessed to comply with the structure of each grid.

4. Gauss-Seidel relaxation is used on each grid, and the coefficient A is precomputed so that we have A^h, A^2h, A^4h and A^8h at hand.

5. The organization of relaxations on multigrids is V-cycle (see [McC89]). They are processed in the following order: Ω^h, Ω^{2h}, Ω^{4h}, Ω^{8h}, Ω^{4h}, Ω^{2h}, Ω^h repetitively.

6. The transgrid linear interpolation and error correction appear at the backward process of the V-cycle.

7. The same stopping condition as in Section 8.2 is adopted.

6.7 Experiment Results

We have experimented on each relaxation method for fluid simulation in Ω. All methods are good for the simulation while MAM gives the most satisfactory results. Its time complexity is close to $O(n \log n)$ (see Table 2 and Fig. 68). Table 2 shows

Table 3: The comparison of the performances of different relaxation methods in fluid simulation.

M	r	NEWTON		SOR		MAM	
		I.N.	T(sec.)	I.N	T(sec.)	I.N	T(sec.)
64	1×10^{-3}	162	8.51	144	4.3	22	1.1
	5×10^{-4}	752	23.0	178	5.0	88	4.0
128	1×10^{-3}	160	25.9	62	17.4	10	2.8
	5×10^{-4}	318	46.6	68	18.4	41	8.3
256	1×10^{-3}	160	139.7	62	96.7	8	16.1
	5×10^{-4}	318	325.1	68	101.0	12	18.4

the comparisons in the performances of Newton, SOR and MAM. Fig. 68 shows the performance behavior of each method in Table 3, where the solid line is for the Newton method, the dashed line for SOR and the dashdoted line for MAM. In this table, the intervals for optimal successive factor selection in SOR is 15 iterations except that it is 10 in the case of M = 64.

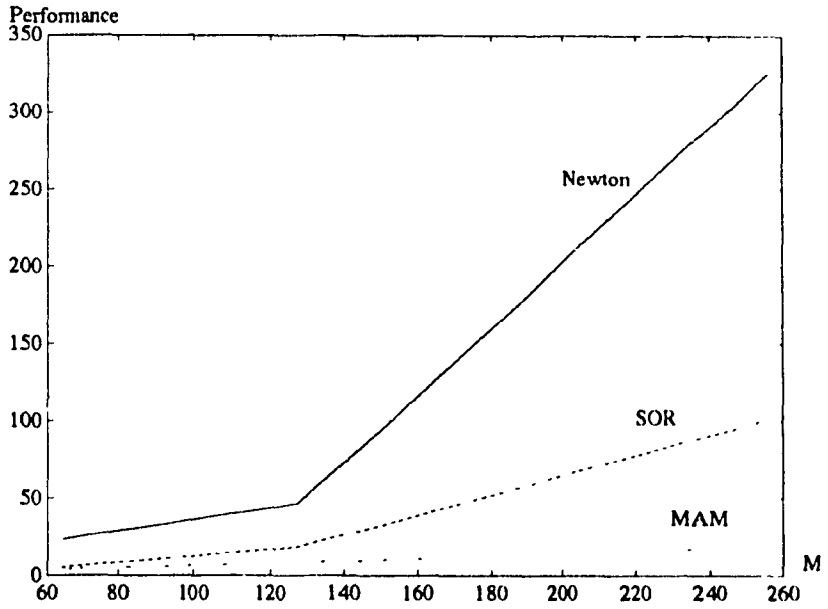


Figure 68: The curves of the performances by different relaxation methods in second (in y-direction) versus the increase of M (in x-direction).

Chapter 7

Parallel Processing

In recent years, parallelism becomes more and more popular in computer architecture and computation technology. Using parallel processing can break the time constraints caused by hardware and substantially speed up computer operations. In light of the parallel processing of this fluid model, we have done experiments in two aspects. First, in fluid simulation, large matrices are used for information storage. They are good structures for parallel computation. We reordered the mesh points (nodes) in the matrix and adopted block iteration to solve it (see [HY81]). Second, we ran this model on strong parallel computational facilities such as a supercomputer for enhancing its performance (see [LBT92]). In this section, we will talk about the ideas and experiments on the two issues. Also, we will talk about decoupled planning techniques in multiple robot path planning and navigation using this model, which, if supported by parallel means, can greatly improve its performance (also see [LBT92]).

7.1 Block Iteration and Parallel Computation

As we stated in Chapter 2, the simulation of the ideal fluid in Ω is realized by computing Poisson's equation with finite difference method. We solve,

$$Ax = f, \tag{150}$$

where A is the coefficient matrix, x is an unknown vector and f is the known vector. For finite difference method, the five-point formula for a mesh point is

$$a_{i,j}x_{i,j} - b_{i,j}x_{i-1,j} - c_{i,j}x_{i,j+1} - d_{i,j}x_{i+1,j} - e_{i,j}x_{i,j-1} = -b_{i,j}. \tag{151}$$

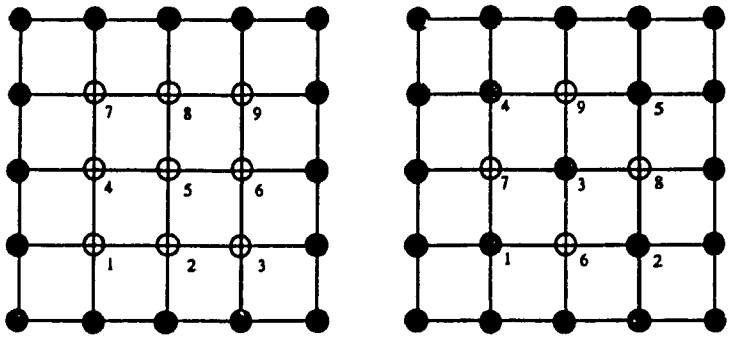
Assume A is a square matrix with the size of $M \times M$. For Eq. 150, the block iteration method can be used by first converting it into the form

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,q} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,q} \\ \cdots & \cdots & \cdots & \cdots \\ A_{q,1} & A_{q,2} & \cdots & A_{q,q} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_m \end{bmatrix}. \tag{152}$$

Here, A is divided into $q \times q$ blocks, and each block has the same size. Now, Eq. 152 makes relaxation processing possible through q-level parallel processing.

For converting Eq. 150 to Eq. 152, reordering the mesh points of the concerned domain is necessary. For example, when $q = 2$, we have the classical *red/black ordering*. Fig. 69 shows a simple example with a mesh of only 5×5. This is a closed domain. The dark nodes are in the boundary, i.e., the subset of $\mathcal{B}$, and all other nodes are computational nodes, i.e., subset of $\mathcal{F}$ or $\partial\mathcal{F}$. Fig. 69a shows the normal ordering of the mesh points and Fig. 69b the red/black ordering. From Eq. 24 and Eq. 26, the Neumann boundary condition, Eq. 150 for Fig. 69a may be written as

$$\begin{bmatrix} 1 & -\frac{1}{2} & 0 & -\frac{1}{2} & & & & & \\ -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & -1 & & 0 & & \\ 0 & -\frac{1}{2} & 1 & 0 & 0 & -\frac{1}{2} & & & \\ -\frac{1}{2} & 0 & 0 & 2 & 1 & 0 & -\frac{1}{2} & & \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & \\ & & -\frac{1}{2} & 0 & -1 & 2 & 0 & 0 & -\frac{1}{2} \\ & & & -\frac{1}{2} & 0 & 0 & 1 & -\frac{1}{2} & 0 \\ & & 0 & & -1 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} \\ & & & & & -\frac{1}{2} & 0 & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{bmatrix}. \tag{153}$$



a. b.

Figure 69: The ordering of mesh points for point partitioning. a). natural ordering. b). Red/black ordering.

If we reorder the points in the mesh as in Fig. 69b, Eq. 150 may be like this,

$$\begin{bmatrix} 1 & & & & & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ & 1 & & & & -\frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ & & 4 & & & -1 & -1 & -1 & -1 \\ & & & 1 & & 0 & -\frac{1}{2} & 0 & -\frac{1}{2} \\ & & & & 1 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & -1 & 0 & 0 & 2 & & & \\ -\frac{1}{2} & 0 & -1 & -\frac{1}{2} & 0 & & 2 & & \\ 0 & -\frac{1}{2} & -1 & 0 & -\frac{1}{2} & & & 2 & \\ 0 & 0 & -1 & -\frac{1}{2} & -\frac{1}{2} & & & & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{bmatrix}. \tag{154}$$

Now, the block division is very clear. Hence, for simplicity, the red/black point partitioning for Eq. 150 can be rewritten as

$$\begin{bmatrix} D_R & H \\ H^T & D_B \end{bmatrix} \begin{bmatrix} u_R \\ u_B \end{bmatrix} = \begin{bmatrix} b_R \\ b_B \end{bmatrix}. \tag{155}$$

where

$$D_R = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 4 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \text{ and } D_B = \begin{bmatrix} 2 & & & \\ & 2 & & \\ & & 2 & \\ & & & 2 \end{bmatrix}. \tag{156}$$

and

$$H = \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ -1 & -1 & -1 & -1 \\ 0 & -\frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}. \tag{157}$$

Obviously, two processors can be used to compute Eq. 155 parallelly. This method can be expanded to divide a matrix into many more blocks with the number of blocks equal to 2^n, $n = 1, 2, ...$

7.2 Real-time Performance in Supercomputing

Choosing an advanced computation technique to improve the performance of this model is very important. On the other hand, the computational facility, i.e., the hardware support, is another crucial factor for real time performance of this model. It is an ideal way to be able to connect the robot to a high speed computer. Luckily, we have opportunity to try our program on NEC SX3/44, a supercomputer with 22 giga FLOPS. SX3 is basically a vector machine and it has four CPU's, each of which contains a powerful pipe set with 256 processors. It is highly facilitated for large matrix computations. In this experiment, the relaxation technique used for simulation is SOR, which is stated in Section 8.5 for this experiment. The result is tabulated in Table 4.

In the table, the performances on Sparc-station (SUN4.1 with 16 MFLOPS) are cited for comparison with those on NEC SX3 in different M and different μ. Here, M is the size of the matrix to be solved and μ is the stopping condition stated in Eq. 127

Table 4: The comparisons of the performances of the same program on Sparc-station and NEC SX3 with different m and R.

M	μ	Sparc (in second)	NEC SX3 (in second)	Speed Up
64	5×10^{-3}	1.6	0.2	8.0
	1×10^{-3}	23.3	1.3	18.1
	5×10^{-4}	50.1	1.9	26.3
	1×10^{-4}	148.4	4.2	35.3
128	5×10^{-3}	8.8	0.8	9.1
	1×10^{-3}	31.8	3.7	9.6
	5×10^{-4}	116.7	7.8	14.9
	1×10^{-4}	993.8	32.2	30.7
256	5×10^{-3}	30.1	3.3	9.1
	1×10^{-3}	147.8	15.3	9.6
	5×10^{-4}	283.3	30.3	9.3
	1×10^{-4}	1890.7	75.5	25.0
512	5×10^{-3}	112.2	13.6	8.2
	1×10^{-3}	522.4	58.2	8.9
	5×10^{-4}	1764.8	81.7	21.6
	1×10^{-4}	6359.7	147.7	43.0

in Section 9.2. The rates of performance speedup by using NEC SX3 are calculated in the table. The conclusion from the experiments on NEC SX3 is obvious. By the strong support of a supercomputer, the differential equation in the fluid model can be solved in a second or two, therefore, this model is feasible for robot path planning and navigation in real time, provided that we restrict $M \leq 256$ in two dimensions and choose $\mu \leq 1 * 10^{-3}$. In practice, $M = 256$ is large enough for general working domains.

7.3 Parallel Processing for Multiple Robots

Assume there are robots $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n$ working in the same world space $\mathcal{W}$ and they are projected to the same C-space, Ω of dimension m. In any instant, a robot $\mathcal{A}_i$ must be either at its initial position, $q(S_i^+)$, or at its goal position, $q(S_i^-)$, or on its

path, τ_i. For finding collision free paths for all the robots in Ω, we are dealing with $m \times n$ dimensions. It is inapplicable when n is large, for this is an exponential-hard problem with dimensions (see [Can88]).

7.3.1 Reduction of Dimensions in Ω

We can reduce the dimension by mapping the configuration of each robot into separate configuration spaces. That is, for all $\mathcal{A}_i \in \mathcal{A}$, there exists $\Omega_i \in \Omega$ such that $\mathcal{A}_i(q) \in \mathcal{F}_i$, $0 \leq i \leq n$. In the same manner, each Ω_i consists of two types of regions, $\mathcal{F}_i$ and $\mathcal{B}_i$. Let τ_i be a global path for $\mathcal{A}_i$ in $\mathcal{F}_i$, then

$$\tau_i : [0,k] = \{q_i \in \mathcal{F}_i \mid \mathcal{A}_i(0) = q(S_i^+) \cup \sum_{j=1}^{k-1} \mathcal{A}_i(q_j) \cup \mathcal{A}_i(k) = q(S_i^-)\}, \tag{158}$$

where q_j is the configuration of $\mathcal{A}_i$ at step j. And

$$\mathcal{F}_i = \overline{\Omega_i \cap \mathcal{B}}. \tag{159}$$

$$\mathcal{B} \xleftarrow{T_i} \bigcup_{j=1}^{m} B_j, \text{ for } i = 1,2,...,n, \tag{160}$$

where B_j are stationary obstacles in $\mathcal{W}$ and T_i is the transformation mapping $\cup_{j=1}^{m} B_j$ into Ω_i. Eq. 160 implies that all the Ω_i have the same number of stationary obstacles with the same relative locations (However, the exact boundaries of $\mathcal{B}$ in different Ω_i are dependent on the shapes of $\mathcal{A}_i$ (see [Lat91]).

When we process each Ω_i independently, we only face a dimension of m. Thus we sacrifice storage for achieving lower dimensions. However, this tradeoff is particularly significant in improving the performance when parallel and vector processing are used.

7.3.2 Decoupled Planning

In the presence of multiple robots in $\mathcal{W}$, proper interactions between the robots are very important for obtaining a collision-free path. This technique depends heavily on information exchange (updating) between the configuration spaces. It is called *decoupled planning*. Each robot $\mathcal{A}_i$ has its own Ω_i for its path planning. Ω_i is independent

of Ω_j for $1 \leq i, j \leq n$, $i \neq j$. The path τ_i for $\mathcal{A}_i$ must be in $\mathcal{F}_i$ and

$$\mathcal{F}_i = \overline{\Omega_i \cap (\mathcal{PB}_i \cup \mathcal{TB}_i)}, \tag{161}$$

and

$$\mathcal{PB}_i \xleftarrow{T_i} \bigcup_{j=1}^{m} B_j, \tag{162}$$

$$\mathcal{TB}_i \xleftarrow{T_i} \bigcup_{k=1, k\neq i}^{n} \mathcal{A}_k. \tag{163}$$

We call $\mathcal{PB}_i$ *permanent obstacles* in Ω_i and $\mathcal{TB}_i$ *temporary obstacles* in Ω_i which are marked differently.

Eq. 161 to Eq. 163 state that in dynamic motion planning for multiple robots, if $\mathcal{A}_i$ has taken a step forward, its new position has to be projected in all Ω_i with $i \neq k$. This means updating each Ω_i, $i = 0, 1, ..., n$, by T_i, is required at each step so that when it is time for $\mathcal{A}_i$ to plan its path, the region of $\mathcal{F}_i$ is exactly known (e.g., the coordinates of $\mathcal{PB}$ and $\mathcal{TB}$ in Ω). Fortunately, the updating processes are independent with each other by using individual functions T_i and requiring access to different memory locations (i.e., the space for $\Omega_1, \Omega_2, ..., \Omega_n$). This makes parallel processing possible to speed up the execution.

After all, parallel processing has great potential in enhancing the performance of fluid simulation and path planning of this model, especially, when multiple robots are involved.

Chapter 8

High Efficient Path Planning

In Chapter 6, various kinds of techniques in solving Poisson's Equation are discussed. Those techniques compute the solutions of the entire domain. Generally, the time spent on them by the current computation facilities is still too long in view of dynamic path planning. Therefore, these techniques are practical only when very powerful parallel machines are available, which are supposed to solve the differential equations in milliseconds.

On the other hand, the actual use of the solutions is generally limited to a small part of the domain which the robot's route may cross. To save computing time, one option is to avoid solving the differential equation completely, since our purpose of computation is to find the path. Whenever a path is found, further computation is not necessary. In this section, we have two proposals about how to effectively use the intermediate solutions of the fluid simulation in path planning. However, since the solution is not complete, and what we have simulated in the concerned domain is only an unsteady flow, some features of ideal fluid will be lost (see Chapter 2). Therefore, the solutions can hardly be reused.

Also in this chapter, we briefly talk about the use of potential gradients as reference for controlling a physical robot.

8.1 Sensor-based path planning

One technique we have experimented can achieve very high efficiency in planning the path for a point robot in 2-D invariant space. We call it the *Sensor-based path planning*. It is similar to [SN92], but with Poisson's Equation as the navigation function, which is combined with Neumann boundary conditions. Assume the robot and its destination are both in $\mathcal{F}$, and on a slope. The robot is at the highest altitude, and its destination is at the lowest altitude. Assume there is water coming out of the robot position (i.e., the source) , and flowing downward, it must finally reach the robot's destination sooner or later. Then we can install a sensor at the robot's destination. Whenever, it "tastes" the water (or senses that the value of potentials at the destination becomes nonzero), the path from the robot to its destination is found. According to this simple idea, some experiments have been done.

Poisson's Equation with Neumann conditions is still used for simulating the fluid in Ω. and the system of equations is formulated as follows:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = -f, \tag{164}$$

where

$$f(x,y) = \begin{cases} b & \text{if } (x,y) = G(x,y), \\ 0 & \text{otherwise.} \end{cases}$$

b is a positive constant, and $G(x,y)$ represents the goal position of the robot. The difference with what we stated in Chapter 2 is that there is no sink initialized. Therefore the fluid simulated in Ω is unsteady. However, since what is of concern to us here is the path of the robot, the simulation (or iterative computation) can be stopped right away after the path is found. This strategy makes the planning highly efficient. Now we use the intermediate solution of Poisson's Equation as the navigation function. The whole region may not satisfy the properties in Chapter 3, however, it can just be ignored, because this solution is used only once. For any next planning, the region will be recomputed. To exactly describe the formulas, the time factor should be introduced in the equation, thus we have,

$$\frac{\partial}{\partial t} \triangle \phi = -f, \tag{165}$$

where t shows the solution at certain instant (see [Way59]).

Clearly, the "water" sensor of the robot can be used as a stopping tool for the computation as well. Then we can use the same techniques stated in Chapter 4 to plan the path. The only case that will fail the planning is that no connection between the robot and its destination exists in $\mathcal{F}$. Then the program will run forever. One simple way to avoid this is to set up a time threshold. When the threshold is exceeded and still no reaction from the sensor, failure should be declared and computations terminated.

8.2 Controlled Simulation by Path Finder

Another approach to achieve highly efficiency is to integrate the simulation with a path finder. A path finder is an algorithm similar to SFM. It is invoked interactively with the computation of the navigation function. The navigation function is the same as stated in Chapter 2. After some number of iterations on the function, the path finder checks the intermediate solutions on the domain, and tries to find a path from the robot point to its destination point. If it succeeds, then the computation terminates. Otherwise, let the computation go on and wait for the next check. This process may be repeated for a few times before a path is found.

Although the time complexity of the path finder is constant with the number of grid nodes, it is still not economical if it is to be invoked at each iteration. The best way is to start the check after n iterations, and check it repetitively afterwards in regular bases. For example, invoke it when $n\%m = 0$, (% is a modular operation), where m is the number of iterations, which is the needed time before the change of the solutions becomes significant.

The time of solving could be reduced by half of that spent by the algorithm stated in [SN92] using diffusion equation. Although the value decreases on the grid is in the same manner, in the iteration of our method, values are added to the system at two points instead of one: the robot's initial position and its destination. This is important especially because the arithmetic representation of the values is machine-dependent. Truncation makes very small values become zero. A number of iterations

is needed before the value on a grid in a distance to the source or sink is significant. In our case, a positive value is added at the source while a negative value is added at the sink (see Eq. 25). Then when the two values meet on a grid node, our path planning algorithm based on deepest descending technique will find it, at the same time terminates the computation.

The time complexity of the above two algorithms is only related to the number of grid nodes it needs to pass in order to connect the robot to its goal. This measure of the distance in a grid is called L-metric in [SN92]. For experimenting the above algorithms, in order to reduce the damage of truncation, 64 bit floating point representation should be used for the potentials. This is because in the fluid simulation, five-point formula is used (see Section 1, Chapter 6), the reduction rate of the value on the concerned point to its closest neighborhood is quadratic. Assume that at point P, a value of r is given, after one iteration, its neighbor nodes with one L-metric distance to it, only a value of r^{-2} is added. If a node has n L-metric distance, then only r^{-2^n} will be added. Clearly, the time complexity of the above algorithms is not constant. However, they are time-efficient because, in general, the density of the grid does not need to be high.

8.3 Idea of Robot Speed Control

One very interesting discovery is that the potential gradients can be used for controlling the speed of a real robot's movement. Because of the limitation of the content of the thesis, we did not go deep into this aspect. However, we believe the following idea is worth being recorded here.

From Eq.16, the velocity of an object can be calculated from the potential gradients. However, The result from Eq.16 cannot be directly applied to robot navigation because when R is constant, the narrower the channel, the faster the fluid flows. On the contrary, considering the physical constraints for a real robot, we should allow it to move slower in narrow channel and faster in open space. Also at the beginning, the speed should be increased gradually and at the end, the robot should slow down so that it can station itself accurately at the destination. These requirements can

be inversely satisfied by the characteristics of ideal flow in Ω! Thus we can use the computed velocity potentials for the purpose of speed control of a real robot. For the speed v between two points, p and p_{prev}, we can have

$$v(p, p_{prev}) = \frac{\alpha}{|\phi(p) - \phi(p_{prev})|} = \frac{\alpha}{g(p, p_{prev})}. \tag{166}$$

where α is the velocity control factor. It can be obtained from experiments. In reality, upper and lower bounds of the robot's speed of navigation may be defined.

Chapter 9

Conclusion

Based on the ideal fluid in fluid mechanics, we have explored the theories and practices in using the flow of an ideal fluid for robot path planning and navigation in a closed 2-dimensional domain. They include the simulation of ideal fluids, the analysis of generic properties, the path finding configuration and the dynamics. A basic path planning algorithm (SFM) which seeks the path by following the natural flow of the fluid is given. Two other heuristic techniques are proposed to improve the natural flow path. We also propose a bitmap technique for collision avoidance. Computation techniques are systematically experimented for solving the partial differential equation-Poisson's equation, which is the core of the ideal fluid simulation. Also the experiments on a powerful supercomputer is performed. After all those experiments, we are confident in that, in using advanced parallel computation techniques and high-powered computational facilities, this model is promising in real-time performance.

On the other hand, just like all other numerical potential functions, solving path planning in this model requires simulation of fluid in the entire domain (i.e., all the robot's activity related area). Therefore, computation takes the lion share of the time consumed. The path planning algorithm itself on the function is linear and constant because it only involves the evaluation of the gradient of the solutions of the potential functions. It takes far less than a second. In real time performance, this model is quite suitable for a static situation, where the goal point and obstacles are fixed. Then the solution may be reused again and again. It does not need to be recomputed unless

the obstacles and goal points change position. However, in a dynamic situation, the solution must be recomputed whenever the configuration of the environment changes.

Practically, the potential function may not need to be completely solved. We can use its intermediate (or incomplete) solution to plan the path. This technique can greatly reduce time in computation, and make this model possible to be used directly in dynamic situations.

There are some features of the fluid model remained to be studied. For example, the adjacent streamlines (i.e., the natural paths) tend to be parallel to each other provided they are not close to the source, the sink or the boundaries (see Section 2.2.3). Thus they actually subdivide the free space evenly into a few separate open channels (the width of the channel is related to the resolution of the discretization of the domain). Obviously, this parallel characteristics can help collision avoidance for multiple robots when they meet and surpass in a time-invariant environment. Another example is that the velocity gradients computed in the fluid simulation can be used for the speed control of a physical robot. In fact, they are inversely proportional to the speed requirements of a robot's movement (see Chapter 8). Besides, our further researches could, at least, be in the following directions:

- Motion planning for 3-D objects using the fluid model.
- Fluid simulation and motion planning in a closed domain with multiple sources and sinks.
- Motion planning in an open domain with a fluid model.

References

[BL89] B Barraquand, J. Langlois and J. Latombe. Numerical potential field technique for robot path planning. Technical report, Dept. of Computer Science, Stanford University, 1989. ReportNo. STAN-CS-89-1285.

[Bro83] R.A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(3), 1983.

[BS85] I. N. Bronshtein and K. A. Semendyayev. *Handbook of Mathematics.* Van Nostran Reinhold Company, New York, 1985.

[BT83] R.A. Brooks and Lozano-Pérez. T. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the 8th International Conference on Artificial Intelligence*, Karlsruhe, FRG, 1983.

[Can88] J.F. Canny. *The Complexity of Robot Motion Planning.* PhD thesis, 1988.

[CJ90] C.I. Connolly and Burns J.B. Path planning using laplace's equation. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1990.

[Don84] B.R. Donald. Motion plannig with six degrees of freedom. Technical report, Artificial Intelligenc Laboratory, MIT, 1984. Report No. AI-TR-791.

[DT88] L. Dorst and K. Trovato. Optimal path planning by cost wave propagation in metric configuration space. In *Proceedings of SPIE - The International Society for Optical Engineering, Mobile Robots III*, volume 1007, 1988.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry, Monographs on Theoretical Computer Science.* Springer-Verlag, Berlin, 1987.

[HA92] Y.K Hwang and N. Ahuja. Cross motion planning – a survey. *ACM Computing Survey*, 1992.

[Hua64] H. C. Huang. On numerical solutions of neumann problems of elliptic equations (in chinese). *Applied Mathematics and Numerical Mathematics*, 1, No. 2, 1964.

[HY81] Louis A. Hageman and David M. Young. *Applied Iterative Methods.* Computer Science and Applied Mathematics. Academic Press, Inc., San Diego, 1981.

[K.84] Jänich, K. *Topology.* Springer-Verlag, New York, 1984.

[KB91] Y. Koren and J. Borenstein. Potential firld methods and their inherent limitations for mobile robot navigation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1398–1404, 1991.

[Kha80] O. Khatib. *Commande Dynamique dans l'Espace Opérationnel des robots manipulateurs en présence d'Obstacles.* PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'espace, Toulouse, 1980.

[Kha85] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 500–505, March 1985.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

[LBT92] Z. X. Li, T.D. Bui, and L. Tao. Real time robot motion planning and navigation – fluid model and supercomputing. In *Proceedings of Supercomputing Symposium'92*, Montréal, June 1992.

[Li80] Z. C. Li. The conservative difference schemes of a sort of nonlinear heat conduction equation and the choice of "optimum" relaxation parameter in sor-newton method. June 1980. (in Chinese).

[LP83] T. Lozano-Pérez. Spatial planning: A configuration space approach. In *IEEE Transactions on Computers*, volume C-32(2), 1983.

[LPW79] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. In *Communications of the ACM*, volume 22(10), 1979.

[Mas90] B.S. Massey. *Mechanics of Fluid.* Chapman and Hall, six edition, 1990.

[McC89] S.E. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations, Frontiers in Applied Mathematics.* SIAM, Philadelphia, 1989.

[Mil70] J. Milnor. *Morse Theory volume 51 of Annals of Mathematics Studies.* Princeton University Press, Princeton, New Jersey, 1870.

[Nil69] N.J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *The 1st International Joint Conference on Artificial Intelligence*, pages 509-520, Washington D.C., 1969.

[OY82] C. Ó'Dúlaing and C.K. Yap. A retractioin method for planning the motion of a disc. *Journal of Algorithms*, 6:1104-1111, 1982.

[RK88] E. Rimon and D. E. Koditschek. Exact robot navigation using cost functions: The case of distinct spherical boundaries in e^n. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1791-1796, April 1988.

[Roy88] D.N. Roy. *Applied Fluid Mechanics.* Ellis Norwood Limited, chichester, 1988.

[SN92] G. Schmidt and W. Neubauer. High-speed robot path planning in time-varying environment employing a diffusion equation strategy. *Robotics Systems Edited by S.G. Tzafestas*, 1992.

[SS83] J.T. Schwartz and M. Sharir. On the piano movers' problem: I. the case if a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[TB91] L. Tarassenko and A. Blake. Analogue computation of collision-free paths. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 540-545, 1991.

[Van83] J.S. Vandergraft. *Introduction to Numerical Computation.* Academic Press, Inc., Montréal, New York, second edition, 1983.

[Var62] R.S. Varga. *Matrix Iterative Analysis.* Prentice-Hall, Englewood Cliffs, New York, 1962.

[Way59] Harold Wayland. *Differential Equations Applied in Science and Engineering.* D. Van Nostrand Company, Inc., 1959.

[ZL91] D. Zhu and J. C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7, No. 1, February 1991.