ORNL/TM-12410

Robot Navigation in Unknown Terrains: Introductory Survey of Non-Heuristic Algorithms

Nageswara S.V. Rao

Srikumar Kareti
Weimin Shi
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

S. Sitharama Iyengar
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803

DATE PUBLISHED — July 1993

Research sponsored by the
Office of Basic Energy Sciences
U. S. Department of Energy
and
Robotics and Machine Intelligence Program
National Science Foundation

Prepared by the
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
managed by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-84OR21400

Contents

List of Figures

List of Tables

Acknowledgements

The support of this work by Oscar Manley of the Basic Energy Sciences Program in the Department of Energy is acknowledged by the first author. The support of Howard Moraff of Robotics and Machine Intelligence Program at the National Science Foundation is acknowledged by the first three authors. The comments of Reinhold Mann, E. M. Oblow, David B. Reister and Judd P. Jones of the Oak Ridge National Laboratory on various versions of the manuscript are appreciated. Part of this work was performed while the first author was with Old Dominion University, Norfolk, Virginia. The encouragement and support of Kurt J. Maly, Chair of the Department of Computer Science, Old Dominion University is acknowledged.

Abstract

A formal framework for navigating a robot in a geometric terrain populated by an unknown set of obstacles is considered. Here the terrain model is not a priori known, but the robot is equipped with a sensor system (vision or touch) employed for the purpose of navigation. Our focus is restricted to the *non-heuristic algorithms* which can be theoretically shown to be correct within a given framework of models for the robot, terrain and sensor system. These formulations, although abstract and simplified compared to real-life scenarios, provide foundations for practical systems by highlighting the underlying critical issues. First, we consider the algorithms that are shown to navigate correctly without much consideration given to the performance parameters such as distance traversed, etc. Second, we consider non-heuristic algorithms that guarantee bounds on the distance traversed or the ratio of the distance traversed to the shortest path length (computed if the terrain model is known). Then we consider the navigation of robots with very limited computational capabilities such as finite automata, etc.

1 Introduction

As the robot and computer technologies progress into the next century, more and more tasks are likely to be performed by autonomous machines. In particular, mobile robots could be employed to perform a variety of operations including (a) tasks in environments that are not suitable for human operation, e.g., nuclear plants and waste sites, chemical and toxic industries, (b) monotonous and tedious tasks such as parts delivery and movements in manufacturing plants, and (c) operations such as extra-terrestrial and underwater explorations, etc. One of the basic components in the operation of such robots is the capability to autonomously navigate in terrains; particularly in exploratory applications, the robots must deal with terrains whose models at best are only partially-known. Advances in various areas such as engineering, computer science, applied mathematics, etc., are required to fully achieve such autonomous navigation capabilities. In this survey we consider computer science aspects of navigational methods in unknown terrains from an algorithmic perspective.

The area of robot path planning and navigation has been studied by various researchers over the last decades, resulting in a large number of works. Several aspects of this fascinating area can be found in the recent book by Latombe [42] and the survey paper by Hwang and Ahuja [33]. There are two basic formulations of the path planning and navigation problem based on the availability of the terrain model. In a *known terrain*, the terrain model is given as input, and the motion planning problem becomes one of geometric programming; there are a large number of techniques proposed to solve this problem (see Latombe [42], Sharir [76]). In an *unknown terrain*, the terrain model is not known [1] but the robot obtains local terrain information by employing a sensor (vision or touch) system; several works of this formulation are described in Hwang and Ahuja [33]. One of the fundamental differences between these formulations is that a path can be preplanned in the former, whereas in the latter a path must be incrementally computed as the newer parts of the terrain are explored. To illuminate the differences between these two formulations consider an example of a human being required to move from one location to a destination location (e.g. main office or a vending room or an exit) in an unfamiliar building. If the floor plan of the building is given, one can plan a path and move along the path. On the other hand, if no floor plan is available, one must systematically search the building for the destination, say, by using visual information. Further, this problem becomes harder if the interior of the building is dark and the human being does not have any light sources; then, one has to rely on touch sensing alone.

In an unknown terrain, we have two critical aspects: (a) the computation is based on local (or partial) information, and (b) sensing is an integral part of the navigation. Because of the first aspect, the algorithms for unknown terrains are often called *online* algorithms (Kleinberg [41] and Bar-Eli et al [4]). In the light of second aspect, an algorithm for an unknown terrain is required to schedule the sensor operations, and this aspect is absent in known terrains. Moreover, in unknown terrains differ-

[1] In some known terrain cases, where the terrain model is large but the robot navigates within a restricted locality, it might be advantageous to consider only certain parts of the terrain; in such cases algorithms for unknown terrains could be suitably employed.

ent classes of algorithms are, in general, required to navigate robots equipped with different types of sensors.

In this survey, we consider a very specialized class of methods for the unknown terrains navigation problem, namely the *non-heuristic* algorithms; here the correctness of the algorithm is guaranteed within the stated framework of models for the robot, terrain and sensor system. Such algorithms are very important in the mission-critical operations, e.g., shutting down a malfunctioning reactor. We will concentrate on the algorithmic issues under the assumptions of ideal sensors. For the majority of the works, we consider a point-sized robot navigating in two-dimensional terrains. For the unknown terrains navigation, there are several methods that are demonstrated to work well in practical situations, but are not designed to be non-heuristic (see [33]). There are two types of such works. The first kind deals with navigating robots in real-world environments, such as those discussed in Elfes [24], and Turchen and Wong [80] (just to name a few). The second kind deals with a framework of terrains and sensors in a precise formulation but the algorithms are not guaranteed to converge to a destination such as methods discussed in Iyengar et al [34] and Chan and Tam [13] (again just to name a few). These works are fairly extensive and are outside the scope of this survey.

Over the past few years, the topic of non-heuristic navigation algorithms in unknown terrains has received increasing attention by the researchers in the areas of robotics, computer science and engineering; of particular importance are the methods that ensure some properties such as performance guarantees, etc. The focus here is to obtain provably correct algorithms for navigating automata or robots in terrains whose maps are not known ahead of time. At the outset, the formulations of these problems appear to be of only theoretical interest; however, these methods constitute an important guide to a number of practical solutions as evidenced from the works of Lumelsky [45]. Such line of thought seems to have been followed by a number of researchers since as far back as 1873, in the form of maze searching problems studied by Weiner (Ore [57]). Before the advent of computers and electronic circuits, the majority of these works have been basically theoretical. Subsequently, several contributions to this field have been made by a number of researchers working in diverse areas. Many of these results are scattered in various publications, and an overall introductory treatment of these (particularly early and recent) works is not available in a single location. In view of the recent upsurge of interest in these problems, such a treatment will be helpful to non-specialists and newcomers to this area.

We mainly consider the *navigation problem* that deals with moving a robot to a destination while avoiding a certain set of obstacles on the way. The obstacles are detected using a sensor system (since they are not a priori known). Consequently, solutions to this problem vary greatly with the sensory system of the robot. The other factors affecting the algorithm are the assumptions on the obstacle terrain and the computational power of the robot. We also consider the *terrain model acquisition problem* that deals with building a terrain map by exploring the terrain using sensors. Most of our discussion deals with a point-sized *robot* amidst two-dimensional obstacles in the plane. Obstacles can be simple closed curves (of finite perimeter) [44], or polygonal [61] or have boundaries consisting of sequences of line segments and circular

arcs [62]. We basically consider *touch* and *vision* sensors, which are assumed to operate without errors. The algorithms based on touch sensors assume that the robot has some type of capability to move along the boundaries of the obstacles.

The algorithms surveyed in this paper can be classified into three broad categories based on the overall objectives. The first category deals with algorithms that are shown to navigate correctly without giving much consideration to the performance parameters such as distance traversed, etc. In the second category, the main objective is to guarantee bounds on the distance traversed or ratio of the distance traversed to the shortest path length computed if the terrain model is known. The third category deals with robots with limited computational capabilities such as finite automata, etc. Also, we can classify the navigation algorithms for unknown terrains based on the sensor systems used by the robot. Typically these algorithms employ either touch or vision sensors. A taxonomy based on these characterizations is provided in the next section.

The treatment in this paper is informal and elementary, and is intended to highlight the basic ideas of various methods; the technical details of the works can be obtained from the appropriate references.

The organization of the paper is as follows. We provide a classification of various navigational methods of unknown terrains in Section 2. Algorithms for simple maze searching are discussed in Section 3. These works constitute some of the earliest efforts to solve the navigational problems within formal frameworks. Although many of these works are fairly limited in their applicability, they provide some of the basic ideas that have been subsequently used. In Section 4, we consider algorithms based on touch sensors; these algorithms are pioneered by Lumelsky [45] and could be considered an inspiration to a number of subsequent works. Algorithms based on vision sensors are presented in Section 5. We consider two types of vision sensors: discrete sensors perform 360 degrees scan from the present location, and continuous sensors "see" all visible parts of the terrain as the robot navigates. In terms of information, a continuous sensor can simulate a discrete sensor but not vice versa if the latter is restricted to perform only a finite number of scan operations. Section 6 deals with the works that have been done in past few years as a part of renewed interest in this area; major inspiration seems to be the challenge of achieving some type of optimality in navigating in unknown terrains. Section 7 deals with computational issues involved in solving the navigation problem by considering a robot with very limited computational capabilities; several works show the limitations of robots in searching mazes. The brief description in Section 8 is intended to provide some related information on algorithms for searching unknown graphs; several graph methods have been employed to solve a number of navigation problems, and an insight into the former will help understand some important issues of the latter. In terms of algorithmic content, the problem of navigating in a geometric terrain is easier than that in graphs, mainly due to the presence of spatial information in the former (see Blum and Kozen [7]).

2 A Taxonomy of Navigation Algorithms

Algorithms for various formulations of the navigation problem in unknown terrains have been studied by a diverse set of researchers, e.g., mathematicians, electrical engineers, computer scientists, etc. Although the focus and treatment of these algorithms can differ considerably, often they can be visualized to be having some common underlying themes. Instead of a strict classification (which is very difficult to provide), we provide some broad keywords and phrases that characterize some classes of these algorithms. We consider the criteria of (I) overall objectives, and (II) sensor systems.

(I) **Overall Objectives:** We can classify the existing methods into three classes based on the overall objectives of the navigation.

(A) In the *Class A* methods, the main goal is to guarantee that the navigation objective is achieved, e.g., reaching a destination point, acquiring a model of the terrain, etc. In general, these algorithms are not designed to optimize parameters such as distance traversed, etc. Early works on this class of algorithms can be traced back to Sutherland [79] who presents an outline of a proof for the algorithm of Shannon's mouse proposed in late 1940. In eighties, interest in these algorithms has been rejuvenated by the works of Lumelsky [45]. Several of the early maze searching algorithms can be included in this class; discussion on several maze searching algorithms and their relation to robot navigation algorithms can be found in Lumelsky [42] and Sankaranarayanan and Masuda [70].

(B) The *Class B* methods are intended to optimize parameters such as distance traversed, figure of merit, etc. This area has attracted the attention of several researchers over the past few years. Although several performance measures, such as number of scan operations, number of elementary motion commands, etc., can be considered, recent works deal with either minimizing the distance traversed by the robot (Baeza-Yates et al [3]) or the ratio of the distance traversed to the shortest possible distance when the terrain model is known (Papadimitriou and Yanakakis [58], Blum et al [5], etc.). This line of algorithms is expected to receive increasing attention in future.

(C) The *Class C* algorithms attempt to extract the basic computational issues involved in these problems along the lines of theory of computation. For example, assuming that the robot has the computational capability of a finite state automata, one might be interested in the type of navigational problems that can be solved. Most of these works are restricted to the terrains of mazes. Some of the early work in this area is pioneered by Budach [11].

(II) **Sensor System:** There are two different varieties of sensors, namely *touch* and *vision*, that have been studied in literature.

Class	Subclassification	Representative References
Class A	maze searching	Shannon's mouse [79], Tarry and Tremaux [57], Fraenkel [27], Pledge algorithm [1]
	touch sensor	Lumelsky [45],Cox and Yap [18], Sankaranarayanan and Vidyasagar [73]
	continuous vision	Sutherland [79], Lumelsky et al [50], Lumelsky and Skewis [51]
	discrete vision	Rao [61], Choo et al [15], Foux et al. [26]
Class B	searching in plane	Baeza-Yates [3], Kao et al [38]
	figure of merit	Papadimitriou and Yannakakis [58], Blum et al [5], Bar-Eli et al [4], Deng et al [20], Klein [40], Kleinberg [41] Kalyanasundaram and Pruhs [37, 35, 36]
Class C	restricted computation	Budach [10, 11], Coy [19], Dopp [22], Shah [75], Blum and Kozen [7]

Table 1: A taxonomy of non-heuristic navigation algorithms.

(A) *Touch Sensors:* Typically a touch sensor detects when the robot touches an obstacle. Several algorithms based on such sensors have been extensively studied by Lumelsky [45] and by many other researchers. Early use of touch sensors goes back to the Pledge algorithm [1]. Some of the more recent works based on these sensors are due to Cox and Yap [18] and Sankaranarayanan and Vidyasagar [72, 71, 73].

(B) *Vision Sensors:* A vision sensor typically provides the information visible to the robot; there are two basic characterizations of a vision sensor: *continuous* and *discrete* sensors. As the robot navigates along a path, a continuous sensor can detect all parts of the terrain that are visible. Some of the early navigation algorithms based on continuous vision sensors are due to Sutherland [79]. More recently, the algorithms of Lumelsky and Skewis [51] solve the navigation problem using the continuous vision sensors; the terrain model acquisition problem is solved by Lumelsky et al [50], and Deng et al [20]. The discrete vision sensor provides a 360 degrees scan from a single position of the robot, i.e. the sensor obtains the boundary of all obstacles that are visible from a single point. Such operation is called a *scan* and the robot is required to perform only a finite number of such operations. Rao [61] studied algorithms based on the discrete vision sensors for solving the navigation and terrain model acquisition problem (the latter has also been studied by Choo et al [15]). If the number of scan operations is bounded, then there are navigational problems that cannot be solved by using discrete scan sensors, e.g. in terrains where obstacle boundaries are sequences of line segments and circular arcs [62]. Based on the range that a vision system is capable of, there are infinite distance sensors and finite distance sensors. Many of the vision based algorithms

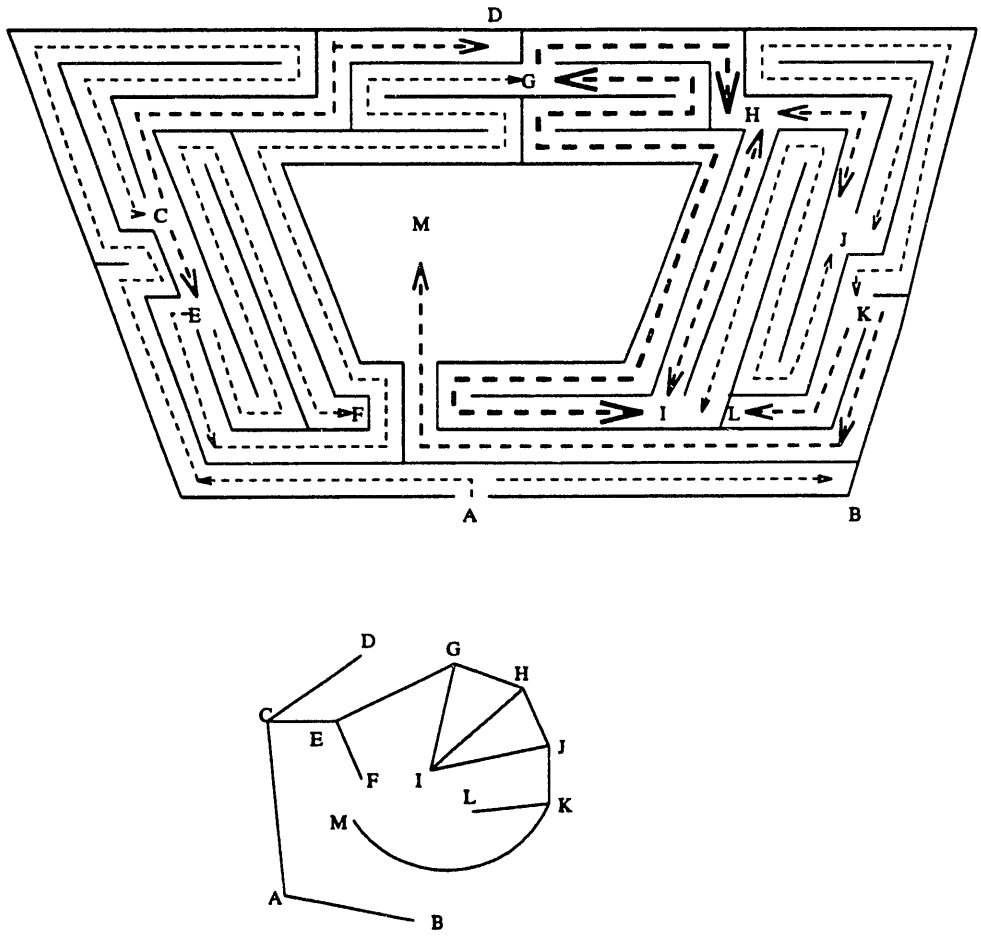


Figure 1: Graphical representation of a maze

assume that the range of vision is unlimited; Lumelsky and Skewis [51] describe algorithms when the radius of vision is limited. In many cases, especially in maze searching by Fraenkel [27], a vision or a touch sensor system that is capable of identifying the "corridors" or "paths" is implicitly assumed.

A taxonomy of the works described in this paper is provided in Table 1 with some representative references.

3 Simple Maze Searching

Maze searching algorithms have been studied since as early as 1873 by Weiner (Ore [57]), and interest in such problems can be traced back to the Euler's work on Königsberg bridge problem (see Chapter 1 of Harary [30] for details of this problem). Typically, in these problems we have an automaton with the ability to touch and/or see. There are two objectives of maze-searching algorithms: first to search for

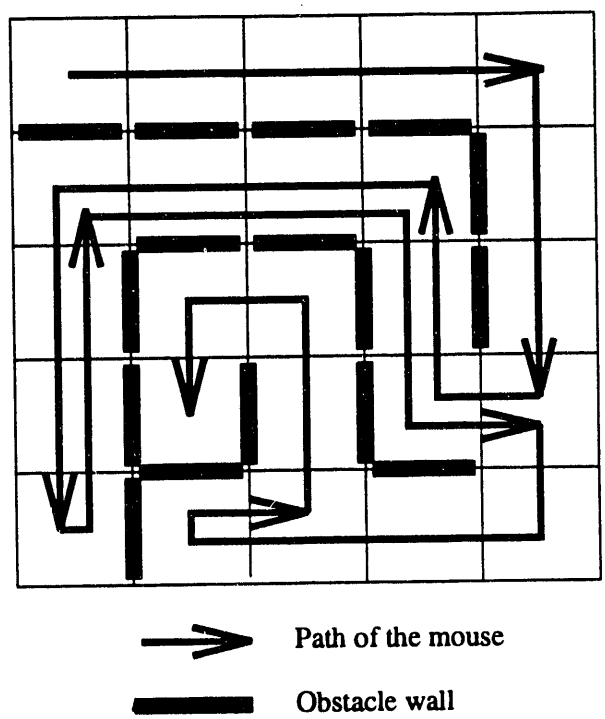


Figure 2: Path followed by Shannon's mouse

a "treasure" hidden in an unknown location inside the maze, and second to escape out of a maze from an unknown location.

Early efforts to solve maze problems are based on modeling the maze as a graph (see Fig. 1.) and invoking a suitable graph search algorithm. Here in Fig. 1 each corridor is represented by a graph edge, and a vertex represents a location where two corridors meet or a corridor dead-ends. A systematic exploration of the graph edges can result in finding a way out of the maze. In Fig. 1, entire interior of the maze can be seen if each edge has been traversed (equivalently each corridor has been traversed) using a continuous vision sensor; at the termination of such algorithm the robot is guaranteed to find the treasure or exit. Such ideas have subsequently resulted in a number of useful navigation algorithms both in known and unknown terrains; see Lumelsky [49] for discussion on the relation between maze-searching algorithms and navigation algorithms in unknown terrains.

In this section we first discuss an interesting maze searching algorithm due to Shannon (Sutherland [79]). Early maze-searching algorithms based on graph searching methods discussed in Ore [57] and Fraenkel [27] will be described Sections 3.2 and 3.3 respectively. The Pledge algorithm, based on a touch sensor and a compass-like device, is described in Section 3.4.

3.1 Shannon's Mouse

In late 1940's Claude Shannon built a maze-solving mouse that is capable of finding "cheese" stored in one of the squares of a 25-square checkerboard maze (described in

Sutherland [79]). Two adjacent squares are separated by removable aluminum walls.

He proposed a maze-searching algorithm which is the first of its kind. Each square is assigned two bits which indicate an arrow showing which way the mouse went in the last last visit (if any) to the square. When exploring the maze, the mouse always attempts to leave each square it entered in a direction 90 degrees to the left of the recorded direction, updating its recording. If the mouse is struck by a wall in that direction, it returns to the center of the square and tries again another 90 degrees to the left. With this algorithm the mouse will eventually find the cheese; this algorithm, however, appears inefficient because the mouse will sometimes leave a square by the very opening it used in entering. See Fig. 2 for an execution of this algorithm. An informal correctness proof of this algorithm is given by Sutherland [79]. Notice that the mouse here has at most four directions to move from any cell, and it finds out that an adjacent cell is not reachable by "bumping" into the separating wall.

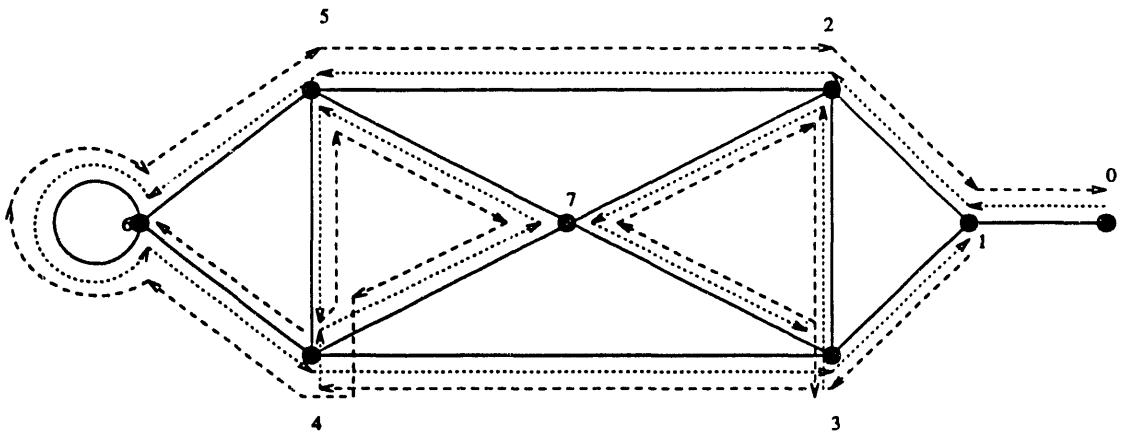


Figure 3: Execution of Tarry's algorithm

3.2 Tarry and Trémaux Algorithms

The connection between graph searching and maze searching has been discovered in several early works. A graph model of a maze (such as the one in Fig. 1) is employed by the algorithms of this and the next section.

The Tarry's algorithm [57] constructs a *cyclic directed path* passing through each edge once and only once in each direction. The algorithm starts at an arbitrary vertex a_0 and follows a path P marking each edge with the direction in which it has been traversed. When one arrives at some vertex g for the first time, the entering edge is marked specially. When one reaches a vertex g, one always follows next an edge (g, r) which either has not been previously traversed, or if it has been, it has been traversed only in the opposite direction. However the entering edge should be followed only as a last resort, i.e., when there are no other edges available. The execution of the Tarry's algorithm will terminate at the initial vertex a_0, with each edge traversed twice, once in each direction. An execution of Tarry's algorithm is shown in Fig. 3. In spirit,

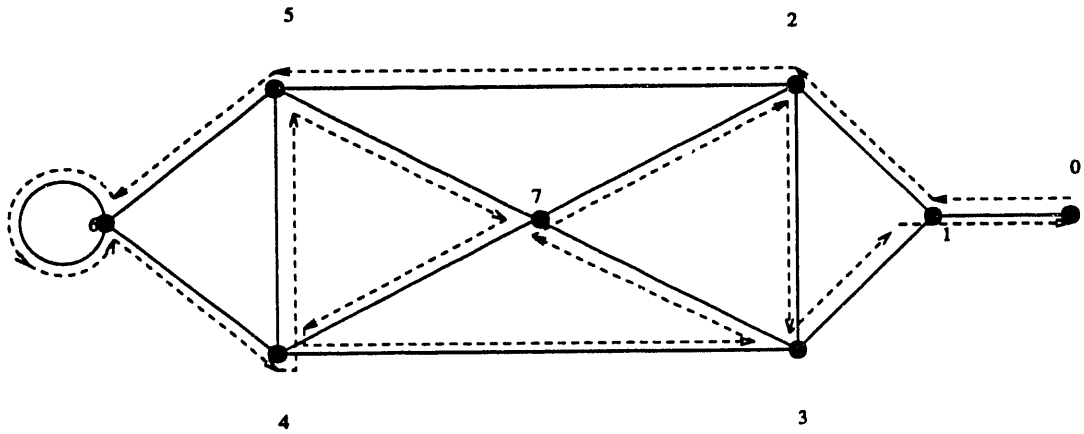


Figure 4: Execution of Fraenkel's algorithm

this algorithm [2] is similar to the popular depth-first search algorithm used in graphs [2].

The same result is obtained in a different method proposed by Tremaux [57]. The procedure adopted is that of *progressive covering* of the graph. The technique ensures that all the vertices within a certain distance have been visited. It is easy to touch upon the vertices of distances 1: one passes through the various edges at a_0 to their end points, each time returning to a_0. Each edge $E = (a_0, a_1)$ is marked once as one leaves a_0 and at a_1 it is marked as the entering edge. To reach a vertex at a distance 2 from a_0, one selects some open edge E $= (a_0, a_1)$ and marks it again; at a_1 a similar procedure is applied and reachable vertices from a_1 are marked. At a_1 if one traverses and reaches a vertex which has already been visited then it is marked closed. If all the vertices from a_1 are closed then the edge (a_0, a_1) is marked closed. The operation is continued until all edges at a_0 are marked twice. This algorithm is similar to the *breadth-first search* algorithm used for searching graphs [2].

3.3 Fraenkel's Algorithm

An improvement to the Trémaux and Tarry's algorithm has been proposed by Fraenkel [27, 28]. In this algorithm every edge is traversed once and at most once in each direction. We assume that upon the arrival at a vertex v, its entrance edge and the edges incident to v which have been traversed previously along with the direction are known. Let $\rho(v)$ be the *valence* of v, i.e., the number of edges incident to v and let v_0 be the initial vertex. Without loss of generality we assume $\rho(v_0) = 1$ of the initial terminal of this alley is 1.

The algorithm proceeds as follows:

[2] By visualizing two contra-directed edges for each corridor, we can see that this algorithm defines a subclass of *directed Euler graphs* (a directed graph is called a *directed Euler graph* if each node has the same number of incoming and outgoing arcs [17]). The path taken by the robot runs through each directed edge precisely once and such path is called the *directed Euler tour*. For this subclass of directed graphs, the depth-first search algorithm yields the directed Euler tour.

(1) Start out from v_0 with the counter initialized to zero. The counter is increased by unity upon arrival at a vertex that has not been traversed before.

(2) If we arrive at a vertex v such that before entering it there was at least one edge incident to it which was not yet traversed, and upon arrival at v there remains at most one such edge, decrease the counter by 1.

(3) As long as the counter is positive, the tour is conducted according to Tarry's algorithm, but, whenever possible, an edge not traversed before, is used in preference over an edge traversed before.

(4) Suppose the counter becomes zero at vertex v_k. If there is an untraversed edge incident to v_k follow it. Otherwise, leave all vertices via their entrance edges.

An execution of Fraenkel's algorithm is shown in Fig. 4. It is to be noted that the above solution is not unique and several other valid solutions exist.

Notice that the algorithms of Tarry, Tremaux and Fraenkel assume that the sensors are adequate to navigate the automaton along the required corridors. Further they assume that the robot can identify a corner when it revisits; this capability can be implemented by using real arithmetic and suitably storing the points visited by the automaton.

3.4 Pledge Algorithm

The Pledge algorithm deals with navigating a point automaton with touch sensing, and a compass that can measure the "amount of turn". The automaton is trapped inside a maze, and it is required to escape out of the maze. The following algorithm for this problem has been reportedly invented by a 12 year old boy in Exeter, England (reported in Abelson and diSessa [1]). Fig. 5 illustrates a point automaton escaping a maze using Pledge Algorithm by the following steps.

(1) Choose an arbitrary fixed direction call it F_{init} and face this way.

(2) Walk following F_{init} until you detect an obstacle by front sensor.

(3) Turn left and follow the obstacle boundary keeping the obstacle on the right side.

(4) Follow the obstacle around, until the total turning angle is zero. Go back to step 2.

Notice that the automaton only needs a primitive computational ability of adding and subtracting the amount of turn, which is a real number. The sensor system must enable the automaton to (a) navigate along the obstacle boundaries, and (b) measure amount of turn. Pledge algorithm is the first non-heuristic algorithm based on touch sensing. A detailed proof of correctness of the algorithm is given by Abelson DiSessa [1]. This algorithm inspired some robot navigation algorithms such as algorithm Curve1 of Sankaranarayanan and Masuda [70].

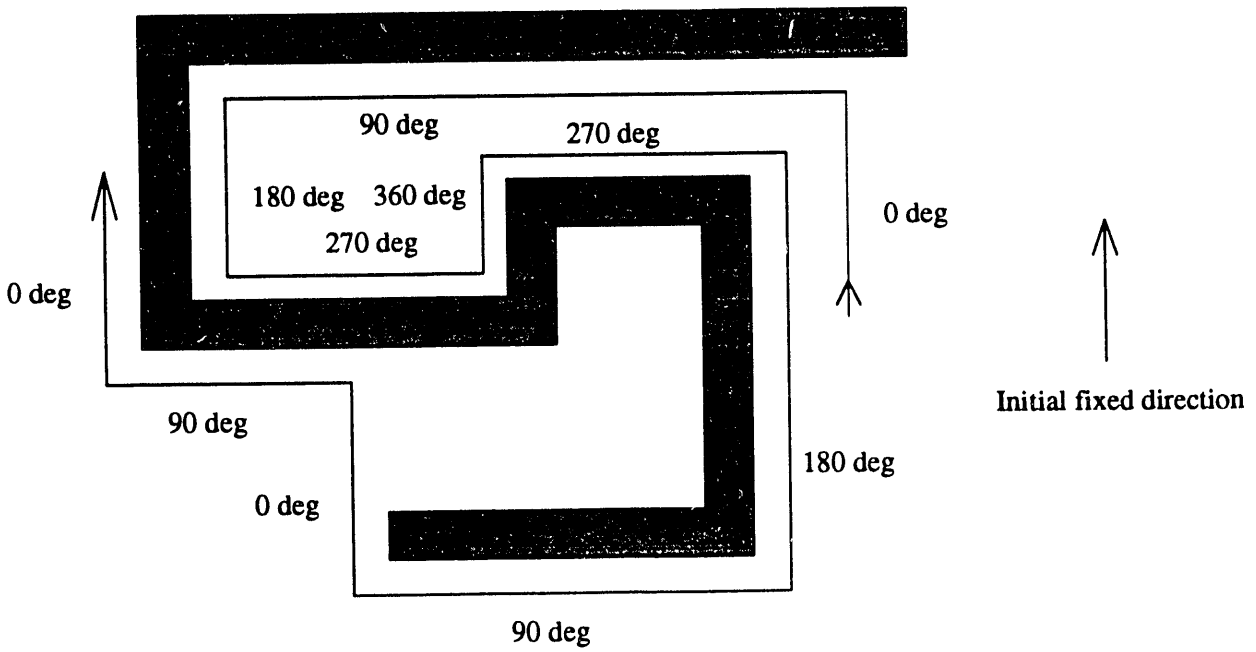


Figure 5: Execution of Pledge algorithm

3.5 Closure of Section

There are several features of these maze searching algorithms that are subsequently used in robot navigation algorithms. First notice that the Shannon's mouse discovers obstacles by bumping into them and each obstacle is known to be a wall separating two cells; also, the direction of movement of the mouse is restricted to four known directions. The algorithms of Tarry, Tremaux and Fraenkel are high level strategies based on the assumption that corridors can somehow be detected and followed. The Pledge algorithm is based on detecting obstacles by touching them.

In terms of the memory used, the Pledge algorithm is unique in that it requires just one variable denoting the amount of turn, whereas the other algorithms explicitly store information about the locations that have been visited earlier.

4 Navigation Using Touch Sensing

The interest in non-heuristic algorithms for navigation in unknown terrains has been rekindled in mid eighties due to the pioneering works of Lumelsky and Stepanov [53]. As a result a number of algorithms based on touch sensing have been subsequently studied; these algorithms will be the focus of this section.

We first discuss two basic algorithms Bug1 and Bug2 of Lumelsky and Stepanov [53]. The ideas behind these algorithms have been subsequently extended to several contexts. Then we discuss the algorithms of Sankaranarayanan and Vidyasagar [73] and Sankaranarayanan and Masuda [70]. Then we consider a navigation algorithm for a ladder based on a touch sensor due to Cox and Yap [18]. We then deviate from the main line of discussion of this paper to discuss the applications of the algorithms

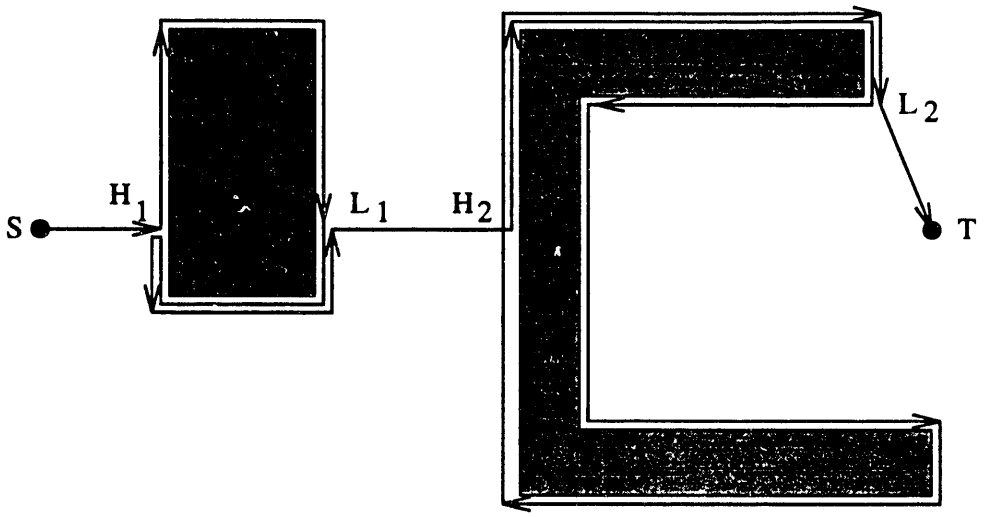


Figure 6: Execution of Bug1 algorithm

of Lumelsky et al. [46, 47] to manipulators.

4.1 Lumelsky's Algorithms

A point robot capable of touch sensing is considered here; the robot can detect a contact with an obstacle and also follow its boundary. Obstacle boundaries are simple closed curves of finite length. Two basic algorithms, *Bug1* and *Bug 2*, are proposed in [53]. Here the problem is to reach a specified destination T from the present location S. Some simple strategies will not correctly navigate the robot to the destination; we discuss these aspects briefly in Section 5.4.

In the algorithm Bug1 the automaton meets the ith obstacle at a *hit point* H_i, $i = 1, 2, \ldots$ and leaves it at a *leave point* L_i, $i = 1, 2, \ldots$; $L_0 = S$. The behavior of the automaton is illustrated in Fig. 6. The algorithm proceeds as follows:

(1) From the point L_{i-1} move towards the target along a straight line until one of the following occurs. (a) if the target is reached, then terminate the algorithm; (b) if an obstacle is encountered, define hit point, H_i and go to Step 2.

(2) Turn left and using this local direction, follow the obstacle boundary. Stop if target is reached. Else, after having traversed the whole boundary and having returned to H_i, define a new *leave point* L_i, which is a point on obstacle boundary closest to the destination.

(3) Take shortest distance path along the obstacle boundary to the point L_i after returning to H_i. Apply the target reachability test. If the target is reachable then the algorithm terminates. Otherwise increment i and go to Step 1.

A proof of correctness of Bug1 is described in [53]. By the execution of this algorithm the length of the path P produced will never exceed the limit $D + 1.5 \cdot \sum_i p_i$ where D is the straight line distance between the target and the start points, and p_i

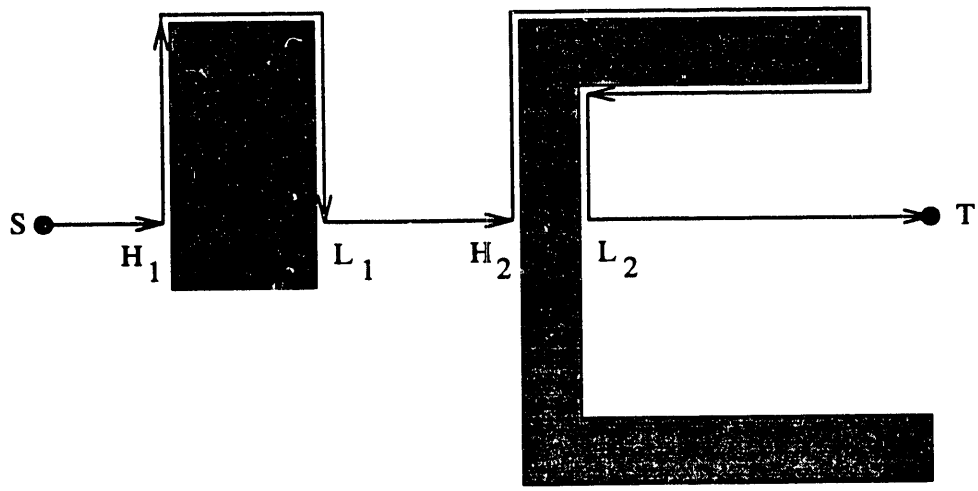


Figure 7: Execution of Bug2 algorithm

refers to the perimeter of the obstacles intersecting the disc of radius D centered at the target.

For the algorithm Bug2, the automaton starts from L_0. Let $d(P)$ denote the distance between a point P and the target point. The algorithm consists of the following steps.

(1) From point L_{i-1}, move along the straight line ST until one of the following occurs: (a) if the target is reached, the algorithm terminates; (b) if an obstacle is encountered, a *hit point*, H_i, is defined.

(2) Using the accepted local direction, follow the obstacle boundary until one of the following occurs. (a) The target is reached, and the algorithm terminates. (b) The line ST is met at a point Q such that the distance $d(Q) < d(H_i)$, and the line QT does not cross the current obstacle at the point Q. Define the leave point $L_i = Q$. Increment j and go to Step 1. (c) The automaton returns to H_i and thus completes a closed curve without having defined the next hit point, H_{i+1}. In (c) the target is trapped and cannot be reached, and the algorithm stops.

In this algorithm a path segment navigated around an obstacle is often (but not always) shorter than the perimeter of the obstacle (Fig. 7). The proof of correctness of this algorithm is described in [53]. The length of the path generated never exceeds $D + \sum_i \frac{n_i p_i}{2}$, where p_i refers to the perimeter of the obstacles intersecting the straight line segment ST and n_i is the times the ith obstacle is visited.

A comprehensive treatment - including a general lower bound and correctness proofs - of the algorithms of Lumelsky and his associates is given in [45].

4.2 Sankaranarayanan's Algorithms

Several extensions to the early versions of Lumelsky's algorithms have been proposed by Sankaranarayanan and Vidyasagar [72, 71, 73] and Sankaranarayanan and Masuda

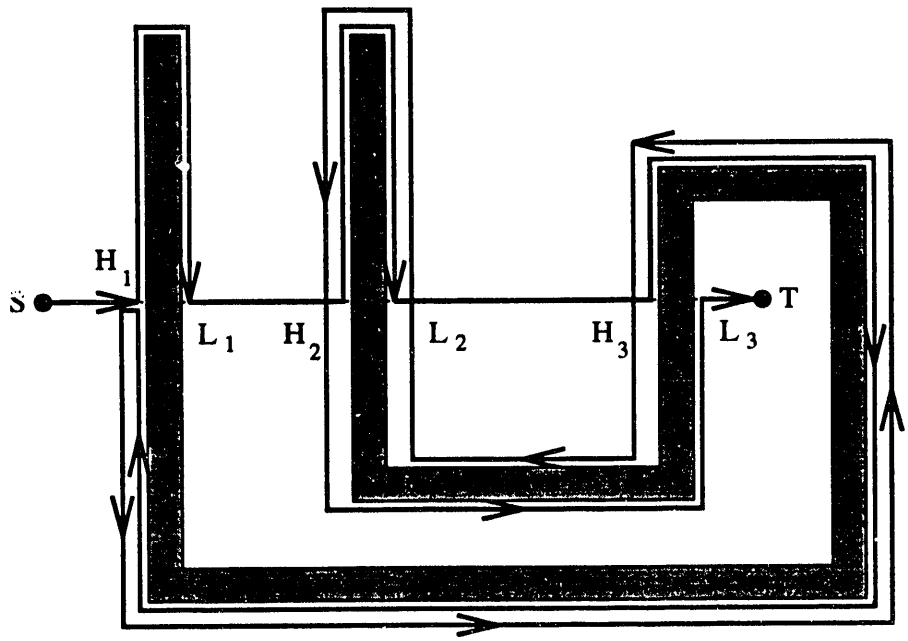


Figure 8: Execution of algorithm Alg1

[70]; these algorithms lead to generalized solutions. We discuss *Alg1* and *Alg2* of [71, 72] and *Curve1* of [70] in this section.

As a nice intermediary between the algorithms Bug1 and Bug2, a new algorithm Alg1 has been proposed by Sankaranarayanan and Vidyasagar [71] in which the automaton travels less along the obstacle boundaries and more along the straight line segments towards the target point T. Algorithm Alg1 can be briefly described as follows.

(1) The robot moves along the line *M-line* joining the start (S) and destination (T) locations until an obstacle is met.

(2) The robot follows the boundary in a specified local direction (say left). The robot leaves the obstacle boundary at a point L if and only if the following two conditions are satisfied: (a) robot can moves along the line joining L to destination; (b) L is the closest point to destination on the M-line ever visited by the robot.

(3) After meeting an obstacle at hit point H_j and moving along that obstacle boundary, the robot can meet a previously defined hit point or leave point Q_k ($k < j$). If that happens, the robot retraces its path back to H_j and moves along the section of the obstacle boundary on the other side of H_j.

Note that Bug1 traverses along the entire boundary of every obstacle it encounters and Bug2 avoids such traversals in simple cases by attempting to navigate only along a part of the boundary; however in complex terrains, Bug2 may repeatedly visit the same obstacle. Alg1 avoids this problem. The path length generated by this algorithm is upperbound by $2\sum p_i + D$. See Fig. 8 for an execution of this algorithm.

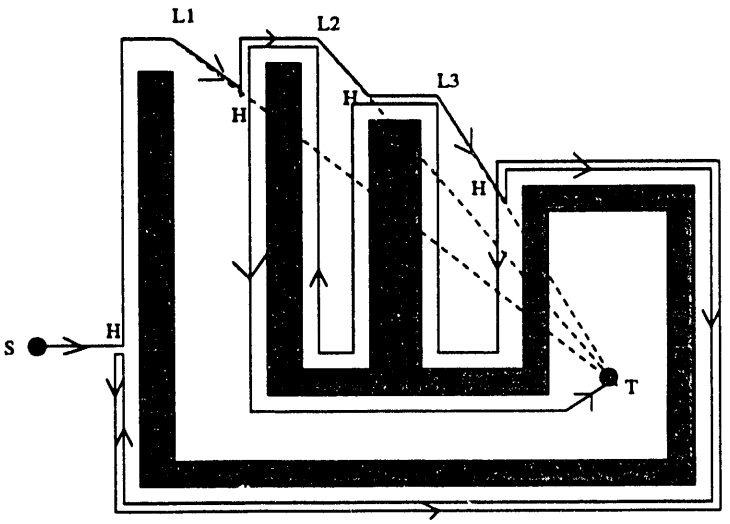


Figure 9: Execution of algorithm Alg2

The algorithm Alg2 proposed by Sankaranarayanan and Vidyasagar [72] can be described briefly as follows.

(1) The automaton travels along a straight line towards the destination T until an obstacle is met.

(2) The automaton leaves the obstacle boundary at a point L if and only if the following two conditions are satisfied: (a) at L, the automaton can move along a line segment LT that does not enter the obstacle, and (b) L is closest to T among all x ever visited by the automaton prior to visiting L.

(3) If a previously defined hit or a leave point is met then the following rule is applied: if returning to a hit point H, the automaton moves along the unvisited section of the obstacle boundary, which starts at H.

In this algorithm the length of the path generated never exceeds the limit $2\sum_i p_i + D$. An execution of Alg2 is shown in Fig.9.

Sankaranarayanan and Vidyasagar [73] classify the touch based navigation algorithms into classes I and II. Algorithms of Class I traverse the entire boundary of every obstacle they encounter (such as Bug1) at least once before leaving it; algorithms of Class II leave at least one obstacle before traversing its entire boundary. For the former class they show a lower bound of $1.5\sum p_i + D$ for the distance traversed by the robot and for the latter class of algorithms they show a lower bound of $2\sum p_i + D$ [73].

The algorithms Bug1, Bug2, Alg1 and Alg2 are called *metric* algorithms since they use information such as position, distance, etc. Sankaranarayanan and Masuda [70] present the algorithm Curve1 that uses non-metric information to follow a track in a terrain populated by unknown obstacles. A *guide track* is a non self-intersecting curve ST, connecting the source S and the target T. The algorithm is based on the topological property observed in the Jordan-Curve theorem that the curve ST

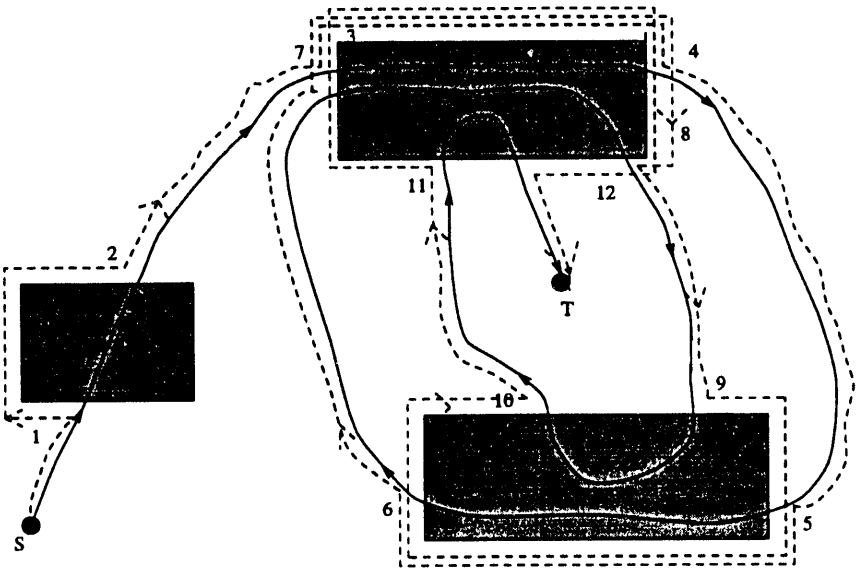


Figure 10: Execution of Curve1 Algorithm

intersects an obstacle boundary at even number of points. The automaton has a counter C associated with it. The algorithm is described as below.

(1) Start from the point S. Set the counter C to zero.

(2) Move along the curve ST until one of the following occurs: (a) the target T is reached, stop; (b) an obstacle is met, then follow the obstacle boundary in the local direction *left*.

(3) Follow the obstacle boundary until one of the following occurs:

   (a) The target T is reached, then stop;

   (b) The curve ST is met at a point P. One of the following steps is executed.

      i. The counter C reads zero and, at P, the automaton *can* move along curve ST towards T. Follow the curve ST away from the obstacle.

      ii. The counter C reads non-zero and, at P, the automaton *can* move along curve ST towards T. Decrement the counter C by unity and move along the obstacle boundary. Goto Step 3.

      iii. At P, the automaton *cannot* move along curve ST towards T. Increment the counter C by unity and continue moving along the obstacle boundary. Goto Step 3.

Fig. 10 illustrates the execution of the algorithm Curve1. This algorithm is inspired by the Pledge algorithm discussed in Section 3.4; notice that the robot in this case is able to follow the track with just a compass-like device.

4.3 Cox and Yap's Algorithm

Navigation of a rod or ladder using a touch sensor has been studied by Cox and Yap [18] in two-dimensional terrains populated by polygonal obstacles. The ladder can be thought of as an autonomous vehicle which can carry out "guarded move" instructions consisting of motion along a smooth curve or a compliant motion which maintains specified obstacle contacts until an event occurs, such as contact with a new obstacle or a coordinate taking on a specified value.

Let FP be the set of all positions of the ladder that do not cause a collision with obstacles. The idea of this algorithm is to have the ladder search the environment while keeping in contact with the obstacles, dynamically constructing a road-map of the environment. The road-map consists of a superset of the edges of the topological boundary of FP. The notion of road-map has been extensively used in known terrains (see Canny [12] for details on road-map), and in vision based algorithms in unknown terrains (Rao [61]).

All guarded move instructions specify two constraints that define a curve in FP; the instruction will specify which of two directions to move in that curve. Each constraint is of the following form: (a) maintain contact of the ladder with a particular corner, (b) maintain contact of a ladder endpoint with a particular wall, (c) maintain a certain orientation of the ladder, (d) restrict an endpoint of the ladder to move along a particular line in physical space.

A guarded move is terminated automatically when one of the following event occurs: (a) ladder makes a new obstacle contact; (b) one or several of the coordinates reaches some specific value(s); (c) an endpoint of the ladder reaches some specified point in physical space.

Cox and Yap [18] dynamically construct a road-map of the relevant portions of the environment with a motion that has a path complexity bounded by $O(k) = O(n^2)$, where n is the number of obstacle walls, and k is the total number of pairs of obstacle corners and walls within a distance less than or equal to the length of the ladder. This algorithm is a synthesis of retraction techniques (used extensively in known terrains [76] and to a limited extent in unknown terrains [67]) and the technique of Lumelsky [45].

4.4 Navigation of Manipulators

So far we discussed mobile robots navigating in unknown terrains. We now illustrate that the basic underlying algorithms can also be applied to robot manipulators (see Paul [59] for general discussions on manipulator programming and control). These algorithms are due to Lumelsky and his group [46, 47]. Lumelsky and Sun [52] present a comprehensive treatment of motion planning methods for manipulators within the framework of touch sensors. Our objective here is only to provide some feel for the applicability of previous algorithms to the case of manipulators; this section is not intended to be an exhaustive survey on non-heuristic algorithms for manipulators.

A *planar manipulator arm* can be visualized as a set of line segments (called links) joined at end points. Base link can be translated or rotated with respect to a fixed

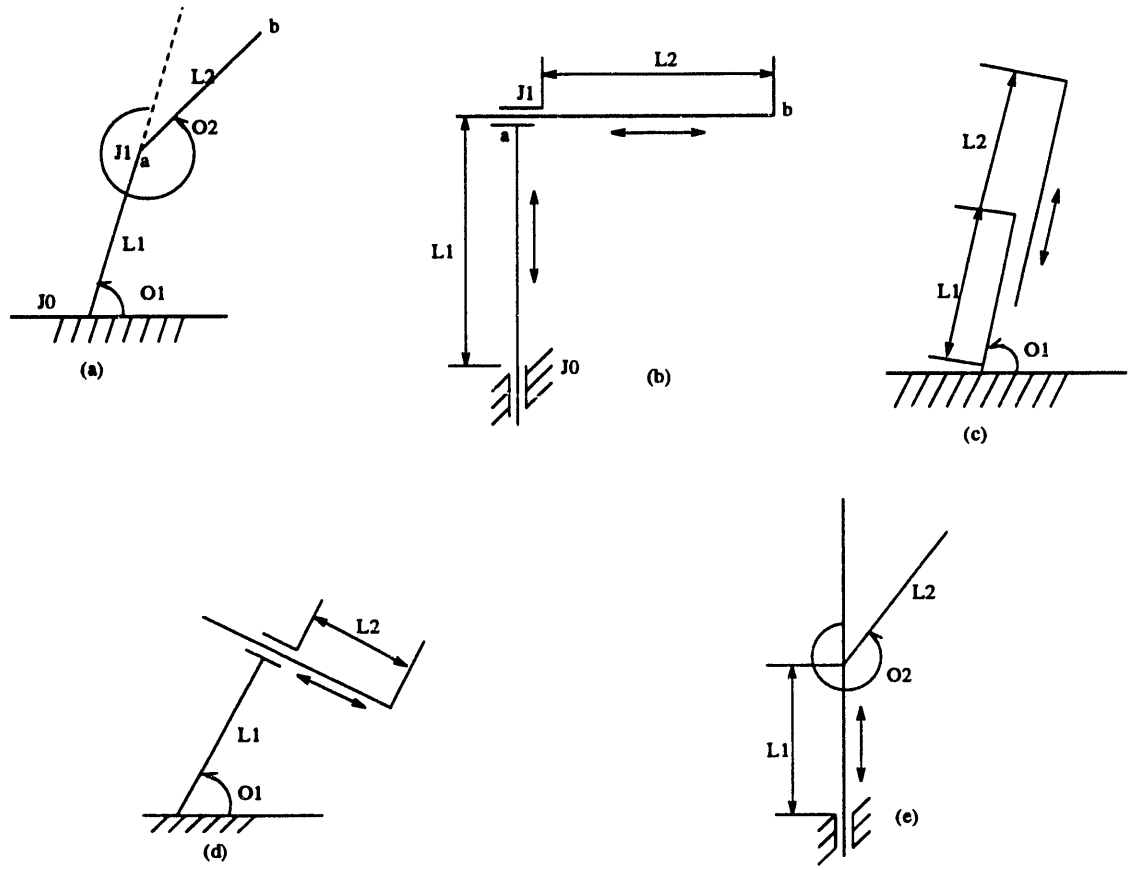


Figure 11: Five Effective Combinations

base. And any other link can be translated or rotated with respect to the links it is joined to; in the former case we have a *sliding joint* and in the latter we have a *revolute joint.* Computational aspects of manipulators have been studied extensively in known terrains (for example see Hopcroft et al [31]). The entire manipulator operates in the plane where the source and target point lie. We consider only manipulator arms with two degrees of freedom; each position of the manipulator is given by a pair of variables, which are either angles or linear translations. The arm is able to do following actions: (a) move the endpoint along parts of a known simple curve, connecting S and T (call this curve M-line), compute the coordinates of consecutive points along the M-line and transform them into the corresponding joint values if necessary; (b) when the arm's body contacts an obstacle, identify the points of contact.

By suitably joining links with sliding and revolute joints, we can potentially have 32 arbitrary combinations for the planar arm connections. Because some joints are not admissible, some are equivalent, and some are not meaningful, only 5 combinations are required in practice (see Fig.11). Here we choose a manipulator with two revolute joints (Fig.11(a)) as an example to present the navigation algorithm. Since each position of this arm is specified by a pair of angles, we can imagine the manipulator to be a point moving on a surface of a torus. The regions of the torus that correspond to positions that are not attainable by the manipulator due to the obstacles constitute

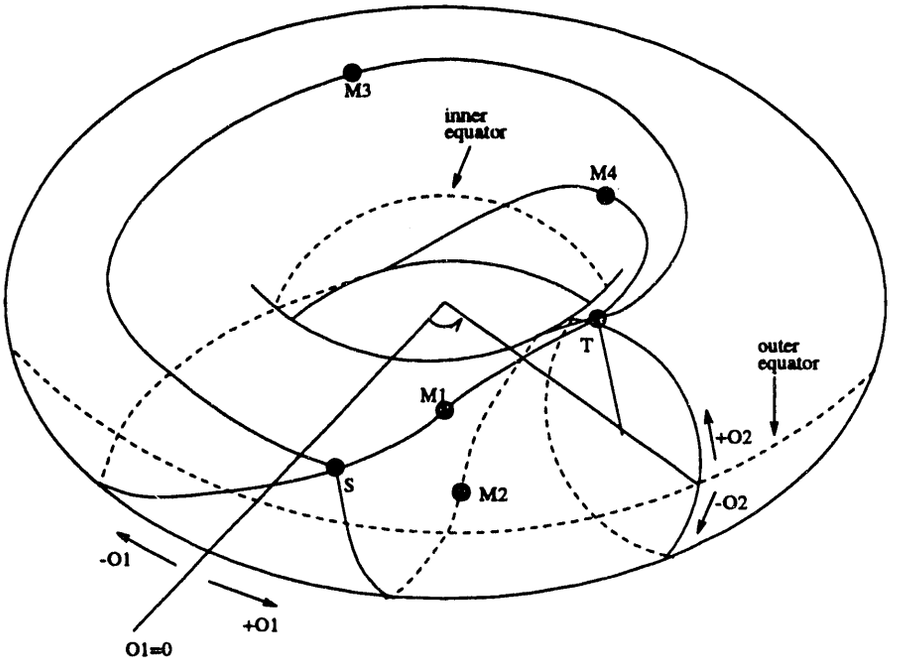


Figure 12: Image Space of Planar Arm Manipulator Algorithm

virtual obstacles. Thus the navigation problem of this manipulator is reduced to that for a point robot amidst virtual obstacles on the torus. We have two different types of virtual obstacles: type I obstacles form a single closed curve on the surface of the torus, and type II obstacles form a band-like structure (on the torus) limited by two simple closed curves. During the motion, some points of the arm body meet obstacles. We have hit points H_j corresponding to points of intersection between the M-line image (projected onto torus) and boundaries of the virtual obstacles. While following the virtual boundary, the arm may meet the M-line more than once. The point at which the arm leaves the obstacle is called the leave point L_j. Because of the structure of the surface of torus, we can have four types of M-lines denoted by M_1, M_2, M_3 and M_4 as shown in Fig.12; let these lines be ordered by their lengths, with M_1 being shortest one. Now the following is a brief outline of the procedure using Lumelsky's method for a manipulator with two revolute joints (refer to Fig. 12). The manipulator uses two counters $C1$ and $C2$ which will record the angles as the robot moves; it also uses a Boolean variable called the *flag.*

(1) Let M-line be M_1-line, flag be set down and $j = 1$.

(2) Let $C1$ and $C2$ be initialized to zero. The arm moves following the M-line from L_{j-1} until the target is reached, otherwise it must hit obstacle and define the hit point H_j.

(3) Set up the counters $C1$ and $C2$. The arm follows the virtual boundary until the target is reached; otherwise either (a) M-line is met at a distance d from T such that $d < d(H_j, T)$ and point L_j is defined, then the increase j go back to

step 2, or (b) if the arm returns to H_j without contacting the M-line ever, then go to next step.

(4) At this step there are two different cases: after checking the obstacle range (that is in the counter $C1$ and $C2$) if the status belongs to following Case (a) then stop the procedure. If not go to Case (b) then go to step 5.

   (a) The range of $C1$ and $C2$ is (0,0) (this is a Type I obstacle), or the range is not (0,0) (this is Type II obstacle) and the flag is up. The result is that target is unreachable.

   (b) The range is not (0,0) and the flag is down: designate shorter of M_3 and M_4 as the M-line, if the range is $(0, n2); |n2| \geq 1$ as an integer; or designate shorter of M_2 or M_4 as the M-line, when the range is $(n1, 0), |n1| \geq 1$; or designate shorter of M_2, M_3 and M_4 as the M-line, when the range is $(n1, n2); |n1|, |n2| \geq 1$.

(5) Reset the arm to start point, set the flag up. Let j=1, go back to step 2.

Now consider a three-dimensional arm that requires at least three degrees of freedom (three links and three joints). Thus maneuvering a body around another body in three-dimensional space presents an infinite number of alternatives and precludes direct application of the strategy of following simple closed curves in the image space. The natural constraints imposed by the arm kinematics may still allow one to reduce the problem to simpler cases. See Lumelsky [43, 48]. Lumelsky and Sun [52] and Sun and Lumelsky [78] for discussions on navigation algorithms for 3D manipulators.

5 Navigation Using Vision

Vision is the most commonly used sense for navigation by human beings. It is both interesting and challenging as to how to use visual information to navigate robots. In this section, we address the problem of navigating robots using continuous and discrete vision sensors. Robot equipped with a discrete vision sensor performs a *scan* operation from a location to return the *visibility polygon* which is the polygonal region of all points visible to the robot from its location (a point is visible if the line segment joining the present location to the point is not intersected by any obstacle). See Fig. 13(a) for an example of the visibility polygon obtained by a discrete vision sensor. Certain computation cost and time is associated with each scan operation, and it is critical that only a finite number of scan operations are performed during navigation. Using a continuous vision sensor, the robot can obtain all the visible points as it moves along a path; more precisely, the robot navigating along a path S will detect the union of all visibility polygons from all locations on S (see Fig. 13(b)). In general, this operation of a continuous vision sensor can be simulated by performing an infinite number of discrete scan operations, but, such simulation is not possible if only a finite number of scan operations are allowed.

The type of vision sensor employed by the robot impacts the navigational algorithm. A navigational algorithm designed for a discrete scan sensor can be executed

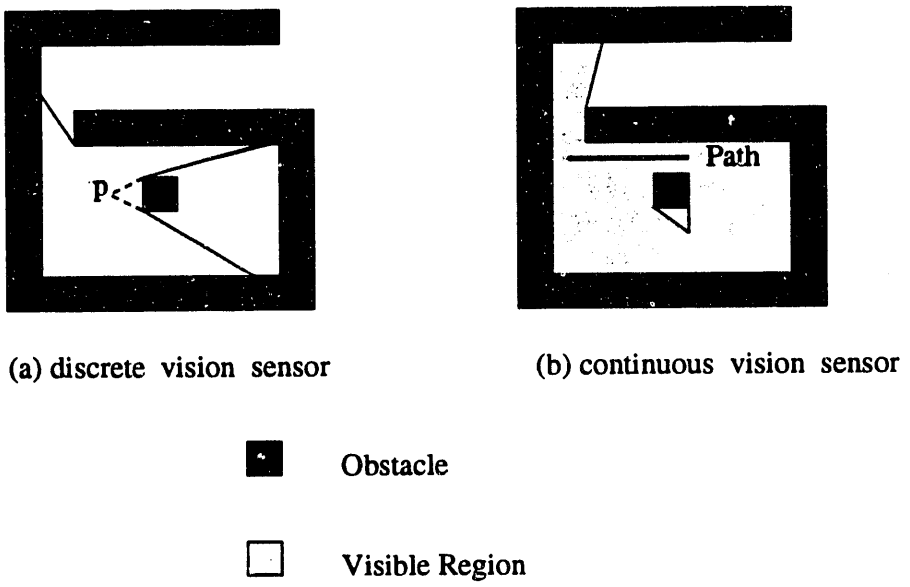


Figure 13: Vision scan operations.

by using a continuous vision sensor but not vice versa. Also, as will be discussed later, there are navigation problems that can be solved using continuous vision sensors but not by using a finite number of scan operations performed by a discrete vision sensor. Also, touch sensing, at least in theory, can be implemented using continuous vision sensors.

In this section, we first discuss Sutherland's algorithm which is the first formal navigation algorithm based on a continuous vision sensor [79]. We then consider the navigational framework of Rao [61] which establishes that a graph algorithm can be utilized on a geometric structure of the terrain to solve the navigation and terrain model acquisition problems; this work is mainly concerned with a polygonal terrain in which a discrete sensor is adequate for navigational purposes. Then we consider the navigation problem and the terrain model acquisition problem using continuous vision sensors due to Lumelsky and Skewis [51], and Lumelsky, Mukhopadhyay and Sun [50] respectively. We also briefly mention some other works based on continuous vision sensors.

5.1 Sutherland's Algorithm

The robot uses a continuous vision sensor to detect "hide regions" behind the obstacles; these regions are called "spurs". In Fig. 14, consider the robot initially located at S; the region on the other side of the point $P1$ defines spur 1, and as the robot moves to the next location the points of spur 1 are all completely seen but a new spur (spur 2) is detected. Here each of $P1$ and $P2$ is called the "point defining the spur". The main idea of Sutherland's algorithm is to move the automaton until all the hide regions are seen. The Sutherland's algorithm can be described in the following steps:

(1) Automaton scans the horizon completely. If there is no spur, then automaton can reach target directly.

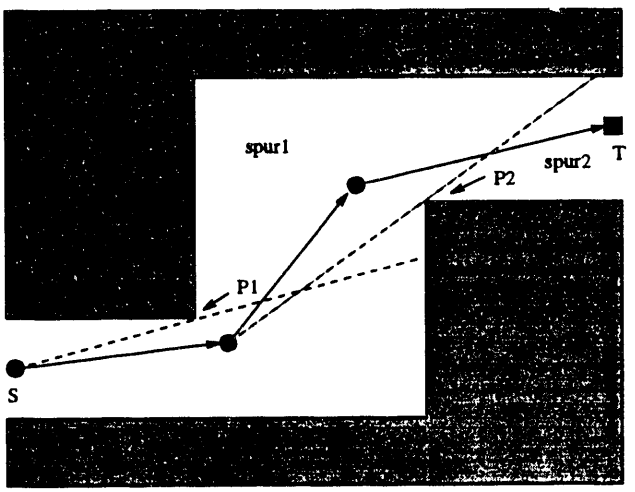


Figure 14: Execution of Sutherland Algorithm

(2) When there is spur in the vision (if there is more than one spur then one is chosen arbitrarily), the automaton judges which side of the open doorway the spur is located and allows for the width of the robot. For example, if the spur is to the left hand side of an open doorway, the automaton moves towards but slightly to the right so that when the automaton gets in doorway, there is enough distance between automaton and obstacle to avoid hitting the obstacle. Then the automaton moves to explore the chosen spur by getting around the doorway of the spur.

(3) As the automaton keeps navigating in the terrain, some spurs disappear and possible new ones will appear. Automaton continues to scan. If there are more spurs the automaton goes back to step 2; otherwise it implies that the whole maze has been explored, the target must be in the view, and should be reached.

The Fig. 14 shows the automaton executing the Sutherland's algorithm to reach a goal.

The approach used in the Sutherland's algorithm can be traced in several other subsequent works. Similar algorithm has been recently described in Deng, Kameda and Papadimitriou [20]. If the terrain is polygonal then the above algorithm can be conceptualized as a finite graph search as shown in the next section; also in this case, a finite number of scan operations using a discrete vision sensor will be sufficient for navigational purposes.

5.2 Framework for Discrete Vision Sensors

An algorithmic paradigm that yields correct algorithms to solve the navigation problem and the terrain model acquisition problem has been proposed by Rao [61]. A finite graph called the *navigation course* ξ, is used as an underlying structure for the navigational purposes. Initially navigation course is not known, but it is incrementally constructed from the sensor operations. The robot executes a graph algorithm

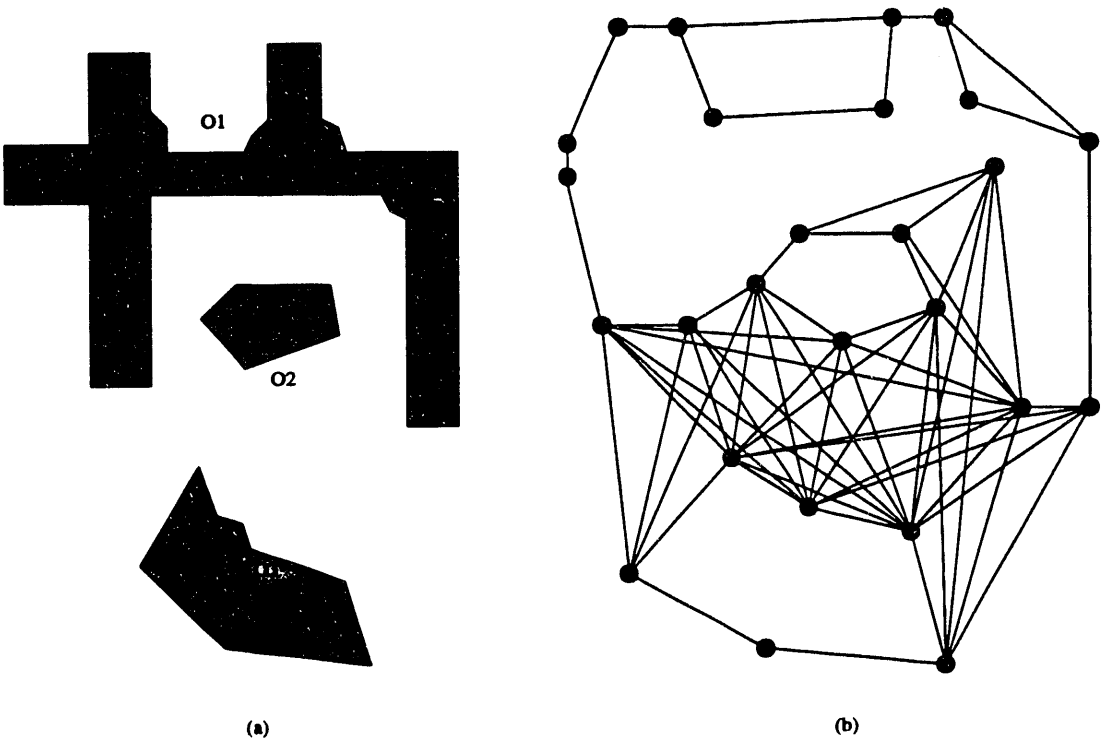


Figure 15: Restricted Visibility Graph

using the navigation course. In solving the navigation problem, the robot starts at a vertex of the navigation course, and carries out navigation until it reaches a vertex from which the destination point is found to be reachable. In solving the terrain model acquisition problem, the robot systematically visits all vertices of the navigation course.

In order that a graph exploration algorithm terminates, the navigation course must contain a finite number of edges and vertices, i.e., must satisfy *finiteness* property. It must satisfy the *terrain-visibility* property which requires that every point in the free-space is visible from some vertex of the navigation course. It must also satisfy the *connectivity* property which requires that every pair of vertices be connected by a graph path on the navigation course. We require that adjacency list of a ξ-vertex can be constructed from the information of a single scan; this property is called the *local-constructibility*. For a navigation course ξ that satisfies the properties of finiteness, connectivity, terrain-visibility and local-constructibility, any graph search algorithm (e.g. depth-first search) can be employed to solve the navigation and the terrain model acquisition problems.

We now discuss three examples of the navigation course. We first discuss the *restricted visibility graph*, $RVG = (V, E)$ which is defined as follows [65]: (a) V is the set of all convex obstacle vertices, (b) an edge (v_1, v_2), for $v_1, v_2 \in V$ represents the fact that the line joining v_1 and v_2 either corresponds to an obstacle edge or does not intersect any obstacle polygon. See Fig. 15 for an example of RVG.

The second structure VD is based on the *Voronoi diagram*, and can be described as follows [67]. Consider terrain O of Fig. 16. The convex hull CH of O is the

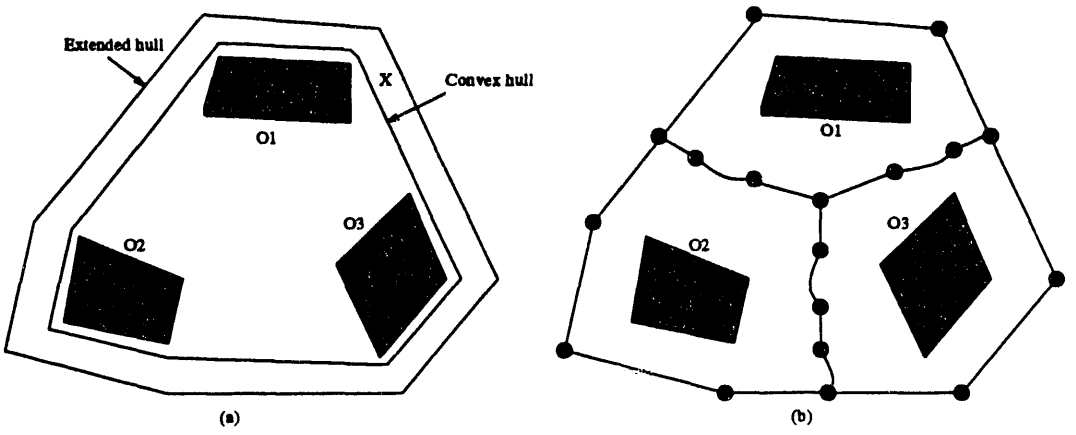


Figure 16: Voronoi Diagram of Terrain

minimal polygonal region that encloses all obstacle polygons. The extended hull EH is the convex polygonal region obtained by pushing out the edges of CH by a distance x. The Voronoi diagram of the terrain is the locus of points that are closest to at least two points on the obstacle boundary. The Voronoi diagram consists of straight line segments and second order curve segments. The VD is obtained by taking the union of the Voronoi diagram contained in EH, and the boundary of EH as in Fig. 16(b).

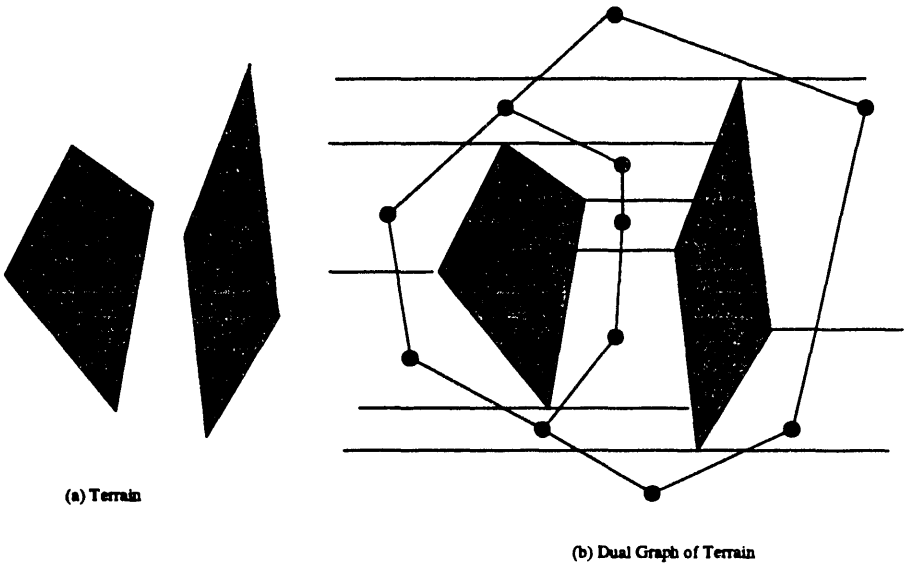


Figure 17: Dual Graph of Terrain

There many other ways of generating navigational courses, based on dual graphs corresponding to decompositions such as trapezoidal decomposition, triangulation, etc., of free space [64]. Consider the terrain of Fig. 17. We decompose the free-space into trapezoids by sweeping a horizontal line. When this line reaches a vertex, we extend (at most two and at least one) line segments from this vertex into free-space until obstacle boundary is reached or to infinity. The free-space is then decomposed

into trapezoids by these line segments. Then a dual graph is obtained by denoting each trapezoid by a node and joining two nodes by an edge if and only if the corresponding trapezoids share a boundary edge.

The terrain model acquisition problem is first formulated and solved for two and three dimensional terrains by Rao et al [66] using the visibility graph as the navigation course. Later the restricted visibility graph obtained by removing concave corners from the visibility graph, is shown to suffice for two-dimensional terrains [65].

Discrete vision sensors are sufficient to navigate in polygonal terrains. But, they are inadequate to navigate in more general terrains, if we are constrained to perform only a finite number of scan operations. For example, if each obstacle boundary consists of a sequence of line segments and circular arcs, then Rao [62] showed that the navigational problems cannot be solved by using discrete sensors; these problem can be solved using Voronoi diagram methods using continuous vision sensors. However, if we stipulate a precision such that the portions where obstacle boundaries are closer than a specified value, are taken as obstacles, then both the navigation problem and terrain model acquisition problem can be solved using the methods of visibility graphs and Voronoi diagrams [62].

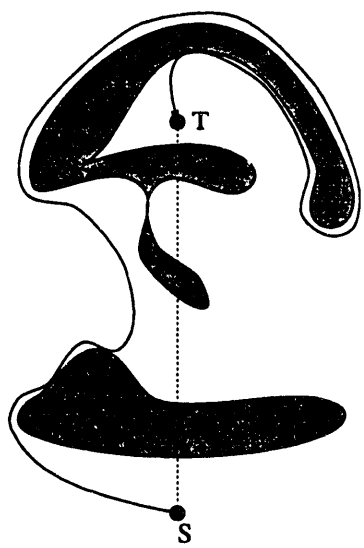
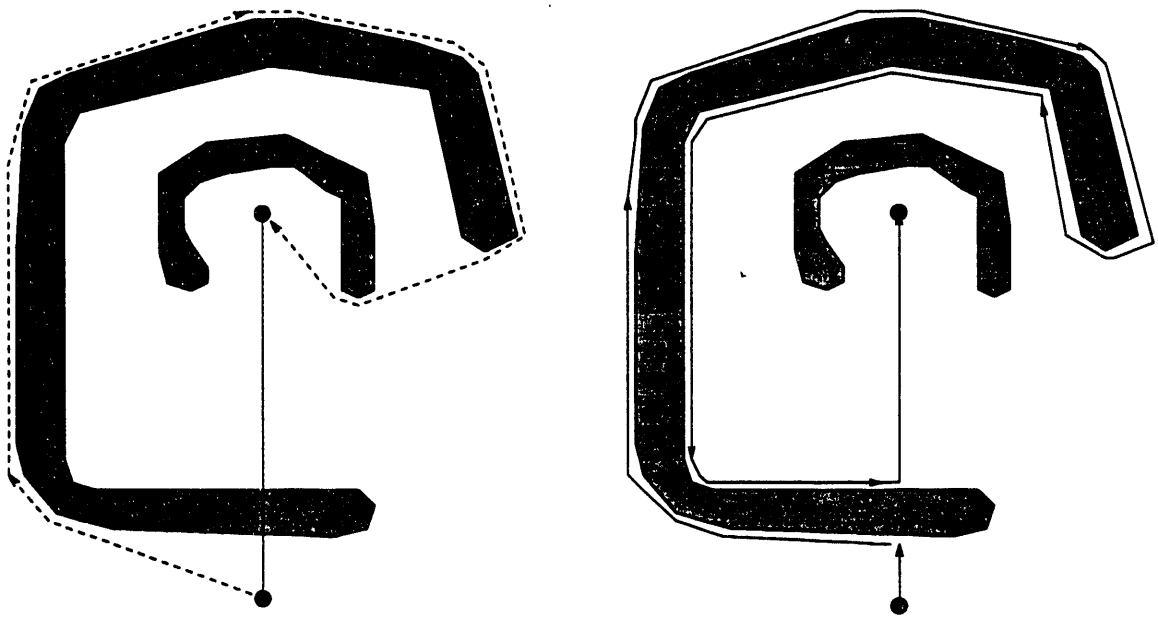


Figure 18: Execution of VisBug21

When the robot is circular in shape, Rao and Iyengar [65] propose a navigation course based on a visibility graph. Here the robot must be capable of performing straight line motion, rotating around the center and around a point on the periphery of the circle. The visibility polygons returned by the sensor is from a fixed point p on the robot, and the robot locates p at certain points in the terrain to perform scan operations.

For a robot of polygonal shape with an ability to translate in any direction, Foux et al. [1993] propose a navigation algorithm. They convert this problem to that of a point robot by using the well-known obstacle growing technique of Lozano-Perez and Wesley [1979]. Their algorithm is based on employing the Dijkstra's shortest path

algorithm on a structure similar to the visibility graph corresponding to the parts of the terrain that have been seen so far. An interesting feature of this work is that they employ a heuristic that the boundaries known to the robot constitute the entire set of obstacles at any stage of navigation; they constuct some portions of visibility graph in the regions that have not been seen so far, based on this heuristic.



(a) execution of VisBug22 (b) execution of Bug2

Figure 19: Execution of VisBug22

5.3 Continuous Vision Sensors

Lumelsky and Skewis [51] discuss navigation algorithms using continuous vision sensors where the sensor has a fixed radius within which it can detect the visible obstacle boundaries. Here the obstacles are simple closed curves (not necessarily polygons); thus the algorithms of last section are not adequate. Two algorithms VisBug-21 and VisBug-22 are proposed to solve the navigation problem. These algorithms are based on the Bug2 algorithm (discussed in section 4.1) that uses a touch sensor.

Informally the algorithm VisBug-21 "mentally" reconstructs within the range of vision the segment of the path that would have been produced by Bug2; then the farthest point on this segment is made an intermediate target, and the robot makes a step towards the target. If the radius of vision is zero, then this algorithm will be identical to Bug2. If the radius of vision is small then the robot moves around the obstacles and gets into the vision of the M-line so that it is seen. On the other hand if the radius of vision is large then the robot does not move close to the M-line. See Fig. 18 for an execution of this algorithm.

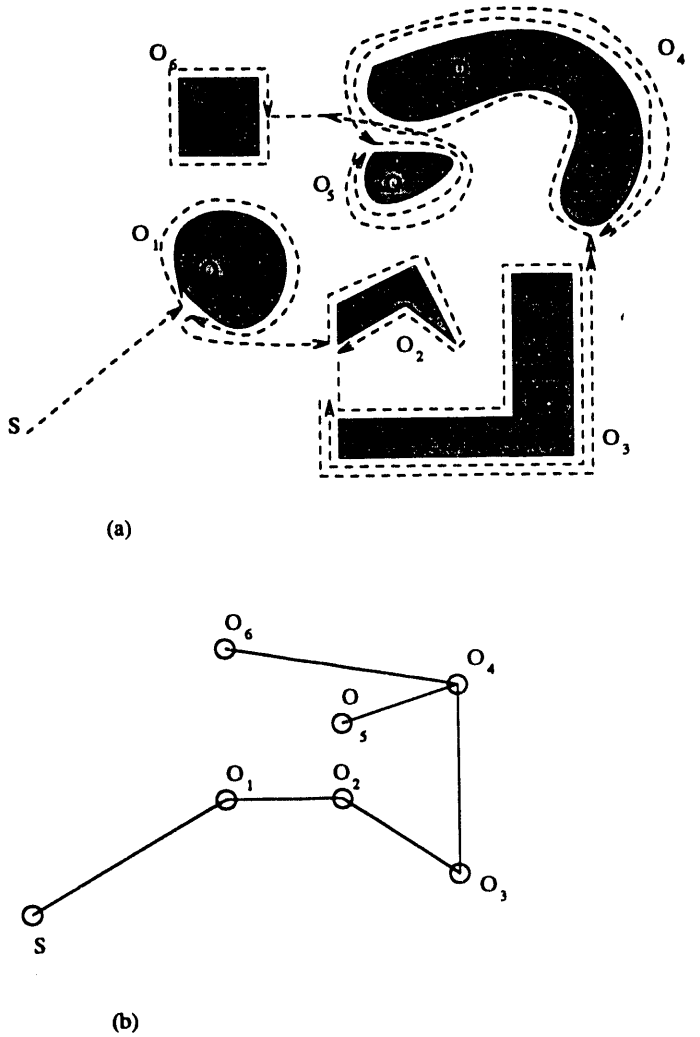


Figure 20: Sightseer Strategy

The second algorithm, VisBug22, does not follow the path of Bug2 completely, but tries to compute intermediate goals on M-line and chooses the goal closest to the target T; the robot then moves to this intermediate goal and repeats the process. An execution of this algorithm is given in Fig. 19(a); here radius of vision is much larger than the diameter of the obstacles. Compare this path to that produced by Bug2 in Fig. 19(b). See Lumelsky and Skewis [51] for more details on these algorithms.

The terrain model acquisition problem using continuous vision sensors with a radius of vision d is considered by Lumelsky et al [50]. Two distinct models of the environment are studied here. In Model (a), the terrain is either finite or infinite and at least one obstacle is visible from the starting position and all the obstacles are mutually visible from each other. In Model (b), the terrain must be finite and the obstacles need not be visible from each other. Two algorithms called the *Sightseer strategy* and *Seed Spreader strategy* are proposed in [50].

The Sightseer algorithm applies to Model (a). Standing at its starting position, the robot scans for a visible obstacle. If no obstacle is visible its job is done. Otherwise,

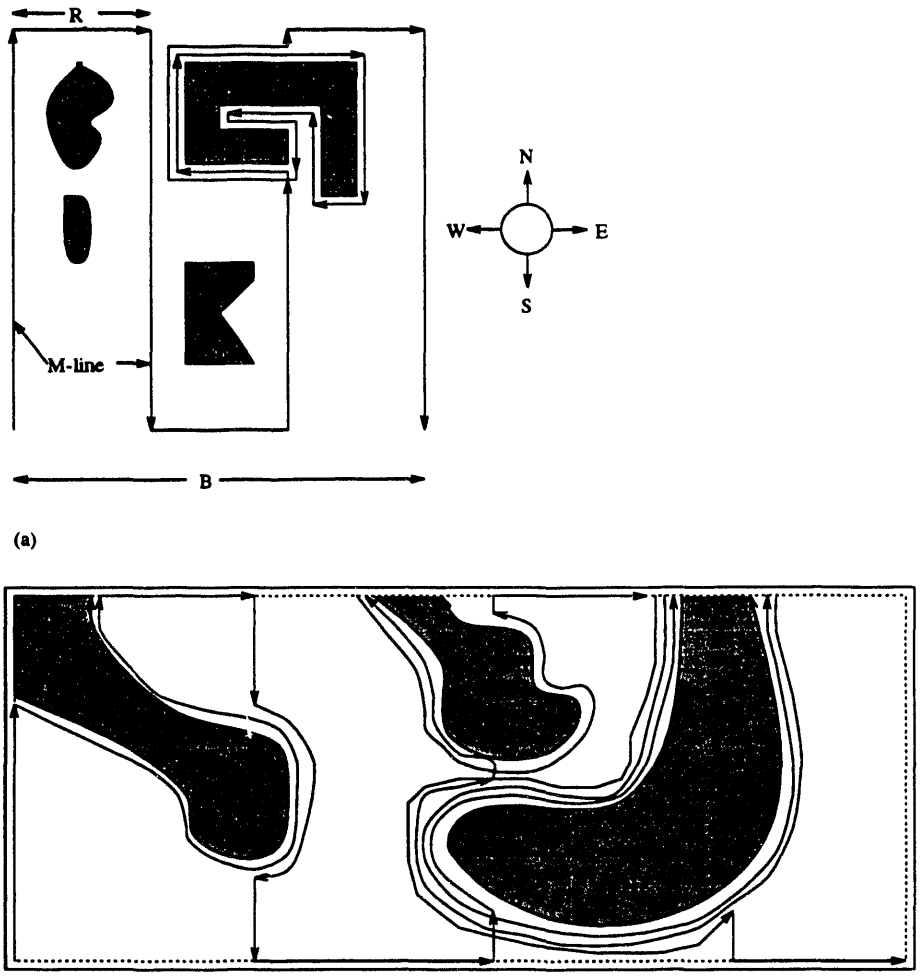
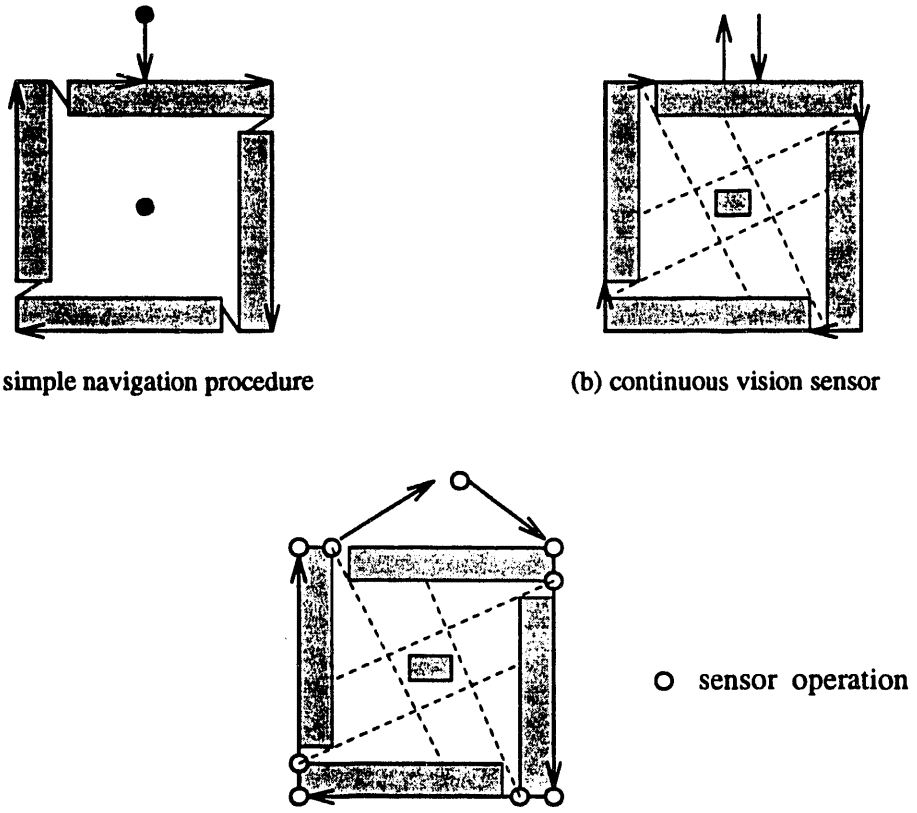


Figure 21: The Seed Spreader strategy

the robot linear navigates towards the nearest obstacle and then circumnavigates it completely and marks it as visited. The robot now chooses the nearest unvisited obstacle from the obstacle it has just visited. Since the next obstacle it has to visit is visible from the current obstacle, the robot does not encounter any other obstacles. When no other obstacle is visible from the current obstacle, the robot backtracks. The process ends when no unvisited obstacles remain. See Fig. 20 for an execution of this algorithm.

The Seed Spreader strategy is applied when the terrain has many obstacles in it and the obstacles are "nicely distributed" and are of "nice geometry". The strategy is to encircle a group of obstacles say by a rectangular path and circumnavigate each of them. We divide the terrain into a number of equidimentional strips and we hope that the obstacles within are wholely acquirable without actually visiting them in the course of navigation around the strip. However when it becomes apparent that an obstacle cannot be wholely acquirable by staying on the perimeter of the strip then the robot moves off the strip to the obstacle and circumnavigates. The path the robot takes on the strip is called the M-line. Various special cases that arise due multiple



(a) simple navigation procedure

(b) continuous vision sensor

(c) discrete scan vision sensor

Figure 22: Test case for simple navigation methods

hit of the same object and due to an obstacle on the M-line are resolved as shown in the Fig. 21.

5.4 A Test Case

Many of the navigation algorithms discussed so far are fairly intuitive, but, many of the intuitive methods that appear to be non-heuristic will not be so after a closer inspection. Consider a point robot with a touch sensor. One simple navigation method for this robot could be: move towards the goal until an obstacle is encountered or the goal is reached; if an obstacle is encountered then turn left and navigate around the obstacle until the robot can move towards the destination again and then repeat the same step (until the destination is reached). In this case the robot will get stuck in an infinite loop around an obstacle configuration in the case shown in Fig. 22(a).

The same test case can be used in the context of a robot with vision sensors. Consider a case where a robot concludes that the terrain model is completely acquired when there are no partially seen obstacles, i.e., all obstacle that have been detected so far are completely seen. Such conclusion would be useful in deciding if the terrain model is completely acquired or the destination is not reachable. This algorithm fails to detect some obstacles in the terrain. In the case of continuous scan shown in

Fig. 22(b), the robot has completely seen the boundaries of all obstacles that it has detected, but failed to detect the obstacle contained in the interior; the seed spreader algorithm of Lumelsky et al [50] fails to acquire the terrain in this case. Similar failure occurs for the discrete vision scan based algorithm of Choo et al [15] as shown in Fig. 22(c).

The configuration in Fig. 22 can be often useful in testing new algorithms for the navigation and/or terrain model acquisition in unknown terrains.

6 Algorithms with Performance Guarantees

In the last few years, there had been an increasing interest in the algorithms that guarantee performance in some way. The algorithms of last two sections only guarantee that they solve the required navigational problems, but are not aimed at guaranteeing any performance parameters.

In this section we consider algorithms that minimize the distance traversed or the figure of merit, which is the ratio of distance traversed to the shortest path length. We first consider the case of minimizing (among a class of algorithms) the total distance traversed by a robot operating in plane. These problems are pioneered by Baeza-Yates et al [3]. Then we consider the algorithms of Papadimitriou and Yannakakis [58], Blum et al [3] and Bar_Eli et al [4] that optimize the above-mentioned ratio.

Some of the algorithms of the last sections provide bounds for certain performance parameters; in fact some of them are optimal within restricted classes of algorithms. But, their main focus is not the optimization of the parameters. Lumelsky [45] proves a lower bound on the distance traversed by the robot in terms of the sums of the boundary lengths of the obstacles encountered by the robot. Bounds in terms of the lengths of the depth-first trees of the navigation course are shown for the discrete vision algorithms by Rao [63]. Also, in the context of discrete vision algorithms, Rao [63] showed that the solutions that implement A^* algorithm are shown to achieve optimal number of scan operations among the class of all solutions that employ admissible graph search algorithms (see Pearl [60] for an extensive discussion on A^* algorithms and admissible graph search algorithms). Among the class I and class II of [73] algorithms based on touch sensing the Bug1 of Lumelsky and Stepanov [53] and Alg2 of Sankaranarayanan and Vidyasagar [72] achieve the optimal worst-case path lengths.

Here we first consider algorithms for searching in a plane due to Baeza-Yates et al [3]; this work is often considered a starting point in the renewed interest in algorithms with performance guarantees. We then discuss algorithms to solve navigation problems that optimize figures of merit. Then we consider an algorithm to solve the terrain model acquisition problem due to Deng, Kameda Papadimitriou [20]. Finally we consider the problem of crossing a street, which is a special case of a navigation problem [40].

6.1 Searching in Plane

Baeza-Yates et al [3] solve several problems dealing with an automaton capable of computation with real numbers in plane. The robot is searching for an object in plane such that for each new probe a cost proportional to the distance of the probe position relative to the current probe position, is incurred. The objective is to minimize the total cost incurred by the robot.

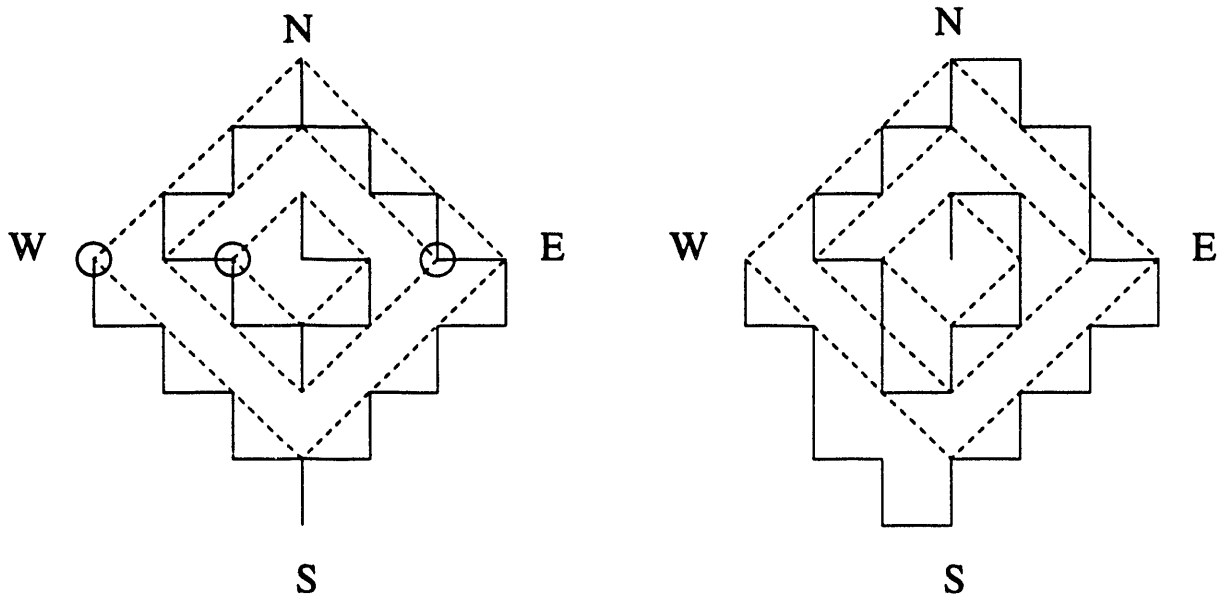
Two basic search problems in the plane are considered in [3]. First consider a robot that is searching for a point at an unknown distance n on a line. The robot can sense the point only when it is directly located above the required point. Any algorithm to solve this problem can be described as function $f(i)$, where $f(i)$ is the number of steps it makes to the left (or right) before the ith turn and where the odd terms are the number of steps to the left and the even terms are to the right as measured from the starting location of the robot. They propose the *Linear Spiral Search* where $f(i) = 2^i$, $i \geq 1$. The total distance walked by the robot is $9n$ steps which is shown to be optimal up to lower order terms.

The second problem consists of starting from the origin and searching for a lattice point located at an unknown point at a distance n (n is also unknown); the robot can move left, right, up or down in one step. Any algorithm for this problem is shown to traverse a distance of $2n^2 + 4n + 1$ steps. They propose *Balanced Algorithm* and *Flipped Balanced Algorithm* that locate the point in $2n^2 + 5n + 2$ steps. The execution of the balanced and the flipped balanced algorithm is shown in Fig. 23. The balanced algorithm can be described as follows. Let a diamond (shown in dotted lines in Fig. 23) of distance a correspond to all lattice points at distance of a units from origin. The algorithm operates in steps. In ith step, the robot visits half of the points of diamond of distance i. The second algorithm is a flipped version of the first. A close observation of the Fig. 23 reveals that the *Flipped Balanced Algorithm* is superior to *Balanced Algorithm.*

In particular the following formulations are discussed when the robot is searching for an unknown line in plane. Consider that the automaton is at the origin in the plane and we are searching for a line that is at a distance of n steps from the origin. The following is a list of results from [3], depending on the information available about the line.

Given	number of steps needed
normal to the line	n
line's distance	$3n$
line's distance and that it is horizontal or vertical	$3\sqrt{2}n$
line's distance	$(1 + \sqrt{3} + 7\pi/6)n$
lines slope	$9n$
line is horizontal or vertical	$13.02n$
nothing	$13.81n$

Given s set of m rays diverging from a common point, the problem of searching for a point located on one of the rays is studied by Baeza-Yates et al [3]; they propose a deterministic algorithm. A randomized algorithm for this problem is proposed by Kao et al [38].



(a) balanced algorithm

NSESWSNWN

(b) flipped balanced algorithm

NESSWSWNN

—— path of the robot

Figure 23: Balanced algorithms for better worst case behavior

6.2 Algorithms with Figure of Merit

Let $R(S)$ denote the distance traversed by a robot in going from s to t in a scene S, and let $d(S)$ denote the length of the shortest path between s and t. Further $S(n)$ denote the scenes in which the Euclidean distance between s and t is n. Following [58], the *figure of merit* for the robot is defined as $\rho(R,n) = \max_{S \in S(n)} \frac{R(S)}{d(S)}$. This figure of merit is first independently studied by Papadimitriou and Yannakakis [58] and Eades et al [23], and later by Blum et al [3] and Bar_Eli et al [4]. When this ratio is constant the algorithm is said to be *competitive* following similar approaches in the area of data structures [77].

6.2.1 Navigation Problem

If s and t are two points in plane and all obstacles are squares, then $\rho(R,n)$ is shown to be at least 1.5, and an algorithm attaining $\rho(R,n) \leq 1.5 + o(1)$ for all n, is proposed by Papadimitriou and Yannakakis [58].

Consider a two-dimensional terrain with rectangular obstacles with their edges parallel to the axes (if the obstacles are allowed to intersect or be of skewed orientation no bounded figure of merit is possible). The robot is equipped with continuous vision sensor. In this case it is shown that no strategy that achieves a bounded ratio is possible [58]. Then terrains populated by square obstacles are considered. In this case they show that there can be no strategy that achieves the figure of merit better

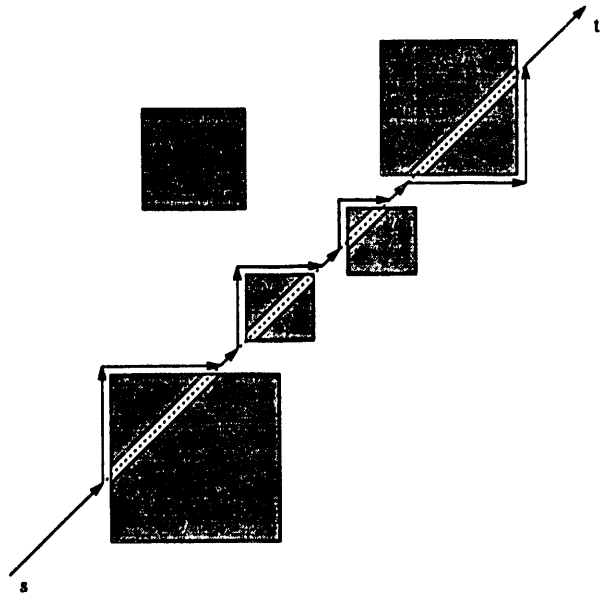


Figure 24: Case of 45 degrees

than $\frac{3}{2}$, which holds for the case of unit square obstacles. They propose an algorithm that achieves a ratio of $\frac{\sqrt{26}}{3}$ by combining two intuitive strategies. First consider the nearest corner heuristic: proceed horizontally and when faced by an obstacle go to its nearest corner. This strategy leads to a factor arbitrarily close to 2. As a second heuristic, consider that the line joining s to t makes 45 degrees then the heuristic that chooses the vertex closer to the line will ensure a ratio of $\sqrt{2}$ as shown in Fig. 24. Then the navigation algorithm uses the nearest point heuristic until the position forms an angle 45 degrees with t, then uses the second heuristic. This mixed heuristic is shown to achieve a ratio of $\frac{\sqrt{26}}{3}$.

Then the special case of unit square obstacle is considered. Define $\epsilon = \frac{1}{\sqrt{n}}$. When faced with an obstacle the robot has bias β towards the corner which is closer to the x-axis. We prefer the corner closer to x-axis if it is less than $\frac{1}{2+\beta}$ away; otherwise, we choose the other corner. Initially $\beta = \epsilon$. Every time we choose the corner farther away from x-axis, we increase β by ϵ, and in the other case we decrease β by ϵ if $\beta = \epsilon$ already. It is shown that this heuristic, called the *bias heuristic* achieves a ratio arbitrarily close to $\frac{3}{2}$ as n grows.

Blum, Raghavan and Schieber [5] formulate the *room problem* where the goal is to traverse from a corner to the center of a square room provided that the obstacles have the form of rectangles aligned with the walls of the room. They propose an algorithm with the figure of merit of $O(2^{\sqrt{\ln n}})$. An algorithm with improved figure of merit of $O(\ln n)$ is proposed by Bar-Eli, Berman, Fiat and Yan [4].

The navigation problem inside a simple rectilinear polygon is studied by Kleinberg [41] who shows that the notion of *essential cut* introduced in Chin and Ntafos [14] is a critical feature in determining the figure of merit. An algorithm for a simple rectilinear polygon with m essential cuts that achieves a figure of merit of $O(m)$ is proposed in [41].

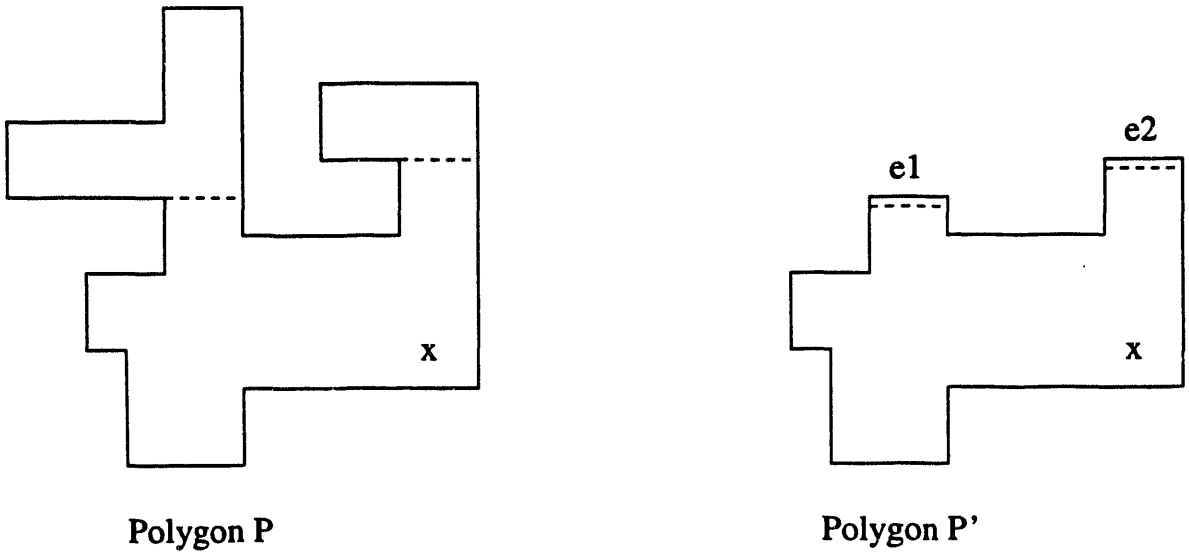


Figure 25: Polygon $P\prime$.

The problem of searching for an object that can be recognized when it is in the field of vision but with unknown location is studied by Kalyanasundaram and Pruhs [37] for terrains populated by convex obstacles. This problem together with the terrain model acquisition problem is studied by them; we provide a brief discussion of their work in the next section.

6.2.2 Terrain Model Acquisition Problem

The problem of terrain model acquisition using continuous vision sensors is studied by Deng, Kameda and Papadimitriou [20]. They show a result that there is no competitive strategy for a polygonal room with arbitrary polygonal obstacles even if all polygons are parallelograms. If the polygonal room contains a bounded number of polygons then they propose a competitive algorithm to explore the interior of general polygonal room populated by a bounded number of polygonal obstacles. This algorithm has a figure of merit of the order of thousands.

If the room is rectilinear and populated by k rectilinear obstacles, then they propose a more efficient algorithm which achieves figure of merit of $O(k)$. Since the general algorithm is quite involved, we will present only an outline of the algorithm for the rectilinear case. This algorithm consists of two parts: first part explores the interior of a polygon which is the boundary of the room and the second part explores each of the exteriors of the obstacles. We now briefly describe the first part. Let P be the boundary polygon. We extend each side s until it hits the side of the polygon, and thus we form a set of *extended line segments.* If it is necessary to cross the segment of s in order to see s, then this segment is called *necessary.* If there is no way to cross e_1 without also crossing e_2, then we remove e_2 from the set of necessary segments. Then we obtain another polygon $P\prime$ from P by removing the farther side of e as shown in Fig. 25.

The algorithm visits all essential segments of P in a clockwise manner as fol-

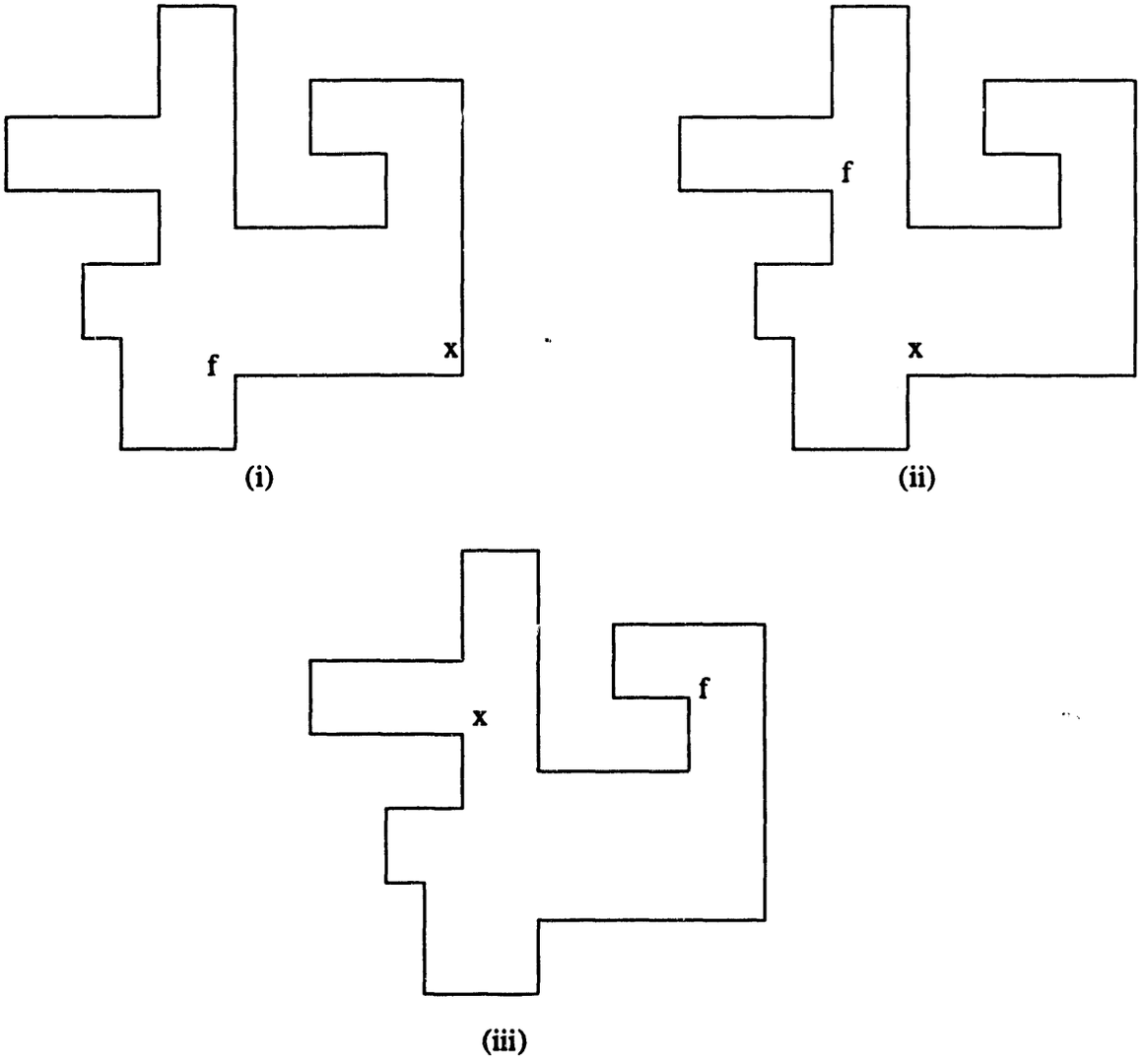


Figure 26: Cases for the algorithm of Deng, Kameda and Papadimitriou.

lows. The strategy is to maintain a current map M of the polygon containing all features that have been seen so far. This map consists of several disconnected pieces of boundary of P. Let C denote the part that contains x_0 and f its end in the clockwise direction as in Fig. 26. Let f lie on the line segment $l(f)$. As C expands f will move in the clockwise direction: it "jumps" when other fragments of the boundary are merged into C. As the algorithm proceeds, M, C and f are updated according to the three following cases. Let $x = x_0$ and M, C and f correspond to portions visible from x_0.

(i) If f can be seen from x, and is at 270 degrees corner, move perpendicularly to $l(f)$ towards $l(f)$ until we arrive at $l(f)$ or its extension. If another part of the boundary is encountered, we move parallel to $l(f)$ towards f as necessary.

(ii) If f can be seen from x, and is at an interior point of the line segment $l(f)$, then we follow step (i).

(iii) If f cannot be seen from x, then compute the shortest path from x to f on which f becomes visible as early as possible and follow it until f is visible.

The cases (i) through (iii) are illustrated in Fig. 26.

A lower bound of $\sqrt{2}$ for the figure of merit is established for this problem in [20] and it is open question to bridge the gap between the bounds. This deterministic algorithm achieves a figure of merit of 2, and Kleinberg [41] reduced it to 5/4 using a randomized algorithm. See [20, 41] for a discussion on various interesting open problems on this topic.

For the terrain composed of convex polygonal obstacles, the terrain model acquisition problem is studied by Kalyanasundaram and Pruhs [37]. The *aspect ratio* of a convex polygonal object O_i is defined to be R/r where R is the radius of the smallest circle that circumscribes O_i and r is the largest circle that inscribes O_i. Consider a terrain of k convex polygonal obstacles. Let α and $\bar{\alpha}$ denote the maximum and average of the aspect ratios of the obstacles. Then let $M(k, \bar{\alpha})$ be defined as $\min(k, \sqrt{k\bar{\alpha}})$. We now briefly consider the works of Kalyanasundaram and Pruhs [35, 37]. They show a lower bound of $\Omega(M(k, \bar{\alpha}))$ for the figure of merit of any algorithm. Then they study three algorithms Nearest Neighbor, Bifold Nearest Neighbor and Tourist. In the Nearest Neighbor strategy, the robot from the start position, picks the nearest obstacles, and moves to it and then circumnavigates it. Then in each step, it picks the obstacle nearest to the present obstacle (that has just been circumnavigated), and moves to it and circumnavigates it. The figure of merit of this simple strategy is shown to be $\Omega(\alpha M(k, \bar{\alpha})$ and $O(\alpha \log k M(k, \bar{\alpha}))$. The Bifold Nearest Neighbor algorithm operates in phases; in each phase it explores the terrain contained in a square R_i region centered at the start location. In the next phase, the R_{i+1} is obtained by doubling the length of the side of R_i. In each phase, the Nearest Neighbor algorithm is used with modifications such that the next obstacle to be explored must interset R_i. This method is shown to perform better than Nearest Neighbor with the figure of merit bounded by $\Theta(\log k M(k, \bar{\alpha}))$. The other algorithm Tourist operates on a graph based on a grid superimposed on the terrain in phases that operate on exponentially

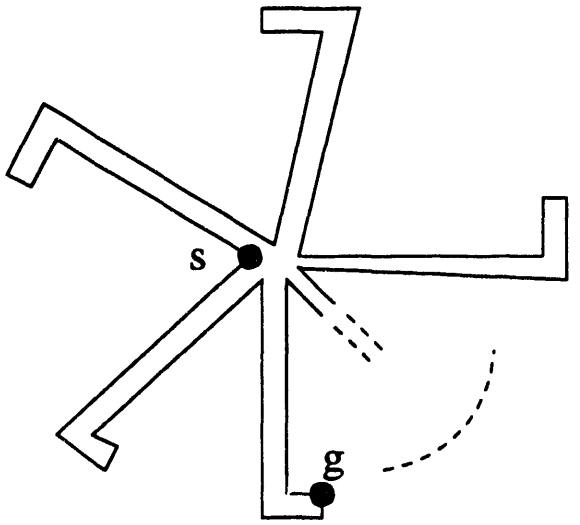


Figure 27: Illustration of unbounded value for figure of merit.

growing neighborhoods. This algorithm is shown to be optimal with figure of merit of $\Theta(M(k, \bar{\alpha}))$.

Kalyanasundaram and Pruhs [36] solve the visual traveling salesperson problem which requires that the perimeter of every obstacle be traversed. They present an on-line algorithm which achieves path lengths no more than 17 times the shortest possible path if the terrain model is available. Using this algorithm, the robot can obtain the entire terrain model, but such tour is not always necessary to map the entire terrain model. For example, it is sufficient to visit convex obstacle vertices to obtain the terrain model (by the terrain-visibility property of the restricted visibility graph described in Section 5.2). This algorithm is justified when a close proximity to obstacle is needed in order to accurately map the terrain boundaries (see [36] for an interesting discussion on these aspects).

6.2.3 Walking an Unknown Street

Along the same spirit of the algorithms last subsections, Klein [40] studied the problem of navigating from a source vertex s to a goal vertex g inside a simple polygon P (with no holes). Here the robot uses a continuous vision sensor such that newly encountered convex vertices can be detected as the robot is in motion. This problem is akin to that encountered by a human being finding a place in an unknown city. The main objective is to minimize $D_S(P)$ the ratio of length of the path traversed by the robot using strategy S to that of a shortest path. It can be easily shown that if the polygon is general, then this ratio can be unbounded; for example in Fig. 27 the robot may have to explore every branch before it sees g. Let L and R denote the oriented boundary chains leading from s to g. Then P is called a *street* if and only if L and R are mutually weakly visible, i.e., if each point of L can be seen from at least one point of R and vice versa. See Fig. 28 for an example of a street. For this special case, Klein [40] proposed an algorithm with $D_{lad}(\mathrm{P}) < 1 + \frac{3}{2}\pi$.

Each point of L can be connected to some point of R by a line segment contained

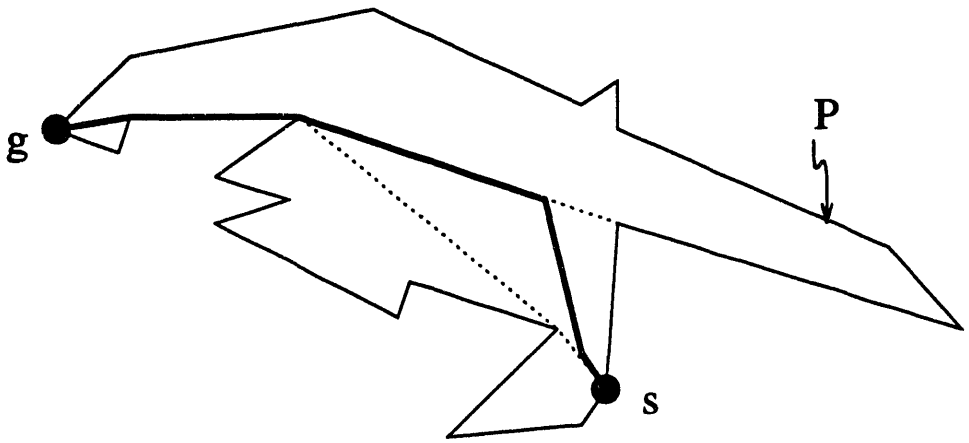


Figure 28: Execution of Klein's algorithm for walking a street.

in the polygon, and vice versa. The basic idea of this algorithm is to cross all these line segments on the way from s to g. This algorithm consists of a *high-level* and a *low-level* strategy. The former finds the path subject to the following invariants: at each position p on the path either the robot can see the goal, or which of the visible corners ahead is visited by the shortest path from s to g or robot can identify two corners ahead one of which is visited by the shortest path. In the last case, the robot uses the low-level strategy to choose a point t on the line joining these two vertices. In the first case the robot moves to g, and in the second the robot moves to the identified vertex visited by the shortest path. See Fig. 28 for an example.

Recently, Kleinberg [41] proposed a method that improves the figure of merit of Klein's algorithm by more than a factor of 3. If P is unknown and minimization of $D_S(P)$ is not a primary criterion, then this problem can be easily solved using the restricted visibility graph or Voronoi diagram methods of Rao [61]. If the terrain model is known, the shortest path can be computed with a complexity of $O(\log n + k)$ time with a preprocessing step of complexity $O(n)$ by Guibas and Hershberger [29], where n is the number of edges of P and k is the number of line segments of a shortest path.

7 Restricted Computational Models

Algorithms discussed in the last three sections assume that the robot can store and manipulate real numbers. Even in the case of Pledge algorithm, the robot is required to store and manipulate real numbers. In particular if the robot is capable of performing real arithmetic, then the problem of exploring a maze can be easily accomplished using algorithms of Rao [61] and Lumelsky [45]. Some of the fundamental questions of this section deal with capabilities of robots that are computationally less powerful than those assumed in earlier works such as [61, 45].

Informally a *maze* is a finite, two-dimensional, obstructed checkerboard as shown in Fig. 29. A finite automaton is a device that can only be in a finite number of states and operates by reading from a finite set of input symbols (see [32] for a formal

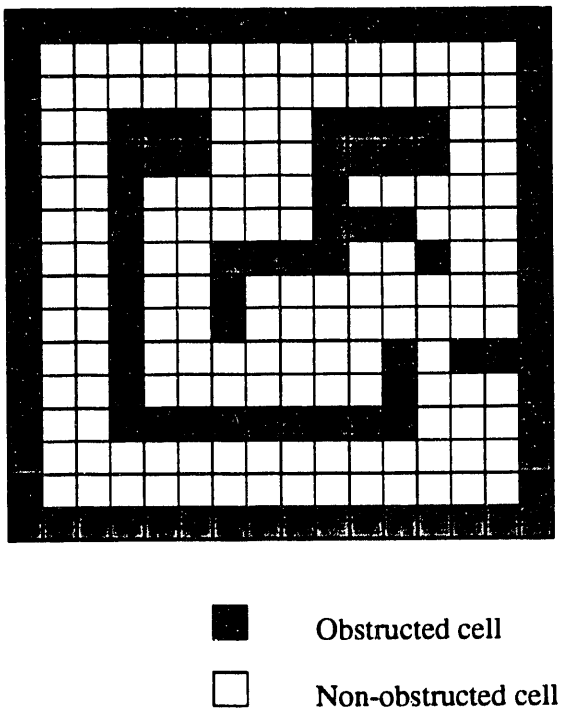


Figure 29: Example of a maze.

definition of finite automata). A finite automaton consists of finite control with start and halt states and a transition function. The automaton can be in any state chosen from a finite set of states. At any step, the automaton looks at input and its current state and decides the next state specified by a transition function. In the context of maze searching, the automaton determines which of its neighbor cells are reachable in one step and then, depending on its state, moves north, east, south or west one cell. In 1974 Lothar Budach gave a proof that no single finite automaton can search all mazes (this rather long proof runs for 175 manuscript pages) [10]. Here "search" means visiting all cells of a maze; note that using such search the automaton could find "cheese" stored in an unknown cell. Thus the power of a finite automaton is inadequate to search the mazes.

We first describe the algorithms of Coy that use more powerful computational models than a finite state automata. Then we consider the case of finite automata, but the robot is equipped with pebbles that it can drop in a cell, recognize and pick up; these pebbles are used as markers on certain cells (when the coordinates of cells cannot be computed). Shah [75] illustrated an algorithm for a finite automaton that uses five pebbles. Then Blum and Kozen [7] showed that a finite automaton with two pebbles can achieve the same result. The algorithms for finite automata with pebbles are described briefly in the last part of this section.

7.1 Algorithms of Coy

Dopp posed the following question [22]: does there exist a finite automaton that finds a way out of every finite open maze from any initial position and finally moves arbitrarily far away ? Muller [54] and Budach [10] gave negative answers to this question; the latter used the original formulation of Dopp, and Muller used a graph-theoretic variation of the formulation. Coy [19] considers the cases of finite automaton, pushdown automaton, linear bounded automaton and a Turing machine; in terms of computational capabilities this sequence represents strictly increasing power (see Hopcroft and Ullman [32] for a formal discussion on these models). Informally, a finite automaton equipped with a stack is called a push down automaton; the computations performed by the former can be performed by latter and not vice versa, i.e., there are computations that are performed by push down automata that cannot be performed by finite automata. The Turing machines represent the most powerful of these four models, and a linear bounded automaton is a Turing machine constrained to use a storage whose size is proportional to the size of the input. Then the answer to Dopp's question is negative for the case of finite state automata and push down automata. This problem can be solved by a linear bounded automata (hence by a Turing machine). See Coy [19] for details.

We now briefly and informally describe the algorithm of Coy [19] for a Turing machine. First a Turing machine can be (informally) visualized as a finite state automaton equipped with an infinitely long tape of cells. The read/write head of the Turing machine scans one cell at a time; in a *move* a Turing machine can read the symbol in the cell and write a symbol on the cell and move to the left or right. Each move is realized by a control that resembles the finite state machine. An algorithm corresponds to the finite state control, and each execution of such algorithm uses working space in terms of cells written/read on the tape. The input to the Turing machine is given as a symbols written on a sequence of cells on the tape which the Turing machine reads. A Turing machine is called a *linear bounded automata* if the working space (number of tape cells) used is restricted to the length of the input (in terms of number of cells).

We can think of an algorithm which allows the construction of a robot equipped with Turing machine that is able to reach from any initial position in a maze any position in the maze as follows. An arbitrary maze can be described as an infinite tree with an initial node P_0 (initial placement of the robot). This infinite tree has a finite subtree such that every node of the maze is represented by at least one node of the finite subtree. If the robot is able to start in P_0 walk to a node P_1 and return to P_0 and if the robot may enumerate the paths from P_0 to all nodes P_i in the finite subtree then it will visit all the cells of the maze. This task is done by the robot by enumerating (in base 4), in increasing order, every path from P_0 to some P_i of the finite subtree may be described by a natural number $w_k, \ldots, w_1$. Here each direction w_i specifies the next cell to be visited by the robot, and the robot takes the direction w_1 first , w_2 next and so on to reach w_k. To reach P_0 again, it will simply read $w_k, \ldots, w-1$ backwards. As there are only finitely many paths in the subtree which covers all cells of the maze, there exists an upper bound $w_o > w$ on the numbers to be

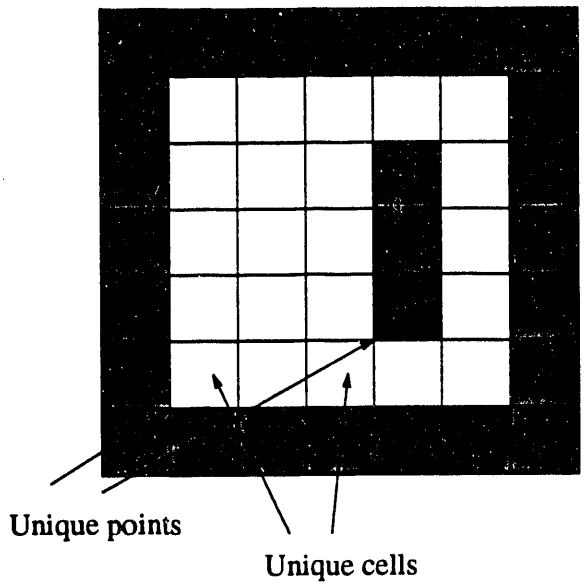


Figure 30: Unique points and cells.

generated. The result that a Turing machine can solve the problem posed by Dopp is provided by Dopp himself. This algorithm of Coy actually shows that a linear bounded automaton is sufficient to solve this problem.

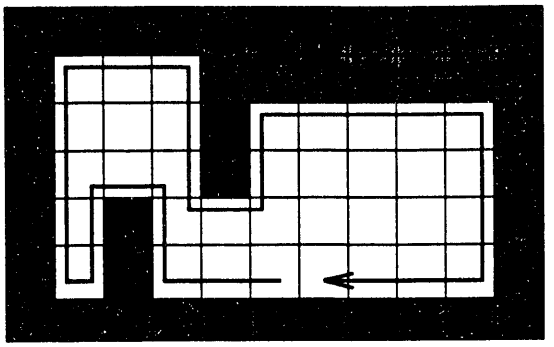
7.2 Automata with Pebbles

Automaton with pebbles have been first investigated by Blum and Hewitt [6] and then subsequently by Mylopoulos [55] and Savitch [74]; these automata have been studied in image processing applications in Shah [75] and Rosenfeld [69]. Although the early focus of these automata is not directed towards robotic applications, the underlying principles shed some light on some navigation algorithms.

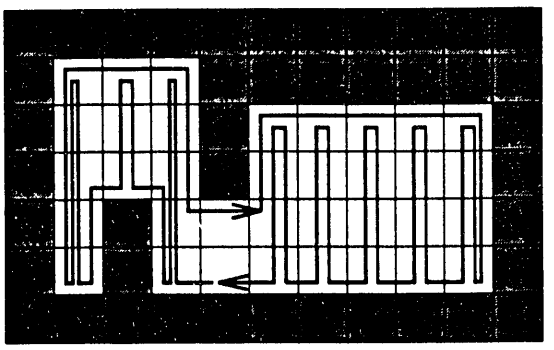
As in the last section finite automaton consists of finite control with start and halt states and a transition function. We may equip the automaton with a finite number of *pebbles* each with a unique name. At the start, the automaton is carrying all the pebbles. There after, it is always carrying some subset of its pebbles and the rest are lying on non-obstructed cells of the maze. In each step the automaton determines the names of the pebbles it is carrying and the names of the pebbles lying on the cell it is visiting. It may use this information to help determine its next transition. In each step the automaton may pick up pebbles from the cell it is visiting or deposit some.

An automaton can also be equipped with a *counter*. A counter holds a non-negative integer and can be initialized to zero. In each step, the automaton can increment or decrement the counter by one and test for zero.

An outline of Shah's algorithm for searching a maze is as follows. Let m_i, $i = 1, 2, \ldots, 5$ denote the ith pebble. The robot finds the leftmost point of the topmost row using three pebbles. Then the robot traverses a complete row at a time. A horizontal sequence of non-obstructed cells is called a *segment* if the neighboring cells of the leftmost and right most cells are obstructed. The pebble m_5 is placed on the



(a) traversal of boundary



(b) traversal of the maze

Figure 31: Traversal of simply connected region.

right end of a segment; the m_4 is dropped at the leftmost segment to the right of m_5. Then it is shown that m_4 and m_5 will be on the same border. Now the robot scans the new segment left to right and repeats the process of finding the leftmost segment to the right of the current segment and in the row as the current segment. Therefore the robot scans every segment of a row until it finds that there is no segment to the right of the current segment. This means that the current row is scanned completely, so the robot moves to the next lower row, if any. If there is no row below the present row then the robot has scanned all the rows, therefore it halts.

Blum and Kozen [7] show a theorem that there exits a finite automaton with one counter which can search any maze and halt. As a corollary to this theorem they show that there is a finite automaton with two pebbles which can search any maze and halt, and also that there are two finite automata which together can search any maze and halt. Their algorithm is based on Unique Point Lemma which can be informally presented as follows. The *unique point* of a boundary, $BDRY$ is the unique point $(x_0, y_0) \in BDRY$ such that for all $(x, y) \in BDRY$, either $y_o \leq y$, or $y = y_0$ and $x_0 \leq x$. A *unique cell* of a boundary is the unique white cell whose NE or SW vertex is the boundary's unique point. See Fig. 30 for examples. Then the lemma states that there exists both 2 pebble and 1 counter automata that one may place on any white cell C, of a maze together with 2 pebbles or an empty counter in state q_{NE} (or

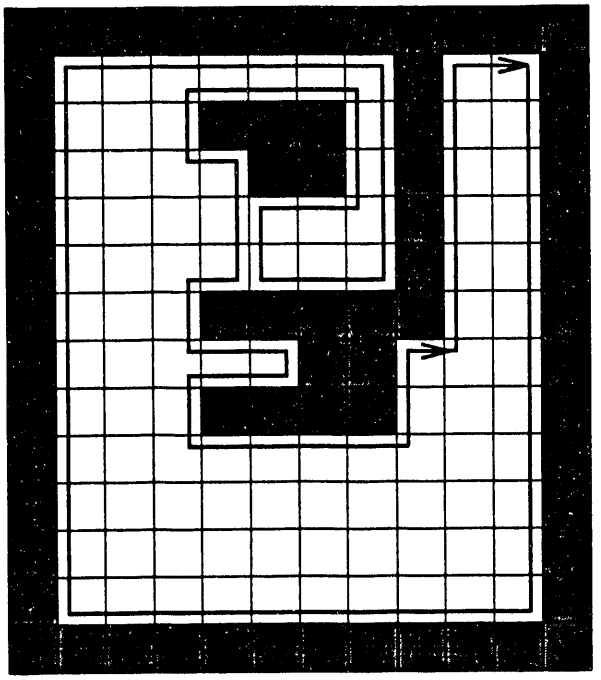


Figure 32: Execution of algorithm of Blum and Kozen.

q_{SW}). The automaton, after some moving about in the maze will return to C with its 2 pebbles or empty counter and stop in state q_{NE}^{YES} (q_{SW}^{YES}) if the NE (SW) vertex of C is a unique point of a boundary and in q_{NE}^{NO} (q_{SW}^{NO}) if not.

A *barrier* is any maximal connected set of obstructed cells, and the unique infinite barrier is called the *border*. The *boundary* of a barrier is the set of edges that separate black cells from the white cells called *boundary cells*. Let us assign to every vertex (corner point of a cell) of a maze a color. Unique points are colored green and all other vertices are to be colored white. Then they define a green-eyed automaton as a finite automaton so that in any cell and in any state there, the automaton can determine which, if any, of the cell's 4 vertices are green.

If all unobstructed cells form a simply connected region, i.e., if their only barrier is the border, then a finite automaton can visit all boundary cells by moving from one cell to the next, keeping the boundary always on the left. To search an entire simply connected maze, modify the above procedure so that each time the automaton steps from one boundary cell to the next, it first goes into a subroutine that causes the automaton to move north until it reaches a barrier and then to return south whence it came before going on to the next step. This way, each white cell interior to a simply connected maze gets visited immediately after the white cell beneath it gets visited. See Fig. 31 for illustrations of these algorithms.

Any maze can be converted to a simply connected one by relabeling all vertical edges that lie between each green vertex and the barrier immediately beneath it as boundary edges. Once this boundary is identified, we use the same algorithm for the simply connected region. See Fig. 32 for the execution of the algorithm. In order to halt, the finite green-eyed automaton has to check that it twice visited the (only)

border cell having a green vertex. The rest of the algorithm is identical to the case of simply connected region.

Blum and Kozen [7] also prove that (a) there are two automata which together can search all mazes; (b) there is a logspace algorithm to search mazes, a vast improvement over the naive linear space algorithm which constructs a map of the maze.

8 Exploring Unknown Graphs

This section is intended to provide some results in graph exploration algorithms that can illustrate that some search problems that are easily accomplished in searching a geometric terrain become computationally very hard when required on graphs. A complete survey of graph algorithms for searching unknown graphs is not intended here; we only discuss some graph problems that appear to be very similar to the robot navigation algorithms in unknown terrains.

In terms of searching by an automata of the type of last section, Blum and Sakoda [8] posed the question of weather it is easier to search mazes than planar graphs (planar graphs are the graphs that can be embedded in plane such that no two edges intersect [30]). Mazes and regular planar graphs (planar graphs where each node has the same number of neighbors) appear similar on surface, but they differ substantially. The main difference is that an automaton in maze has a compass that can distinguish N, E, S, W directions. A compass can provide valuable information. Blum and Kozen [7] show that no single finite automaton can search all finite planar graphs; they show that no automaton can search the subclass of planar called the cubic graphs. They further show that no three automata can search all planar cubic graphs. Thus they show that in terms of the finite automata the mazes are easier to search than graphs.

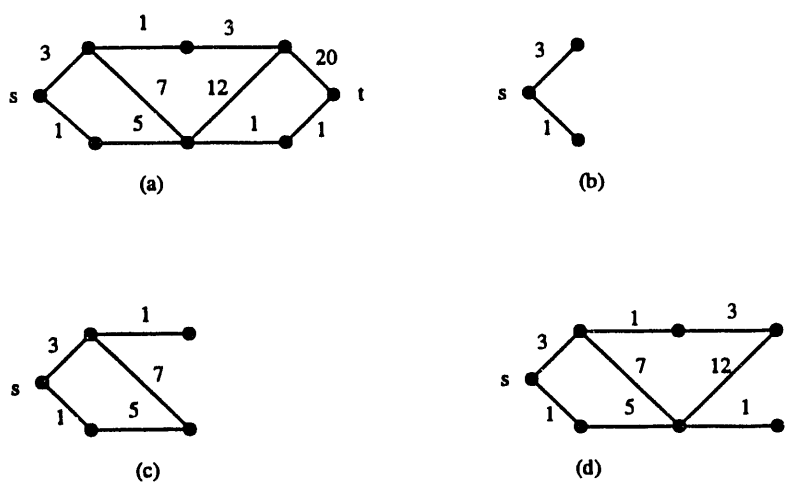


Figure 33: Searching layered graph of width two

Papadimitriou and Yannakakis [58] first posed the problem of searching a layered graph that is dynamically specified. A layered graph is a graph in which the nodes are partitioned into layers $L_1, L_2, \ldots, L_n$ and all edges are between adjacent layers. The edges between L_i and $L_i + 1$ and their lengths become known only when a node in

L_i is reached; the number of layers is also unknown. Edges can be traversed forward or backwards, and the lengths are non negative integers when we switch from a node u to node v, we we incur a cost proportional to the distance of the path from u to v. Fig. 33(a) shows a layered graph, where a shortest path from node s to node t is desired. If the graph is completely known in advance, then a path can be computed using dynamic programming methods or using the more general Dijkstra's algorithm [2]. The graph is dynamically given in that it is given one stage at a time. In the beginning only part shown in Fig. 33(b) is known. In the next stage the known part of the graph is shown in Fig. 33(c). Then a natural choice in this case would be to make the lower choice in the first step; then the lower edge of length 5 is revealed. A reasonable strategy could be to persist on the present path until there is a path on the other side of length less than half of the present one. In the second step, the algorithm persists on lower choice in Fig. 33(c), but switches to upper choice in the next step as in Fig. 33(d). This strategy is shown to be optimal for two-layered graphs and achieves a worst-case ratio of distance traveled to shortest path equal to 9, the best possible.

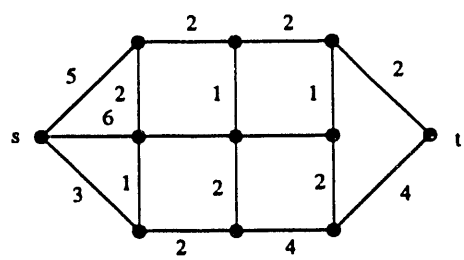


Figure 34: Canadian traveler's problem

Further work on the layered graph traversal is done by Fiat et al [25]. The *width* of layered graph is $\max\{|L_i|\}$. They give a deterministic on-line algorithm which achieves the ratio of $O(9^w)$ for graphs with width w. Several other related results are provided in [25]

Papadimitriou and Yannakakis [58] also discuss *Canadian Traveler's Problem.* Consider Fig. 34. The road map (a graph) is known to us, but some roads (edges) could be unsuitable to travel due to snowfall; but, the status of a road is revealed only when an adjacent node is visited. They address the problem of devising a strategy which guarantees a given ratio to the shortest path. This problem is shown to be PSPACE-complete, which indicates that this problem is computationally very hard (see [2] for details on PSPACE-complete problems).

Deng and Papadimitriou [21] consider the following problem of exploring a directed and strongly connected graph. At each point we have a map of all nodes and edges that have been visited, and these nodes and edges can be recognized if they are visited again. We know how many unexplored edges emanate form each node we have visited, but cannot tell where each edges leads until we follow it. The objective is to minimize the ratio of the total number of edges traversed divided by the optimum number of edge traversals, had we known the graph. A graph is *Eulerian* if there exists a path that visits each edge precisely once. For Eulerian graphs the ratio cannot be better

than two and can be achieved by a simple algorithm [21]: (a) take unexplored edges whenever possible; (b) if stuck, consider the closed walk of unexplored edges just completed, and retrace it, stopping at nodes that have unexplored edges, and apply this algorithm recursively from each such node. The *deficiency* of a graph is the number of edges to be added to make it Eulerian. An algorithm that achieves a bounded worst-case ratio when the deficiency is bounded is given in [21]. They also show that if partial information about the graph is given, minimizing the worst-case ratio is PSPACE-complete. An illuminating example of learning by a child and its connection to the exploration of an unknown graph is given in [21].

Given a weighted planar graph G such that the nodes adjacent to a vertex v are revealed when v is visited, the problem of visiting all vertices of G is studied by Kalyanasundaram and Pruhs [36]. They propose an algorithm with a cost no more than 16 times the length of the traveling salesperson tour, which is the shortest path that runs through all nodes of a graph and comes back to the start node. This algorithm runs in $O(n^2 \log n)$ time on a graph of n nodes. Intuitively, this algorithm performs depth-first search on small regions of the graph and occasionally jumps between such regions.

9 Conclusions and Open Problems

Algorithmic approaches for navigating robots in geometric terrains populated by unknown set of obstacles are considered. Here the terrain model is not a priori known, but the robot is equipped with a sensor system that is employed for the purpose of navigation. We are interested in *non-heuristic algorithms* that can be theoretically shown to be correct within a given framework of models for the robot, terrain and sensor system. These methods are abstracted and simplified compared to real-life scenarios. But, they yield useful results in (a) providing foundations for practical systems by highlighting the underlying critical issues, and (b) concentrating on some central aspects. We broadly classified these algorithms into three categories. First, we considered the algorithms that are shown to navigate correctly without much consideration for the performance parameters such as distance traversed, etc. Second, we considered non-heuristic algorithms that guarantee bounds on the distance traversed or the ratio of the distance traversed to the shortest path length (computed if the terrain model is known). Then we considered robots with very limited computational capabilities such as finite automata, etc.

In spite of the long history of the non-heuristic algorithms, this area is generally considered to be in its infancy; compared to its counterpart in known terrains, many issues of this area are open for further investigation. Some of the topics for future study can be described as follows.

(a) **Sensory systems:** Most of the existing non-heuristic algorithms are based on ideal sensors. It would be interesting to study algorithms that can perform in the presence of sensory errors. Also, there are several sensors such as ultrasonic and laser range finders that can measure the distance to an obstacle in a given direction. There are no non-heuristic navigation that can guarantee that the

robot can navigate to a destination using sensors that return the distance to the obstacle along a discrete direction (if only a finite number of distance probes are allowed). The problem of inferring a shape of a polygon by probing in a number of directions has been solved by Cole and Yap [16]; works along the same spirit that enable a mobile robot to infer its environment by using results of probe operations will be of interest. It has been observed in practice that many non-trivial navigational tasks require a systems of sensors, since a single sensor is of limited utility (see special issue edited by Brady [9]). The problem of navigating using a system of similar or disparate sensors could be investigated; in general it may be more efficient to use an array of laser range finders or ultrasonic sensors in solving the navigation problems.

(b) **Robot Systems:** Majority of the algorithms in this survey are restricted to point robots (exceptions include circular robot of Rao and Iyengar [65] and polygonal robot of Foux et al [26]). In known terrains, the problem of navigating a non-point robot can be reduced to that of a point robot in configuration space amidst "suitably grown" obstacles. It is unclear if such method is directly applicable to the present case since we need to ensure that the required portions of the configurations must be incrementally constructed based on the sensor readings. Also when non-point robots are considered, the motion primitives of the robot must be given explicit consideration; for example, a car can move along certain paths but cannot move along arbitrary curves. Such problems have been investigated in the known terrains formulations under the title of non-holonomic path planning. Such works are needed for unknown terrains formulations in order to yield practical implementations for real-life robots.

(c) **Terrain Models:** Most of the terrains discussed here are two-dimensional (with some exceptions such as Rao et al [66]); it would be interesting to see if some of the techniques of 2-dimensional terrains can be extended to 3-dimensional terrains. Also, for algorithms that store the terrain models, most works deal with polygonal terrains or mazes; such methods for non-polygonal terrains will be of interest. Terrains with moving obstacles could be another topic for future investigation.

(d) **Performance Parameters:** In many real-life robot systems, the sensor operations could be memory intensive and computationally time-consuming. For example, simulating a 360 degree scan using a vision system would be very expensive. In such applications, algorithms that perform less number of scan operations would be preferred. Some preliminary work that shows that among the class of admissible graph search algorithms, the A^* algorithm uses least number of scan operations has been presented in Rao [63]. However, more general results are needed in order to be of practical value. The algorithms that guarantee bounds on the distance traversed or figure of merit are being actively pursued.

(e) **Systems of Robots:** In general, it would be interesting to see if some navigational objectives can be better achieved by employing more than one robot.

Apart from the method of Blum and Kozen [7] that enables two automata to collectively search a maze, we are unaware of non-heuristic navigation algorithms that employ a system of robots.

(f) **Robot Learning:** If a robot is required to visit a sequence of destination points (in stead of just one), then the robot can potentially store the terrain model in the places that it has been and use this information in improving the performance of the subsequent navigation. In general, the robot can use the terrain information to avoid performing sensor operations in known regions and also avoid getting into local detours. This type of learning is referred to as *incidental learning* by Rao [61] (see also Oommen et al [56]). Also in the case entire terrain model is completely built, the robot can switch off sensors and navigate in optimal paths (when possible). A formal treatment of incidental learning can be very useful in practical systems. At present this area has been investigated only to a limited extent.

(g) **Potential Field Methods:** In known terrains, the potential field methods pioneered by Khatib [39], have been found to be very useful; exact navigation algorithms based on these methods are described in Rimon and Koditschek [68]. In these methods, a field is used to guide the robot to the destination; this field consists of an attracting field located at the destination and a repulsive field due to obstacles. It would be interesting to see if any of these methods (in particular local minima free fields) can be computed incrementally so as to yield non-heuristic algorithms for unknown terrains.

References

[1] H. Abelson and A. diSessa. *Turtle Geometry.* MIT Press, Cambridge, MA, 1980. pp. 191-198.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison- Wesley Pub., Reading, MA, 1974.

[3] R. A. Baeza-Yates, J. C. Culberson, and G. J. Rawlins. Searching in the plane. *Information and Computation*, 1991. to appear.

[4] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan. On-line navigation in a room. In *Proc. 3rd Annual ACM-SIAM symposium on Discrete Algorithms*, pages 237–249, 1992.

[5] A. Blum, P. Raghavan, and B. Schieber. Navigation in unfamiliar terrain. In *19th 23rd Symposium on Theory of Computing*, pages 494–504, 1991.

[6] M. Blum and C. Hewitt. Automata on a two dimensional tape. In *Proceedings of Eigth IEEE Conference on Switching and Automata Theory*, pages 155–160, 1967.

[7] M. Blum and D. Kozen. On the power of compass (or why the mazes are easier to search than graphs. In *18th Annual Symposium on Foundations of Computer Science*, pages 132–142, October 1978.

[8] M. Blum and W. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *17th Annual Symposium on Foundations of Computer Science*, pages 147–161, October 1977.

[9] M. Brady. Forward to the special issue on sensor data fusion. *International Journal of Robotics Research*, 7:2–4, 1988.

[10] L. Budach. Automata and labyrinths. unpublished manuscript, 1974.

[11] L. Budach. On the solution of the labyrinth problem for finite automata. *EIK*, 11(10-12):661–672, 1975.

[12] J. Canny. *Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1987. doctoral dissertation.

[13] R. H. T. Chan and P. K. S. Tam. Environmental model acquisition and robot navigation in unknown terrains. In *Thirteenth IASTED International Symposium on Robotics and Manufacturing*, pages 5–8, 1990.

[14] W. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6:9–31, 1991.

[15] C. Y. Choo, J. M. Smith, and N. M. Nasrabadi. An efficient terrain model acquisition algorithm for a mobile robot. In *Proceedings of 1991 IEEE International Conference on Robotics and Automation*, pages 306–311, 1991.

[16] R. Cole and C. K. Yap. Shape from probing. *Journal of Algorithms*, 8(1):19–38, 1987.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[18] J. Cox and C. K. Yap. On-line motion planning: Moving a planar arm by probing an unknown environment. Technical report, Courant Institute of Mathematical Sciences, New York University, New York, July 1988.

[19] W. Coy. *Fundamentals of Computation Thoery*, chapter Automata in labyrinths, pages 65–71. 1977.

[20] X. Deng, T. Kameda, and C. H. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd Ann. Symp. on Foundations of Computer Scienc e*, pages 298–303, 1991.

[21] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. In *Proc. 31st Ann. Symp. on Foundations of Computer Science*, pages 355–361, October 1990.

[22] K. Dopp. Automaten in labyrinthen. *EIK*, 7(3):167–190, 1971.

[23] P. Eades, X. Lin, and N. C. Wormald. Performance guarantees for motion planning with temporal uncertainity. In *Proc. First Canadian Conference on Computational Geometry*, 1989.

[24] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3:249–265, 1987.

[25] A. Fiat, D. P. Foster amd H. Karloff, Y. Rabani, and Y. Ravid. Competitive algorithms for layered graph traversals. In *Proc. 32st Ann. Symp. on Foundations of Computer Science*, pages 288–297, 1991.

[26] G. Foux, M. Heymann, and A. Bruckstein. Two-dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Transactions on Robotics and Automation*, 9(1):96–102, 1993.

[27] A. S. Fraenkel. Ecomic traversal of labyrinths. *Mathematics Magazine*, 43:125–130, 1970.

[28] A. S. Fraenkel. Ecomic traversal of labyrinths. *Mathematics Magazine*, 44:12, 1971.

[29] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In *Proc. 3rd ACM Symp. in Computational Geometry*, pages 50–63, 1987.

[30] F. Harary. *Graph Theory.* Addison-Wesley Pub. Co., Reading, MA, 1969.

[31] J. Hopcroft, D. Joseph, and S. Whitesides. *Planning, Geometry and Complexity of Robot Motion*, chapter Movement problems for 2-dimensional linkage, pages 282–303. iAblex Publishing Co., Norwood, New Jersey, 1987.

[32] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Pub. Co., Reading, MA, 1979.

[33] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[34] S.S. Iyengar, C.C. Jorgensen, N. S. V. Rao, and C.R. Weisbin. Robot navigation using learned spatial graphs. *Robotica*, 4:93–100, 1986.

[35] B. Kalyanasundaram and K. Pruhs. A competitive analysis of nearest neighbor based algorithms for searching unknown scenes. In *Proceedings of Ninth Annual Symposium on Theoretical Aspects of Computer Science*, pages 147–157, 1992.

[36] B. Kalyanasundaram and K. Pruhs. Constructing competitive tours from local information. Technical report, Dept. of Computer Science, Univ. of Pittsburgh, 1992.

[37] B. Kalyanasundaram and K. Pruhs. A competitive analysis of algorithms for searching unknown scenes. Technical report, Dept. of Computer Science, Univ. of Pittsburgh, 1993.

[38] M. Kao, J. Reif, and S. Tate. Searching in an unknown environment: An optimal randomized algorithm for cow-path problem. In *Proceedings of 4th ACM-SIAM Symposium on Discrete algorithms*, 1993.

[39] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[40] R. Klein. Walking an unknown street with bounded detour. In *Proceedings of 32nd Annual Symposium on Foundations of Computer Science*, pages 304–313, 1991.

[41] J. M. Kleinberg. Algorithms for on-line navigation. Technical Report TR 92-1316, Department of Computer Science, Cornell University, Ithaca, New York, 1992.

[42] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Pub., Boston, 1991.

[43] V. Lumelsky. Continuous motion planning in unknown environment for a 3d cartesian robot arm. In *Proceedings IEEE International Conf. on Robotics and Automation*, April 1986.

[44] V. Lumelsky. Continuous robot motion planning in unknown environment. In *Adaptive and Learning Systems: Theory and Applications*, 1986.

[45] V. Lumelsky. Algorithmic and complexity issues of robot motion in uncertain environment. *Journal of Complexity*, 3:146–182, 1987.

[46] V. Lumelsky. Algorithmic issues of sensor-based robot motion planning. In *26th IEEE Conference on Decision and Control*, pages 1796–1801, 1987.

[47] V. Lumelsky. Dynamic path planning for a planar articulated robot arm moving amidst unknown obstacles. *Automica, J. IFAC(International Federation of Automatic Control*, 1987.

[48] V. Lumelsky. Gross motion planning for a simple 3d articulated robot arms moving amdist unknown arbitrarily shaped obstacles. In *Proceedings IEEE International Conf. on Robotics and Automation*, April 1987.

[49] V. Lumelsky. A comparative study on the path length performance of maze-searching and robot mation planning algorithms. *IEEE Transactions on Robotics and Automation*, 7(1):57–66, 1991.

[50] V. Lumelsky, S. Mukhopadhyay, and K. Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*, 6(4):462–472, 1990.

[51] V. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man and Cybernetics*, 20(5):1058–1069, 1990.

[52] V. Lumelsky and K. Sun. A unified methodology for motion planning with uncertainty for 2d and 3d two-link robot arm manipulators. *International Journal of Robotics Research*, 9(5):89–104, 1990.

[53] V. J. Lumelsky and A. A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE transactions on Automatic control*, pages 1058–1063, 1986.

[54] H. Muller. Endliche automaten und labyrinthen. *EIK*, 7(4):261–264, 1971.

[55] J. Mylopoulos. On the recognition of topological invariants by 4-way finite automata. *Computer Graphics and Image Processing*, 1:308–316, 1972.

[56] B. J. Oomen, S. S. Iyengar, N. S. V. Rao, and R. L. Kashyap. Robot navigation in unknown terrains using learned visibility graphs. part i: The disjoint convex obstacle case. *IEEE Journal of Robotics and Automation*, 3:672–681, 1987.

[57] O. Ore. *Theory of Graphs.* American Mathematical Society, Providence, RI, 1962.

[58] C. H. Papadimitriou and M. Yanakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[59] R. Paul. *Robot Manipulators: Mathematics, Programming and Conrol.* MIT Press, Cambridge, MA, 1981.

[60] J. Pearl. *Heuristics: Intelligent Search Strategies For Computer Problem Solving.* Addison Wesley Pub., Reading, Mass., 1984.

[61] N. S. V. Rao. Algorithmic framework for learned robot navigation in unknown terrains. *IEEE Computer*, 22:37–43, 1989.

[62] N. S. V. Rao. Robot navigation in unknown generalized polygonal terrains using a discrete scan sensor. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 871–876, October 1991.

[63] N. S. V. Rao. Performance trade-offs in robot navigation algorithms in unknown terrains. *ORSA Journal on Computing*, 4(2):218–224, 1992.

[64] N. S. V. Rao. Robot navigation in unknown generalized polygonal terrains using a vision sensors, 1993. manuscript.

[65] N. S. V. Rao and S. S. Iyengar. Autonomous robot navigation in unknown terrains: Visibility graph based methods. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1443–1449, 1990.

[66] N. S. V. Rao, S. S. Iyengar, B. J. Oomen, and R. L. Kashyap. On terrain model acquisition by a point robot and polyhedral obstacles. *IEEE Journal of Robotics and Automation*, 4:450–455, 1988.

[67] N. S. V. Rao, N. Stoltzfus, and S. S. Iyengar. A 'retraction' method for learned navigation in unknown terrains. *IEEE Transactions on Robotics and Automation*, 7(5):699–707, 1991.

[68] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[69] A. Rosenfeld. Connectivity in digital pictures. *Journal of Association of Computing Machinery*, 17:146–160, 1970.

[70] A. Sankaranarayanan and Isao Masuda. A new algorithm for robot curve-following amidst unknown obstacles, and a generalization of maze-searching. In *Proceedings of 1992 IEEE International Conference on Robotics and Automation*, pages 2487–2494, 1992.

[71] A. Sankaranarayanan and M. Vidyasagar. A new path planning algorithm for a point object amidst unknown obstacles in a plane. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1930–1936, 1990.

[72] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm developments. In *Proceedings of IEEE Conference on Decision and Control*, pages 1111–1119, December 1990.

[73] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: The universal lower bound on the worst case path lengths, and a classification of algorithms. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1734–1741, December 1991.

[74] W. J. Savitch. Maze recognition automata and nondeterministic tape complexity. *Journal of Computer and System Sciences*, 7:389–403, 1973.

[75] A. P. Shah. Pebble automata on arrays. *Computer Graphics and Image Processing*, 3:236–246, 1974.

[76] M. Sharir. Algorithmic motion planning in robotics. *IEEE Computer*, pages 9–20, March 1989.

[77] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28, February 1985.

[78] K. Sun and V. Lumelsky. Path planning among unknown obstacles: the case of a three-dimensional catesian arm. *IEEE Transactions on Robotics and Automation*, 8(6):776–786, 1992.

[79] I. Sutherland. A method for solving arbitrary wall mazes by computer. *IEEE Trans. on Computers*, C-18(12):1092–1097, 1969.

[80] M. Turchen and A. Wong. Low level learning for a mobile robot: Environmental model acquisition. In *Proceedings of 2nd International Conference on AI Applications*, 1985.

INTERNAL DISTRIBUTION

1. B. R. Appleton
2. M. Beckerman
3. C. W. Glover
4. J. P. Jones
5. R. C. Mann
6. E. M. Oblow
7. F. G. Pin
8-11. N. S. V. Rao
12. D. B. Reister
13. M. A. Unseren
14. R. C. Ward
15-16. Laboratory Records
17. Laboratory Records - RC
18. ORNL Patent Office
19. Central Research Library
20. EPMD Reports Office

EXTERNAL DISTRIBUTION

21. Professor Narendra Ahuja, Beackman Institute and Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801.

22. Professor Peter Allen, Department of Computer Science, 450 Computer Science, Columbia University, New York, NY 10027

23. Professor Ricardo A. Baeza-Yates, Department of Computer Science, Universidad de Chile, Casilla 2777, Santiago, Chile

24. Professor E. Bar-Eli, Department of Computer Science, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel

25. Professor Wayne Book, Department of Mechanical Engineering, J. S. coon Bldg, Rm 306, Georgia Institute of Technology, Atlanta, GA 30332

26.. Professor J. Michael Brady, Department of Engineering Science, Oxford University, 19 Parks Road, Oxford OX1 3PJ, England

27. Professor Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street, Cambridge, MA 02138

28. Professor Avrim Blum, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

29. Professor John F. Canny, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94729

30. Professor Joseph C. Culberson, Department of Computing Science, University of Alberta, Edmonton, Alberta, T6G 2H1, Canada

31. Professor Xiaotie Deng, Department of Computer Science, York University, North York, Ont M 3J 1P3, Canada

32. Professor Steven Dubowsky, Massachusetts Institute of Technology, Building 3, Room 469A, 77 Massachusetts Ave., Cambridge, MA 02139

33. Professor Donald J. Dudziak, Department of Engineering, North Carolina State University, 110B Burlington Engineering Labs, Raleigh, NC 27695-7909

34. Professor Amos Fiat, Computer Science Department, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel

35. Dr. Yong K. Hwang, Sandia National Laboratory, Albuquerque, New Mexico 87185

36-39. Professor S. Sitharama Iyengar, Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803

40. Professor Avi Kak, Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907

41. Professor Tiko Kameda, School of Computing Science, Simon Fraser University Burnaby, BC V5A 1S6

42-45. Srikumar Kareti, Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162

46. Professor Howard Karloff, Department of Computer Science, University of Chicago, Chicago, IL 60637

47. Professor R. L. Kashyap, Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907

48. Professor Rolf Klein, Praktische Informatik VI, Fern Universitat-GH-Hagan, Feithstrabe 152, D-5800 Hagen, Germany

49. Professor J. M. Kleinberg, Department of Computer Science, Cornell University, Ithaca, NY 148-7501

50. Professor Jean Claude Latombe, Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305

51. Dr. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815

52. Professor Tomas Lozano-Perez, Artificial Intelligence Laboratory, MIT, 545 Technology Square, Cambridge, MA 02139

53. Professor Vladimir Lumelsky, Department of Mechanical Engineering, University of Wisconsin, Madison, WI 53706

54. Professor K. J. Maly, Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162

55. Dr. Oscar P. Manley, Division of Engineering, Mathematical and Geosciences, Office of Basic Energy Sciences, ER-15, U.S. Department of Energy - Germantown, Washington, DC 20545

56. Dr. Alan Meyrowitz, Artificial Intelligence Technologies, Naval Research Laboratory, 4555 Overlook Ave. SW, Washington DC 20375-5000

57. Dr. Howard Moraff, Director, Robotics and Machine Intelligence Program, National Science Foundation, 1800 G. Street, N. W., Washington, D.C. 20550

58. Professor Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801

59. Professor Christos Papadimitriou, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093.

60. Professor Gregory J. E. Rawlins, Department of Computer Science, Indiana University, 101 Lindley Hall, Bloomington, IN 47405

61. Dr. Prabhakar Raghavan, IBM Research Division, TJ Watson Research Center, Yorktown Heights, NY 10598

62. Professor John Reif, Department of Computer Science, Duke University, Durham, NC 27706

63. Dr. A. Sankaranarayanan, Secom IS Lab, 6-11-23 Shimorenjaku, Mitaka, Tokyo 181, Japan

64. Professor Micha Sharir, New York University, Robotics Laboratory, 719 Broadway, New York, NY 10003

65-68. Weimin Shi, Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162

69. Professor Wes Snyder, Department of Radiology, Bowman Gray School of Medicine, 300 S. Hawthrone Dr., Winston-Salem, NC 27103

70. Dr. M. Vidyasagar, Center for Artificial Intelligence and Robotics, Raj Bhavan Circle, High Grounds, Bangalore 560001, India

71. Dr. B. W. Wah, National Science Foundation, 1800 G Street, NW, Washington, DC 20550

72. Dr. C. W. Weisbin, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109-8099

73. Professor Mary F. Wheeler, Department of Mathematics, Rice University, P. O. Box 1892, Houston, TX 77251

74. Professor Chee-Keng Yap, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

75. Office of the Assistant Manager for Energy Research and Development, Oak Ridge Operations, Department of Energy, P. O. Box 2008, Oak Ridge, TN 37831.

76-77. Office of Scientific and technical Information, Department of Energy, P. O. Box 62, Oak Ridge, TN 37831.

END

DATE
FILMED
10/6/93