ELSEVIER

Parallel Computing 21 (1995) 1749–1767

PARALLEL
COMPUTING

Parallel path planning on the distributed array processor [†]

Chang Shu [a,*], Hilary Buxton [b]

[a] *Department of Mechanical and Aerospace Engineering, Carleton University, Ottawa K1S 5B6, Canada*

[b] *School of Cognitive and Computing Sciences, University of Sussex, Brighton BN1 9QH, UK*

Received 15 September 1992; revised 22 June 1995

Abstract

An approach to the path planning problem is presented in which a distributed representation of the workspace is used, and related to it, parallel processing techniques are presented to search the world model. The representation, which is based on configuration space and can be computed in parallel, naturally captures the real world geometry. The search is based on a cellular automaton and diffuses strengths from the goal to find the shortest path. The approach can be extended to deal with rotational as well as translational motions. Paths can be found at different resolutions of the workspace. It also has the useful property that the complexity of the search is almost constant with varying degrees of environmental clutter.

Keywords: Path planning; Distributed representation; Robotics; SIMD machines

1. Introduction

Effective and efficient generation of motion which causes objects in the world to move in some desired way is central to the field of robotics. For example, we may want to generate a motion which will cause an autonomous vehicle to move to a specific position or a motion which will move a silicon wafer from one point to another or even a motion to move one moving object around another. Another major concern is that these motions are collision-free. The basic problem can be stated as: Given the initial and desired final configurations of a rigid object in two-

[†] The experimental work reported here was carried out while the authors were at Dept. of Computer Science, Queen Mary and Westfield College, University of London.

[*] Corresponding author, E-mail:cshu@mrco2.carleton.ca

or three-dimensional space, and given the description of the obstacles in the space, and given the description of the obstacles in the space, determine whether there is a continuous motion of the object from the initial configuration to the final configuration, and find such a motion if one exists. This problem is also called *Movers' Problem* or *Findpath* problem.

Theoretically, given algebraic descriptions of all the motion constraints, the generalized movers' problem is solvable. Schwartz and Sharir [10] provided the first general solution to the problem. They gave an algorithm which runs in a time polynomial in the number of algebraic constraints, but doubly exponential in the number of degrees of freedom. This result was later improved by Canny [2] who provided an algorithm solving the problem in time $O(n^k \log n)$, where n is the number of geometric constraints and k is the degrees of freedom of the system. This result reduces the time complexity of the algorithm from double exponential in k to single exponential. However, the running time of the algorithm also includes a large constant of proportionality, which depends on the degree and coefficient magnitude of the constraint polynomials. It is shown that general motion planning is intrinsically hard [9]. Certain variants of the general motion problem are proved to be intractable. In particular, the three-dimensional Euclidean shortest path problem is NP-hard [2].

Since Schwartz and Sharir, many algorithms have been proposed for special cases, e.g. [4,8,11]. These algorithms assume polygonal or polyhedral models of the environment, and work for special shaped robots such as line segments and discs. A common feature of these methods is that they are based on exact models and exploit mathematical structures of the motion constraints. Therefore, they usually provide algorithms which provably converge and which seek to improve worst-case asymptotic efficiency. These algorithms are generally difficult to implement. Given the high complexity of the motion planning problem, many researchers have been working on approximate methods, e.g. [1,6,19]. These methods usually discretise the motion constraints. They use various heuristics and aim at good average-case efficiency. Our approach uses an effective and efficient search of a distributed configuration space which can find the shortest path. It also has the advantage that it is robust to environmental complexity for a range of problems.

2. Distributed respresentation

An important technique, used in many motion planning algorithms, is obstacle transformation. The idea is to transform the obstacles in such a way that the robot can be seen as a point in a higher dimensioned space, called *Configuration Space*. This reduces the problem of planning the motion of a dimensioned object to the problem of planning motion of a point.

Formally, let $\mathcal{A}$ be a rigid object moving in a Euclidean Space $\mathcal{W}$, called *Workspace*. Let $B_1, B_2, \ldots, B_n$ be fixed rigid objects in $\mathcal{W}$. We attach a coordinate frame $\mathcal{F}_{\mathcal{W}}$ to $\mathcal{W}$ and a coordinate frame $\mathcal{F}_{\mathcal{A}}$ to $\mathcal{A}$. The position of $\mathcal{A}$ can be specified by the origin of $\mathcal{F}_{\mathcal{A}}$ in $\mathcal{F}_{w}$ and the orientation of $\mathcal{A}$ can be specified by the orientation of $\mathcal{A}_{\mathcal{A}}$ relative to $\mathcal{F}_{\mathcal{W}}$.

A configuration of an object is a two-element tuple ($\mathscr{P}$, Θ), where $\mathscr{P}$ and Θ are the position and orientation of $\mathscr{F}_{\mathscr{A}}$ with respect to $\mathscr{F}_{\mathscr{W}}$. The set of all configurations of $\mathscr{A}$ is called the configuration space of $\mathscr{A}$, denoted $\mathscr{C}space_{\mathscr{A}}$, or $\mathscr{C}$ for short. As the motions of $\mathscr{A}$ are constrained by the obstacles, not all possible configurations in $\mathscr{C}space_{\mathscr{A}}$ are valid. Those invalid configurations form a region in the $\mathscr{C}$space$_{\mathscr{A}}$ which is important to the path of $\mathscr{A}$. This region is called the 'C-obstacle region', denoted as $\mathscr{C}\mathscr{B}$. The complement of $\mathscr{C}\mathscr{B}$ is called 'free space' and is denoted as $\mathscr{F}\mathscr{P}$.

Much research has focused on obtaining a discrete representation of $\mathscr{F}\mathscr{P}$. Most methods use a *connectivity graph* to characterise $\mathscr{F}\mathscr{P}$ and eventually reduce the problem to a graph search. These algorithms assume a piecewise linear model of the environment. While we can approximate any object to any degree of accuracy using piecewise linear objects, and efficient piecewise linear algorithms are found in some cases, there are several disadvantages of this assumption. First, it often takes too many straight line segments to model a simple curve realistically. Second, it takes a considerable effort to construct a good piecewise linear approximation to a real object. Third, algorithms based on this assumption are not robust, for a small error in the model may cause devastating consequences for the planning algorithm.

In this paper, we propose to use a distributed array model to represent the world. The construction of this model is simple. We first discretise the space into regular cells. Then we register those elements in the binary array corresponding to the cells occupied by objects in the space with value 1, and register other elements with value 0. Then the environment under consideration is represented as a collection of bits. In the case of two dimensions, it is equivalent to the *bitmap* representation in computer graphics. In computer vision, this representation is called *spatial occupancy enumeration.* Like the piecewise linear model, a binary array can approximate any object to any degree of accuracy. However, there are other advantages to the use of this model. First, it is robust. Geometric features are represented implicitly in the binary array. A global feature is distributed in many local parts of the representation. This makes the subsequent reasoning algorithms tolerant of incomplete and partially incorrect information in the original representation. Second, it is easily obtainable from sensory data. This facilitates a motion planner encapsulated in an autonomous system consisting of vision or other sensing devices. Such a system could work in an unknown or partially known environment.

To formalise the representation, consider an object $\mathscr{A}$ moving in a two-dimensional workspace $\mathscr{W}=\mathscr{R}^2$. The configuration space $\mathscr{C}$ is $\mathscr{R}^2$ if the robot moves in a fixed orientation, or $\mathscr{R}^2 \times [0,2\pi]$ if the robot rotates as well as translates in the plane. We divide the workspace $\mathscr{W}$ into regular square grids. Each grid represents either a free space area or an obstacle area. We define a map f:$\mathscr{W} \rightarrow \{0,1\}$:

$$\forall x \in \mathscr{W}, f(x) = \begin{cases} 1 & \text{if } x \text{ is in an obstacle area} \\ 0 & \text{if } x \text{ is in a free area} \end{cases}$$

Therefore the workspace is modelled as a binary array, denoted as $\mathcal{BW}$. $\mathcal{BW}$ is a conservative approximation of the workspace $\mathcal{W}$. Each point in $\mathcal{BW}$ represents an area in $\mathcal{W}$. $\mathcal{BW}$ can be made arbitrary close to $\mathcal{W}$ by increasing the resolution. The subsequent planning algorithms work on $\mathcal{BW}$.

Obviously, the binary array representation is data intensive. Therefore, the manipulation of this model can be expensive when the resolution is high. However, by using massively parallel computers, we can deal with many cells in parallel. The theme of this paper is to identify parallelism in the processing of this model, and develop parallel techniques for manipulating the model so as to develop robust and efficient solutions to the motion planning problem.

Lozano-Pérez [6] showed that the configuration space obstacles due to an object moving with a fixed orientation can be obtained by set difference operations on the point sets of the obstacles and the moving object. This result was then used to derive an algorithm for computing C-obstacles for polygonal obstacles and moving objects, where the set operations are only required to perform on the vertices of the polygons. For general environments, the result serves only as a definition of the C-obstacles rather than a procedure for computing them, for there are infinite number of set operations to be performed. We show, in this paper, the set difference definition of configuration space obstacles can be further defined by a geometric transformation, *translation*, which is easy to implement on our proposed parallel architecture, the Distributed Array Processor (DAP). We describe a parallel algorithm which performs translation operations on all obstacles in the workspace simultaneously. Given a discretised map representing the workspace, the algorithm produces a new discretised man representing the configuration space.

3. Computational model

We consider a SIMD computer which consists of a $N \times N$ array of processors (usually $N = 2^m$). each processor element (PE) in the array is connected with its four neighbours, and each may operate on the data in its own local memory. A central control unit broadcasts an instruction to all PE's, and all enabled PE's simultaneously execute this instruction. This computer supports the parallel manipulations of $N \times N$ matrix data. For any matrices A, B and C of the same data type, there are operations of the form:

$$C = A * B$$

performed in one machine cycle, where ' $*$ ' can be normal arithmetic operations or logical operations depending on the data type of the matrices.

Suppose a_{ij} and $b_{ij} (i = 1, \ldots, N,\ j = 1, \ldots, N)$ are elements of Boolean matrices A and B respectively, and c_{ij} denotes elements of Boolean matrix C. As will be seen, we are particularly interested in the operations:

(1) $C = A \vee B$, where $c_{ij} \vee b_{ij}$;
(2) $C = A \wedge B$, where $c_{ij} = a_{ij} \wedge b_{ij}$;
(3) $C = \neg A$, where $c_{ij} = \neg a_{ij}$.

In addition, the machine provides all PE's the function of simultaneous access to their neighbours. This function is expressed as four shift operations, `shiftnorth`, `shiftsouth`, `shifteast`, and `shiftwest`. For matrices A and B, if $B =$ `shiftnorth`(A), then B is the marix obtained by shifting A's elements by one row to the north. The other three shift functions perform similar operations except that the directions in which they shift the matrix are different.

As our distributed representation involves amount of data arranged in two- or three-dimensional arrays, it naturally appeals to the distributed processor array model. We map elements of the data array to the PEs and apply homogeneous matrix operations to the whole area of the data. In this way, we can first construct the configuration space and then search in this space efficiently.

4. Computing C-obstacles

To compute the configuration space $\mathscr{C}$ involves mapping the discretised representation of workspace $\mathscr{BW}$ to a space, such that the moving object is transformed to a point with obstacles grown at the same time. Our essential data structure is a binary array representing both the workspace and the configuration space. When $\mathscr{A}$ is a disc or when $\mathscr{A}$ is a dimensioned object allowed to translate (i.e. the origin of $\mathscr{F}_{\mathscr{A}}$ can follow any path in $\mathscr{BW}$) without rotation (i.e. $\mathscr{F}_{\mathscr{A}}$ has a fixed orientation with respect to $\mathscr{F}_{\mathscr{W}}$), the configuration space $\mathscr{C}$ can be represented as a two-dimensional binary array. When $\mathscr{A}$ is allowed to rotate as well as translate, $\mathscr{C}$ can be represented as a three-dimensional binary array. In this section we examine the case where $\mathscr{A}$ can only translate in the plane. We construct the configuration space $\mathscr{C}$ in the form of a two-dimensional binary array and show that $\mathscr{C}$ can be generated in parallel. The case that includes rotational motions is discussed in Section 6.

4.1 The algorithm

For a general asymmetric moving object $\mathscr{A}$, we propose a method which subdivides $\mathscr{A}$ into a collection of vertical line segments and incrementally build the C-obstacles. Observe that the C-obstacles due to a vertical line segments moving with a fixed orientation in a plane are easy to compute. We shall show that the C-obstacles due to $\mathscr{A}$ can be obtained by the union of the transformed C-obstacles due to its component parts.

Objects in the plane can be viewed as sets of points and computing of C-obstacles can be viewed as transformations on these sets. The following definition defines a few operations on sets of points.

Definition. Let S_1 and S_2 be sets of points in the plane. The *set sum, set difference,* and *set negation* are defined as:

$$S_1 + S_2 = \{s_1 + s_2 | s_1 \in S_1, s_2 \in S_2\}$$
$$S_1 - S_2 = \{s_1 - s_2 | s_1 \in S_1, s_2 \in S_2\}$$
$$-S_1 = \{-s_1 | s_1 \in S_1\}$$

Lemma 1. *For any sets $\mathscr{A}$ and $\mathscr{B}$, $\mathscr{C}\mathscr{B}_{\mathscr{A}} = \mathscr{B} - \mathscr{A}(0)$.*

Proof. Cf. Theorem 1 of [6]. □

Consider an arbitrary line segment $\mathscr{L}$ as a moving object, there is a immediate corollary of this lemma.

Corollary 1. *For any set $\mathscr{B}$ and a line segment $\mathscr{L}$, $\mathscr{C}\mathscr{B}_{\mathscr{L}} = \mathscr{B} - \mathscr{L}(0)$.*

In [6], by using the result of Lemma 1, an efficient algorithm for computing CB's for polygon objects was provided. The algorithm exploits properties of convex polygons so that CB's can be computed by performing set operations only on the vertices of the polygons. Since we consider a more general problem which allows the obstacles to be arbitrary shapes, there are no special properties to use in order to reduce the number of set operations. Instead, we introduce another geometric operation, *translation*, to define the C-obstacles due to a line segment $\mathscr{L}$.

Definition. Let p be an arbitrary point and X be a set of points in the plane. The transformation T_p such that $T_p(X) = p + X$ is called the *translation* of set X in the direction $\overline{OP}$, where O is the coordinate origin.

Definition. The swept volume of a set X in the plane along a line segment l is the set of points $S_l(X)$ such that

$$S_l(X) = \bigcup_{a \in l} T_a(X).$$

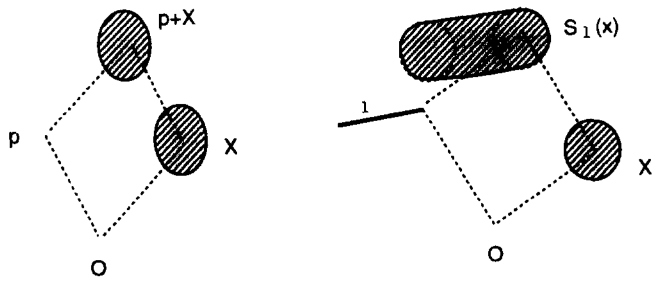

Fig. 1. Illustrations of the notions of translation and swept volume.

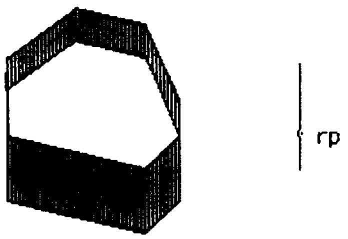

Fig. 2. A C-obstacle due to a vertical line.

In other words, the swept volume of X along l is the set of all the translations of X along the line l. Fig. 1 illustrates the notion of translation and swept volume.

Theorem 1. *For any set $\mathscr{B}$ and line segment $\mathscr{L}$ in the plane, $\mathscr{C}\mathscr{B}_{\mathscr{L}} = S_{-\mathscr{L}}(\mathscr{B})$.*

Proof. For any $s \in S_{-\mathscr{L}}(\mathscr{B})$, according to the definition of swept volume, there exists a point $a \in \mathscr{L}$ such that $s \in T_{-a}(\mathscr{B})$. Then there exists $b \in \mathscr{B}$ such that $s = b - a$. It follows $s \in \mathscr{B} - \mathscr{L}$. By Lemma 1, $s \in \mathscr{C}\mathscr{B}_{\mathscr{L}}$.

On the other hand, if $s \in \mathscr{C}\mathscr{B}_{\mathscr{L}}$, there exists $b \in \mathscr{B}$ and $a \in \mathscr{L}$ such that $s = b - a$, So $s = b + (-a)$. Then $s \in T_{-a}(\mathscr{B})$. Therefore $s \in S_{-\mathscr{L}}(\mathscr{B})$. □

Theorem 1 implies that the C-obstacle of a planar object $\mathscr{B}$ due to a line segment $\mathscr{L}$ is the swept volume of $\mathscr{B}$ along $\mathscr{L}$. In our case, we are interested in a vertical line segment $\mathscr{L}(A,B)$ with a reference point rp on it. Thus $L(A,B) = L(A,rp) \cup L(rp,B)$,i.e. $L(A,B)$ consists of two vertical line segments with reference points on ends. For an object $\mathscr{B}$, the C-obstacle due to $L(A,rp)$ is the swept volume of $\mathscr{B}$ along $L(A, rp)$ and the C-obstacle due to $L(rp,B)$ is the swept volume of $\mathscr{B}$ along $L(rp, \mathscr{B})$. The overall C-obstacle due to $L(A,B)$ is the union of these two parts. Fig. 2 illustrates this situation. The swept volume in the vertical direction can be computed by applying shift-union operations to the map plane. Algorithm CBL shown in Fig. 3 computes the $\mathscr{C}\mathscr{B}_{\mathscr{L}}$ for a vertical line segment $\mathscr{L}$.

Suppose an object $\mathscr{A}$ is divided into a number of parts. The following theorem

```
Algorithm CBL(BW, $L(A,B)$)
  begin
    $l_1 \leftarrow |B - rp|$;
    $l_2 \leftarrow |rp - A|$;
    CB := BW;
    for i:=1 to $l_1$ do
      CB := CB ∨ shiftnorth(CB);
    end;
    for j:= 1 to $l_2$ do
      CB := CB ∨ shiftsouth(CB);
    end;
  end
```

Fig. 3. Algorithm for computing $\mathscr{C}\mathscr{B}_{\mathscr{L}_i}$.

establishes the realtionship between the C-obstacles due to $\mathscr{A}$ and the C-obstacles due to its component parts.

Theorem 2. *Let $\mathscr{A}$ be a two-dimensional object moving with a fixed orientation in the plane. Suppose $\mathscr{A}$ can be decomposed into $n(n > 1)$ parts, $\mathscr{A}_1^A, \ldots, \mathscr{A}_n^A$, then $\mathscr{C}\mathscr{B}_{\mathscr{A}} = \bigcup_{i=1}^{n}(\mathscr{C}\mathscr{B}_{\mathscr{A}i} - rp_i)$ where $\mathscr{A}_i$ is $\mathscr{A}_i^A$ as an independent moving object with its reference point at rp_i.*

Proof. The region of the plane occupied by $\mathscr{A}_i^A$ at the its initial configuration of $\mathscr{A}$ is denoted by $\mathscr{A}_i^A(0)$, and the region of the plane occupied by $\mathscr{A}_i$ at its initial configuration is denoted by $\mathscr{A}_i(0)$. Obviously, $\mathscr{A}_i^A(0) = rp_i + \mathscr{A}_i(0)$ See Fig. 4. Similarly, for any $x \in R^2$, we have $\mathscr{A}_i^A(x) = rp_i + \mathscr{A}_i(x)$.

For any $x \in \mathscr{C}\mathscr{B}_{\mathscr{A}}$, there exists $b \in \mathscr{B} \cap \mathscr{A}(x)$. Thus $b \in \mathscr{B}$ and $b \in \mathscr{A}(x)$. That b belongs to $\mathscr{A}(x)$ implies b must be in a certain component of $\mathscr{A}$ at configuration x, i.e. there exist i, such that $b \in \mathscr{A}_i^A(x)$. From 4.1, there exists $a \in \mathscr{A}_i(x)$, such that $b = rp_i + a$. Then there exists $a' \in \mathscr{A}_i(0)$ such that $a = x + a'$. This is equivalent to $b = rp_i + x + a'$ which follows $x = b - a' - rp_i$. So $x \in \mathscr{B} - \mathscr{A}_i(0) - rp_i$. By Lemma 1, $x \in \mathscr{C}\mathscr{B}_{\mathscr{A}i} - rp_i$. Therefore $\mathscr{C}\mathscr{B}_{\mathscr{A}} \subseteq \bigcup_{i=1}^{n}(\mathscr{C}\mathscr{B}_{\mathscr{A}i} - rp_i)$.

On the other hand, if $x \in \mathscr{C}\mathscr{B}_{\mathscr{A}_i} - rp_i$ for some i, then $x = x' - rp_i$ for some $x' \in \mathscr{C}\mathscr{B}_{\mathscr{A}i}$. This implies that here exists b such that $b \in \mathscr{B}$ and $b \in \mathscr{A}_i(x')$. Therefore, there exists $a \in \mathscr{A}_i(0)$ such that $b = x' + a$. Equivalently $x' = b - a$. Then $x = b - a - rp_i = b - (a + rp_i)$. Let $a' = a + rp_i$. Since $a \in \mathscr{A}_i(0)$, we have $a' \in \mathscr{A}_i^A(0)$. Thus $a' \in \mathscr{A}(0)$. It follows $x \in \mathscr{B} - \mathscr{A}(0)$. So $x \in \mathscr{C}\mathscr{B}_{\mathscr{A}}$. Therefore, $\bigcup_{i=1}^{n}(\mathscr{C}\mathscr{B}_{\mathscr{A}_i} - rp_i) \subseteq \mathscr{C}\mathscr{B}_{\mathscr{A}}$, and our proof is complete. □

Note that $\mathscr{C}\mathscr{B}_{\mathscr{A}i} - rp_i$ is actually a translation of $\mathscr{C}\mathscr{B}_{\mathscr{A}i}$ by $-rp_i$, i.e. $T_{-rp_i}(\mathscr{C}\mathscr{B}_{\mathscr{A}i})$. This theorem states a method for computing C-obstacles by decomposing the moving object and computing, translating, and union the C-obstacles due to its component parts. The simplest decomposition of a planar shape is the rectangle decomposition. In our binary array representation, objects are approximated by a set of regular squares. Those squares which have the same x-coordinates form a

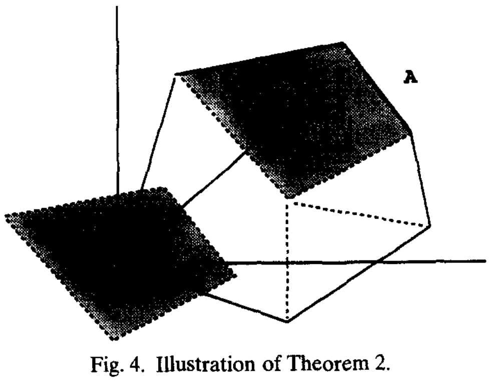

Fig. 4. Illustration of Theorem 2.

rectangle with one unit width and the object can be seen as a set of these rectangles with different heights. When a sufficiently high resolution is used, the widths of these rectangles can be neglected and the object can be seen as a set of vertical line segments. Algorithm CBL is used for computing the C-obstacles due to each of the verical line segments. These C-obstacles are translated according to the relative positions of the individual vertical lines' reference point and that of the moving object's reference point. A translation in an arbitrary direction can be broken down into a translation in x-direction followed by a translation in y-direction. Suppose $(rp(x), rp(y))$ is the coordinate of $\mathscr{A}$'s reference point and $(rp_i(x), rp_i(y))$ is the coordinate of $\mathscr{A}_i$'s reference point. Then the actual value of translation of $\mathscr{C}\mathscr{B}_{\mathscr{A}i}$ is $|rp(x) - rp_i(x)|$ in x-direction and $|rp(y) - rp_i(y)|$ in y-direction. Algorithm CB shown in Fig. 5 details the computation of C-obstacles for genetal shaped moving objects.

4.2 Complexity

Assume the resolution of BW is $l \times l$ and we use l^2 processors. Since the main loops in the algorithm consist of shift and union operations, we can use the number of parallel shift and union operations to measure the running time of the algorithm. As $\mathscr{A} = \bigcup_{i=1}^{n} \mathscr{L}_i$, where $\mathscr{L}_i$ is a vertical line segment with length m_i measured in the number of cells contained in it. The main loop in the Algorithm

```
Algorithm CB(BW, A, rp);
    begin
        $x_{min}$ ← the minimal $x$-coordinate of A;
        $x_{max}$ ← the maximal $x$-coordinate of A;
        CB ← BW;
        for i=$x_{min}$ to $x_{max}$ do
            CB1 ← CBL(BW, L(i));
            if rp(x)- i > 0 then
                    for j=1 to rp(x) - i do
                        CB1 ← shifteast(CB1);
                    end
            else
                    for j=1 to i - rp(x) do
                        CB1 ← shiftwest(CB1);
                    end
            m ← rp(y) - $rp_i(y)$;
            if m >0 then
                    for k =1 to m do
                            CB1 ← shiftsouth(CB1);
                    end
            else
                    for k = 1 to |m| do
                            CB1 ← shiftnorth(CB1);
                    end
            CB ← CB ∨ CB1;
        end
    end
```

Fig. 5. Algorithm for computing C-obstacles.

CB compute each C-obstacle due to every vertical line spanning from $x_{\min}$ to $x_{\max}$. For each iteration, there are three groups of operations contribute to the time spent on this iteration: the time for CBL, the time for shifting CB1 to the right position, and the time to union the newly obtained CB's with the existing one. Let $n = x_{max} - x_{min} + 1$. The overall time for algorithm CB is:

$$T = \sum_{i=1}^{n} m_i + \sum_{i=1}^{n} |x_i - rp(x)| + \sum_{i=1}^{n} |y_i - rp(y)| + n$$

where (x_i, y_i) is the coordinate of $\mathscr{S}_i$'s reference point. $\sum_{i=x_{min}}^{x_{max}} m_i$ is the number of cells in A. For the second term, $\sum_{i=1}^{n} |x_i - rp(x)|$, x_i belongs to the series x_{min}, $x_{min} + 1, \ldots, rp(x), \ldots, x_{max}$. Let $n_1 = rp(x) - x_{min} + 1$ and $n_2 = x_{max} - rp(x)$. We have

$$\sum_{i=1}^{n} |x_i - rp(x)| = \sum_{i=1}^{n_1} i + \sum_{i=1}^{n_2} i = \frac{n^2 + n - 2n_1 n_2}{2}$$

The worst case occurs when $rp(x) = x_{max}$. In the case $n_2 = 0$ and the second term becomes $\frac{1}{2}(n^2 + n)$.

For the third term, $\sum_{i=1}^{n} |y_i - rp(y)|$, the worst case is that for every $i (1 \leq i \leq n)$, $|y_i - rp(y)|$ reaches its maximum. This is the case when we choose $rp(y)$ to be the maximal y-coordinate of A, y_{max}, and choose y_i to be the minimal y-coordinate of A, y_{min}, or vice versa. Let $m = y_{max} - y_{min}$. Then $\sum_{i=1}^{n} |y_i - rp(y)| \leq mn$.

Thus, a conservative estimation of the running time of Algorithm CB can be

$$T_w = s + \frac{n^2 + n}{2} + mn + n$$

where s is the number of cells in A. Generally, s, n^2, and mn are in the same order of magnitude, while $s \leq mn$ and $mn \leq n^2$ when $m \leq n$. Let $k = max\{m,n\}$. The time complexity of the algorithm is $O(k^2)$. Dehne et al. [3] presented a parallel algorithm for the same task. Their algorithm runs in $O(\sqrt{n})$ time on a Mesh-of Processors architecture with n processors. This is better than ours, but it requires pipeline operations.

5. Search algorithm

Given the configuration space $\mathscr{C}$ represented as a binary array of two or three dimensions, the next step is to search in the free space to find a sequence of grids which connect the initial and the goal configurations. We present in this section a parallel search method which is based on diffusion strengths in free space. This search method was proposed by Steels [15]. It differs from the graph-search methods, such as [16], because there is no explicit connectivity graph being constructed. It is similar to the potential field approaches, but no attempt is made to define a potential function over the workspace or configuration space. We model the algorithm as a cellur automaton and represent the search algorithm in terms of inter-cell communication rules.

5.1 Cellular automata

Cellular automata are mathematical models of systems in which space and time are discrete. They are generally used as a tool for investigation of self-organisation and non-linear dynamical systems [17,18]. In its simplest form, a cellular automaton consists of a line of sites, with each site initially having a value. It evolves in discrete time steps. At each time step, the value of each site is updated according to a definite rule. The new value on each site is specified in terms of the values of its neighbours. One example of a cellular automaton rule is

$$a_i^{t+1} = (a_{i-1}^t + a_{i+1}^t) \bmod 2$$

where a_i^t is the value of site i at time t.

In this section, we use a two-dimensional cellular automaton which involves rules based on values of the four neighbourhoods of each site. An example of a two-dimensional cellular automaton rule is

$$a_{ij}^{t+1} = \left(a_{i-1j}^t + a_{ij-1}^t + a_{ij+1}^t + a_{i+1j}^t\right) \bmod 2$$

Since the rule is applied to all sites simultaneously at each time step, it makes a good model for SIMD parallel computers.

5.2 Strength space

We define strengths on the grids of configuration space $\mathscr{C}$. These strengths encode information about the grids which can be spread in the free space. They are the strengths to move to the south, north, west, and east (Fig. 6). We shall use $a_{ij}(N)$, $a_{ij}(S)$, $a_{ij}(E)$, $a_{ij}(W)$ to denote the first four bits of a_{ij}, and use $a_{ij}(\text{Free})$ to denote the fifth bit.

That a grid has a strength in the north means, if the robot were at the location of the grid, it may move one step to the north, and similarly for the strengths in the

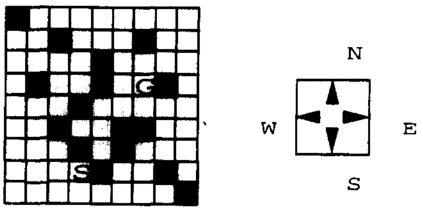

Fig. 6. Strengths on grids.

south, west, and east. We define the fifth strength as invariant, i.e. for all t

$$a_{ij}^{t+1}(Free) = a_{ij}^{t}(Free)$$

5.3 Search by diffusing strengths

The general search strategy is that we first assign strengths to the immediate four neighbourhood grid points of the goal configuration, and then diffuse these strengths in parallel in the four directions according to some rules. Initially, the four grid points neighbouring the goal grid was each assigned a strength which points to the goal grid, as shown in Fig. 7. The diffusion rules are specified in the following formula:

$$a_{ij}^{t+1}(N) = \left(a_{ij}^{t}(N) \vee a_{ij+1}^{t}(N) \vee a_{ij+1}^{t}(E) V a_{ij+1}^{t}(W)\right) \wedge a_{ij}^{t}(Free) \tag{1}$$

$$a_{ij}^{t+1}(S) = \left(a_{ij}^{t}(S) \vee a_{ij-1}^{t}(S) \vee a_{ij-1}^{t}(E) \vee a_{ij-1}^{t}(W)\right) \wedge a_{ij}^{t}(Free) \tag{2}$$

$$a_{ij}^{t+1}(E) = \left(a_{ij}^{t}(E) \vee a_{i+1j}^{t}(E) \vee a_{i+1j}^{t}(S) \vee a_{i+1j}^{t}(N)\right) \wedge a_{ij}^{t}(Free) \tag{3}$$

$$a_{ij}^{t+1}(W) = \left(a_{ij}^{t}(W) \vee a_{i-1j}^{t}(W) \vee a_{i-1j}^{t}(S) \vee \vee a_{i-1j}^{t}(N)\right) \wedge a_{ij}^{t}(Free) \tag{4}$$

Eqs. (1)–(4) specify how to compute the strengths on an arbitrary grid in one step of the iteration. Eq. (1) indicates that the north strength value on the next step depends on its present value as well as the east and west stength value of its north neighbourhood ('$\vee$' and '$\wedge$' represent logical 'or' and 'and' respectively). It is necessary to consider the east and west strength when the north strength is diffused because otherwise the flow of strengths would not be able to avoid the obstacles. Note that grids in a obstacle area cannot acquire any strength in any direction. Thus Eq. (1) implies that for any free grid a_{ij} acquires a north strength on this iteration if

(1) its present north strength value in 1, or
(2) its north neighbourhood grid has a north strength, or
(3) its north neighbourhood grid has an east or west strength.

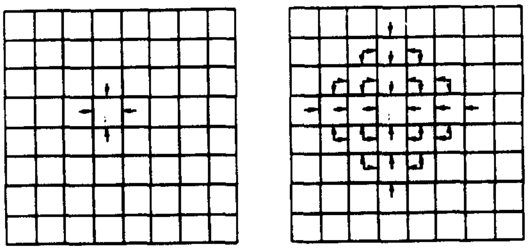
Fig. 7. The initial strength space and the configuration of the strength space after two steps of diffusion.

The implications of Eq. (2) through Eq. (4) should be apparent from the above explanation.

The diffusion process terminates in two situations: (1) the starting grid point acquires a strength; and (2) there is no space to diffuse. In the first case, the robot makes a one-step move according to the strength value on its location grid. If the grid has strengths in more than one direction, any of them can be chosen as the direction for the robot to move. Fig. 7 shows the configuration of the strength space after two steps of diffusion. In the second case, the algorithm concludes that there is no path exists. To make the robot move further steps towards the goal, the strength space is cleared and repeats the same process as described above.

5.4 Analysis

In this subsection, we analyse the length of the path generated by the algorithm introduced in the above subsection and discuss the complexity of the algorithm. We shall prove that the algorithm gives a shortest path in terms of the number of grids comprising of the path.

Definition. The *length* of a path in a grid plane is the number of grids which comprise of the path.

Definition. The *distance* between two positions in the grid plane is the length of the shortest path linking these two positions. We use $dist(p)$ to denote the distance of a point p to the goal position.

Lemma 2. *For any point p in the grid plane, if dist(p) = n, then in the n-th iteration of the algorithm Diffusion, p acquires its first strength.*

Proof. We proof by induction on n. When $n = 1$, p is a neighbour of the goal and is initially assigned a strength pointing to the goal. Thus, it acquires a strength in the first iteration. Suppose the lemma is true for $n = k$, where $k > 1$. For a point q which has a distance $k + 1$, it must have a neighbour that has a distance k. By induction hypothesis, this neighbour acquires its first strength in the kth iteration. According to the diffusion rules (Eqs. (1)–(4)), q acquires its first strength in the $(k+1)$th iteration. □

Theorem 1. *The algorithm Diffusion returns a move which makes the starting point s move one step on its shortest path toward the goal point g.*

Proof. If $dist(s) = 1$, the theorem is obviously true. Now we consider the case $dist(s) = n(n > 1)$. Suppose $ss_{n-1}\ s_{n-2}...s_1 g$ is the shortest path from s to g. Then s_{n-1} is a neighbour of s and $dist(s_{n-1}) = n - 1$. Without loss generality, we assume that s_{n-1} is the eastern neighbour of s. By lemma 2, in the $(n-1)$-th iteration, s_{n-1} acquires some strengths and s acquires no strength (otherwise *dist(s)* would be $n - 1$). It follows that $s_{n-1}\ (W) \neq true$, because by Eq. (4), in the $(n-1)$th iteration, $s_{n-1}(W)$ becomes true only if $s(N)$, $s(S)$, or $s(W)$ is true. Therefore, one of $s_{n-1}(E)$, $s_{n-1}(N)$, or $s_{n-1}(S)$ must be true. By Eq. (3), in the next iteration, i.e. the nth iteration, $s(E)$ becomes true. This means in the nth iteration, s acquires a strength pointing to its eastern neighbour s_{n-1}. By the definition of the algorithm, s move to s_{n-1}. □

Suppose there are n cells in the grid plane ($n = l^2$ for some integer l). On each iteration, the diffusion algorithm diffuses strengths to at least one new cell, so the algorithm finds if there is a path in $O(n)$ time.

6. Moving in rotation space

6.1 Computing configuration space

Now we consider the case where $\mathscr{A}$ is allowed to rotate as well as translate in a two-dimensional plane, i.e. $\mathscr{C} = R^2 \times [0, 2\pi]$.[1] The configuration space obstacles are three-dimensional volumes in R^3. As the workspace is modelled as a bit plane where free space and obstacle areas are distinguished by different pixel values, we model the three-dimensional configuration space as a discretised volume in which the space occupied by the C-obstacles is represented by a collection of voxels whose union encompasses the C-obstacles. The approach we adopted to compute the three-dimensional C-obstacles is to discretise the range of orientation and approximate the three-dimensional C-obstacles by a collection of two-dimensional C-obstacles which are corresponding C-obstacles due to $\mathscr{A}$ at a series of fixed orientations. This method, called 'slice projection', was due to Lozano-Pérez [5–7].

In our particular case, the configuration space $\mathscr{C}$ can be represented as a set of three-element tuples:

$$C = \{(x,y,\theta) | (x,y) \in R^2, \theta \in [0,2\pi|]\}.$$

The algorithm for computing the slice approximation of the C-obstacles decomposing the range of the orientations of $\mathscr{A}$ into a finite number of intervals. Let $I = [\theta, \theta']$ be the range of valid orientations of $\mathscr{A}$. If we want to approximate the three-dimensional configuration space with n θ-slices, we divide I into n non-overlap intervals, $[\theta_1, \theta_1'], \ldots, [\theta_n, \theta_n']$. Then the slice-projection approximation of the C-obstacles are:

$$\mathscr{P}\mathscr{B}_{\mathscr{A}[\theta,\theta']} = \left\{\mathscr{P}\mathscr{B}_{\mathscr{A}[\theta_i,\theta_i']} | 1 \le i \le n\right\}.$$

For any interval $[\theta_1, \theta_1']$, we choose a number of sampling points $\alpha_1, \alpha_2, \ldots, \alpha_m$,

[1] $(x,y,2\pi)$ is identified with $(x,y,0)$.

where $\alpha_i \in [\theta_1, \theta_1']$. $\mathscr{PB}_{\mathscr{A}[\theta_i,\theta_i']}$ is approximated by the union of C-obstacles due to $\mathscr{A}$ at fixed orientations $\alpha_1, \alpha_2, \ldots, \alpha_m$, i.e.

$$\mathscr{PB}_{\mathscr{A}[\theta_i,\theta_i']} \approx \bigcup_{i=1}^{m} \mathscr{CB}_{\mathscr{A}_{\alpha_i}}.$$

In the case that $\mathscr{A}$ is a polygon, the swept volume of $\mathscr{A}$ over the range $[\theta_1, \theta_1']$ can be approximated by a polygon $\mathscr{A}'$ and $\mathscr{PB}_{\mathscr{A}[\theta_i,\theta_i']}$ is the C-obstacle due to $\mathscr{A}'$ [5]. This is a more conservative approximation of the original C-obstacles. The approximated configuration space $\mathscr{C}$ in our bit array representation is represented by a series of bit planes, each corresponding a θ-slice of the exact configuration space.

6.2 Diffusion in 3-space

The diffusion in the three-dimensional configuration space is handled analogously to that of two-dimensional diffusions except that in addition to diffusions in the (x,y)-plane, there have to be diffusions in the (x,z)-plane. We introduce two more strengths: the strength to move *up* and the strength to move *down*, denoted $a_{ijk}(U)$ and $a_{ijk}(D)$ respectively.

As in the two-dimensional case, the diffusion of a strength involves looking at each cell's six neighbours. For a cell a_{ijk}, if we want to decide whether $a_{ijk}(N)$ is true or not, we need to examine its north neighbour a_{ii+1k}. If either $a_{ii+1k}(N)$, $a_{ii+1k}(E)$, $a_{ii+1k}(W)$, $a_{ii+1k}(U)$, $a_{ii+1k}(D)$ is true, then $a_{ijk}(N)$ is set to true. The decisions of other strengths of a a_{ijk} are similar. Thus we can define the equations of diffusion strengths in three-dimensional space (the other five directions are similiar).

$$\begin{aligned} a_{ijk}^{t+1}(N) = \big(a_{ijk}^{t}(N) \vee\ a_{ij+1k}^{t}(N) \vee a_{ij+1k}^{t}(E) \vee a_{ij+1k}^{t}(W) \vee \\ \times a_{ij+1k}^{t}(U) \vee a_{ij+1k}^{t}(D)\big) \wedge a_{ijk}^{t}(Free). \qquad (5) \end{aligned}$$

The algorithm implementing these equations makes use of parallelism in the (x,y)-plane. Initially, the free neighbours of the goal point are set strengths pointing to it. There are then two parts of iterations: (1) diffusion in the (x,y)-planes for S, N, E, W strengths; and (2) diffusion vertically for U and D strengths.

Suppose the workspace consists of n cells and we use n processors ($n = l^2$). Then the configuration space consists of $n\sqrt{n}$ cells. It's not hard to find that the algorithm runs in $O(n^4\sqrt{n})$.

7. Experimental results

The algorithms were originally implemented on AMT DAP510 which is an SIMD machine (ideal structure for implementing cellular automata) with 32×32 array of processors. Each processor is connected to its four nearest neighbours (north, south, east, and west) and each processor has its own memory. The program was written in DAP Fortran which exploits the bit serial nature of the individual processors to give fast execution time for the binary representations. Many compuations are conveniently expressed with the built-in function 'shift', whose functions we defined in our compuational model in Section 3.

When working on a map of configuration space with a resolution of 32×32, each grid of the map is directly mapped onto the corresponding processor. When working on resolutions which are greater than 32×32, the map is represented by mutiple 32×32 matricx. Parallel instructions are applied on single matrix, while between matrixes, instructions have to be excuted repeatedly. We present the results of two groups of experiments demonstrating features of the path planning methods discussed in this paper.

7.1 Rotational motions

The algorithm was tested in many environments with the robot being allowed to rotate as well as translate. Here we present six examples (Fig. 8) typically illustrated in the motion planning literature. Table 1 shows the timing results of these examples. These results compare favourably with ones reported that have been generated by appoximate cell decomposition methods. For instance, Example 3 was first reported in [1] requiring tens of minutes on a Lisp machine. This result was recently greatly improved by Zhu and Latombe [19] whose algorithm runs in 5.5 minutes to solve this problem on a Machintosh II. Our algorithm is able to solve this problem in 84.20 seconds (1.40 minutes) on the AMT DAP510.

Note that these examples use simple polygonal environments which favours methods based on piecewise linear models in the comparison. Fig. 9 illustrates a path around arbitrarily curved obstacles. Our algorithm solves this problem in less than 30 seconds.

8. Conclusion

We have presented an effective and efficient approach to the search of a static configuration space. There are two characteristics of this approach, namely, the distributed representation of the workspace and, related to it, the use of parallel processing techniques to manipulate the world model. The distributed representation is a natural abstraction for the geometry of the real world objects. Unlike the piecewise linear model, or more generally, the semi-algebraic model which represents the boundaries of objects, it can be thought of as the kind of representation which characterises the whole object. The main advantage of using this representation in motion planning is that it is easily obtainable from sensory systems, such as vision, and therefore is most useful for motion planning problems in unknown environments.

The data intensive nature of the distributed representation suggests the use of a massively parallel architecture. We used a Distributed Array Processor (DAP)

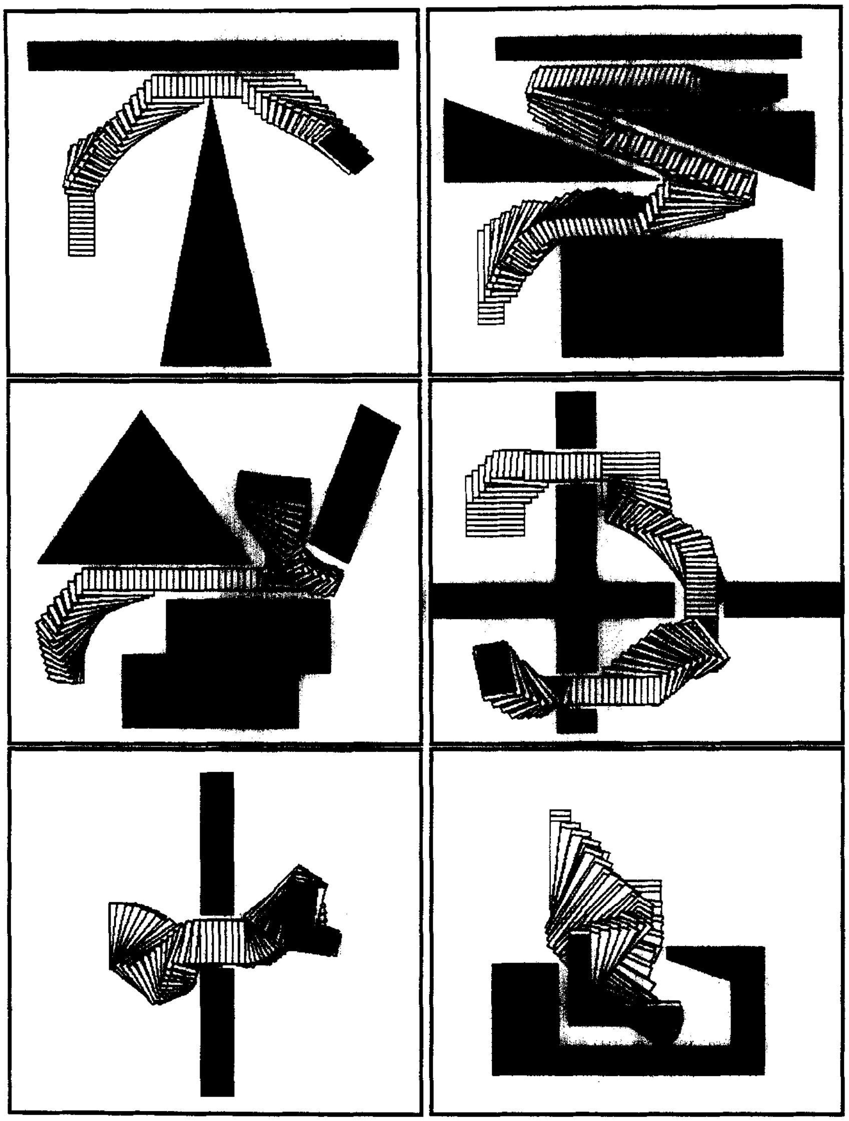
Fig. 8. Path planning with rotation - Example 1–6.

model which has simple local connections among processors. We found that by using simple parallel matrix operations, many geometric transformations for motion planning can be achieved efficiently. In particular, we first described an algorithm for computing configuration space obstacles. The algorithm runs in time basically proportional to the size of the robot. Second, given a discrete representation of the configuration space, the search in this space can also be performed in parallel. The parallel search algorithm given in this paper exploits local constraints by diffusion stengths in the space. It is interesting to note that the complexity of

Table 1
Timing for experiments with rotation motions. Times are given in seconds

Example	Rotation pattern	C-space	Diffusion	Total
1	19.45	0.58	57.40	77.62
2	19.45	0.83	170.78	191.25
3	19.45	0.72	63.83	85.20
4	19.45	0.67	156.72	180.04
5	19.45	21.19	17.78	58.62
6	19.45	22.04	3.68	45.36

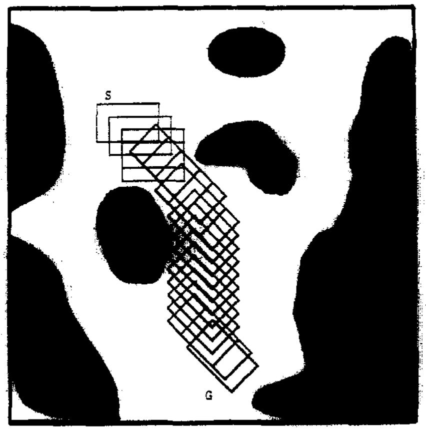

Fig. 9. Path planning in an environment filled with curved obstacles.

this algorithm depends on the resolution of the workspace map rather than the complexity of the environment. This is a unique feature of algorithms based on a distributed representation. It implies that this algorithm can handle complicated environments efficiently. The algorithm finds shortest path in $O(n^2)$ time for fixed orientation motions and in $O(n^4\sqrt{n})$ time for rotational as well as translational motion, where n is the number of cells comprising the environment map. In the special case where the robot can be approximated as a disc, the performance of the algorithm can be improved by adaptive use of a hierarchy of resolutions [13]. In conclusion, the approach presented here seems well suited to a range of problems in robotic path planning as it can deliver the shortest path effectively in a known amount time. It is also possible to extend the approach in a very natural way to dynamic motion planning problems [14].

References

[1] R.A. Brooks and T. Lozano-Pérez, A subdivision algorithm in configuration space for findpath with rotation, *IEEE Trans. Systems, Man, and Cybernetics* 15(2) (1985) 224–233.
[2] J.F. Canny, *The Complexity of Robot Motion Planning* (MIT Press, Cambridge, MA, 1988).

[3] F. Dehne, A-L Hassenklover and J-R Sack, Computing the configuration space for a robot on a mesh of processors, *Parallel Comput.* 12 (1989) 221–231.

[4] D. Leven and M. Sharir, An efficient and simple motion planning algorithm for a ladder amidst Polygonal barriers, *J. Algorithms* 8 (1987) 192–215.

[5] T. Lozano-Pérez, Automatic planning of Manipulator transfer movements, *IEEE Trans. Systems, Man, and Cybernetics,* SMC-11(10) (1981) 681–698.

[6] T. Lozano-Pérez. Spatial planning: A configuration space approach, *IEEE Trans. Computers* C-32 (2) (1983) 108–120.

[7] T. Lozano-Pérez, A simple motion-planning algorithm for general robot manipulators, *IEEE Trans. Robotics and Automation,* RA – 3(3) (1987) 224–238.

[8] C. Ó'Dúnlaing and C.K. Yap, A retraction method for planning the motion of a disc, *J. Algorithms,* 6 (1982) 187–192.

[9] J.H. Reif, Complexity of the generalized mover's problem, in [12] (1987) 267–281.

[10] J.T. Schwartz and M. Sharir, On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds, *Advances in Applied Math.* 4 (1983) 298–351.

[11] J.T. Schwartz and M. Sharir, On the piano movers' problem: The case of a rod moving in three dimensional space amidst polyhedral obstacles. *Commun. Pure and Applied Math.* 37 (1983) 815–848.

[12] J.T. Schwartz, M. Sharir, and J.E. Hopcroft, *Planning, Geometry, and Complexity of Robot Motion* (Ablex, Norwood, NJ, 1987).

[13] C. Shu ad H. Buxton, A parallel path planning algorithm for mobile robots, *In Proc. 1st Inter. Conf. Automation, Robotics and Computer Vision*, Singapore (1990) 489–493.

[14] C. Shu and H. Buxton, Dynamic motion planning using a distributed representation, *J. Intelligent and Robotic Systems*, 12 (1995) 1–12.

[15] L. Steels, Steps towards common sense, Technical Report VUB AI Lab Memo 88-3, Vrije Universiteit Brussel, 1988.

[16] C.E. Thorpe, FIDO: Vision and navigation for a robot rover, Technical Report No. CMU-CS-84-168, Department of Computer Science, Carnegie-Mellon University, 1984.

[17] I. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modelling* (MIT Press, Cambridge, MA, 1987).

[18] S. Walfram, Statistical mechanics of cellular automata, *Rev. Modern Physics* 55 (3) (1983) 601–644.

[19] D. Zhu and J-C Latombe, New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. Robotics and Automation* 7 (1) (1991) 9–20.