CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Global Path Planning Approaches for Material Movements in a Worksite

by
Jean-Claude Latombe

TECHNICAL REPORT

Number 3

May, 1988

Stanford University

Global Path Planning Approaches for Material Movements in a Worksite

Jean-Claude Latombe

Robotics Laboratory, Computer Science Department
Stanford University, Stanford, CA 94305, USA

Abstract

In various environments (e.g., manufacturing shopfloors, civil engineering construction sites, space) there is an increasing need to efficiently transport objects from locations to other locations. Although most practical material transportation robotic system built so far have been quite primitive, we believe that in many areas one can significantly gain in efficiency, reliability and flexibility by automatically planning the motions of the transportation devices. While process planning provides a high-level 'logical' and possibly 'temporal' specification of material movements, motion planning says how these movements are to be 'physically' carried out. Thus, motion planning is the natural intermediate stage between process planning and task execution. In this paper, we survey techniques for planning mobile robot paths among obstacles, which have been developed over the last few years. We focus on the so-called 'global' techniques. We describe in detail the three most common approaches, which are based on the notions of cell decomposition, free space retraction, and visibility graph, respectively. Within the first two approaches, we survey both the so-called exact and approximate techniques. Although this paper is far from exploring all the facets of motion planning, it gives a fundamental and detailed presentation of issues, which are of general interest to all motion planning problems. These issues are likely to be of prime importance in future material transportation systems.

Key-Words

Material Tranportation, Robotics, Autonomous Robots, Spatial Reasoning, Motion Planning, Path Planning.

1 Introduction

In various environments there is an increasing need to efficiently transport objects from locations to other locations. For instance, in a machining shop-floor blanks and partly cut parts, as well as tools (e.g., fixtures, cutting tools) have to be moved from storage areas or machines to other machines, where the machining process plans proceed further. In a civil engineering construction site, a large variety of materials such as earth, steel bars, concrete beams, pipes, and prefabricated components have to be frequently moved over a large and unstructured working area, in a much un-repetitive fashion. In space, construction of platforms will require moving frame components out from the aircraft to the partially assembled platform structure. In most of these application areas, material

handling and transportation turn out to be a critical issue responsible for an important share of the operating costs, and also for many delays. Mobile robots or similar devices are likely candidates to improve the way these tasks are currently performed.

Robotized material movements in a worksite requires that the transportation robots be provided with advanced capabilities for planning their motions. In fact, motion planning is the natural intermediate stage between process planning (or task planning), which determines and specifies the various operations that have to be executed in order to perform a task and the actual execution of these operations. Indeed, process planning provides a 'logical' and possibly 'temporal' specification of material movements. However, it does not say how these movements are to be 'physically' carried out. This is the purpose of motion planning.

We believe that in many areas one can significantly gain in efficiciency, reliability and flexibility by automating the motion planning stage. We envision an architecture integrating process planning and motion planning in a hierarchical fashion. In many cases, process planning will be performed off-line, possibly in a non-automated fashion, while motion planning will be an on-line capability implemented in the robot controller. In situations where process plans may often change, both process and motion planning capabilities will have to be available on-line; then, motion planning will still occur at a more detailed level and probably within a shorter time perspective, but it will directly return feedback constraints to the process planning stage.

Most practical material transportation robotic systems built so far have been quite primitive, using only low-level control techniques. In these systems, motion planning has been done by hand and expressed in the form of task-specific programs driving the robots through the planned paths. Often, control is simplified further by materializing potential paths using specifically designed environmental features (e.g., wiring guides). Typically, robots controlled in this manner can only follow predetermined routes, limiting their application to carefully engineered environments and, in these environments, restricting their flexibility and versatility. In addition, the burden of programming the robots is put on the users.

On the other hand, most of the early robotic projects based on the use of Artificial Intelligence techniques attempted (rather unsuccessfully) to directly connect a high-level planning and monitoring system to a low-level execution controller, thus skipping the motion planning level. In fact, for a long period of time, motion planning has been considered as a secondary and relatively easy problem. But, experience has shown that efficient techniques are needed at this intermediate level. This is not surprising. A robot operates in a physical world in which geometry plays an important role, and it seems rather natural that a somewhat autonomous robot be equipped with advanced techniques for reasoning about this world. Experience and theoretical studies (e.g., [43,49]) have also shown that motion planning is quite complex, both from the conceptual and computational points of view.

Motion planning requires to make use of spatial knowledge about the workspace in order to plan collision-free paths among fixed and moving obstacles. It must also be able to deal with lack of knowledge about the environment and to generate robust plans

allowing to face contingencies, such as unexpected obstacles. It should also be capable of reasoning about uncertainties in object locations, robot control and sensing, and able to generate sensory-based motion plans coping with these uncertainties. Therefore, motion planning has many facets. In this paper, we present techniques for planning paths among obstacles, which have been developed over the last few years. Our discussion is limited in scope and focuses on basic issues along the following lines:

1. We concentrate on the motion planning problem for a *moving rigid object*. This means that we mainly address the case of a mobile robot. The presented techniques are directly applicable to ground-based mobile robots, whose workspace can be realistically approximated as a two-dimensional workspace. The principle of most of the techniques can easily be extended to free-flying robots operating in three-dimensional workspaces, at the expense of increased computations. The problem of planning the motions of an articulated object, such as a manipulator arm, is not addressed in this paper, although several of the presented techniques can be generalized to this case (see [49,33]).

2. We concentrate on the so-called *global* path planning techniques. All these techniques consist of, first constructing a representation of the global topology of the space of collision-free positions and orientations of the robot in the form of a graph, and next searching this graph for a path. Another family of path planning techniques are known as the *local* techniques. These techniques base their decisions only on local considerations. Since they avoid to explicitly represent free space topology, they may occasionally be faster than global techniques. But because they tend to search very large graphs, their worst-case computational complexity is much higher than the worst-case complexity of global methods. The most well-known and successful local path planning technique is the Potential Field method [22].

3. We assume that the robot has a complete and accurate world model prior to planning. This means that we do not consider the case where the workspace is 'discovered' by the robot as it moves, nor the case where there are significant uncertainties in the robot's model. Dealing with incomplete knowledge is often considered an easier problem when local techniques are used rather than global ones. However, we believe that this is basically an architectural issue, in which one has to appropriately interweave motion planning within a limited subset of the workspace (the portion of space for which the robot has a model) and execution (which allows the robot to extend its spatial knowledge of the workspace through sensing). Dealing with uncertainties is more difficult. It requires to produce sensory-based motion plans, not just paths. Other techniques than thosed presented below are needed to that purpose. We refer the reader to [24] for a presentation of the motion planning problem in the presence of uncertainty.

4. We only consider the case of a single robot moving among fixed obstacles. Several of the methods presented below can easily be extended, in principle, to handle the case of multiple robots. One general approach is to consider the set of all the robots as a single multiple-bodied robot with many degrees of freedom. However, motion planning is known to be NP-hard in the number of degree of freedom [43], so that this approach results in a substantial increase of computational complexity. Another approach to multiple-robot path planning is the so-called *prioritarized* approach [18], which consists

of planning the motion of one robot at a time using one of the techniques described in this paper, and considering the robots whose motions have already been planned as obstacles to the other robot. This approach, however, is not guaranteed to succeed whenever collision-free paths exist for all the robots. The case of moving obstacles is easy to handle as long as there are no bounds on the velocity and the acceleration of the robots. Dealing with such bounds in a rigorous fashion makes the problem much harder.

Thus, this paper is far from exploring all the facets of motion planning. However, it gives a fundamental and detailed presentation of issues which are of general interest to the other facets of motion planning. Furthermore, these issues are likely to be of prime importance in future material transportation systems, if we want to increase the flexibility and ease of use of these systems. Path planning techniques also have more specific applications; for instance, moving long bars, pipes and beams through a building under construction is known to raise tough planning problems interfering with the building construction process, whether material handling is robotized or not. Nevertheless, due to the limited scope or the paper and its theoretical nature, the reader should be constantly aware that a practical implementation of any of the techniques described below requires a lot of domain-specific engineering.

There exist two major global approaches to motion planning, namely *decomposition* and *retraction*. These approaches are 'universal' in the sense that they are applicable to path planning in general, not just to specific instances of this problem. Decomposition consists of representing 'free space', i.e. the set of collision-free positions and orientations of the robot as a finite collection of cells. Path planning is then reduced to finding a sequence of free cells such that any two successive cells in the sequence are 'adjacent'. Retraction consists of mapping free space on a network of curves, so that path planning is reduced to searching this network.

Within both approaches, there are two sorts of techniques, the *exact* and the *approximate* ones. The exact techniques are based on rather involved mathematics and are complete in the sense that they can produce a path whenever one exists. The approximate techniques are based on various approximations and/or assumptions, and they are not complete. Implementation of the exact techniques raises difficult issues, such as doing exact computations with rational or algebraic numbers. Implementing the approximate techniques is usually easier.

Exact decomposition techniques are presented in Sections 3 through 5. Approximate decomposition techniques are described in Sections 6 and 7. The concept of exact retraction is introduced in Section 8, and an approximate method based on this concept is detailed in Section 9. Finally, Section 10 describes a third approach to path planning, known as the *visibility graph* approach. Unlike the previous two, this latter approach is not universal, but, due to its conceptual simplicity, it is a popular one.

Our presentation of path planning methods makes intensive use of the concept of *configuration space*. We describe this concept in the following section and we formalize the path planning problem.

2 Configuration Space

The path planning problem may be formulated as the problem of constructing a continuous collision-free sequence of positions and orientations of an object (the robot) moving among obstacles from an initial position/orientation to a final one. Configuration space is a conceptual tool that makes the constraints on the moving object explicit.

Let us consider an object $\mathcal{A}$, which we indifferently call the *moving object* or the *robot*. $\mathcal{A}$ can move in a space, $\mathcal{W}$, called *workspace*, of dimension 2 or 3. Typically, in the case of a ground-based mobile robot, $\mathcal{W}$ is approximated as a space of dimension 2 and is isomorphic to $\Re^2$. In the case of a general manipulator arm or a three-dimensional free-flying object, its dimension is 3 and it is isomorphic to $\Re^3$.

Let us assume that a Cartesian frame $\mathcal{F}_\mathcal{A}$ (resp. $\mathcal{F}_\mathcal{W}$) has been attached to $\mathcal{A}$ (resp. $\mathcal{W}$).

DEFINITION 1: *A* **configuration** $\mathbf{c}$ *of* $\mathcal{A}$ *is a specification of the position and orientation of* $\mathcal{F}_\mathcal{A}$ *with respect to* $\mathcal{F}_\mathcal{W}$. *The* **configuration space** *of* $\mathcal{A}$ *is the space, denoted* $\mathcal{C}$, *of all the possible configurations of* $\mathcal{A}$. *The subset of* $\mathcal{W}$ *occupied by* $\mathcal{A}$ *at configuration* $\mathbf{c}$ *is denoted* $\mathcal{A}(\mathbf{c})$. *In the same fashion, the point* a *on* $\mathcal{A}$ *at configuration* $\mathbf{c}$ *is denoted* $a(\mathbf{c})$ *in* $\mathcal{W}$.

The configuration space $\mathcal{C}$ of $\mathcal{A}$ is a manifold of dimension m, which can be embedded in an Euclidean space of dimension N ($N \geq m$) [1,51]. The topology on $\mathcal{C}$ is the subspace topology induced by the Euclidean metric in the N-dimensional space or, equivalently, by the following distance function d on $\mathcal{C}$:

$$\forall \mathbf{c}, \mathbf{c}' \in \mathcal{C} \;\; [d(\mathbf{c}, \mathbf{c}') = \max_{a \in \mathcal{A}} distance(a(\mathbf{c}), a(\mathbf{c}'))]$$

where *distance* denotes the Euclidean distance in $\mathcal{W}$.

For instance:

- If $\mathcal{A}$ is only allowed to translate in $\mathcal{W} = \Re^k$ ($k = 2$ or 3), then $\mathcal{A}$'s configuration can be defined as the coordinates of the origin of $\mathcal{F}_\mathcal{A}$ in $\mathcal{F}_\mathcal{W}$. Thus, $\mathcal{C} = \Re^k$ (more precisely, $\mathcal{C}$ is isomorphic to $\Re^k$) and $m = N = k$.

- If $\mathcal{A}$ translates and rotates in $\Re^2$, $\mathcal{A}$'s configuration can be represented as a list of three parameters (x, y, θ), the coordinates x and y of the origin of $\mathcal{F}_\mathcal{A}$ and the angle θ between the x-axes of $\mathcal{F}_\mathcal{A}$ and $\mathcal{F}_\mathcal{W}$. This parametrization, however, does no make explicit the fact that in this case $\mathcal{C}$ is a cylinder, i.e. that the orientation is defined by an angle modulo 2π. In fact, $\mathcal{C} = \Re^2 \times S^1$, where S^1 is the unit circle in $\Re^2$. Thus, $\mathcal{C}$ is a manifold of dimension 3 (a 3-cylinder) that can be embedded in an Euclidean space of dimension 4 [51].

- If $\mathcal{A}$ translates and rotates in $\Re^3$, $\mathcal{A}$'s configuration can be represented as a list of six parameters, say the three coordinates of the origin of $\mathcal{F}_\mathcal{A}$ and the three Euler angles [6] specifying the orientations of $\mathcal{F}_\mathcal{A}$'s axes with respect to $\mathcal{F}_\mathcal{W}$. Again this parametrization does not reflect the topology of $\mathcal{C}$. In this case, $\mathcal{C} = \Re^3 \times SO(3)$, where SO(3) is the

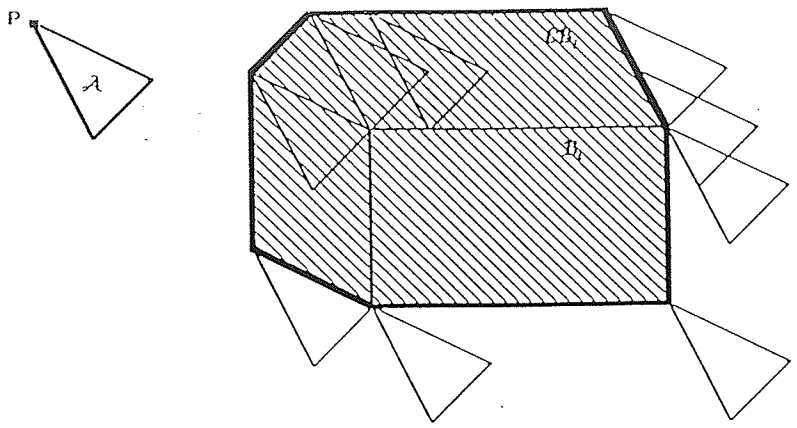

Figure 1: **C-obstacle in the Convex Polygonal Case**

special orthogonal group made of 3×3 matrices with orthonormal rows (and columns) and determinant 1. The space of 3×3 matrices with real coefficients is a vector space of dimension 9. Othonormality of the rows generates 6 independent polynomial equations specifying two three-dimensional subsets of this space. The constraint that the determinant should be $+1$ selects one of these subsets. The set $\Re^3 \times SO(3)$ is a manifold of dimension 6, which can be embedded in the Euclidean space of dimension 12. The constraints on the matrices in $SO(3)$ determines the manifold in $\Re^{12}$.

One of the reasons why path planning is difficult is that, except in the translational case, $\mathcal{C}$ is a multiply-connected manifold. This can easily be illustrated in the case where $\mathcal{C} = \Re^2 \times S^1$. Any two paths which make, respectively, i and j $(i \neq j)$ net turns around S^1 are not homotopic (i.e. one cannot be continuously deformed into the other) [21].

Let us assume now that the workspace $\mathcal{W}$ contains obstacles $\mathcal{B}_i$ $(i = 1, 2, ...)$. We model all objects, both $\mathcal{A}$ and the $\mathcal{B}_i$s, as closed bounded (i.e., compact) subsets of $\mathcal{W}$. Each $\mathcal{B}_i$ is mapped in $\mathcal{C}$ as another region, $\mathcal{CB}_i$, called a C-obstacle and defined as follows:

DEFINITION 2: *The obstacle region $\mathcal{B}_i$ in $\mathcal{W}$ maps into $\mathcal{C}$ as the region $\mathcal{CB}_i = \{\mathbf{c} \in \mathcal{C} \,/\, \mathcal{A}(\mathbf{c}) \cap \mathcal{B}_i \neq \emptyset\}$. $\mathcal{CB}_i$ is called a* **C-obstacle**.

Figure 1 illustrates the case where both $\mathcal{A}$ and $\mathcal{B}_i$ are convex polygonal regions in $\Re^2$. If $\mathcal{A}$ keeps a fixed orientation, then $\mathcal{C} = \Re^2$, and the C-obstacle corresponding to $\mathcal{B}_i$ is shown in Figure 1. It is easy to see that $\mathcal{CB}_i$ is another convex polygonal region. If $\mathcal{A}$ can both rotate and translate, then $\mathcal{C} = \Re^2 \times S^1$, and the representation of $\mathcal{CB}_i$ in a (x, y, θ) Cartesian frame, where $\theta \in [0, 2\pi)$ represents the orientation of $\mathcal{A}$ (modulo 2π) is a three-dimensional volume bounded by patches of ruled surfaces. Every cross-section through this volume is a polygonal region representing the C-obstacle in $\Re^2$ for a fixed orientation of $\mathcal{A}$ [32].

There exist rather straighforward techniques for computing the representation of $\mathcal{CB}_i$

from the representation of $\mathcal{A}$ and $\mathcal{B}_i$, when both objects are convex polygons or polyhedra [32,16]. The computed representation of $\mathcal{CB}_i$ when both $\mathcal{A}$ and $\mathcal{B}_i$ are convex polygons is of the form:

$$(x, y, \theta) \in \mathcal{CB}_i \Leftrightarrow \bigwedge_k [\theta \in \delta_{ik} \Rightarrow (a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) \leq 0)]$$

where δ_{ik} is a subinterval of $[0, 2\pi)$ and $a_{ik}(\theta)$, $b_{ik}(\theta)$, and $c_{ik}(\theta)$ are continuous function of θ. The inequality $a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) \leq 0$ is called a **C-constraint** (the equality represents a ruled surface). The interval δ_{ik} defines the range of orientations of $\mathcal{A}$ in which the C-constraint applies. In the case where $\mathcal{A}$ has a fixed orientation (i.e., it can only translate), the representation simplifies to an expression of the form:

$$(x, y, \theta) \in \mathcal{CB}_i \Leftrightarrow \bigwedge_{k'} (a_{ik'}x + b_{ik'}y + c_{ik'} \leq 0).$$

If, more generally, both $\mathcal{A}$ and $\mathcal{B}_i$ are represented as semi-algebraic sets, then $\mathcal{CB}_i$ is also a semi-algebraic set, whose representation is computable (see Section 5).

C-obstacles explicitly describe the constraints on the motions of $\mathcal{A}$. A second reason why path planning is difficult is that, whenever the moving object $\mathcal{A}$ is allowed to rotate, C-obstacles may be complex regions bounded by curved surfaces.

The set of collision-free positions and orientations of the robot $\mathcal{A}$ is defined as follows:

DEFINITION 3: Free space *is* $\mathcal{C}_{free} = \{\mathbf{c} \in \mathcal{C} \,/\, \mathcal{A}(\mathbf{c}) \cap (\bigcup_i \mathcal{B}_i) = \emptyset\}$.

In the same way, we can define valid space as the set of configurations of $\mathcal{A}$ where the interior of $\mathcal{A}$ and the interiors of the $\mathcal{B}_i$s have null intersection:

DEFINITION 4: Valid Space *is* $\mathcal{C}_{valid} = \{\mathbf{c} \in \mathcal{C} \,/\, int(\mathcal{A}(\mathbf{c})) \cap (\bigcup_i int(\mathcal{B}_i)) = \emptyset\}$[1].

At a valid configuration, $\mathcal{A}$ may touch an obstacle.

A collision-free (more simply, a free) path of $\mathcal{A}$ in $\mathcal{C}$ is formalized as follows:

DEFINITION 4: *A* **free path** *of* $\mathcal{A}$ *between an initial configuration* $\mathbf{c}_1$ *and a final configuration* $\mathbf{c}_2$ *is a continuous map* $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, *with* $\tau(0) = \mathbf{c}_1$ *and* $\tau(1) = \mathbf{c}_2$.

Thus, given a configuration space $\mathcal{C}$, C-obstacles $\mathcal{CB}_i$ in this space, and an initial and final configurations $\mathbf{c}_1$ and $\mathbf{c}_2$, collision-free path planning consists of constructing a mapping τ as defined above. We now describe various techniques applicable to this problem.

3 Exact Decomposition: The Translational Case

The principle of an exact decomposition method is to decompose free space into non-overlapping cells whose union is equal to free space. Any such decomposition is not

[1] Throughout this paper, $int(S)$, $cl(S)$, and $\partial(S)$ respectively denote the interior, the closure, and the boundary of the subset $S \subset E$, where E is a topoligical space.

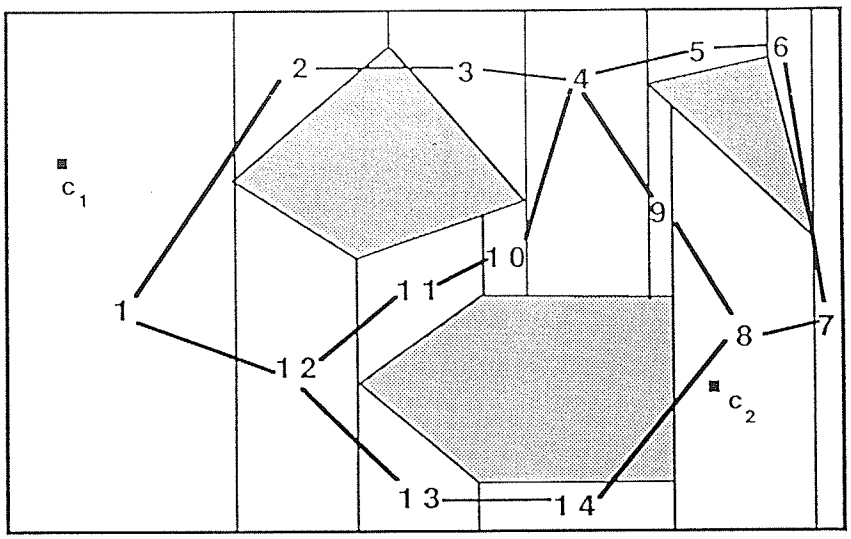

Figure 2: **Vertical Trapezoidal Decomposition of Free Space**

appropriate, however. Cells should have the following two characteristics. First, they should be simple enough so that we can find a path between any two points in a cell. Second, it should be possible to test adjacency between cells and to find a path crossing the common portion of boundary between two adjacent cells.

In this section, we introduce the exact cell decomposition approach in the simple case where the moving object $\mathcal{A}$ is a polygon only allowed to translate among fixed polygonal obstacles. By decomposing the moving object and the obstacles into convex polygons, we can easily construct possibly overlapping convex polygonal C-obstacles $\mathcal{CB}_i$, $i = 1$ to q (see Section 2). Let n be the total number of vertices of these C-obstacles.

Let us assume that $\mathcal{C}_{free}$ is bounded by a rectangle R whose edges are parallel to the x and y-axes embedded in $\mathcal{C}$. Thus $\mathcal{C}_{free} = int(R) - \bigcup_i \mathcal{CB}_i$. The edges of R are equivalent to mechanical stops on the physical translations of $\mathcal{A}$ along the x and y-axes.

DEFINITION 5: *A* **open convex decomposition** *of $\mathcal{C}_{free}$ is a finite collection of open convex polygons, called* **cells**, *such that any two cells do not intersect and the closed union of all the cells is equal to $cl(\mathcal{C}_{free})$. Two cells c and c' are* **adjacent** *if only if $cl(c) \cap cl(c')$ is a line segment of positive length (i.e., a set of non-null measure).*

Figure 2 shows a possible convex decomposition of the free space and the *connectivity graph* representing the adjacency relation between the cells. Cells are labelled by numbers and the non-directed arcs of the connectivity graph are displayed as bold lines connecting these numbers.

Consider an initial configuration $\mathbf{c}_1$ and a final configuration $\mathbf{c}_2$, which are both in free space. We want to generate a free path from $\mathbf{c}_1$ to $\mathbf{c}_2$. The exact decomposition approach in this specific case proceeds as follows [12]:

1. Generate a convex decomposition of $\mathcal{C}_{free}$.

2. Construct the connectivity graph CG representing the adjacency relation between

cells.

3. Search CG for a path linking the cell containing $\mathbf{c}_1$ to the cell containing $\mathbf{c}_2$.

4. If the search terminates successfully return the corresponding sequence of cells; otherwise, return failure.

The output is a sequence $c_1, ..., c_p$ of cells such that $\mathbf{c}_1 \in c_1$, $\mathbf{c}_2 \in c_p$ and $\forall j \in [1, p-1]$, c_j and c_{j+1} are adjacent. Let β_j designate the common edge of c_j and c_{j+1}, i.e. $\beta_j = cl(c_j) \cap cl(c_{j+1})$. One way to transform the sequence of cells into a path is to consider the midpoints Q_j of every edge β_j, and to connect $\mathbf{c}_1$ to Q_1, Q_j to Q_{j+1} ($j = 1$ to $p-2$), and Q_{p-1} to $\mathbf{c}_2$ by straight line segments.

The generation of a convex decomposition of a polygon is a classical problem in Computational Geometry. The 'Optimal Convex Decomposition' problem is solvable in time polynomial in the number n of edges [13] when there is no hole. But, as shown by Lingas [29], the presence of holes makes this problem NP-hard. Nevertheless, a non-optimal decomposition can be generated rather efficiently as shown below.

A simple technique consists of sweeping a line L parallel to the y-axis, inside R, from left to right ('line-sweep' paradigm [42]). Whenever the line encounters a new vertex X, if X is not inside a C-obstacle, one vertical line segment is created connecting X to the edges immediately above and below it (see Figure 2). The C-obstacles edges, the boundary of R and the vertical line segments determine a *vertical trapezoidal decomposition* of C_{free}. Two cells of this decomposition are adjacent if their closures share a portion of a vertical segment. Before line sweeping, the vertices of the C-obstacles can be sorted along the x-axis in $O(n \log n)$. Then, as the vertical line L is swept from left to right the segments of the intersections $L \cap \mathcal{CB}_i$ (for all $i \in [1, q]$) are kept dynamically in a balanced tree. During line sweeping, using this structure, it is possible to concurrently create the vertical line segments emanating from vertices, generate the connectivity graph CG between the trapezoidal cells, and identify the cells containing the initial and final configurations $\mathbf{c}_1$ and $\mathbf{c}_2$. This whole sweep-line technique produces the decomposition of C_{free} in $O(n \log n)$ time. The number of generated trapezoidal cells is $O(n)$. The number of arcs in CG is also $O(n)$.

Graph searching can be done in various ways. A systematic 'breadth-first' exploration of the graph takes $O(n)$ time. Indeed, each node is explored at most once and each arc is traversed at most once. Average time can be improved by using heuristics to guide the search (see [36]).

Therefore, the total time complexity of the method described above is $O(n \log n)$.

The method can be extended to the three-dimensional case. But the decomposition phase is more time consumming. In particular, the optimal decomposition of a polyhedron without hole is known to be NP-hard [29]. However, non-optimal decompositions of polyhedra with holes are still possible in reasonable time.

4 Exact Decomposition: Case of a Segment

We now illustrate the exact decomposition approach to path planning in a more difficult, but still specific case, where C-obstacles are bounded by curved surfaces. The robot $\mathcal{A}$ is a line segment (sometimes called a *ladder*) moving in a two-dimensional workspace $\mathcal{W}$ among non-intersecting compact (possibly non-convex) polygonal obstacles $\mathcal{B}_i$. $\mathcal{A}$'s end-points are denoted P and Q. The configuration space $\mathcal{C}$ of $\mathcal{A}$ is $\Re^2 \times S^1$. We represent each configuration by a triplet (x, y, θ), where x and y are the coordinates of P in $\mathcal{F}_\mathcal{W}$ and $\theta \in [0, 2\pi)$ is the angle (modulo 2π) between the x-axis of $\mathcal{F}_\mathcal{W}$ and the segment PQ.

The method presented below is due to Schwartz and Sharir [47]. The basic idea is to decompose the set of $\mathcal{A}$'s positions into *regular regions*, to 'lift' these regions into *cells*, and to represent the adjacency relation among the cells as a *connectivity graph*. The regular regions are such that the C-obstacles maintain a constant 'structure' in the cylinders above these regions, and the cells projecting on a regular region make this structure explicit. The boundaries of the regular regions are called *critical curves*.

The time complexity of the technique described below is $O(n^5)$, where n is the number of vertices (or edges) of the obstacles. Assymptotically more efficient techniques have been recently proposed (e.g. [27]). However, the method described below is a good illustration of the exact decomposition approach and its principle is rather simple.

Throughout this section, we assume that the boundary of every obstacle in the workspace is partitioned into *open* edges and vertices.

4.1 Critical and Regular Positions of the Segment

DEFINITION 6: *A position (x, y) of $\mathcal{A}$ is* **admissible** *if there exists at least one orientation θ such that $(x, y, \theta) \in \mathcal{C}_{free}$.*

For each admissible position (x, y) of $\mathcal{A}$, the set $F(x, y)$ of all free orientations of $\mathcal{A}$ – i.e., $F(x, y) = \{\theta \ / \ (x, y, \theta) \in \mathcal{C}_{free}\}$ – is a finite collection of open intervals. Each interval endpoint is a contact orientation of $\mathcal{A}$, i.e. $\mathcal{A}$ touches the boundary of at least one obstacle. In a similar fashion, we denote $V(x, y) = \{\theta \ / \ (x, y, \theta) \in \mathcal{C}_{valid}\}$ the set of all valid orientations of $\mathcal{A}$ at position (x, y). It is made of the closure of the open intervals in $F(x, y)$ and eventual isolated orientations. The set $L(x, y) = V(x, y) - F(x, y)$ contains a finite number of orientations, which are called **limit orientations** of $\mathcal{A}$ at (x, y).

DEFINITION 7: *Let θ be a limit orientation of $\mathcal{A}$ at (x, y) and s an obstacle's edge or vertex touched by $\mathcal{A}(x, y, \theta)$. If for all arbitrarily small clockwise (resp. counterclockwise) rotations of $\mathcal{A}$ about P, s (if s is an edge) or at least one edge abutting at s (if s is a vertex) intersects PQ, then s is called a* **clockwise** *(resp.* **counterclockwise***)* **stop** *at (x, y).*

DEFINITION 8: *A limit orientation θ at (x, y) is* **exceptional** *if and only if*

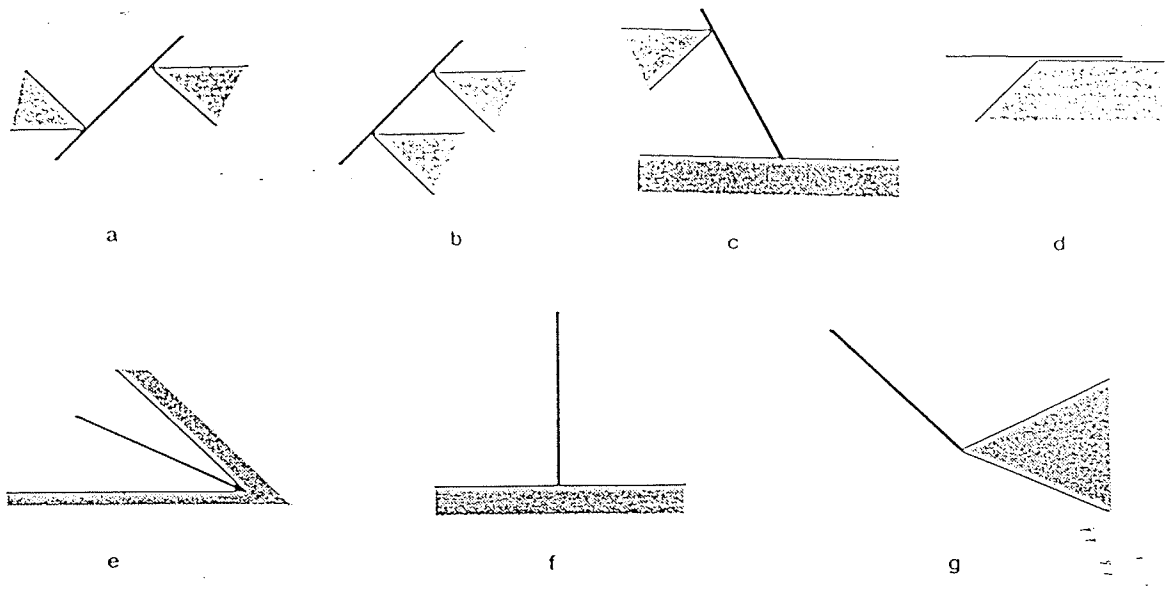

Figure 3: **Exceptional Orientations of the Moving Objects**

$\mathcal{A}(x, y, \theta)$ does not touch a unique stop. A position (x, y) is **regular** *if it has no exceptional orientation; otherwise it is* **critical**.

Thus, a limit orientation θ is exceptional at (x, y) if $\mathcal{A}(x, y, \theta)$ is touching more than one stop, or if it is touching no stop. $\mathcal{A}(x, y, \theta)$ touches more than one stop if it touches two vertices (Figure 3 *a* and *b*), or a vertex and an edge (Figure 3 *c* and *d*), or a concave vertex (Figure 3 *e*). Notice that in the latter case, the same vertex is counted as two stops (a clockwise one and a counterclockwise one). $\mathcal{A}(x, y, \theta)$ touches no stop if the only contact of $\mathcal{A}$ with an obstacle is at the Q extremity in one of the following two ways: either $\mathcal{A}(x, y, \theta)$ is perpendicular to an edge with Q touching that edge (Figure 3 *f*), or Q coincides with a convex vertex X and, in a sufficiently small neighborhood of X, the line perpendicular to PQ at Q intersects obstacles only at X (Figure 3 *g*).

Let (x, y) be a regular position of $\mathcal{A}$. The set $V(x, y)$ consists of finitely many disjoint closed intervals, none being an isolated point. For each such interval $[\theta_1, \theta_2]$, the unique stop touched by $\mathcal{A}(x, y, \theta_1)$ (resp. $\mathcal{A}(x, y, \theta_2)$) is a clockwise (resp. counterclockwise) stop. If all orientations of $\mathcal{A}$ are free, then $F(x, y) = [0, 2\pi)$, where $[0, 2\pi)$ is the 'circular' interval with no extremity.

If θ is a limit orientation at the regular position (x, y), $s(x, y, \theta)$ designates the unique stop touched by $\mathcal{A}(x, y, \theta)$. We denote $\sigma(x, y)$ the set of all the pairs $[s(x, y, \theta), s(x, y, \theta')]$ such that $s(x, y, \theta)$ (resp. $s(x, y, \theta')$) is a clockwise (resp. counterclockwise) stop at (x, y) and the interval $(\theta, \theta') \subseteq F(x, y)$. If $F(x, y) = [0, 2\pi)$, then we write $\sigma(x, y) = \{[\Omega, \Omega]\}$, where Ω designates a 'nonexistent' stop. Given a pair $[s_1, s_2] \neq [\Omega, \Omega]$ in $\sigma(x, y, \theta)$, we denote $\lambda_1(x, y, s_1)$ the unique orientation such that $\mathcal{A}$ positioned at (x, y) touches s_1 in such a way that s_1 is clockwise stop. Similarly, we denote $\lambda_2(x, y, s_2)$ the unique orientation such that $\mathcal{A}$ positioned at (x, y) touches s_2 in such a way that s_2 is a counterclockwise stop. By convention: $\lambda_1(x, y, \Omega) = 0$ and $\lambda_2(x, y, \Omega) = 2\pi$.

One can show the following:

LEMMA 1: *Let (x, y) be a regular position of $\mathcal{A}$. There exists an open neighborhood U of this position that only consists of regular positions and such that, $\forall (x', y') \in U$, we have $\sigma(x', y') = \sigma(x, y)$ (i.e., there is a 1-1 onto correspondance between the angular intervals of $F(x', y')$ and those of $F(x, y)$). Moreover, if $[s_1, s_2]$ belongs to $\sigma(x', y') = \sigma(x, y)$, then $\lambda_1(x', y', s_1)$ and $\lambda_2(x', y', s_2)$ are continuous functions of (x', y') for $(x', y') \in U$.*

Thus the set of regular positions of $\mathcal{A}$ is an open subset of $\Re^2$. We now analyze how this set is separated into regions by critical positions.

4.2 Critical Curves and Regular Regions

The locus of all critical positions of $\mathcal{A}$ is the union of a finite collection of curves, called **critical curves**. The intuition behind these curves is that $\sigma(x, y)$, i.e. the structure of C-obstacles above the position (x, y), changes when a critical curve is crossed.

Let d be the length of the segment $\mathcal{A}$. Critical curves are obtained[2], by considering the exceptional orientations shown in Figure 3:

(1) For each obstacle edge E, the line segment at distance d from E is a **Type I** critical curve (see Figure 4 *a*).

(2) For each obstacle vertex X, the circular arc of radius d about X and bounded by the two half-lines containing the edges adjacent to X is a **Type II** critical curve (see Figure 4 *b* and *c*).

(3) For each pair of obstacle vertices X_1 and X_2, the line segment traced by P as $\mathcal{A}$ translates while touching both X_1 and X_2 is a **Type III** critical curve (see Figure 4 *d*).

(4) For each obstacle edge E and convex vertex X at the extremity of E, the line segment traced by P as $\mathcal{A}$ slides along E is a **Type IV** critical curve (see Figure 4 *e*).

(5) For each obstacle edge E and convex vertex X not at the extremity of E, the curve traced by P as $\mathcal{A}$ moves touching both E and X is a **Type V** critical curve (see Figure 4 *f*).

The Type V critical curve is a conchoid of Nicomedes, which is described by an algebraic equation of degree 4.

The collection of critical curves is finite in a workspace with finitely many obstacle edges. Each curve extremity is located either on an obstacle boundary, or at the extremity of another critical curve. In addition, every critical curve is a smooth algebraic curve of degree 1 (Types I, III, and IV), 2 (Type II), or 4 (Type V). Thus, every two critical curves can have only finitely many intersections, except if they coincide. A coincidence happens, for example, when two parallel edges are separated by the distance $2d$. In

[2]This classification differs slightly from that of Schwartz and Sharir [47]. The differences, however, are only superficial.

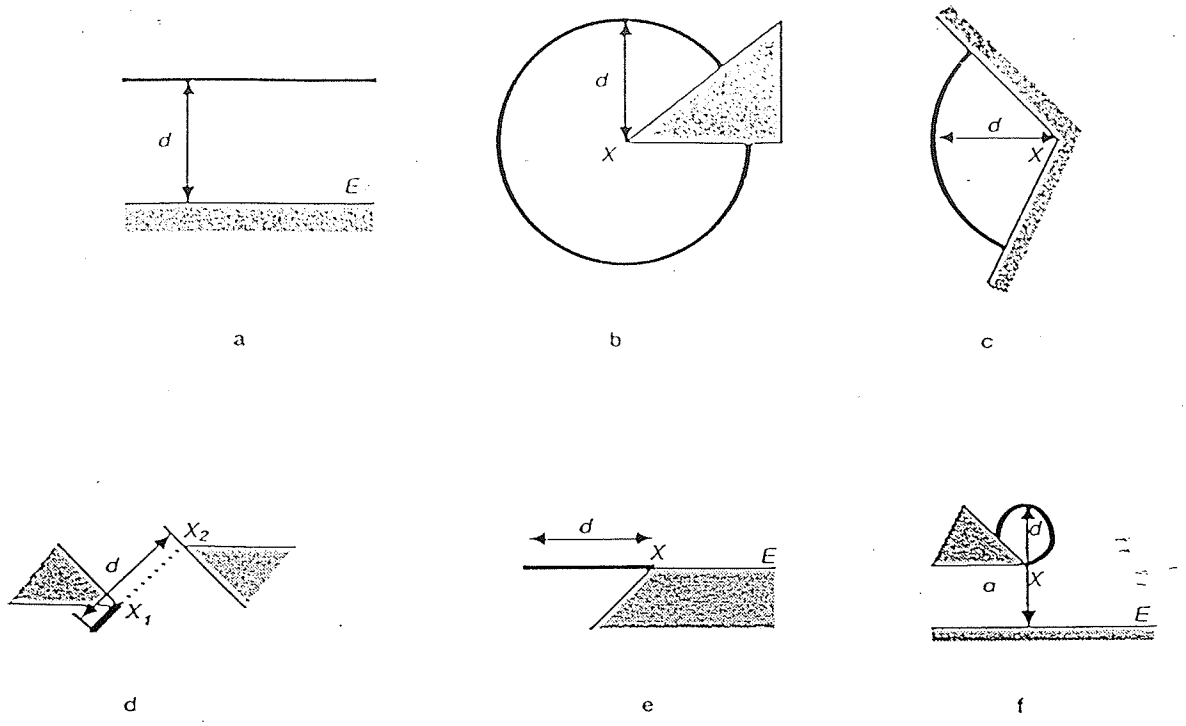

Figure 4: **Critical Curves**

theory, all coincidences can be eliminated by shifting some obstacles by very small amounts. Thus, we will assume no curve coincidence.

Notice here that several critical curves constructed as above are likely to be supersets of critical positions of $\mathcal{A}$. Indeed, each curve is built by considering one or two features (edge and/or vertex). Hence, the exceptional orientations for some sections of these curves, which make these curve critical, may correspond to non-valid configurations. As we will see, such a 'redundant' section of a critical curve needs not be explicitely removed.

It follows from Lemma 1 and the above discussion that:

LEMMA 2: *The set of admissible positions of $\mathcal{A}$ is divided by critical curves into a finite collection of open connected regions R, called* **regular region**. *The set $\sigma(x, y)$ is constant in each such region. In addition, for any such region R, any $(x, y) \in R$ and any pair $[s_1, s_2] \in \sigma(x, y)$, the region*

$$c(R, s_1, s_2) = \{(x, y, \theta) \;/\; (x, y) \in R \text{ and } \theta \in (\lambda_1(x, y, s_1), \lambda_2(x, y, s_2))\}$$

is an open connected region of C_{free}. It is called a **cell**[3].

Each cell $C = c(R, s_1, s_2)$ is homeomorphic to $\Re^3$, hence has no hole.

[3]Throughout this paper, the word *cell* is given several formal definitions. Within each section, however, the word keeps the same definition.

For any cell $C = c(R, s_1, s_2)$, $\phi_C(x, y)$ and $\psi_C(x, y)$ designate the limit clockwise and counterclockwise orientations at (x, y) such that:

$$(x, y, \theta) \in C \quad \Leftrightarrow \quad (x, y) \in R \text{ and } \theta \in (\phi_C(x, y), \psi_C(x, y)).$$

In addition, we write $\sigma(R) = \sigma(x, y)$, where (x, y) is an arbitrary point in R.

The boundary of any regular region consists of closed sections of critical curves and closed sections of obstacles edges.

4.3 Connectivity Graph

At this point we have decomposed free space into open connected cells separated by 'vertical' surfaces projecting on the (x, y)-plane along the critical curves. Let us now examine how these surfaces can be crossed in order to connect adjacent cells. We will analyze the crossing rules in more detail in the next subsection.

The following lemma drawn from Differential Geometry allows us to discard some transitions between cells which would be quite cumbersome to analyze.

LEMMA 3: *If there is a free path τ' between two free configurations projecting on the (x, y)-plane at regular positions, then there is another free path $\tau : t \in [0, 1] \mapsto (x(t), y(t), \theta(t)) \in C_{free}$ between the same two configurations that satisfies:*
- The curve $(x(t), y(t))$ has a well-defined tangent everywhere along its length.
- The curve $(x(t), y(t))$ does not pass through any point common to two critical curves.
- At each intersection of $(x(t), y(t))$ with a critical curve β, there is a neighborhood U of this point in which $\theta(t)$ is constant and the tangent vector $(\dot{x}(t), \dot{y}(t))$ is constant and lies transversal to β.

Let $C = c(R, s_1, s_2)$ be a cell. We extend by continuity the domain of ϕ_C and ψ_C to the closure $cl(R)$ of R.

DEFINITION 9: *Two cells $C = c(R, s_1, s_2)$ and $C' = c(R', s'_1, s'_2)$ are* **adjacent** *if and only if R and R' share an open section β of a critical curve and, for all $(x, y) \in \beta$, $(\phi_C(x, y), \psi_C(x, y)) \cap (\phi_{C'}(x, y), \psi_{C'}(x, y)) \neq \emptyset$.*

It immediately follows from Lemma 3 that if two cells C and C' are adjacent, any configuration in C can be connected to any configuration in C' by a differentiable path whose projection on the (x, y)-plane crosses β transversally, with a constant orientation in some neighborhood of the crossing point.

DEFINITION 10: *The* **connectivity graph** *CG is the graph whose nodes are all cells $c(R, s_1, s_2)$, where R is a maximal regular region in the (x, y)-plane and $[s_1, s_2] \in \sigma(R)$. An undirected arc connects any two nodes labeled by adjacent cells.*

It follows from the above that:

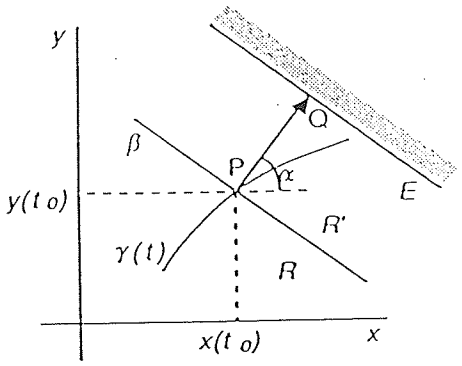

Figure 5: **Crossing a Type I Curve**

THEOREM 1: *Given two configurations* $\mathbf{c}_1 = (x_1, y_1, \theta_1)$ *and* $\mathbf{c}_2 = (x_2, y_2, \theta_2)$ *in* $\mathcal{C}_{free}$, *such that neither* (x_1, y_1), *nor* (x_2, y_2) *lies on a critical curve, there exists a free path between them if and only if the two cells containing* $\mathbf{c}_1$ *and* $\mathbf{c}_2$ *are connected by a path in the connectivity graph CG.*

This theorem is the basis for the algorithm sketched in Subsection 4.5. Before presenting the algorithm, let us analyze the crossing of a critical curve in more detail.

4.4 Crossing Rules

Let β designate a critical curve section not intersected by any other critical curve. C and C' designate two cells $c(R, s_1, s_2)$ and $c(R', s_1', s_2')$, such that $[s_1, s_2] \in \sigma(R)$, $[s_1', s_2'] \in \sigma(R')$, and β forms part of the boundaries of R and R'. $\tau : t \in [0,1] \mapsto (x(t), y(t), \theta(t))$ is a path such that the projected curve $\gamma(t) = (x(t), y(t))$ crosses β transversally at $t = t_0$, with $\theta(t)$ constant and equal to θ_0 in some interval $(t_0 - \epsilon, t_0 + \epsilon)$.

When $t \rightarrow t_0$ with $t < t_0$:

$$(\phi_C(x(t), y(t)), \psi_C(x(t), y(t))) \rightarrow (\phi_C(x(t_0), y(t_0)), \psi_C(x(t_0), y(t_0))).$$

When $t \rightarrow t_0$ with $t > t_0$:

$$(\phi_{C'}(x(t), y(t)), \psi_{C'}(x(t), y(t))) \rightarrow (\phi_{C'}(x(t_0), y(t_0)), \psi_{C'}(x(t_0), y(t_0))).$$

The **crossing rule** for a certain type of curve section β is the description of all the connections between cells C and C'. Let us consider the case of a Type I curve as an example, and analyze it in detail.

If β is a Type I curve section, it is a line segment parallel to an obstacle edge E at distance d. Let R' be the region between β and E and α the orientation of the segment $\mathcal{A}$ when P is on β and Q on E (see Figure 5).

If $\alpha \notin (\phi_C(x(t_0), y(t_0)), \psi_C(x(t_0), y(t_0)))$, then the two cells C and C' are adjacent if and only if $(\phi_C(x(t_0), y(t_0)), \psi_C(x(t_0), y(t_0))) = (\phi_{C'}(x(t_0), y(t_0)), \psi_{C'}(x(t_0), y(t_0)))$.

If $\alpha \in (\phi_C(x(t_0), y(t_0)), \psi_C(x(t_0), y(t_0)))$, then when $\gamma(t)$ crosses β the interval of free orientations breaks into exactly two subintervals. The two cells C and C' are adjacent if and only if $(\phi_{C'}(x(t_0), y(t_0)), \psi_{C'}(x(t_0), y(t_0))) = (\phi_C(x(t_0), y(t_0)), \alpha)$ or $(\phi_{C'}(x(t_0), y(t_0)), \psi_{C'}(x(t_0), y(t_0))) = (\alpha, \psi_C(x(t_0), y(t_0)))$.

Thus, the crossing rule for a Type I critical curve section is:

The two cells $c(R, s_1, s_2)$ *and* $c(R', s_1', s_2')$ *are adjacent if and only if* $[s_1', s_2'] = [s_1, s_2]$, *or* $[s_1', s_2'] = [s_1, E]$, *or* $[s_1', s_2'] = [E, s_2]$.

It is straightforward to analyze in the same fashion all the other cases. Such an analysis shows that whenever $\gamma(t)$ crosses β the set $\sigma(x, y)$ is changing in one of the following ways:

- One pair in $\sigma(x, y)$ disappears, or one new pair appears.

- One element in one pair in $\sigma(x, y)$ changes.

The crossing rules for the different types of critical curves can be summarized by:

Connect $c(R, s_1, s_2)$ *to* $c(R', s_1, s_2)$ *for each* $[s_1, s_2] \in \sigma(R) \cap \sigma(R')$, *and connect each* $c(R, s_1, s_2)$, $[s_1, s_2] \in \sigma(R) - \sigma(R')$, *if any, to each* $c(R', s_1', s_2')$, $[s_1', s_2'] \in \sigma(R') - \sigma(R)$, *if any.*

This generic rule handles redundant sections of critical curves, since in that case $\sigma(R) = \sigma(R')$.

4.5 Sketch of the Algorithm

Given a set of compact polygonal obstacles, the length d of the segment $\mathcal{A}$, and the initial and final free configurations $\mathbf{c}_1 = (x_1, y_1, \theta_1)$ and $\mathbf{c}_2 = (x_2, y_2, \theta_2)$ of $\mathcal{A}$, such that neither configuration projects on the (x, y)-plane on a critical curve, the path finding algorithm sketched below generates a free path from $\mathbf{c}_1$ to $\mathbf{c}_2$ whenever one exists, and returns failure otherwise.

This algorithm does not explicitly build the regular regions by collecting the curve sections bounding them together. Instead, for every curve section β, it gives an orientation to β and we designate the two regions separated by β by $right(\beta)$ and $left(\beta)$. As a consequence, the algorithm builds a connectivity graph which is slightly different from the one defined above. In this new graph, each cell appears as many times as there are curve sections forming the boundary of the regular region on which the cell projects.

Since each regular region may be bounded by both portions of critical curves and portions of obstacles edges, it is convenient in the algorithm to treat edges as a special kind of critical curve which cannot be crossed. In addition, since each curve section β may be followed in two directions, it is also convenient to implicitly treat it as a pair of oppositely oriented sections (β^+, β^-), such that $right(\beta^+)$ and $left(\beta^-)$ (resp. $left(\beta^+)$ and $right(\beta^-)$) are identified.

At several points in the algorithm, we have to compute $\sigma(x, y)$ and the limit orientations corresponding to the clockwise and counterclockwise stops appearing in $\sigma(x, y)$. This can be done by rotating a half-line emanating from (x, y) and using a kind of line-sweep algorithm.

With these preliminary remarks, the algorithm is the following:

1. **Compute the boundary of the regular regions.** Compute the set of all critical curves (including eventual redundant sections). Add all obstacle edges to this set. Find intersection points of the closures of these curves with each other. For each curve, sort these intersection points according to their order on this curve (this is easily done by defining a suitable parametrization for each possible type of curve). Decompose each curve into sections, each lying between two consecutive intersection points.

2. **Compute the adjacency relation between cells.** For every curve section β, which is part of a critical curve, compute $\sigma(right(\beta))$ and $\sigma(left(\beta))$, use the general crossing rule, and build a map $ADJACENT$ which maps every cell projecting on $right(\beta)$ into the adjacent cells projecting on $left(\beta)$. If $\sigma(right(\beta)) = \sigma(left(\beta))$, β is redundant and may be removed from further consideration. For every curve section β which is part of an obstacle edge, set the map $ADJACENT$ to $\emptyset$.

3. **Form clusters of intersecting curve sections.** At every intersection point Z of two or more curve sections, form the circular list of all curve sections emerging from Z sorted in clockwise order (this list is called a *cluster*). To this end, compute and sort the outgoing tangential directions of the curve sections. Whenever two such directions are equal, compute and sort higher-order derivative directions.

4. **Locate initial and final cells.** Draw the line segment between (x_1, y_1) and (x_2, y_2). Compute the intersections of this segment with all curve sections. Find the curve sections β_1 and β_2 whose intersections with the line segment are nearest from (x_1, y_1) and (x_2, y_2), respectively. Identify the side of β_1 (resp. β_2) that contains (x_1, y_1) (resp. (x_2, y_2)). Find the pair $[s_{11}, s_{21}]$ (resp. $[s_{12}, s_{22}]$) such that $\theta_1 \in (\lambda_1(x_1, y_1, s_{11}), \lambda_2(x_1, y_1, s_{21}))$ (resp. $\theta_2 \in (\lambda_1(x_2, y_2, s_{12}), \lambda_2(x_2, y_2, s_{22}))$). Let C_i, $i = 1,2$, be the cell containing (x_i, y_i, θ_i).

5. **Search the connectivity graph.** Build a connectivity graph defined as follows. Each node is a cell $c(R, s_1, s_2)$ where R is identified as one side of a curve section β, i.e. $right(\beta)$ or $left(\beta)$. Two nodes $c(R, s_1, s_2)$ and $c(R', s_1', s_2')$ are linked by an undirected arc if one of the following two conditions is satisfied:

   - $R = left(\beta)$, $R' = right(\beta')$, β' follows β in a cluster, and $[s_1, s_2] = [s_1', s_2']$.
   - $R = right(\beta)$, $R' = left(\beta)$, and $c(R', s_1', s_2') \in ADJACENT(c(R, s_1, s_2))$.

   Initialize the graph with node C_1. Then, expand each node that has not been expanded yet. Stop either when the node C_2 has been generated, or when there is no more nodes to expand. In the first case, output the sequence of cells on the path connecting C_1 to C_2 in the graph. In the second case, indicate failure.

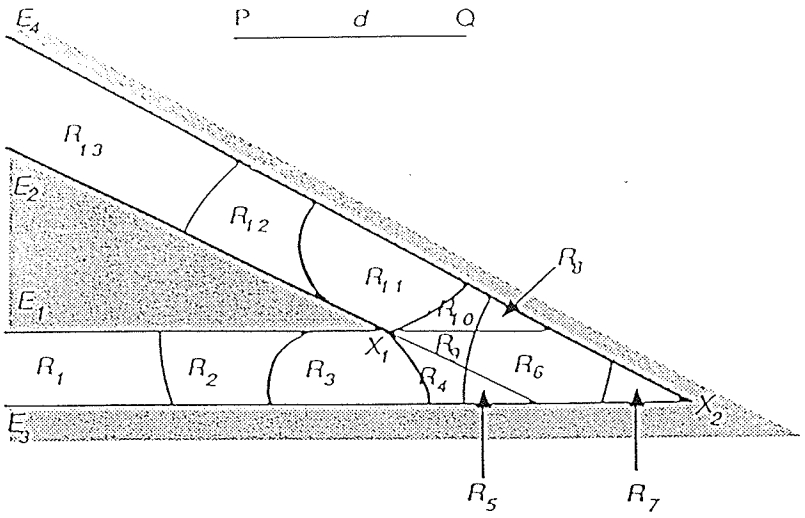

Figure 6: **The Corner Example**

The union of the set of cells output by the algorithm and the set of common boundaries between every two successive cells is a region of C_{free} homeomorphic to $\Re^3$. It is 'easy' to transform the sequence into a free path, by following and crossing the specified curve sections appropriately and maintaining the orientation in the appropriate open interval.

Notice that in order to implement the algorithm, one would have to pay attention to details not considered above. One delicate issue is related to the sorting of intersection points. With limited precision arithmetic, the identification of two separately calculated intersection points as being the same point is indeed problematical.

A detailed analysis of the algorithm shows that its time complexity is $O(n^5)$.

4.6 Example

Consider the 'corner example' shown at Figure 6. Only the non-redundant critical curves sections are shown in the figure (i.e., we assume that the redundant ones are removed at Step 2 of the algorithm). There are 13 regular regions denoted R_1 through R_{13}. We leave the identification of each critical curve section as an exercise to the reader.

We have:

$$\sigma(R_1) = \{[E_1, E_3], [E_3, E_1]\}$$
$$\sigma(R_2) = \{[E_1, E_3], [E_3, X_1]\}$$
$$\sigma(R_3) = \{[E_1, E_3], [E_3, E_4]\}$$
$$\sigma(R_4) = \{[E_1, E_3], [E_3, E_4], [E_4, X_1]\}$$
$$\sigma(R_5) = \{[E_1, E_3], [E_4, X_1]\}$$
$$\sigma(R_6) = \{[E_1, E_3], [E_4, E_2]\}$$
$$\sigma(R_7) = \{[E_4, E_3]\}$$
$$\sigma(R_8) = \{[E_4, E_2], [X_1, E_3]\}$$
$$\sigma(R_9) = \{[E_4, E_2], [E_1, E_3], [E_3, E_4]\}$$
$$\sigma(R_{10}) = \{[E_4, E_2], [X_1, E_3], [E_3, E_4]\}$$
$$\sigma(R_{11}) = \{[E_4, E_2], [E_1, E_4]\}$$

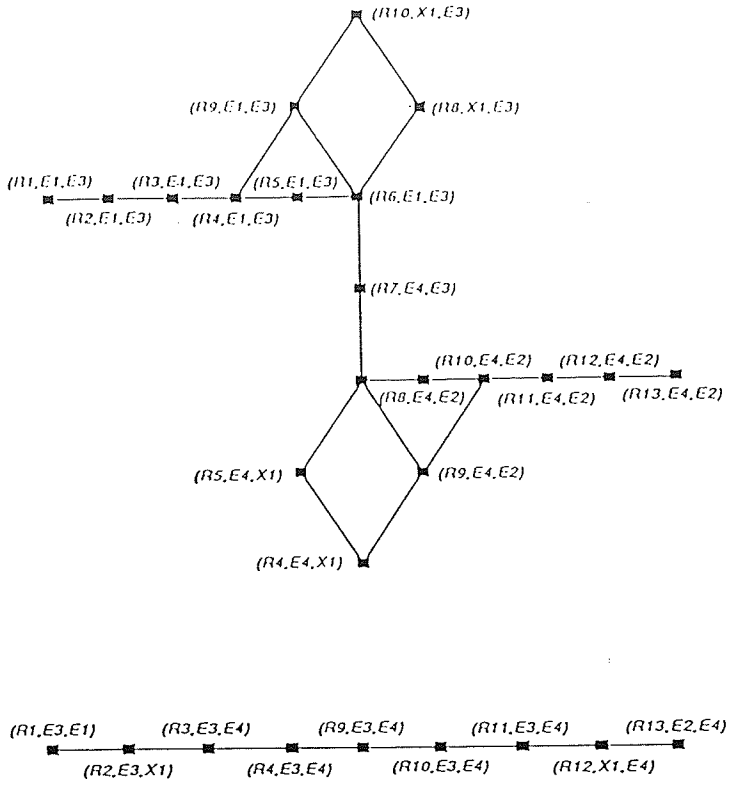

Figure 7: **Connectivity Graph for the Corner Example**

$$\sigma(R_{12}) = \{[E_4, E_2], [X_1, E_4]\}$$
$$\sigma(R_{13}) = \{[E_4, E_2], [E_2, E_4]\}$$

Figure 7 shows the resulting connectivity graph. This graph contains two connected components, which means that $\mathcal{C}_{free}$ has also two connected components. Practically, this implies that the segement $\mathcal{A}$ cannot rotate around the corner. But it can move from one end of the corner to the other with a 'forward - backward' or 'backward - forward' motion.

5 Exact Decomposition: General Method

In the previous two sections we illustrated the exact cell decomposition approach to path planning in two specific cases. Below, we conclude our presentation of the approach by the description of a general method which put no limitation[4] on the dimension k of the workspace ($k = 2$ or 3) and on the possible motions of the robot $\mathcal{A}$. The only constraint imposed by the method is that both the moving object $\mathcal{A}$ and the obstacles $\mathcal{B}_i$ be described as semi-algebraic sets (see Subsection 5.1).

This general method works according to the same principle as the methods described in the previous section. It consists of exactly partitioning $\mathcal{C}_{free}$ into a finite collection of semi-algebraic connected cells, computing the adjacency relation between the cells and representing it as a graph, searching this graph, and transforming the obtained sequence

[4]This general method is also applicable to articulated objects, i.e. manipulator arms.

of cells into a path. As in Section 4, cells in $\mathcal{C}_{free}$ are obtained by first computing their projection in a lower-dimensional space and then lifting them in $\mathcal{C}$. Since the dimension m of $\mathcal{C}$ may be greater than 3, the projection/lift mechanism is recursive, that is, it produces projections of cells on spaces of dimensions $m-1, m-2, ..., 1$, before lifting them back through the same sequence of spaces. Due to this recursive computation, this path finding method is sometimes called 'projection method'.

The method is due to Schwartz and Sharir [48], who considered the more general case of an articulated robot (e.g., a manipulator arm) with many degrees of freedom. It makes use of a well-known result by Collins [15] for deciding the satisfiability of Tarski sentences. We expose this result first. The general path finding algorithm based on this result is sketched next. Our presentation, which is inspired from [54], concentrates on the main ideas of the method. The actual algorithm requires to pay attention to many details, including exact computation with algebraic numbers, which are not covered here. We refer the reader to the original paper for more detail.

5.1 Semi-Algebraic Representation of C-Obstacles

Let $Q[X_1; ...; X_n]$ be the ring of polynomials in n real variables with rational coefficients.

Let us assume that every object in $\mathcal{W}$ is represented as a conjunction of inequations of the form $P(x) \leq 0$, where $x \in \Re^k$ and $P \in Q[X_1; ...; X_k]$. A point $x_0 \in \Re^k$ belongs to the object if and only if the all the inequalities are verified at this point.

Let us denote $\Lambda_{\mathcal{A}}(a)$ and $\Lambda_{\mathcal{B}_i}(b)$, with a and $b \in \Re^k$, the representation of $\mathcal{A}$ and an obstacle $\mathcal{B}_i$, respectively. Without loss of generality, we assume that $\Lambda_{\mathcal{A}}$ represents $\mathcal{A}$ at configuration $\mathbf{0}$, i.e. when $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{F}_{\mathcal{W}}$ coincide. As mentioned previously, the configuration space $\mathcal{C}$ can always be embedded in an Euclidean space of some dimension N. Any configuration $\mathbf{c}$ of $\mathcal{A}$ can be represented as a vector of N real numbers related by $N-m$ independent polynomial equations with rational coefficients expressing the fact that $\mathbf{c}$ belongs to a manifold of dimension m.

Let us denote $\mathbf{x}$ the representation of $\mathbf{c}$ as a N-dimensional vector and $\Gamma(\mathbf{x})$ the conjunction of the $N-m$ polynomial equations constraining this representation. Let $\Upsilon(\mathbf{x}, a, b)$, with $a \in \Re^k$ and $b \in \Re^k$, be the conjunction of polynomial equations expressing that a point, a, fixed with respect to $\mathcal{F}_{\mathcal{A}}$ is such that $a(\mathbf{x})$ coincides with a point, b, fixed with respect to $\mathcal{F}_{\mathcal{W}}$.

The C-obstacle corresponding to $\mathcal{B}_i$ can be represented in $\Re^N$ by the following expression:

$$\mathcal{CB}_i = \{\mathbf{x} \in \Re^N \ / \ \exists a \in \Re^k \exists b \in \Re^k : \Lambda_{\mathcal{A}}(a) \wedge \Lambda_{\mathcal{B}_i}(b) \wedge \Upsilon(\mathbf{x}, a, b) \wedge \Gamma(\mathbf{x})\}.$$

Such an expression is an instance of a Tarski sentence over $\Re^N$.

DEFINITION 11: *An* **atomic polynomial expression** *over $\Re^j$ is one of the form:*

$$P(x) \bowtie 0$$

where $x \in \Re^j$ and $P \in Q[X_1; ...; X_j]$ and $\bowtie$ can be any comparator in $\{=, \neq, >, <, \geq, \leq\}$. A **polynomial expression** *over $\Re^j$ is any finite boolean combination of atomic polyno-*

mial expressions over $\Re^j$. A subset of $\Re^j$ whose points verify a polynomial expression is called a **semi-algebraic set**. *A* **Tarski sentence** *over $\Re^j$ is any polynomial expression over $\Re^j$ prefixed by a finite number of $\exists$ and $\forall$ quantifiers.*

Tarski showed that any Tarski sentence has an equivalent quantifier-free polynomial expression [52]. Collins [15] gave a constructive proof of this result (see Subsection 5.2). Thus:

THEOREM 2: *If both the moving object $\mathcal{A}$ and the obstacles $\mathcal{B}_i$ are represented as semi-algebraic sets in $\Re^k$, then all C-obstacles, hence C_{free}, are semi-algebraic subsets of $\Re^N$.*

5.2 Cylindrical Algebraic Decomposition of $\Re^N$

DEFINITION 12: *A* **decomposition** *of $\Re^N$ is a finite collection Δ of disjoint connected subsets of $\Re^N$ whose union is $\Re^N$. Each element c of Δ is called a* **cell**. *Δ is an* **algebraic decomposition** *if each cell in Δ is a semi-algebraic set homeomorphic to $\Re^j$, with $j = 0, 1, ..., N$ (if $j = 0$, then the cell consists of a single algebraic point).*

The fact that each cell in an algebraic decomposition is homeomorphic to some $\Re^j$ means that it contains no hole.

Two cells c_1 and c_2 of a decomposition Δ are **adjacent**[5] if and only $(cl(c_1) \cap c_2) \cup (c_1 \cap cl(c_2)) \neq \emptyset$.

A **sample** of an algrebraic decomposition Δ of $\Re^N$ is a function $\sigma : \Delta \rightarrow \Re^N$ such that for all $c \in \Delta$, $\sigma(c) \in c$ and each coordinate of $\sigma(c)$ is an algebraic number[6].

The **projection** of a point $\mathbf{x}_N = (x_1, ..., x_N) \in \Re^N$ is $\mathbf{x}_{N-1} = (x_1, ..., x_{N-1}) \in \Re^{N-1}$, obtained by omitting the Nth component of the point. The projection of a subset of $\Re^N$ is the set of projections of its members.

DEFINITION 13: *A* **cylindrical algebraic decomposition** *Δ_N of $\Re^N$ $(N > 0)$ is an algebraic decomposition recursively defined as follows:*
- For $N > 1$, there exists a cylindrical algebraic decomposition Δ_{N-1} of $\Re^{N-1}$ such that for each cell c in Δ_N there is a cell c' in Δ_{N-1} that is the projection of c. Δ_{N-1} is called the **base decomposition** *of Δ_N, and c is said to be* **based** *on c'.*
- For $N = 1$, Δ_1 is just a partitioning of $\Re$ into a finite set of algebraic numbers and into the finite and infinite open intervals bounded by these numbers.

The collection of all the cells c in Δ_N which have the same projection c' in Δ_{N-1} is called the **cylinder** over c'.

[5]This definition of adjacency is consistent with the definitions given in the previous two sections. The difference of formulation comes from the fact that here we consider cells of different dimensions.

[6]The set of *algebraic numbers* is the set of real roots of polynomials in $Q[X]$.

A **cylindrical sample** of a cylindrical algebraic decomposition of $\Re^N$, Δ_N, is a sample $\sigma : \Delta_N \rightarrow \Re^N$, such that:
- if c_1 and c_2 are cells of Δ_N having the same base then $\sigma(c_1)$ and $\sigma(c_2)$ have the same projection in $\Re^{N-1}$
- if $N > 1$, σ recursively induces a cylindrical sample of the base decomposition Δ_{N-1} of Δ_N.

DEFINITION 14: *Let signum(a) be -1, 0, or $+1$ whenever $a < 0$, $a = 0$, or $a > 0$, respectively. Let $\mathcal{F}$ be a set of functions of N real variables. A decomposition Δ of $\Re^N$ is said to be* **$\mathcal{F}$-invariant** *if, for each cell c in Δ and each function f in $\mathcal{F}$, signum$(f(\mathrm{x}))$ is constant as $\mathrm{x} = (x_1, ..., x_N)$ ranges over c.*

THEOREM 3 (Collins): *Given any set $\mathcal{P}$ of polynomials in $Q[X_1; ...; X_N]$ there is an effective algorithm which constructs a $\mathcal{P}$-invariant cylindrical algebraic decomposition Δ_N of $\Re^N$ together with a cylindrical sample of Δ. The time complexity of the algorithm is double-exponential in n.*

The constructive proof of this theorem is the Collins algorithm constructing the $\mathcal{P}$-invariant decomposition Δ_N (we do not describe it here). Each of the cells produced by the algorithm is described by a quantifier-free polynomial expression over $\Re^N$. The algorithm also produces the base decomposition Δ_{N-1} of Δ_N, the base decomposition Δ_{N-2} of Δ_{N-1}, etc. The Collins algorithm takes time polynomial in the number of polynomials in $\mathcal{P}$ (*geometric* complexity) and in their maximum degree (*algebraic* complexity)[7], with double exponential dependence on N. The number of cells generated by the algorithm has the same order of magnitude[8].

5.3 Application to the First-Order Theory of Reals

An immediate application of the Collins algorithm is the construction of a procedure capable of deciding the satisfiability of input Tarski sentences. Two more specific applications are of special interest here: truth decision and quantifier elimination. We illustrate these two applications below on simple examples. It is easy to generalize these examples into general procedures. In the subsequent subsection we will use quantification elimination to construct a cylindrical algebraic decomposition of C_{free} and truth decision to test adjacency between cells of this decomposition.

- **Truth decision:** The problem is to determine the truth value of a Tarski sentence that includes no free variable. For instance, let us consider the sentence $(\exists x)(\forall y)[\phi(x, y)]$, where both x and y are variables in $\Re$. Let P be the set of polynomials appearing in expression ϕ. Using Collins algorithm, we can build a P-invariant cylindrical algebraic decomposition Δ of $\Re^2$, a base decomposition Δ' of Δ, and a cylindrical sample $\sigma : \Delta \rightarrow \Re^2$ of Δ. The above given sentence is true if and only if: $\forall c' \in \Delta'$, there exists a cell $c \in \Delta$ contained in the cylinder over c' such that $\sigma(c) = (x_c, y_c)$ and $\phi(x_c, y_c)$ holds.

[7] The algorithm is also polynomial in the magnitude of coefficients in $\mathcal{P}$.
[8] The number of cells has no dependence on the magnitude of coefficients.

- **Quantifier elimination:** The problem is to transform a given Tarski sentence with quantifiers into a quantifier-free polynomial expression. For instance consider the sentence $(\exists y)\phi(x,y)$, where ϕ is a polynomial expression over $\Re^2$. Variable x is free, so that the sentence defines a semi-algebraic subset of $\Re$. We wish to define this subset by a quantifier-free polynomial expression over $\Re$. Let Δ, Δ', and σ be as above. We denote ψ_c the quantifier-free polynomial expression defining cell c, for any cell c in Δ or Δ'. Let $\Psi(x)$ be the disjunction of all the $\psi_{c'}$, where c' is the base cell of at least one $c \in \Delta$ such that $\sigma(c) = (x_c, y_c)$ and $\phi(x_c, y_c)$ holds. The two Tarski sentences $(\exists y)\phi(x,y)$ and $\Psi(x)$ define the same semi-algebraic subset of $\Re$.

5.4 General Approach for Path Planning

We now show how the Collins result can be applied to path planning. We assume that the moving object $\mathcal{A}$ and the obstacles $\mathcal{B}_i$ are represented as semi-algebraic subsets of $\Re^k$. Let the defining formula of $\mathcal{A}$ and $\mathcal{B}_i$ be $\Lambda_{\mathcal{A}}(a)$ and $\Lambda_{\mathcal{B}_i}(b)$, respectively, where a and $b \in \Re^k$. As noted in Subsection 5.1, the C-obstacle $\mathcal{CB}_i$ corresponding to the actual obstacle $\mathcal{B}_i$ in the workspace can be represented by a Tarski sentence, and:

$$\bigcup_i \mathcal{CB}_i = \{\mathbf{x} \in \Re^N \;/\; \exists a \in \Re^k \; \exists b \in \Re^k : \; \Lambda_{\mathcal{A}}(a) \wedge (\bigvee_i \Lambda_{\mathcal{B}_i}(b)) \wedge \Upsilon(\mathbf{x}, a, b) \wedge \Gamma(\mathbf{x})\}.$$

We can use the Collins algorithm to generate a quantifier-free polynomial expression over $\Re^N$, $\Psi(\mathbf{x}) = \bigvee \psi_c(\mathbf{x})$, defining $\mathcal{C}_{free}$, together with a $\mathcal{P}$-invariant cylindrical algebraic decomposition Δ of $\Re^N$ ($\mathcal{P}$ is the set of polynomials in Ψ), the successive base decompositions of Δ, and a cylindrical sample σ of Δ. This computation can be done in doubly exponential time in the dimension N. Then, for each cell $c \in \Delta$, we can check if it is part of $\mathcal{C}_{free}$ by testing if $\sigma(c)$ satisfies Ψ.

In order to construct a path from an initial configuration $\mathbf{c}_1$ represented by $\mathbf{x}_1$ to a final configuration $\mathbf{c}_2$ represented by $\mathbf{x}_2$, we have to find a sequence $\{c_1, c_2, ..., c_p\}$ of cells of Δ, such that $\mathbf{x}_1 \in c_1$, $\mathbf{x}_2 \in c_p$, and c_i is adjacent to c_{i+1} for all $i \in [1, p-1]$. By evaluating the polynomial expression defining each cell in $\mathcal{C}_{free}$, with $\mathbf{x} = \mathbf{x}_1$ and $\mathbf{x} = \mathbf{x}_2$, we can identify the first and last cells of the sequence. Constructing the rest of the sequence requires to build and search the connectivity graph CG representing the adjacency relation on the set of cells partitioning $\mathcal{C}_{free}$. The adjacency condition for two cells c_1 and c_2 is $(cl(c_1) \cap c_2) \cup (c_1 \cap cl(c_2)) \neq \emptyset$, which is equivalent to $(cl(c_1) \cap c_2 \neq \emptyset) \vee (c_1 \cap cl(c_2) \neq \emptyset)$. Any of the two conditions in the later disjunction can be expressed as a Tarski sentence. For instance, $c_1 \cap cl(c_2) \neq \emptyset$ holds if and only if the following Tarski sentence is satisfied [2]:

$$(\exists \mathbf{x})(\forall \varepsilon)(\exists \mathbf{y})[(\varepsilon > 0) \wedge \phi_{c_1}(\mathbf{x}) \Rightarrow (\mathit{distance}(\mathbf{x}, \mathbf{y}) < \varepsilon) \wedge \phi_{c_2}(\mathbf{y})].$$

(Read: It exists a point $\mathbf{x}$ in c_1 such that for every $\varepsilon > 0$, there exists a point $\mathbf{y}$ in c_2 distant of $\mathbf{x}$ by less than ε.)

Thus, testing adjacency between two cells and consequently building the connectivity graph CG are decidable problems that can be solved by applying the Collins algorithm to the above Tarski sentence.

As mentioned in Subsection 5.2 the time complexity for generating the cell decomposition of $\mathcal{C}_{free}$ is polynomial in the number of polynomials in $\mathcal{P}$ and in their maximum degree, with a double exponential dependence on N. For a rigid object N is fixed. Therefore, the decomposition of $\mathcal{C}_{free}$ is generated in polynomial time. The number of produced cells is of the same order, so that the connectivity graph CG can be built in polynomial time if adjacency of two cells can be checked in polynomial time. The number of polynomials in the expression ψ_c defining each cell in the decomposition of $\mathcal{C}_{free}$ and their maximum degree are polynomial since the algorithm that constructs these expressions is itself polynomial. Therefore, since the number of variables appearing in the Tarski sentences representing adjacency between two cells is $2N+1$, hence fixed, the construction of the connectivity graph CG is polynomial. However, it has a multiple exponential dependence on N.

A more efficient test of cell adjacency than the above one is proposed in [48]. We will only sketch it here. The improved test requires that $\mathcal{C}$ be represented[9] as a Euclidean space $\Re^m$, where m is the dimension of $\mathcal{C}$, and that the cylindrical decomposition Δ of $\Re^m$ be 'well-based' (see [48]), a concept which recursively applies to the base decomposition of Δ. Since a well-based $\mathcal{P}$-invariant cylindrical algebraic decomposition of an Euclidean space may not exist, without rotating the coordinate axes, this leads to some additional technical complications, which we will not present here. Well-basedness implies that the closure of any cell $c \in \Delta$ is a union of cells in Δ. In addition, we can assume that the initial and final configurations of the robot are contained in cells of the same dimension as $\mathcal{C}$, i.e. m, since the other cells form a measure zero subset of $\mathcal{C}$ (if the assumption is not satisfied at one configuration, an arbitrarily small shift of the configuration will achieve it). Then it can be proven that, without losing completeness, path finding can be confined to those cells in Δ that have dimension m or dimension $m-1$ (i.e., codimension 1), where m is the dimension of $\mathcal{C}$. Thus, adjacency has only to be tested between a cell of dimension m and a cell of dimension $m-1$, which can be done rather efficiently. If two such cells are adjacent, the second is contained in the closure of the first. Using this test, Schwartz and Sharir describe a general path-planning algorithm whose time complexity is polynomial in the number of polynomials in $\mathcal{P}$ and in their maximal degree, with a double dependence on m.

Consider that CG has been constructed and that a sequence of adjacent cells $\{c_1, ..., c_p\}$ has been found by searching it. We now want to generate a path τ from the initial to the goal configuration, which completely lies in the sequence of cells (recall that each cell is a connected set). We assume that $\mathcal{C}$ has been represented as a Euclidean space $\Re^m$ (see the above paragraph) and that the decomposition Δ is well-based, so that $c_1, ..., c_p$ are alternatively m-dimensional and $(m-1)$-dimensional cells (odd indices correspond to m-dimensional cells and even indices correspond to $(m-1)$-dimensional cells). Without loss of completeness, we can construct τ as the concatenation of $(p+1)/2$ sub-paths connecting $\mathbf{x}_1$ to $\sigma(c_2)$, $\sigma(c_2)$ to $\sigma(c_4)$, ..., $\sigma(c_{p-3})$ to $\sigma(c_{p-1})$, and $\sigma(c_{p-1})$ to $\mathbf{x}_2$. Let us consider that the projection τ' of τ has already been generated within c'_1, ..., c'_p,

[9] If $\mathcal{C}$ is of the form $\Re^k \times S^1$, this requirement leads to consider two distinct copies of $\Re^m$ ($m = k+1$). If $\mathcal{C}$ is of the form $\Re^3 \times SO(3)$ ($m = 6$), it leads to remove one singular orientation, but since the corresponding set of points in $\mathcal{C}$ has a dimension less or equal to $m-2$, i.e. a codimension at least 2, omission of these points does not affect the connectivity of $\mathcal{C}_{free}$.

the base cells of c_1, ..., c_p. Each sub-path in τ' can be lifted in c_1, ..., c_p as follows. Assume i to be odd. The x_m coordinate along the sub-path of τ in any cell c_i can be obtained by linearly interpolating between the values of x_m at the beginning and the end of the sub-path, using the proportion of distance between the boundaries of c_i in the $+x_m$ and $-x_m$ directions and by extending the interpolation to the closure of c_i, which includes c_{i+1}. Two cases have to be distinguished in order to construct the interpolation equation: (1) the two cells c_i and c_{i+1} have the same base cell (i.e., $c'_i = c'_{i+1}$), and (2) the two cells c_i and c_{i+1} have adjacent base cells ($c'_i \neq c'_{i+1}$). In both cases, if c_i is a semi-infinite cell, an artificial boundary has to be generated by adding (resp. substracting) a fixed appropriate amount to the cell boundary below (resp. above) the sub-path in order to make the interpolation possible. The projected path τ' can be generated in the same way by lifting in Δ' a path τ'' generated in the base decomposition of Δ'. Thus, recursively, the problem is reduced to the generation of a path in a base decomposition in $\Re$, which is trivial. The constructed free path is piecewise of class C^2.

Therefore:

THEOREM 4 (Schwartz and Sharir): *The problem of planning a free path of a semi-algebraic rigid object $\mathcal{A}$ among semi-algebraic fixed rigid obstacles $\mathcal{B}_i$ can be solved in time polynomial in the number of polynomials defining the objects in the workspace and in the maximal degree of these polynomials.*

So, if both $\mathcal{A}$ and the obstacles are polygonal (resp. polyhedral) objects in a two-dimensional (resp. three-dimensional) workspace, the time complexity of the above method is polynomial in the number of edges (resp. faces) of the objects.

The above general approach to path finding requires performing exact computations with algebraic numbers. This is a delicate issue not treated here. The paper by Schwartz and Sharir [47] provides many details on exact computation with algebraic numbers. It also analyzes the effect of the growth in size of the integers involved in the computations on the total time complexity of the algorithm. It turns out that the algorithm still requires time polynomial in the number of polynomials defining the object and in the maximal degree of these polynomials. However, it is also polynomial in the size of the coefficients in the polynomials.

The Collins decomposition generates many cells allowing its application to the general problem of deciding Tarski sentences. It is probably too general for path finding. The more specific (and more elaborate) method presented in Section 4, although based on the same general idea, generates fewer cells in the three-dimensional configuration space of the segment robot. Each of these cells turns out to be a union of the Collins cells which would have been produced by the above method[10]. This suggests that decompositions coarser than the Collins one may still be appropriate for path planning and that general algorithms of less time complexity than the algorithm presented above exist. Such an approach has been explored by Canny [10], resulting in an algorithm which is simply exponential in the number of degrees of freedom.

[10]The trapezoidal decomposition presented in Section 3 also produces cells that are unions of Collins cells.

6 Approximate Decomposition: Translational Case

In this section and the next, we describe another type of path finding decomposition methods based on approximate decomposition of configuration space. We will only consider the cases of polygonal and polyhedral workspaces.

An approximate decomposition method typically consists of decomposing $\mathcal{C}$ into a collection of disjoint open rectangloids. Two cells are adjacent if their closures share a portion of an edge. Each cell is compared to the C-constraints defining the C-obstacles and is labelled EMPTY, FULL, or MIXED, depending on whether it lies completely inside $\mathcal{C}_{free}$, completely outside, or partly inside and outside. The problem of finding a free path is thus transformed into the problem of finding a sequence of adjacent EMPTY cells. If no such sequence is found, MIXED cells may be decomposed further if there is some chance that a free path traverse them. This leads to an interesting hierarchical decomposition of configuration space, which allows to adapt the size of the cells to the occupancy of $\mathcal{C}$ by C-obstacles.

Unlike exact decomposition methods, the methods described below are not based on a preliminary analysis of critical surves or surfaces. Hence, the boundaries of the cells which they produce have no 'physical' meaning. Instead, the rationale of approximate decomposition methods is the simplicity of the shape of the cells in order to allow repetition of elementary computations.

As with exact decomposition, we begin our presentation of approximate decomposition methods with the simple case where the moving object $\mathcal{A}$ is a polygon only allowed to translate among fixed polygonal obstacles. Each C-obstacle $\mathcal{CB}_i$, $i = 1$ to q, is a convex polygon that can be represented as a conjunction of C-constraints (see Section 2):

$$(x, y) \in \mathcal{CB}_i \Leftrightarrow \bigwedge_k (a_{ik}x + b_{ik}y + c_{ik} \leq 0).$$

As in Section 3, we assume that $\mathcal{C}_{free} = int(R) - \bigcup_i \mathcal{CB}_i$, where R is a given rectangle whose faces are oriented according to the x and y-axes embedded in $\mathcal{C}$. The method described below is based on an approach first introduced by Brooks and Lozano-Pérez [31,7,8].

Consider an initial configuration $\mathbf{c}_1$ and a final configuration $\mathbf{c}_2$, both in free space. We want to generate a free path from $\mathbf{c}_1$ to $\mathbf{c}_2$. The overall path finding algorithm proceeds in a hierarchical fashion as follows:

1. Generate an initial decomposition of R into disjoint open rectangular cells whose closed union is equal to $cl(R)$, and label every cell by EMPTY, FULL, or MIXED.

2. For $i = 1, 2$ do: while the cell c containing $\mathbf{c}_i$ is MIXED and larger than a preset size, decompose c into smaller cells and label the new cells; at the end of the iteration, if c is still not EMPTY, return failure; otherwise, if $i = 1$, c is called the *initial cell* and if $i = 2$, it is called the *final cell.*

3. Build the connectivity graph CG_{EMPTY} representing the adjacency relation between EMPTY cells and search it for a path from the initial cell to the final cell.

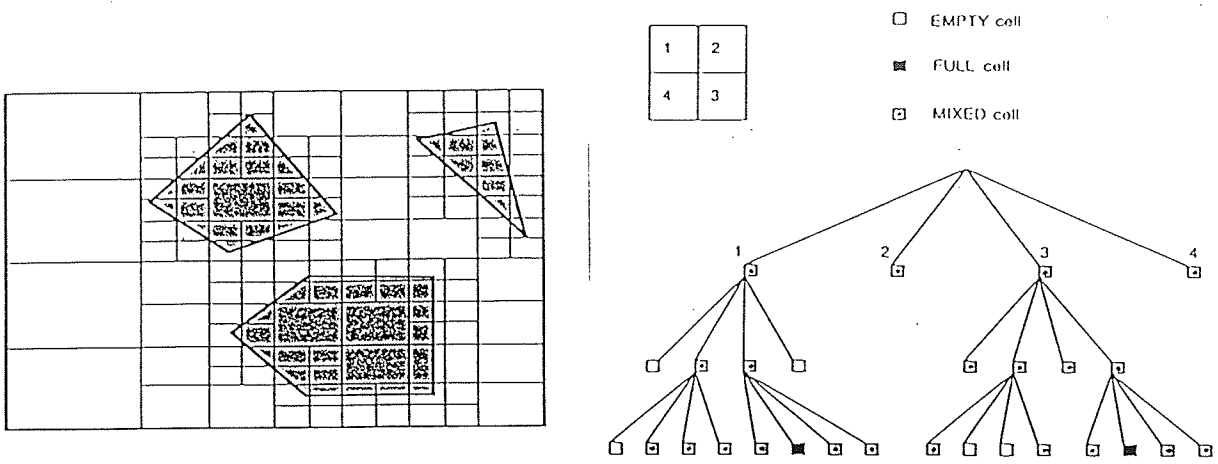

Figure 8: **Example of a Quadtree**

4. If the search of Step 3 terminates successfully, return the corresponding sequence of cells.

5. If the search of Step 3 terminates unsuccessfully, consider all EMPTY cells and those MIXED cells which are larger than a preset size. Build the connectivity graph $CG_{\texttt{MIXED}}$ representing the adjacency relation between these cells, and search it for a path from the initial cell to the final cell.

6. If the search of Step 5 terminates successfully, decompose the MIXED cells along the path into smaller cells, label these new cells, and go back to Step 3.

7. If the search of Step 5 terminates unsuccessfully, return failure.

If successful, this algorithm returns a sequence of rectangular cells. A free path can easily be produced by linking $\mathbf{c}_1$ to $\mathbf{c}_2$ by a succession of straight segments passing through the mid-points of the edges separating consecutive cells.

The hierarchical decomposition of R can be done by recursively dividing R and the successively obtained MIXED cells into four identical rectangles. Such a decomposition produces a quadtree [46,20,34,4].

DEFINITION 15: *A* **quadtree** *is a tree of degree 4. Each node represents a rectangle, and is labelled* EMPTY, FULL, *or* MIXED. *A node may have successors only if it is labelled* MIXED. *The four successors of a* MIXED *node representing a rectangle r represent four rectangles of the same size obtained by decomposing r.*

Figure 8 shows the quadtree decomposition of a simple configuration space and a portion of the corresponding quadtree graph. The depth of a node in the quadtree determines the size of the corresponding cell. The height of the quadtree determines the resolution of the decomposition. During the construction of the quadtree, it is easy to maintain a representation of the adjacency relation between the leaves of the quadtree.

In the above algorithm we must specify the resolution σ_{init} of the initial decomposition. All MIXED cells whose depth is less or equal to σ_{init} are decomposed. Later on only those MIXED cells which are located on a potential path are decomposed further. A maximal resolution σ_{max} must also be specified. No MIXED cell whose depth is equal to σ_{max} gets decomposed further.

In order to label cells, one can associate with every MIXED cell a logical expression, called **C-sentence**, specifying the C-obstacles intersecting the cell. A C-sentence is of the form [8]:

$$\bigvee_i \bigwedge_k e_{ik}$$

where e_{ik} is a C-constraint of the form $a_{ik}x + b_{ik}y + c_{ik} \leq 0$. Each conjunct $\wedge_k e_{ik}$ corresponds to a particular C-obstacle $\mathcal{CB}_i$. When a cell c is decomposed into smaller cells the C-sentence associated with c is used to label each new MIXED cell. During the labelling process (described below), it may get simplified and the simplified C-sentence is associated to the new cell. The rectangle R may be regarded as the unique successor of a fictitious cell whose C-sentence is the disjunction of the expressions describing the C-obstacles.

The algorithm for labelling a new cell and simplifying the C-sentence is shown below [8]. Its input parameters are c and S. c designates the new cell. S designates a C-sentence, whose initial value is the C-sentence associated with the parent of c. The cell c is said to be **inside** a C-constraint e_{ik} if all points (x, y) in c satisfy e_{ik}; it is **outside** e_{ik} if all points (x, y) in c contradict e_{ik}; it is **cut** by the C-constraint if it is neither inside, nor outside e_{ik}. The computation for testing whether a cell c is inside a C-constraint e_{ik}, or outside, or neither is quite simple:
- c is inside e_{ik} if its four vertices verify: $a_{ik}x + b_{ik}y + c_{ik} \leq 0$.
- c is outside e_{ik} if its four vertices verify: $a_{ik}x + b_{ik}y + c_{ik} \geq 0$.
- c is cut by e_{ik} otherwise.

```
procedure LABEL(c,S)
for each conjunct Λ in S do
    for each C-constraint e in Λ do
        if c is inside e
            then remove e from Λ;
            else if c is outside e
                then remove Λ from S;
                    exit from loop;
            endif;
        endif;
    enddo;
    if Λ is empty
        then label c with FULL;
            exit from procedure;
    endif;
enddo;
if S is empty
```

```
        then  label c with EMPTY;
        else  label c with MIXED;
              associate S with c;
    endif;
    endprocedure;
```

Attachment of simplified C-sentences to MIXED cells may considerably accelerate the labelling process, since the number of C-constraints to be considered usually decreases rapidly with the size of the cell.

Notice, however, that the procedure LABEL is somewhat conservative. Indeed, it treats each line $a_{ik}x + b_{ik}y + c_{ik} = 0$, as an infinite line, independently of the other lines, and therefore some cells may get labelled MIXED, while they do not actually intersect any C-obstacle. However, such an 'error', which never leads to generate incorrect paths, is ultimately corrected, when the cell gets decomposed further.

Unlike exact decomposition methods, the above method is not complete since it cannot be guaranteed to find a free path, whenever such a path exists. This is due to the fact that it approximates free space as a collection of cells of predefined shape. Thus, at any resolution, the collection of EMPTY cells is a conservative approximation of $\mathcal{C}_{free}$, i.e. the closed union of the EMPTY cells is included in $\mathcal{C}_{free}$. However, at the eventual expense of generating a very large number of cells, the representation can be as close as we want from the exact free space by setting σ_{max} appropriately. The above method guarantees that every MIXED cell which may contain a portion of a free path and whose depth is less than σ_{max} will be decomposed into smaller cells if a free path is not found before. Thus, the overall path finding method is said to be **resolution-complete**.

The above approximate decomposition method can easily be extended to the three-dimensional case. The translating object $\mathcal{A}$ and the obstacles are polyhedra. The C-obstacles $\mathcal{CB}_i$ are convex polyhedra. We assume that $\mathcal{C}_{free} = int(R) - \bigcup_i \mathcal{CB}_i$, where R is a parallelepiped whose faces are oriented according to the x, y, and z-axes embedded in $\mathcal{C}$. The path finding method remains basically the same. Only the representation of the decomposition of R is changed into a tree of degree 8, called an **octree** [20,34,4]. An octree is obtained by recursively decomposing a parallelepiped into 8 identical parallelepipeds. Comparing a cell c of an octree to a C-constraint, now of the form $a_{ik}x + b_{ik}y + c_{ik}z + d_{ik} \leq 0$, requires checking the 8 vertices against the constraint.

Notice that, although the extension to the three-dimensional case is conceptually straightforward, the number of cells in an octree augments much quicker with the height of the tree than the number of cells in a quadtree.

7 Approximate Decomposition with Rotation

Extending the previous method to the case of a moving polygonal/polyhedral object $\mathcal{A}$ allowed to both translate and rotate among polygonal/polyhedral obstacles is not difficult, conceptually. However, C-constraints are more complex than in the translational case. Indeed, each of them is applicable only for some orientations of $\mathcal{A}$. Moreover, C-constraints determine curved C-surfaces.

In Subsection 7.1, we directly extend the hierarchical cell decomposition method described above to the rotational case. In Subsections 7.2 and 7.3, we present variants of the approximate decomposition approach to path finding.

We only treat the case of a two-dimensional workspace. Extension to the three-dimensional case is usually not difficult, but may result into impractical algorithms. Indeed, the number of cells to be generated is roughly exponential in the dimension of $\mathcal{C}$. One practical way to deal with a three-dimensional object allowed to both translate and rotate is to limit the number degrees of freedom to be considered at every position. For example, one could allow the object to rotate only in some intermediate regions. There is no general method, however, to determine these intermediate regions.

Hence, throughout this section $\mathcal{C} = \Re^2 \times S^1$. For practical reasons, we will represent it as $\Re^2 \times [0, 2\pi]$. We will also assume that the range of free positions of $\mathcal{A}$ is bounded by a rectangle $R \subset \Re^2$. Thus, the representation of $\mathcal{C}_{free}$ is bounded by the $R \times [0, 2\pi]$ parallelepiped.

7.1 Hierarchical Cell Decomposition

The approximate cell decomposition of the previous section can readily be extended to the case of an object allowed to both translate and rotate in a two-dimensional workspace. Indeed, we can represent the three-dimensional parallellepiped $R \times [0, 2\pi]$ by an octree and apply the same path finding and cell labelling algorithms. However, there are some differences related to how cells are to be compared to C-constraints.

Each C-obstacle $\mathcal{CB}_i$ now determines in $R \times [0, 2\pi]$ a volume bounded by patches of ruled surfaces. We have:

$$(x, y, \theta) \in \mathcal{CB}_i \Leftrightarrow \bigwedge_k [\theta \in \delta_{ik} \Rightarrow (a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) \leq 0)]$$

where δ_{ik} is the angular interval in which the C-constraint in the right-hand side applies (see Section 2). Thus, the C-sentence associated with each `MIXED` cell still has the form:

$$\bigvee_i \bigwedge_k e_{ik}$$

but e_{ik} now denotes a conditional C-constraint of the form $\theta \in \delta_{ik} \Rightarrow (a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) \leq 0)$.

Let $c = (x_c, x'_c) \times (y_c, y'_c) \times (\theta_c, \theta'_c)$ be a parallelepipedic cell. c is **inside** e_{ik} if and only if:
- $(\theta_c, \theta'_c) \subset \delta_{ik}$ (i.e., the constraint applies in the whole cell),
- and $\forall (x, y, \theta) \in c$: $a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) \leq 0$.

It is **outside** e_{ik} if and only if:
- $(\theta_c, \theta'_c) \cap \delta_{ik} = \emptyset$ (i.e., the constraint applies nowhere in the cell),
- or $\forall (x, y, \theta) \in (x_c, x'_c) \times (y_c, y'_c) \times [(\theta_c, \theta'_c) \cap \delta_{ik}]$: $a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) > 0$.

This means that we have to compare a parallelepiped $(x_c, x'_c) \times (y_c, y'_c) \times (\theta_1, \theta_2) \subseteq c$, where $(\theta_1, \theta_2) = (\theta_c, \theta'_c) \cap \delta_{ik}$, to a constraint $a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) \leq 0$.

Consider the projection on the (x, y)-plane of the portion of the C-surface $a_{ik}(\theta)x + b_{ik}(\theta)y + c_{ik}(\theta) = 0$, which is comprised within the interval (θ_1, θ_2). Let (x, y) be a point outside this projection; the segment connecting (x, y, θ_1) to (x, y, θ_2) is completely inside or completely outside the constraints $a_{ik}(\theta)x + b_{ik}(\theta) + c_{ik}(\theta) \leq 0$. Inversely, let (x, y) be a point inside the projection of the surface; the segment connecting (x, y, θ_1) to (x, y, θ_2) crosses the C-surface, so that part of it lies inside the constraint and part of it lies outside. Thus, we can compare a cell to a constraint by first projecting the (θ_1, θ_2) slice of the C-surface on the (x, y)-plane, and then by comparing a rectangle to this projection. The computations necessary for this comparaison are rather straightforward. We refer the reader to [8] for a detailed description.

Extending the above method to a polyhedron allowed to both translate and rotate in three-dimensional workspace requires to approximate a six-dimensional configuration space by a 2^6-tree (a quadtree is a 2^2-tree, an octree is a 2^3-tree). This is very likely to be impractical. Indeed, in such a tree, the number of cells at depth 1 is 64; at depth 2, it is 4096, at depth 3, it is 262144, etc. And at depth 3, each dimension is divided in only 8 intevals; for the angular axis, these intervals have an amplitude of $\pi/4$. Except in very favorable situations, such a coarse resolution prevents the method from finding a free path.

7.2 Orientation Slicing

This method, due to Lozano-Pérez [31,32], is based on a decomposition of the range $[0, 2\pi]$ of possible orientations of $\mathcal{A}$ into a finite number of intervals. The basic idea is to consider the areas swept out by $\mathcal{A}$ in all these intervals (see Figure 9) and to treat these areas as new objects only allowed to translate. A path is generated as a sequence of sub-paths, each for one of these objects, such that any two consecutive sub-paths corresponds to adjacent angular intervals.

The overall algorithm is the following:

1. Decompose the range of orientations of $\mathcal{A}$, $[0, 2\pi]$, into a finite numbe of consecutive closed intervals $[\theta_k, \theta'_k]$, $k = 1$ to p, such that $\theta_1 = 0$, $\theta'_k = \theta_{k+1}$ (for all $k \in [1, p-1]$), and $\theta'_p = 2\pi$ (θ_k is allowed to be equal to θ'_k).

2. For every $k \in [1, p]$, approximate the area swept out by $\mathcal{A}$ when its position (x, y) is fixed and when its orientation θ varies from θ_k to θ'_k by a bounding convex polygon denoted $\mathcal{SA}_k$. Attach a frame $\mathcal{F}_{\mathcal{SA}_k}$ to $\mathcal{SA}_k$, which coincides with $\mathcal{F}_{\mathcal{A}}$ when $\mathcal{A}$'s orientation is θ.

3. For every $k \in [1, p]$, treat the polygon $\mathcal{SA}_k$ as a moving object only allowed to translate. The configuration space of $\mathcal{SA}_k$ is $\mathcal{C}^k = \Re^2$. Compute the C-obstacles $\mathcal{CB}_{ki}$ (convex polygons) in $\mathcal{C}^k$.

4. For every $k \in [1, p]$, decompose $\mathcal{C}^k_{free}$ into convex polygons, called **cells**, and construct the connectivity graph CG_k representing the adjacency relation between these cells. If θ_1 (resp. θ_2) $\in [\theta_k, \theta'_k]$, then label the cell containing (x_1, y_1) (resp. (x_2, y_2)) as the *initial* (resp. *final*) cell.

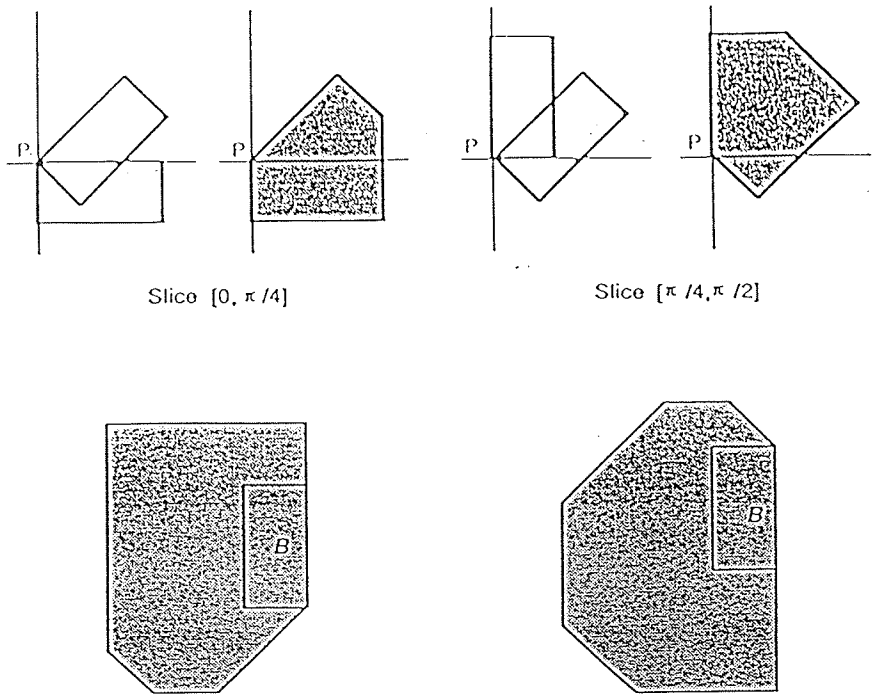

Figure 9: **Illustration of the Orientation Slicing Method**

5. Combine the graphs CG_k into a new connectivity graph CG by linking any two nodes $X_1 \in CG_{k_1}$ and $X_2 \in CG_{k_2}$ $(k_1 \neq k_2)$, whenever these two nodes represent overlapping cells when $\mathcal{C}^{k_1}$ and $\mathcal{C}^{k_2}$ are superposed, and $\theta'_{k_1} = \theta_{k_2}$ (modulo 2π).

6. Search CG for a path linking the initial cell to the final cell. If the search terminates successfully, then return the sequence of cells along the path; otherwise, return failure.

We can regard $\mathcal{CB}_{k,i}$ as a bounding polygonal approximation of the projection of the slice through $\mathcal{CB}_i$ comprised between θ_k and θ'_k. Thus $\mathcal{CB}_{k,i} \times [\theta_k, \theta'_k]$ is a prism bounding $\mathcal{CB}_i$ within the $[\theta_k, \theta'_k]$ slice. Any two cells in CG_k can be adjacent only by their vertical faces. Two cells in CG, which are not contained in the same slice, can be adjacent if they belong to two adjacent slices and if the intersection of the top face of one with the bottom face of the other is a two-dimensional region.

The above method may fail to find a free path if the intervals $[\theta_k, \theta'_k]$ are too large. In case of failure, it is possible to refine these intervals and to run the method again. However, unlike with the method of Subsection 7.1, there is no obvious information that can be used to select which intervals has to be refined. In addition, there is no simple way to capitalize on the computations performed for one interval in order to simplify future computations when this interval gets decomposed. Therefore, refining all intervals may result in excessive computing time.

Extension of the above method to a polyhedral object moving in a three-dimensional workspace is simple in principle, but impractical in most cases.

7.3 Position Partitioning

This method, whose principle is due to Germain [19], has several similarities with the exact decomposition method described in Section 4. However, it does not make use of the critical curves defined there.

Assume, without loss of generality, that the origin of $\mathcal{F}_\mathcal{A}$ is in $\mathcal{A}$. Then, the set of possibly admissible position of $\mathcal{A}$ (i.e., the set of positions which admit free orientations of $\mathcal{A}$) is the empty subset of the workspace, $\mathcal{E} = int(R) - \bigcup \mathcal{B}_i$. The position partitioning method consists of decomposing this set into a finite collection of open convex polygons R_k, $k = 1$ to p, such that $\bigcup_k cl(R_k) \subseteq cl(\mathcal{E})$ and $\forall k \neq k'\ R_k \cap R_{k'} = \emptyset$. For each polygon R_k, we compute the range of orientations of $\mathcal{A}$ which are free for all positions of $\mathcal{A}$ in R_k. Let us denote $F(R_k)$ this range. Then, we approximate $\mathcal{C}_{free}$ by prismatic cells of the form $R_k \times I_{kl}$, where $I_{kl} = (\theta_{kl}, \theta'_{kl})$ is a maximal connected interval in $F(R_k)$. Finally, we construct the connectivity graph among these cells and we search the graph. All these computations are described in more detail below.

The decomposition of $\mathcal{E}$ into convex polygons may be an exact decomposition – then, $\bigcup_k cl(R_k) = cl(\mathcal{E})$ – or it may be an approximate decomposition – then, $\bigcup_k cl(R_k) \subset cl(\mathcal{E})$. Approximate decomposition makes it possible using only very simple polygons, e.g. rectangles.

The computation of $F(R_k)$ is the core of the position partitioning method. Below, we give two lemmas which reduce this computation to the computation of the set of valid orientations of $\mathcal{A}$ at a single position among transformed obstacles.

The set of valid orientations of $\mathcal{A}$ at (x, y) is $V(x, y) = \{\theta \ / \ (x, y, \theta) \in \mathcal{C}_{valid}\}$. For any $R_k \subset \mathcal{E}$, we define:

$$V(R_k) = \bigcap_{(x,y)\in R_k} V(x, y).$$

The first lemma states that the set of free orientations of $\mathcal{A}$ over the open region R_k is equal to the set of valid orientations of $\mathcal{A}$ over the closure of R_k. For more clarity, we denote $\overline{R_k}$ the closure of R_k. Thus: $\overline{R_k} = cl(R_k)$.

LEMMA 4: $F(R_k) = V(\overline{R_k})$.

Let $V_\mathcal{B}(x, y) = \{\theta \ / \ int(\mathcal{A}(x, y, \theta)) \cap \mathcal{B} = \emptyset\}$, where $\mathcal{B}$ is a closed region of $\Re^2$. Thus, the set of valid orientations of $\mathcal{A}$ at position (x, y) among the obstacles $\mathcal{B}_i$ is $V(x, y) = V_{\bigcup_i \mathcal{B}_i}(x, y)$. The following lemma reduces the computation of $V(\overline{R_k})$ to the computation of $V_{\mathcal{B}'}(x_k, y_k)$ at a single point (x_k, y_k) defined with respect to R_k, for a certain region $\mathcal{B}'$ obtained by 'growing' $\mathcal{B} = \bigcup_i \mathcal{B}_i$ by $\ominus\overline{R_k}$. This is illustrated by Figure 10 in the typical case where (x_k, y_k) is a vertex of R_k.

LEMMA 5: *Let* $\mathcal{B} = \bigcup_i \mathcal{B}_i$. *We have*

$$V(\overline{R_k}) = V_{\mathcal{B}\ominus\overline{R_k}(0,0)}(x_k, y_k)$$

where (x_k, y_k) *is a point fixed relatively to* R_k, $\overline{R_k}(0, 0)$ *designates the polygon* $\overline{R_k}$ *obtained by translating the point* (x_k, y_k) *at the origin of the* (x, y)*-plane, and* $\ominus$ *is the*

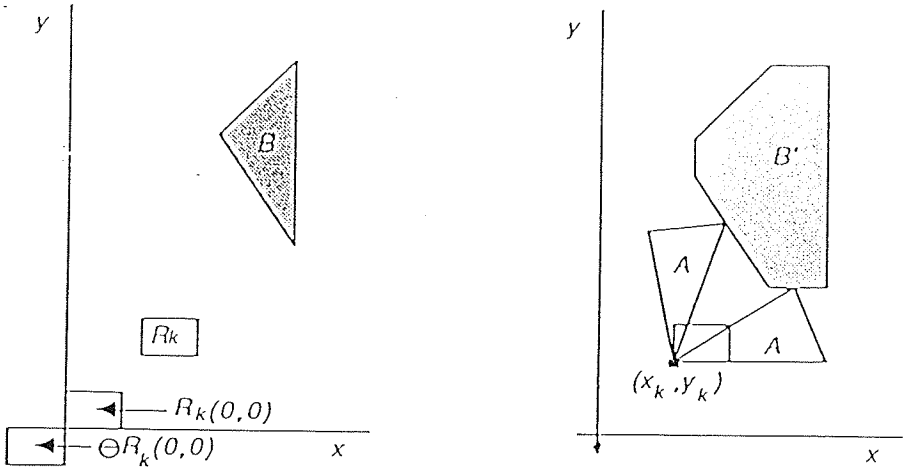

Figure 10: **Growing an Obstacle by a Cell**

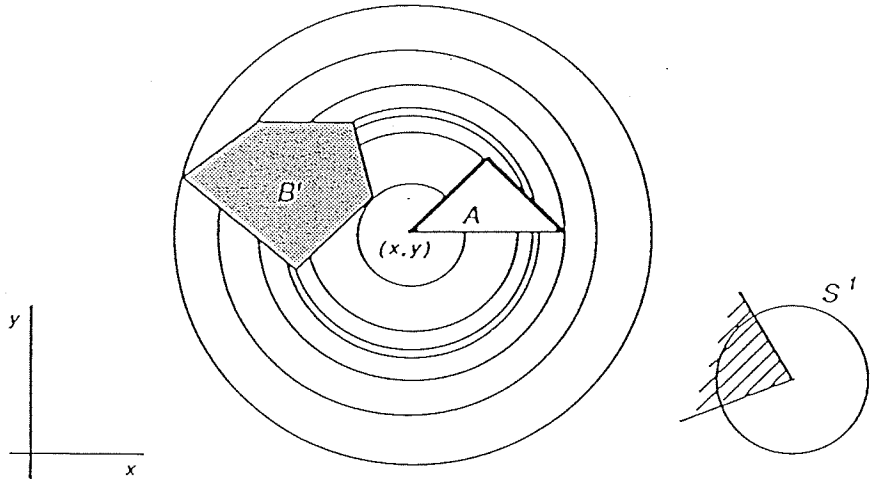

Figure 11: **Computation of $V_{\mathcal{B}'}(x, y)$**

Minkowski operator for set difference.

The region $\mathcal{B} \ominus \overline{R_k}(0,0)$ is a closed polygonal region if $\Re^2$. Therefore, $V_{\mathcal{B} \ominus \overline{R_k}(0,0)}(x_k, y_k)$ (hence $F(R_k)$) is a finite union of closed angular intervals. Some of these intervals may consist of a single orientation. It is easy to compute $V_{\mathcal{B}'}(x, y)$ using a technique inspired from the line-sweep paradigm, after having determined the intersections of the circles centered at (x, y) and passing through the vertices of $\mathcal{A}$ (resp. $\mathcal{B}'$), with the edges of $\mathcal{B}'$ (resp. $\mathcal{A}$) [33], as Figure 11 illustrates.

At this point, we know how to approximate free space as a collection of parallelepipedic cells of the form $R_k \times I_{kl}$. Two cells $R_k \times I_{kl}$ and $R_{k'} \times I_{k'l'}$ are **adjacent** if and only if:
- The boundaries of R_k and $R_{k'}$ share an open portion of an edge.
- $I_{kl} \cap I_{k'l'}$ is an interval of non-null measure.

It is easy to verify that two adjacent cells can be connected by a free path.

Path planning consists of searching the connectivity graph CG representing the adjacency relation between cells.

This method may fail to generate a free path while one exists. In case of failure, one may attempt to refine the decomposition of the empty space $\mathcal{E}$ and run again the method. However, like with the orientation slicing method, no simple information is available to determine which part of the decomposition to refine. Clearly, there is a tradeoff between the resolution of the decomposition and the computing cost.

Extending the position partitioning method to the case of a polyhedron moving in a three-dimensional workspace is delicate. Indeed, the set $V_{\mathcal{B}\ominus\overline{R_i}(0,0)}(x_k, y_k, z_k)$ would be more difficult to characterize and of course to compute.

8 Retraction Method (Principle)

An alternative to the decomposition approach to path planning is the retraction approach. The basic idea is to approximate free space as a net of curves contained in free space and describing the topology of free space. A mapping (retraction) associates a unique point on these curves to any free configuration. Once the net of curves has been generated, path planning is reduced to the combinatorial search problem of finding a path in the net from the retraction of the initial configuration to the retraction of the final configuration.

In this section, we present the theoretical principle of retraction in the simple case of a polygonal object $\mathcal{A}$ translating among polygonal obstacles; in this case, the net of curves is a Voronoi diagram, a well-known construct in Computational geometry [17,23,28]. In the next section, we will present an ad-hoc method based on the retraction idea, applicable to the case of a two-dimensional object translating and rotating among polygonal obstacles. Additional information on the retraction approach can be found in the literature (e.g. [37,38,39,40,10]).

The configuration space of $\mathcal{A}$ is $\mathcal{C} = \Re^2$, and the C-obstacles are (possibly overlapping) convex polygons $\mathcal{CB}_i$. Assume that the range of possible positions of $\mathcal{A}$ is bounded by a polygon R. Thus: $\mathcal{C}_{free} = R - \bigcup_i \mathcal{CB}_i$. The boundary of $\mathcal{C}_{free}$ is represented as a set of vertices and open edges.

DEFINITION 16: *Let $\beta = \partial(\mathcal{C}_{free})$. For any $\mathbf{c} \in \mathcal{C}_{free}$, let:*

$$clearance(\mathbf{c}) = \min_{(x,y)\in\beta} distance(\mathbf{c}, (x, y))$$

and

$$near(\mathbf{c}) = \{(x, y) \;/\; distance(\mathbf{c}, (x, y)) = clearance(\mathbf{c})\}$$

where 'distance' denotes the Euclidean distance in $\Re^2$. The **Voronoi diagram** *of $\mathcal{C}_{free}$ is the set $\{\mathbf{c} \in \mathcal{C}_{free} \;/\; card(near(\mathbf{c})) > 1\}$, where $card(E)$ is the cardinal of set E. It is denoted $Vor(\mathcal{C}_{free})$.*

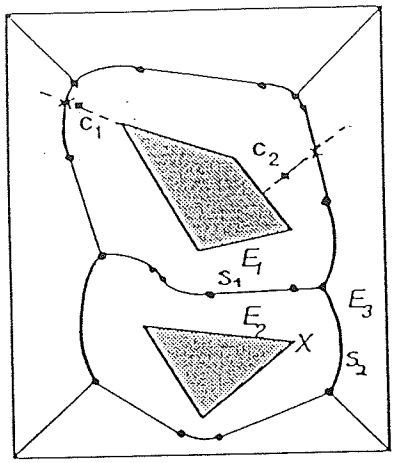

Figure 12: **Example of Voronoi Diagram**

Figure 12 shows a simple Voronoi diagram in a configuration space containing two C-obstacles. One can see that $Vor(\mathcal{C}_{free})$ consists of a finite collection of straight and parabolic curve segments. A straight segment in $Vor(\mathcal{C}_{free})$ is the set of configurations that are closest from the same pair of edges. A parabolic segment is the set of configurations that are closest from the same pair consisting of one edge and a vertex. The pair (edge,edge) or (edge,vertex) of closest elements from any configuration on a straight or parabolic segment determines the equation of the curve supporting the segment. Each straight or parabolic segment is called an **arc** of the Voronoi diagram. Each endpoint of a segment is called an **edge** of the diagram.

Let us now consider a free configuration $\mathbf{c} \notin Vor(\mathcal{C}_{free})$. There exists a unique edge or vertex Z of $\beta = cl(\mathcal{C}_{free})$ which is closer from $\mathbf{c}$ than any other edge or vertex in β. Let $p \in \beta$ be the point such that $clearance(\mathbf{c}) = distance(\mathbf{c}, p)$. If Z is an edge then $p \in Z$; if it is a vertex then $p = Z$. Consider the line L passing through $\mathbf{c}$ and p. If we move along L away from p and beyond $\mathbf{c}$, then $clearance(\mathbf{c})$ increases at maximal rates[11] until it reaches a point in $Vor(\mathcal{C}_{free})$. This point, denoted $Im(\mathbf{c})$, is the **image** of $\mathbf{c}$ on the Voronoi diagram. Passing this point, either $clearance(\mathbf{c})$ drecreases, or the direction of maximum positive increase changes. The function Im can be extended to all $\mathcal{C}_{free}$ by putting $Im(\mathbf{c}) = \mathbf{c}$ for all $\mathbf{c} \in Vor(\mathcal{C}_{free})$. It is easy to verify that the function Im is continuous. In topology a continuous function $f : E \rightarrow F \subset E$, whose restriction to F is the identity is called a *retraction* [21].

Let $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ be a free path between two free configurations $\mathbf{c}_1$ and $\mathbf{c}_2$. Thanks to the continuity of Im, $Im \circ \tau : [0,1] \rightarrow Vor(\mathcal{C}_{free})$ is a path between $Im(\mathbf{c}_1)$ and $Im(\mathbf{c}_2)$. This gives the following theorem, which expresses that the connectivity of $\mathcal{C}_{free}$ and the connectivity of $Vor(\mathcal{C}_{free})$ are 'equivalent':

THEOREM 5: *Let $\mathcal{C}_{free}$ be the interior of a polygonal region with polygonal holes. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be two configurations in $\mathcal{C}_{free}$. There exists a free path from $\mathbf{c}_1$ to $\mathbf{c}_2$ if and*

[11] It is easy to verify that $\nabla clearance(\mathbf{c})$ is a vector supported by L and pointing away from p.

only if there exists a path from $Im(\mathbf{c}_1)$ *to* $Im(\mathbf{c}_2)$ *with* $Vor(\mathcal{C}_{free})$.

The path finding algorithm known as the *retraction method* derives directly from the above theorem. It consists of the following steps:

1. Compute the Voronoi diagram $Vor(\mathcal{C}_{free})$.
2. Compute the points $Im(\mathbf{c}_1)$ and $Im(\mathbf{c}_2)$ and identify the edges of the Voronoi diagram containing these two points.
3. Search $Vor(\mathcal{C}_{free})$ for a sequence of arcs $e_1, ..., e_q$, such that $Im(\mathbf{c}_1) \in e_1$, $Im(\mathbf{c}_2) \in e_2$, and $\forall i \in [1, q-1]$ e_i and e_{i+1} have a common end-point.
4. If the search terminates successfully return $Im(\mathbf{c}_1)$, $Im(\mathbf{c}_2)$, and the sequence of arcs along the path in $Vor(\mathcal{C}_{free})$; otherwise, return failure.

There exist algorithms for computing the Voronoi diagram in $O(n \log^2 n)$ time (see [26,28]) and in $O(n \log n)$ time (see [23]), where n is the number of vertices in $\beta = \partial(\mathcal{C}_{free})$. The number of arcs in the diagram is $O(n)$. At Step 2, for each configuration $\mathbf{c}_i$ $(i = 1, 2)$, one can:

- First, compute both the distance of $\mathbf{c}_i$ to every edge and every vertex of β, and the point p_i of β to which it is closest,

- Second, determine the intersections of the ray drawn from p_i through $\mathbf{c}_i$ with the $Vor(\mathcal{C}_{free})$, and select the intersection which is closest from p_i.

These computations require $O(n)$ time. Finally, the search at Step 3 can be done using the A^* algorithm and takes $O(n)$ time. Thus the overall time complexity of this path planning method is $O(n \log^2 n)$ or $O(n \log n)$. The most expensive step is Step 1, but it depends on $\mathcal{C}_{free}$ only. Subsequent path finding in the same free space between other pairs of configurations can be done omitting Step 1 and takes $O(n)$ time. In addition, there exists linear techniques to update the Voronoi diagram when a few C-obstacles are changed.

9 Freeway Method

The path finding method presented in this section is an ad hoc retraction-like method applicable to a robot moving both in translation and rotation in a two-dimensional workspace among fixed polygonal obstacles. The method is due to Brooks [9], whose original goal was to capture the effects of both translating and rotating an object into a more intuitive representation of free space than configuration space. In fact, since configuration space is just a general concept, the 'new' representation can easily be related to configuration space. Instead, it turns out that the actual contribution of the freeway method is that it makes it possible to build a description of the connectivity of $\mathcal{C}_{free}$, known as the *freeway net*, without having to completely compute C-obstacles. The intuition behind the freeway method is similar to the intuition behind applying retraction to path finding, i.e. always keep the moving object as far away as possible

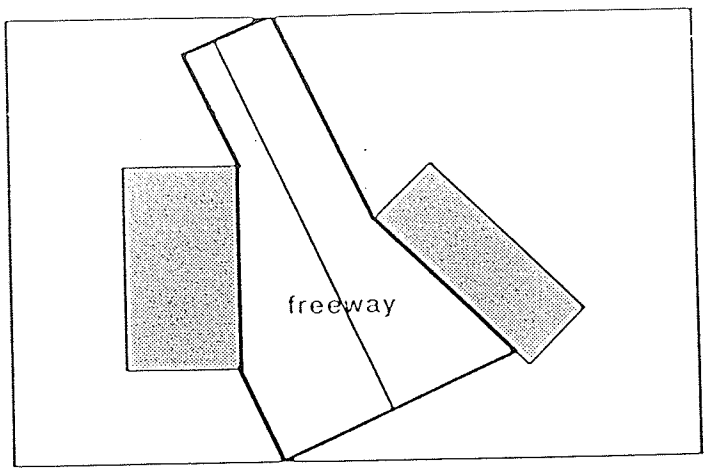

Figure 13: **Example of a Freeway**

from a pair of obstacles. Consequently, the freeway net resembles a Voronoi diagram, but its construction is based on some ad hoc assumptions and it incompletely describes the connectivity of $\mathcal{C}_{free}$, so that the freeway method itself is incomplete. Nevertheless, experiments have shown that it can solve many problems efficiently, when the workspace is not densily occupied by obstacles.

The freeway method consists of extracting geometric figures called *freeways* from the workspace, to connect them into a graph called the freeway net, and to search this graph. A freeway is a straight linear generalized cylinder [5] whose axis, the *spine*, is annotated with a description of the free orientations along the freeway. Figure 13 illustrates the geometry of a freeway between two obstacles in a two-dimensional workspace bounded by a rectangle. Figure 14 shows additional freeways in the same workspace. The moving object $\mathcal{A}$ can be displaced along a freeway (or part of it) if there is a connected set of free orientations of $\mathcal{A}$ along the spine. In addition, whenever two spines intersect, $\mathcal{A}$ can transfer from one freeway to the other if the ranges of free orientations of $\mathcal{A}$ along both spines have non-empty intersection. The freeway net is a representation of the possible motions of $\mathcal{A}$ along spines and between spines. This graph describes (incompletely) the connectivity of $\mathcal{C}_{free}$. A path in this graph determines a class of homotopic free paths of $\mathcal{A}$.

The freeway method successively consists of extracting freeways from the description the obstacles $\mathcal{B}_i$ in the workspace, computing the range of free orientations at some points along the spines of the freeways, constructing the freeway net, and searching this graph for a path. We present these steps in the following subsections. We assume that $\mathcal{A}$ and the $\mathcal{B}_i$'s are modeled as polygonal regions of $\Re^2$ and that the workspace is itself a bounded polygonal region of $\Re^2$. We denote $\mathcal{E}$ the subset of the workspace not occupied by obstacles.

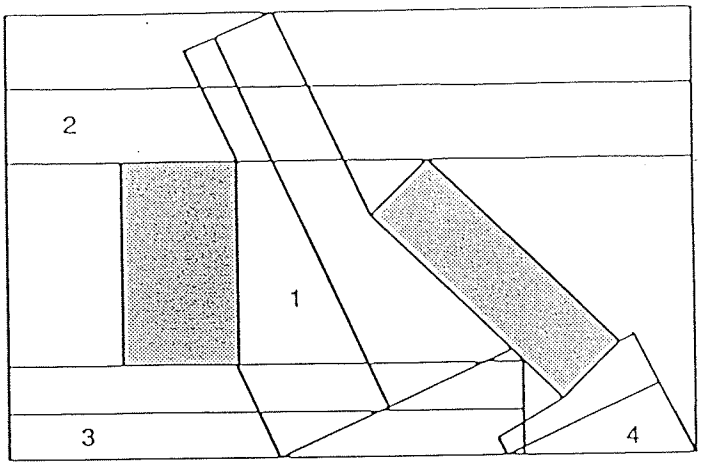

Figure 14: **Overlapping Freeways**

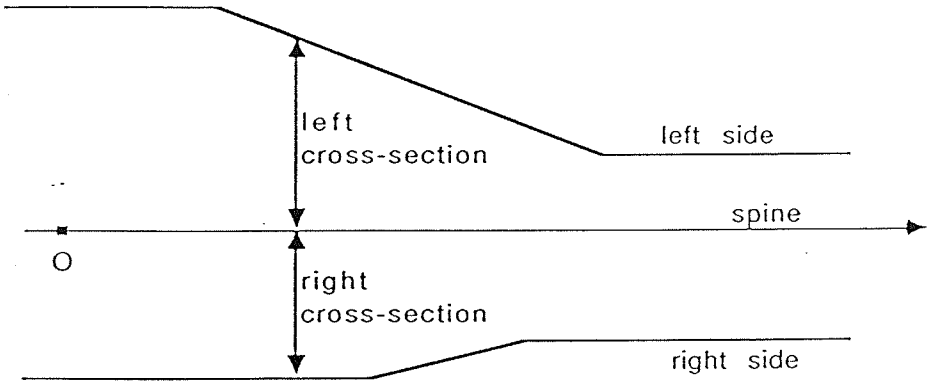

Figure 15: **Two-Dimensional Straight Linear Generalized Cylinder**

9.1 Extraction of Freeways

The geometry of a freeway in $\Re^2$ is that of a truncated two-dimensional straight linear generalized cylinder (see Figure 15).

DEFINITION 17: *A* **two-dimensional straight linear generalized cylinder** *is a region of* $\Re^2$ *obtained by sweeping a straight line segment, the* **cross-section**, *along and perpendicularly to a straight line, the* **spine**. *An origin and an orientation is defined on the spine. The cross-section is partitioned by the spine into two segments, the* **right** *and the* **left** *cross-sections. The lengths of the right and left cross-sections are continuous, piecewise linear functions of the abscissa along the spine. These functions are the right and left* **radii**. *The two lines drawn by the extremities of the cross-section are called the right and left* **sides** *of the cylinder.*

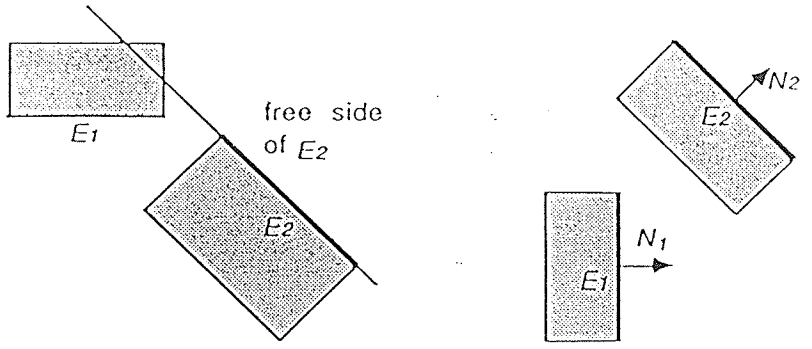

Figure 16: **Pairs of Edges Not Producing a Generalized Cylinder**

Freeways are extracted from $\mathcal{E}$ by considering all pairs of edges in $\partial(\mathcal{E})$. Any edge E is contained in an infinite line that divides $\Re^2$ into two half-planes. The outgoing normal vector of E points toward the half-plane called the *free half-plane* of E. A pair of edges (E_1, E_2) produces a generalized cylinder if and only if it satisfies the following two conditions:

(1) For $i, j \in \{1, 2\}$, $i \neq j$, one extremity of E_i is in the free half-plane of E_j.

(2) The inner product of the outgoing normal vector of E_1 and E_2 is negative.

These two conditions impose that E_1 and E_2 'face' each other. Figure 16 show examples of pairs of edges that do not satisfy these conditions.

Given a pair of edges (E_1, E_2) satisfying the above two conditions, a generalized cylinder (from now on, we omit the other qualifiers) GC is constructed as follows. GC's spine is the bisector of the angle formed by the lines containing E_1 and E_2; if these lines are parallel, the spine is parallel and equidistant to both of them. Each side of GC is made of one edge (E_1 or E_2) extended at each extremity by a half-line parallel to the spine. This construction is illustrated by Figure 17. The choice of the bisector for spine is reminiscent of the construction of a Voronoi diagram in the workspace.

GC partly lies outside $\mathcal{E}$ and we now select those pieces of GC which completely lie within $\mathcal{E}$ as illustrated by Figure 18. First, GC is intersected with the region $\Re^2 - \mathcal{E}$. The intersection is then normally projected on the spine, and the corresponding slices are removed from the cylinder. The truncated GC is thus sliced into several truncated generalized cylinders. Truncated cylinders whose sides do not include portions of both E_1 and E_2 are discarded. In the example of Figure 18, it only remains one slice denoted Φ. This way of removing slices from generalized cylinders is empirical and may seem a bit radical. However, one must keep in mind that there are usually many generalized cylinders leading to many overlapping freeways. In particular, some of the edges bounding the obstacles intersecting GC can usually be paired with either E_1 or E_2 to produce other generalized cylinders overlapping GC. The success of the freeway

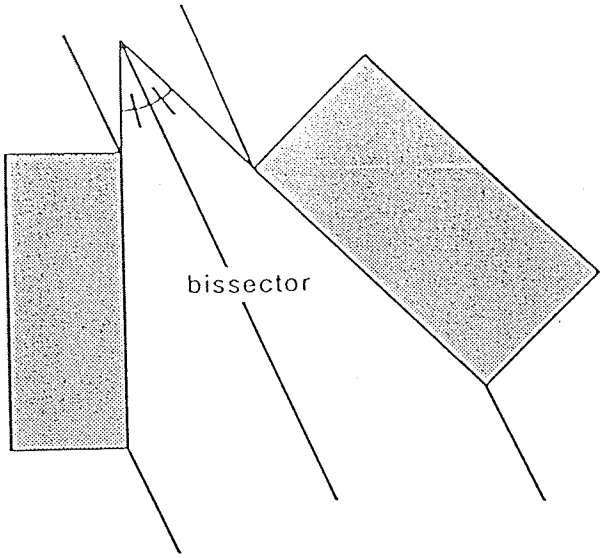

Figure 17: **Construction of a Generalized Cylinder**

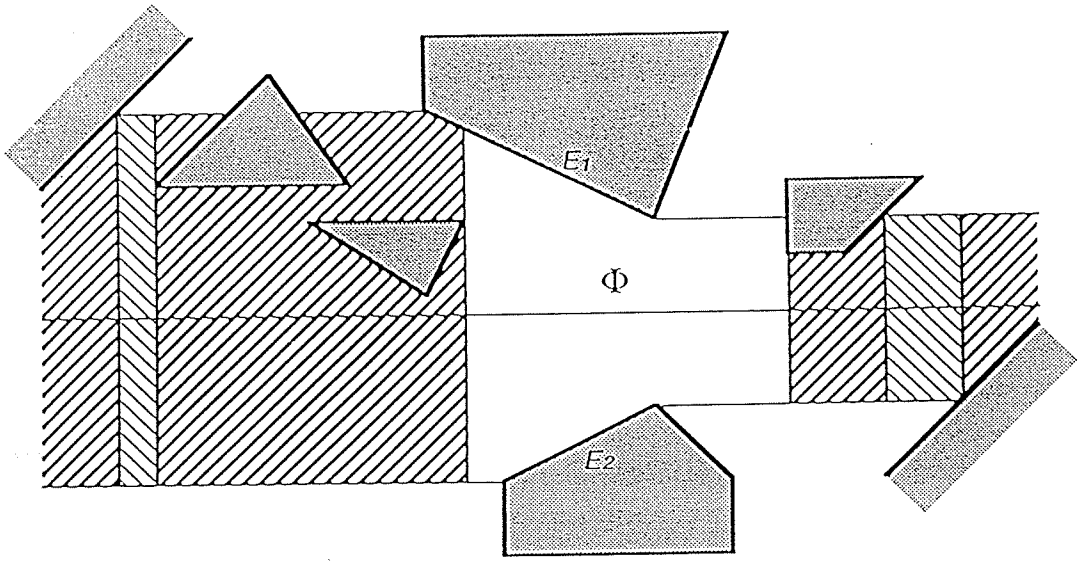

Figure 18: **Removing Slices from a Generalized Cylinder**

method derives from this multiplicity of freeways.

When all the pairs of edges of $\partial(\mathcal{E})$ have been considered, the remaining truncated generalized cylinders are the freeways to be used for path finding.

Each freeway is an instance of the general case shown in Figure 19. The spine of a freeway is oriented from the 'big' end toward the 'small' end; if the freeway has parallel sides, an arbitrary orientation is selected. The origin of spine abscissae is taken at the freeway's end (when it exists, the big end), such that any point in the freeway projects on the spine at a positive abscissa. Thus, the geometry of every freeway is completely specified by 7 parameters illustrated in Figure 19: L, B_l, B_r, S_l, S_r, ϕ and W.

Extraction of the initial non-truncated set of generalized cylinders takes $O(n^2)$, where n is the number of edges in $\partial(\mathcal{E})$. Intersecting $\Re^2 - \mathcal{E}$ with a generalized cylinder can be

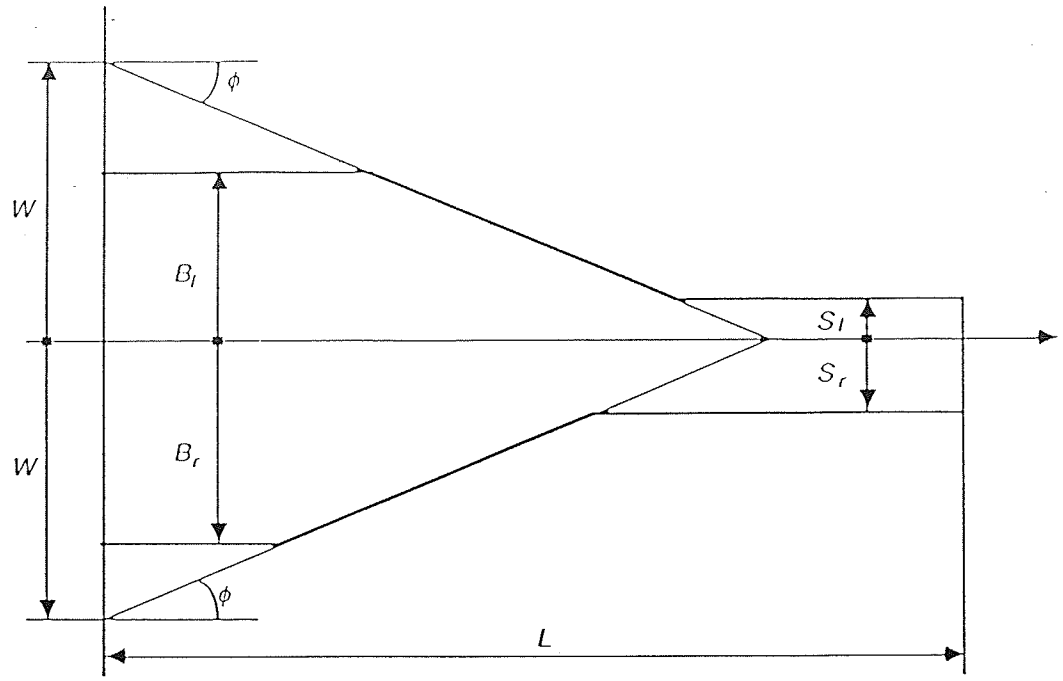

Figure 19: **Parameters Specifying a Freeway**

done in time $O(n)$. If the intersection is not empty, the intersection can be projected on the spine in time $O(n)$. Therefore, the time complexity of the complete operation is $O(n^3)$.

9.2 Freeways as Cross-Sections Through Configuration Space

After having extracted freeways, the next step is to compute the range of free orientations of $\mathcal{A}$ when it moves along the spine of a freeway. Before explaining how this can effectively be done, we must define more accurately what it means for $\mathcal{A}$ to 'move along' the spine.

We represent a configuration c of $\mathcal{A}$ by (x, y, θ), where x and y are the coordinates of the origin of $O_{\mathcal{A}}$ (the origin of $\mathcal{F}_{\mathcal{A}}$) in $\mathcal{F}_{\mathcal{W}}$, and $\theta \in [0, 2\pi)$ is the angle (modulo 2π) between the x-axes of $\mathcal{F}_{\mathcal{W}}$ and $\mathcal{F}_{\mathcal{A}}$.

When $\mathcal{A}$ 'moves along' the spine of a freeway, we impose that $O_{\mathcal{A}}$ stays on the spine. In other words, we constraint potential paths in $\mathcal{C}_{free}$ to be contained into planes projecting on the (x, y)-plane of $\mathcal{C}$ along the spines of the extracted freeways. This makes sense since each spine is equally distant from two obstacle edges. Determining the range of free orientations of $\mathcal{A}$ along the spine of a freeway is thus equivalent to computing the intersection of a plane parallel to the θ-axis with C-obstacles. In fact, as we will show later, the range of free orientations has to be determined only at some points on the spines.

Since $O_{\mathcal{A}}$ moves on a line which is equidistant from two obstacle edges, it is preferable to select it so that the maximum distance from $O_{\mathcal{A}}$ to the points on the boundary of $\mathcal{A}$ is the smallest possible. This is achieved by taking $O_{\mathcal{A}}$ at the center of the minimum

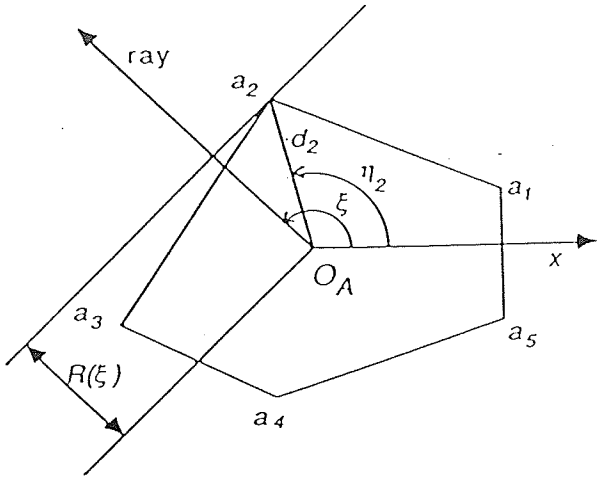

Figure 20: **Radius Function of the Moving Object**

spanning circle[12] of the set of vertices of $\mathcal{A}$. The computation of the minimum spanning circle of a set of points is a classical problem in Computational Geometry [42]. A naive algorithm does the computation in $O(n_{\mathcal{A}}^4)$ time, where $n_{\mathcal{A}}$ is the number of vertices of $\mathcal{A}$. Improved algorithms do the computation in $O(n_{\mathcal{A}}^2)$ [42], or even in $O(n_{\mathcal{A}} \log n_{\mathcal{A}})$ ($O(n_{\mathcal{A}})$ if we already know the convex hull of the points, which is the case when $\mathcal{A}$ is a convex polygon) [11].

9.3 Determination of Free Orientations

We now describe the technique proposed by Brooks for approximating the range of free orientations of $\mathcal{A}$ along a spine of a freeway. This technique makes use of a function, called the radius function of $\mathcal{A}$, and of its inverse. We first define these functions (see Figure 20).

DEFINITION 18: *A half-line issued from $O_{\mathcal{A}}$ is called a* **ray**. *The angle ξ (modulo 2π) between the x-axis of $\mathcal{F}_{\mathcal{A}}$ and the ray is called the* **angle** *of the ray with respect to $\mathcal{A}$. The* **radius function** *$R(\xi)$ of $\mathcal{A}$ is defined as the infimum of the distances from $O_{\mathcal{A}}$ to the lines which both are normal to a ray of angle ξ and do not intersect $\mathcal{A}$.*

We have (see Figure 20):

$$R(\xi) = \max_{1 \leq i \leq n_{\mathcal{A}}} \{d_i \cos(\xi - \eta_i)\}$$

where:
- d_i is the distance from $O_{\mathcal{A}}$ to the ith vertex of $\mathcal{A}$,
- η_i is the angle of the ray passing through the ith vertex of $\mathcal{A}$.

$R(\xi)$ specifies how close a line perpendicular to the ray of angle ξ can be from $O_{\mathcal{A}}$ without intersecting the interior of $\mathcal{A}$. The inverse image of an interval $[0, r]$ by R is

[12] The minimum spanning circle of a set of points is the smallest circle that encloses all of them.

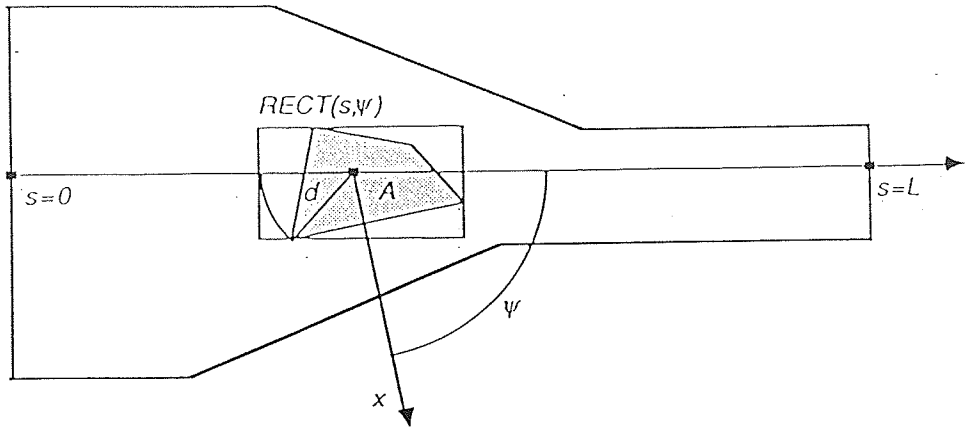

Figure 21: **Bounding $\mathcal{A}$ by a rectangle**

the function $R^{-1}(r)$ defined by:

$$R^{-1}(r) = \{\xi / R(\xi) < r\}.$$

$\forall \xi \in R^{-1}(r)$, a line perpendicular to the ray of angle ξ and distant from $O_{\mathcal{A}}$ by more than r is guaranteed to have no intersection with $\mathcal{A}$.

Let us assume that $\mathcal{A}$ is a convex polygon (otherwise, we approximate it by its convex hull). Then, $R^{-1}(r)$ can easily be computed as follows:

$$R^{-1}(r) = [0, 2\pi) - \bigcup_{1 \leq i \leq n_{\mathcal{A}}} \mathcal{I}_i$$

where:
- $\mathcal{I}_i = \emptyset$ if $r > d_i$,
- $\mathcal{I}_i = [\eta_i - |\arccos \frac{r}{d_i}|, \eta_i + |\arccos \frac{r}{d_i}|]$, modulo 2π, if $r \leq d_i$.

Let us now use the inverse radius function for computing the free orientations of $\mathcal{A}$ along the spine of a freeway Φ. In order to simplify computation, we approximate $\mathcal{A}$ by a bounding rectangle. By doing so, we obtain a subrange of the free orientations of $\mathcal{A}$.

Let us denote s the spine abscissa of $O_{\mathcal{A}}$ and ψ the angle between the x-axis of $\mathcal{F}_{\mathcal{A}}$ and Φ's spine. We enclose $\mathcal{A}$ in a rectangle $RECT(s, \psi)$ defined as follows (see Figure 21). Two sides of $RECT(s, \psi)$ are parallel to the spine, the *right* and *left* ones. The distance from $O_{\mathcal{A}}$ to these sides are $R(\psi - \pi/2)$ and $R(\psi + \pi/2)$, respectively. The other two sides are the *front* and the *rear* sides. The distance from $O_{\mathcal{A}}$ to the front side is $R(\psi)$. The distance from $O_{\mathcal{A}}$ to the rear side is conservatively $d = \max_{1 \leq i \leq n_{\mathcal{A}}}\{d_i\}$. The later choice is conservative, but the independence of d from ψ presents practical advantages as it will be appearent below. Notice that the bounding rectangle $RECT(s, \psi)$ depends on the orientation of $\mathcal{A}$ relative to the spine of Φ.

Let $\mathcal{V}(s)$, with $0 \leq s \leq L$, denote the set of orientations ψ, when $O_{\mathcal{A}}$'s spine abscissa is s, such that $RECT(s, \psi)$ is completely contained within Φ. Since $RECT(s, \psi)$ is a bounding approximation of $\mathcal{A}$, $\mathcal{V}(s)$ is a subset of the range of free orientations of $\mathcal{A}$. $\mathcal{V}(s)$ can be computed using the formulas given in the following lemma.

LEMMA 6: *For a freeway Φ constructed from two non-parallel edges, we have:*

- *if* $0 \leq s \leq d$, *then:*

$$\mathcal{V}(s) = \emptyset; \tag{1}$$

- *if* $d < s \leq L$, *then:*

$$\begin{aligned}\mathcal{V}(s) = R^{-1}(L-s) \bigcap ([(R^{-1}(S_r) + \frac{\pi}{2}) \cap (R^{-1}(S_l) - \frac{\pi}{2})] \\ \bigcup [(R^{-1}(B_r) + \tfrac{\pi}{2}) \cap (R^{-1}(B_l) - \tfrac{\pi}{2}) \cap R^{-1}_{-\frac{\pi}{2},\alpha}(\alpha s + W) \cap R^{-1}_{\frac{\pi}{2},\alpha}(\alpha s + W)])\end{aligned} \tag{2}$$

where $\alpha = \tan\phi$, $R_{\zeta,\alpha} = R(\psi + \zeta) - \alpha R(\psi)$, *and* $R^{-1}_{\zeta,\alpha}(u) = \{\psi \ / \ R_{\zeta,\alpha}(\psi) < u\}$.

In the particular case where Φ is a traditional 1-cylinder of radius W, relation 1 is unchanged and relation 2 simplifies to:

$$\mathcal{V}(s) = R^{-1}(L-s) \cap (R^{-1}(W) + \frac{\pi}{2}) \cap (R^{-1}(W) - \frac{\pi}{2}).$$

Relation 1 is obvious, since for $0 \leq s \leq d$, the rear side of $RECT(s, \psi)$ does not lie within Φ, whatever is ψ's value. (Notice here the pratical advantage of the conservative definition of d.) The best way to understand relation 2 is to parse it:

- The first part of the main conjunct, $R^{-1}(L-s)$, expresses the constraint that the abscissa of the front side of the rectangle must be less than L.

- The first sub-expression in the second part of the main conjunct, $(R^{-1}(S_r) + \frac{\pi}{2}) \cap (R^{-1}(S_l) - \frac{\pi}{2})$, expresses that the distance from the spine to the light (resp. left) side of Φ is nowhere smaller that S_r (resp. S_l). The conjunction of $R^{-1}(L-s)$ and this sub-expression is a sufficient condition insuring that $RECT(s, \psi)$ lies within Φ (for $s > d$). However, it is a too strong condition.

- The next two elements in relation 2, $(R^{-1}(B_r) + \frac{\pi}{2}) \cap (R^{-1}(B_l) - \frac{\pi}{2})$, express the fact that the distance from the spine to the right and left sides of Φ cannot be greater than B_r and B_l, respectively. These two conditions are too weak and remained to be composed with conditions considering the non-parallel sides of Φ.

- The last two elements in relation 2, $R^{-1}_{-\frac{\pi}{2},\alpha}(\alpha s + W) \cap R^{-1}_{\frac{\pi}{2},\alpha}(\alpha s + W)$, express a necessary and sufficient condition that the bounding rectangle be between the two lines containing the non-parallel sides of Φ. Indeed, for each line, we can first write:

$$R(\psi - \frac{\pi}{2}) \leq \alpha(s + R(\psi)) + W$$

and

$$R(\psi + \frac{\pi}{2}) \leq \alpha(s + R(\psi)) + W$$

where $\alpha = \tan\phi$. The above two elements in 2 directly derive from these relations.

Using formula 2 to compute $\mathcal{V}(s)$ requires being able to compute $R^{-1}_{\zeta,\alpha}(u)$, where $R_{\zeta,\alpha}(\psi) = R(\psi + \zeta) - \alpha R(\psi)$. This can be done easily by noticing that the sum of

two radius functions has the same form as a radius function, whose inverse can be computed as shown above. Indeed, if:

$$R(\xi) = \max_{1 \leq i \leq n} \{d_i \cos(\xi - \eta_i)\}$$

and

$$S(\xi) = \max_{1 \leq j \leq n'} \{d'_j \cos(\xi - \eta'_j)\}$$

then:

$$R(\xi) + S(\xi) = \max_{1 \leq i \leq n,\ 1 \leq j \leq n'} (d_i \cos(\xi - \eta_i) + d'_j \cos(\xi - \eta'_j))$$

where each term inside the 'max' can be written in the form $e_{ij} \cos(\xi - \nu_{ij})$.

Since both the right and left radii of Φ are non increasing functions of the spine abscissa, one can easily verify the important following lemma:

LEMMA 7: $\forall s_1, s_2 \in [0, L] : s_1 \leq s_2 \Rightarrow \mathcal{V}(s_2) \subseteq \mathcal{V}(s_1)$.

This property is the key to the construction of the freeway net, because it will permit us to determine free orientations of $\mathcal{A}$ only at some points along freeways' spines.

Notice that $\mathcal{V}(s)$ may consist of several disconnected intervals.

9.4 Construction and Search of the Freeway Net

At this point, we have built a collection of freeways. For each freeway Φ, the expressions giving $\mathcal{V}(s)$ describe the complement of a bounding approximation of the cross-section of C-obstacles in a plane containing the spine of Φ and perpendicular to the (x, y)-plane. We thus have a partial representation of free space. We now use this representation to construct the freeway net FN, which determines all the possible ways for $\mathcal{A}$ to move along spines.

Let $\mathcal{X}$ be the set of every point P such that the spines of two freeways intersect at P inside both freeways. Since there are $O(n^2)$ freeways (n is the number of edges in $\partial(\mathcal{E})$), a straightforward computation of $\mathcal{X}$ requires $O(n^4)$ time.

The nodes of FN are created as follows:

for every freeway Φ do:
  for every point P of Φ's spine that is in $\mathcal{X}$ do:
    - let s be the spine abscissa of P;
    - create one node of FN for every maximal connected interval in $\mathcal{V}(s)$.

The corresponding interval of $\mathcal{V}(s)$ is associated to each node of FN. Since each interval is originally defined with respect to a spine's orientation, an appropriate constant angle has to be added to both extremities of the interval, so that all the intervals are defined with respect to the same reference, for instance the x-axis.

In addition, the initial and goal positions of $O_{\mathcal{A}}$ are used to create the *initial* and *goal* nodes of FN. We assume here, for simplification, that each of these positions is on a

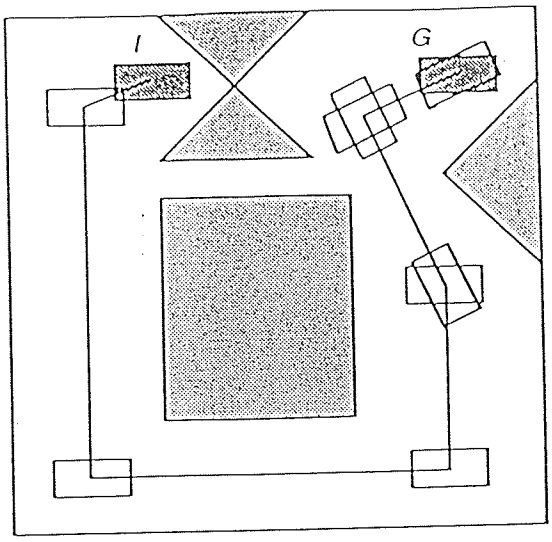

Figure 22: **A Path Produced Using the Freeway Method**

spine. The interval of $\mathcal{V}(s)$ containing the initial (resp. goal) orientation of $\mathcal{A}$ at the initial (resp. goal) position is associated with the corresponding node of FN. If the initial and goal positions of $\mathcal{A}$ are not located on freeway spines, then a technique has to be devised for moving $\mathcal{A}$ from the initial position to a spine, and from a spine to the goal position.

The arcs connecting the nodes of FN are created as follows:

(1) Let N_1 and N_2 be two nodes corresponding to two different points on the same freeway spine at abscissae s_1 and s_2. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the two intervals of free orientations of $\mathcal{A}$ associated with these nodes. If $\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$, then N_1 and N_2 are connected by an arc in FN. Indeed, thanks to Lemma 7, at any orientation in $\mathcal{I}_1 \cap \mathcal{I}_2$, $\mathcal{A}$ can move in free space from abscissa s_1 to abscissa s_2, and vice-versa.

(2) Let N_1 and N_2 be two nodes corresponding to the same point in space on two different spines. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the intervals of free orientations associated with these nodes. If $\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$, then N_1 and N_2 are connected by an arc in FN. Indeed, at the corresponding point, $\mathcal{A}$ can transfer in free space from one of the freeway to the other.

The constructed freeway net can then be searched for a path between the initial node to the goal node. A A^* algorithm can be used [36]; various heuristic functions are possible. A path in the net determines a class of homotopic free paths in $\mathcal{C}_{free}$. A particular path can be selected by rotating $\mathcal{A}$, if necessary, only at the intersections between spines. Figure 22 shows a path produced using this technique (this example is drawn from Brooks's paper [9]).

Experiments with the freeway method has shown that it works fast in 'relatively uncluttered' workspace. However, the method is not complete, and may not always find a free path when one exists. Indeed, it embodies several empirical choices, whose justification is merely intuitive, for instance: the spine of each freeway is the bisector of the two lines containing a pair of edges, each freeway is extended at each vertex by parallels to the spine, $\mathcal{O}_\mathcal{A}$ is constrained to remain on spines, etc. One of the major drawbacks when

the workspace is densely occupied by obstacles, is that freeways are often too short to contain the rectangle enclosing $\mathcal{A}$. Even in rather uncluttered workspaces the freeway method may fail.

Some of the limitations of the method could be alleviated. For example, it is possible to construct additional freeways by introducing fictitious edges in the workspace and by considering spines which are not bisectors of the lines containing edges. But, the size of the freeway net and the cost of searching it would increase significantly. Probably, it is better to take the freeway method as it is, i.e. fast and incomplete, and not try to make it too complicate, i.e. slow and still incomplete.

10 Visibility Graph

Besides decomposition and retraction, the visibility graph method is another approach to path finding. However, unlike the other two, it is not universal, i.e. its principle is only applicable to the specific case where configuration space is made of (generalized) polygonal obstacles. Nevertheless, historically, the visibility graph method has been one of the earliest path finding methods used to control a robot (SHAKEY project [35]). Due to its conceptual simplicity, it remains quite popular for implementing mobile robot systems. Furthermore, it can be combined with other types of methods to handle more difficult cases, e.g. to deal with rotation.

We have $\mathcal{C} = \Re^2$ and the C-obstacles are possibly overlapping convex polygons. The principle of the visibility graph method is to construct a path as the concatenation of line segments connecting the initial to the final configurations through C-obstacle vertices. Therefore, it produces paths which lie in valid space[13].

Let us consider a polygonal object $\mathcal{A}$ translating among convex polygonal obstacles $\mathcal{B}_i$. We want to plan a path between any given initial and final valid configurations $\mathbf{c}_1$ and $\mathbf{c}_2$.

DEFINITION 19: *Let VG be the graph constructed as follows:*
- VG's nodes are the initial and final configurations $\mathbf{c}_1$ and $\mathbf{c}_2$, and the vertices of the C-obstacles.
- Two nodes are connected by an undirected arc if and only if they are 'mutually visible', i.e. the line segment joining the corresponding two points lies completely in valid space.
VG is called the **visibility graph** *in $\mathcal{C}$.*

Figure 23 shows the visibility graph for a simple configuration space with three C-obstacles (the graph arcs include the obstacle edges).

The visibility graph contains a path from $\mathbf{c}_1$ to $\mathbf{c}_2$ if and only if there is a valid path between the two configurations. In addition, if a valid path exists, the shortest valid path (according to the Euclidean metric in $\Re^2$) is made of line segments passing through vertices of the C-obstacles. The graph VG contains such a path, which is shown in bold line in Figure 23.

[13] One can grow C-obstacles slightly, so that the visibility graph method produces free paths.

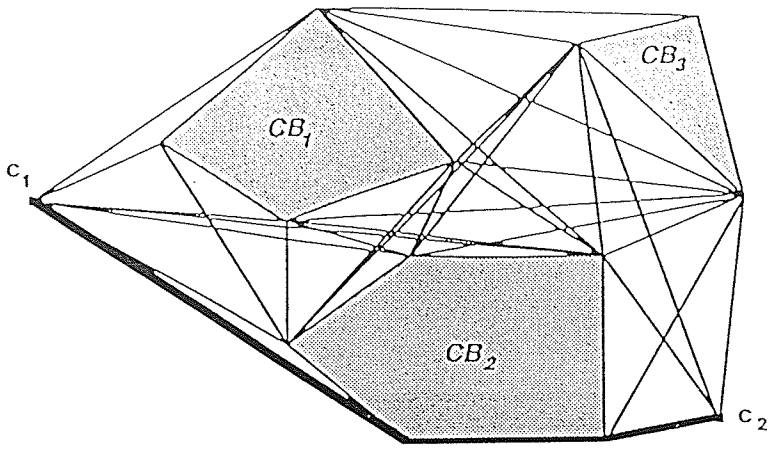

Figure 23: **Example of Visibility Graph**

The overall algorithm of the visibility graph method is the following:

1. Compute the vertices of the C-obstacles and construct the visibility graph VG.
2. Starting at $\mathbf{c}_1$, search VG for a path from $\mathbf{c}_1$ to $\mathbf{c}_2$.
3. If $\mathbf{c}_2$ is attained, then return the list of vertices along the path; otherwise indicate failure.

In order to construct VG, one may consider each node X in VG. By rotating a half-line emanating from X and using a line-sweep algorithm, one can construct the arcs connecting X to other nodes in VG.

Searching VG can be done using the A^* algorithm [36]. The heuristic function f to guide the search is a function which maps each node X into an estimate of the length of the shortest path from $\mathbf{c}_1$ to $\mathbf{c}_2$ passing through X. We may take:

$$f(\mathbf{c}_1, \mathbf{c}_2, X) = g(\mathbf{c}_1, X) + h(X, \mathbf{c}_2)$$

where $g(\mathbf{c}_1, X)$ is the length of the shortest path discovered so far between $\mathbf{c}_1$ and X, and $h(X, \mathbf{c}_2)$ is the Euclidean distance in $\Re^2$ between X and $\mathbf{c}_2$. Since we have $0 \leq h(X, \mathbf{c}_2) \leq d_{valid}(X, \mathbf{c}_2)$, where $d_{valid}(X, \mathbf{c}_2)$ denotes the (unknown) length of the shortest path between X and $\mathbf{c}_2$ in $\mathcal{C}_{valid}$, the heuristic function is admissible. Then, the visibility graph method is guaranteed to return the shortest path from $\mathbf{c}_1$ to $\mathbf{c}_2$, if a valid path exists, and to indicate failure, otherwise.

LEMMA 8: *With a A^* search algorithm using an admissible heuristic function, the visibility graph method is complete and generates the shortest valid path whenever a valid path exists.*

Let n be the number of vertices of the C-obstacles. The construction of the arcs outgoing from a node in VG using the rotating half line-sweep takes $O(n \log n)$ time, so that the total time for constructing VG is $O(n^2 \log n)$. The resulting graph contains $O(n)$ node and $O(n^2)$ arcs, so that the combinatorial search of VG using the A^* algorithm with

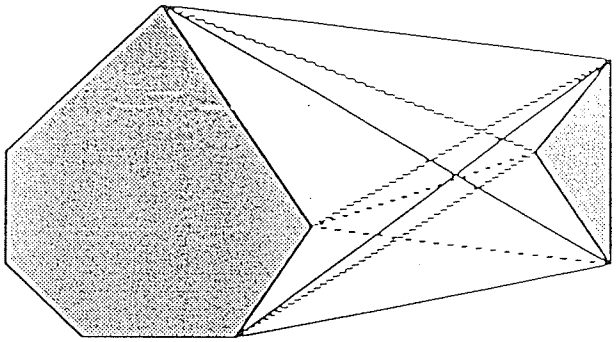

Figure 24: **Simplification of the Visibility Graph**

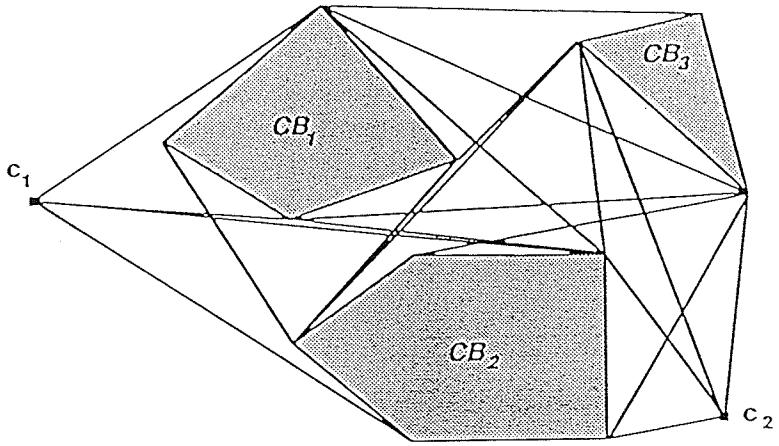

Figure 25: **Simplified Visibility Graph**

an admissible heuristics takes $O(n^2)$ time. Therefore, the total time complexity of the visibility graph method is $O(n^2 \log n)$.

The above method can be improved in several ways. For instance, some arcs in VG need not be generated. Indeed, the line segment connecting two points X_1 and X_2 in free space may be part of a shortest path, only if the infinite line containing the two points does not intersect the interiors of the C-obstacles to which these points belong in an arbitrarily small neighborhood of these points. The remaining segments are called *supporting segments* and there are exactly four of them among the vertices of any two disjoint convex polygons [45]. This is illustrated by Figure 24. The supporting segments are shown as plain lines, the non-supporting segment in free space are shown as dashed lines. Thus, considering only supporting segments reduces the size of the search graph (see Figure 25). In addition, more efficient graph construction techniques are applicable if the C-obstacles are known to be both convex and disjoint [53,44,3,45].

The visibility graph technique can be extended to the case where both $\mathcal{A}$ and the $\mathcal{B}_i$s are **generalized polygons** bounded by line segments and circular arcs. In this case, C-obstacles are also generalized polygons (see [25]). The 'generalized' visibility graph is built as explained above using actual vertices of the generalized polygonal C-obstacles, and is completed by fictitious vertices as follows [25]:

- Let X be a vertex (or the initial or final configuration) and E a circular arc. If there

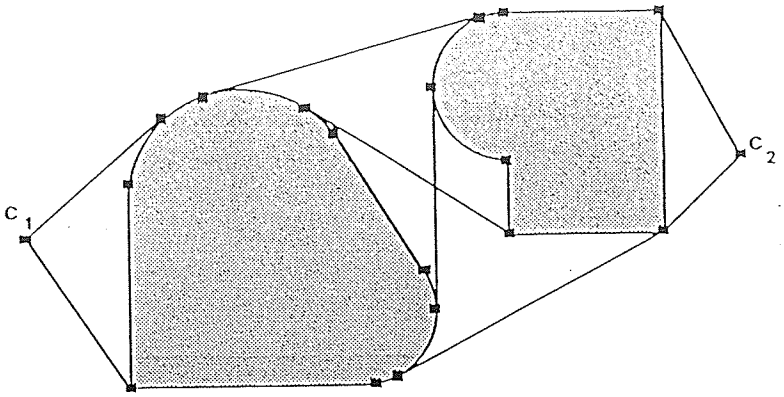

Figure 26: **Generalized Visibility Graph**

exists a point X' on E such that the open line segment joining X and X' lies in free space and the infinite line supporting the segment is tangent to E, then X' is included as a fictitious vertex in the generalized visibility graph, and X and X' are connected by an arc.

- Let E and E' be two circular edges. If there exists a point X on E and a point X' on E' such that the open line segment between these two points lies in free space and the infinite line supporting the segment is tangent to both E and E', then X and X' are included in the graph as fictitious vertices, and they are connected by an arc.

- Any two vertices X and X' located on the same circular edge of a C-obstacle are connected by an arc.

Figure 26 shows the generalized visibility graph in a configuration space with two generalized polygonal C-obstacles. Only supporting segments are shown in the figure.

The same rotating line-sweep technique as above is applicable and allows to compute the generalized visibility graph in $O(n^2 \log n)$ time.

Unfortunately, the visibility graph method is not applicable to the case of a three-dimensional polyhedral configuration space. Indeed, in such a space, there may exist no valid path consisting of a sequence of straight line segments joining the initial and final configurations through C-obstacle vertices. For instance, the polyhedral C-obstacles may form kinds of 'tunnels'. However, the problem of finding shortest valid paths among polyhedral obstacles has recently attracted interest in Computational Geometry. It can be shown that, as illustrated by Figure 27, shortest valid paths are sequences of line segments adjacent at points on polyhedron edges. Canny [10] shows that the problem of generating such paths is NP-hard in n and the best bound so far is due to Reif and Storer [44] who give a $2^{n^{O(1)}}$-time algorithm. Papadimitriou [41] gives an algorithm that finds a path which is at most $(1+\epsilon)$ times the length of the shortest path, in time that is polynomial in n and $1/\epsilon$. The basic idea of this algorithm is to break edges into short segments and to search a graph with these segments as nodes.

Algorithms for finding shortest paths among polyhedra are quite technical and of limited interest in robot path planning. In fact, the visibility graph method is attractive for robot motion planning in two-dimensional workspace, more because it is conceptually

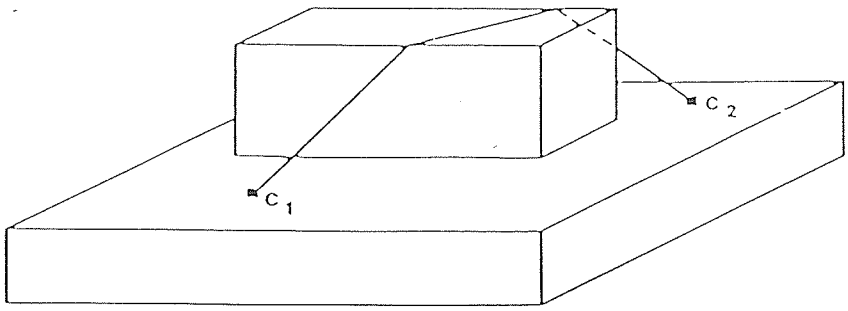

Figure 27: **Shortest Path Among Polyhedral C-Obstacles**

simple, easy to implement, and relatively time efficient, rather than because it generates shortest paths. It is more appropriate to apply other methods such as those described in the previous sections to the three-dimensional path planning problem.

The visibility graph method can neither be extended to the case of a robot translating and rotating in a two-dimensional workspace. However, it can be combined with other techniques in order to handle rotations of the robots. For instance, it can be combined with the 'orientation slicing' technique described in Subsection 7.2. A visibility graph VG_k can be built for every orientation interval $[\theta_k, \theta'_k]$, by first computing a bounding polygonal approximation of the area swept out by the moving object when it rotates from θ_k to θ'_k about the reference point, and next, considering this approximated area as a translating object, computing the polygonal C-obstacles to this object. The graphs VG_k can be combined into a larger graph VG by linking any two nodes $X_1 \in VG_{k_1}$ and $X_2 \in VG_{k_2}$ $(k_1 \neq k_2)$, whenever the two points X_1 and X_2 can be connected by a straight segment intersecting no C-obstacle in both $\mathcal{C}^{k_1}$ and $\mathcal{C}^{k_2}$, and $\theta'_{k_1} = \theta_{k_2}$ (modulo 2π). A path consisting of interweaved pure translations and pure rotations can be generated by searching VG.

11 Conclusion

Until rather recently path planning has not been considered a central and difficult problem in the development of advanced autonomous robots. It was often thought that appropriate heuristics would be sufficient to solve it in almost every case. Research on motion planning became quite active after the mid-70's, when the importance of the problem started being recognized. Theoretical studies and practical implementations showed it was also a difficult problem.

Work on path planning has attracted the interest of a many researchers in the 80's. Much progress has been done and the problem is now fairly well understood. Several approaches have been proposed and, within each of them, different techniques have been developed. Some of these methods may have practical applications, provided they are carefully engineered if order to fit the characteristics of the application tasks. Indeed, the computational complexity of both the path planning problem (lower bound) and the proposed techniques (higher bounds) has been analyzed in depth. Since this complexity increases exponentially with the number of degrees of freedom of the robotic systems,

it is clear that there is no universal efficient solution. But experiments with various implementations have shown that some of the existing techniques can be very efficient in restricted domains.

It is clear that path planning for one robot among fixed obstacles is only one facet of the larger motion planning problem. Research on the other facets (dealing with multiple robots, mobile obstacles, dynamic constraints, incomplete knowledge, inexact knowledge, ...) currently attracts a lot of interest, but is still at an early stage and the application of existing approaches is more remote.

Existing path planning techniques can be applied to material movements in several ways. For example, they can be implemented as utilities in a CAD system, in order to allow the designer to anticipate the effects of his decisions on the execution process. Several software packages in the construction industry include tools for simulating the movements of material (such as pipes and beams) and machines, some have collision-checking capabilities, but none can perform automatic path planning automatically. Automatic path planning could also be useful within process planners. But it is clear that a major goal should be to make them available on-line on automatic material transportation robotic systems. In addition to making these systems more autonomous, this would greatly facilitate the integration of material physical movements with the updatings of information (e.g., where objects are at every instant, what processing steps have been performed so far), which is necessary to control material flow systems.

Acknowledgements

The author thanks Dr. Jocelyne Pertin-Troccaz who presented this paper at the NATO-Advanced Research Workshop on *Advanced Information Technologies for Industrial Material Flow Systems*, in Grenoble, France June 13-17.

References

[1] V.I.Arnold (1978). *Mathematical Methods of Classical Mechanics.* Springer-Verlag, New York.

[2] D.S.Arnon (1979). *A cellular decomposition Algorithm for Semi-Algebraic Sets.* Technical Report No. 353, Computer Science, University of Wisconsin.

[3] T.Asano, T.Asano, L.Guibas, J.Hershberger, H.Imai (1986). *Visibility of Disjoint Polygons.*, Algorithmitica 1 (1), 49-63.

[4] D.Ayala, P.Brunet, R.Juan, I.Navazo (1985). *Object Representation by Means of Nonminimal Division Quadtrees and Octrees.* ACM Transactions on Graphics, 4, 1 (January).

[5] T.O.Binford (1971). *Visual Perception by Computers.* Proc. of the IEEE Conference on Systems Science and Cybernetics, Miami, FL (December).

[6] O.Bottema, B.Roth (1979). *Theoretical Kinematics.* North-Holland Publishing Co., New York.

[7] R.A.Brooks, T.Lozano-Pérez (1982). *A Subdivision Algorithm in Configuration Space for Findpath with Rotation.* AI Memo 684, AI Laboratory, MIT (December).

[8] R.A.Brooks, T.Lozano-Pérez (1983). *A Subdivision Algorithm in Configuration Space for Findpath with Rotation.* Proc. of the Eighth International Joint Conference on Artificial Intelligence (IJCAI), Karlsruhe, FRG, August 1983.

[9] R.A.Brooks(1983). *Solving the Find-Path Problem by Good Representation of Free Space.* IEEE Transactions on Systems, Man and Cybernetics, SMC-13, No. 3.

[10] J.F.Canny (1987). *The Complexity of Robot Motion Planning.* Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA (May).

[11] C.Castells, R.Melville (1985). *An Unusual Algorithm for the Minimum Spanning Circle Problem.* Technical Report, Electrical Engineering and Computer Science, John Hopkins University, Baltimore, MD.

[12] R.Chatila (1982). *Path Planning and Environmental Learning in a Mobile Robot System.* Proceedings of the European Conference on Artificial Intelligence (ECAI), Orsay, France.

[13] B.Chazelle, D.P.Dobkin (1979). *Decomposing a Polygon into Convex Parts.* Proceedings of the 11th Annual ACM Symposium on Theory of Computing, 38-48.

[14] B.Chazelle (1985). *Approximation and Decomposition of Shapes.* In [50].

[15] G.E.Collins (1975). *Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition.* In Lecture Notes in Computer Science, No. 33, Springer-Verlag, New York, pp. 135-183.

[16] B.R.Donald (1984). *Motion Planning with Six Degrees of Freedom.* Technical Report 791, Artificial Intelligence Laboratory, MIT, Cambridge, MA (May).

[17] R.L.Drysdale III (1979). *Generalized Voronoi Diagrams and Geometric Searching.* Ph.D. Dissertation, Report STAN-CS-79-705, Department of Computer Science, Stanford University.

[18] M.Erdmann, T.Lozano-Pérez (1986). *On Multiple Moving Objects.* A.I.Memo No. 883, Artificial Intelligence Laboratory, MIT, Cambridge, MA (May).

[19] F.Germain (1984). *Planification de Trajectoires Sans Collision.* DEA Report, LIFIA Laboratory, National Polytechnic Institute of Grenoble.

[20] C.L.Jackins, S.L.Tanimoto (1980). *Octrees and their Use in Representing Three-Dimensional Objects.* Computer Graphics and Information Processing, 14, 3 (November).

[21] K.Jänich (1984). *Topology.* Springer-Verlag, New-York.

[22] O.Khatib (1986). *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots.* International Journal of Robotics Research 5, 1.

[23] D.G.Kirkpatrick (1979). *Efficient Computation of Continuous Skeletons.* Proceedings of the 20th Symposium on Foundations of Computer Science. 18-27.

[24] J.C.Latombe (1988). *Motion Planning With Uncertainty: The Preimage Backchaining Approach.* Technical Report No. STAN-CS-88-1196, Computer Science, Stanford University (March).

[25] J.P.Laumond (1987). *Obstacle Growing in a Non Polygonal World.* Information Processing Letters (April).

[26] D.T.Lee, R.L.Drysdale (1981). *Generalization of Voronoi Diagrams in the Plane.* SIAM Journal of Computing, 10, 73-87.

[27] D.Leven, M.Sharir (1985). *An Efficient and Simple Motion Planning Algorithm for a Ladder Moving in Two-Dimensional Space Amidst Polygonal Barriers.* Proc. of the First ACM Symposium on Computational Geometry, 211-227.

[28] D.Leven, M.Sharir (1987). *Intersection and Proximity Problems and Voronoi Diagrams.* In [50], 187-228.

[29] A.Lingas (1982). *The Power of Non-Rectilinear Holes.* Proceedings of the 9th Colloquium on Automata, Languages and Programming, Aarhus, LNCS Springer-Verlag, 369-383.

[30] T.Lozano-Pérez, M.A.Wesley (1979). *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles.* Communications of the ACM, Vol. 22, 10, 560-570 (October).

[31] T.Lozano-Pérez (1981). *Automatic Planning of Manipulator Transfer Movements.* IEEE Transactions on Systems, Man and Cybernetics, SMC-11.

[32] T.Lozano-Pérez (1983). *Spatial Planning: A Configuration Space Approach.* IEEE Transactions on Computers, C-32, No. 2.

[33] T.Lozano-Pérez (1987). *A Simple Motion-Planning Algorithm for General Manipulators.* IEEE Journal of Robotics and Automation, RA-3, 3 (June).

[34] D.Meagher (1982). *Geometric Modeling Using Octree Encoding.* Computer Graphics and Image Processing, 19.

[35] N.J.Nilsson (1969). *A Mobile Automaton: An Application of Artificial Intelligence Techniques.* Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI), Washington D.C., 509-520.

[36] N.J.Nilsson (1980). *Principles of Artificial Intelligence.* Morgan Kaufmann, Los Altos, CA.

[37] C.Ó'Dúnlaing, C.K.Yap (1982). *A Retraction Method for Planning the Motion of a Disc.* Journal of Algorithms, 6, 104-111.

[38] C.Ó'Dúnlaing, M.Sharir, C.K.Yap (1983). *Retraction: A New Approach to Motion Planning.* In Proc. of the 15th ACM Symposium on the Theory of Computing, Boston, 207-220.

[39] C.Ó'Dúnlaing, M.Sharir, C.K.Yap (1984). *Generalized Voronoi Diagrams for Moving a Ladder: I. Topological Analysis.* Technical Report No. 32, Robotics Laboratory, Courant Institute, New-York University.

[40] C.Ó'Dúnlaing, M.Sharir, C.K.Yap (1984). *Generalized Voronoi Diagrams for Moving a Ladder: II. Efficient Construction of the Diagram.* Technical Report No. 32, Robotics Laboratory, Courant Institute, New-York University.

[41] C.H.Papadimitriou (1985). *An Algorithm for Shortest-Path Motion in Three Dimensions.* Information Processing Letters 20, 259-263 (June).

[42] F,P.Preparata, M.I.Shamos (1985). *Computational Geometry: An Introduction.* Springer-Verlag, New York.

[43] J.Reif (1979). *Complexity of the Mover's Problem and Generalizations.* Proceedings of the 20th Symposium on the Foundations of Computer Science, 421-427.

[44] J.Reif, J.Storer (1985). *Shortest Paths in Euclidean Space with Polyhedral Obstacles.* Technical Report CS-85-121, Computer Science Department, Brandeis University (April).

[45] H.Rohnert (1986). *Shortest Paths in the Plane with Convex Polygonal Obstacles.* Information Processing Letters 23, 71-76 (August).

[46] H.Samet (1980). *Region Representation: Quadtrees from Boundary Codes.* Communications of the ACM, 23, 3 (March).

[47] J.T.Schwartz, M.Sharir (1983). *On the 'Piano Movers' Problem. I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers.* Communications on Pure and Applied Mathematics, 36, 345-398.

[48] J.T.Schwartz, M.Sharir (1983). *On the 'Piano Movers' Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds.* Advances in Applied Mathematics, 4, 298-351.

[49] J.T.Schwartz, M.Sharir, J.Hopcroft (1987). *Planning, Geometry, and Complexity of Robot Motion.* Ablex Publising Corp., Norwood, New Jersey.

[50] J.T.Schwartz, C.K.Yap (1987). *Algorithmic and Geometric Aspects of Robotics.* LEA, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey.

[51] M.Spivak (1979). *A Comprehensive Introduction to Differential Geometry.* Publish or Perish, Inc., Wilmington, Delaware.

[52] A.Tarski (1951). *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, CA.

[53] E.Welzl (1985). *Constructing the Visibility Graph for n line segments in $O(n^2)$.* Information Processing Letters 20 (4), 167-171 (May).

[54] C.K.Yap (1987). *Algorithmic Motion Planning.* In [50], 95-143.