August 1989 Report No. STAN-CS-89-1279

(2)

New Heuristic Algorithms for Efficient Hierarchical Path Planning

by

David Zhu and Jean-Claude Latombe

Department of Computer Science

Stanford University

Stanford, California 94305

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION IS UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Stanford University; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): Stanford, CA 94305	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: DARPA/CIS/CIFE; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAAA21-89-C-0002/Latombe Seed Res./Latombe Research Support
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification): New Heuristic Algorithms for Efficient Hierarchical Path Planning

12. PERSONAL AUTHOR(S): David Zhu and Jean-Claude Latombe

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
research	FROM _____ TO _____	89-8-21	42

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

One of the ultimate goals of robotics research is to create autonomous robots. Progress toward this goal requires advances in many domains, including automatic motion planning. The "basic problem" in motion planning is to construct a collision-free path for a moving object among fixed obstacles. Several approaches have been proposed, including cell decomposition, retraction, and potential field. Nevertheless, most existing planners still lack efficiency, or reliability, or both. In this paper, we consider one of the most popular approaches to path planning: hierarchical approximate cell decomposition. We propose a set of new algorithms for constructing more efficient and reliable path planners based on this general approach. These algorithms concern the hierarchical decomposition of the robot's configuration space into rectangloid cells, and the search of the connectivity graphs built at each level of decomposition. We have implemented these algorithms in a path planner and experimented with this planner on various examples. Some are described in the paper. These experiments show that our planner is significantly faster than previous planners based on the same general approach.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Jean-Claude Latombe	22b TELEPHONE (Include Area Code): 415-72300350; 22c OFFICE SYMBOL

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

New Heuristic Algorithms for Efficient Hierarchical Path Planning

David Zhu and Jean-Claude Latombe
Robotics Laboratory, Computer Science Department
Stanford University

Abstract: *One of the ultimate goals of robotics research is to create autonomous robots. Progress toward this goal requires advances in many domains, including automatic motion planning. The "basic problem" in motion planning is to construct a collision-free path for a moving object among fixed obstacles. Several approaches have been proposed, including cell decomposition, retraction, and potential field. Nevertheless, most existing planners still lack efficiency, or reliability, or both. In this paper, we consider one of the most popular approaches to path planning: hierarchical approximate cell decomposition. We propose a set of new algorithms for constructing more efficient and reliable path planners based on this general approach. These algorithms concern the hierarchical decomposition of the robot's configuration space into rectangloid cells, and the search of the connectivity graphs built at each level of decomposition. We have implemented these algorithms in a path planner and experimented with this planner on various examples. Some are described in the paper. These experiments show that our planner is significantly faster than previous planners based on the same general approach.*

Acknowledgements: This research was funded by DARPA contract DAAA21-89-C0002 (Army), CIS (Center of Integrated Systems), and CIFE (Center for Integrated Facility Engineering).

1 Introduction

One of the ultimate goals of robotics research is to create autonomous robots. Such robots will accept high-level descriptions of tasks and will execute them without further human intervention. The input descriptions will specify *what* the user wants done, rather than *how* to do it. Progress toward this goal requires advances in many domains, including automatic reasoning, perception, and real-time control. One of the key topics in reasoning is motion planning. It is aimed at providing robots with the capability of deciding which motion commands to execute in order to achieve specified arrangements of physical objects. During the last ten years, it has emerged as a major research area [Lozano-Perez, 1983] [Schwartz, Hopcroft and Sharir, 1987].

The "basic problem" – planning a collision-free path for a moving object among fixed obstacles – has attracted a lot of attention. Several approaches have been proposed, e.g.: exact cell decomposition [Schwartz and Sharir, 1983], approximate

cell decomposition [Brooks and Lozano-Pérez, 1983], retraction [Ó'Dúnlaing, Sharir and Yap, 1983], potential field [Khatib, 1986] [Khosla and Volpe, 1988] [Rimon and Koditschek, 1988]. Most of the existing methods, however, still lack efficiency, or reliability, or both, in practical cases.

In this paper, we consider one of the most popular approaches to motion planning: hierarchical approximate cell decomposition. It was first introduced in [Brooks and Lozano-Pérez, 1983], with subsequent contributions by other authors (e.g., [Laugier and Germain, 1985] [Faverjon, 1986] [Kambhampati and Davis, 1986]). We propose a set of new algorithms for constructing more efficient path planners based on this approach. We have implemented these algorithms in a planner and experimented with them on a variety of examples. Taking into account the relative speed of the computers, our planner is significantly (approximately 10 times) faster than previous planners using the same approach. Like other path planners based on cell decomposition, our planner generates a sequence of cells, which we call a "channel". Hence, the robot does not commit itself to a single path at planning time, so that it can adapt its path at execution time, using, say, a potential field method for avoiding unexpected obstacles. We have implemented this approach on an actual mobile robot moving in a time varying office environment [Choi, Zhu and Latombe, 1989].

The hierarchical approximate cell decomposition approach consists of decomposing the configuration space of the moving object into rectangloid cells at successive levels of approximation. Cells are classified to be EMPTY or FULL, depending on whether they lie entirely outside or entirely inside the obstacles. If they are neither EMPTY, nor FULL, they are labelled MIXED. At each level of approximation, the planner searches the graph of the adjacency relation among the cells for a sequence of adjacent EMPTY cells connecting the initial configuration of the robot to the goal configuration. If no such sequence is found, it decomposes some MIXED cells into smaller cells, label them appropriately, and searches again for a sequence of EMPTY cells. The process ends when a solution has been found, or it is guaranteed that no solution can be found, or MIXED cells are smaller than some prespecified size.

Most motion planning approaches have their own advantages and drawbacks, which have to be weighted in function of the context in which they are considered. Nevertheless, the hierarchical cell decomposition approach presents a rather unique combination of attractive features. It is relatively easy to implement even when the moving object can both translate and rotate. It is reasonably efficient when the number of degrees of freedom of the moving object is small, and parallelization is possible for achieving better performances. It is complete under reasonable assumptions, if we accept no upper bound on the worst-case time complexity, and "resolution-complete" otherwise. Finally, as mentioned above, it produces a channel, which leaves some freedom at execution time, for instance, to optimize dynamic behavior and avoid

unexpected obstacles.

However, despite the conceptual simplicity of the main two steps of the approach – cell decomposition and graph searching –, their efficient implementation raises delicate questions not thoroughly addressed in previous publications. While we were implementing and experimenting with a planner for a mobile robot, we found out that efficiency can be sharply increased by shifting from naive answers to these questions, to more sophisticated ones. In this paper, we present in detail the new algorithms which we have developed, and we report on our experimentation with the implemented planner.

The problem of decomposing a MIXED cell is to maximize the volume of the EMPTY and FULL cells resulting from the decomposition, in order to make it possible finding a path (or the absence of path) as quickly as possible. In particular, the blind 2^n-tree (e.g., quadtree, octree) decomposition technique has the drawback of decomposing many MIXED cells into smaller MIXED cells. We propose a new "constraint reformulation" technique, which provides better results than earlier decomposition techniques. It consists of approximating the obstacles intersecting with the MIXED cell to be decomposed by a collection of rectangloids, and computing the complement of these rectangloids in the cell. Two types of approximation are used, "bounding" and "bounded" approximations. The first is used to produce EMPTY cells, the second to produce FULL cells.

The problem of graph searching is to take advantage of unsuccessful search work done at lower levels of approximation, since most of the search graph remains the same from one level to the next (only the portions of the graph, which correspond to decomposed MIXED cells, are modified). We propose a set of search techniques based on appropriate representation of the search graph and on the recording of failure conditions. These techniques, inspired from those implemented in dependency-directed backtracking systems [Stallman and Sussman, 1977] [Latombe, 1979], avoid the path planner to run into the same mistakes several times.

The paper consists of four sections, in addition to the introduction and the conclusion. In Section 2 we review the concepts underlying the hierarchical cell decomposition approach and introduces the necessary terminology. In Sections 3 and 4, we investigate the cell decomposition and graph searching problems, respectively, and we describe in detail the new algorithms which we have developed. In Section 5 we present some of the experiments which we conducted with the implemented planner.

2 Background and Overview

2.1 Configuration space

Let us consider a rigid object $\mathcal{A}$ moving in a Euclidean space $\mathcal{W} = \mathbf{R}^N$ – called the **workspace** – among fixed obstacles $\mathcal{B}_i$, $i = 1, ..., q$. Both $\mathcal{A}$ and the $\mathcal{B}_i$'s are closed regions in $\mathbf{R}^N$.

A standard Cartesian coordinate frame, denoted by $\mathcal{F}_\mathcal{W}$, is embedded in $\mathcal{W}$. Another Cartesian frame, denoted by $\mathcal{F}_\mathcal{A}$, is embedded in $\mathcal{A}$. The origin of $\mathcal{F}_\mathcal{A}$, denoted by $O_\mathcal{A}$, is called the **reference point** of $\mathcal{A}$.

A **configuration** 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 by $\mathcal{C}$, of all the possible configurations of $\mathcal{A}$. The unique configuration where $\mathcal{F}_\mathcal{A}$ and $\mathcal{F}_\mathcal{W}$ coincide is called the **reference configuration** of $\mathcal{A}$. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration $\mathbf{q}$ is denoted by $\mathcal{A}(\mathbf{q})$.

The obstacles $\mathcal{B}_i$'s map in $\mathcal{C}$ as closed regions denoted by $\mathcal{CB}_i$ and called **C-obstacles**. These regions are defined by:

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C} \;/\; \mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\}.$$

The region:

$$\mathcal{C}_{free} = \mathcal{C} - \bigcup_{i \in [1,q]} \mathcal{CB}_i$$

is called the **free space**. A **collision-free path** (more simply, a **free path**) is any continuous map $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$.

In this paper, we consider the case where $\mathcal{A}$ is a two-dimensional object, which translates and rotates in $\mathcal{W} = \mathbf{R}^2$. In this case, $\mathcal{C}$ is a manifold diffeomorphic to $\mathbf{R}^2 \times S^1$, where S^1 denotes the unit circle [Spivak, 1979]. We parameterize a configuration in this manifold by a triplet (x, y, θ), where $x, y \in \mathbf{R}^2$ are the coordinates of $O_\mathcal{A}$ 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 $\mathcal{F}_\mathcal{A}$.

2.2 C-obstacles in Polygonal Case

Throughout the paper, both $\mathcal{A}$ and the $\mathcal{B}_i$'s are polygons. Under this condition, the cross-section of each $\mathcal{CB}_i$ at any orientation θ is also a polygon defined by (see [Lozano-Pérez, 1983]):

$$\mathcal{CB}_i = \mathcal{B}_i \ominus \mathcal{A}(0, 0, \theta)$$

where $\mathcal{A}(0, 0, \theta)$ denotes the region occupied by $\mathcal{A}$ at configuration $(0, 0, \theta)$ and $\ominus$ is the symbol for the Minkowski difference. Each edge of this polygon is the locus of $O_\mathcal{A}$

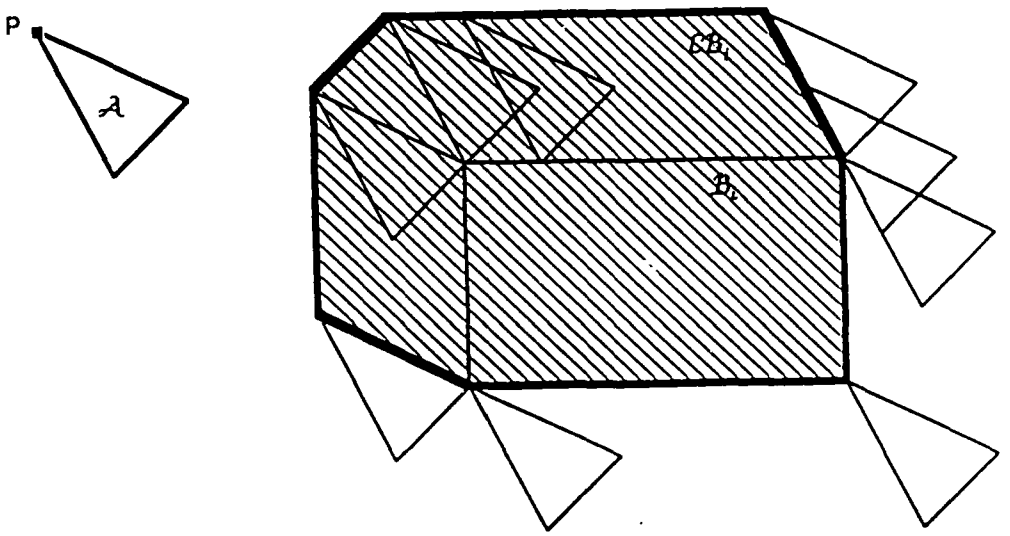

Figure 1. Both $\mathcal{A}$ and $\mathcal{B}$ are polygons. The cross-section through the C-obstacle $\mathcal{CB}$ at a fixed orientation θ of $\mathcal{A}$ is another polygon equal to $\mathcal{B} \ominus \mathcal{A}(0,0,\theta)$ (see text).

when $\mathcal{A}$ translates at fixed orientation θ, in such a way that an edge (resp. a vertex) of $\mathcal{A}$ stays in contact with a vertex (resp. an edge) of $\mathcal{B}_i$ (see Figure 1). A contact between an edge of $\mathcal{A}$ and a vertex of $\mathcal{B}_i$ is called a **Type A** contact. When $\mathcal{A}$ rotates slightly, the corresponding edge of $\mathcal{CB}_i$ rotates by the same angle. A contact between a vertex of $\mathcal{A}$ and an edge of $\mathcal{B}_i$ is called a **Type B** contact. When $\mathcal{A}$ rotates slightly, the corresponding edge of $\mathcal{CB}_i$ translates.

Therefore, the C-obstacle $\mathcal{CB}$ corresponding to an obstacle $\mathcal{B}$ is a three-dimensional volume without hole, which is bounded by patches of ruled surfaces, which we call C-facets. Each C-facet is generated by a straight line segment of variable length, which remains parallel to the xy plane and either translates or rotates. A C-facet created by a contact of Type A (resp. Type B) is called a Type A (resp. Type B) C-facet. Each C-facet is comprised between two limit orientations, beyond which the contact that creates the C-facet is no longer feasible.

The geometry of $\mathcal{CB}$, when $\mathcal{A}$ and $\mathcal{B}$ are polygons, is studied in depth in various publications, e.g. [Lozano-Pérez, 1983] [Donald, 1984] [Avnaim and Boissonnat, 1988] [Brost, 1989].

If $\mathcal{A}$ and $\mathcal{B}$ are both convex polygons, $\mathcal{CB}$ is bounded by $O(n_A n_B)$ C-facets, where n_A

and n_B are the number of edges in $\mathcal{A}$ and $\mathcal{B}$, respectively. If $\mathcal{A}$ and $\mathcal{B}$ are non-convex, $\mathcal{CB}$ has $\Omega(n_A^3 n_B^3)$ C-facets [Avnaim and Boissonnat, 1988].

2.3 Rectangloid Decomposition

In the following, we will assume, without practical loss of generality, that the range of possible values for x and y are closed intervals $[x_{min}, x_{max}]$ and $[y_{min}, y_{max}]$. We represent $\mathcal{C}$ as a closed rectangloid

$$\mathcal{K} = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [0, 2\pi] \subset \mathbf{R}^3$$

with the two cross-sections at $\theta = 0$ and $\theta = 2\pi$ procedurally identified.

Let κ be a rectangloid, i.e. a region of the form:

$$[x_1, x_2] \times [y_1, y_2] \times [\theta_1, \theta_2] \subseteq \mathcal{K}.$$

A **rectangloid decomposition** of κ is a collection of rectangloids, $\{\kappa_j\}_{j=1,\ldots,n}$, such that:

1. κ is equal to the union of the κ_j, i.e.:

$$\kappa = \bigcup_{j=1}^{n} \kappa_j.$$

2. The κ_j are non-overlapping[1]:

$$\forall j_1, j_2 \in [1, n],\ j_1 \neq j_2 :\ int(\kappa_{j_1}) \cap int(\kappa_{j_2}) = \emptyset.$$

Each rectangloid κ_j is called a **cell** of κ in the decomposition. Two cells κ_{j_1} and κ_{j_2} are **adjacent** iff their intersection is not a set of measure zero in $\mathbf{R}^2$, i.e. the intersection of two of their faces is a surface of non-zero area. (The intersection is computed by taking into account that the cross-sections at 0 and 2π are identified.)

A cell κ_j is classified as:

- EMPTY, iff its interior $int(\kappa_j)$ intersects no C-obstacle, i.e. $int(\kappa_j) \cap \bigcup_i \mathcal{CB}_i = \emptyset$;

- FULL, iff κ_j is entirely contained in the union of the C-obstacles, i.e. $\kappa_j \subset \bigcup_i \mathcal{CB}_i$;

- MIXED, otherwise.

Given an initial configuration

$$\mathbf{q}_{init} = (x_{init}, y_{init}, \theta_{init})$$

[1] Throughout this paper, we say that two closed sets are non-overlapping iff their interiors do not intersect.

and a goal configuration

$$q_{goal} = (x_{goal}, y_{goal}, \theta_{goal}),$$

a rectangloid decomposition of $\mathcal{K}$, $\{\kappa_j\}_{j=1,...,n}$, is **admissible** iff it contains a sequence of EMPTY cells κ_k, $k = 1, ..., p$, called an EMPTY **channel**, such that:

- $(x_{init}, y_{init}, \theta_{init}) \in \kappa_1$;

- $(x_{goal}, y_{goal}, \theta_{goal}) \in \kappa_p$;

- $\forall k \in [1, p-1]$: κ_k and κ_{k+1} are adjacent.

Let $(\kappa_k)_{k=1,...,p}$ be an EMPTY channel. Any path connecting (x_{init}, y_{init}, θ_{init}) to (x_{goal}, y_{goal}, θ_{goal}) and lying entirely in the interior of $\bigcup_{k=1}^{p} \kappa_k$ is a free path. Such a path can easily be constructed as an open polygonal line. If necessary, it can also be smoothed by fitting splines curves [Kant and Zucker, 1986].

A rectangloid decomposition is said to be **semi-admissible** if it contains a sequence of EMPTY and MIXED cells κ_k, $k = 1, ..., p$, called MIXED **channel**, with the same three properties as above. A MIXED channel may contain a free path connecting the initial to the goal configuration, but there is no guarantee that this is the case.

One could think of decomposing configuration space into fine cells at a single level of approximation, labelling the cells EMPTY of FULL depending on whether their interiors lie entirely in free space or not, and searching the connectivity graph of the EMPTY cell (see [Gouzènes, 1984] [Lozano-Pérez, 1987]). The problem with this one-level decomposition approach is that either the number of cells is large and the planner is always time inefficient, or the number of cells is relatively small (i.e., the decomposition is coarse) and the planner often fails to find free paths, while some exist. The motivation for the hierarchical decomposition introduced below is that in general large chunks of configuration space may be labelled EMPTY or FULL, drastically reducing the number of cells and consequently the size of the search graph.

2.4 Hierarchical Path Planning

Hierarchical path planning consists of generating an EMPTY channel by constructing successive rectangloid decompositions of $\mathcal{K}$, until an admissible decomposition has been generated and an EMPTY channel has been extracted. Let $\mathcal{P}_l$, $l = 0, 1, ...$, denote the successive decompositions of $\mathcal{K}$, with $\mathcal{P}_0 = \{\mathcal{K}\}$. Each decomposition $\mathcal{P}_l$, $l > 0$, is obtained from the previous one $\mathcal{P}_{l-1}$, by decomposing one or several MIXED cells, the other cells being unchanged.

Whenever a decomposition $\mathcal{P}_l$, $l \geq 0$, is generated and its cells labelled EMPTY, FULL, or MIXED, an undirected graph denoted by $\mathcal{CCG}_l$ is constructed:

- The nodes of $\mathcal{CCG}_l$ are the EMPTY and MIXED cells in $\mathcal{P}_l$.

- Any two nodes are connected by a link iff they are adjacent.

This graph is called the **cell-connectivity graph**, or **ccg**, of the decomposition $\mathcal{P}_l$.

Once constructed[2], the graph CCG_l is searched for an EMPTY or MIXED channel. Three outcomes are possible:

1. An EMPTY channel is found. Then, the planner returns success.

2. No EMPTY or MIXED channel is found. Then, the planner returns failure.

3. A MIXED channel is found, but no EMPTY channel.

In the third case, the planner proceeds recursively by decomposing the MIXED cells contained in the MIXED channel. Hence, the planner iteratively refines the "interesting" areas.

Let us assume that the region occupied by the C-obstacles can be expressed as the union of disjoint sub-regions, each equal to the closure of its interior ("full-bodiness assumption")[3]. Then, the planning process sketched above can be made complete – i.e., guaranteed to terminate successfully, whenever a free path exists, and to return failure, otherwise. This requires that some details of the algorithms be worked out appropriately. However, for an unknown problem, there is no time bound on the process, since there is no lower bound on the size of the cells which may have to be generated.

The worst-case time complexity can be bounded at the expense of completeness, by imposing constraints on the decomposition of any MIXED cell κ. One possible constraint is that the total volume of the EMPTY and FULL cells extracted from κ be greater than a predefined ratio of the volume of κ; in addition, every MIXED cell resulting from the decomposition of κ which has any of its dimensions (side length) smaller than a predefined value is relabelled FULL. Another possible constraint is that every generated cells should have dimensions greater than prespecified values. The choice of the constraints on the decomposition of a MIXED cell typically depends on the decomposition technique. If the decomposition algorithm fails to find a decomposition of κ satisfying the constraint, κ is said to be non-decomposable and is re-labelled FULL.

If constraints are imposed on MIXED cell decomposition, it is no longer guaranteed that a free path is found, whenever one such path exists. However, the planner can be made "resolution-complete", i.e. if an EMPTY channel exists at the resolution determined by the constraints, it will be found.

[2]Actually, the graph is a by-product of the decomposition and is constructed concurrently with the decomposition.

[3]In Solid Modelling, a closed set that is the closure of its interior is called a *regular* set.

If the assumption of "full-bodiness" of the disjoint C-obstacle regions is not satisfied, then in the absence of constraints on the decomposition of MIXED cells, the planning process may loop for ever. The constraints given above guarantee termination in bounded time.

In the presence of uncertainty in robot control, a minimal size requirement may also have to be imposed to both the cells in a channel and the intersection of two successive cells in the channel, in order to allow the robot to move safely despite uncertainty.

3 Cell Decomposition

3.1 Issue

The subproblem considered in this section is that of generating a rectangloid decomposition $\{\kappa_j\}_{j=1,\ldots,n}$ of a given MIXED cell κ. For the efficiency of the planning process, the generated decomposition should simultaneously satisfy the following two goals:

1. The number of cells in the decomposition should be reasonably small, in order to keep the size of the search graph as tractable as possible. This goal directly relates to the motivation for the overall hierarchical strategy: we want the planner to consider "details" only when it is necessary.

2. The volume of the EMPTY and FULL cells should be large relatively to the total volume of κ. This goal is aimed at reducing as quickly as possible the "uncertain area" – i.e., the MIXED cells. Clearly, decomposing a MIXED cell into smaller cells would not be useful, if all of these cells were MIXED.

These two goals may be conflicting since one obvious way to achieve the second goal is to produce many small cells, which is in contradiction with the first one. A good quantitative measure of the efficiency of the decomposition of a cell may be:

$$E = \frac{1}{N_{EMPTY} + N_{MIXED}} \frac{V_{EMPTY} + V_{FULL}}{V_\kappa}$$

where:

- N_{EMPTY} (resp. N_{MIXED}) is the number of newly generated cells labelled EMPTY (resp. MIXED),

- V_{EMPTY} (resp. V_{FULL}) is the total volume of newly generated EMPTY (resp. FULL) cells,

- V_κ is the volume of the cell κ.

3.2 Previous Approaches

One simple method of decomposing a MIXED cell is to partition it into 2^m cells of equal dimensions, where m is the dimension of configuration space (3 in our case). The overall decomposition of configuration space obtained with this method is called "quadtree" when $m = 2$ and "octree" when $m = 3$ [Ayala et al, 1985]. The application of this method is reported in [Kambhampati and Davis, 1986] (quadtree) and [Faverjon, 1986] (octree). The advantage of this method is that it leads to a decomposition easily representable in a tree structure of degree 2^m. The drawback is that most of the time none or a few of the newly generated cells are EMPTY or FULL. The method tends to produce a huge number of cells.

Another method is described in [Brooks and Lozano-Pérez, 1983]. The basic idea is to consider potential cuts of the cell, score them, and choose the best. The potential cuts are chosen wherever a C-surface[4] will go through a vertex of one of the new cells generated by the decomposition. The scoring function favors cuts which do not generate small cells, i.e. the cuts closer from the mid-points of each edge are preferred. The scoring function also attempts to minimize the number of C-surfaces intersected by each new cell, in order to reduce future computations. This method has the drawback of treating each C-surface (not C-facets) individually and to combine the effect of the various C-surfaces in a global scoring function. Although certainly better than a 2^m-tree, the resulting decomposition may still be far from optimal. It may also incorrectly[5] label a cell that intersects no C-obstacle as MIXED.

In [Lozano-Pérez, 1983], Lozano-Pérez describes a method which consists of slicing the orientation axis into intervals, computing the area swept out by $\mathcal{A}$ when it rotates about its reference point in each interval, approximating the swept area as a polygon, and growing the obstacles by this polygon. The result is a decomposition of configuration space into prismatic cells, which are either empty or not empty (the latter are not characterized further). A path is search among the empty cells only. If it fails, the angular intervals are refined. The decomposition method is very similar to the "swept-area" method proposed below. Our method, however, generates EMPTY, FULL and MIXED cells.

3.3 Constraint Reformulation

We propose a different approach for decomposing MIXED cells. It consists of first approximating each C-obstacle lying in the cell κ to be decomposed as a collection of non-overlapping rectangloids. The complement of a union of rectangloids within a rectangloid region is also a union of rectangloids, which can easily be computed.

[4] A *C-surface* is the infinite ruled surface that support a C-facet.

[5] This conservative "error" gets corrected when the decomposition proceeds deeper.

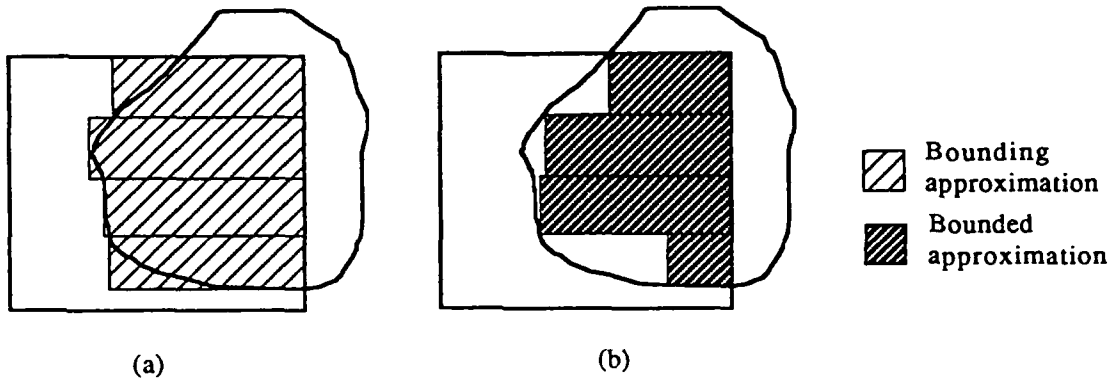

Figure 2. This figure illustrates the notion of bounding (a) and bounded (b) approximations of a region in a two-dimensional space.

This yields a rectangloid decomposition of κ. We call this approach *contraint reformulation*, since it basically consists of reformulating the constraints imposed by the C-obstacles into a form directly compatible with the format of the decomposition of $\mathcal{K}$ into rectangloid cells.

Let $\kappa = [x_1, x_2] \times [y_1, y_2] \times [\theta_1, \theta_2]$ be the MIXED cell to be decomposed. For every $i \in [1, q]$, we denote by $\mathcal{CB}_i[\kappa]$ the portion of $\mathcal{CB}_i$ contained in κ, i.e.:

$$\mathcal{CB}_i[\kappa] = \mathcal{CB}_i \cap \kappa.$$

Our planner generates and uses two types of approximation of C-obstacles, bounding and bounded approximations:

1. A **bounding approximation** of $\mathcal{CB}_i[\kappa]$ is a collection of non-overlapping rectangloids $\mathcal{R}_{ik}$'s, $k = 1, ..., p$, with $\forall k \in [1, p] : \mathcal{R}_{ik} \subset \kappa$ and $\mathcal{CB}_i[\kappa] \subseteq \bigcup_{k=1,...,p} \mathcal{R}_{ik}$.

2. A **bounded approximation** of $\mathcal{CB}_i[\kappa]$ is a collection of non-overlapping rectangloids $\mathcal{R}'_{ik}$'s, $k = 1, ..., p'$, with $\bigcup_{k=1,...,p'} \mathcal{R}'_{ik} \subseteq \mathcal{CB}_i[\kappa]$.

The EMPTY cells of the decomposition of κ are obtained by computing the complement of $\bigcup_i \bigcup_k \mathcal{R}_{ik}$ in κ. The FULL cells are the $\mathcal{R}'_{ik}$'s. The MIXED cells are obtained by computing the complement of $\bigcup_i \bigcup_k \mathcal{R}'_{ik}$ in every $\mathcal{R}_{ik}$.

Figure 2 illustrates the notion of bounding and bounded approximations in a two-dimensional space. Figure 3 illustrates the rectangloid decomposition of a cell κ into EMPTY, FULL and MIXED cells, using these two approximations.

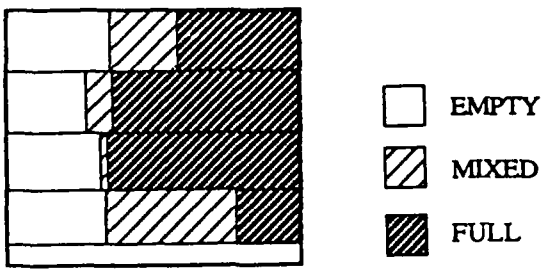

Figure 3. This figure illustrates the decomposition of a rectangular cell into a collection of EMPTY, FULL and MIXED cells built using the bounding and bounded approximations of Figure 2.

3.4 Outline of the Algorithm

There are infinitely many ways to generate bounding and bounded approximations of a C-obstacle $\mathcal{CB}$ in a cell κ. Our method computes "outer" and "inner" projections of $\mathcal{CB}$ on the xy plane, next on the x or y axis. Then, it lifts back the projection into rectangloid cells. It consists of the following two steps:

1. **Decomposition of $[\theta_1, \theta_2]$:** The $[\theta_1, \theta_2]$ interval is cut into non-overlapping subintervals $[\gamma_u, \gamma_{u+1}]$, $u = 1, ..., r$, $r \geq 1$, with $\gamma_1 = \theta_1$ and $\gamma_{r+1} = \theta_2$. We denote by κ^u the rectangloid $[x_1, x_2] \times [y_1, y_2] \times [\gamma_u, \gamma_{u+1}]$.

   For every $u \in [1, r]$, we compute the **outer projection** and the **inner projection** of $\mathcal{CB}[\kappa^u]$ on the xy plane. These two projections, which we respectively denote by $\mathcal{OCB}_{xy}[\kappa^u]$ and $\mathcal{ICB}_{xy}[\kappa^u]$, are defined by:

$$\mathcal{OCB}_{xy}[\kappa^u] = \{(x,y) \;/\; \exists \theta \in [\gamma_u, \gamma_{u+1}] : \; (x,y,\theta) \in \mathcal{CB}[\kappa^u]\},$$

$$\mathcal{ICB}_{xy}[\kappa^u] = \{(x,y) \;/\; \forall \theta \in [\gamma_u, \gamma_{u+1}] : \; (x,y,\theta) \in \mathcal{CB}[\kappa^u]\}.$$

   Clearly, we have: $\mathcal{ICB}_{xy}[\kappa^u] \subseteq \mathcal{OCB}_{xy}[\kappa^u]$.

2. **Decomposition of $[x_1, x_2]$ and $[y_1, y_2]$:** For every $u \in [1, r]$, the interval $[x_1, x_2]$ or $[y_1, y_2]$, whichever is longer, is cut into non-overlapping subintervals.

   Let us assume that $[x_1, x_2]$ is subdivided (a similar presentation would be made in the case where $[y_1, y_2]$ was decomposed). The generated subintervals are $[a_v, a_{v+1}]$, $v = 1, ..., s$, $s \geq 1$ with $a_1 = x_1$ and $a_{s+1} = x_2$. We denote by κ^{uv} the rectangloid $[a_v, a_{v+1}] \times [y_1, y_2] \times [\gamma_u, \gamma_{u+1}]$.

For every $v \in [1, s]$, we compute the outer projection of $\mathcal{OCB}_{xy}[\kappa^u]$ on the y axis, i.e.:

$$\mathcal{OCB}_y[\kappa^{uv}] = \{y \ / \ \exists x \in [a_v, a_{v+1}] : \ (x, y) \in \mathcal{OCB}_{xy}[\kappa^u]\}$$

and the inner projection of $\mathcal{ICB}_{xy}[\kappa^u]$ on the y axis, i.e.:

$$\mathcal{ICB}_y[\kappa^{uv}] = \{y \ / \ \forall x \in [a_v, a_{v+1}] : \ (x, y) \in \mathcal{ICB}_{xy}[\kappa^u]\}.$$

Both $\mathcal{OCB}_y[\kappa^{uv}]$ and $\mathcal{ICB}_y[\kappa^{uv}]$ are sets of intervals. $\mathcal{ICB}_y[\kappa^{uv}] \subseteq \mathcal{OCB}_y[\kappa^{uv}]$.

Each rectangloid $[a_v, a_{v+1}] \times [b, b'] \times [\gamma_u, \gamma_{u+1}]$, where $[b, b'] \in \mathcal{OCB}_y[\kappa^{uv}]$ (resp. $\mathcal{ICB}_y[\kappa^{uv}]$) is a rectangloid $\mathcal{R}_{ik}$ (resp. $\mathcal{R}'_{ik}$) in the bounding (resp. bounded) approximation of $\mathcal{CB}_i[\kappa]$ (see Subsection 3.3).

The choice of the γ_u's and the a_v's is empirical. Various heuristics can be used, but most of them seem to have limited effect on the average efficiency of the method. The only useful heuristic guideline is to keep the three dimensions of every MIXED cell approximately "homogeneous". Let δx, δy and $\delta\theta$ be these dimensions. We say they are "homogeneous" iff $\delta x \approx \delta y \approx \rho\delta\theta$, where ρ is the maximal distance between $O_{\mathcal{A}}$ and the points in the boundary of $\mathcal{A}$.

In the next two subsections, we describe the computation of the outer and inner projections of $\mathcal{CB}[\kappa^u]$ on the xy plane. We propose two different methods. The first, called "projection" method, consists of computing the projection of the surface of $\mathcal{CB}$ comprised between γ_u and γ_{u+1}, and clipping the subset of the projection contained in $[x_1, x_2] \times [y_1, y_2]$. The second method, called "swept-area" method, consists of computing the "outer" and the "inner" swept areas of $\mathcal{A}$ when it rotates around the reference point from orientation γ_u to orientation γ_{u+1}, and growing the obstacle $\mathcal{B}$ by these two areas. The projection method is preferred when the interval $[\gamma_u, \gamma_{u+1}]$ is small. The swept-area method runs significantly faster when this interval is large; however, it does not exactly compute $\mathcal{ICB}_{xy}[\kappa^u]$, but a subset of it. The computation of $\mathcal{OCB}_y[\kappa^{uv}]$ and $\mathcal{ICB}_y[\kappa^{uv}]$ is quite simple and not described below.

In the following, we assume that the reference point $O_{\mathcal{A}}$ is chosen in the interior of $\mathcal{A}$. In fact, the choice of $O_{\mathcal{A}}$ has some impact on the efficiency of the decomposition. The "best" location of $O_{\mathcal{A}}$ should minimize the area of the outer projection of $\mathcal{CB}[\kappa^u]$ on the xy plane, while maximizing the area of the inner projection, so that less MIXED cells are ultimately generated. The center of the "smallest enclosing circle" of $\mathcal{A}$ is probably close to minimize the area of the outer projection, while the center of the "largest enclosed circle" of $\mathcal{A}$ should be close to maximize the area of the inner projection. A compromise between these two locations is in general necessary, since they do not coincide.

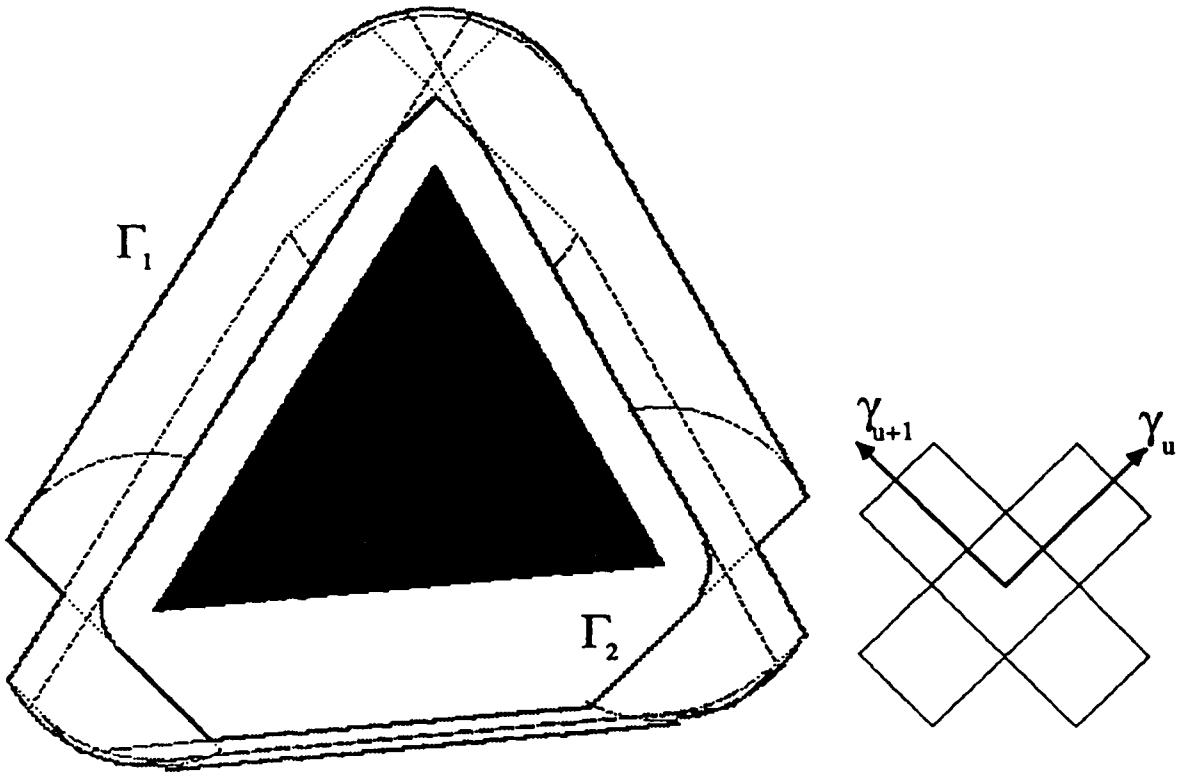

Figure 4. The C-patches comprised in an angular slice $[\gamma_u, \gamma_{u+1}]$ project on the xy plane according to generalized polygons, whose union is a "donut" shaped region bounded by an outer contour Γ_1 and an inner contour Γ_2.

3.5 Projection Method

Principle. Let us first assume that both $\mathcal{A}$ and $\mathcal{B}$ are convex polygons. In $\mathbf{R}^2 \times [0, 2\pi]$, $\mathcal{CB}$ is a volume without hole, which is bounded by C-facets (see Subsection 2.2). Consider the point (x_0, y_0) in the xy plane and the segment $\{(x_0, y_0, \theta) \ / \ \theta \in [\gamma_u, \gamma_{u+1}]\}$ above this point. If the segment pierces a C-facet, then (x_0, y_0) is in $\mathcal{OCB}_{xy}[\kappa^u]$, but not in $\mathcal{ICB}_{xy}[\kappa^u]$. If it pierces no C-facet, then either (x_0, y_0) is not in $\mathcal{OCB}_{xy}[\kappa^u]$, or it is in $\mathcal{ICB}_{xy}[\kappa^u]$. The segment $\{(x_0, y_0, \theta) \ / \ \theta \in [\gamma_u, \gamma_{u+1}]\}$ pierces a C-facet e iff (x_0, y_0) lies in the projection of e in the xy plane.

We show below that each C-facet comprised between γ_u and γ_{u+1} projects on the xy plane according to a generalized polygon[6]. The projection of the boundary of $\mathcal{CB}$ that is comprised between γ_u and γ_{u+1} is the union of generalized polygons, each being the projection of a C-facet. This union is a "donut" shaped region (see Figure 4),

[6] A *generalized polygon* is a compact two-dimensional region bounded by a simple curve consisting of straight segments and circular arcs.

with an outer boundary Γ_1 and an inner boundary Γ_2. The compact region bounded by Γ_1 is $\mathcal{OCB}_{xy}[\kappa^u]$. The compact region bounded by Γ_2 is $\mathcal{ICB}_{xy}[\kappa^u]$.

Therefore, the projection method consists of: (1) projecting all the C-facets (to the extent they are contained in the interval $[\gamma_u, \gamma_{u+1}]$) on the xy plane; (2) clipping the union of the projections by the rectangle $[x_1, x_2] \times [y_1, y_2]$ and, within this rectangle, tracking Γ_1 and Γ_2.

The projection method may be made faster by associating with each interval $[\gamma_u, \gamma_{u+1}]$ the list of all the C-obstacles having a non-empty intersection with the interval, and for each of these C-obstacles the list of the C-facets that intersect the interval. If the interval gets decomposed further, only these C-facets have to be considered.

The projection method is efficient when the number of C-facets which have to be projected is reasonably small, that is when the interval $[\gamma_u, \gamma_{u+1}]$ is small.

Projection of Type A C-Facet. Let e_A be a C-facet of Type A comprised between the limit orientations ϕ_1 and ϕ_2. The contact that generates e_A is illustrated in Figure 5 a. It occurs between an edge E of $\mathcal{A}$ and a vertex b of $\mathcal{B}$. E connects two vertices of $\mathcal{A}$, a_1 and a_2. b is the extremity of two edges of $\mathcal{B}$, F_1 and F_2. O_A projects on the supporting line of E at the point p. We assume below that p is located between a_1 and a_2. (The case where p is outside the segment $\overline{a_1 a_2}$ is treated in a very similar fashion.) The orientation ϕ_1 (resp. ϕ_2) is achieved when the edge E is aligned with the edge F_1 (resp. F_2). Assuming that $[\phi_1, \phi_2] \subseteq [\gamma_u, \gamma_{u+1}]$, the projection of e_A on the xy plane is shown in Figure 5 **d**. It is obtained as the union of two regions shown in Figures 5 **b** and **c**.

The region in Figure 5 **b** is the locus of O_A, when $\mathcal{A}$ translates and rotates while the edge segment $\overline{a_1 p}$ stays in contact with the vertex b. This region is bounded by two circular arcs and two straight segments. The two arcs are centered at b. The smaller one is the locus of O_A when p coincides with b and $\mathcal{A}$ rotates from ϕ_1 to ϕ_2. The larger arc is the locus of O_A when a_1 coincides with b and $\mathcal{A}$ rotates from ϕ_1 to ϕ_2. The straight segments are the loci of O_A when $\mathcal{A}$ translates at fixed orientations ϕ_1 and ϕ_2 from the position where p and b coincide to the position where a_1 and b coincide. The region in Figure 5 **c** is the locus of O_A, when $\mathcal{A}$ translates and rotates while the edge segment $\overline{p a_2}$ stays in contact with the vertex b.

The case of e_A projecting for the orientation range $[\gamma_u, \gamma_{u+1}] \subset [\phi_1, \phi_2]$ can be treated in the same way, by drawing fictitious edges F_1 and F_2 from vertex b (see Figure 5 e), so that, when E is aligned with F_1 (resp. F_2), $\mathcal{A}$'s orientation is γ_u (resp. γ_{u+1}).

Projection of a Type B C-Facet. Let e_B be a C-facet of Type B comprised between the limit orientations ϕ_1 and ϕ_2. The contact that generates e_B is illustrated

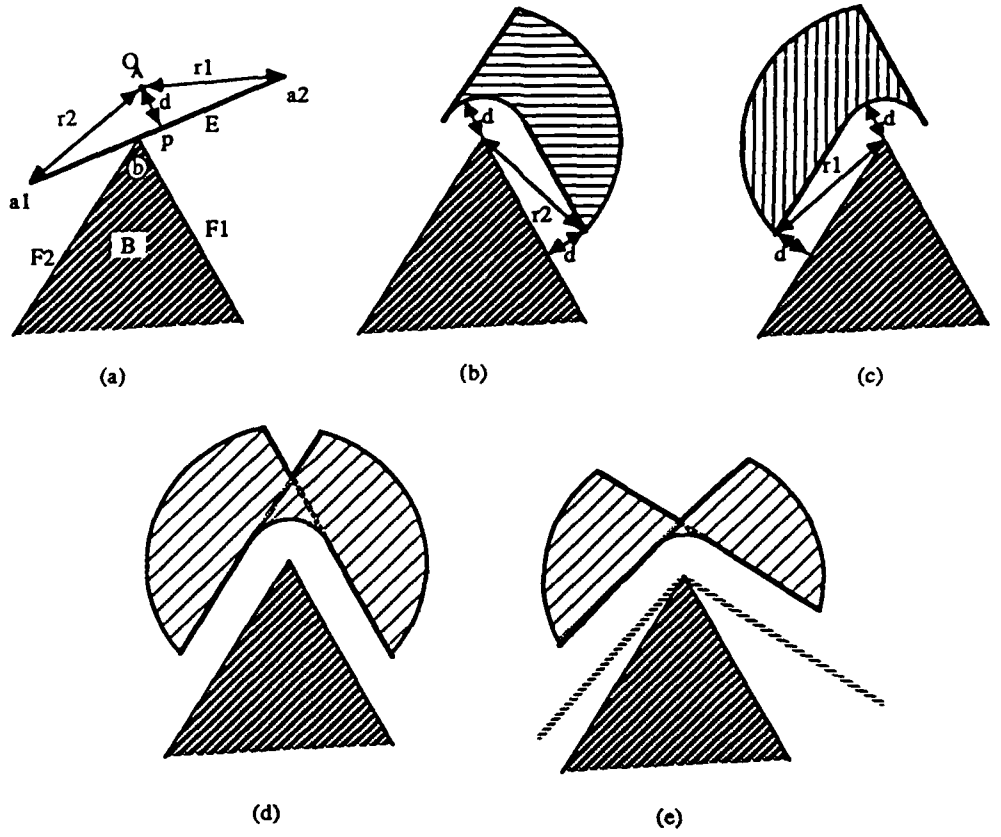

Figure 5. This figure illustrates the computation of the projection of a C-patch of Type A on the xy plane (see text).

in Figure 6 a. It occurs between a vertex a of $\mathcal{A}$ and an edge F of $\mathcal{B}$. a is the extremity of two edges of $\mathcal{A}$, E_1 and E_2. F connects two vertices of $\mathcal{B}$, b_1 and b_2. Since we assumed $\mathcal{A}$ to be convex, $O_{\mathcal{A}}$ lies within the convex angular sector bounded by the two half-lines supporting E_1 and E_2 and erected from a. The orientation ϕ_1 (resp. ϕ_2) is achieved when the edge E_2 (resp. E_1) is aligned with the edge F. Assuming that $[\phi_1, \phi_2] \subseteq [\gamma_u, \gamma_{u+1}]$, the projection of e_B on the xy plane is shown in Figure 6 d. It is obtained as the union of two regions, shown in Figures 6 b and c.

Let ψ be the orientation of $\mathcal{A}$ when a lies in F and the segment $\overline{aO_{\mathcal{A}}}$ is perpendicular to F. The region in Figure 6 b is the locus of $O_{\mathcal{A}}$, when $\mathcal{A}$ translates and rotates with the vertex a staying in F and the orientation θ ranging over $[\phi_1, \psi]$. (If $\psi < \phi_1$, then the region is empty.) This region is bounded by two circular arcs and two straight segments. The two arcs are centered at b_1 and b_2, respectively, and have the same radius equal to the distance between a and $O_{\mathcal{A}}$. The region in Figure 6 c is the

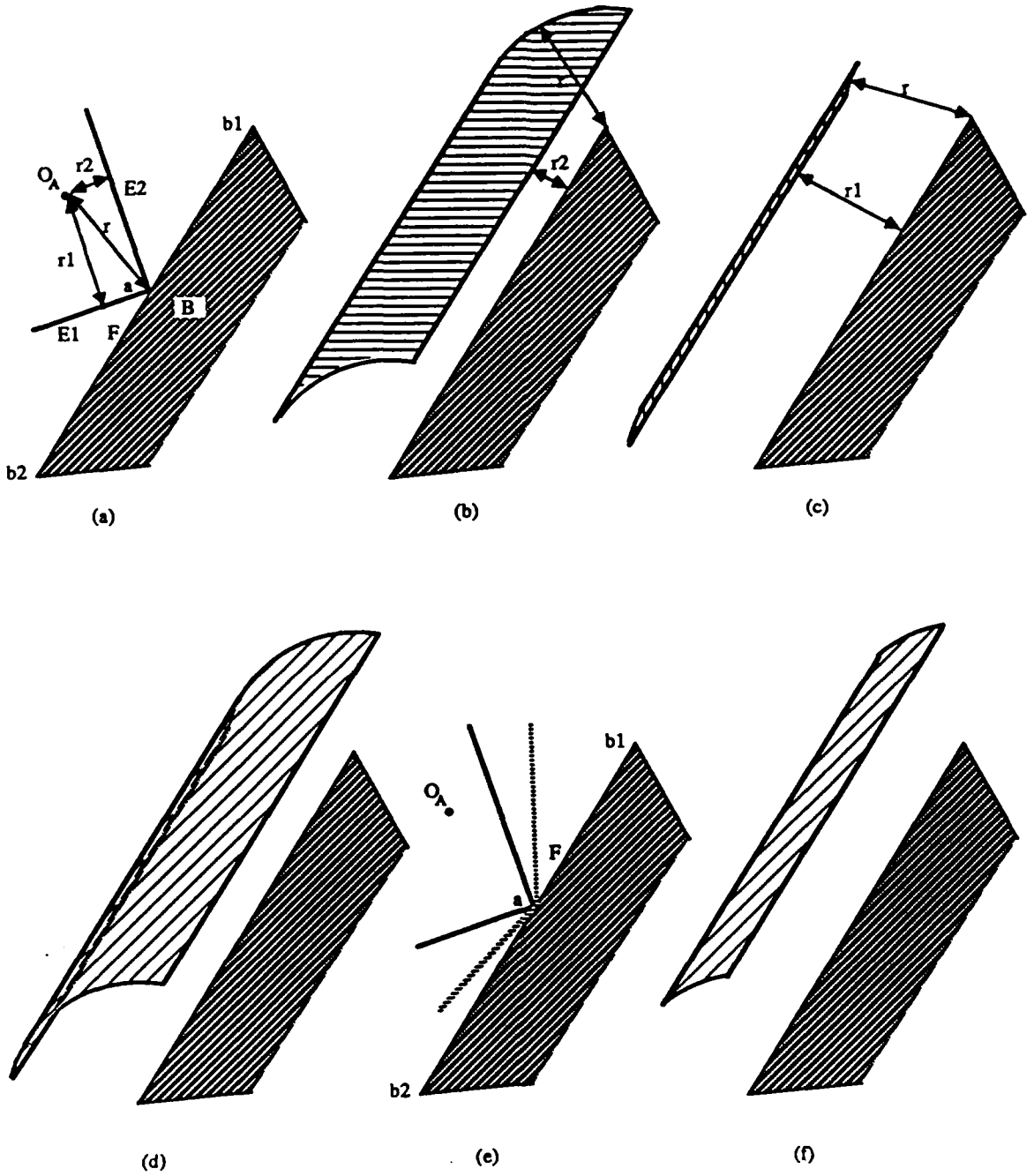

Figure 6. This figure illustrates the computation of the projection of a C-patch of Type B on the xy plane (see text).

locus of $O_{\mathcal{A}}$, when $\mathcal{A}$ translates and rotates with a staying in F and the orientation θ ranging over $[\psi, \phi_2]$. (If $\psi > \phi_2$, the region is empty.)

The case of e_B projecting for the orientation range $[\gamma_u, \gamma_{u+1}] \subset [\phi_1, \phi_2]$ can be treated in the same way, by drawing fictitious edges E_1 and E_2 from vertex a (see Figure 6 e and f), so that, when E_1 (resp E_2) is aligned with F, $\mathcal{A}$'s orientation is γ_u (resp. γ_{u+1}).

Computation of $\mathcal{OCB}_{xy}[\kappa^u]$ and $\mathcal{ICB}_{xy}[\kappa^u]$. The projection of every C-facet is a generalized polygon with a small number of edges – two to four straight line segments and two or three circular arcs. The union of all the generalized polygons forms a "donut" shaped region (see Figure 4).

Γ_1 and Γ_2 can be extracted by a line-sweep technique. This is a well-known technique in Computational Geometry (e.g., see [Preparata and Shamos, 1985]), whose application to the computation of the intersections of arbitrary generalized polygons is described in [Laumond, 1987]. It consists of sweeping a line across the plane. For example, the line is parallel to the x axis and is swept bottom-up. At each instant, the "status" of the line – i.e., the list of the intersections of the line with the generalized polygons – is represented in a balanced tree [Aho, Hopcroft and Ullman, 1983]. The status of the line changes in a qualitative fashion only at a finite number of ordinates, called "events", where the line is either tangent to a C-obstacle or passing through the intersection of two generalized polygons. At every event, the list of future events, which is also represented in a balanced tree, is updated.

The line-sweep process starts at the bottom-most ordinates of all the points in the generalized polygons. Below this ordinate, the line intersects no generalized polygon and thus lies entirely outside $\mathcal{OCB}_{xy}[\kappa^u]$. During the sweeping process, the contours Γ_1 and Γ_2 are tracked by labelling the intervals between the ordinates listed in the sweep-line status as being outside $\mathcal{OCB}_{xy}[\kappa^u]$, inside $\mathcal{OCB}_{xy}[\kappa^u]$ but outside $\mathcal{ICB}_{xy}[\kappa^u]$, or inside $\mathcal{ICB}_{xy}[\kappa^u]$.

Once Γ_1 and Γ_2 have been extracted in the form of sequences of straight segments and circular arcs, it is not difficult to clip them by the rectangle $[x_1, x_2] \times [y_1, y_2]$. The projections $\mathcal{OCB}_y[\kappa^{uv}]$ and $\mathcal{ICB}_y[\kappa^{uv}]$ are easily computed by determining the points of Γ_1 and Γ_2 at abscissae a_v, $v = 1, ..., s+1$, and the other extremal points of Γ_1 and Γ_2 within each interval $[a_v, a_{v+1}]$, $v = 1, ..., s$. In fact, all these computations can be done during line sweeping. If the interval $[x_1, x_2]$ is decomposed into sub-intervals, the sweep line has to be parallel to the x axis, otherwise it should be parallel to the y axis.

The overall line-sweep process takes time $O((n+m)\log n)$, where n is the number of C-facets that intersect with the interval $[\gamma_u, \gamma_{u+1}]$ and m is the number of intersections

of the generalized polygons. We know that $n \leq n_{\mathcal{A}} n_{\mathcal{B}}$ (see Subsection 2.2). On the other hand, $m \in O(n^2)$, but it is usually much smaller.

One way to improve the efficiency of the algorithm could be to restrict the line-sweep process to the rectangle $[x_1, x_2] \times [y_1, y_2]$. However, there seems to be no simple way of establishing the initial status of the sweeping line if it does not start at the bootom-most ordinate (or left-most abscissa). Nevertheless, the process can be stopped as soon as the sweeping line leaves the rectangle.

Generalization. If $\mathcal{A}$ is a non-convex polygon that can be decomposed into convex components, such that the interiors of all these components have a non-empty intersection, then the reference point can be selected within this intersection and the above method directly applies to each component taken separately.

If $\mathcal{A}$ is non-convex and cannot be represented as the union of overlapping convex components (for instance, it is a $\sqcup$-shaped object), the above method can still be applied, but with some changes. Since the reference point will be outside some of the convex components, the shape of the projection of the corresponding C-facets will be different, but not difficult to establish.

Avnaim and Boissonnat [Avnaim and Boissonnat, 1988] describe an algorithm of time complexity $O(n_{\mathcal{A}}^3 n_{\mathcal{B}}^3 \log n_{\mathcal{A}} n_{\mathcal{B}})$ for computing the description of the boundary of $\mathcal{CB}$, when both $\mathcal{A}$ and $\mathcal{B}$ are non-convex. In the case where $\mathcal{A}$ is non-convex, using this algorithm first and projecting the C-obstacles on the xy plane next, would probably be an efficient method. However, we have not implemented it.

3.6 Swept-Area Method

Principle. The swept-area method consists of first computing two areas swept out by $\mathcal{A}$ when it rotates about its reference point from orientation γ_u to orientation γ_{u+1}: the *outer swept area* denoted by $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$ and the *inner swept area* denoted by $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$. They are defined as follows:

$$\mathcal{OSA}[\gamma_u, \gamma_{u+1}] = \bigcup_{\theta \in [\gamma_u, \gamma_{u+1}]} \mathcal{A}(0,0,\theta) = \{(x,y) \,/\, \exists \theta \in [\gamma_u, \gamma_{u+1}] : (x,y) \in \mathcal{A}(0,0,\theta)\},$$

$$\mathcal{ISA}[\gamma_u, \gamma_{u+1}] = \bigcap_{\theta \in [\gamma_u, \gamma_{u+1}]} \mathcal{A}(0,0,\theta) = \{(x,y) \,/\, \forall \theta \in [\gamma_u, \gamma_{u+1}] : (x,y) \in \mathcal{A}(0,0,\theta)\}.$$

Then, the method regards both $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$ and $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$ as moving objects, which can only translate in the plane, and it maps the obstacle $\mathcal{B}$ in the configuration spaces, $\mathbf{R}^2$, of these objects. The mapping is obtained by "growing" $\mathcal{B}$ inversely to the shape of these two objects, leading to two regions $\mathcal{OB}[\gamma_u, \gamma_{u+1}]$ and $\mathcal{IB}[\gamma_u, \gamma_{u+1}]$ formally defined as follows (see Subsection 2.2):

$$\mathcal{OB}[\gamma_u, \gamma_{u+1}] = \mathcal{B} \ominus \mathcal{OSA}[\gamma_u, \gamma_{u+1}]$$
$$\mathcal{IB}[\gamma_u, \gamma_{u+1}] = \mathcal{B} \ominus \mathcal{ISA}[\gamma_u, \gamma_{u+1}].$$

It is shown in [Laumond, 1987] that the Minkowski sum (resp. difference) of two generalized polygons is a generalized polygon, which can be computed in time $O(n_1 + n_2)$, if the two input polygons are convex, and $O(n_1 n_2)$, otherwise. (n_1 and n_2 denote the number of edges of the input generalized polygons.)

The swept-area method returns two regions:

$$[x_1, x_2] \times [y_1, y_2] \times \mathcal{OB}[\gamma_u, \gamma_{u+1}]$$

and:

$$[x_1, x_2] \times [y_1, y_2] \times \mathcal{IB}[\gamma_u, \gamma_{u+1}].$$

The first is exactly $\mathcal{OCB}_{xy}[\kappa^u]$, as shown in [Lozano-Pérez, 1983]. The second is strictly included in $\mathcal{ICB}_{xy}[\kappa^u]$, which leads the overall decomposition algorithm to generate a set of FULL cells of less total volume than with the projection method.

In practice, the swept-area method is significantly more efficient than the projection method when the intervals $[\gamma_u, \gamma_{u+1}]$ are large. When these intervals become small enough, the projection method is preferred because it exactly computes $\mathcal{ICB}_{xy}[\kappa^u]$.

Computation of $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$. The outer swept area $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$ is a generalized polygon (see Figure 7). The straight edges of this polygon are portions of edges of $\mathcal{A}(0, 0, \gamma_u)$ and $\mathcal{A}(0, 0, \gamma_{u+1})$. Each circular edge is the locus of a vertex of $\mathcal{A}$, when $\mathcal{A}$ rotates from γ_u to γ_{u+1} about the reference point.

The contour of $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$ is traced out in time $O(n_{\mathcal{A}}^2)$ starting at the vertex the most distant from $O_{\mathcal{A}}$.

Computation of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$. The inner swept area $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$ is a generalized polygon (see Figure 8). The straight edges of this polygon are portions of the edges of $\mathcal{A}(0, 0, \gamma_u)$ and $\mathcal{A}(0, 0, \gamma_{u+1})$. Each circular edge is the locus of a point obtained by projecting the reference point on an edge of $\mathcal{A}$, if that projection falls between the two extremities of the edge.

The contour of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$ can be traced out in time $O(n_{\mathcal{A}}^2)$. The starting point is the first intersection of a ray (any one) drawn from the refence point with the potential edges of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$. (Since the reference point is in the interior of $\mathcal{A}$, it is also in the interior of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$.)

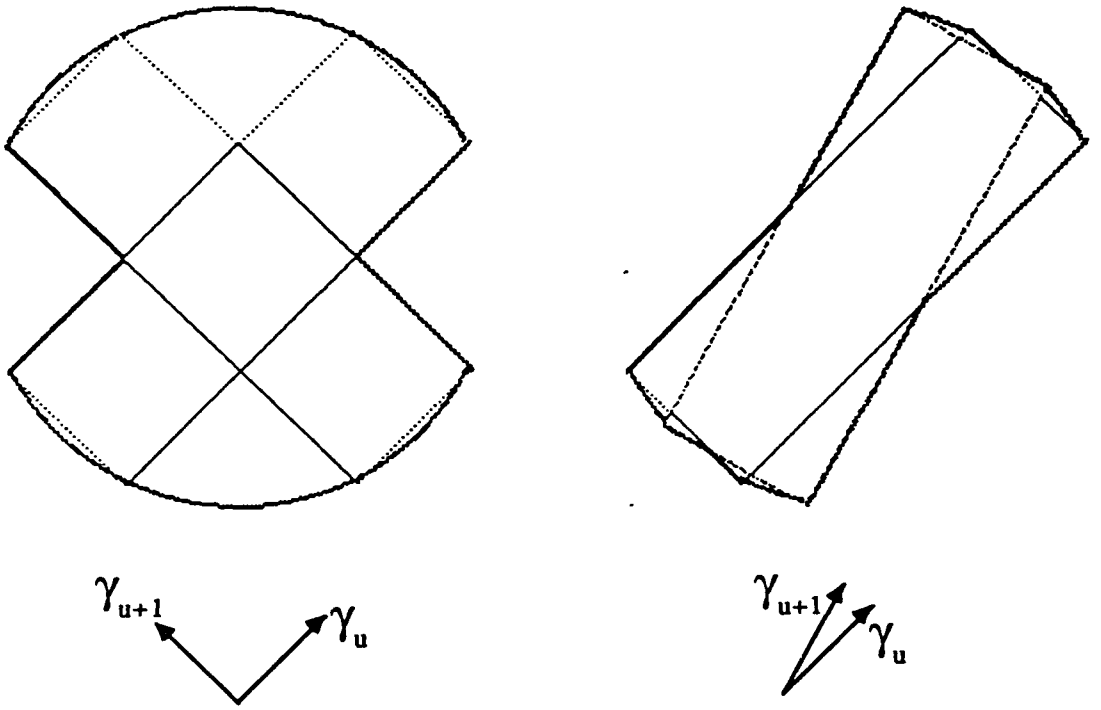

Figure 7. The contour of the outer swept area of $\mathcal{A}$ when it rotates about its reference point from orientation γ_u to orientation γ_{u+1} is made of portions of edges of $\mathcal{A}(0,0,\gamma_u)$ and $\mathcal{A}(0,0,\gamma_{u+1})$, and circular segments traced by vertices of $\mathcal{A}$ during the rotation.

4 Construction of a Channel

4.1 First-Cut Algorithm

Remember from Subsection 2.3 that a channel is a sequence of adjacent EMPTY or MIXED cells connecting the initial configuration $\mathbf{q}_{init} = (x_{init}, y_{init}, \theta_{init})$ to the goal configuration $\mathbf{q}_{goal} = (x_{goal}, y_{goal}, \theta_{goal})$. A channel is EMPTY, if it only contains EMPTY cells; otherwise, it is MIXED. Below, we call an EMPTY channel an E-channel, and a MIXED channel an M-channel.

A simple first-cut search algorithm for generating an E-channel is the following:

1. Generate a first partition $\mathcal{P}_0$ of $\mathcal{K}$. Construct the graph $\mathcal{CCG}_0$ corresponding to this decomposition. Set i to 0.

2. Search $\mathcal{CCG}_i$ for a channel. If an E-channel is found, return success. If no channel is found, return failure.

3. Let Π be the M-channel generated at Step 2. Set $\mathcal{P}_{i+1}$ to $\mathcal{P}_i$ and i to $i+1$. For

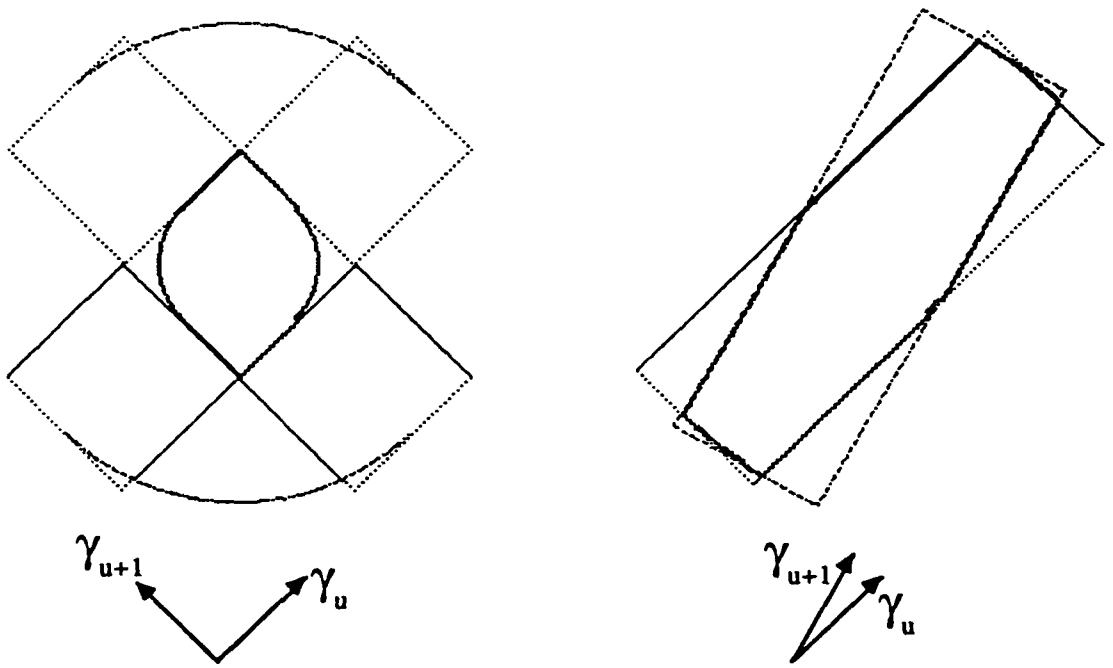

Figure 8. The contour of the inner swept area of $\mathcal{A}$ when it rotates about its reference point from orientation γ_u to orientation γ_{u+1} is made of portions of edges of $\mathcal{A}(0,0,\gamma_u)$ and $\mathcal{A}(0,0,\gamma_{u+1})$, and circular segments. These circular segments are traced by the projections of $O_{\mathcal{A}}$ on the edges of $\mathcal{A}$ during the rotation.

every MIXED cell κ in Π partition κ into a set $\mathcal{P}_\kappa$ of smaller cells and set $\mathcal{P}_i$ to $[\mathcal{P}_i \backslash \kappa] \cup \mathcal{P}_\kappa$. Goto Step 2.

This algorithm searches successive ccg's until an E-channel is found. Each ccg CCG_i, $i \neq 0$, is obtained from the the previous ccg, i.e. CCG_{i-1}, by only expanding some of the MIXED nodes, which belong to the M-channel generated in CCG_{i-1}.

The search for a channel in a ccg may be guided by various types of heuristics. In general, a ccg is not searched for an E-channel before it is searched for an M-channel. Indeed, although it is natural that the heuritics put an extra cost on MIXED cells in order to generate an E-channel quicker, it may also be appropriate to prefer shorter channels over longer ones (according to some metrics). Thus, although an E-channel may exist in a ccg CCG_i, it may be preferable to generate a significantly shorter M-channel instead, and refine CCG_i accordingly. Notice that any E-channel existing in CCG_i still exists in all its successors CCG_{i+j}, $j = 1, 2, ...$

4.2 Improved Algorithm

The major drawback of the simple first-cut algorithm given above is that the search work performed in CCG_i, if it does not return success, is not used to help the search of CCG_{i+1}. This drawback can be remedied as follows. Rather than reconstructing a full ccg, whenever MIXED cells along an M-channel are refined, a ccg representing the decomposition of every refined cell κ is generated separately and recursively searched for a "subchannel". This subchannel is a sequence of adjacent EMPTY or MIXED cells produced by the decomposition of κ.

The new algorithm hence generates a hierarchy of ccg's. The ccg at the top of the hierarchy corresponds to the initial decomposition of C – i.e., $\mathcal{K}$ – and is denoted by CCG_C. Every other ccg corresponds to the decomposition of a certain MIXED cell, say κ, and is denoted by CCG_κ. A channel Π is first generated in CCG_C. If Π is an E-channel, the planner exits with success; otherwise, each MIXED cell κ in Π is decomposed recursively, and a subchannel Π_κ, if any, is generated in CCG_κ. This subchannel is substituted for κ in Π.

In order to make the algorithm work properly, however, one must be careful that each subchannel Π_κ connects appropriately to the rest of Π [Kambhampati and Davis, 1986]. This can be worked out as explained below, by generating a complete channel connecting q_{init} to q_{goal} at every level of refinement.

Let Π^1 be an M-channel extracted from the top-level ccg, i.e. CCG_C. Set Π^2 to the empty sequence of cell. (Π^2 will be incrementally augmented into a refinement of Π^1.) Each cell κ in Π^1 is considered in the order it appears in Π^1. If κ is EMPTY, it is simply appended to the current Π^2. If κ is MIXED, it is first decomposed into a set $\mathcal{P}_\kappa$ of smaller cells and a ccg CCG_κ is built using this decomposition; then, CCG_κ is searched for a subchannel Π_κ satisfying the following four conditions:

- If κ is the first cell in Π^1 (hence, it contains q_{init}), then the first cell in Π_κ also contains q_{init}.
- If κ is the last cell in Π^1 (hence, it contains q_{goal}), then the last cell in Π_κ also contains q_{goal}.
- If κ is not the first cell in Π^1, the first cell of Π_κ is adjacent to the last cell of the current Π^2.
- If κ is not the last cell in Π^1, the last cell of Π_κ is adjacent to the cell following κ in Π^1.

In the following, a sequence of adjacent EMPTY or MIXED cells in the decomposition of κ is called a **subchannel** iff it satisfies these conditions.

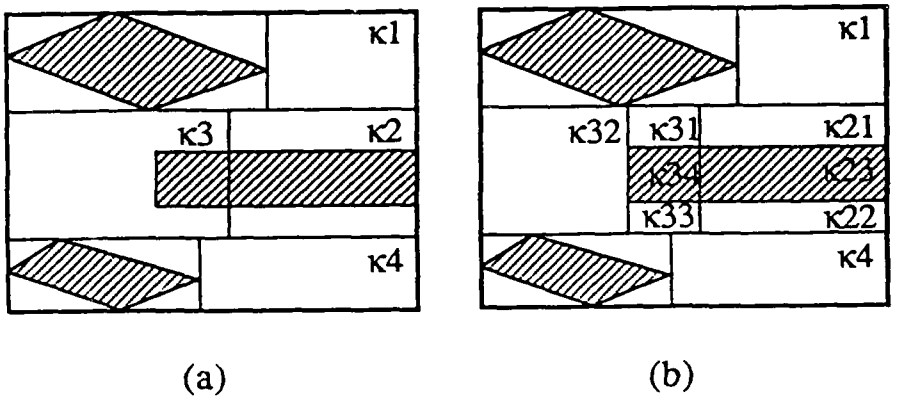

Figure 9. Figure a represents a decomposition of a rectangle. In order to ultimately construct an E-channel at a deeper level of decomposition, the planner must consider the sequence $(\kappa_1, \kappa_2, \kappa_3, \kappa_2, \kappa_4)$, which contains the cell κ_2 twice, as an M-channel. This leads the planner to decompose the two MIXED cells κ_2 and κ_3 into smaller cells (for example, as in b) and generate the E-channel $(\kappa_1, \kappa_{21}, \kappa_{31}, \kappa_{32}, \kappa_{33}, \kappa_{22}, \kappa_4)$.

Assuming that the planner succeeds in generating Π_κ, Π^2 is modified by appending Π_κ to it. The case when the planner fails in generating Π_κ will be examined in Subsection 4.4.

When all the cells in Π^1 have been considered successfully, we have obtained a channel Π^2, which is a refinement of Π^1. If Π^2 is an E-channel, the planning problem is solved; the planner returns Π^2 and success. Otherwise, Π^2 is recursively refined into a third channel Π^3 by decomposing the MIXED cells it contains. Etc...

4.3 Cell Occurrences in a Channel

An E-channel is now generated by refining an M-channel without reproducing a global ccg. Therefore, it is important that we allow an M-channel to contain several times the same MIXED cell (i.e., loops). Indeed, different occurrences of the same MIXED cell may lead to different subchannels. This need is better explained using an example.

Consider the two-dimensional example of Figure 9. In a, the sequence of cells $(\kappa_1, \kappa_2, \kappa_3, \kappa_2, \kappa_4)$, which contains κ_2 twice, has to be considered as an M-channel. (The grey area represents C-obstacles, so that κ_2 and κ_3 are MIXED, while κ_1 and κ_4 are EMPTY.) We assume that κ_2 and κ_3 are decomposed as shown in Figure 9 b. When the two occurrences of κ_2 are refined they produce different EMPTY sub-

channels. This is illustrated in Figure 9 b, where κ_2 is partitioned into three smaller cells, two EMPTY ones (κ_{21} and κ_{22}) and a FULL one (κ_{23}). The first occurrence of κ_2 leads to a one-cell subchannel made of κ_{21}, while the second occurrence leads to another one-cell subchannel made of κ_{23}. On the other hand, the decomposition of κ_3 produces four cells, three EMPTY ones, κ_{31}, κ_{32}, and κ_{33}, and a FULL one, κ_{34}. From this decomposition the EMPTY subchannel $(\kappa_{31}, \kappa_{32}, \kappa_{33})$ is extracted. The generated E-channel is $(\kappa_1, \kappa_{21}, \kappa_{31}, \kappa_{32}, \kappa_{33}, \kappa_{22}, \kappa_4)$.

Since the same cell may appear several times in a channel Π^k, we now refer to the elements in a channel as **cell occurrences** rather than just cells. We denote by ω_i^k the ith cell occurrence in Π^k and $cell(\omega)$ the cell of which ω is an occurrence. A cell occurrence ω is said MIXED (resp. EMPTY) iff the cell $cell(\omega)$ is MIXED (resp. EMPTY). A subchannel generated as a refinement of a MIXED cell occurrence ω is denoted by Π_ω; it is constructed by searching $CCG_{cell(\omega)}$.

In Subsection 4.6, we will examine the impact of allowing loops in a channel on the search of a ccg.

4.4 Failure Recovery

Consider the situation when the algorithm of Subsection 4.2 fails to refine a MIXED cell occurrence ω_i^k into a subchannel. We distinguish three cases:

Case (1):

$i = 1$. Hence, $\omega_i^k = \omega_1^k$ is the first cell occurrence in Π^k. The search proceeds by generating a new channel Π^k, i.e. Π^{k-1} is re-considered and refined into a new channel Π^k. If no new channel Π^k can be generated and $k = 1$, the planner returns failure. If no new channel Π^k can be generated and $k \neq 1$, the planner recursively attempts to generate a new channel Π^{k-1}.

Case (2):

$i > 1$ and ω_{i-1}^k is EMPTY. The search proceeds as described in case (1). (Some differences will appear in the next subsection.)

Case (3):

$i > 1$ and ω_{i-1}^k is MIXED. The search proceeds by generating another subchannel $\Pi_{\omega_{i-1}^k}$, if any, in the ccg of $cell(\omega_{i-1}^k)$. If another subchannel exists in this ccg, the planner substitutes it for the previously generated one in Π^{k+1}, and tries again to refine ω_i^k into a subchannel by searching in the ccg of $cell(\omega_i^k)$. Otherwise, it iteratively treats ω_{i-1}^k as it just treated ω_i^k. (Either (1), (2), or (3) applies to ω_{i-1}^k.)

4.5 Recording Failure Conditions

One can organize (sub)channel generation and failure recovery in a systematic fashion in order to avoid infinite looping. However, this may not prevent the planner to make the same mistakes several times. Higher efficiency can be obtained by remembering the (weakest) conditions of the failures in order to avoid reproducing them [Stallman and Sussman, 1977] [Latombe, 1979]. One way to remember failure conditions is to annotate cells and use the annotations as follows. Cases (1), (2) and (3) below refer to the same cases as in the previous subsection. n_k denotes the number of cell occurrences in Π^k.

Case (1):

Let $\xi = cell(\omega_1^k)$ and $\psi = cell(\omega_2^k)$ in the current Π^k. If $n_k = 1$, by convention $\psi = \lambda$, where λ denotes the inexisting cell. The cell ξ is annotated with $A1 : [\psi]$, which is called a *type 1 annotation.*

This annotation is later used as follows: If a new channel Π^k is generated, which contains an occurrence ω_ξ of ξ in first position, then an occurrence of ψ should not be considered a valid successor of ω_ξ. In the particular case where $\psi = \lambda$, this means that ω_ξ must not be the last cell occurrence of the new Π^k.

Notice that the first cell occurrence in the new Π^k is necessarily an occurrence of ξ, since ξ contains $\mathbf{q_{init}}$. But there may be other occurrences of ξ appearing in this new Π^k, to which the above annotation does not apply.

Case (2):

Let $\xi = cell(\omega_i^k)$ $(i > 1)$, $\varphi = cell(\omega_{i-1}^k)$ and $\psi = cell(\omega_{i+1}^k)$ in the current Π^k. If $i = n_k$, then $\psi = \lambda$ (with the same meaning of λ as above). ξ is annotated with $A2 : [\varphi, \psi]$, which is called a *type 2 annotation.*

This annotation is later used as follows: If a new channel Π^k is generated, which contains an occurrence ω_ξ of ξ preceded by an occurrence of φ, then an occurrence of ψ should not be considered a valid successor of ω_ξ. In the particular case where $\psi = \lambda$, this means that ω_ξ must not be the last cell occurrence of the new Π^k.

Notice that if $\psi \neq \lambda$ the construction is symmetrical, i.e., if ξ cannot be traversed by a subchannel connecting φ to ψ, it can neither be traversed by a subchannel connecting ψ to φ. Therefore, the annotation $A2 : [\varphi, \psi]$ is later used commutatively.

Case (3):

In this case, failure conditions are more involved than in cases (1) and (2), and may require sequences of cells of arbitrary length to be considered. The cell $\xi = cell(\omega_{i-1}^k)$ is annotated with an expression of the form $A3 : [\varphi, (\psi_1 \psi_2 ... \psi_q)]$, called

type 3 annotation, whose construction will be explained below.

This annotation is later used as follows: If a subchannel Π_{ω_ξ} is generated (by searching CCG_ξ) as a refinement of an occurrence ω_ξ of ξ, while ω_ξ is followed in the current channel Π^k by a sequence $\omega_1, \omega_2, ..., \omega_q$ of cell occurrences, such that for all $j \in [1,q]$ $cell(\omega_j) = \psi_j$ (if $\psi_q = \lambda$, the condition $cell(\omega_q) = \lambda$ requires that ω_{q-1} be the last cell occurrence in the current Π^k), then the last cell occurrence in Π_{ω_ξ} must not be an occurrence of φ.

Let us now examine how the annotation was built. Let $\psi_1 = cell(\omega_i^k)$ and $\psi_2 = cell(\omega_{i+1}^k)$, when the planner failed to refine ω_i^k. If $n_k = i$, then $\psi_2 = \lambda$. Let also φ be the last cell occurrence in the (incomplete) current Π^{k+1}, i.e. φ is the last cell occurrence of the subchannel $\Pi_{\omega_{i-1}^k}$ previously generated. For every type 3 annotation $A3 : [\varphi', (\psi'_1 ... \psi'_p)]$ attached to ψ_1, if ω_i^k is followed in the current Π^k by the sequence $\omega_{i+1}^k, ..., \omega_{i+p}^k$, with $cell(\omega_{i+j}^k) = \psi'_j$ for all $j \in [1,p]$ and $i + p = k_k + 1$ if $\psi'_p = \lambda$, ξ is annotated with $A3 : [\varphi, (\psi_1 \psi_2 \psi_{p+1})]$, where $\psi_{j+1} = \psi'_j$ for all $j \in [1,p]$. If no type 3 annotation has been attached to ξ during this iteration (this happens in particular if no type 3 annotation is currently attached with ψ_1), then ξ is annotated with $A3 : [\varphi, (\psi_1 \psi_2)]$.

4.6 Search of a Cell-Connectivity Graph

The planning algorithm builds a hierarchy of ccg's and searches each of these ccg's separately. The ccg at the top of the hierarchy is CCG_C and is searched to generate the channel Π^1. Each other ccg CCG_κ represents the decomposition $\mathcal{P}_\kappa$ of a cell κ which belongs to the parent ccg of CCG_κ in the ccg hierarchy.

Remember that C is just a particular cell $\mathcal{K}$. We denote by Π^0 the channel that only contains ω_1^0, with $cell(\omega_1^0) = \mathcal{K}$. Π^1 is a refinement of Π^0. In the very particular case where there is no C-obstacle in the configuration space, then $\mathcal{K}$ is an EMPTY cell, and Π^0 is an E-channel, so that Π^1 does not have to be generated.

Let us consider that we have generated an M-channel Π^k ($k \geq 0$), which we now refine into Π^{k+1}, and that we are currently considering the MIXED cell occurrence $\omega_i^k \in \Pi^k$. Let $\kappa = cell(\omega_i^k)$. If κ has already been decomposed, we just reuse the ccg CCG_κ that was previously generated; otherwise, we decompose κ into a set $\mathcal{P}_\kappa$ of smaller cells using the method described in Section 3 and we build CCG_κ. We then search CCG_κ for a subchannel that will be appended to the tail of the current Π^{k+1}.

The search of CCG_κ consists of first determining the possible initial and goal cells of the subchannel to be generated:

- If $i = 1$, ω_i^k is the first cell of Π^k. Then the only possible initial cell in CCG_κ is the cell of $\mathcal{P}_\kappa$ that contains q_{init}. Otherwise, the possible initial cells of CCG_κ are all

the cells of $\mathcal{P}_\kappa$ which are adjacent to the cell – call it φ – of the last cell occurrence ω_φ in the current Π^{k+1} and whose insertion in Π^{k+1} as cell occurrences succeding ω_φ does not violate any annotation of type 1 or 2 attached to φ.

- If $i = n_k$, ω_i^k is the last cell occurrence of Π^k. The only possible goal cell in $\mathcal{CCG}_\kappa$ is the cell of $\mathcal{P}_\kappa$ that contains $\mathbf{q}_{goal}$. Otherwise, the possible goal cells of $\mathcal{CCG}_\kappa$ are all the cells of $\mathcal{P}_\kappa$ which are adjacent to $cell(\omega_{i+1}^k)$ and do not violate any annotation of type 3 attached to κ.

If there is no possible initial or goal cell in $\mathcal{CCG}_\kappa$, the planner considers that situation as a failure to refine ω_i^k and applies the treatment described in Subsections 4.4 and 4.5.

If at least one initial and one goal cell are established in $\mathcal{CCG}_\kappa$, the planner searches that graph for a subchannel $\Pi_{\omega_i^k}$ linking any of the initial cells to any of the goal cells. It does that in a way such that no annotation of type 1 or 2 attached to the cells in the generated subchannel is violated. (If $i > 1$, checking the annotations of type 2 attached to the first cell occurrence in this subchannel requires the cell of the last cell occurrence in the current Π^{k+1} to be considered.)

As noted previously, the subchannel $\Pi_{\omega_i^k}$ to be generated in $\mathcal{CCG}_\kappa$ should be allowed to contain several occurrences of any MIXED cell in $\mathcal{P}_\kappa$. However, it seems quite realistic in an implementation to limit the number of occurrences of the same cell in a subchannel[7].

Since there may be multiple occurrences of the same cell in a subchannel, the actual graph that is searched to refine a cell occurrence ω_i^k is not $\mathcal{CCG}_\kappa$, but a graph derived from it. This graph contains the subset of subchannels (with and without loops) in $\mathcal{CCG}_\kappa$, which do not include two occurrences of the same EMPTY cell or more occurrences of the same MIXED cell than a predefined number (see paragraph above). This search graph is made (partly) explicit while it is searched. Since a MIXED cell is adjacent to a finite number of other cells, the search graph is finite. Hence its search is guaranteed to terminate, either with success or with failure. If it terminates with success the generated subchannel $\Pi_{\omega_i^k}$ is appended to Π^{k+1}. If it terminates with failure, the treatment described in Subsections 4.4 and 4.5 is applied. Thanks to the annotations attached to the cells, the search cannot iterate for ever through the same sequence of failures.

[7] In our implementation, this number is not bounded. However, we apply a best-first search strategy, which always considers the (sub)channels of minimal "cost" first. The cost of a (sub)channel grows with its length and the number of MIXED cells it contains.

4.7 Boundary-Connectivity Graph

Cell-connectivity graphs are one way to represent connectivity among cells. Another way is to build a graph called **boundary-connectivity graph**, or **bcg**. Given a partition $\mathcal{P}_\kappa$ of a cell κ into non-overlapping cells, the bcg $\mathcal{BCG}_\kappa$ is defined as follows:

- Each node of $\mathcal{BCG}_\kappa$ corresponds to the intersection of the boundaries of two adjacent cells.
- Two nodes of $\mathcal{BCG}_\kappa$ are connected by a link iff the two nodes are portions of the boundary of the same cell.

Each link in a bcg can be regarded as the connection between two cells (each cell includes the boundary portion associated with one extremity of the link) through a third one.

For the same cell κ, the size of $\mathcal{BCG}_\kappa$ is larger than $\mathcal{CCG}_\kappa$'s size. However, bcg's presents some advantages over ccg's because they represent more explicitely the connectivity of the cells in $\mathcal{P}_\kappa$. In particular, using bcg's removes the need for type 1 and 2 annotations. Indeed, attaching a type 1 or 2 annotation to a cell in a ccg is equivalent to removing a link in a bcg. However, in order to use bcg's properly one must be careful about various details. In particular, the initial and goal configurations should appear as nodes in some of the bcg's. Moreover, since type 2 annotations may involve two cells of two different partitions $\mathcal{P}_{\kappa_1}$ and $\mathcal{P}_{\kappa_2}$, bcg's should also be connected among them.

One possible way to use bcg's is sketched below. Every M-channel Π^{k-1} $(k > 0)$ is still a sequence of cell occurrences. When it is refined into Π^k, an *extended bcg* $\mathcal{BCG}^k$ is constructed as the bcg of all the cells which are either EMPTY cells in Π^k or cells in the decomposition $\mathcal{P}_\kappa$ of a MIXED cell κ in Π^k. In addition, q_{init} and q_{goal} are included in $\mathcal{BCG}^k$ are nodes. The node corresponding to q_{init} (resp. q_{goal}) is connected by a link to every node representing a boundary portion of the cell containing q_{init} (resp. q_{goal}). When Π^k is refined, parts of $\mathcal{BCG}^k$ may already exist and are simply re-used. The links having q_{init} as one extremity are called *initial links*. The links having q_{goal} as one extremity are called *goal links*. The links contained in the bcg $\mathcal{BCG}_\kappa$ of a cell κ are called *intra-links*. All the other links are called *inter-links*.

The refinement of Π^{k-1} into Π^k is essentially done in the same way as before. The only difference is that rather than searching the ccg of the MIXED cell occurrences along Π^{k-1}, the planner now searches the portion of $\mathcal{BCG}^k$ that corresponds to the bcg of the MIXED cell occurrence being currently considered. The initial links, goal links, and inter-links are used only to determine the possible initial and goal nodes of this search. Case (1) and (2) failures lead to remove some links from $\mathcal{BCG}^k$. Annotations of type 3 are still needed and used as before.

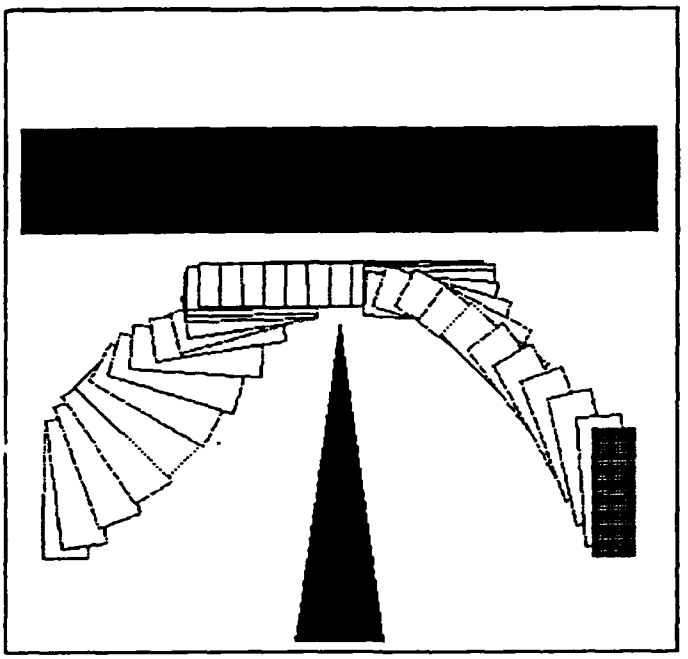

Figure 10. Example 1.

5 Implementation and Experimentation

We have implemented the techniques described in the above sections in a path planner. The planner is written in Allegro Common Lisp and runs on an Apple Macintosh II computer.

We ran the planner on many examples including both convex and non-convex moving objects. Twelve of these examples are illustrated in Figures 10 through 21. In each figure, we show a path extracted from the generated channel. Statistics – CPU time, total number of cells generated (N_{total}), number of EMPTY cells generated (N_{EMPTY}), number of EMPTY cells used in the output channel (N_{used}), and decomposition efficiency measure (E) – characterizing the efficiency of the planner are given in Table 1.

Examples 4 and 8 have been previously reported in [Brooks and Lozano-Perez, 1982] (BLP planner), the best previous planner known to the authors using the hierarchical approximate cell decomposition approach. In Table 2, we give the comparison of the statistics on these two examples with our planner and BLP planner[8]. We should point out that for Example 4 the path generated by our planner is "simpler" than the

[8]The BLP planner was implemented on a LISP machine, which runs LISP approximately 5 times faster than the Macintosh II runs Allegro Common Lisp (according to our own benchmark). In addition, since we do not count the number of FULL cells in our planner's statistics (they have no effect on the search graph), we subtracted this number from the statistics of the BLP planner.

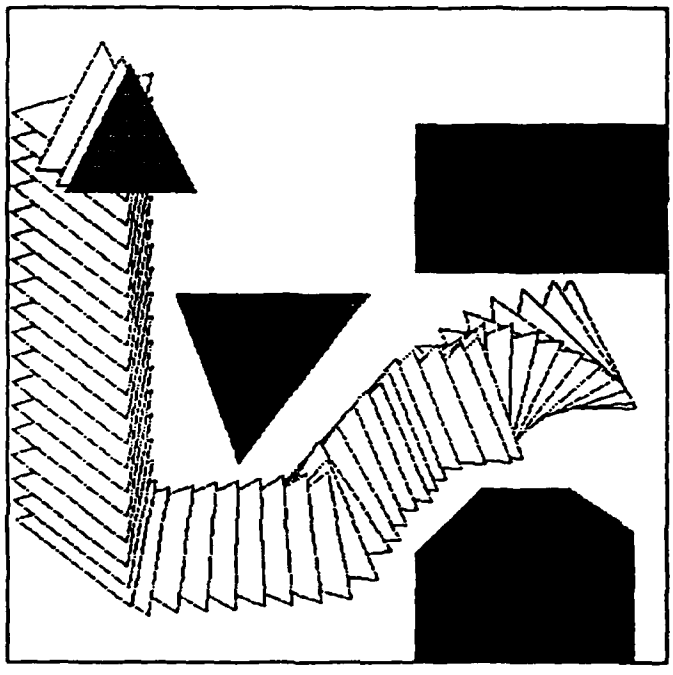

Figure 11. Example 2.

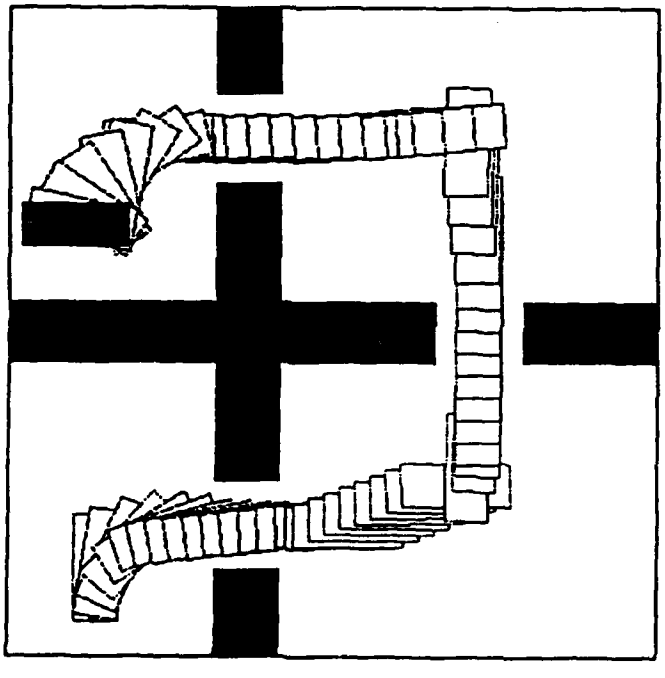

Figure 12. Example 3.

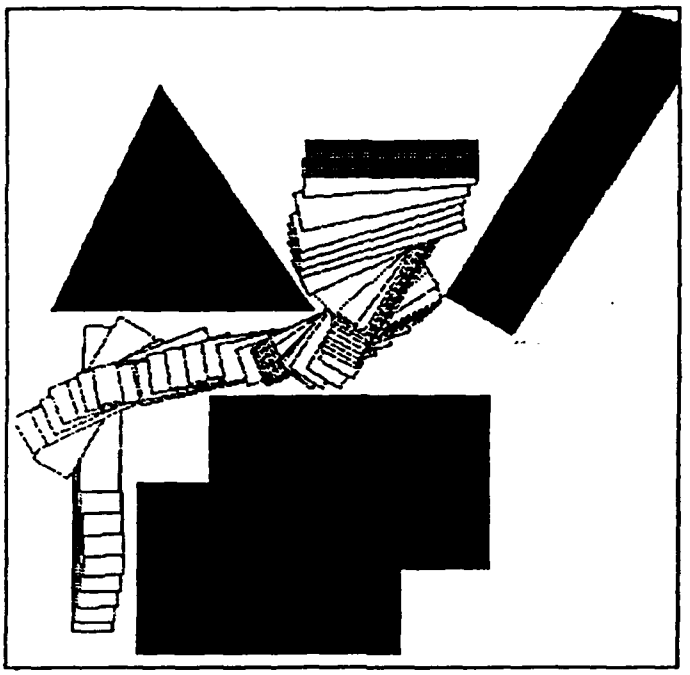

Figure 13. Example 4.

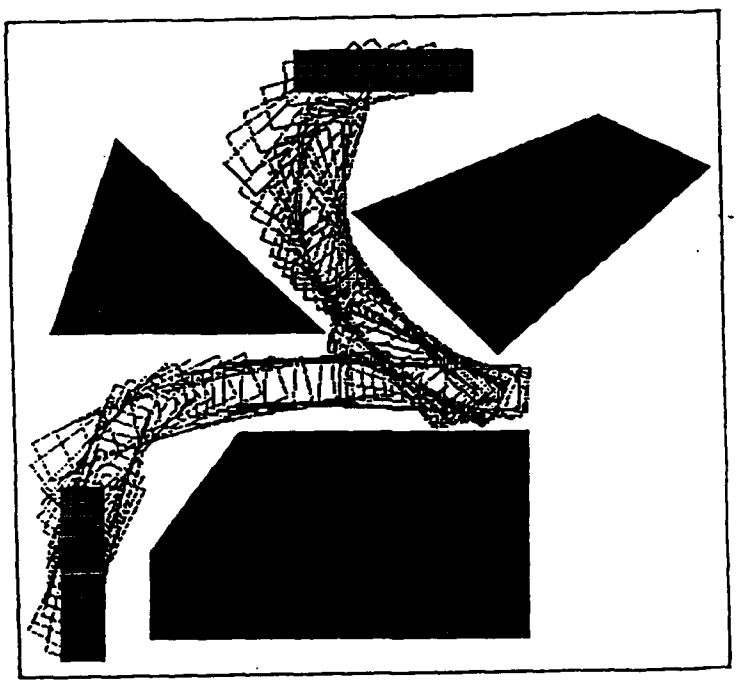

Figure 14. Example 5.

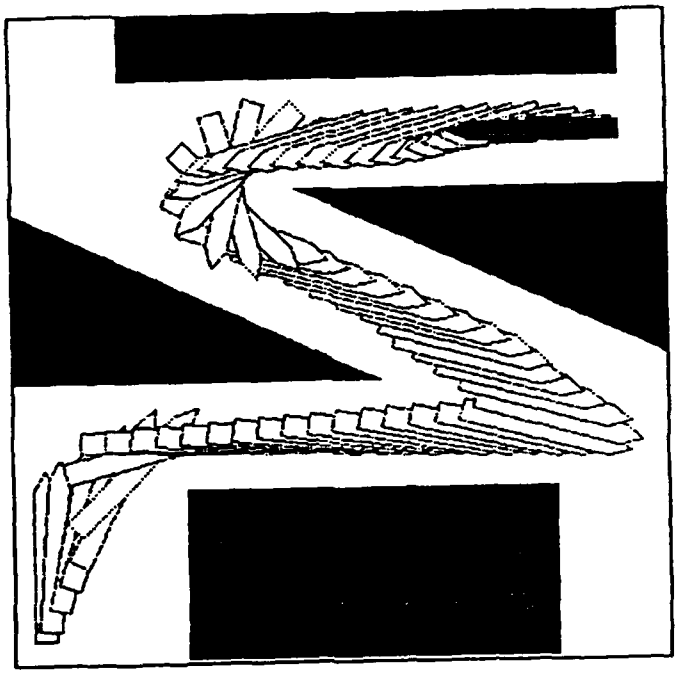

Figure 15. Example 6.

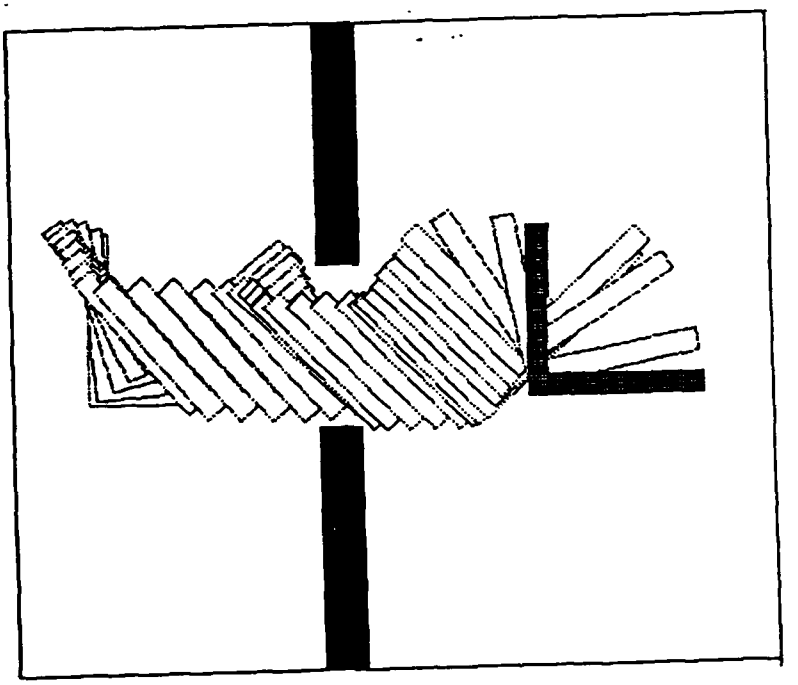

Figure 16. Example 7.

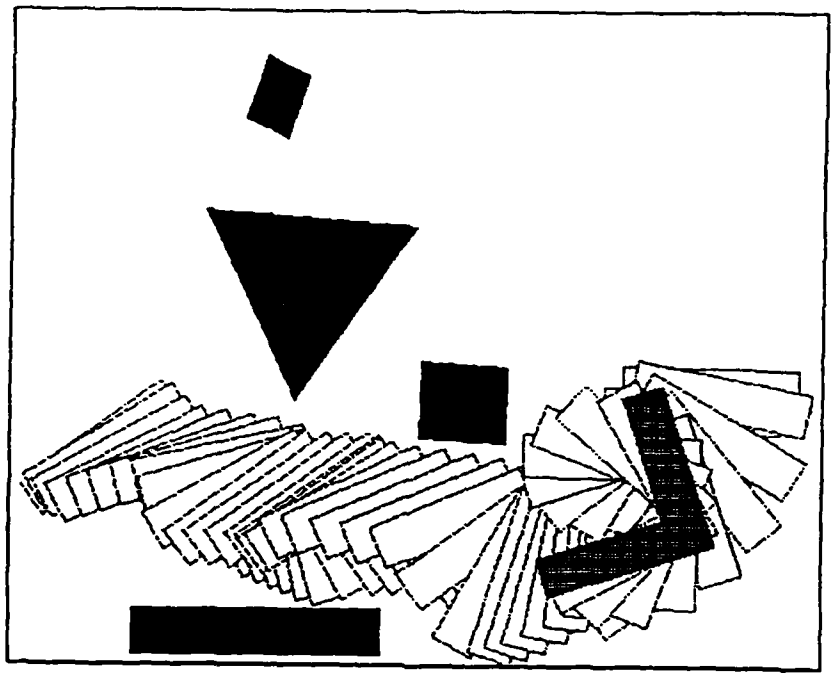

Figure 17. Example 8.

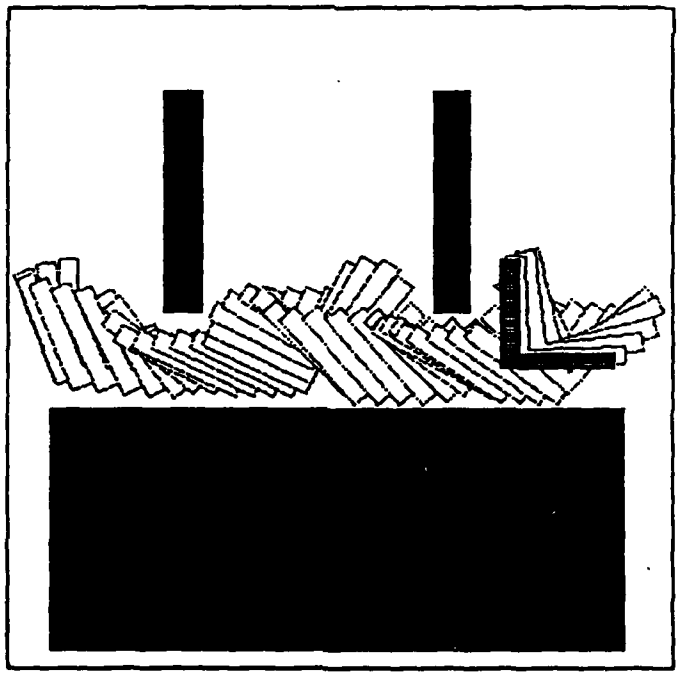

Figure 18. Example 9.

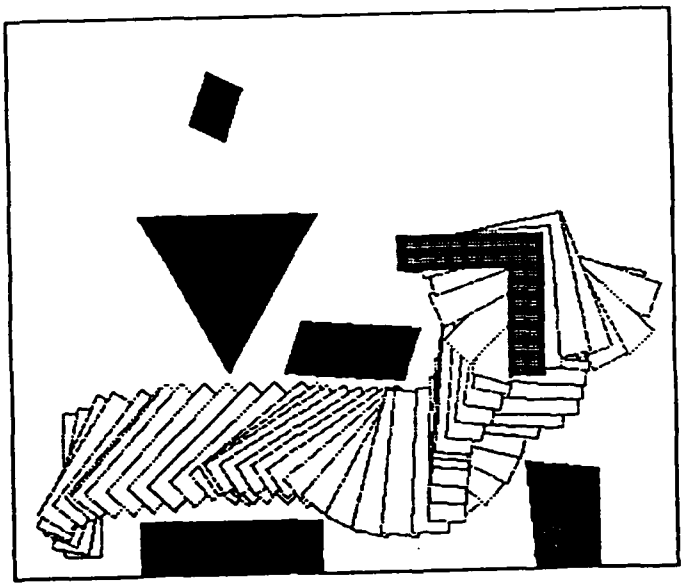

Figure 19. Example 10.

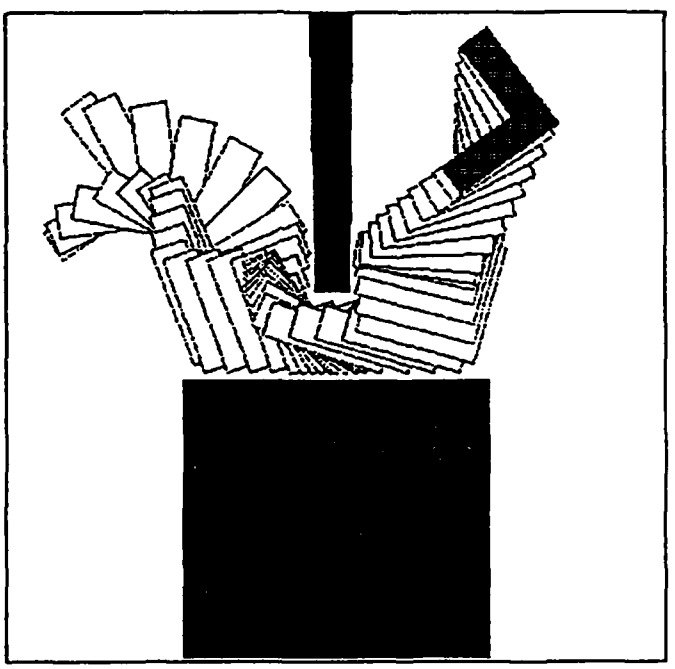

Figure 20. Example 11.

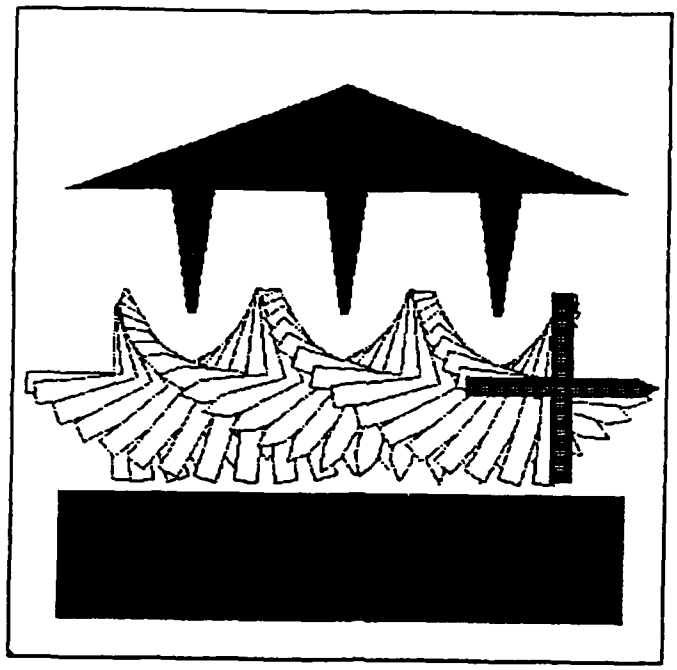

Figure 21. Example 12.

path generated by BLP planner, in the sense that it does not include a backing-up maneuver.

For examples 2, 6, 11, and 12, we compared the number of cells generated by our planner with the number of cells that a similar planner using the octree decomposition technique (see Subsection 3.2) would generate. Based on partial implementation, we obtained a conservative estimate (i.e., lower bound) of the number of cells generated by the octree decomposition technique. The data are reported in Table 3. Example 11 was extracted from [Avanaim, Boissonnat and Faverjon, 1988]. According to this paper, the planner described in [Faverjon, 1984], which uses the octree decomposition technique, generates more than 10,000 cells.

Therefore, the above results demonstrate major improvements brought by our algorithms to the approximate cell decomposition method.

In addition to the previous experiments, we tried to characterize empirically the efficiency of the planner as a function of the "complexity" of the workspace – i.e., the number of edges of the obstacles – and the "sparsity" of the workspace – i.e., the distance between obstacles. (The "clutteredness" of the workspace is the inverse of the sparsity.) To that purpose we run our planner on several hundreds problems using randomly generated environments with pre-specified complexity and sparsity. Figure 22 shows twelve of these examples. The vertical axis points down toward higher complexity and the horizontal axis points right toward smaller sparsity (i.e.,

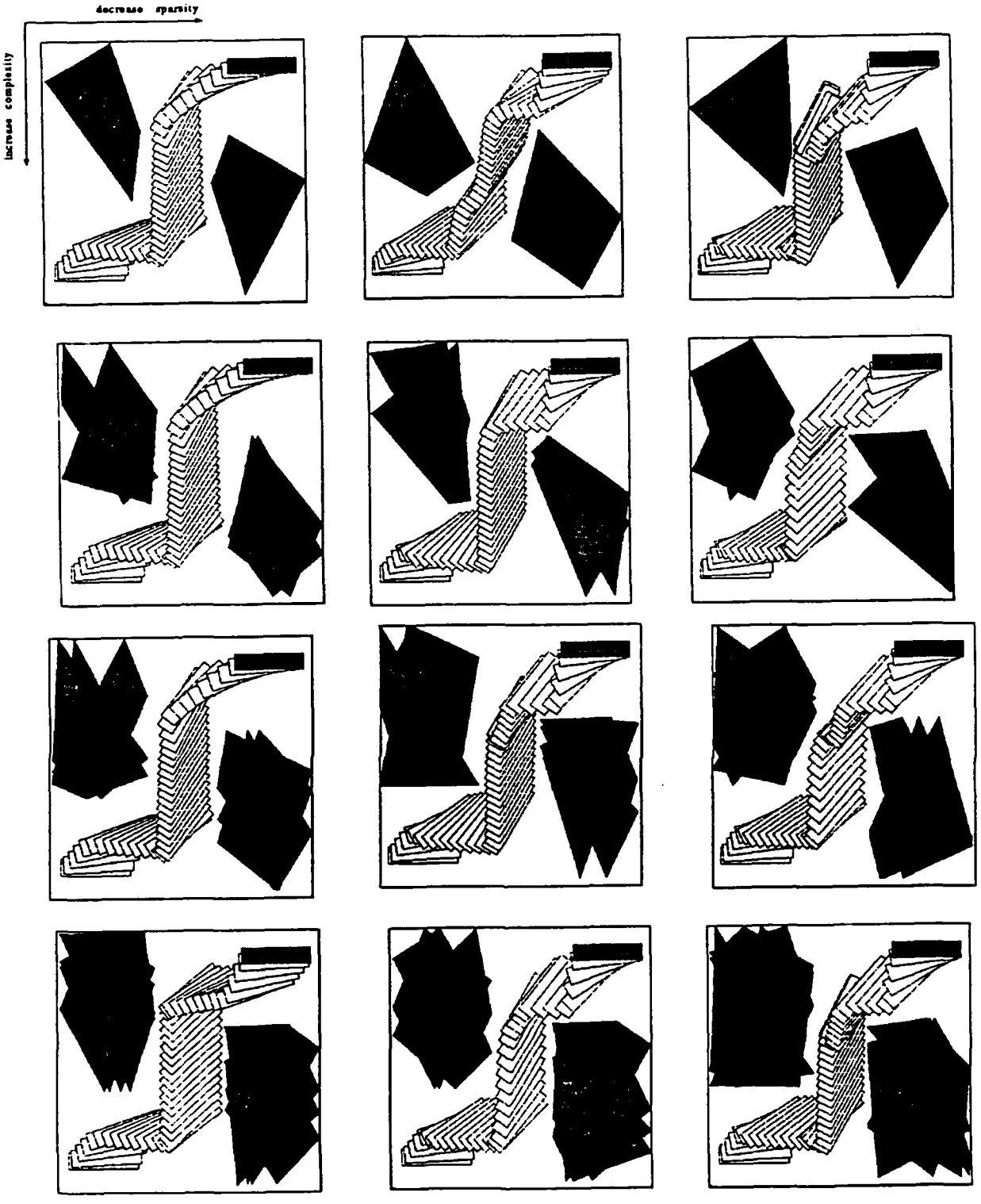

Figure 22. We ran our planner on many randomly generated examples with different complexity – number of edges – and sparsity – distance between obstacles – of the workspace. The purpose was to gather statistics about the effiency of the planner as a function of these parameters.

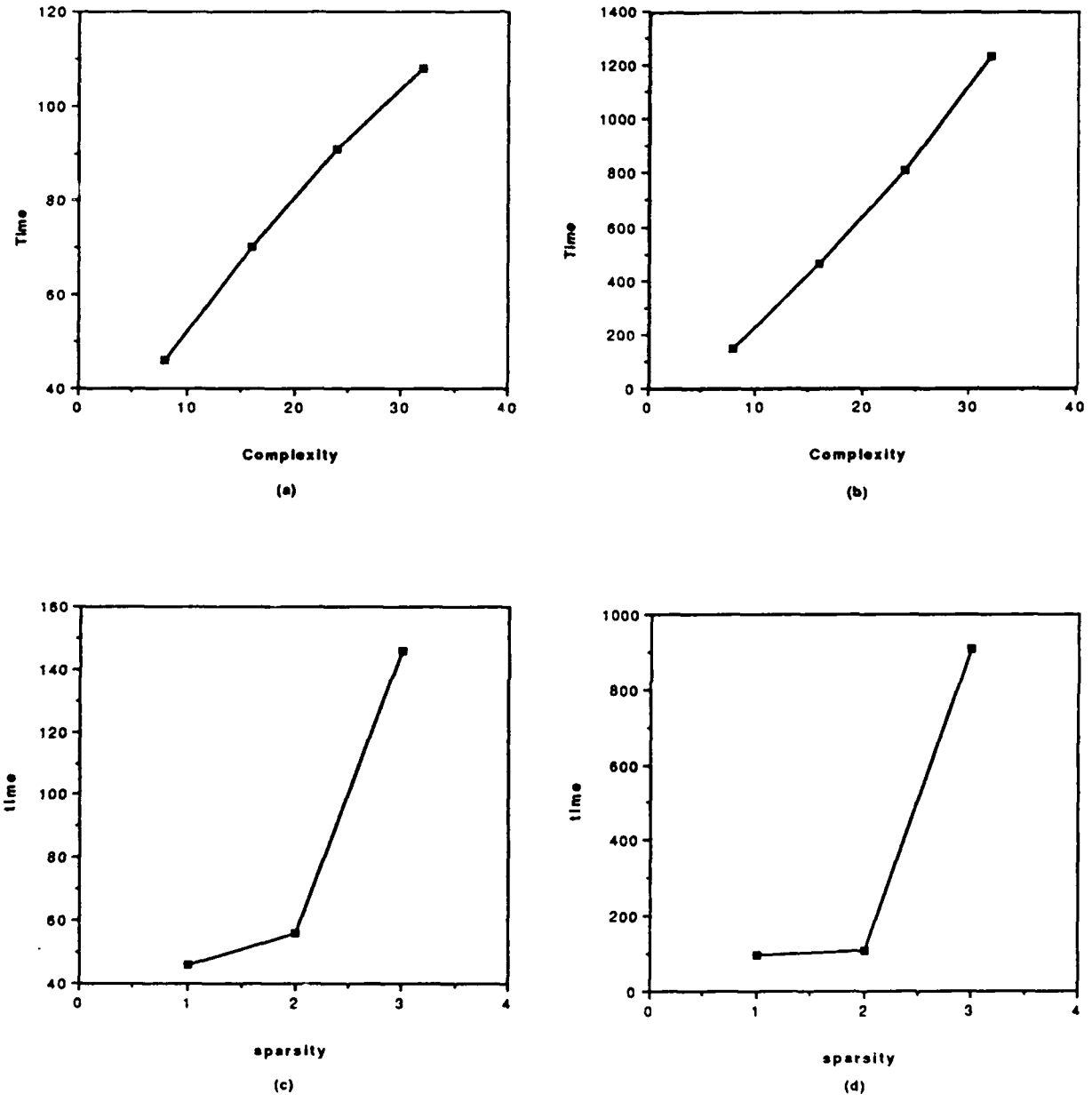

Figure 23. These diagrams illustrate the variation of the running time of the planner as a function of the complexity (a and b) and the sparsity (c and d) of the workspace.

example	*CPU time (min)*	N_{total}	N_{EMPTY}	N_{used}	E (×1000))
1	0.6	98	35	12	9.4
2	0.9	140	74	18	8.0
3	1.5	210	104	10	6.9
4	5.5	293	96	36	4.8
5	6.0	324	114	42	4.9
6	5.0	218	116	29	5.3
7	0.9	160	77	22	12.0
8	2.5	205	88	17	6.1
9	6.5	170	25	13	9.8
10	9.8	206	46	19	2.1
11	11.0	312	72	21	7.9
12	10.5	369	121	18	4.3

Table 1: **Statistics for the Twelve Examples**

	Example 4				*Example 8*			
planner	*CPU time*	N_{total}	N_{EMPTY}	N_{used}	*CPU time*	N_{total}	N_{EMPTY}	N_{used}
Ours	5.5	293	96	36	2.5	205	88	17
BLP	*tens*	644	120	87	*tens*	782	62	29

Table 2: **Comparision with the BLP Planner**

greater clutteredness). (The two obstacles in the workspace are enclosed within two rectangular boxes. The sparsity is measured as the distance between the vertical sides of these two boxes.) For each complexity and sparsity, we have generated and run tens of examples. The mean value of the running time obtained for some series of examples are depicted in Figure 23. Figure 23 a corresponds to the first column of Figure 22, Figure 23 b to the third column, Figure 23 c to the first row, and Figure 23 d to the third row. Figures 23 a and b show that, within the range of our experiments, the running time varies linearly with the complexity. Figures 23 c and d show that above some clutteredness, the running time increases very quickly. This results from the fact that the planner has to generate many small cells before it has any chance to find an E-channel.

6 Conclusion

We have described new heuristic algorithms for path planning using the hierarchical approximate cell decomposition approach. An important feature of this approach is that it decomposes the robot's configuration space into cells of predefined shape – typically, rectangloids – at successive levels of approximation. The two main steps of

Num cells	*example2*	*example6*	*example11*	*example12*
Ours	140	218	312	389
Octree	> 500	> 2000	> 5000	> 5000

Table 3: **Comparison with the Octree Method**

the approach are cell decomposition and graph searching.

With respect to cell decomposition, we have proposed an algorithm that decomposes a MIXED rectangloid cell into a collection of EMPTY, FULL and MIXED rectangloid cells. It attempts to minimize the total volume of the generated MIXED cells, while producing a reasonably small number of cells. Our algorithm is based on the construction of a bounding and a bounded rectangloid approximations of the C-obstacles. Experience shows that it produces much better results than with earlier decomposition techniques. Additional improvements could be possible, for example by allowing cells having more involved shapes. However, such an improvement would probably make other components of the planner (e.g., the construction of the search graph) more delicate to implement and eventually less efficient to run.

With respect to graph searching, we have proposed a set of techniques for allowing the planner to take advantage of the work carried out at other levels of detail for conducting the search at the current level of decomposition. Some of the techniques, inspired from dependency-directed backtracking, allows the planner to record the conditions of past mistakes so that it does not repeat them.

We have implemented the proposed algorithms in a path planner, with which we have conducted a variety of experiments. These experiments show that our planner is significantly faster than previous planners based on the same general approach.

Our current research is aimed at developing a motion planner/controller capable of dealing with some un-expected obstacles. Our approach described in [Choi, Zhu and Latombe, 1989] consists of controlling the robot in a channel using a potential field approach. We have conducted experiments with this approach both with a simulated robot and an actual one. We currently work on the interaction between the planner and the controller, so that if the robot gets stuck in a channel (for example, the channel may be completely obstructed by an obstacle), the controller may ask the planner to modify the channel accordingly. This kind of interaction requires to planner to be fast.

References

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1983) *Data Structures and Algorithms*, Addison-Wesley, Reading, MA.

Avnaim, F. and Boissonnat, J.D. (1988) *Polygon Placement Under Translation and Rotation*, Technical Report No. 889, INRIA, Sophia-Antipolis, France.

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

Brooks, R.A., and Lozano-Pérez, T. (1982) *A Subdivision Algorithm in Configuration Space for Findpath with Rotation*, AI Memo 684, AI Laboratory, MIT, Cambridge, MA.

Brooks, R.A. and Lozano-Pérez, T. (1983) "A Subdivision Algorithm in Configuration Space for Finpath with Rotation," *Eighth International Joint Conference on Artificial Intelligence*, 799-806, Karlsruhe, FRG.

Brost, R.C. (1989) "Computing Metric and Topological Properties of Configuration-Space Obstacles," *IEEE International Conference on Robotics and Automation*, Scottsdale, Arizona.

Choi, W., Zhu, D.J. and Latombe, J.C. (1989) "Contingency-Tolerant Motion Planning and Control," *IEEE Workshop on Intelligent Robots and Systems*, IROS'89, Tsukuba, Japan.

Donald, B.R. (1984) *Motion Planning With Six Degrees of Freedom*, Report No. AI-TR-791, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

Faverjon, B. (1986) "Object Level Programming Using an Octree in the Configuration Space of a Manipulator," *IEEE International Conference on Robotics and Automation*, San Francisco, CA.

Gouzènes, L. (1984) "Strategies for Solving Collision-Free Trajectories Problems for Mobile and Manipulator Robots," *International Journal of Robotics Research*, 3(4), 51-65.

Kambhampati, S. and Davis, L.S. (1986) "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, RA-2 (3), 135-145.

Kant, K. and Zucker, S.W. (1986) *Planning Smooth Collision-Free Trajectories: Path, Velocity and Splines in Free Space*, Technical Report, Computer Vision and Robotics Laboratory, McGill University, Montréal.

Khatib, O. (1986) "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, 5(1), 90-98.

Khosla, P. and Volpe, R. (1988) "Superquadric Artificial Potentials for Obstacle Avoidance and Approach," *IEEE International Conference of Robotics and Automation*, Philadelphia, PA, 1178-1784.

Latombe, J.C. (1979) "Failure Processing in a System for Designing Complex Assemblies," *Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, Tokyo, Japan.

Laugier, C. and Germain, F. (1985) "An Adaptative Collision-Free Trajectory Planner," *International Conference on Advanced Robotics (ICAR)*, Tokyo, Japan.

Laumond, J.P. (1987). "Obstacle Growing in a Non Polygonal World," *Information Processing Letters, Information Processing Letters*, 25, 41-50.

Lozano-Pérez, T. (1983) "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, C-32(2), 108-120.

Lozano-Pérez, T. (1987) "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation*, RA-3(3), 224-238.

Ó'Dúnlaing, C., Sharir, M. and Yap, C.K. (1983) "Retraction: A New Approach to Motion Planning," 15^{th} *FOCS*, 207-220.

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

Rimon, E. and Koditschek, D.E. (1988) "Exact Robot Navigation using Cost Functions: The Case of Distinct Spherical Boundaries in E^n," *IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1791-1796.

Schwartz, J.T. and Sharir, M. (1983) "On the 'Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds", *Advances in Applied Mathematics*, Academic Press 4, 298-351; also in [Schwartz, Sharir and Hopcroft, 1987].

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

Spivak, M. (1979) *A Comprehensive Introduction to Differential Geometry*, Publish or Perish, Wilmington, DE.

Stallman, R.M. and Sussman, G.J. (1977) "Forward Reasoning and Dependency-Directed Backtracking in a System for Computed-Aided Circuit Analysis," *Artificial Intelligence*, 9(2), 135-196.