



HAL
open science

A practical approach to motion-planning for manipulators with many degrees of freedom

Bernard Faverjon, Pierre Tournassoud

► **To cite this version:**

Bernard Faverjon, Pierre Tournassoud. A practical approach to motion-planning for manipulators with many degrees of freedom. [Research Report] RR-0951, INRIA. 1988. inria-00075608

HAL Id: inria-00075608

<https://inria.hal.science/inria-00075608>

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IRIA

UNITE DE RECHERCHE
IRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 951

Programme 6

A PRACTICAL APPROACH TO MOTION-PLANNING FOR MANIPULATORS WITH MANY DEGREES OF FREEDOM

**Bernard FAVERJON
Pierre TOURNASSOUD**

Décembre 1988



A Practical Approach to Motion-Planning for Manipulators with Many Degrees of Freedom

B. Faverjon * and P. Tournassoud **

(*) INRIA Sophia-Antipolis, Av. Emile Hugues, 06565 Valbonne, France

(**) INRIA, Domaine de Voluceau, BP 105, 78153 Le Chesnay Cedex, France

Abstract

This report gives unified tools to attack the motion planning problem for general manipulator robots in a 3D environment. These tools are hierarchical geometric models of objects, for faster computation of critical distances by use of thresholding, general techniques to compute distances and their variations when the configuration of the system changes, and structuration of Free Space in Configuration Space as a 2^n -tree. They are used both for local computation of safe trajectories, by mapping the control of critical distances in Cartesian Space into constraints in Configuration Space, and to build global models of Free Space in C. Space. For systems with over three or four significant degrees of freedom (e.g., a six-joint manipulator carrying a bulky load), we propose to systematically decompose the problem into two levels, memorizing the results of execution of the local planner by use of learning techniques. In addition, we extend the motion-planning problem to a broader class of tasks, defined by the control of a set of measures on the system.

Une approche pratique pour la planification de trajectoires des manipulateurs à nombre élevé de degrés de liberté.

Résumé

Ce rapport décrit un ensemble d'outils cohérents pour traiter le problème de la planification de trajectoires des robots manipulateurs à nombre élevé de degrés de liberté. Ces outils sont des modèles géométriques hiérarchiques des objets, pour calculer efficacement les distances critiques par seuillage, des techniques générales pour calculer ces distances et leurs variations lorsque la configuration du système évolue, et la structuration de l'espace libre dans l'espace des configurations en grille hiérarchique. Ils sont utilisés à la fois pour générer localement des trajectoires sûres, et pour construire un modèle global de l'espace libre. Dans le cas de systèmes qui comptent plus de trois ou quatre degrés de liberté significatifs, nous proposons de décomposer systématiquement le problème sur deux niveaux, en mémorisant les résultats d'exécution d'un calculateur local par des techniques d'apprentissage. Nous étendons aussi la définition du problème à une plus large classe de tâches, définies par le contrôle d'un ensemble de mesures du système.

1 Introduction

1.1 Why is Path-planning for Manipulators Difficult ?

The problem of motion-planning for manipulators is that of finding a path from an initial to a goal configuration, avoiding collisions with obstacles in the workspace. This ability is crucial in order to achieve *task-level programming*, this is, being allowed to describe a task in terms of geometric relationships between objects, rather than purely joint angle commands. We will attack in this paper the general problem of motion-planning for manipulators with prismatic and revolute joints (eventually co-operating robots) in a three-dimensional environment.

Searching for a path involves representing the geometry of the problem in the *Configuration Space* of the system, chosen among the spaces of independent parameters describing the position of all moving bodies (e.g., joint angles for a manipulator robot). Conceptually, planning motions for the robot is then reduced to planning motions for a point representing the configuration. Path-planning algorithms will be composed of two critical steps. The first one is the computation of the transform of obstacles in Configuration Space, this is, the set of configurations for which there is collision. The latter consists in structuring *Free Space*, the complement of the transform of obstacles in Configuration Space, so that the search for a path can be performed efficiently.

This construction is difficult, for two main reasons. First, we are faced with the difficulty of representing rotations in three-dimensional space, which makes it quite irksome to compute transforms of obstacles for kinematic chains with many revolute joints. The second reason is the inherent exponential complexity of the problem with the number of degrees of freedom of the system. Many instances of the general motion-planning problem are indeed provably *NP-hard* or *PSPACE-hard* with the number of degrees of freedom, even if there exist general polynomial algorithms to answer the problem with a fixed number of degrees of freedom (Schwartz and Sharir [SS83]).

The first point has found a solution with the idea of discretizing Configuration Space, giving approximate, conservative solutions. Answering the latter point will always be difficult. A tricky answer has been to reduce the number of degrees of freedom taken into

account to describe the search space (e.g., Brooks [Bro83]). Another -treacherous- answer is to forget the requirement of good termination of the motion (reach the goal) and make the system incrementally move towards the goal using minimization technique. This latter class of approaches, illustrated by the *Potential Field* method (Khatib [Kha86]), more inspired by control theory, has proved to be very powerful for describing a large family of robotic tasks.

1.2 Bases of our Approach

Our approach deeply relies on the on the ideas of Lozano-Pérez to represent the problem as a graph search in a discretized Configuration Space [Loz87]. The graph represents adjacency between elementary cells of safe configurations.

Other tools that we introduce are original. These are essentially hierarchical geometric models of fixed and moving solids, to accelerate the computation of critical distances by use of thresholding, general techniques to compute distances between solids and their variations when the configuration of the system changes, and structuration of Free Space in Configuration Space as a hierarchical grid (2^n -tree).

These tools are used to attack the problem under a number of angles.

Local Planning: A first order model of the variation of distances, mapped into Configuration Space, is used to build a simple local model of Free Space. This model is first used as a basis for minimization techniques by a local planner. As opposed to the standard *Potential Field* approach, the idea is to separate the realization of the task, described by the minimization of a set of measures of the problem, and eventually some geometric constraints, from the constraints of anti-collision between the robots and the environment. This makes it possible to respect accurately both the geometric constraints on the task, and anti-collision constraints that remain very close to the geometry of the problem.

Global Planning: Similar local models are used to build incrementally a representation of Free Space as a connectivity graph between free cells. The amount of memory required is important, as small cells are needed to describe the boundaries of the transforms of obstacles. Due to memory limitations and construction cost, this can only be

achieved up to three or four degrees of freedom, for example, to represent motions of the arm of a manipulator (its first three links). For planning the path of a manipulator carrying a small load, we propose to have local and global techniques collaborate, local six-dimensional models of Free Space being used at both ends of the trajectory to generate *fine motions* close to the obstacles, and during *gross-motions* of the arm as heuristics to influence the search of a path for the first three links.

Mixed Approach: For systems with over three or four significant degrees of freedom (e.g., a six-joint manipulator with a bulky load), we propose to systematically decompose the motion-planning problem into two levels. We build a graph whose nodes represent relatively large cells of Configuration Space. Transitions between adjacent cells are weighted by the probability for the local planner to succeed in moving the system from one cell to the other. These probabilities are used by a minimum cost path finding algorithm to generate subgoals for the local planner. They are updated using Bayesian rules, from the results of the execution of trajectories planned locally. As we deal with the high complexity of graph searching only at the relevant level of the description, it is possible to apply this technique to robotic systems with more degrees of freedom.

2 Geometric Models

2.1 A Hierarchical Description of Solids

A key element in motion-planning algorithms is the geometric representation of objects they use. In our implementation solids are approximated by unions of simple convex volumes that we call *primitives*. These are 3D-rectangles, prisms, cylinders, truncated cones and spheres.

Each solid is represented by a hierarchical structure that we call an *Assembly Tree*. This is a binary tree with a primitive attached to each node. This primitive contains both primitives attached to its sons. The object is most precisely described by the unions of leaf primitives, but any cross-section of the tree provides a *conservative* approximation (that contains the object).

This hierarchical structure is computed automatically by the CAD system. When we

assemble two objects, a new node is created with the smallest volume primitive containing both of them attached to it. Figure 1 illustrates this construction.

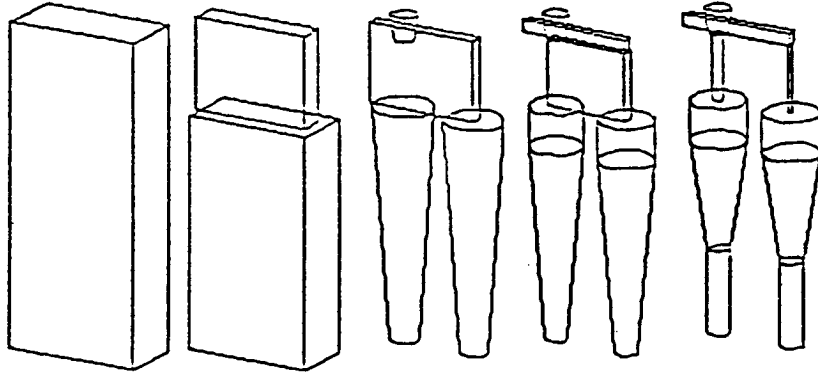


Figure 1: An Assembly Tree representing part of the environment. Any cross-section of the tree is an approximation containing the exact description.

A straightforward application is to test efficiently the intersection of two solids, starting at the roots and performing a balanced descent in both Assembly Trees.

2.2 A Hierarchical Description of Manipulators

We describe a robot by an open articulated chain of $n + 1$ solid bodies $B^i, i = 0, \dots, n$, each body being described by an Assembly Tree. The i -th joint, of parameter q^i , links bodies B^{i-1} and B^i . It is translational or revolute.

A n degree of freedom manipulator is represented by a pyramid with $n + 1$ levels. Level $k, 0 \leq k \leq n$, represents a virtual manipulator M^k with k degrees of freedom, the last links being replaced by their swept volumes when joints $k + 1$ to n vary freely. More precisely, M^k is the union :

- of bodies B^1, \dots, B^k ,
- and of the Assembly Tree representing volumes $SV^{i,k}$ swept by bodies $B^i, i > k$, when q^1, \dots, q^k are fixed and q^{k+1}, \dots, q^i vary freely.

We call the root of this tree *Swept Volume* of level k , and denote it SV^k . Figure 2 illustrates this pyramidal structure. for the first three links of a vertical manipulator. Leaf primitives

at each level are represented in figure 3. A node of this structure contains the node directly below it, and, if it is not a leaf or a body, two sons on the same level.

2.3 Application to Fast Intersection Tests

Those structures, Assembly Trees for solids and virtual manipulators for robots, are very efficient for checking intersection and computing distances making use of thresholding.

We illustrate this point by the example of an intersection test between two manipulators *Robot₁* and *Robot₂*. We update a list of intersecting pairs of nodes, using alternatively :

- the tree structure at fixed levels (k_1, k_2) ,
- the inheritance property from a level to the one below, yielding pairs of levels $(k_1 + 1, k_2)$ or $(k_1, k_2 + 1)$.

Updating at levels $(0,0)$ realizes a preprocessing that yields once and for all the list of initial interactions at levels $(1,1)$. We have detailed those algorithms in [FT86]. Figure 4 illustrates the sequence of tests for two configurations of the system. The example is limited for clarity to the first three links of the robots. For trajectories along which two robots cross, coming very close one from the other, we observed that the average cost for one collision test was 0.08 s^1 . Each manipulator consisted in a six-joint articulated chain described by 14 elementary convex primitives.

3 On Local Planning

In this section we propose a local, memory-less method to generate trajectories for one or more manipulators. If it has many features similar to the *Potential Field* method, it also makes use of different tools and exhibits different behaviours.

At each time increment, we will perform the following computations.

- From the geometric models of the robots and their environment, we compute the list of pairs of primitives that need to be separated : a pair consists in two primitives lying

¹All CPU times given in this paper were measured on a SUN3 workstation.

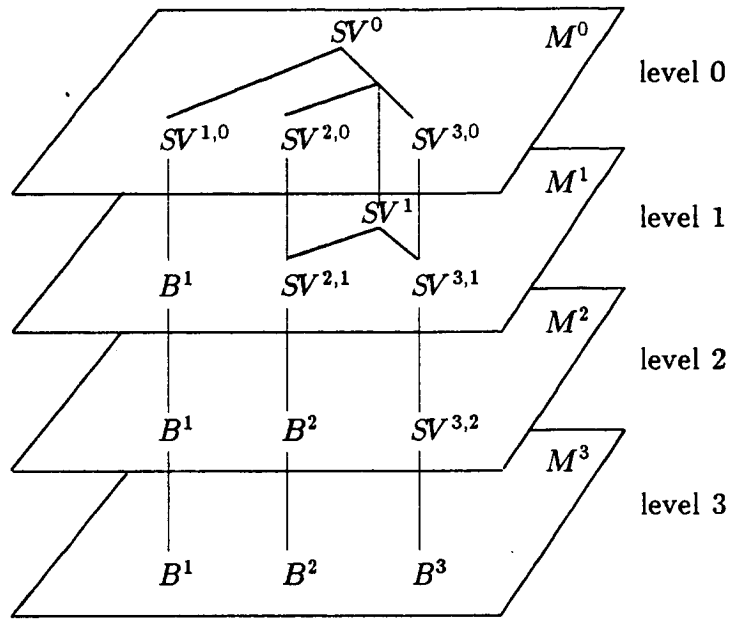


Figure 2: A robot is represented by a pyramid with $n + 1$ levels.

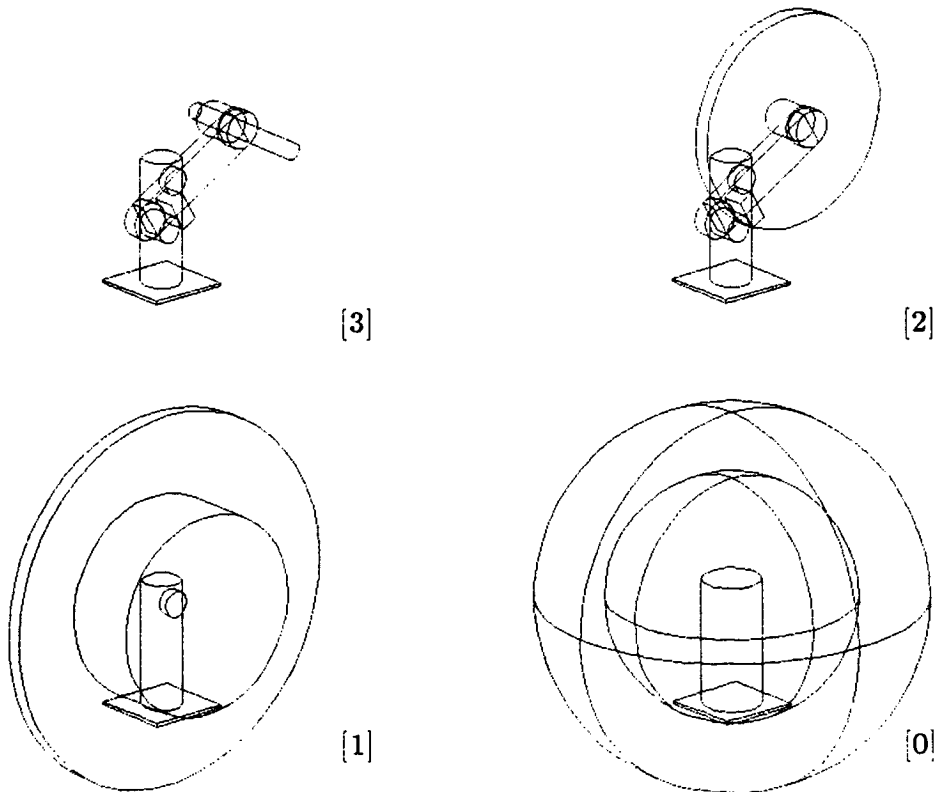
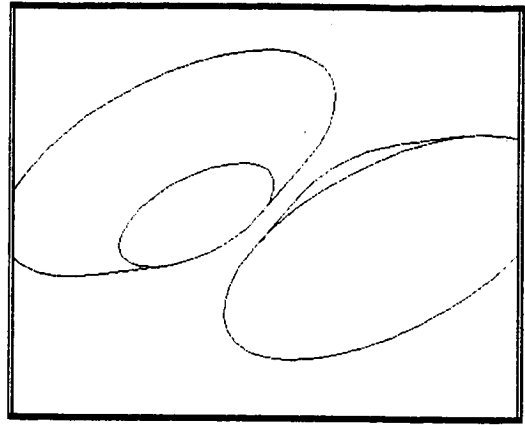
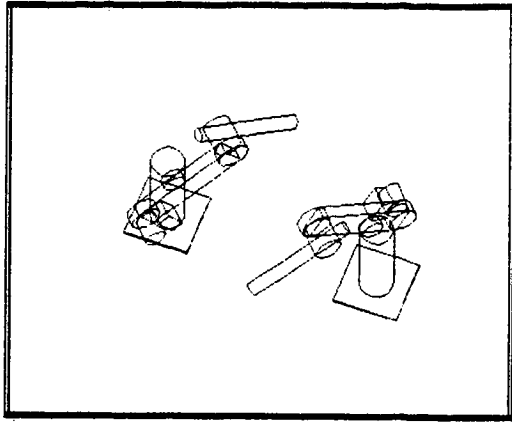
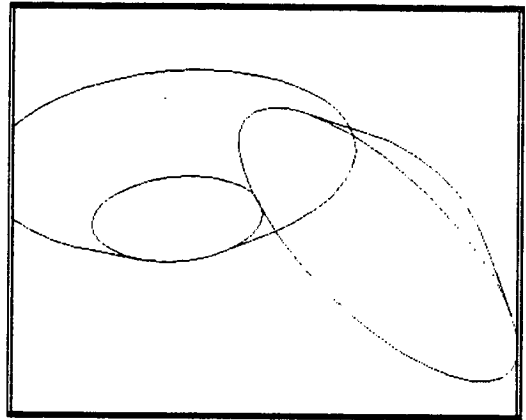
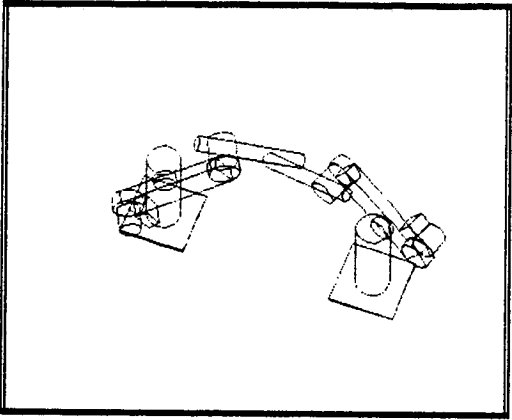


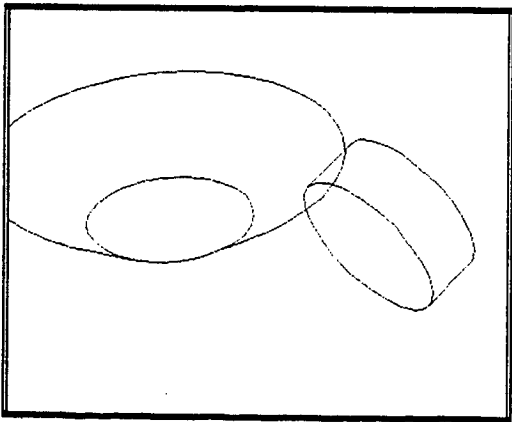
Figure 3: Leaf primitives at each level.



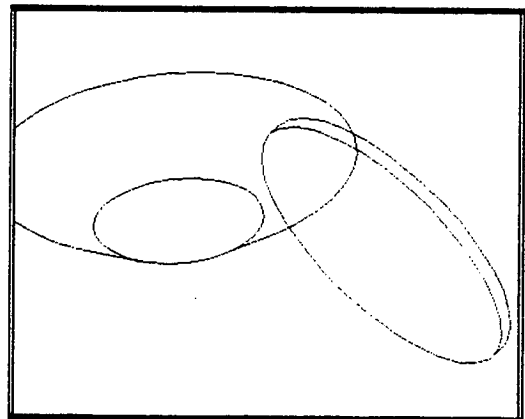
$k_1 = 1 \quad \not\cap \quad k_2 = 1$



$k_1 = 1 \quad \cap \quad k_2 = 1$

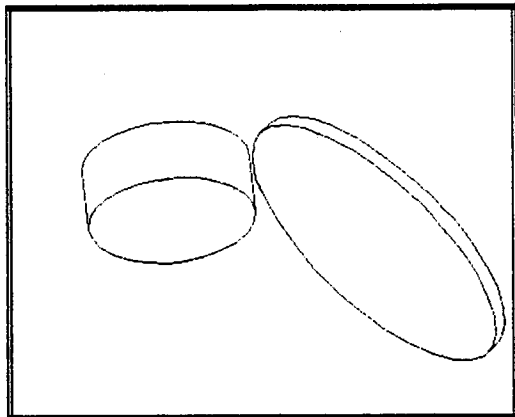


$k_1 = 1 \quad \not\cap \quad k_2 = 1$

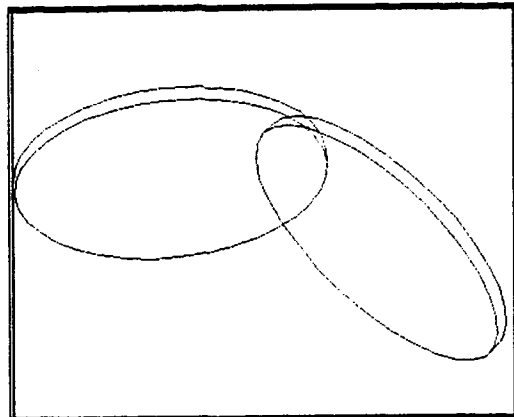


$k_1 = 1 \quad \cap \quad k_2 = 1$

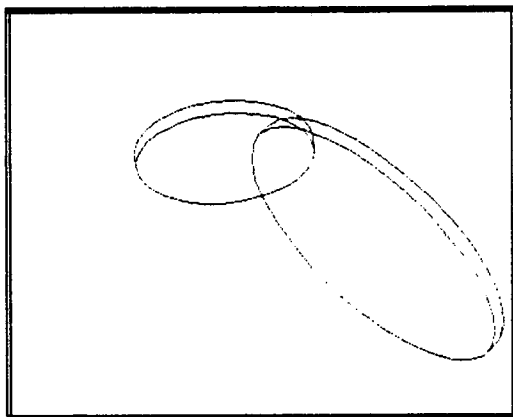
Figure 4: Checking collision between two manipulators.



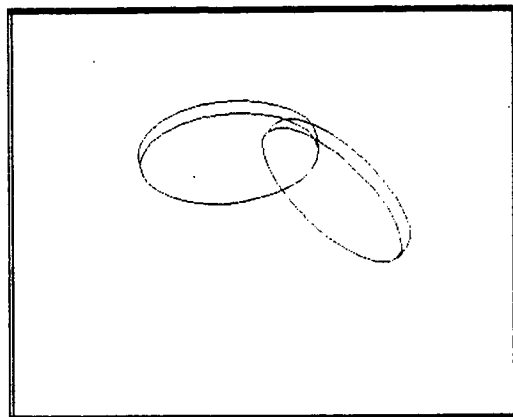
$k_1 = 1 \quad \emptyset \quad k_2 = 1$



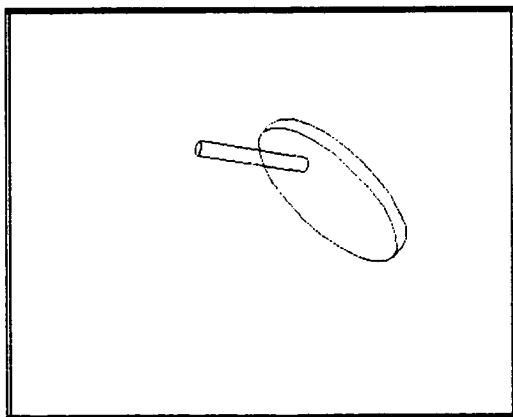
$k_1 = 1 \quad \cap \quad k_2 = 1$



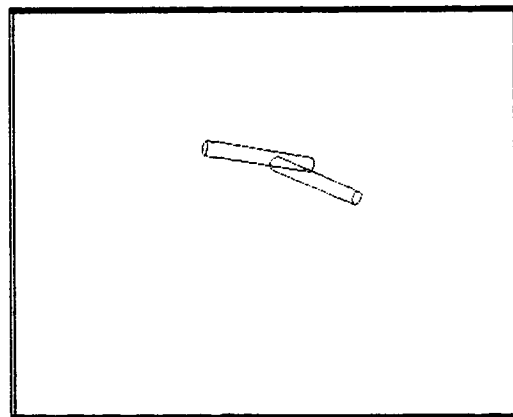
$k_1 = 2 \quad \cap \quad k_2 = 1$



$k_1 = 2 \quad \cap \quad k_2 = 2$



$k_1 = 3 \quad \cap \quad k_2 = 2$



$k_1 = 3 \quad \cap \quad k_2 = 3$

at less than a conservative influence distance σ , function of the maximum Cartesian displacement for points on these solids. One primitive is part of the description of a body of a manipulator, the other part of a fixed obstacle. Both of them also can be mobile primitives, on the same kinematic chain, on two different robots in a multi-robot application.

- From the list of interactions we compute a local view of obstacles as seen in the Configuration Space of the system, that translates the control of critical distances between pairs of interacting solids.
- Finally, we will use minimization techniques to get closer to the realization of the task, while remaining in the local, conservative model of Free Space we have built.

3.1 An Example

Let us first give an example of a trajectory obtained with this algorithm, for a redundant 10 link 8 degree of freedom arm carrying a part to be welded, in the complex environment of a nuclear plant reactor. Figure 5 is a three-dimensional view of the environment, figure 6 shows a docking position for the load. Tasks that can be performed consist in reaching a goal configuration, or in achieving geometric constraints on the final position of the part, described in Cartesian space. Such is the case for the example illustrated in figure 7. Note that it was necessary to provide one sub-goal along the trajectory.

Hierarchical descriptions introduced in section 2 are used for efficiently updating the current model of the environment when the robot is moving. This model consists in a list of nodes from the Assembly Tree describing the environment. When a primitive of the robot moves towards a given node, a more precise description is obtained by replacing this node by its sons in the tree. On the contrary, the robot may move away from an obstacle, so that two nodes with the same father now lie at a distance greater than the threshold σ from the robot. Then they are replaced by their father in the list.

In the application illustrated in figure 7, the environment is composed of about 200 primitives and the robot of 20. A straightforward use of the hierarchical structures enables to deal with only 50 interactions at a time, compared with the 4000 potential ones.

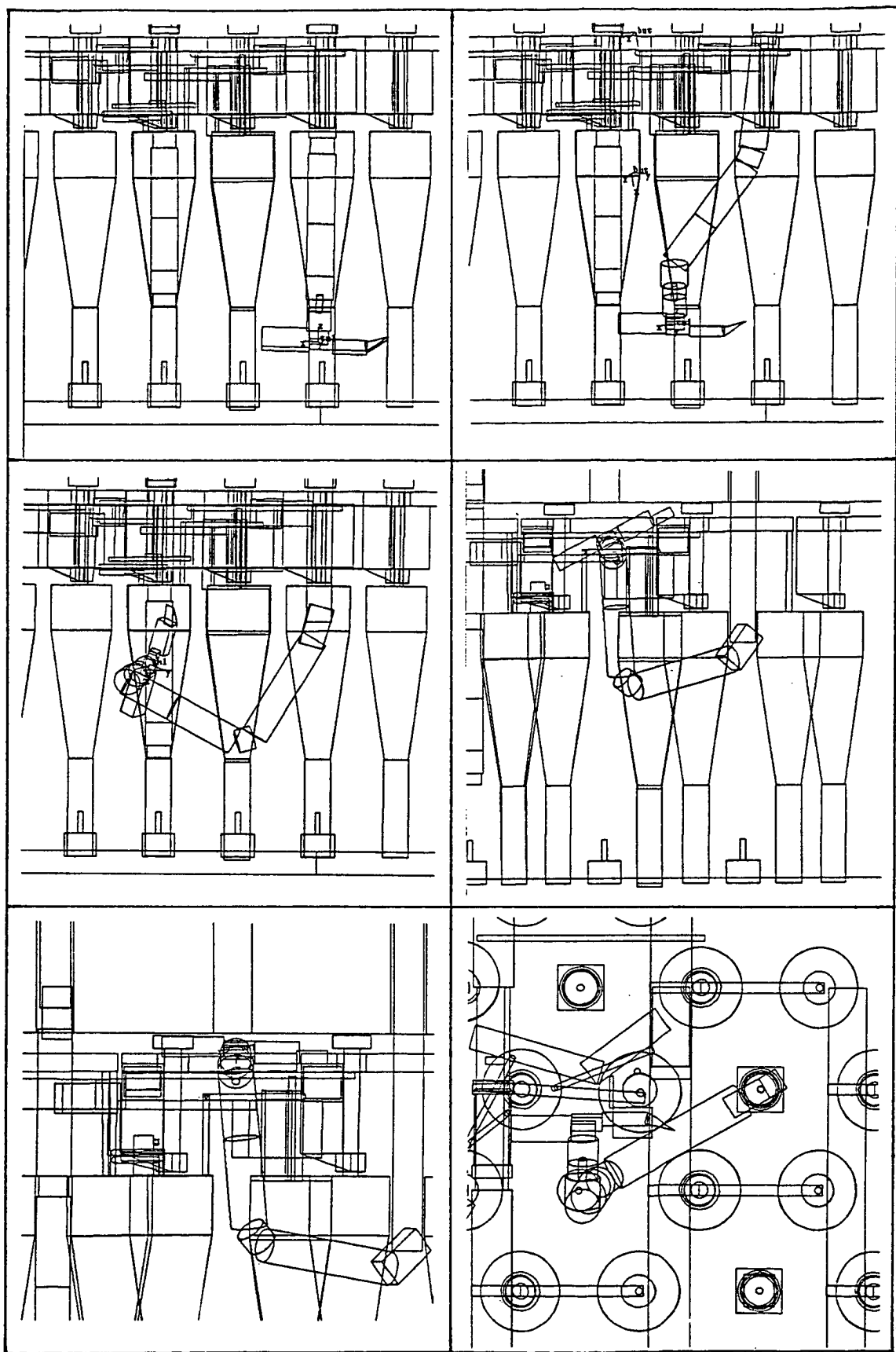


Figure 7: A trajectory obtained : the goal is described in terms of geometric constraints on the final position of the load.

Furthermore, most of the corresponding primitives of the environment lie at a distance superior to the threshold σ from the primitive on the robot. We can perform a worst case updating of the distance using a conservative approximation of the maximum velocity of points on the moving primitive. Then only nodes lying at an estimated distance lower than σ from the robot need to be examined precisely at next step, typically 10 at each time increment. This enables real-time computation of collision-free trajectories for this particular manipulator, which moves only at a maximum velocity of 20 cm per second at the tip of the hand.

3.2 Computing the Distance Between Two Convex Objects

We begin by defining tools for the efficient computation of distances between bodies of the robots and the environment. In the sequel, S_1 and S_2 denote two convex solids in Cartesian space, \mathbf{x}_1 and \mathbf{x}_2 two points belonging respectively to S_1 and S_2 , and \mathbf{n} a unit vector. Notation $(\mathbf{u}|\mathbf{v})$ refers to the inner product of vectors \mathbf{u} and \mathbf{v} .

The Euclidean Distance between S_1 and S_2 , equal to $\min_{\mathbf{x}_1 \in S_1, \mathbf{x}_2 \in S_2} \|\mathbf{x}_1 - \mathbf{x}_2\|$, can be computed by alternatively projecting a point of S_1 onto S_2 , the point of S_2 that we obtain onto S_1 , etc..., until the distance between the points converges. Yet this method has a major drawback : no thresholding based on a maximum influence distance can be done, as the current estimation is always bigger than the exact distance, which is unknown.

For this reason, we use the following definition of the distance between two solids :

$$d(S_1, S_2) = \max_{\|\mathbf{n}\|=1} \min_{\mathbf{x}_1 \in S_1, \mathbf{x}_2 \in S_2} (\mathbf{n}|\mathbf{x}_1 - \mathbf{x}_2) \quad (1)$$

With this definition $d(S_1, S_2)$ is equal to the Euclidean Distance only in the case of non overlapping objects. If they do overlap, the "distance" defined by equation 1 becomes negative and measures how far objects interpenetrate.

We call *distance between S_1 and S_2 oriented in the direction \mathbf{n}* the scalar :

$$\delta(\mathbf{n}) = \min_{\mathbf{x}_1 \in S_1, \mathbf{x}_2 \in S_2} (\mathbf{n}|\mathbf{x}_1 - \mathbf{x}_2) \quad (2)$$

Note that we have $d(S_1, S_2) = \max_{\|\mathbf{n}\|=1} \delta(\mathbf{n})$.

The interesting point in this definition is that $\delta(\mathbf{n})$ is always lower than $d(S_1, S_2)$. We have defined the influence distance σ as a threshold on interactions between objects : if

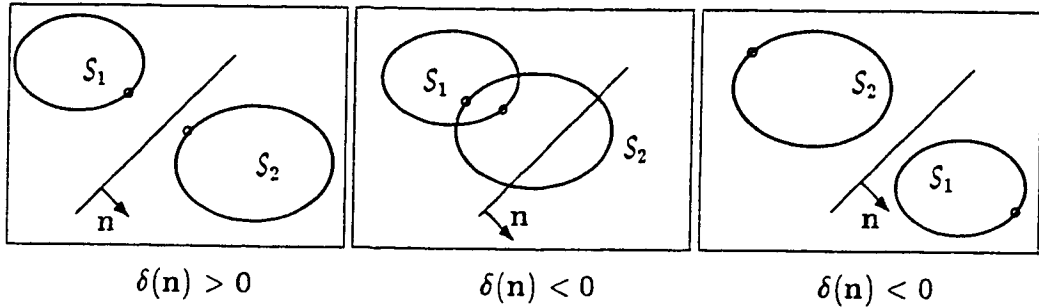


Figure 8: Generic examples of the oriented distance.

objects lie at a distance bigger than σ , we will not impose any constraint between them. Then if $\delta(\mathbf{n})$ is greater than σ for some value of \mathbf{n} , the same stands for the exact distance and we can declare these objects to be non interacting.

The computation of $\delta(\mathbf{n})$ can be decomposed into :

$$\delta(\mathbf{n}) = \min_{\mathbf{x}_2 \in S_2} (\mathbf{n} \cdot \mathbf{ox}_2) + \min_{\mathbf{x}_1 \in S_1} (-\mathbf{n} \cdot \mathbf{ox}_1) \quad (3)$$

for an arbitrary reference point \mathbf{o} . Point \mathbf{x}_2 realizing the minimum will be called *first point of S_2 in direction \mathbf{n}* . Then point \mathbf{x}_1 realizing the minimum is the first point of S_1 in the opposite direction, $-\mathbf{n}$. To compute the distance $d(S_1, S_2)$ in the case of non overlapping objects, we use the following procedure, illustrated in figure 9.

1. Select an arbitrary point \mathbf{y}_1 in S_1 .
2. Project \mathbf{y}_1 onto S_2 . Call \mathbf{y}_2 the projection.
3. Let $\mathbf{n} = \mathbf{y}_1\mathbf{y}_2 / \|\mathbf{y}_1\mathbf{y}_2\|$. Compute the first point of S_1 in direction $-\mathbf{n}$, \mathbf{x}_1 .

Then $\delta(\mathbf{n}) = (\mathbf{n} \cdot \mathbf{x}_1\mathbf{y}_2)$, and we have :

$$\delta(\mathbf{n}) \leq d(S_1, S_2) \leq \|\mathbf{y}_1\mathbf{y}_2\| \quad (4)$$

The same procedure is run again starting with point \mathbf{y}_2 of S_2 obtained above. It can be shown that $\delta(\mathbf{n})$ and $\|\mathbf{y}_1\mathbf{y}_2\|$ both converge towards $d(S_1, S_2)$ if the procedure is repeatedly

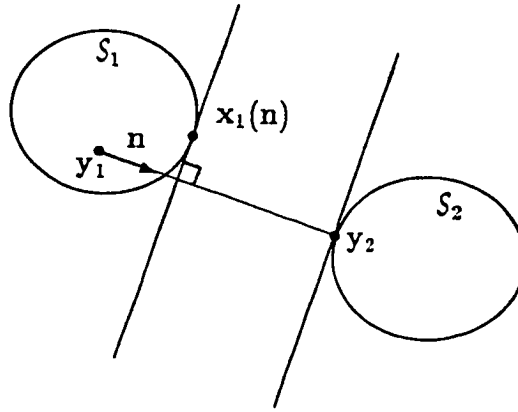


Figure 9: Computing the distance between two solids. The actual distance lies between $\delta(\mathbf{n})$ and $\|y_1 y_2\|$.

applied. In practice, the process is stopped as soon as $\|y_1 y_2\| - \delta(\mathbf{n})$ is smaller than a given precision, or when $\delta(\mathbf{n})$ is bigger than the influence distance σ .

The projection of a point onto a primitive, or the calculation of its first point in some direction \mathbf{n} , just requires a call to a simple function (of unit cost) dependent on the type of the primitive.

We now consider that S_1 and S_2 are moving solids. Their position and orientation are described by vectors of configuration parameters \mathbf{q}_1 and \mathbf{q}_2 . An important property of the distance $d(S_1, S_2)$, when calculated between two moving objects, is that it can always be differentiated.

Proposition: *The time derivative of the distance between moving solids S_1 and S_2 writes :*

$$\dot{d} = (\mathbf{n} | \mathbf{J}_2 \dot{\mathbf{q}}_2 - \mathbf{J}_1 \dot{\mathbf{q}}_1), \quad (5)$$

where

- \mathbf{J}_1 and \mathbf{J}_2 are the Jacobian matrices for S_1 and S_2 calculated respectively at points \mathbf{x}_1 and \mathbf{x}_2 realizing the minimum distance,
- \mathbf{n} is the unit vector on the line $\mathbf{x}_1 \mathbf{x}_2$. It realizes the optimum of equation 1.

Writing $\mathbf{J} = (-\mathbf{J}_1 \quad \mathbf{J}_2)$, and $\mathbf{q} = (\mathbf{q}_1^t, \mathbf{q}_2^t)^t$ the global configuration vector, equation 5 also

writes :

$$\dot{d} = (\mathbf{n}|\mathbf{J}\dot{\mathbf{q}}) = (\mathbf{J}^t\mathbf{n}|\dot{\mathbf{q}}). \quad (6)$$

We call \mathbf{J} the “relative Jacobian” at points \mathbf{x}_1 and \mathbf{x}_2 .

Proof: We write $d = (\mathbf{n}|\mathbf{x}_1\mathbf{x}_2)$ for unit vector \mathbf{n} and points $\mathbf{x}_1, \mathbf{x}_2$ verifying the minimum distance. Vector \mathbf{n} is directed along the line joining \mathbf{x}_1 and \mathbf{x}_2 , and points from S_1 towards S_2 . The point \mathbf{x}_i of S_i verifying the minimum distance is not a point bound to the solid, but describes a trajectory on its surface. Hence its time derivative is equal to :

$$\dot{\mathbf{x}}_i = \dot{\mathbf{x}}_i^{S_i} + \mathbf{v}_i, \quad \text{with} \quad (7)$$

- $\dot{\mathbf{x}}_i^{S_i}$ the relative velocity of this point in a frame bound to S_i ,
- \mathbf{v}_i the velocity of the point bound to S_i coinciding with \mathbf{x}_i at time t .

The time derivative of the distance writes $\dot{d} = (\mathbf{n}|\dot{\mathbf{x}}_2 - \dot{\mathbf{x}}_1) + (\dot{\mathbf{n}}|\mathbf{x}_1\mathbf{x}_2)$. The second inner product is equal to 0 as the norm of \mathbf{n} remains constant, yielding $(\dot{\mathbf{n}}|\mathbf{n}) = 0$. The first inner product is also equal to $(\mathbf{n}|\mathbf{v}_2 - \mathbf{v}_1)$. Indeed, as point $\mathbf{x}_i, i = 1, 2$, describes a trajectory on the surface of S_i , the inner product $(\mathbf{n}|\dot{\mathbf{x}}_i^{S_i})$ is constantly equal to zero. We obtain equation 6. \square

3.3 Constraints in Configuration Space, a Substitute to the Potential Field Method

In the well-known *Potential Field* method, the robot navigates in a potential field sum of a term that attracts it towards the target, and of repulsive terms generated by obstacles. The trajectory is usually obtained by following the gradient of the potential field.

Such methods can work properly in the case there are only few obstacles in the environment. But as they imply that terms of different nature be arbitrarily added in a single minimization function, problems appear in more complex environments. These include:

- oscillations between opposite obstacles,
- arbitrary higher repulsion for two adjacent obstacles than for a unique one,

- difficulty to get close to an obstacle (by adjusting weighting coefficients of the different terms to be minimized).

In our approach the task is described by a minimization problem, and eventually some geometric constraints. As opposed to the *Potential Field* method, anti-collision is translated into geometric constraints in Configuration Space, and is not included in the function to minimize. This enables to better control all relevant measures of the problem.

3.4 The Velocity Damper

In this section we describe how to translate an interaction between two convex primitives into a constraint on the displacements of the manipulator in Configuration Space. Between a moving primitive and an obstacle, or between two moving primitives, we impose a constraint that we call *velocity damper* : it expresses that the distance between them must not decrease too fast when it is less than the influence distance σ . We start with the following remark.

Proposition: *There cannot be any collision if we impose :*

$$\dot{d} \geq -\xi \frac{d - \varepsilon}{\sigma - \varepsilon}, \quad (8)$$

where ε is the security distance at which the robot must stop and ξ a positive coefficient for adjusting convergence speed.

Proof: We prove that $d - \varepsilon$ remains higher than a decreasing exponential. To see this, write $F(t) = (d - \varepsilon) \exp -\xi t / (\sigma - \varepsilon)$, and check that inequation 8 implies $\dot{F}(t) \geq 0$. We deduce that $F(t) \geq F(0)$ for $t \geq 0$, or equivalently :

$$d \geq \varepsilon + F(0) \exp -\xi t (\sigma - \varepsilon). \quad (9)$$

□

We recall (equation 6) that $\dot{d} = (\mathbf{n} | \mathbf{J}\dot{\mathbf{q}}) = (\mathbf{J}'\mathbf{n} | \dot{\mathbf{q}})$, where \mathbf{n} is the unit vector on the line of minimum distance and \mathbf{J} the relative Jacobian matrix at points \mathbf{x}_1 and \mathbf{x}_2 realizing the minimum (see Figure 10). Thus inequality 8 can be translated into a simple linear

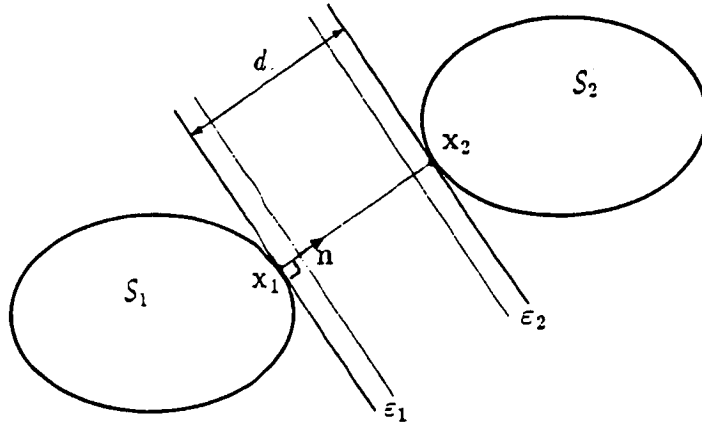


Figure 10: Writing the velocity damper constraint ($\varepsilon = \varepsilon_1 + \varepsilon_2$).

constraint on configuration parameters increments $\delta\mathbf{q}$ between time t and $t + \delta t$, namely with $\hat{\mathbf{n}} = \mathbf{J}'\mathbf{n}$:

$$(\hat{\mathbf{n}}|\delta\mathbf{q}) \geq -\xi \frac{d - \varepsilon}{\sigma - \varepsilon} \delta t \quad (10)$$

A nice property of these constraints is that they allow to go arbitrarily near from the limit $d = \varepsilon$ in a finite time, though the component of the velocity normal to the obstacle gets very small.

As illustrated by figure 11, combining those constraints and bounds on configuration parameter increments (standing for maximum joint velocities) results in building a local model of Free Space in Configuration Space. This simplified model is convex and conservative (included in the current connected component of Free Space). A positive point is that the generated constraints remain close from the original geometry of the problem. As the minimization we perform will be initiated with zero joint increments, which clearly respect the constraints, redundant constraints will not be examined in the process and will not have any influence on the result.

This local model of Free Space is used in next subsection as a basis for minimization techniques. It can have many other applications. In section 4, similar computations are used for incrementally building a global model of Free Space ; in section 5, for doing some naive geometric reasoning to accelerate the learning of safe displacements.

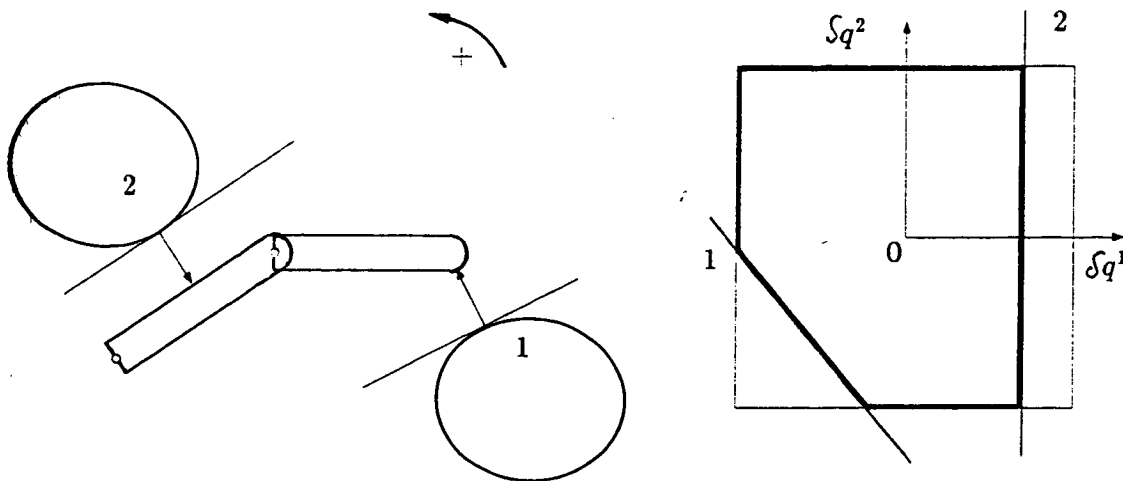


Figure 11: A local model of Free Space in Configuration Space : the simple case of a planar manipulator with two revolute joints.

3.5 Description of a Task

The realization of a task is expressed by the minimization of a set of measures on the system, and eventually the respect of some geometric constraints. Anti-collision constraints are expressed separately by means of the local model of Free Space introduced above.

We will use homogeneous configuration parameters $\mathbf{q} = (q^1, \dots, q^n)$, such that $q^i = \theta^i / \omega^i$ for $i \in \{1, \dots, n\}$, where ω^i is the maximum value of the i -th configuration parameter derivative. Hence, with the norm $\|\mathbf{q}\| = \max_{i=1, \dots, n} |q^i|$ for the vector of configuration parameters, we always have $\|\dot{\mathbf{q}}\| \leq 1$.

A task is described by the control of a set of measures of the problem, a vector $\mathcal{T}(\mathbf{q})$ of a p -dimensional space. We again write its norm $\|\mathcal{T}(\mathbf{q})\|$ for simplicity. We suppose that we want to reach a state \mathbf{q} such that $\mathcal{T}(\mathbf{q}) = 0$. This is always possible by adding a constant vector to \mathcal{T} . Such a state will be called a solution of the problem. For example, if the task simply consists in reaching a goal configuration \mathbf{q}_g , we write $\mathcal{T}(\mathbf{q}) = \mathbf{q} - \mathbf{q}_g$.

The first step consists in translating this task into an optimization problem. We observe that if we simply minimize $\|\mathcal{T}(\mathbf{q})\|$, we move too early on the degrees of freedom that are not constrained, which augments eventualities of dead-locks and leads to unstable behaviours.

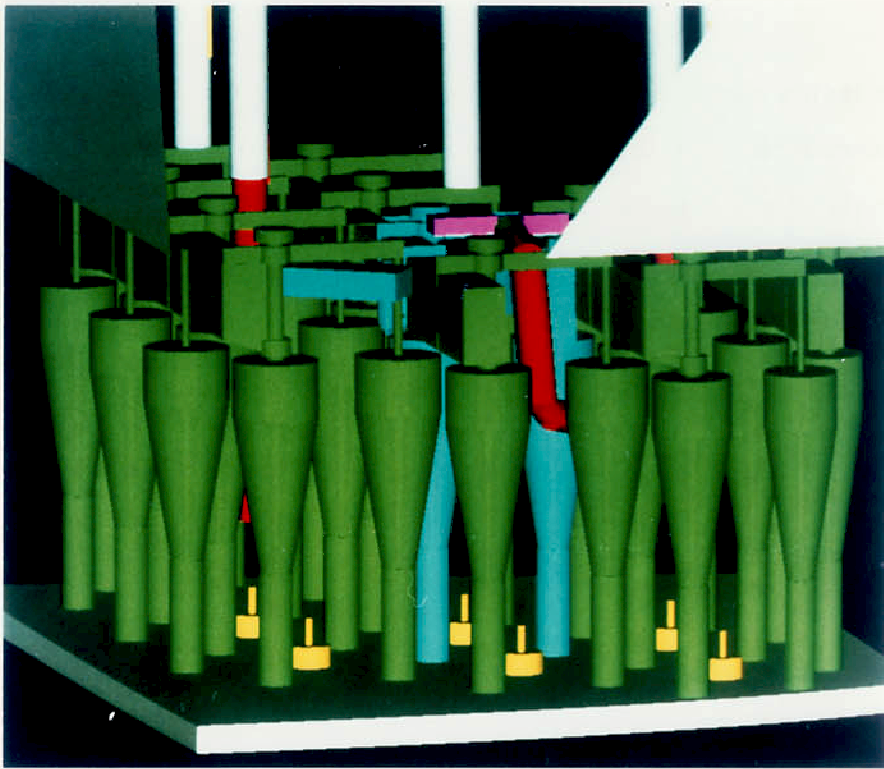


Figure 5: A 10 link robot in the reactor of a nuclear plant : a complex environment.

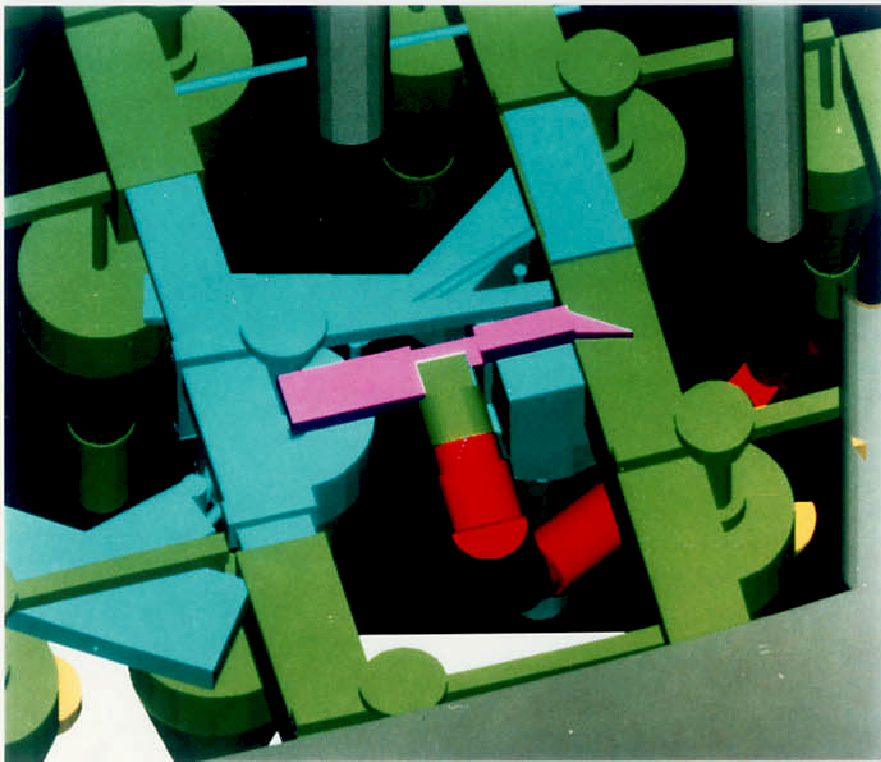


Figure 6: A docking position for the part to be welded.

We write :

$$\mathbf{J}_r = \left(\frac{\partial \tau}{\partial q^1}, \dots, \frac{\partial \tau}{\partial q^n} \right) \quad (11)$$

the *Jacobian of the task* at time t , and $\|\mathbf{J}_r(\mathbf{q})\|$ its algebra norm for configuration \mathbf{q} , equals to $\max_{\delta \mathbf{q} \neq 0} \|\mathbf{J}_r(\mathbf{q}) \delta \mathbf{q}\| / \|\delta \mathbf{q}\|$. Let j_r be the maximum value of $\|\mathbf{J}_r(\mathbf{q})\|$ for all configurations. In practice, we make use of a conservative approximation of j_r .

Writing that we try to follow a straight line in the space where the task is expressed, we formulate the problem :

$$(\mathcal{P}) \text{ minimize } \quad \frac{1}{2} \|\dot{\tau}(\mathbf{q}) - \tilde{\tau}\|^2, \text{ with}$$

$$\tilde{\tau} = -j_r \frac{\tau(\mathbf{q})}{\|\tau(\mathbf{q})\|}$$

Note that $\tilde{\tau}$ is not the derivative of the measure along a reference trajectory, but only the current desired value of $\dot{\tau}(\mathbf{q})$. In particular, we do not memorize eventual deviations caused by an obstacle.

In the case we simply want to reach a goal configuration, this translates easily in terms of desired configuration parameters increments :

$$\delta \mathbf{q}^* = \frac{\mathbf{q}_g - \mathbf{q}}{\max_{i=1, \dots, n} |q_g^i - q^i|} \delta t \quad (12)$$

In the absence of any obstacle, the trajectory then follows a straight line in Configuration Space.

For a given time increment δt , optimal variations $\delta \mathbf{q}^*$ of configuration parameters are computed by solving minimization program (\mathcal{P}') , discrete expression of (\mathcal{P}) :

$$(\mathcal{P}') \text{ minimize } \quad \frac{1}{2} (\mathbf{J}_r^t \mathbf{J}_r \delta \mathbf{q} | \delta \mathbf{q}) - \delta t (\mathbf{J}_r^t \tilde{\tau} | \delta \mathbf{q})$$

We add the constraints of anti-collision generated by velocity dampers :

$$(i) \quad (\hat{\mathbf{n}}_c | \delta \mathbf{q}) \geq \alpha_c \delta t, \text{ for } c \in \{1, \dots, n_c\},$$

n_c being the total number of constraints, and bounds on the variations of joint angles :

$$(ii) \quad \|\delta \mathbf{q}\| \leq \delta t \text{ this is } -\delta t \leq \delta q^i \leq \delta t \text{ for } i \in \{1, \dots, n\}.$$

In the case we simultaneously want to realize k sub-tasks $\tau_1(\mathbf{q}), \dots, \tau_k(\mathbf{q})$, the composed task is simply the vector $\mathcal{T}(\mathbf{q}) = (\tau_1(\mathbf{q}), \dots, \tau_k(\mathbf{q}))$ written in the space cross-product of the spaces of the sub-tasks. We can easily prioritize tasks by adding weighting coefficients p_i and write $\mathcal{T}(\mathbf{q}) = (p_1 \tau_1(\mathbf{q}), \dots, p_k \tau_k(\mathbf{q}))$.

Once a subtask is realized, we can impose it remain as a geometric constraint included in the definition of the task. It is then translated into an equality constraint $\tau_e(\mathbf{q}) = 0$, that we linearize around current value $\tau_i(\mathbf{q})$:

$$\tau_e(\mathbf{q}) + \mathbf{J}_{\tau_e} \delta \mathbf{q}^* = 0. \quad (13)$$

This describes such a subtask as following a line with the tip of a tool, or keeping contact with the surface of an object.

We have thus reduced the control of the measures $\mathcal{T}(\mathbf{q})$ to a simple minimization problem with quadratic criterion and linear constraints. Strong points of this technique is that it enables to control independently relevant measures of the problem, to respect precisely given geometric constraints, and to asymptotically approach an obstacle, for example to generate grasp trajectories.

We must outline again that this method is limited in its applications by eventualities of dead-locks before the goal is reached. This limitation is inherent to any local, memory-less approach.

3.6 The Example of Cooperative Tasks Between Two Manipulators

In this section, we describe some applications for two manipulators sharing a common workspace, with respectively n_1 and n_2 degrees of freedom. We write $\mathbf{q} = (\mathbf{q}_1^t, \mathbf{q}_2^t)^t$ the $n_1 + n_2$ dimensional configuration vector of the system.

Avoidance of the manipulators is a simple application of the velocity damper constraints introduced above, written in the Configuration Space of the whole system. Nevertheless, if the environment is not too cluttered, the velocity dampers of equation 10 do not exhibit the best behaviour. As illustrated by figure 12, we would prefer to react well in advance to possible collisions. For this reason, we introduced the *Tangent Separating Plane* method

for computing smooth coordinated trajectories of two manipulators [FT86]. In short, it consists in forcing the bodies of both manipulators to slide on a chain of virtual obstacles, possibly moving, that separate them. For a pair of interacting primitives, this virtual obstacle is computed among their separating planes, so that it least perturbs their nominal displacements. This implies in particular that it will be tangent to both objects. The resulting constraints again can be translated into simple linear inequations on configuration parameter increments, and fit in the general framework of subsection 3.5.

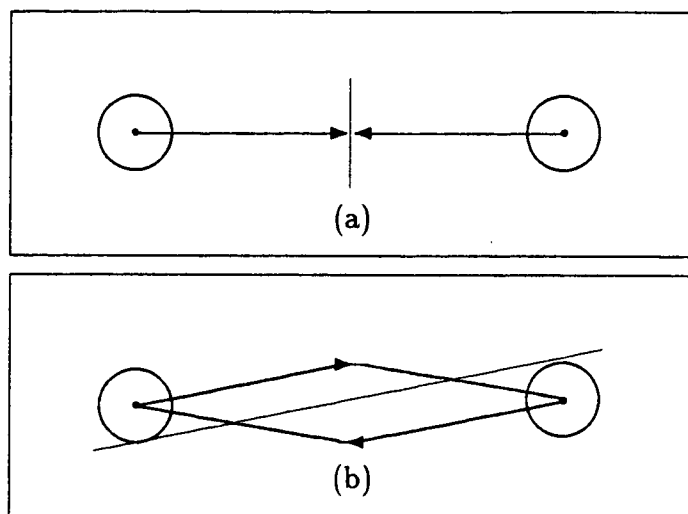


Figure 12: Exchange of the positions of two discs in the plane. Thick lines represent trajectories of the discs. The velocity damper method (a) causes a dead-lock at mid-distance. Sliding on the tangent separating line (b) produces optimal trajectories for both robots.

We now concentrate on tasks of cooperation between two manipulators. Consider the example of figure 13, where we want to transfer an object from one arm to the opposite one. Regrasping will be performed once we achieve a relative position and orientation between their grasp frames. It can be specified using the following sub-tasks \mathcal{T}_t and \mathcal{T}_θ .

- \mathcal{T}_t : Make the position of a point bound to the grasp frame of *Robot*₁ and that of a point bound to the grasp frame of *Robot*₂ coincide. Let \mathbf{x}_1 and \mathbf{x}_2 be the respective positions of these points in an absolute frame. The measure we control is $\mathcal{T}(\mathbf{q}) = \mathbf{x}_2(\mathbf{q}_2) - \mathbf{x}_1(\mathbf{q}_1)$. The Jacobian of the task is the relative Jacobian at points \mathbf{x}_1 and \mathbf{x}_2 .

- \mathcal{T}_θ : Align two axes, one bound to each grasp frames. Let \mathbf{u}_1 and \mathbf{u}_2 be the unit vectors of these axes in an absolute frame. We command the measure $\mathcal{T}(\mathbf{q}) = \mathbf{u}_2(\mathbf{q}_2) - \mathbf{u}_1(\mathbf{q}_1)$. The Jacobian of the task is similar to that above, but restricted to rotations.

A subtask of type \mathcal{T}_t induces three scalar constraints. A subtask of type \mathcal{T}_θ gives two additional independent constraints. Another \mathcal{T}_θ that consists in aligning two axes, which define a compatible geometric relationship with the one above, results in imposing six scalar constraints in all. Hence for a given grasp of *Robot*₁ on the object, the position and orientation of the grasp frame of *Robot*₂ are specified relative to a frame bound to the object. In the case of two six degree of freedom manipulators, regrasping of an object defines a six-dimensional manifold of configurations where the task can be performed (the codimension of the solution manifold equals the rank of the task jacobian).

Symmetries of the task can lead to interesting simplifications. With the example of figure 13, due to the cylindrical symmetry of the object, a subtask \mathcal{T}_t and one subtask \mathcal{T}_θ are enough to describe regrasping. Hence the manifold of solution configurations is 7-dimensional : the task is now easier to perform, as there is inclusion of the respective manifolds.

Once a rigid relationship between grasp frames is realized, we can maintain it as a set of constraints specified in the definition of the task (equation 13). This describes such a task as two robots cooperating in carrying a heavy load. For two six degree of freedom robots we still have six degrees of freedom left. It is thus possible to specify a final configuration to one of the robots, or a final position and orientation for the load. Our controller will not behave as a master-slave system, but will compute the best motion of both arms for the given task, respecting the geometric constraints on the task, and avoiding collisions.

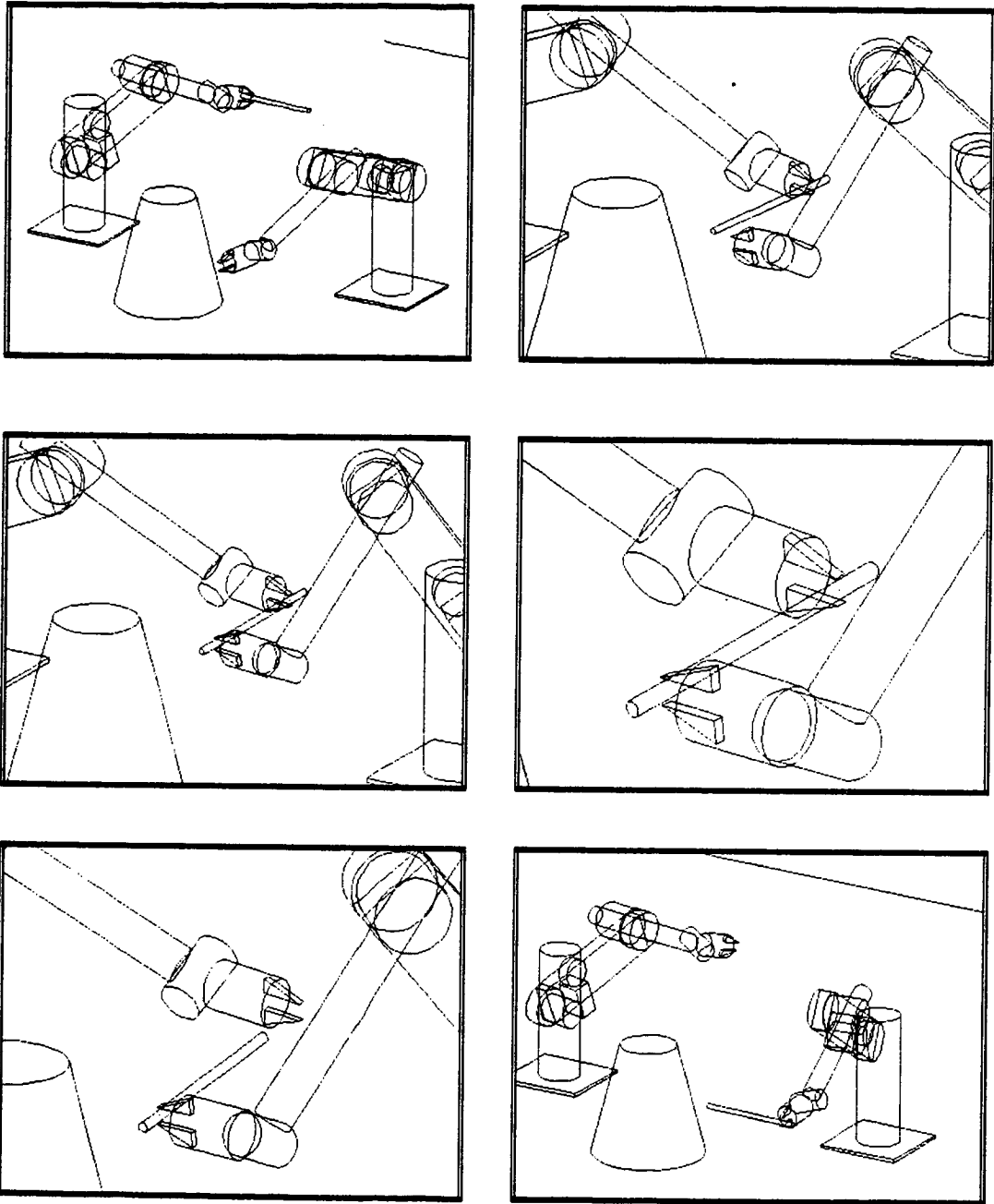


Figure 13: Cooperation of two manipulators : Transfer of a rod. One relation \mathcal{T}_i and one relation \mathcal{T}_θ only need to be specified because of the symmetry of the task.

4 On Global Planning

In this section we apply general tools defined in sections 2 and 3 to the construction of a global model of Free Space. The technique we propose is general, and the model we obtain not too conservative, as the way we “grow” obstacles to ensure safety does take into account the various configurations of the system.

4.1 The Generic Construction of Free Space

We begin by defining a generic construction of Free Space induced by the natural order of the joint variables along the articulated chain. It is based on the hierarchical description of a manipulator introduced in section 2, as a pyramid of $n + 1$ virtual manipulators M^k , $0 \leq k \leq n$. Virtual manipulator M^k is made of the k first bodies B^i , $1 \leq i \leq k$, and of a virtual solid SV^k , describing the volume swept by following bodies when the last $n - k$ joint variables vary freely. The volume occupied by SV^k depends only on the k first joint variables $\mathbf{q}^k = (q^1, \dots, q^k)^t$.

For each of those virtual manipulators M^k , we define the corresponding Free Space FS^k in the k -dimensional Configuration Space $CS^k = Q^1 \times \dots \times Q^k$: it consists in the configuration vectors \mathbf{q}^k such that M^k does not collide with any obstacle in 3D Cartesian space. We also define \widetilde{FS}^k as the subset in CS^k of “possibly free configurations”, the set of configuration vectors \mathbf{q}^k such that none of the k first bodies collide with any obstacle in 3D Cartesian space (that is, virtual bodies are not taken into account). Any set $FS^k \times Q^{k+1} \dots \times Q^n$ provides a conservative approximation of FS . Obviously, \widetilde{FS}^k strictly contains FS^k for $k < n$, and $\widetilde{FS}^n = FS^n = FS$. Furthermore, these sets can be iteratively computed using the relationships :

$$\begin{aligned} FS^1 &= I^1 \\ \widetilde{FS}^1 &= \tilde{I}^1 \\ FS^{k+1} &= FS^k \times Q^{k+1} \cup \{\mathbf{q}^{k+1} | \mathbf{q}^k \in \widetilde{FS}^k \setminus FS^k \text{ and } q^{k+1} \in I^{k+1}(\mathbf{q}^k)\} \\ \widetilde{FS}^{k+1} &= FS^k \times Q^{k+1} \cup \{\mathbf{q}^{k+1} | \mathbf{q}^k \in \widetilde{FS}^k \setminus FS^k \text{ and } q^{k+1} \in \tilde{I}^{k+1}(\mathbf{q}^k)\} \end{aligned}$$

where I^k and \tilde{I}^k are monodimensional subsets of Q^k defined for obstacles O by :

$$\begin{aligned}\tilde{I}^k(\mathbf{q}^{k-1}) &= \{q^k \in Q^k \mid B^k(\mathbf{q}^k) \cap O = \emptyset\} \\ I^k(\mathbf{q}^{k-1}) &= \tilde{I}^k(\mathbf{q}^{k-1}) \cap \{q^k \in Q^k \mid SV^k(\mathbf{q}^k) \cap O = \emptyset\}\end{aligned}$$

Thus only monodimensional free spaces need to be computed for building *FS*. Sets I^k and \tilde{I}^k are indeed finite unions of intervals whose end-points correspond to configurations for which there is contact between a body (real or virtual) and an obstacle. For each body, the list of obstacles it can collide with is initialized using the corresponding leaf of Assembly Tree M^0 , that is, its whole swept volume. The list is pruned during the iterative process by removing the obstacles that do not collide with $M^k(\mathbf{q}^k)$ when computing I^{k+1} and \tilde{I}^{k+1} for a given configuration vector \mathbf{q}^k . Thus only the obstacles possibly yielding a collision are considered when computing monodimensional free spaces.

In practice, parameter ranges are cut into slices : monodimensional free spaces are computed only for a finite number of discrete configuration vectors lying at the center of cuboids of the type $\prod_{i=1}^{k-1} [q_c^i - \Delta q^i, q_c^i + \Delta q^i]$, where Δq^i is half the discretization step for the i -th joint. The sets I^k and \tilde{I}^k must be such that for any value q^k in the set, and for any configuration vector \mathbf{q}^{k-1} in the corresponding cuboid, we obtain a free configuration \mathbf{q}^k . Next section explicits the computation of a monodimensional free space.

4.2 Computation of a Monodimensional Free Space

We now consider a body B^k and one obstacle O , both convex objects. The position of body B^k , either real or virtual, depends on k configuration parameters, and we compute the monodimensional free space in q^k for a discrete configuration vector \mathbf{q}_c^{k-1} at the center of the cuboid $\prod_{i=1}^{k-1} [q_c^i - \Delta q^i, q_c^i + \Delta q^i]$. The method we propose to compute the discrete monodimensional free space is based on a local model of Free Space similar to that introduced in section 3, the only difference being that we use exact distances between objects in place of conservative bounds generated by damper equations. The idea is to count how many elementary cuboids one can stack in the k -th direction while remaining within the local model of Free Space, and then recompute this model at the center of the top and bottom cuboids until no extension of the free interval is possible any more.

More precisely, from equation 6, the variation of the distance between B^k and O writes :

$$\delta d = (\mathbf{n} | \mathbf{J} \delta \mathbf{q}^k) \quad (14)$$

$$= (\mathbf{J}^t \mathbf{n} | \delta \mathbf{q}^k) \quad (15)$$

$$= \sum_{i=1}^k \nu^i \delta q^i \quad (16)$$

If the distance to the obstacle equals d for configuration vector \mathbf{q}_c^k , we deduce that all configurations inside the cuboid are safe as long as q^k verifies inequation :

$$\nu^k q^k \geq \nu^k q_c^k - d + \sum_{i=1}^{k-1} |\nu^i| \Delta q^i. \quad (17)$$

Depending on the sign of ν^k this yields either an upper or a lower bound on q^k . In the case of a revolute joint, another bound must be imposed in order to insure the validity of the first order approximation of the distance. A new distance computation is performed for the more constraining of the two bounds, and the process is iterated until the remaining free interval becomes smaller than Δq^k .

Strong points of this technique are, first, that it is applicable to all types of articulated chains, second, that the way we “grow” moving bodies to ensure safety of the model of Free Space is not too conservative : coefficients ν^i are computed for the exact value of the Jacobian, thus taking into account the various configurations of the system. This method is as well applicable to the computation of configuration obstacles generated by collisions between two moving bodies, either part of the same kinematic chain, or of two different robots.

4.3 Structuration of the Data in a 2^n -tree

This generic construction yields a tessellation of Free Space in very thin cuboids, some of their sides being necessarily as small as the discretization step in that direction. The structuration will aim at reconstructing a more homogeneous representation (cf. [Fav86]). The structure we use is a 2^n -tree, generalization of the *quadtree* introduced by Samet to hierarchically represent binary arrays [Sam84]. It is a tree of degree 2^n whose nodes at level l are n -dimensional cuboids of size 2^{n-l} . Each node is labelled as *empty*, *full* or *mixed*. Mixed nodes only are split up into their 2^n sons, until they become of unit size (Figure 14).

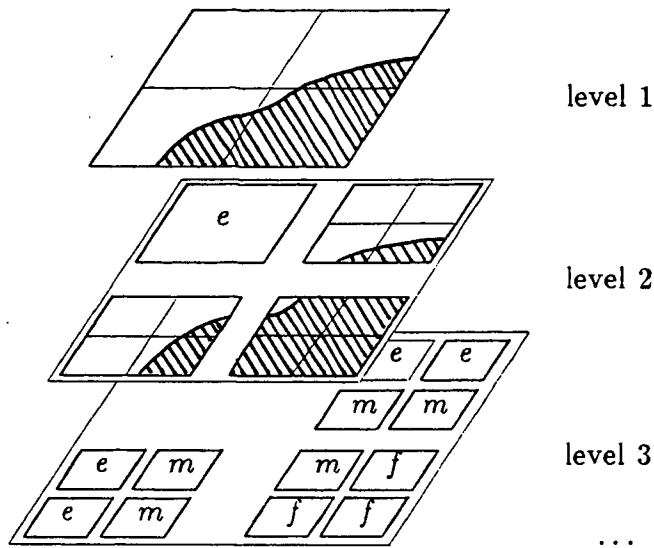


Figure 14: Representing Free Space in the plane with a *quadtree*. Labels *e*, *f* and *m* stand respectively for *empty*, *full* and *mixed* nodes.

In our context, such a structure presents several advantages :

- *Isotropy* : That is, all joints are treated equally.
- *Hierarchy* : Big cells will be sufficient far from configuration obstacles, small ones being only needed to describe in details their boundaries. Thus, when looking for a gross motion, one can use big cells only to obtain quickly a path along which the robot remains far from obstacles. The 2^n -tree data structure also enables efficient boolean operations : this is useful to merge the structures computed for different parts of the environment of the robot.
- *Regularity* : This enables to have access to the neighbors of a cell without explicitly storing the connectivity graph, which is most important considering memory limitations.

Figure 16 shows the transforms of obstacles in the 3D Configuration Space representing the first three links of the manipulator in the environment of figure 15. Figure 17 gives the corresponding *octree* representation of Free Space, at levels 4, 5 and 7 respectively. The total number of nodes for representing Free Space is 10 591, which would correspond

Level	3	4	5	6	7	Total
% nodes	0.06	0.93	5.08	13.55	80.38	100%
% Free Space	19.0	29.0	26.5	19.0	6.5	100%

Table 1: Distribution of the nodes in Free Space.

to 129694 cells of level 7. Table 1 gives the percentage of nodes at each level, and the corresponding percentage of Free Space volume. Note in particular that 93.5% of Free Space is described by nodes up to level 6, with 20% of the nodes only.

4.4 Search for a Collision Free Trajectory

When computing a trajectory between an initial and a goal configuration, we first search for a path in the connectivity graph of the 2^n -tree, which yields a list of cells to be traversed. The search is performed with an A^* algorithm, the criterion used being usually the Manhattan distance between the centers of the cells, and the heuristic function h^* , estimate of the exact cost function h between a node and the goal, the Cartesian distance to the goal. We will see in subsection 4.6 that other heuristics can be used in the case the problem is described by a task function.

As mentioned earlier, a lower bound on the size of the cells to be examined during the search can be imposed. The number of nodes explored is then much smaller, and the search faster, but some solutions in constrained regions of the environment can be lost. However, in most practical applications, one will prefer a safer but longer trajectory than a shorter but very constrained one.

Once a list of free cells has been found, we must still compute a trajectory joining the initial and the final configurations, that remains within those cells. We have chosen to search a trajectory among polygonal lines, using the following recursive procedure :

1. Hypothesize the line segment joining the two end-configurations as a trajectory. If it intersects all the faces common to two adjacent cells, it is a feasible trajectory.
2. Else, on each face not intersected by the line-segment, compute the point that min-

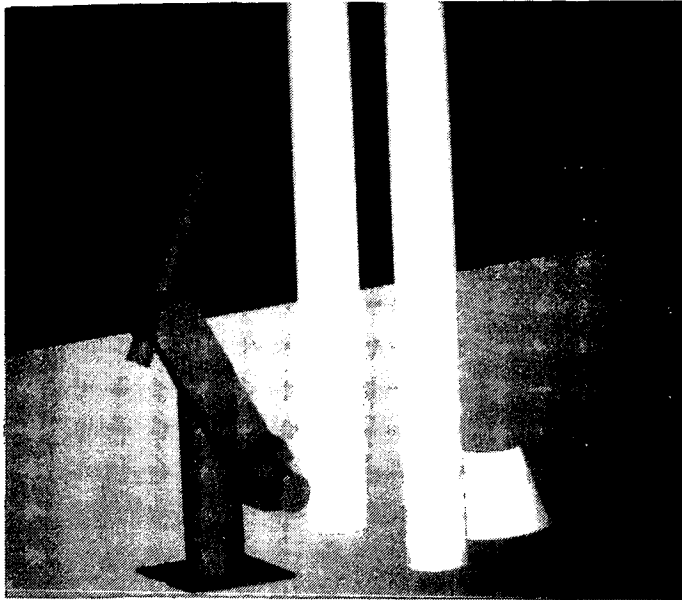


Figure 15: Robot and obstacles in Cartesian Space.

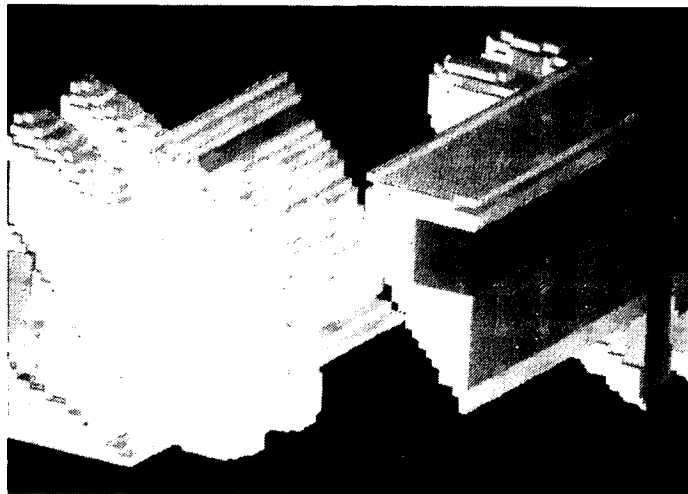
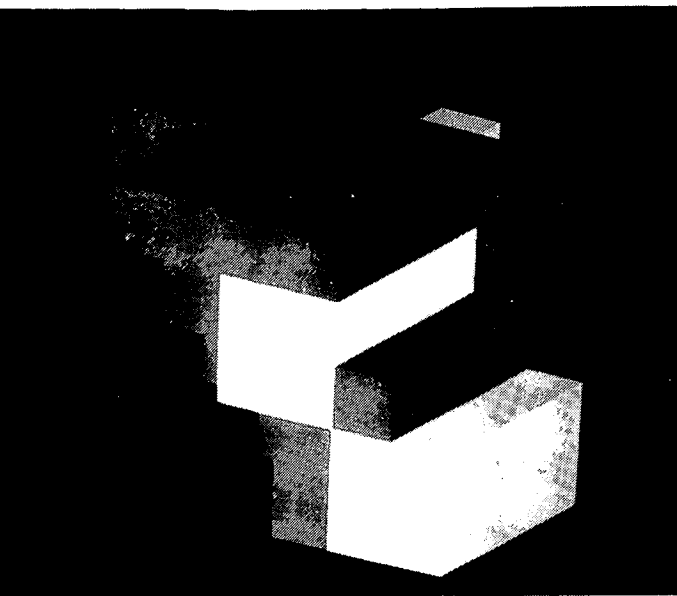
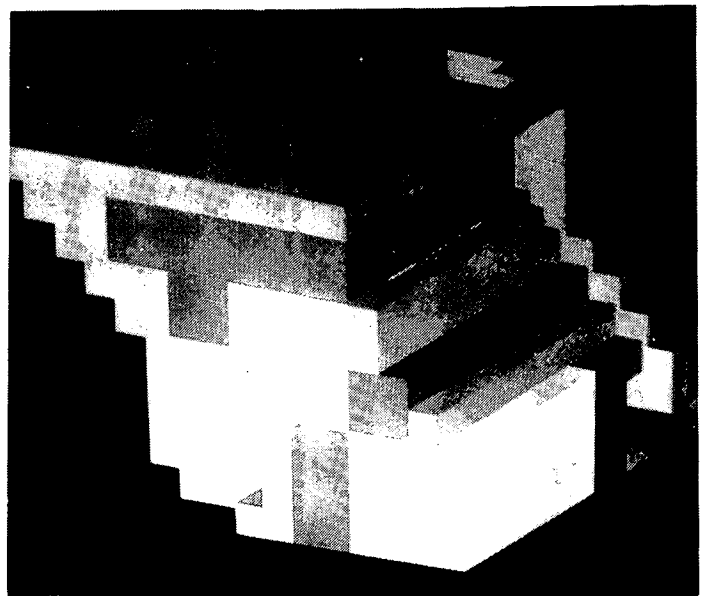


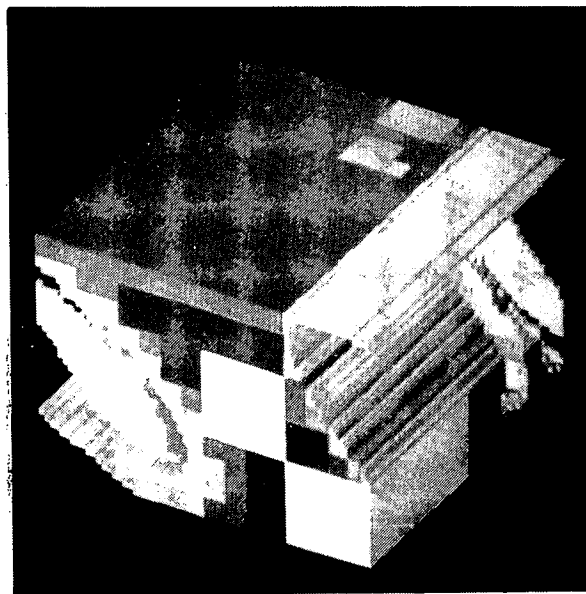
Figure 16: Transforms of obstacles in the Configuration Space of the first three links : we see clearly the transform of the work-table (two cylinders of axis q^1 corresponding to two postures of the robot) and those of the two pillars.



(4)



(5)



(7)

Figure 17: *Octree* representation of Free Space at levels 4, 5 and 7. The freeway between the pillars appears at level 5.

imizes the sum of the length of the two segments to the end-points.

3. Choose the point yielding the largest sum as an intermediate configuration, and call recursively the procedure on both line segments.

In general, this procedure yields a trajectory with few intermediate points. The speed along each line segment can be set according to the size of the corresponding empty cells : the larger the cell is, the faster one can move safely.

4.5 Heuristics for the hand

Although in theory there is no bound in the dimension of the Configuration Space, one is limited to three or four degrees of freedom in practice, because of the combinatorial explosion due to the discretization. However, most of the industrial robots can be decomposed into an arm with three joints, and a hand articulated at the wrist. The hand is usually much smaller than the arm, and does not play an essential part in the search for a gross motion. It can then be replaced by virtual body SV^3 , the volume swept by the last three links when joints q^4, q^5, q^6 vary freely. As the hand is usually close to obstacles at both ends of the motion, a local method is nevertheless needed to compute a motion that pushes the hand away from obstacles. This yields a three step procedure for planning the motion of a six joint manipulator :

1. Use a local method to compute new initial and final configurations away from obstacles. The task consists in maximizing the distance to obstacles, under the collision avoidance constraints.
2. Search for a gross motion for virtual manipulator M^3 . This step yields a list of cells in the *octree*.
3. Search for a collision free trajectory, using the procedure of subsection 4.4. The trajectory in the six dimensional space is obtained by simple linear interpolation for the last three joints.

This algorithm works quite well for manipulators having small hands and moving small objects. If it is not the case, one can no more replace the hand by its real swept volume,

as in many cases no trajectory at all would be found. However, we still influence the construction of the *octree* by using a virtual body to represent the hand, in general a sphere centered at the wrist, smaller than the real swept volume. We then modify the two last steps of the algorithm above as follows :

- In step 2, we add an additional cost to the criterion, that measures the violation of the constraints of the 6D Free Space local model, for an interpolated joint vector. Different orientations of the hand, for the same start and goal cells, may now yield different lists of cells in the *octree*.
- In step 3, we add the constraints generated by the local model of Free Space, to compute intermediate points and test collisions along line segments.

This algorithm may fail in the last step because no correct orientation can be found for the hand in the proposed list of cells. However, it does yield correct trajectories in many practical cases. Figures 18 and 19 show some intermediate configurations computed for two problems with the same initial and final positions of the wrist, but with different orientations of the hand.

With this environment, computation time is of the order of five minutes for the construction of Free Space and its structuration, and ten seconds for the search of a collision-free trajectory.

4.6 When the Goal is Described by a Task Function

Isotropy of the representation of Free Space by a 2^n -tree makes it possible to adapt the search for problems described by a task function (as defined in subsection 3.5).

First, recall that the A^* algorithm works as well with a *list* of goal nodes, if for all of them the heuristic cost $h^*(n)$ equals 0. If in addition $h^*(n) \leq h(n)$ for all nodes of the graph, heuristic function h^* is said to be *admissible*, and the A^* algorithm yields an optimal path from an initial node to the goals.

Of course, the more the heuristic cost sticks to the exact cost $h(n)$, the more the search will be efficient. We evaluate a cell C of the 2^n -tree with the norm of the task vector

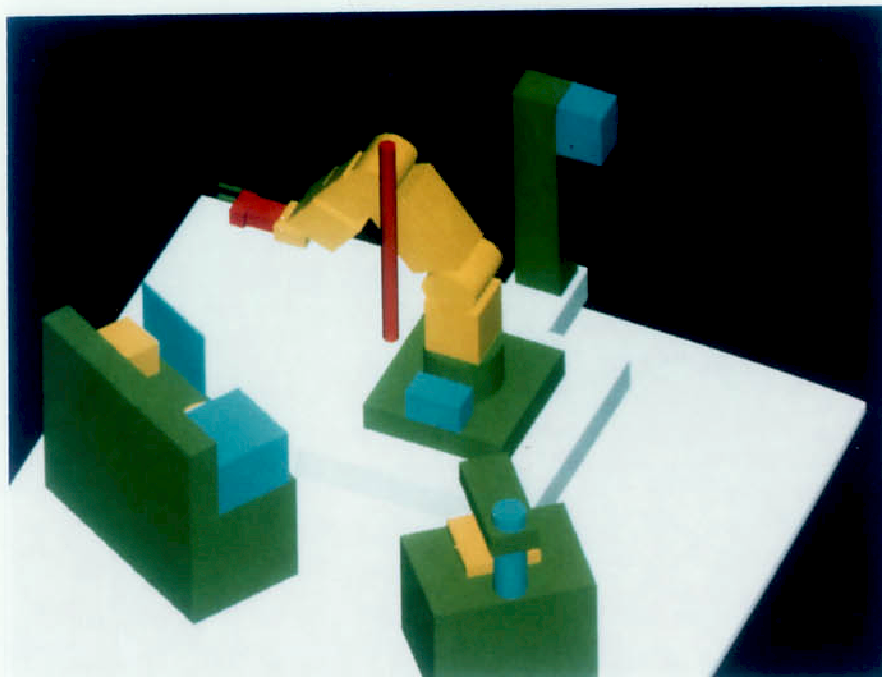
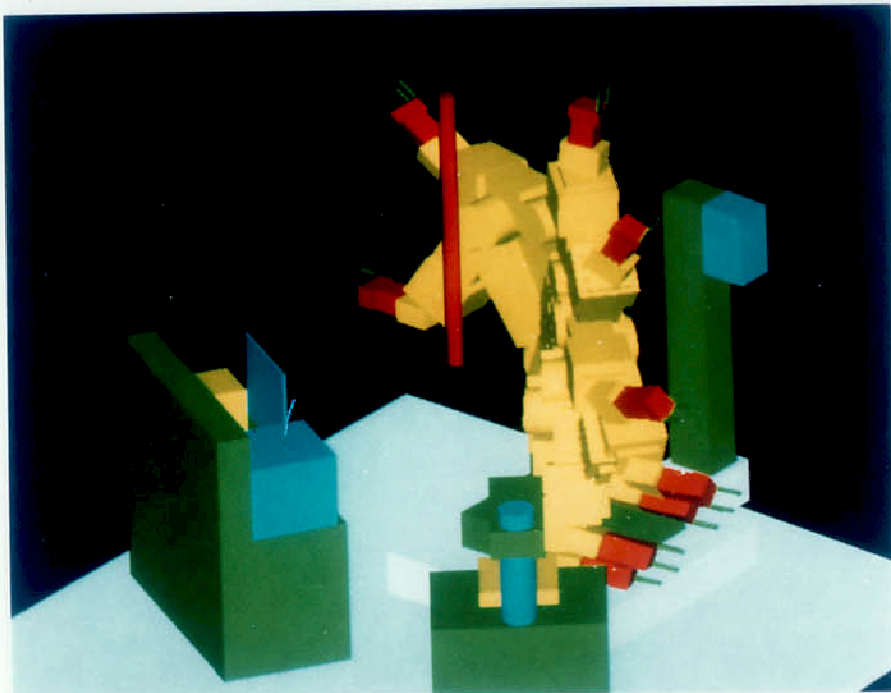
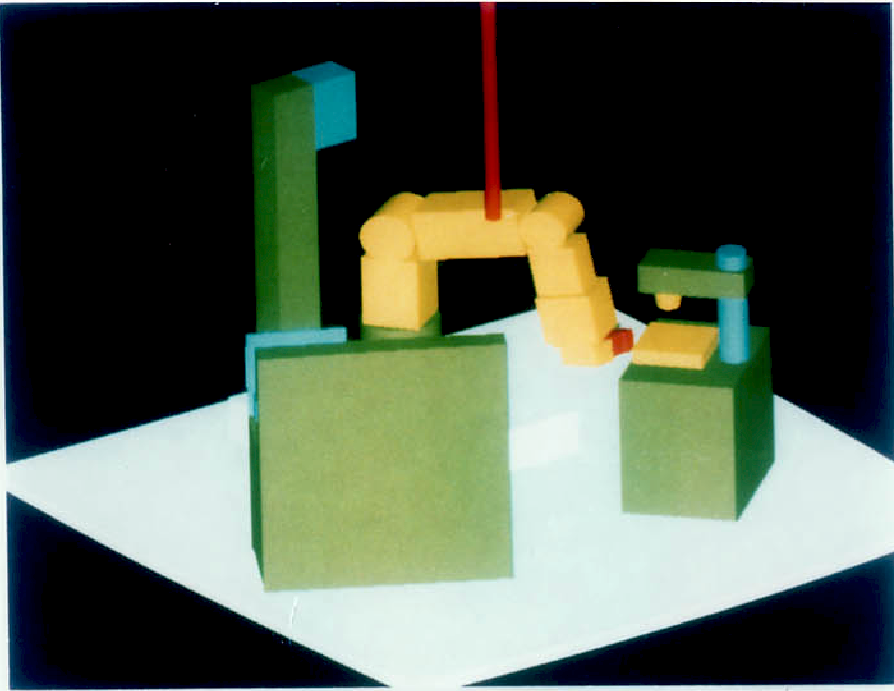


Figure 18: Initial, intermediary, and final configurations.

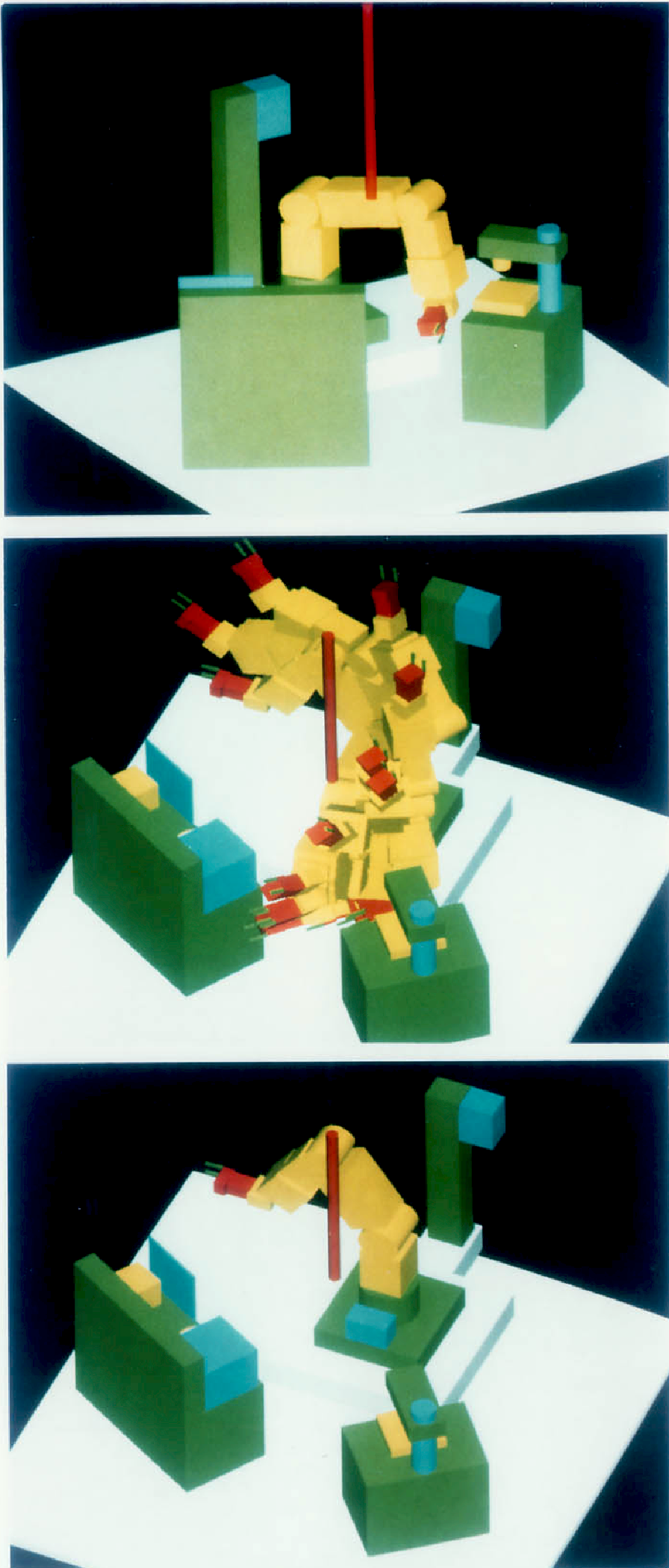


Figure 19: Another trajectory with the same initial and final configurations of the arm, but with different orientations of the hand.

computed at the center of the cuboid, $\|\mathcal{T}(\mathbf{q}_c)\|$, and we guide the search with the heuristic cost function : $h^*(C) = \frac{1}{\lambda}\|\mathcal{T}(\mathbf{q}_c)\|$.

It is easy to see that this heuristic function is admissible for :

$$\lambda \geq \max_{\mathbf{q} \in Cs} \|\mathbf{J}_r(\mathbf{q})\|, \quad (18)$$

with $\|\mathbf{J}_r(\mathbf{q})\|$ the algebra norm of the Jacobian of the task for configuration \mathbf{q} .

A last problem is that of the practical termination of the algorithm. The global planner is coupled with a trajectory generator as the one described in section 3. When we reach during the search a node of the graph such that the norm of the task vector at the center of the cell is less than a threshold ε , we run the local planner and check if we are able to reach a goal configuration. If not, the global search is pursued after we arbitrarily increase the value of the task vector for that node. The value of threshold ε must fit the size of the cell. If there exists inside cell C some configuration \mathbf{q}_g solution for the given task \mathcal{T} , then a first order Taylor expansion yields :

$$\mathcal{T}(\mathbf{q}_c) = -\mathbf{J}_r(\mathbf{q}_c) (\mathbf{q}_g - \mathbf{q}_c) + o(\|\mathbf{q}_g - \mathbf{q}_c\|). \quad (19)$$

A reasonable termination condition thus consists in checking :

$$\|\mathcal{T}(\mathbf{q}_c)\| < \Delta \|\mathbf{J}_r(\mathbf{q}_c)\|, \quad (20)$$

with Δ of the order of half the width of a cell, $\max_{i=1,\dots,n} \Delta q^i$.

5 A Mixed Approach to Planning

On the one hand, global constructions of Free Space are limited by combinatorial explosion with the number of degrees of freedom. On the other, local methods are limited in their scope of applications by the eventualities of dead-locks. One obvious solution would be to give the local method “good” subgoals so that it avoid dead-ends. In this section, we propose to decompose the general problem of path-planning for manipulators with many degrees of freedom into :

- a *low complexity local planner*,
- and a *higher complexity global planner* working on a graph of cells representing relatively large regions of Configuration Space.

We will describe in details the interactions between the two levels, which result in learning automatically good subgoals for the local planner.

The main idea underlying this approach is to take advantage of the power of the local method to follow configuration obstacles boundaries, so as to deal with the high complexity of global planning only at the relevant level of the description. At the global level, no geometric description of the obstacles is stored, but only weights indicating the probability that a trajectory computed locally does not lead to a dead-end. Let us emphasize that this approach is relevant only when it is performed using a coarse quantization of Configuration Space, otherwise it is not better than a classical global approach in terms of computational cost. Dealing with a loose graph is made possible only because of the intrinsic power of the local planner that produces long segments of collision free trajectory.

5.1 A Probabilistic Model of Safe Displacements

We assume that an algorithm that locally computes segments of safe trajectories is available. We write $\mathbf{q} = (q^1, \dots, q^n)^t$ a n dimensional configuration vector describing the state of the system, and partition Configuration Space into cells of the type :

$$C_k = \prod_{i=1}^n [q_k^i - \Delta q^i, q_k^i + \Delta q^i].$$

Again (q_k^1, \dots, q_k^n) are the configuration parameters of the center of cell C_k , and Δq^i equals half the width of a cell in the i -th direction. Starting with this description, we build a graph whose nodes stand for the cells themselves, each node having $2n$ neighbors. It is called the *State Graph* in the sequel. When the robot configuration lies inside a cell C_k , it is said to be in state C_k .

As opposed to a standard Configuration Space approach, we do not label nodes as *free* or *intersecting configuration obstacles*. This would not be relevant as the grid is based on a coarse discretization of parameter ranges, and we want to be allowed to navigate in partially occupied cells.

For each oriented transition between two neighboring cells C_k and C_l in the State Graph, we only memorize a weight that estimates the difficulty for the robot to enter cell C_l when coming from cell C_k . More precisely, we define p_{kl} as the probability for the local planner to succeed in making the robot enter cell C_l from neighboring cell C_k , when aiming at some point located inside C_l .

We call any connected sequence of nodes of the State Graph a *path*. The probability for a path to yield a successful trajectory (with no dead-lock) is defined as the product of probabilities $p_{k_m k_{m+1}}$ along the path. This implies a hypothesis of independence, namely that the probability of realizing a transition is independent of the sequence of nodes we followed so far. In practice, we attribute to each arc of the graph a weight equal to $-\log p_{kl}$, and we minimize the cost

$$g = - \sum_{m=0}^{M-1} \log(p_{k_m k_{m+1}}) \quad (21)$$

for traversed transitions along the path.

Initially, in absence of any information on its environment, the robot initializes the graph with given *a priori* probabilities $p_{kl} = \bar{p}$, $0 < \bar{p} < 1$. Hence the path we compute first is a path of minimum length in terms of the number of cells traversed from the initial configuration to the goal. Later as probabilities vary to reflect the robot's accumulated knowledge of its environment, the path we compute realizes a compromise between minimum distance and the assurance that we will reach the goal.

5.2 The Interface Between the Local and Global Planners

In the simplest implementation of the algorithm, the interactions between both levels are quite limited : the global planner sends subgoals to the local planner, and observes successive configurations of the robot to run the learning process.

5.2.1 Choosing Subgoals on the Path

When executing a path, a simple choice of a subgoal is the center of next cell on the path. However, this may yield a trajectory similar to a drunkard's walk because of the coarse discretization. In order to avoid this reprehensible behavior, we propose another choice that gives a smoother trajectory. The idea is to locally optimize the length of the trajectory. So, the subgoal is chosen as the point on the face common with next cell that minimizes the sum of :

- the distance from the current position to this point,
- and the distance from this point to the center of one of the cells farther on the path.

The farther this cell is chosen, the more we anticipate future displacements. In the case the robot is blocked, nearer cells are used in order to provide new subgoals. An example is shown in figure 20.

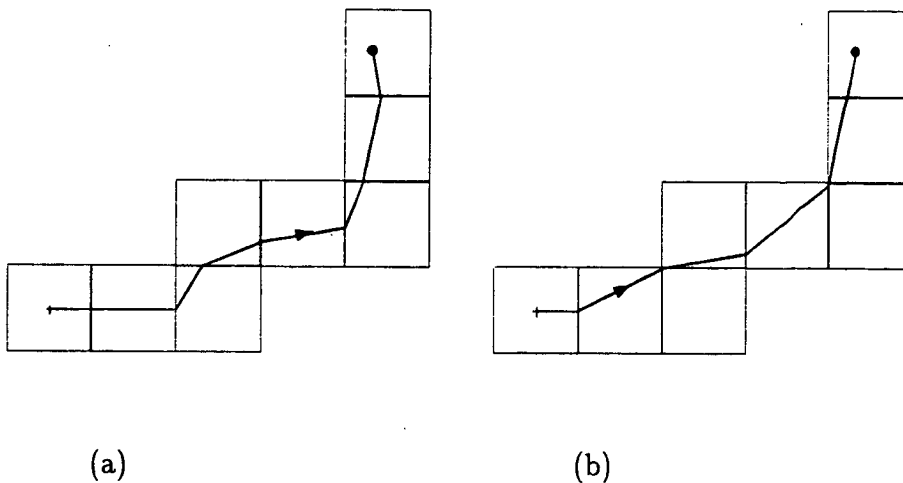


Figure 20: Computation of subgoals : (a) Trajectory with no anticipation. (b) Trajectory with a one-cell anticipation.

5.2.2 Updating Probabilities During Path Execution

Updating of the transition probabilities is first performed whenever the motion supervisor detects a transition in the State Graph. If *the robot performed the transition ordered by the global planner*, the corresponding probability is increased. Else, *the robot has been deflected from its path by an obstacle*, and we decrease the probability of the desired transition. In this case, we must also modify the path so that it remains connected. This is done by adding to the path the current node, and the common neighbor of the current node and next cell on the path (see Fig. 21).

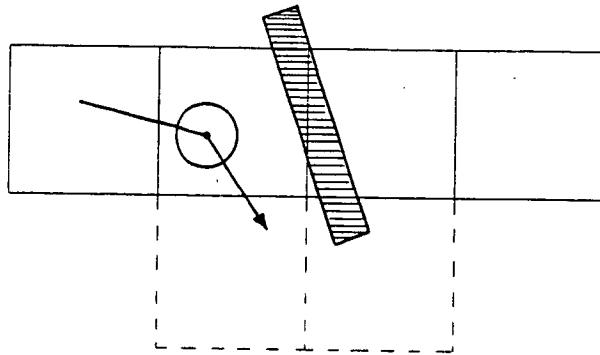
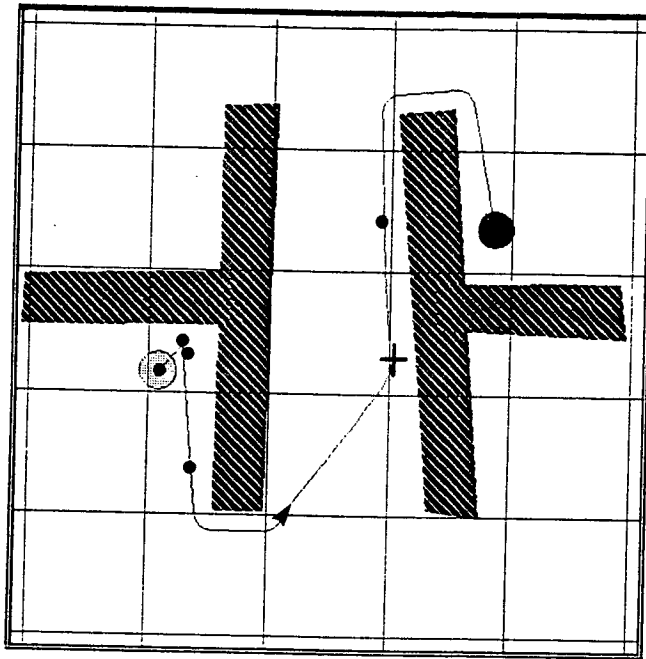


Figure 21: Modification of the path when the actual transition is not the desired one : additional cells are drawn with dotted lines.

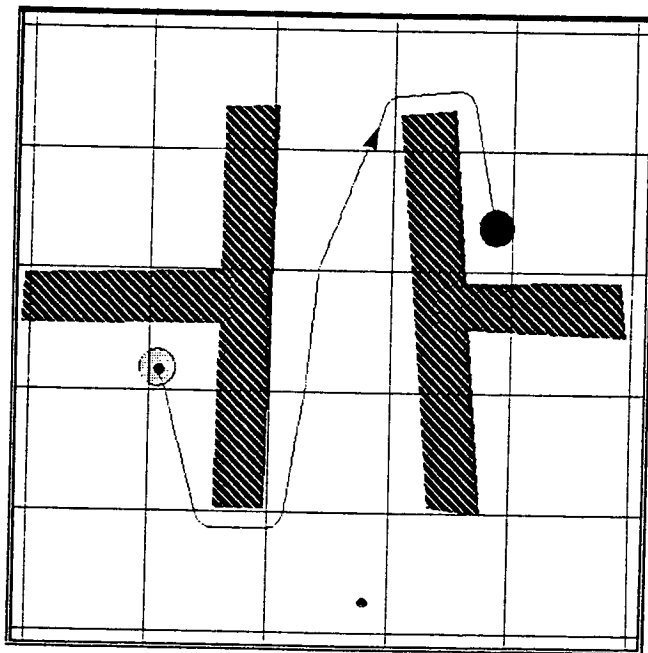
Updating the transition probabilities is also performed whenever *the robot is blocked*. If it is not inside the goal node, we decrease the probability of the desired transition and invoke the global planner with the updated State Graph. As the probability of the problematic transition has been decreased, it will eventually be avoided by the newly computed path. Figure 22 illustrates this learning process for a simple planar example.

If a dead-lock occurs inside the goal node, the goal is declared not reachable. This event may have several causes :

- the goal lies inside an obstacle (or another connected component of Free Space).
- the local method failed to reach the goal because of a difficult layout of obstacles inside the goal node.



(a)



(b)

Figure 22: a) First execution. b) Second execution with the same initial and goal configurations. Circles indicate points where calls to the global planner were needed, the cross a point where the robot was deflected from its path.

In the latter case, a solution would be to move the robot outside of the goal node, setting a subgoal in a neighbor chosen from local information, before aiming again at the goal. This strategy might succeed in making the robot pass round the obstacle. But the main problem in fact is to distinguish between the two cases above. Elements of the answer are given in subsections 5.5 and 5.7.

5.3 A Bayesian Approach for Learning

We call p_{kl} the probability for the robot to enter cell C_l coming from neighboring cell C_k , if it is aiming at some point located inside cell C_l .

When the local planner is realizing such a scheme, two types of events may happen :

- $(x_{kl})^+$ or success in entering cell C_l , event of probability p_{kl} ,
- $(x_{kl})^-$ or failure in entering the cell, event of probability $1 - p_{kl}$.

A failure occurs either when the robot is blocked in its current cell, or when it is forced by the obstacles to enter another neighbor of C_k before C_l . In the following, we omit the subscripts kl for clarity.

We want to estimate the probability p associated with a given transition from the events that happen during executions of trajectories. We use a Bayesian approach as follows.

Let us suppose that e events have happened so far that concern the given transition, $x_i, i = 1, \dots, e$. The value of x_i is 1 in case of success and 0 in case of failure. We denote \mathbf{X}_e the vector (x_1, \dots, x_e) and f the probability distribution function. The Bayesian learning model states :

$$f(p|\mathbf{X}_{e+1}) = \frac{f(x_{e+1}|p) f(p|\mathbf{X}_e)}{\int_0^1 f(x_{e+1}|p) f(p|\mathbf{X}_e) dp}. \quad (22)$$

In our case, $f(x_{e+1}|p)$ equals p for $x_{e+1} = 1$ and $(1 - p)$ for $x_{e+1} = 0$, which can be written :

$$f(x_{e+1}|p) \stackrel{\text{def}}{=} p^{x_{e+1}}(1 - p)^{1-x_{e+1}}. \quad (23)$$

Thus formula 22 can be rewritten :

$$f(p|\mathbf{X}_{e+1}) = \left(\frac{p}{\bar{p}_e}\right)^{x_{e+1}} \left(\frac{1-p}{1-\bar{p}_e}\right)^{1-x_{e+1}} f(p|\mathbf{X}_e), \quad (24)$$

where \bar{p}_e denotes the mean value of p after e events, that is :

$$\bar{p}_e = \int_0^1 p f(p|\mathbf{X}_e) dp \quad (25)$$

Formulae 24 and 25 are used to compute the probability distribution law of p after e events, and its mean value that we will use as an estimate of p . When we have no *a priori* knowledge of the environment, we initialize the probability distribution with the uniform distribution law, that is $f_0(p) = 1$ for p in $[0,1]$, and $\bar{p}_0 = 0.5$.

With this model, we can show that the mean value of p after e events, s of them being successful, is simply :

$$\bar{p}_e = \frac{s + 1}{e + 2} \quad (26)$$

The absence of knowledge that we have modeled by a uniform probability distribution law can be interpreted simply from this formula : the initial state of the system is the same as if we had already made two trials, one of them being a success, and the other a failure. This interpretation is helpful if we want to incorporate *a priori* knowledge : p_0 can be initiated to s_0/n_0 , where s_0 is the number of successes for n_0 hypothetical trials.

As the events x_e are assumed to be independent, the probability distribution law does not depend on the order in which the events happen, and thus all the memory of a transition is expressed by the pair (e, s) . Rules to update probabilities in the three cases of subsection 5.2 are then very simple.

- *the actual transition is the desired one* : we increment by 1 the number e of events and the number s of successes.

- *the robot is deflected from its path, or is blocked* : we increment only the number of events for the desired transition.

5.4 More Information from the Local Model

The idea outlined in this section is to acquire more information from the local model, to accelerate the updating of the graph when a dead-lock occurs. Recall that we work in a high dimensional space : in the case of a failure of the local planner, several transitions with neighbors of the current node may yield similar costs, and more than one call to the global planner is needed in general to make the system move again significantly.

Before searching for the best global path, we thus execute *virtual moves* in all the directions of the $2n$ neighbors, except the one we were aiming at that led to a dead-lock. Probabilities for each of those transitions are then updated by increasing by 1 the number

of events, and the “number of successes” by the ratio $\delta q^i / \delta q_{max}^i$, where δq^i is the increment effectively computed by the local planner on the i -th parameter, for desired increments $(0, \dots, \delta q_{max}^i, \dots, 0)$.

The search algorithm now tends to favor those transitions around the current configuration that are *a priori* safer for the local planner, and one or few calls to the global planner are sufficient for the system to start moving again.

5.5 Heuristics for the Global Planner

When searching for a path from an initial to a goal cell, we use an A^* algorithm maximizing the product of probabilities p_{kl} along the path. This translates into minimizing the cost $g = -\sum_m \log(p_{k_m k_{m+1}})$ for traversed transitions along the path from initial cell C_0 to goal cell C_g . The heuristic cost we use to guide the search is for a node C_k the number of cells N_k that are traversed by a straight line to the goal, weighted by the log of the average probability \bar{p} for all transitions of the State Graph :

$$h_k^* = -N_k \log \bar{p}, \quad (27)$$

This heuristic function is generally not admissible, but gives good results in our experiments. It can be seen as an estimate of the minimum cost we expect if we suppose that the difficulty to move around is about the same everywhere.

It must also be noticed that since all weights are finite in our model and the graph is connected, a path is always returned by the global planner, even when the goal intersects an obstacle, or lies in another connected component of Free Space. As we do not want the robot to roam around indefinitely while trying to reach the goal, we have to decide *a priori* about the feasibility of the trajectory from the cost returned by the search algorithm.

We impose a threshold on the ratio of the optimal cost and the heuristic cost, when we perform the planning starting from some cell C_k :

$$\frac{g_k^*}{h_k^*} < \chi. \quad (28)$$

The heuristic function can be seen as the expected cost in the case of a uniform distribution of the difficulty to move in the environment. Thus, if the optimal cost actually computed is much higher than this estimate, it means that the optimal path is much

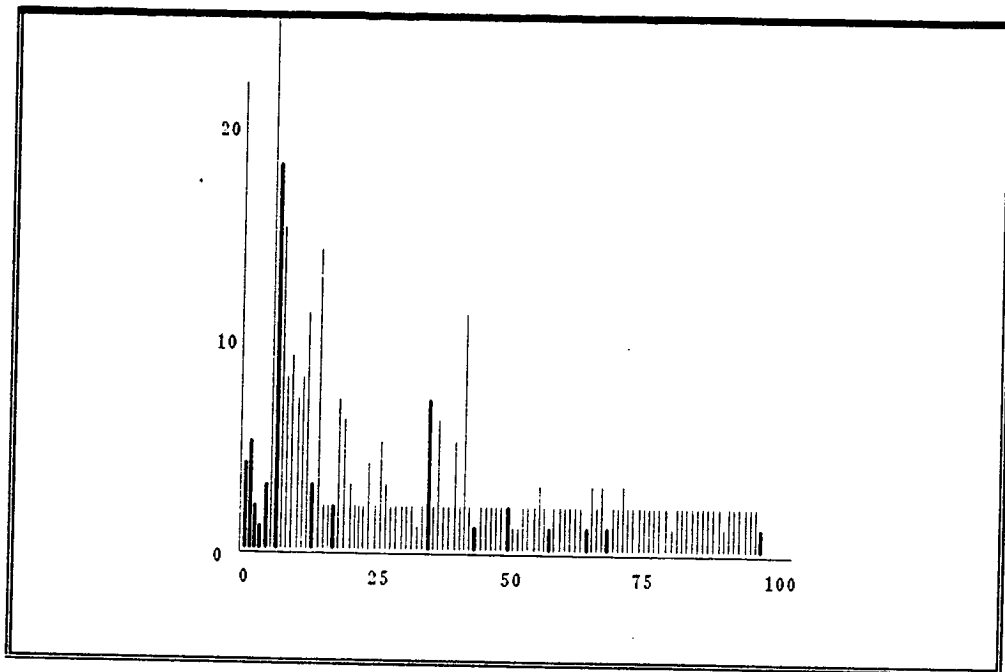
longer than the expected one, or that there exists on the optimal path one or more transitions with a low probability of success. So, the constant χ measures the relative length of the detour we tolerate in order to have a reasonable chance to succeed. The more the environment resembles a labyrinth, the higher χ has to be set.

5.6 Exploration

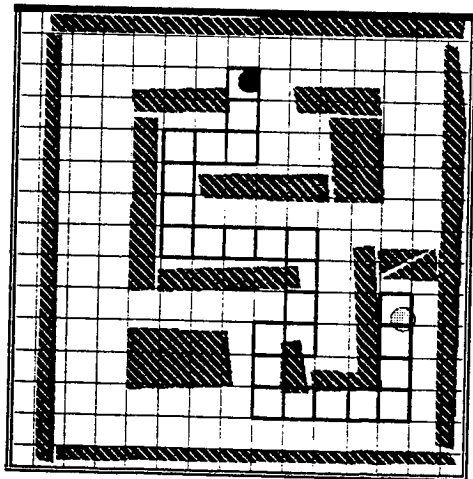
Suppose the robot has no *a priori* knowledge of its environment. We want it to behave properly after a number of executions of trajectories. A straightforward way to proceed is to generate goals inside one of the nodes with the smallest history. The *history* of a node C_k is simply defined as the sum on j of numbers e_{jk} of events that have so far occurred for transitions from neighboring nodes C_j of C_k . The robot will thus try to explore first those nodes for which it has least information.

When the path towards the goal is executed, the history of all traversed nodes is updated at the same time as transition probabilities. A difficulty arises when the path is declared not feasible. This does not give any relevant information on the difficulty to enter the goal node from a neighbor, but we increase its history although the corresponding transitions are not updated, so that the robot does not try again and again to enter a node that it cannot reach. This conveys information such as "this goal configuration probably lies inside an obstacle or another connected component of Free Space".

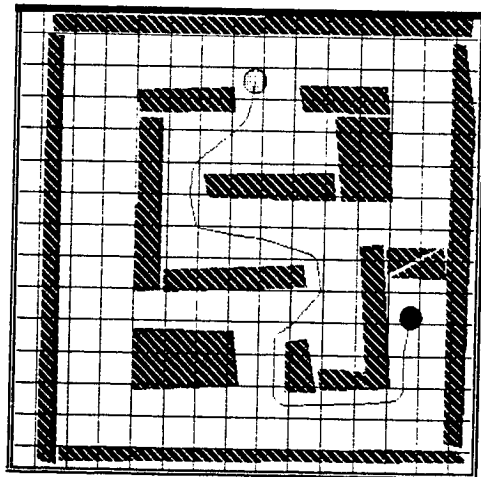
The exploration process ends when all nodes have received a history higher than a specified threshold. Figure 23 illustrates the exploration process, for a sequence of 100 trajectories with goals generated at random in nodes with lowest history. Here the threshold χ for the ratio g_k^*/h_k^* was set to 3. The histogram shows the evolution of the number of calls to the global planner along one trajectory. We observe that the first 50 trajectories were the most significant. Thick lines correspond to trajectories for which the goal was successfully reached, thin lines to failures. It is clear that in the end most of the goals are non reachable, as we select them at places where the robot has least navigated. The impossibility to reach the goal is then generally detected in two calls to the global planner : the first call to get as close as possible from the goal while staying in the current connected component of Free Space, the latter to decide that its cost is too high.



(a)



(b)



(c)

Figure 23: Exploration : a) Histogram for the first 100 trials. b) A path computed after 50 trials. c) The corresponding trajectory.

5.7 Adaptation of the Size of the Grid

In order to decrease the number of nodes in the graph, it may be useful to adapt the discretization of the Configuration Space to the planning difficulty in the various regions. This can be done by analyzing the variation of the estimated transition probabilities. Indeed, if a cell is too large, its transition probabilities will not converge towards 0 or 1, because of alternate success and failure. Such transitions are characterized more precisely by a high value of the entropy, measured by $-\bar{p}_e \log \bar{p}_e - (1 - \bar{p}_e) \log(1 - \bar{p}_e)$, or, more simply, by :

$$\bar{p}_e(1 - \bar{p}_e) = \frac{(s + 1)(e - s + 1)}{(e + 2)^2}. \quad (29)$$

To solve this problem, it is necessary to split the corresponding node into several new cells, thus refining the description of this part of the environment at the global level.

Following section 4, we propose to make use of a 2^n -tree in place of a regular grid. Using such a structure in this context may proceed as follows. Exploration starts with a regular grid corresponding to a relatively high level in the 2^n -tree. When an event occurs for a given transition, we first look at the corresponding entropy and compare it to a threshold (a decreasing function of time). If the entropy is smaller than the threshold, updating is performed as usual. Else, the node we aim at is split into its sons and new transitions are initialized as detailed below.

- Transitions between sons and neighbors of the father are initialized assuming that the events that happened to the former transition are uniformly distributed on the 2^{n-1} new transitions.
- Transitions between two sons are initialized with the *a priori* uniform law.

This process can be used similarly for solving the problem of a dead-lock occurring inside the goal node.

Splitting is performed until the size of a node is smaller than a threshold, function of the ability of the local method to reach a subgoal at a distance corresponding to that threshold.

A question naturally arising is whether one should now influence the cost function considering the width of a cell. We argue that we should not, as long as the hypothesis

of independence of the various transitions stand, which implicitly means that we will use smaller cells only in cluttered areas.

5.8 Two Six Degree of Freedom Examples

5.8.1 Coordinated Motions of two Manipulators Moving Among Obstacles

In this first example, the robotic cell is composed of two 6 degree of freedom manipulators. The local planner builds at each time increment a local view of the obstacles in the 12D Configuration Space of the system, and moves the robots safely. We couple this trajectory generator with a global planner that makes use of a model of the first three degrees of freedom on both articulated chains, the most significant for gross motions of the system. The planner will thus reason on a graph of 6D cells, product of the 3D graphs representing independent motions for each arm.

A 6D cell C is valuated with the probability of success for the trajectory generator to make the system enter this cell, when coming from a neighboring cell and aiming at some configuration inside C . We are compelled to valuate cells themselves and not transitions, because of memory limitations. The path to be executed follows the list of cells which maximizes the product of corresponding probabilities.

To keep the cost of such a model reasonable, we do not build an explicit representation of the whole graph. Motions of the arms are indeed independent outside the area of possible collision between them. This interaction region is represented by the image in Configuration Space of the intersection of the spheres that describe in Cartesian space the maximum swept volume of both manipulators.

Values of probabilities are initialized from the *octree* representations of Free Space for the arm of each manipulator, ignoring the opposite robot. Each *octree* is a structure of about 210 Kbytes, which corresponds to a discretization of configuration parameters in respectively 64, 64 and 44 intervals, for ranges of 270° (the idea being that the contribution of each discretization interval to the precision of a Cartesian displacement of the wrist be roughly equal). Due to memory limitations, the 6D graph is based on a discretization 4 times coarser. The probability of entry in a 3D cell is initialized with the ratio of its free successors at the lowest level in the *octree*. Probabilities of nodes of the interaction

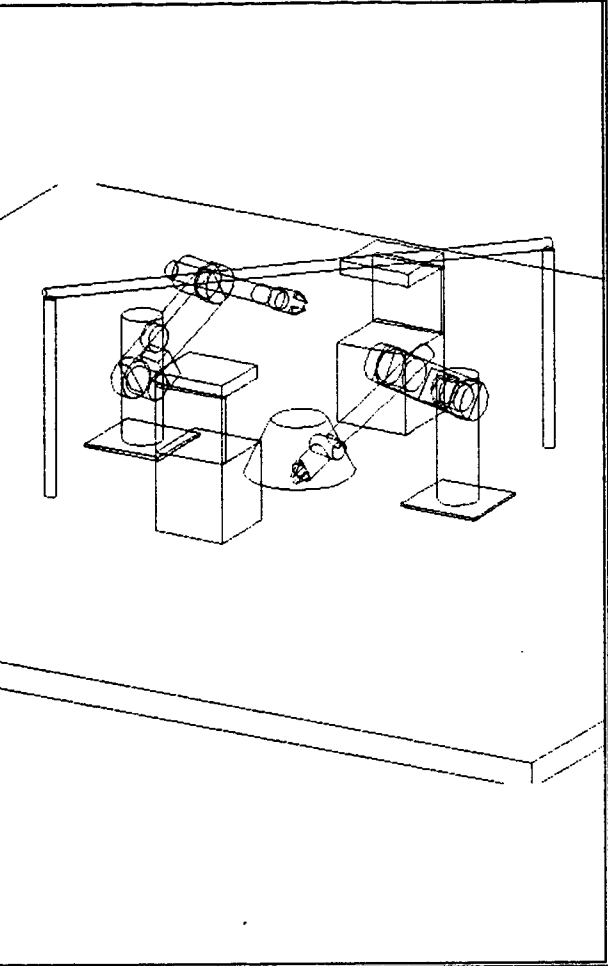
```

INITIALISATIONS PLANNER GLOBAL
nom du fichier octree du robot 1 ?r1mg
nombre d'octets :200010
nom du fichier octree du robot 2 ?r2mg
nombre d'octets :200010
robot 1
increment[ 1 ]=16.875
nb_intervalles[ 1 ]=16
nb_int_zone_interaction[ 1 ]=6
increment[ 2 ]=16.875
nb_intervalles[ 2 ]=16
nb_int_zone_interaction[ 2 ]=7
increment[ 3 ]=24.54546
nb_intervalles[ 3 ]=11
nb_int_zone_interaction[ 3 ]=9
robot 2
increment[ 1 ]=16.875
nb_intervalles[ 1 ]=16
nb_int_zone_interaction[ 1 ]=6
increment[ 2 ]=16.875
nb_intervalles[ 2 ]=16
nb_int_zone_interaction[ 2 ]=7
increment[ 3 ]=24.54546
nb_intervalles[ 3 ]=11
nb_int_zone_interaction[ 3 ]=9
initialisation du graphe d'interaction

INITIALISATIONS GENERATEUR LOCAL
calcul des couples en interaction
couple en interaction : 2 6
couple en interaction : 3 3
couple en interaction : 3 4
couple en interaction : 3 6
couple en interaction : 4 3
couple en interaction : 4 4
couple en interaction : 4 6
couple en interaction : 6 2
couple en interaction : 6 3
couple en interaction : 6 4
couple en interaction : 6 6
calcul noeuds sommets des arbres enveloppes
generation couples robot1-obstacles fixes
generation couples robot2-obstacles fixes

definition vue :
?

```



```

ddl numero 3 :-55. :
ddl numero 4 :0 :
ddl numero 5 :0 :
ddl numero 6 :0 :
robot2 commande en articulaire (0)
ou cartésien (1) ?
positions articulaires finales
ddl numero 1 :75. : -75
ddl numero 2 :30. :
ddl numero 3 :-55. :
ddl numero 4 :0 :
ddl numero 5 :0 :
ddl numero 6 :0 :
definition vue :
?quit
>calcule-trajectoire
nom du fichier ?trajec1
methode globale et apprentissage ? o
calcul global de trajectoire
188 noeuds developpes
17 noeuds sur la trajectoire
execution trajectoire
parametres par defaut(0) ou check-up(1):
blocage a t= 3 1 s
definition vue :
?

```

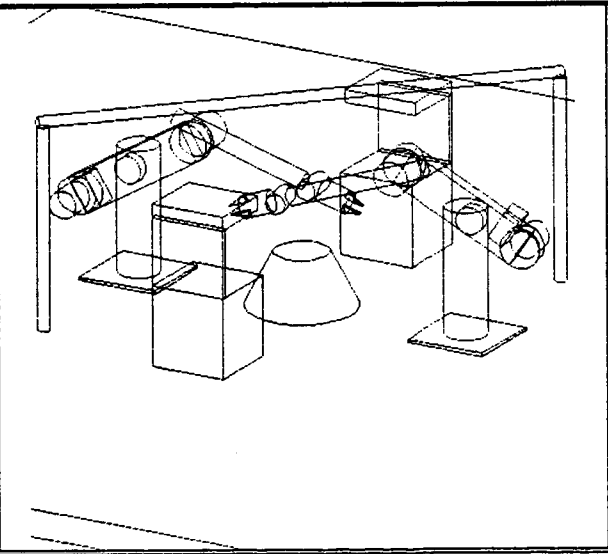


Figure 24: Dead-lock.

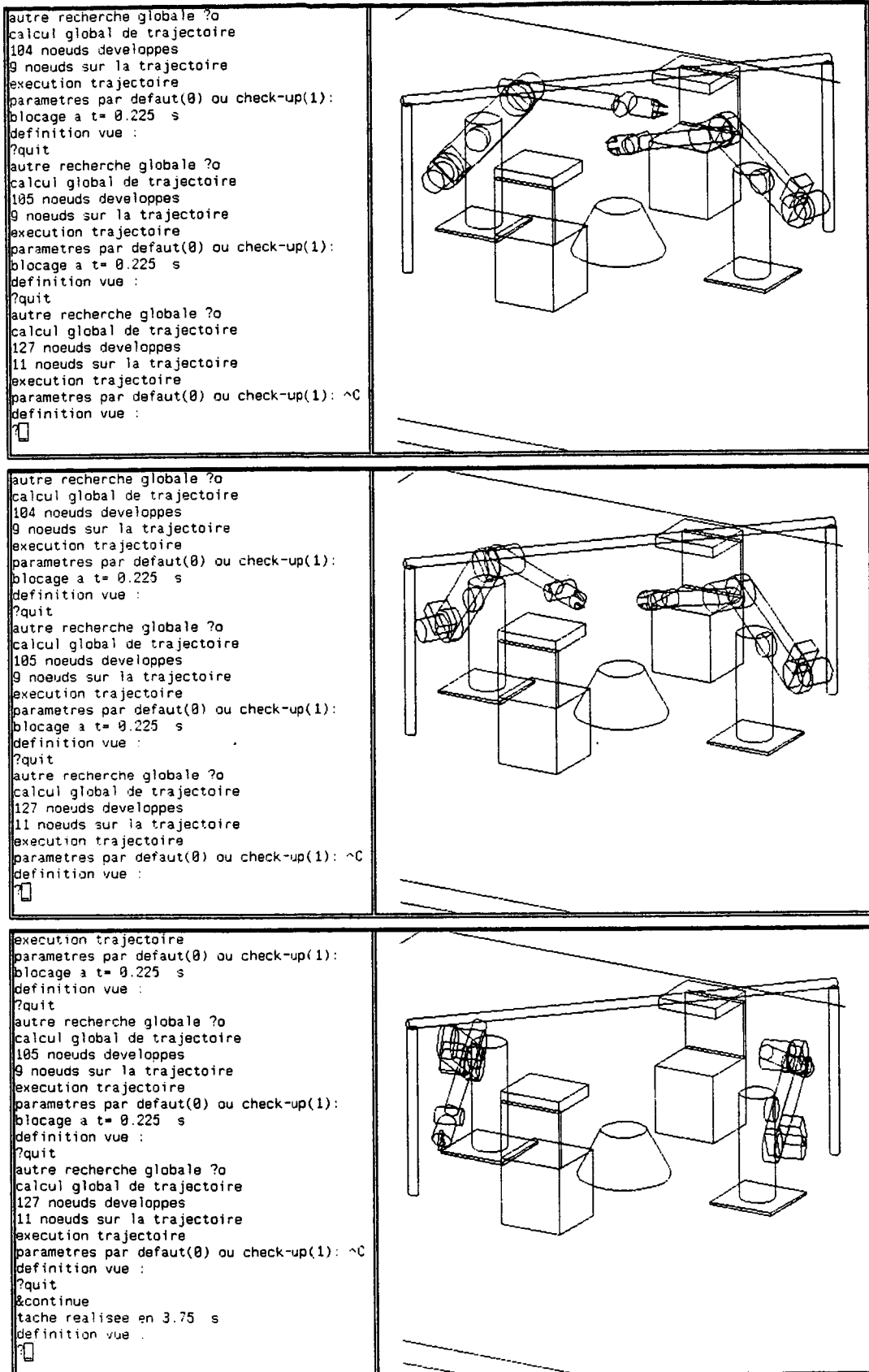


Figure 25: Learning of coordinated motions.

graph are initialized with the average of probabilities for the two 3D cells representing independent motions of the arms. Before learning, the robots thus possess a (simplified) view of fixed obstacles, but ignore possible collisions between them. The learning process will essentially consist in updating probabilities for cells in the interaction region.

The number of nodes of the interaction graph is $6^2 7^2 9^2$, which corresponds to a memory of about 570 Kbytes. The total cost of the representation is thus about 1 Mbytes.

From initial configurations of figure 24, we try to reach opposite values on the first joint of each arm. On the first attempt, dead-lock occurs at mid-distance (Fig. 24). After three modifications of the graph and calls to the global planner, we find a trajectory that necessitates retraction of joint 2 of one arm (Fig. 25). Some possibilities of movements in this part of the environment have been memorized : on a second attempt, a single call to the planner is enough to reach the goal.

5.8.2 Motion Planning for a Manipulator with a Bulky Load

To compute trajectories of a manipulator robot carrying a large object, one cannot simply consider the first three degrees of freedom to describe Free Space. We propose to build a State Graph that takes into account the whole six degrees of freedom. Again, due to memory limitations, the degrees of freedom of the hand can be only coarsely discretized. As in the above example, the State Graph is initialized from the *octree* computed for the 3 degrees of freedom of the arm. Joint ranges of the arm are divided into 16, 16 and 11 intervals, and those of the hand into 6, 4 and 6 (for ranges of 270, 150 and 270 degrees respectively). The total cost of the representation is then about 1.6 Mbytes. The State Graph is updated from results of execution of the local trajectory generator, which uses an exact geometric model of the arm, the hand and the load.

Before learning, the robot is aware of the interactions between its arm and the obstacles. The learning process essentially discriminates between different configurations of the hand, for given positions of the wrist.

Trajectories obtained are illustrated by figures 26 and 27. Computation time is of the order of a few minutes for the first execution of a trajectory.

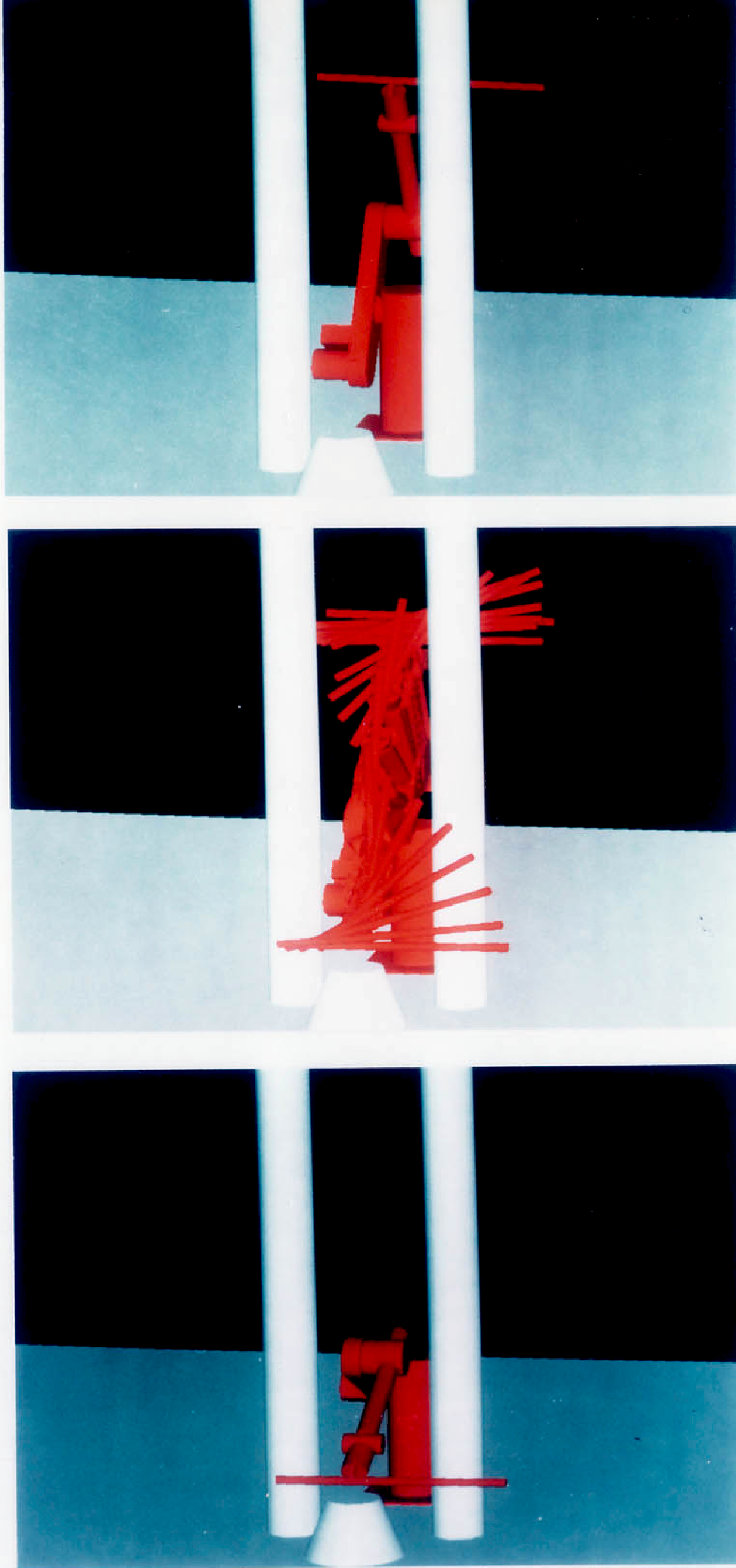


Figure 26: Moving a rod between two columns : Seven calls to the global planner were needed the first time. One is enough on a second trial.

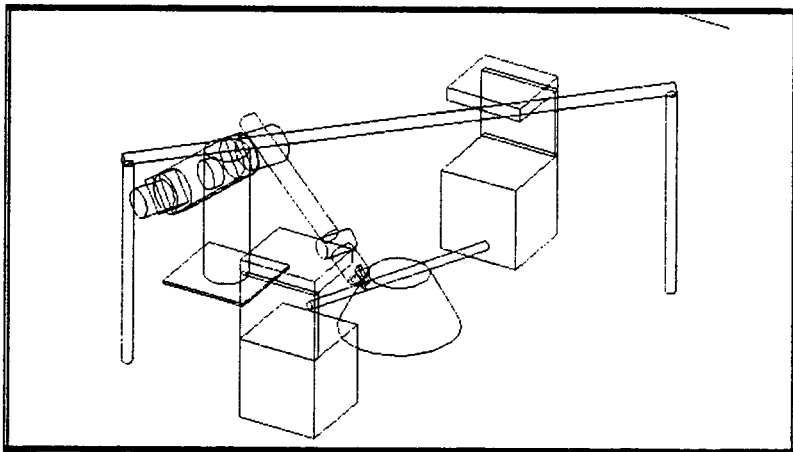
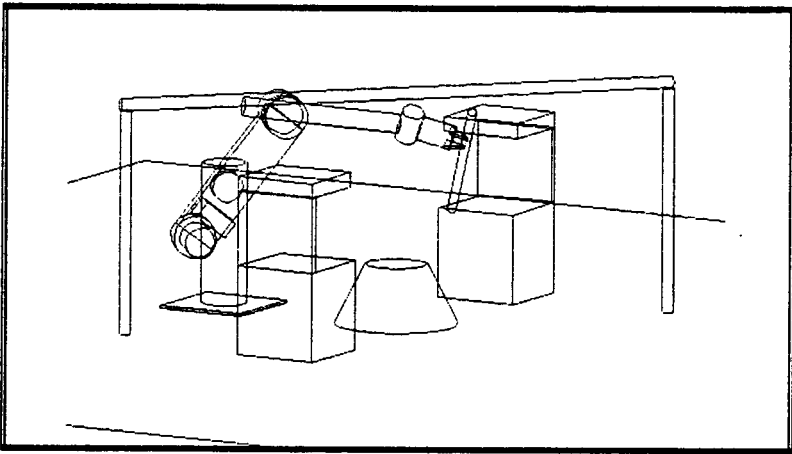
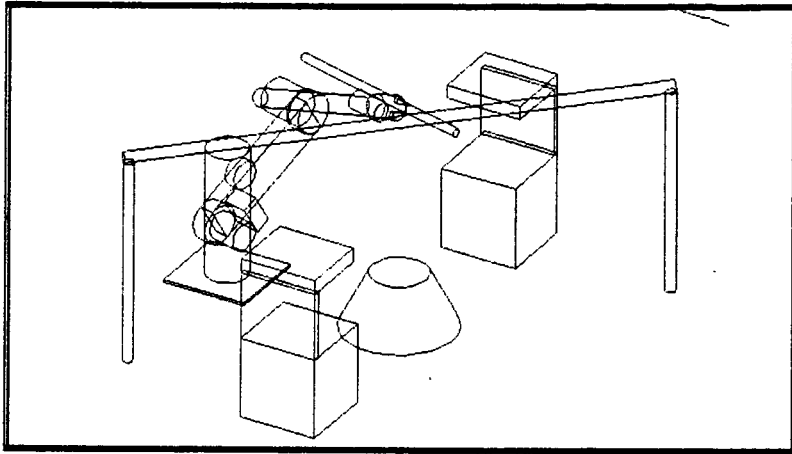


Figure 27: Initial, intermediary and goal configurations.

6 Conclusion

Difficulties we are faced with in order to solve the motion-planning problem for manipulators are twofold : 1) the practical difficulty of dealing with rotations in a three dimensional space, 2) the inherent exponential complexity of the Free Space construction with the number of degrees of freedom.

To answer the first point, we have given general tools to translate the control of critical distances between solids in Cartesian Space into constraints in Configuration Space. This results in building a *local model of Free Space*, that we use both to locally generate safe trajectories by minimization techniques, and to build global discretized models of Free Space.

To answer the second point, we have proposed to decouple the search for the global shape of the trajectory from the fine motion computations. As illustrated by figure 28, the two levels operate with very distinct time-scales : a fast one for the computation of displacements from local information, a slow one for the learning of global strategies from execution of a path. This method can also be adapted to various applications, such as the exploration of an unknown, eventually changing, environment by a mobile robot.

In addition, we have extended the motion-planning problem to a broader class of examples, defined by the control of a set of measures on the system.

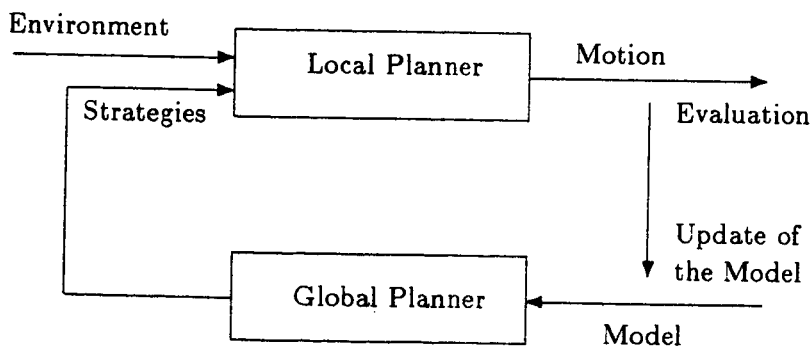


Figure 28: Two time-scales for path-planning.

References

- [Bro83] R.A. Brooks. Planning collision-free motions for pick-and-place operations. *Int. Journal of Robotics Research*, 2(4):19–44, 1983.
- [Fav86] B. Faverjon. Object level programming of industrial robots. In *Proceedings of IEEE Int. Conference on Robotics and Automation*, pages 1406–1412, San Francisco, April 1986.
- [FT86] B. Faverjon and P. Tournassoud. A hierarchical CAD system for multi-robot coordination. In *Proceedings of the NATO Advanced Research Workshop on Languages for Sensor Based Control in Robotics*, Pisa, Italy, September 1986.
- [Kha86] O. Khatib. Real time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–99, Spring 1986.
- [Loz87] T. Lozano-Pérez. A simple motion planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, 3(3):224–238, June 1987.
- [Sam84] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.
- [SS83] J.T. Schwartz and M. Sharir. On the piano movers' problem II: general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, (4):298–351, 1983.

