Two manipulation planning algorithms

Rachid Alami, Jean-Paul Laumond, Thierry Simeon

▶ To cite this version:

Rachid Alami, Jean-Paul Laumond, Thierry Simeon. Two manipulation planning algorithms. Ken Goldberg, Dan Halperin, Jean-Claude Latombe, Randall Wilson. WAFR Proceedings of the workshop on Algorithmic foundations of robotics , A. K. Peters, Ltd. Natick, MA, USA, pp.109-125, 1994, ISBN:1-56881-045-8. hal-01310030

HAL Id: hal-01310030
https://hal.science/hal-01310030

Submitted on 2 May 2016

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.

TWO MANIPULATION PLANNING ALGORITHMS

R. ALAMI, J.P. LAUMOND, T. SIMEON

LAAS REPORT N° 94239

February 1994

LIMITED DISTRIBUTION NOTICE
This report has been submitted for publication outside of CNRS. It has been issued as a Research Report for early peer distribution

The First Workshop on the Algorithmic Foundations of Robotics
A. K. Peters Pub., Boston, MA, 1994.

Two manipulation planning algorithms

R. Alami, J.P. Laumond, and T Siméon
LAAS / CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse, France

This paper addresses the motion planning problem for a robot in presence of movable objects. Motion planning in this context appears as a constrained instance of the coordinated motion planning problem for multiple movable bodies.

Indeed, a solution path (in the configuration space of the robot and all movable objects) is a sequence of transit-paths, where the robot moves alone, and transfer-paths where a movable object "follows" the robot. A major problem is to find the set of configurations where the robot has to "grasp" or "release" objects.

Based on [1, 5], the paper gives an overview of a general approach which consists in building a manipulation graph *whose connected components characterize the existence of solutions. Two planners developed at LAAS/CNRS illustrate how the general formulation can be instantiated in specific cases.*

1 The manipulation planning problem

Robot motion planning usually consists in planning collision-free paths for robots moving amidst fixed obstacles. Nevertheless a robot may have to perform tasks which are more difficult than planning motions only for itself. In some situations, a robot may be able to move objects and to change the structure of its environment. In such a context, the robot moves amidst obstacles but also movable objects. A movable object cannot move by itself; it can move only if it is grasped by the robot. According to the standard terminology, considering movable objects appears as a constrained instance of the *coordinated motion planning problem*, that we call the *manipulation planning problem* [1].

As stated in [1] (see also Latombe's book [10]), a general geometric formulation of the problem can be defined as follows.

1.1 Manipulation task and configuration space(s)

The environment is a 3D (resp. 2D) workspace which consists of three types of bodies: (1) static obstacles, (2) movable objects and (3) a robot.

For the robot and for each object we consider its associated configuration space. Object configuration spaces are 6D (resp. 3D); the robot configuration space is n-dimensional, where n is its number of degrees of freedom. Let CS denote the cartesian product of all objects and robot configuration spaces.

In the following, we will say that c is an incompletely specified configuration when some parameters in c are left unspecified; besides, we will say that a configuration c' "verifies" c when it is included in the subspace defined by c. Such a terminology will be used in order to denote partially specified goals.

Furthermore, we introduce a function Free which gives, for each domain of CS, the set of its free configurations (i.e. configurations where the bodies do not overlap).

A manipulation task is clearly a particular path in Free(CS). The converse does not hold: all paths in Free(CS) do not necessarily correspond to a manipulation task. Indeed a manipulation path is a constrained path in Free(CS). We have now to define geometrically these constraints. There are two types of constraints:

- constraints on the placements of objects; these constraints model the physics of the manipulation context (any object must be in a stable position in the environment),

---

[1] Note that this problem is related to Pick&Place and re-Grasping tasks and not to the dextrous manipulation of an object by a multi-fingered robot hand.

- constraints on object motions; any object motion is a motion induced by a robot motion.

1.2 Placement constraints

All configurations in Free(CS) do not necessarily correspond to a physically valid environment configuration. For example an object can not "levitate", and must be in a stable position. Geometrically speaking, we have to reduce the space of free configurations to a subspace which contains all valid configurations. These constraints concern only the objects. For example, if we constrain a polyhedron to be placed only on top of horizontal faces of polyhedral obstacles or of other objects (which are already in a stable position); its placement constraints will then define a finite number of 3-dimensional manifolds in its configuration space,

We call $PLACEMENT$ the subspace of Free(CS) containing all valid placements for all objects, i.e. placements which respect the physical constraints of the manipulation context. With this definition, all the objects have a fixed and known geometrical relations with the obstacles or with other objects.

$PLACEMENT$ is not more precisely defined; the definition depends on the context, and appears clearly for each context. For a mobile robot in a 2-dimensional euclidean space, amidst movable objects, $PLACEMENT =$ Free(CS).

For the planner presented in Section 2, we assume that each object has a finite number of placements in the environment; then $PLACEMENT$ appears as a finite union of n-dimensional manifolds, where n is the number of degrees of freedom of the robot.

For the planner presented in Section 3, the movable object can be placed anywhere in the environment.

1.3 Motion constraints

We define a grasp mapping $G_O^\top$ as a mapping from the configuration space of the robot (noted CSR) into the configuration space of a given object O (noted CSO), which verifies $G_O^\top(cr) = co$, where $cr \in CSR$ and $co \in CSO$. This mapping models the geometrical relation which is defined by a grasping operation ($\top$ denotes a homogeneous transform between the robot gripper frame and the object reference frame). Such mappings define geometrically the semantics of grasping for a particular manipulation context. They can be in finite or infinite number, and can be given explicitly (as in Section 2) or implicitly (they are defined for example by a contact relation between the robot or the object as in Section 3).

1.4 Problem statement

Definition A *transfer-path* is a path in Free(CS) such that there is one object O and one grasp mapping $G_O^\top$ verifying:

- the configuration parameters of any object $O' \neq O$ are constant along the path
- for any configuration of the path, $G_O^\top(cr) = co$, where cr and co designate respectively the configuration parameters of the robot and O.

Two configurations of Free(CS) connected by a transfer-path are said to be *g-connected.*

We call $GRASP$ the subspace of Free(CS) containing the configurations which are *g-connected* with a configuration of $PLACEMENT$.

Definition: A *transit-path* is a path in Free(CS) such that the configuration parameters of the objects are constant along the path. Two configurations in Free(CS) connected by a transit-path are said to be *t-connected.*

Remark: a transit-path is included in $PLACEMENT$ (but every path in $PLACEMENT$ is not necessary a transit-path).

We are now in position for defining any manipulation task as a manipulation path in Free(CS):

Definition: A *manipulation-path* is a path in Free(CS) which is a finite sequence of transit-paths and transfer-paths. Two configurations in Free(CS) connected by a manipulation-path are said to be *m-connected.*

A manipulation planning problem can then be defined as:

> **Manipulation planning problem:** An initial configuration i and a final (completely or incompletely specified) configuration f being given, does there exists a configuration verifying f which is *m-connected* with i ? If the answer is yes, give a *manipulation path* between i and some configuration verifying f.

1.5 Manipulation graph

The previous definitions lead to a property which models the structure of the solution space:

Lemma: A transit-path and a transfer-path are connected iff both have a common extremity in $PLACEMENT \cap GRASP$.

The manipulation planning problem then appears as a constrained path finding problem inside the various connected components of $PLACEMENT \cap GRASP$ and between them.

In the case of a discrete number of placements and grasps, $PLACEMENT \cap GRASP$ consists of a finite set of configurations.

When the environment contains only one movable object, even if there is an infinite number of placements and grasps, we can prove that two configurations which are in a same connected component of $GRASP \cap PLACEMENT$ are *m-connected* (see Appendix). This property leads to reduce the problem.

In both cases, it is sufficient to study the connectivity of the various connected components of $GRASP \cap PLACEMENT$ by transit-paths and transfer-paths.

We then define a graph whose nodes are the connected components of $GRASP \cap PLACEMENT$. There are two types of edges. A *transit* (resp. *transfer*) edge between two nodes indicates that there exists a transit-path (resp. transfer-path) path linking two configurations of the associated connected components.

This graph is called a *manipulation graph (MG)*. It verifies the fundamental property:

Property: An initial configuration i and a goal (completely or incompletely specified) configuration g being given, there exists a configuration f verifying g and m-connected with i iff:

- there exist a node N_i in MG and a configuration c_i in the associated connected component of $GRASP \cap PLACEMENT$, such that i and c_i are t-connected or g-connected;
- there exist a node N_f in MG and a configuration c_f in the associated connected component of $GRASP \cap PLACEMENT$ such that:
  - c_f and f are t-connected or g-connected;
  - N_i and N_f are in the same connected component of MG

In order to use this method for particular instances of the problem, one needs:

1. to compute the connected components of $GRASP \cap PLACEMENT$;
2. to determine the connectivity of these connected components using transit-paths and transfer-paths;
3. and to provide a method for planning a path in a given connected component of $GRASP \cap PLACEMENT$.

We present, in the sequel, two manipulation planners working respectively when $PLACEMENT \cap GRASP$ is reduced to a finite set of points (Section 2) and when the environment contains only one movable object (Section 3).

2 The case of discrete placements and grasps for several movable objects

In this section, we present a description of a manipulation task planner for the case of discrete placements and grasps for objects[2]. It is directly derived from the general scheme above. It is based on the fact that the connected components of $GRASP \cap PLACEMENT$ are given *a priori* by some discretization and that the construction of transit-paths and transfer-paths can be obtained using a collision-free path planner for a robot amidst stationary obstacles.

It leads to an effective construction of the manipulation graph.

For simplicity reasons, we give a presentation considering only two objects. The extension to a finite number of objects is straightforward. The presentation will be illustrated using the example of Figure 1, i.e. a 2D world where all bodies are polygonal and where the robot is allowed to move only in translation. However, the solution we propose is general.

2.1 Notations

We designate the robot by R and the objects by A and B. Let cr, ca and cb be the configuration pa-

---

[2] This planner has been first introduced in [1]. In this current presentation we have added new experimental results.

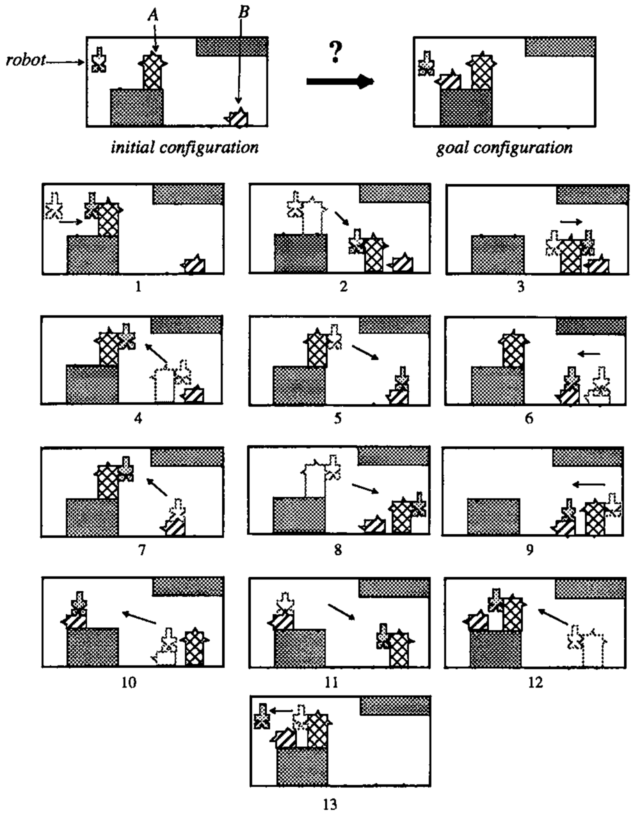


Figure 1: *A manipulation task generated automatically*

rameter vectors and CSR, CSA and CSB the associated configuration spaces. Let n be the dimension of CSR. The configuration space of all movable bodies is: $CS = CSR \times CSA \times CSB$.

Object placements: We assume that each object has a finite number of possible placements in the environment that will physically correspond to stable positions when the robot does not hold the object.

We designate by $p_A^1, \dots p_A^i \dots \in CSA$ and by $p_B^1, \dots p_B^j \dots \in CSB$ the authorized placements for A and B respectively (Figure 2).

Remark: we may also give explicitly - or give means to compute - all authorized placements combinations $(p_A^i, p_B^i) \in CSA \times CSB$, in order to take into account, for example, the possibility of stacking an object on another object.

Object grasps: We assume that each object has a finite number of possible grasps. A given grasp for object A is specified by providing a mapping G_A^i : $CSR \longrightarrow CSA$.

Let G_A^1, G_A^2 G_A^3 and G_B^1, G_B^2 be the authorized grasps for A and B (Figure 2).

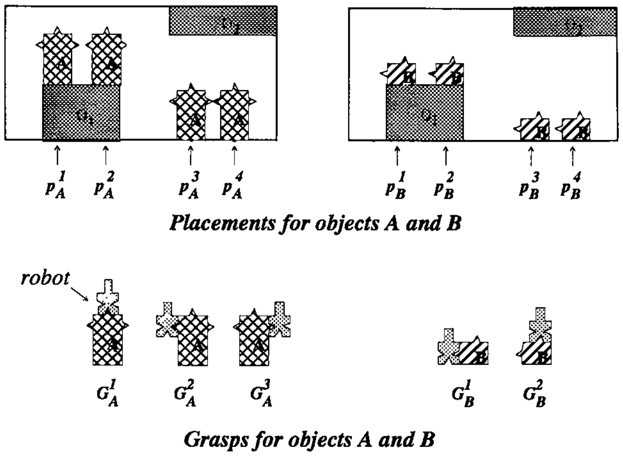


Figure 2: *Placements and grasps for objects* A *and* B

2.2 Building the manipulation graph

2.2.1 Building the nodes

Intuitively, $GRASP \cap PLACEMENT$ is simply the set of all configurations where the robot is authorized to grasp or un-grasp an object in a given placement, taken into account all possible placement combinations for the other objects.

Let I(*grasp* ; *placementA* ; *placementB*) designate the set of all free configurations where the robot can grasp one object using *grasp* while objects are placed in *placementA* and *placementB*.

Thus, $GRASP \cap PLACEMENT$ is the union of all $I(G_A^i ; p_A^j ; p_B^k)$ and $I(G_B^i ; p_A^j ; p_B^k)$.

By definition, $I(G_A^i ; p_A^j ; p_B^k)$ equals

$$\mathrm{Free}(\{cr \in CSR \mid G_A^i(cr) = p_A^j\} \times \{p_A^j\} \times \{p_B^k\})$$

The computation of $\{cr \in CSR \mid G_A^i(cr) = p_A^i\}$ is done using the robot inverse geometric model. If we assume that the robot is not redundant and that the solution does not include a robot singular configuration, then $I(G_A^i ; p_A^j ; p_B^k)$ consists of a finite (and small) number of configurations.

Each node of MG corresponds to a configuration which can be computed easily; it represents a connected component of $GRASP \cap PLACEMENT$ reduced to a single configuration.

2.2.2 Building the edges

The second step in building MG consists of establishing edges between nodes. Two nodes N_1 and N_2 are linked by a transit (resp. transfer) edge if there exists a transit-path (resp. transfer-path) between two configurations of their associated connected components.

All paths involving a node have an extremity in common: the single configuration represented by the node. For a node in $\mathrm{I}(\mathrm{G}^i_A \; ; \; p^j_A \; ; \; p^k_B)$, all transit-paths will correspond to paths where the robot moves alone while A and B are in p^j_A and in p^k_B, and all transfer-paths will correspond to paths where the robot holds A using grasp G^i_A while B is in p^k_B. This is why we define the concept of *task state* which denotes the fact that the robot is in a situation where it moves alone, or it is holding a given object.

Task states and CS slices: We define a *task state* by giving for each object its current placement and, for at most one object, its current grasp: $(grasp \; ; \; placementA \; ; \; placementB)$.

When no object is grasped, *grasp* will be noted "_", for example $(\_ \; ; \; p^1_A \; ; \; p^2_B)$. We call such a state a *transit state*.

When object X is held by the robot, *placementX* will be noted "_", for example $(\mathrm{G}^2_A \; ; \; \_ \; ; \; p^4_B)$. We call such a state an *transfer state*.

Let $\mathrm{C}(grasp \; ; \; placementA \; ; \; placementB)$ denote the set of configurations in CS associated to a given task state. $\mathrm{C}(\_ \; ; \; p^i_A \; ; \; p^j_B)$ and $\mathrm{C}(\mathrm{G}^i_A \; ; \; \_ \; ; \; p^j_B)$ correspond to n-dimensional "slices" of CS where n is the dimension of CSR.

For a transit state:

$$\mathrm{C}(\_ \; ; \; p^i_A \; ; \; p^j_B) = \mathrm{Free}(CSR \times \{p^i_A\} \times \{p^j_B\})$$

i.e. all configurations such that the robot does not overlap object A in placement p^i_A nor object B in placement p^j_B nor the obstacles.

For a transfer state:

$$\mathrm{C}(\mathrm{G}^i_A \; ; \; \_ \; ; \; p^j_B) = \mathrm{Free}(CSR \times \mathrm{G}^i_A(CSR) \times \{p^j_B\})$$

i.e. all configurations such that the robot holding object A in grasp G^i_A does not overlap object B in placement p^j_B nor the obstacles.

Remark: When $\mathrm{C}(\mathrm{G}^i_A \; ; \; \_ \; ; \; p^j_B) = \emptyset$ the corresponding state is invalid.

A node models a transition between a transit state and a transfer state. We say that the node "belongs" to these states. For example, a node in $\mathrm{I}(\mathrm{G}^i_A \; ; \; p^j_A \; ; \; p^k_B)$ "belongs" to the transit state $(\_ \; ; \; p^j_A \; ; \; p^k_B)$ and to the transfer state $(\mathrm{G}^i_A \; ; \; \_ \; ; \; p^k_B)$.

Transit edges: A transit edge can be built between a node N and any node N' which belongs to the same transit state and which is "directly" reachable. This simply means that N and N' represent configurations that are in a same connected component of $\mathrm{C}(\_ \; ; \; p^i_A \; ; \; p^j_B)$.

Transfer edges: A transfer edge can be built between a node N and any node N' that belongs to the same transfer state and that is "directly" reachable. This simply means that N and N' represent configurations that are in a same connected component of $\mathrm{C}(\mathrm{G}^i_A \; ; \; \_ \; ; \; p^j_B)$ or $\mathrm{C}(\mathrm{G}^i_B \; ; \; p^j_A \; ; \_ \,)$

We have then to construct two CS slices for any given node. However, a given CS will be used for a great number of nodes.

Figure 3 represents several nodes in the manipulation graph that corresponds to the example. The drawing at the center of the figure represents the node $\mathrm{I}(\mathrm{G}^2_B \; ; \; p^3_A \; ; \; p^2_B)$. Transit and transfer edges are built using $\mathrm{C}(\_ \; ; \; p^3_A \; ; \; p^2_B)$ and $\mathrm{C}(\mathrm{G}^1_B \; ; \; \_ \; ; \; p^2_B)$.

Figure 4 illustrates the "links" between several configuration space slices that are traversed by the system when it executes the sequence represented in Figure 1. Several states are represented; for each state, the regions in white represent the projection of the connected components of its CS slice onto the robot configuration space. In the initial state of Figure 1, the robot is in the "left" connected component of $\mathrm{C}(\_ \; ; \; p^2_A \; ; \; p^4_B)$. The only possible transition (arc 11) is to move the robot until it is able to grasp object A in G^2_A. The transitions sequence is 11-10-7-9-6-5...Note that the solution involves state $(\_ \; ; \; p^2_A \; ; \; p^4_B)$ twice, but it traverses only once a given connected component of $\mathrm{C}(\_ \; ; \; p^2_A \; ; \; p^4_B)$.

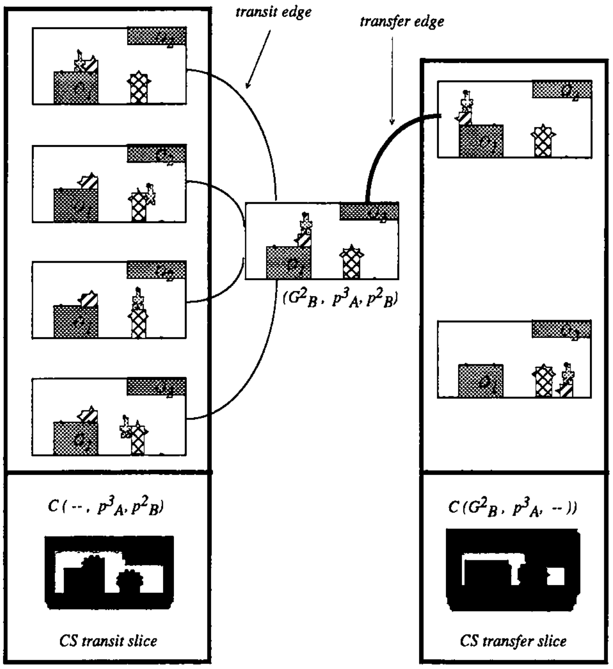

transit edge
transfer edge
(G^2_B, p^3_A, p^2_B)
$C(--, p^3_A, p^2_B)$
$C(G^2_B, p^3_A, --)$
CS transit slice
CS transfer slice

Figure 3: *A partial representation of a manipulation graph*

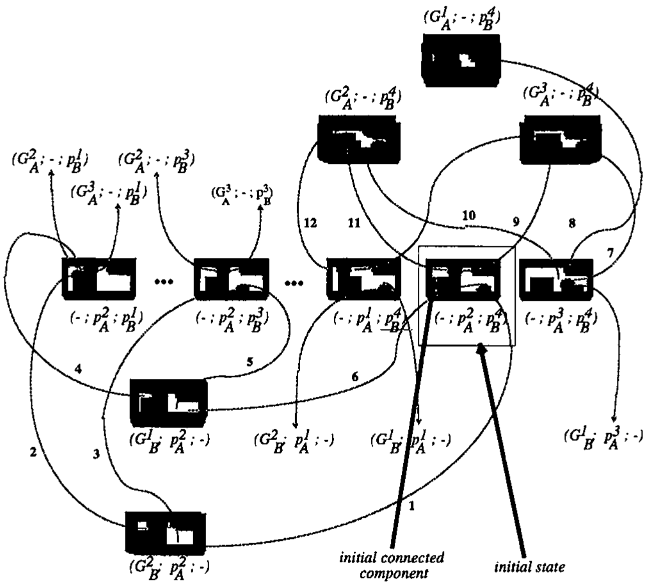

$(G^1_A; -; p^4_B)$
$(G^2_A; -; p^4_B)$
$(G^3_A; -; p^4_B)$
$(G^2_A; -; p^1_B)$
$(G^2_A; -; p^3_B)$
$(G^3_A; -; p^1_B)$
$(G^3_A; -; p^3_B)$
12
11
10
9
8
7
$(-; p^2_A; p^1_B)$
$(-; p^2_A; p^3_B)$
$(-; p^1_A; p^4_B)$
$(-; p^2_A; p^4_B)$
$(-; p^3_A; p^4_B)$
4
5
6
$(G^1_B; p^2_A; -)$
$(G^2_B; p^1_A; -)$
$(G^1_B; p^1_A; -)$
$(G^1_B; p^3_A; -)$
2
3
1
$(G^2_B; p^2_A; -)$
initial connected component
initial state

Figure 4: *Links between Configuration Space slices*

2.3 Search Strategies in the Manipulation Graph

The size of the graph grows rapidly depending on the number of grasps and placements. The number of nodes corresponds to *number of grasps* × *number of placements* (in our simple example, there are $(3+2) \times (4 \times 4) = 80$ nodes). The number of transit slices is equal to the number of legal placements ($(4 \times 4) - 4 = 12$ in the example). The number of transfer slices is equal to the number of combinations of grasps for an object and placements for the other objects ($3 \times 4 + 2 \times 4 = 20$ in the example).

The cost of building an edge is expensive and depends mainly on the cost of computing a n-dimensional *CS* slice (where n is the number of degrees of freedom of the robot). However, a *CS* slice is used several times; for example $C(\_ ; p^j_A ; p^k_B)$ will be used for all nodes in $I(G^i_A ; p^j_A ; p^k_B)$ and in $I(G^i_B ; p^j_A ; p^k_B)$. The first time, it has to be computed; and then, it will only be used in order to find a path.

MG has not to be built completely before execution. It can be explored and built incrementally. Powerful heuristics remain to be explored in order to "drive" the system towards the goal. However, even simple heuristics based only on the distance between the positions of objects allow to limit substantially the construction of the graph.

Note that, if several objects have the same shape and the same grasps and placements, the number of different *CS* slices to build can be considerably reduced. In the case, similar to the example, of two identical objects with 3 different grasps and 4 placements, we have only 12 transfer slices and 6 transit slices. Then, another way to limit the complexity, when exploring the graph edges, is to consider only a gross approximation of objects shape (by classifying them into a limited number of classes: small, elongated, big...) in order to use a same *CS* slice for a great number of nodes.

2.4 Implementation

We have implemented a system based on the method described above. It is composed of two modules: a *Manipulation Task Planner* and a *Motion Planner.*

The *Manipulation Task Planner* builds incrementally the manipulation graph and searches solution paths in it. It makes use of the Motion Planner in order to

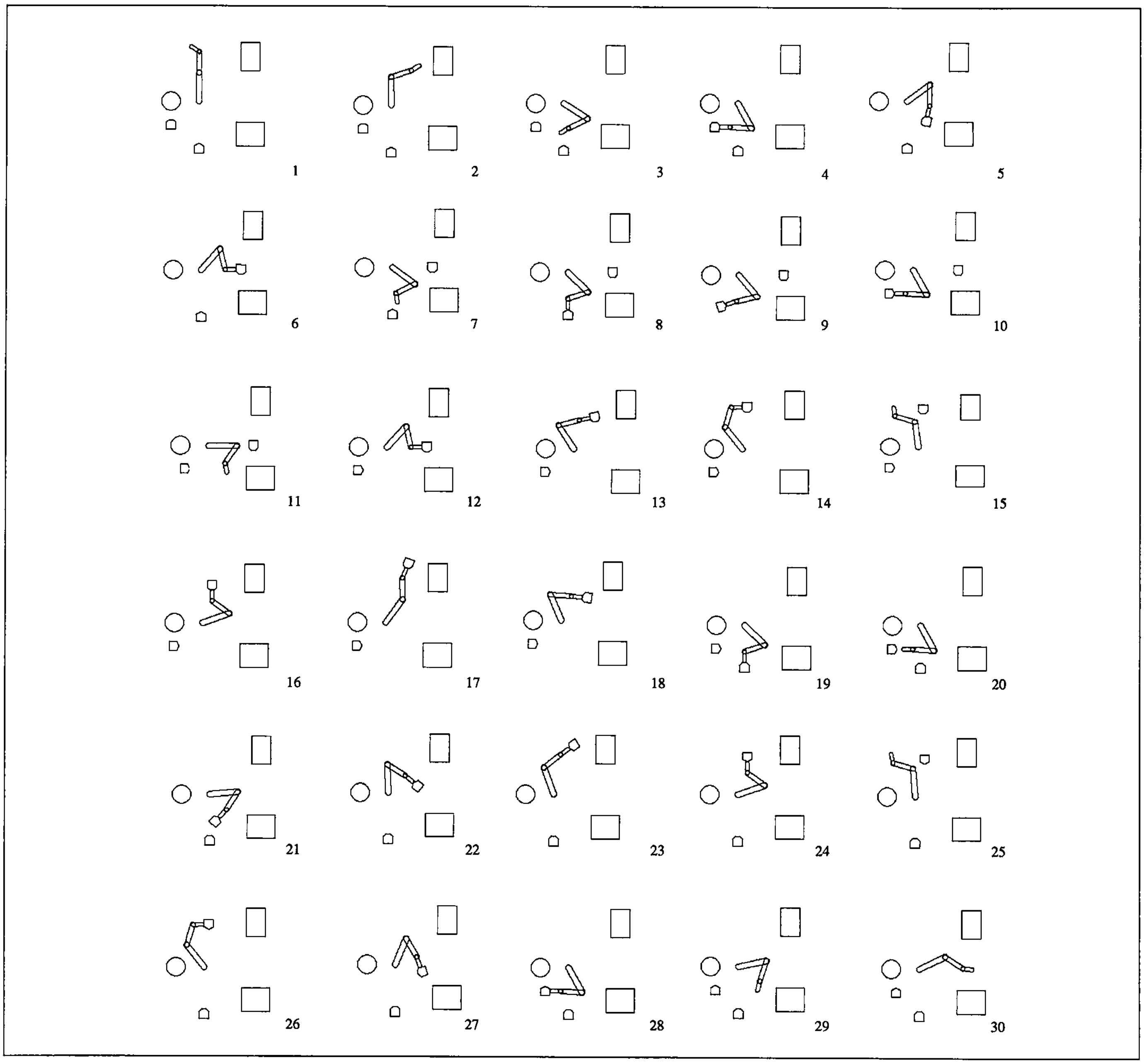


Figure 5: *A manipulation task.*

build the different *CS* slices corresponding to transfer and transit states, to structure them into connected components and to find paths between two robot configurations.

The search in the manipulation graph is performed using a A* algorithm. The cost functions currently used are based on the length of the movable bodies trajectory. The incremental graph construction allows a nice feature: for the first plans, the system is quite slow but it becomes more efficient progressively as it re-uses parts of the graph already developed.

The *Manipulation Task Planner* is implemented in order to be used for an arbitrary number of objects and does not depend on a specific motion planner module.

For the *Motion Planner*, we have first implemented a method working for a polygonal body in translation amidst polygonal obstacles (the obstacles are grown using Minkowsky sum and the trajectory is built using a visibility graph). This has been done mainly in order to demonstrate the feasibility of the approach. Figure 1 shows the plan produced for the example.

A second implementation is based on a general motion planner [17] which works for manipulators is a 3-dimensional workspace. It allows to solve manipulation planning problems as complicated as the example of Figure 5. Note that the sub-sequences 14-15-16 and 24-25-26 correspond to re-grasping operations of an object.

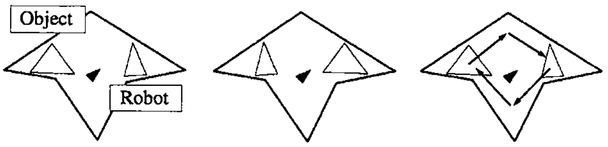
3 The polygonal case for one movable object with an infinite set of grasps

This section describes a method for solving the manipulation problem in the case of a polygonal robot and a polygonal object moving in translation amidst polygonal obstacles. It has been implemented in the case where both polygons are convex. In order to illustrate the different steps, we will rely on the simple example of Figure 6.

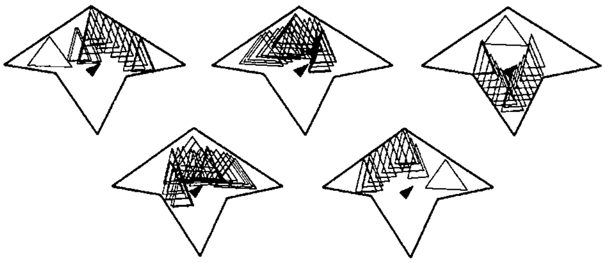
Let us consider $CS = CSR \times CSO$ the configuration space of the robot and the object together. In order to simplify the notations, we denote by :

- ACS the admissible (i.e. without any collision between the bodies) configurations space ($ACS = \mathsf{Free}(CS)$),
- $ACSR(co)$ the admissible configuration space of the robot when the movable object is placed at $co \in CSO$, and
- $ACSO(cr)$ the admissible configuration space of the movable object when the robot lies at $cr \in CSR$.

We assume that the robot has to avoid any contact with the obstacles (hypothesis H, see Appendix). We define $GRASP$ as the subset of all the configurations verifying hypothesis H and such that the robot touches the object. Finally, the movable object may be placed anywhere in the environment, as long as the bodies do not collide. As a consequence, $GRASP \cap PLACEMENT = GRASP$.



A : Initial and goal configurations. B : Coordinated motions.



C : The five stages manipulation path.

Figure 6: *An illustration of manipulation constraints.*

Let $\mathcal{C}$ be the set of connected components of $GRASP$. Let us consider the graph whose nodes are the elements of $\mathcal{C}$ and whose edges correspond to the existence of a transit-path between two configurations of the associated nodes. Thanks to the reduction property (see Appendix), two configurations in $GRASP$ are connected by a manipulation path if and only if they belong to two elements of $\mathcal{C}$ which are in the same connected component of the graph.

Therefore, according to the resolution scheme stated in Section 1, a manipulation graph can be built by :

1. computing the connected components of $GRASP$,
2. and linking them by transit-paths.

In a first step, we compute a cell decomposition of ACS (which solves the coordinated motion problem); then a retraction on the boundary of ACS gives a cell decomposition of $GRASP$. Finally, the connectivity by transit-paths between the various connected components of $GRASP$ is given by a study of the connectivity of $ACSR$, whose structure can be extracted from ACS cells.

3.1 *ACS* cell decomposition and coordinated motions

To compute the cell decomposition of ACS, we have chosen to use an adaptation of the projection method developed by Schwartz and Sharir in [15] for the case of two discs. Any other, and perhaps better ([7, 14, 16]) method could have been used. However, the purpose of this paper is not to give some optimal algorithm, but to demonstrate the feasibility of our approach.

Let us consider an object position co. $ACSR(co)$ is obtained by removing, from $ACSR$ (robot admissible configurations without the object), the set $COL(co)$ of all configurations where the robot and the object collide [3].

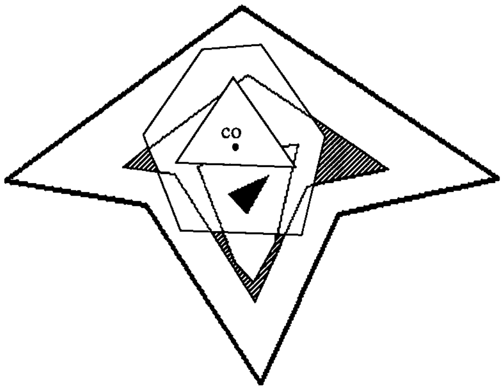


Figure 7: *The hatched areas represent the connected components of $ACSR(co)$.*

Let us observe now the evolution of $ACSR(co)$ with respect to co. In a neighborhood of most object positions, $ACSR(co)$ varies only quantitatively, keeping the same structure. However, at some object positions, some $ACSR(co)$ connected components may appear, disappear, be split or merged. More precisely, the geometrical structure of the connected components of $ACSR(co)$ are modified when some vertices appear or disappear. These changes correspond to specific values of co which constitute a set of *critical curves* (see [15] for a proof). Figure 8 gives an example of a critical curve along which a connected component of $ACSR(co)$ is divided into two separate components. The critical curves provide a decomposition of $ACSO$ into *non-critical regions.*

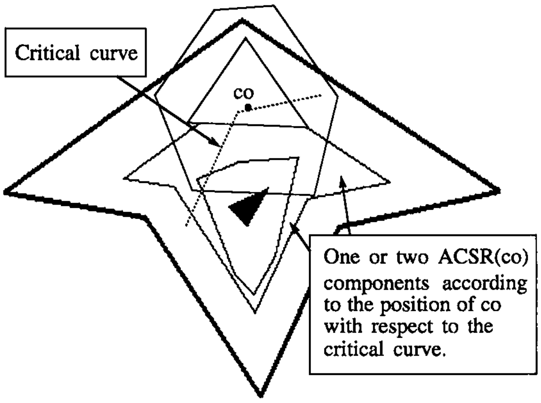

Critical curve
co
One or two ACSR(co) components according to the position of co with respect to the critical curve.

Figure 8: *A critical curve*

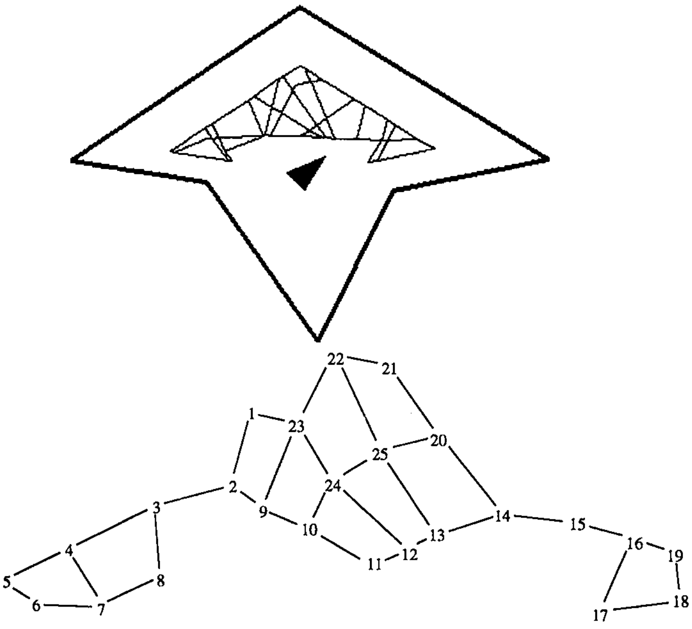


Figure 9: *Cell decomposition of $ACSO$ and the associated graph*

Let us consider a non-critical region R. $\{\{co\} \times ACSR(co) \mid co \in R\}$ constitutes cells of ACS (there are as many cells as the number of connected components of $ACSR(co)$). The set of all such cells is the expected cell decomposition.

---

[3] $COL(co)$ is the polygon obtained by Minkowski difference between the robot and the object, and placed in co.

In order to compute the critical curves, we introduce a symbolic description of $ACSR(co)$. Let us recall that the boundary of $ACSR(co)$ is constituted by $ACSR$ edges and $COL(co)$ edges. We assign a numerical label to all the vertices of $ACSR$ and a literal symbol to the edges of $COL(co)$. We denote by $b[8,9]$ the intersection between the edge b of $COL(co)$ and the segment $[8,9]$ of $ACSR$. Therefore, the bottom connected component in the example of Figure 10 is labeled by the sequence $(3,4,[4,5]b,b[8,9],9,[9,10]b,b[1,2],2)$.

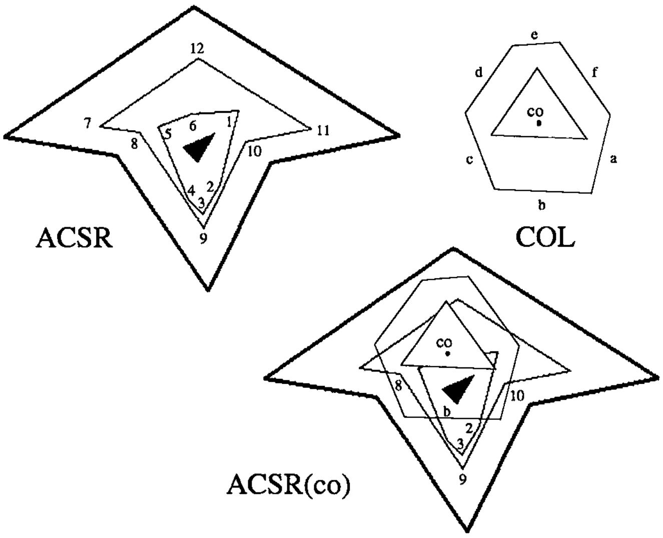


Figure 10: *An illustration of the labels used to characterize the connected components of ACSR(co).*

To compute the critical curves, we do not consider all the possible changes in this sequence. We just need to consider the changes *on the letters* (i.e. the changes induced by $COL(co)$ and not by $ACSR$ vertices). In Figure 10 for instance, when co moves to the bottom, the disappearance of vertex 2 in the sequence above does not induce a critical curve, while the fusion of the two b labels (when edge b meets vertex 3) does.

The critical curves of our example are shown in Figure 9, together with the graph of non-critical regions of $ACSO$.

Let us recall that each non-critical region induces as many cells in ACS as the number of connected components in $ACSR(co)$ when co belongs to the region. Now we structure all these cells into a *coordinated motion graph*. Two cells are adjacent in this graph if and only if :

- the associated non-critical regions are adjacent in $ACSO$,
- and the symbolic descriptions of the associated $ACSR(co)$ components just differ by a letter.

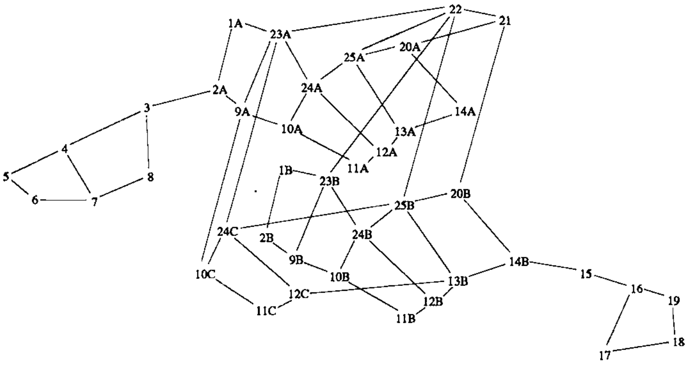


Figure 11: *The coordinated motion graph.*

Figure 11 shows the coordinated motion graph of our example. In Figure 10, co belongs to the non-critical region 10 (Figure 9). This region gives rise to three ACS cells numbered $10A$, $10B$ and $10C$ in Figure 11. $10C$ is the node corresponding to the label mentioned above.

The fundamental property is : there exists a coordinated motion between two configurations in ACS if and only if they belong to nodes of a same connected component of the coordinated motion graph. The proof is exactly the same as in [15].

3.2 *GRASP* cell decomposition and contact motion

Let us consider the above $ACSR(co)$ component labeled by $(3,4,[4,5]b,b[8,9],9,[9,10]b,b[1,2],2)$. There are two edges labeled by b in it. This means that there are two connected sets of configurations where the robot is in contact with the object. It is not possible to go from one set to the other one without leaving the contact. These connected sets are easily extractable from the symbolic description of the $ACSR(co)$ components. In our example,

$([4,5]b, b[8,9])$ and $([9,10]b, b[1,2])$ are the symbolical descriptions of the two classes of contact. By definition, they always contain two terms.

Now, let us consider a non-critical region R. The set $\{\{co\} \times COL(co) \cap ACSR(co) \mid co \in R\}$ constitutes cells of $GRASP$ (there are as many cells as the number of connected components of $COL(co)$ along the $ACSR(co)$ boundary).

We follow the same method as for the coordinated motion problem. We structure the $GRASP$ cells into a *contact graph.* Two cells are adjacent in this graph if and only if :

- the associated non-critical regions are adjacent in $ACSO$,
- and their symbolic descriptions just differ by one term.

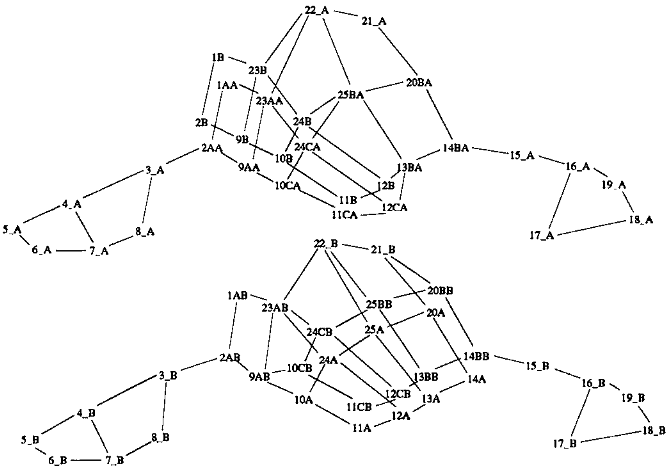


Figure 12: *The contact graph.*

Figure 12 shows the contact graph of our example. Region 10 gives rise to three ACS cells. The frontier of two of them ($10A$ and $10B$) contain one connected component in $GRASP$. They then give rise to two $GRASP$ cells (which keep the same name in the contact graph). The frontier of $10C$ contains two connected components in $GRASP$; they give rise to $GRASP$ cells $10CA$ and $10CB$.

This graph verifies the following property : there exists a motion keeping the contact between the robot and the object, between two configurations in $GRASP$, if and only if both configurations belong to nodes of a same connected component of the contact graph. The proof is exactly of the same kind as for the coordinated motion graph.

3.3 The manipulation graph

Let us come back to our manipulation planning problem. At this time we have captured the connectivity of $GRASP$. We know that two configurations in the same connected component of $GRASP$ may be linked by a manipulation path (Reduction Property).

Now we have to study the existence of transit-paths between $GRASP$ components. This study is very easy from the above labeling. Indeed let us consider $ACSR(co)$ in Figure 10. There are two grasp classes in the bottom component. Nevertheless, it is possible for the robot to move *alone* in this component; that means that the robot can go from a position in the first grasp class to any other one in the second grasp class. Then, these two classes are linked by transit-paths.

The existence of such transit-paths is very easy to compute from the labeling of $ACSR(co)$. Indeed, two $GRASP$ cells are connected by a transit-path if and only if they belong to the frontier of the same ACS cell. In our current example, only the cells $10CA$ and $10CB$ (which come from the same ACS cell $10C$) are linkable by a transit path.

Computing the connectivity of $GRASP$ components by transit-path is equivalent to adding to the contact graph edges between nodes defined from a same ACS cell. These additional edges are referred as "transit edges" in Figure 13. With our notations, two nodes in the contact graph whose "names" contain the same number and the same first letter are linked by a transit edge. The resulting graph is the *manipulation graph.*

3.4 Manipulation path finding

Let us consider an initial configuration c_i and a final one c_f, defining the initial and final positions of the robot and the object. According to the property of the manipulation graph, we use a three-steps procedure :

1. First, we compute the ACS cells C_i and C_f containing c_i and c_f. Then, we compute the set $\mathcal{G}_i$ (resp. $\mathcal{G}_f$) of $GRASP$ cells reachable from c_i (resp. c_f). This computation is a 2-dimensional problem

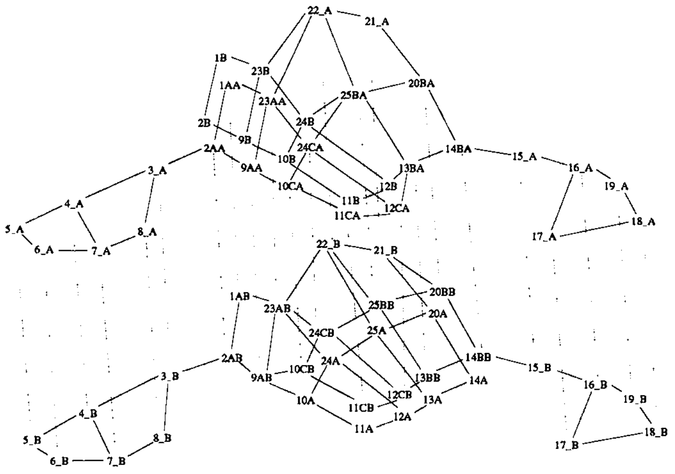


Figure 13: *The manipulation graph, where dotted lines describe transit edges.*

since the transit-paths have to lie in $ACSR(co_i)$ (resp. $ACSR(co_f)$).

2. The second step consists in searching a path in the manipulation graph between a cell in $\mathcal{G}_i$ and a cell in $\mathcal{G}_f$. If no such path exists, the procedure stops. There is no solution. Otherwise, we obtain a sequence $\mathcal{P}ath$ of $GRASP$ cells.
3. Finally the complete path is built from elementary manipulation paths lying in the $GRASP$ cells of $\mathcal{P}ath$ and from transit-paths associated to the transit edges contained in $\mathcal{P}ath$.

Comments on Step 1 : The computation of C_i (resp. C_f) is a 2-dimensional location problem performed in $ACSR(co_i)$ (resp. $ACSR(co_f)$) : we have to determine the connected component of $ACSR(co_i)$ (resp. $ACSR(co_f)$ containing cr_i (resp. cr_f). This fully characterizes C_i (resp. C_f). Then the computation of $\mathcal{G}_i$ (resp. $\mathcal{G}_f$) is very easy, since these $GRASP$ cells belong to the frontier of C_i (resp. C_f) : with our notations, a $GRASP$ cell belonging to the frontier of some ACS cell appears in the contact graph with the same numerical label and the same first letter as the ACS cell appears in the coordinated motion graph.

Comments on Step 2 : The second step is performed using a A^* algorithm. Several cost criteria can be introduced : the length of the complete path, the number of grasping changes... The experimental results will give an illustration of the influence of the cost definition.

Comments on Step 3 : Step 3 consists in computing the complete manipulation path. $\mathcal{P}ath$ is a sequence of $GRASP$ cells. Two consecutive cells in this sequence are linked either by an edge already appearing in the contact graph, or by a transit edge. Moving inside a $GRASP$ cell needs a specific procedure : we have implemented the method presented in the proof of the reduction property . Finally, there remains to compute the transit-path associated to a transit edge : this is a 2-dimensional problem solved in some $ACSR(co)$ slice by a visibility graph method for instance (which is the method we have implemented).

Remark : The algorithm can be adapted in order to take into account partial goals : in this case the goal configuration describes only a goal position for the object (resp. the robot) without specifying a goal position for the robot (resp. the object). Such an extension is easy when the robot goal is unspecified and the object goal is known (the identification of the goal node is given by the non-critical region containing the object). The case of an unspecified object goal is also tractable, but would require some tedious algorithmic details.

3.5 Complexity

In order to evaluate the complexity of the algorithm, we have to distinguish the *decision* part of the manipulation problem (i.e. proving the existence of a solution) from the *complete* problem (i.e. the computation of a solution if any).

In our algorithm, the decision problem is solved by building and searching the graph. The complexity of this part is clearly dominated by the construction of the non-critical regions. Let us denote by n_e, n_r and n_o the number of vertices of the environment, the robot and the object respectively. We assume that both object and robot are convex.

All the critical curves lie in $ACSO$ (Figure 9), whose computation can be done in $O(n_e n_o \log(n_e n_o))$; moreover, the complexity of the admissible configuration space of the object is in $\Omega(n_e n_o)$ (see for instance [6]). Similarly the complexity of the admissible robot configurations subset $ACSR$ is in $\Omega(n_e n_r)$ and can be computed in $O(n_e n_r \log(n_e n_r))$.

The critical curves are defined by the coincidence between an edge (resp. a vertex) of *ACSR* and a vertex (resp. an edge) of *COL*(*co*) boundary (i.e. the set of contacts between the object and the robot). The edge number of *COL*(*co*) is exactly $(n_o + n_r)$. Then, the number of critical curves is in $\Omega(n_e n_r(n_o + n_r))$.

The cell decomposition of *ACSO* is given by the computation of the intersections between the $\Omega(n_e n_r(n_o + n_r))$ critical curves, and the $\Omega(n_e n_o)$ edges of *ACSO* boundary. This can be done in $O((n_e n_r(n_o + n_r) + n_e n_o)^2 \log(n_e n_r(n_o + n_r) + n_e n_o))$ and gives $\Omega((n_e n_r(n_o + n_r) + n_e n_o)^2)$ non-critical regions.

Finally, in each non-critical region, the number of connected components of *ACSR*(*co*) is in $O(n_e n_r)$. They are computed by intersection of the $(n_r + n_o)$ edges of *COL*(*co*), and the $\Omega(n_e n_r)$ edges of *ACSR*, for each non-critical region, each time placing the object on some point of the region. Such an intersection is computed in $O((n_r + n_o) n_e n_r \log(n_r + n_o + n_e n_r))$. The manipulation graph is then obtained in $O((n_e n_r(n_o + n_r) + n_e n_o)^2 (n_r + n_o) n_e n_r \log(n_r + n_o + n_e n_r))$.

The number of robot and object edges may be considered to be small comparing with n_e. Then, defining n as the environment complexity, previously called n_e, our algorithm runs in $O(n^3 \log(n))$ time. We did not try to optimize it at this time. Perhaps more sophisticated cell-decomposition (like [16]) could be used in the same framework.

Finally, the complexity of the complete problem (i.e. the computation of a solution path), is dominated in the worst case by the number of elementary manipulation paths (see Appendix). Therefore the complexity of the complete problem not only depends on the complexity of the environment but also on the "clearance" of the robot in the environment.

3.6 Experimental results

Figures 14 to 17 show results obtained with the above described implementation.

The solution given by the planner to our illustrative example is shown in Figure 6C. The initial and final object positions are respectively in the non-critical regions 6 and 19 (Figure 9). The initial and final configurations of the manipulation problem are respectively in the *ACS* cells 5 and 18 (Figure 11).

Let us consider now the solution path (5_*B*, 4_*B*, 3_*B*, 2*AB*, 9*AB*, 23*AB*, 22_*B*, 22_*A*, 25*BA*, 13*BA*, 14*BA*, 15_*A*, 16_*A*, 19_*A*, 18_*A*). Figure 6Ca shows the transit-path along which the robot first reaches the object. In Figure 6Cb, the object is moved with a sequence of transit and transfer paths along the same connected component of *GRASP*. If we refer to the manipulation graph (Figure 13), the manipulation path is built, at this time, from the sequence (5_*B*, 4_*B*, 3_*B*, 2*AB*, 9*AB*, 23*AB*, 22_*B*) of *GRASP* cells. A transit edge appears between vertices 22_*B* and 22_*A*; the associated transit path is shown in Figure 6Cc. Figure 6Cd describes the subsequence (22_*A*, 25*BA*, 13*BA*, 14*BA*, 15_*A*, 16_*A*, 19_*A*, 18_*A*). A last transit-path allows to reach the robot goal configuration.

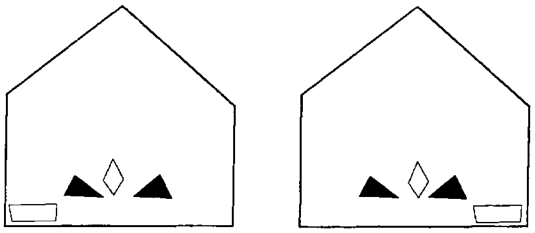


Figure 14: *Initial and goal configurations.*

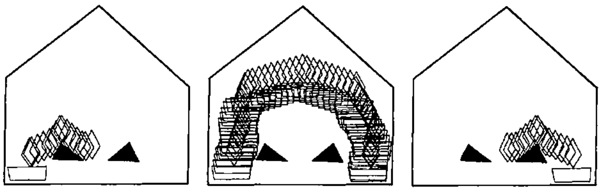


Figure 15: *Solution using a cost function that minimizes the number of grasps.*

Figures 16 to 15 illustrate the influence of heuristics used to find a path through the manipulation graph.

In Figure 16, the weights of transit and transfer links of manipulation graph are nearly equivalents, each referring to the straight line distance between reference points of the *GRASP* cells. Then the path found by the search algorithm involves numerous transit-paths and re-grasps.

On the contrary, Figure 15 describes the obtained

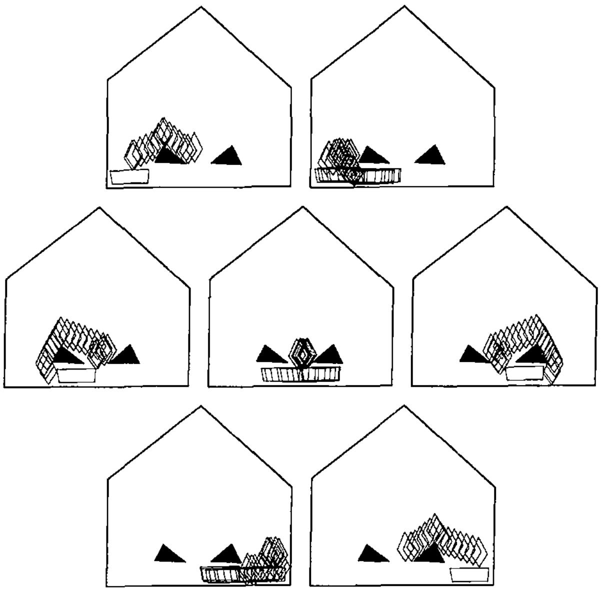


Figure 16: *Solution using a cost function that minimizes the length of the object path.*

solution when transit-paths are heavier, due to a simple multiplying coefficient. A path is then found that allows to keep the same grasp along the whole path.

Finally, Figures 17 and 19 show the solution given to a more intricate situation, involving numerous regrasps. The second and third images detail a transit and a transfer path at the beginning of a contact motion. The heuristic used there takes into account the average length of available grasp frontier for each grasp cell. Then it avoids narrow grasps which could involve numerous re-graspings.

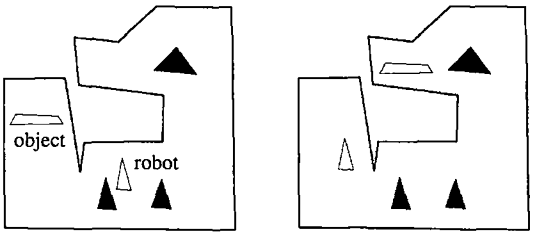


Figure 17: *A more intricate situation.*

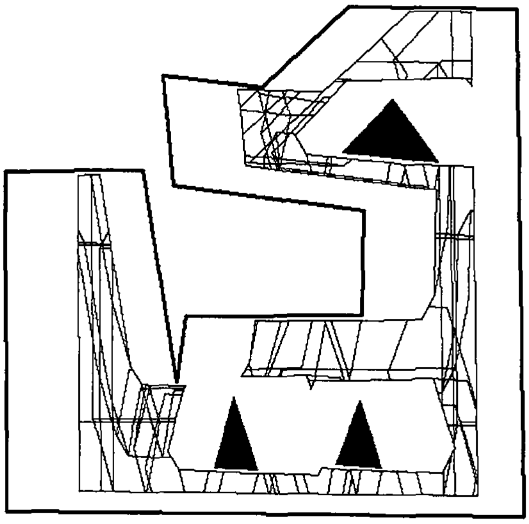


Figure 18: *Non-critical regions.*

The algorithm has been implemented in C on a Sun SPARC Station 2. The computation of the manipulation graph of Figure 13 (including the computation of the non-critical regions) takes 3.4 seconds. The search for a path takes 0.25 seconds for the example in Figure 6. In this simple case, the cell decomposition of the admissible configurations of the object gives 25 non-critical regions, and the manipulation graph consists of 69 nodes and 126 edges. In the example of Figure 14, we obtain 146 non-critical regions, 257 nodes and 475 edges. The case of Figure 17 gives 288 non-critical regions (shown in Figure 18) and its manipulation graph consists of 619 nodes and 1115 edges.

4 Related work and open problems

The first paper that attacks motion planning in presence of movable obstacles is [18]; in this paper, Wilfong gave the first results on the complexity of the problem : he proved that the problem is $PSPACE$-hard (resp. NP-hard) in two dimensional environments where only translations are allowed and when the final configuration specifies (resp. does not specify) the final positions of all the movable objects. In the same reference, Wilfong gives a solution in $O(n^3 \log^2 n)$ (where n is the number of vertices of a polygonal environment) for the case of a convex polygonal robot moving in translation

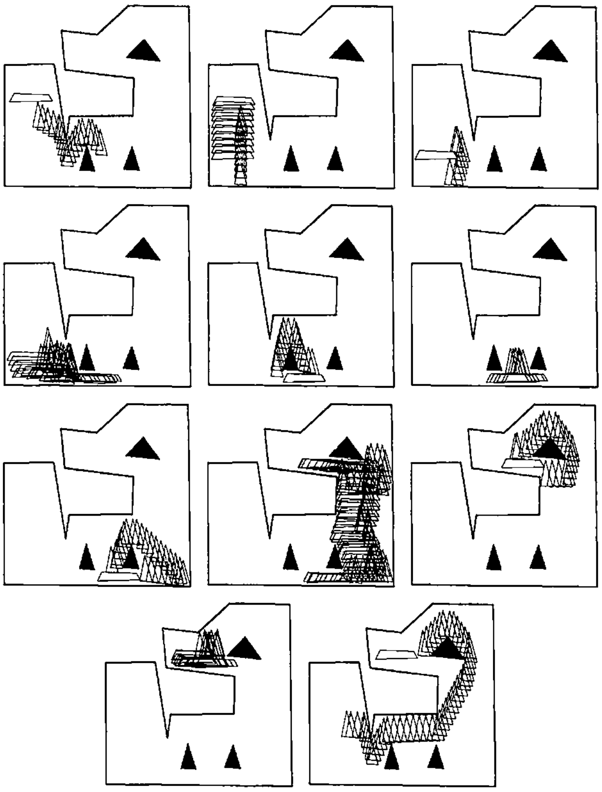


Figure 19: *A manipulation path.*

amidst one polygonal object (with a finite grasp set) and obstacles.

In [12] we presented a general algorithm for the case of one robot and one movable object. We showed [4] how to decompose the space of grasping configurations into a finite number of cells in order to make the problem tractable. However the method makes use of general tools from algebraic geometry, leading to an inefficient algorithm in practice.

More recently, several authors attacked the manipulation planning problem in various contexts.

Koga and Latombe [8, 9] addressed the case of multi-arm manipulation planning where several robots have to cooperate to carry a single movable object amidst obstacles. In this context, re-grasps of the object are often required along the path to avoid collisions and may involve changing the arms grasping the object. In [8] they propose several implemented planners to deal with problems of increasing difficulty for two identical arms in a 2D workspace. For problems involving many degrees of freedom, which is usually the case in multi-arm manipulation, they use an adapted version of the randomized potential field planner [2]. An extension to a robot system made of three 6 DOF arms in a 3D workspace is also presented in [9]. In this work, the number of legal grasps of the object is finite and the algorithms require the object to be held at least by one robot at any time during a re-grasp operation. The approach relies on several simplifications, but it yields to impressive results for complex and realistic problems.

The approach developed by Barraquand and Ferbach [3] consists in translating the manipulation planning problem into convergent series of less constrained subproblems increasingly penalizing the motions that do not satisfy the constraints. Each subproblem is solved using variational dynamic programming.

[4] describes a heuristic algorithm for a circular robot and where all the obstacles can be moved by the robot in order to find its way to its goal.

Finally, let us pinpoint the interesting extension of the manipulation planning problem attacked by Lynch and Mason [13]. In their context, grasping is replaced by pushing. The space of stable pushing directions imposes a set of nonholonomic constraints on the robot motions, which opens issues of controllability.

All the above mentioned studies contribute to establish the manipulation planning problem as a specific and challenging instance of the motion planning problem with constraints. However, several open questions remain, ranging from a theoretical analysis of the problem to the investigation of new practical instances. One key theoretical aspect concerns the conditions under which the reduction property can be extended to the case of several objects and robots. On a practical point of view, the problem represents also a challenge to motion planning techniques because of its additional complexity.

Acknowledgement : This work has been partially supported by the Esprit Programme through Basic Research Action 6546 PROMotion. We are grateful to B. Dacre-Wright who implemented the algorithm described in Section 3.

---

[4] The principles are similar to those presented in Section 3, but they are established for general robot systems.

Appendix : Reduction property

In this appendix, we consider only one movable object. In this case, two configurations belonging to a same connected component of $GRASP \cap PLACEMENT$ are m-connected. In fact, this property holds up to a precise definition of what is a grasping configuration.

Let us consider $CS = CSR \times CSO$ the configuration space of the robot and the object together. Let us denote by $CONTACT$ the domain of the configurations where the robot and the object are in contact. $CONTACT$ is a subset of $\mathsf{Free}(CS)$ boundary.

Hypothesis H**:** We assume that the robot has to avoid any contact with the obstacles.

We then define $GRASP$ as the subset of $CONTACT$ of all the configurations verifying hypothesis H.

Property 1 *Two configurations of a same connected component of* GRASP ∩ PLACEMENT *are connected by a manipulation path.*

Proof: Let a and b be two configurations in a connected component of $GRASP \cap PLACEMENT$. There exists a path $p : [0,1] \mapsto GRASP \cap PLACEMENT$ linking these two configurations[5] ($p(0) = a$, $p(1) = b$). We define p_r and p_o as the projections of p onto CSR and CSO respectively.

Let $c = p(t)$ be any configuration on the path. Thanks to the hypothesis H, $p_r(t)$ lies in an open set of $\mathsf{Free}(CSR)$. We then can find an open disc $D_\epsilon \subset \mathsf{Free}(CSR)$ centered on $p_r(t)$ and with a radius $\epsilon > 0$.

Since p is continuous, there exists $\eta_1 > 0$ such that :

$$\forall \tau \in]t - \eta_1, t + \eta_1[, p_r(\tau) \in D_{\epsilon/2}.$$

Similarly, $p_r - p_o$ is a continuous function. Then there exists $\eta_2 > 0$ such that, for any $\tau \in]t - \eta_2, t + \eta_2[$,

$$\| (p_r(\tau) - p_o(\tau)) - (p_r(t) - p_o(t)) \|_{\mathbf{R}^2} < \epsilon/2.$$

This last assertion means that the relative grasp configuration does not vary more than $\epsilon/2$ along the path p between $p(t - \eta_2)$ and $p(t + \eta_2)$.

Let us consider $\eta = min\{\eta_1, \eta_2\}$:

$$\forall \tau, \sigma \in]t-\eta, t+\eta[, \quad p_o(\sigma) + (p_r(\tau) - p_o(\tau)) \in D_\epsilon . \quad (1)$$

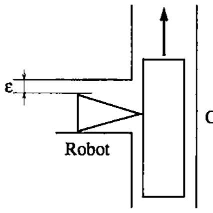


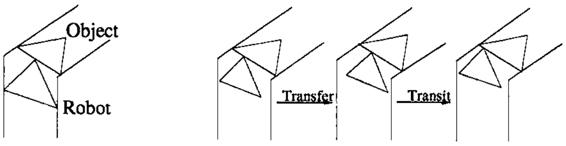
Figure 20: *A case with finite (but great) number of transit and transfer paths.*

Let $c_1 = p(\tau_1)$ and $c_2 = p(\tau_2)$ be any two configurations on p, with τ_1 and τ_2 in $]t-\eta, t+\eta[$ (we assume that $\tau_1 < \tau_2$). We prove now that c_1 and c_2 can be linked by one transfer path followed by one transit path.

Let us consider the path $(p_o(\tau), p_o(\tau) + (p_r(\tau_1) - p_o(\tau_1)))$, with $\tau \in [\tau_1, \tau_2]$. This path is clearly a transfer path with constant grasp $(p_r(\tau_1) - p_o(\tau_1))$, between $p(\tau_1)$ and $(p_o(\tau_2), p_o(\tau_2) + (p_r(\tau_1) - p_o(\tau_1)))$. According to relation 1, this path is admissible. Let us consider the path $(p_o(\tau_2), p_o(\tau_2) + (p_r(\tau) - p_o(\tau)))$, with $\tau \in [\tau_1, \tau_2]$. This path is clearly a transit path between $(p_o(\tau_2), p_o(\tau_2) + (p_r(\tau_1) - p_o(\tau_1)))$ and $p(\tau_2)$. Again, according to relation (1), this path is admissible. The concatenation of both paths constitute a manipulation between $p(\tau_1)$ and $p(\tau_2)$.

As path p_r is a compact set included in an open set of $\mathsf{Free}(CSR)$, we can apply this local transformation on a *finite* covering of $[0, 1]$. We have then a finite number of elementary manipulation paths which constitutes a manipulation path linking a and b. □

Remark : the number of elementary manipulation paths used in the proof of the reduction property depends on the clearance of p_r in $\mathsf{Free}(CSR)$. The worst case is reached in the example Figure 20 where the number of elementary paths is clearly in $O(\frac{1}{\epsilon})$.



A: Hypothesis H unsatisfied

B: Hypothesis H satisfied

Figure 21: *Illustration of hypothesis H.*

---

[5] p designates a path as well as the associated function.

Note that the reduction property does not hold when hypothesis H is not satisfied. Figure 21 illustrates this fact. Figure 21A shows an example where both the robot and the object touch the environment. There exists a coordinated path, but no feasible manipulation path (the robot cannot move the object with a constant grasp). Figure 21B shows how the robot can "manipulate" the object even when this one is in contact.

References

[1] R. Alami, T. Siméon, J.P. Laumond, "A geometrical Approach to planning Manipulation Tasks. The case of discrete placements and grasps." International Symposium on Robotics Research, Tokyo, August 1989.

[2] J. Barraquand and J.C. Latombe, "Robot motion planning : a distributed representation approach," in *Int. Journal of Robotics Research*, 10 (6), 1991.

[3] J. Barraquand and P. Ferbach, "A penalty function method for constrained motion planning," in *IEEE* Conf. on Robotics and Automation, 1994.

[4] P.C. Chen, Y.K. Hwang, "Practical path planning among movable obstacles." in *IEEE* Conf. on Robotics and Automation, 1991.

[5] B. Dacre-Wright, J.P. Laumond, R. Alami, "Motion planning for a robot and a movable object amidst polygonal obstacles," in *IEEE* Conf. on Robotics and Automation, 1992.

[6] S.J. Fortune, "A fast algorithm for polygon containment by translation", ICALP, 1985.

[7] S. Fortune, G. Wilfong, C. Yap, "Coordinated motion of two robot arms." in *IEEE* Conf. on Robotics and Automation, 1986.

[8] Y. Koga and J.C. Latombe, "Experiments in dual-arm manipulation planning," in *IEEE* Conf. on Robotics and Automation, 1992.

[9] Y. Koga and J.C. Latombe, "On multi-arm manipulation planning," in *IEEE* Conf. on Robotics and Automation, 1994.

[10] J.-C. Latombe, *Robot Motion Planning*, Kluwer Press, 1991.

[11] J.P. Laumond, R. Alami, "A new geometrical approach to manipulation task planning: the case of a circular robot amidst polygonal obstacles and a movable circular object." Technical Report LAAS/CNRS 88314, Toulouse, Octobre 1988.

[12] J.P. Laumond, R. Alami, "A geometrical approach to planning manipulation tasks in Robotics.", Technical Report LAAS/CNRS 89261, Toulouse, August 1989.

[13] K. M. Lynch, "Stable pushing : mechanics, controllability and planning," in this volume.

[14] D. Parsons, J. Canny, "A motion planner for multiple mobile robots." in *IEEE* Conf. on Robotics and Automation, 1990.

[15] J.T. Schwartz, M. Sharir, "On the piano mover problem III : Coordinating the motion of several independant bodies : the special case of circular bodies amidst polygonal barriers." Int. J. of Robotics Research, 2 (3), 1983.

[16] M. Sharir, S. Sifrony, "Coordinated motion planning for two independent robots." ACM Symposium on Computational Geometry, 1988.

[17] T. Siméon, "Planning collision-free trajectories by a configuration space approach," in *Geometry and Robotics*, J.D. Boissonnat and J.P. Laumond Eds, Lecture Notes in Computer Science, 391, Springer Verlag, 1989.

[18] G. Wilfong, "Motion planning in the presence of movable obstacles." in *ACM* Symp. on Computational Geometry, 1988.