Journal of Robotic Research, MIT Press. Volume 17, Number 7, July 1998

Manipulation Task Planning for Redundant Robots: A Practical Approach

Juan Manuel Ahuactzin *
e-mail : jmal@udlapvms.pue.udlap.mx
Depto. de Ing. en Sistemas Computacionales
Universidad de las Américas, Puebla
Cholula, Puebla 72820, Mexico

Kamal Gupta
e-mail : kamal@cs.sfu.ca
School of Engineering Science
Simon Fraser University
Burnaby, B.C. V5A 1S6
Canada

Emmanuel Mazer
e-mail : manu@lifia.imag.fr
CNRS-IMAG/LEIBNIZ
38031, Grenoble, France

June 22, 2001

* **Acknowledgemnts** : This work was carried out while the author was a post doctoral fellow at Simon Fraser University and was supported in part by a Collaborative Research and Development grant from National Science and Engineering Research Council, Canada.

Abstract

An emerging paradigm in solving the classical motion planning problem (among static obstacles) is to capture the connectivity of the configuration space using a finite (but possibly large) set of landmarks (or nodes) in it [2, 8, 15, 23, 24]. In this paper, we extend this paradigm to manipulation planning problem, where the goal is to plan the motion of a robot so that it can move a given object from an initial configuration to a final configuration while avoiding collisions with the static obstacles and other movable objects in the environment. Our specific approach adapts the *Adriadne's Clew Algorithm* that has been shown effective for classical motion planning problem [2, 23]. In our approach, landmarks are placed in lower dimensional sub-manifolds of the composite configuration space. These landmarks represent stable grasps that are reachable from the initial configuration. From each new landmark, the planner attempts to reach the goal configuration by executing a local planner, again in a lower (but different) dimensional sub-manifold of the composite configuration space. The approach is resolution complete. We have experimented with this approach for a 7-*dof* manipulator in 3-D environments with one movable object and computation times range between a few minutes to a few tens of minutes – in our experiments, between 3 minutes to 15 minutes – depending on the task difficulty.

1 Introduction

Robot Motion Planning deals with automatic computation of robot motions to achieve some goal arrangement of objects. A main advantage of automatic motion planning is that it relieves the operator of the continual and considerable burden of specifying detailed motion and of collision avoidance. Most research has concentrated on the *classical motion planning problem* – that of planning the collision-free motions of a manipulator from a given initial configuration to a given goal configuration among static obstacles [19]. However, a robot may have to perform tasks which involve movable objects – objects that can be moved by the robot. In fact, a typical pick and place task in an industrial setting would routinely involve grasping a payload and moving it to a desired location. The robot may have to change grasp in order to place the payload in its desired location. The class of problems that deal with movable objects is called the *manipulation planning problem* [19]. One version of this can be stated as follows : plan the motion of a robot so that it can move a given object from an initial configuration to a final configuration while avoiding collisions with the static obstacles and other movable objects in the environment.

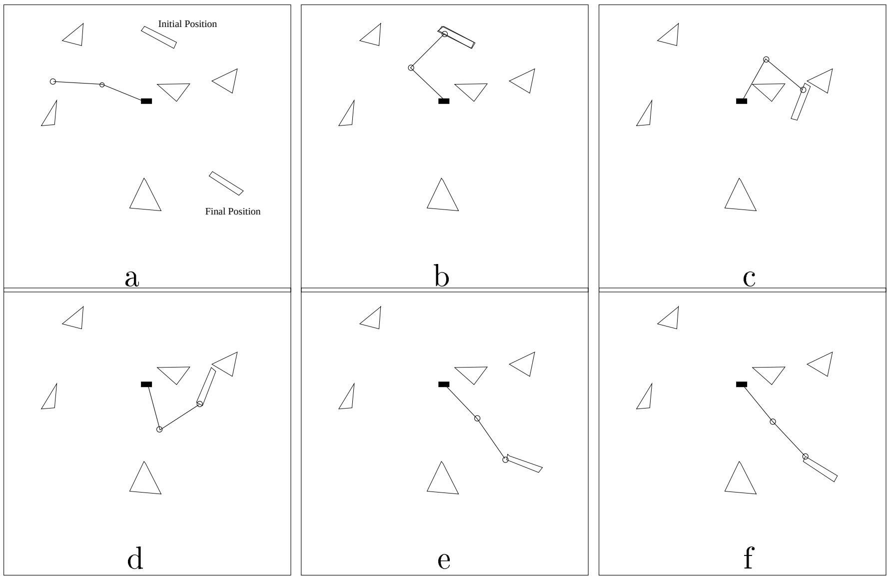


Figure 1: A manipulation path with a 2-*dof* planar arm.

Manipulation planning is a more general and more computationally complex version of the classical motion planning problem [19] since it deals with *movable* objects and involves interaction of the classical problem with regrasping tasks – dynamically changing grasp and ungrasp motions [27]. Alami *et al.* [3, 4] have shown that the solution, a *manipulation path*, can be obtained as a sequence of sub-paths separated by grasp and ungrasp operations. These paths are called *transfer* paths (the robot moves while holding the object in the same grasp), and *transit* paths (the object stays stationary in a stable position and the robot changes grasp) [3]. See Figure 1 for a simple planar example with one movable object. The

2-*dof* arm should move the rectangular object from the initial position (at the top of Figure 1a) to the final position (at the bottom). To move this object, the arm must grasp (with the end-effector) one of the object's edges as shown in 1b. The robot then moves toward (transfer path) the goal but the obstacles prevent the robot from continuing (c). The robot then regrasps the object (transit path) (d). This new grasp permits the robot to reach the goal position (e and f).

1.1 Nature of Manipulation Planning

A formal framework for manipulation planning and the nature of manipulation paths (concatenation of transit and transfer paths) has been investigated in detail by Alami *et al.* [3, 4]. The transit and transfer paths lie in lower-dimensional sub-spaces of the composite configuration space of the robot and the movable object. These sub-spaces are shown in Figure 2b and 3b respectively. Figure 2a shows the initial configuration of the manipulator and the movable object (a quadrilateral) $\mathcal{M}$. A grasp point ($\circ$ on one edge of $\mathcal{M}$) is also shown. The sub-manifold (formally defined later) of the composite configuration space with $\mathcal{M}$ placed as in 2a is shown in 2b and corresponds to all (collision-free) configurations of the robot while $\mathcal{M}$ stays in its place. The grey shaded region(s) correspond to collisions between the robot and $\mathcal{M}$. A transit path, with the initial configuration of the manipulator as in Figure 2a, and the final configuration as the given grasp point, is also shown in 2b. The physical path corresponding to the transit path in 2b, followed by the manipulator in its workspace is shown in 2c.

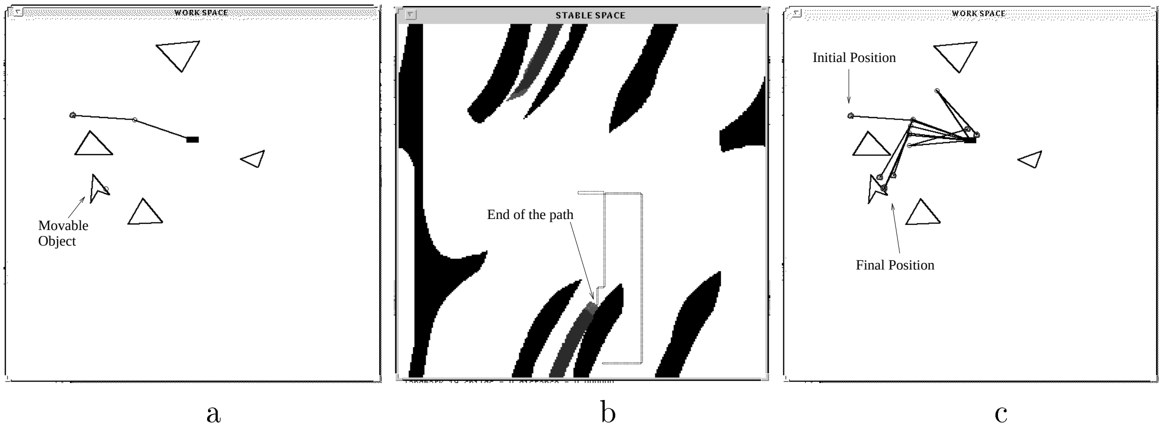
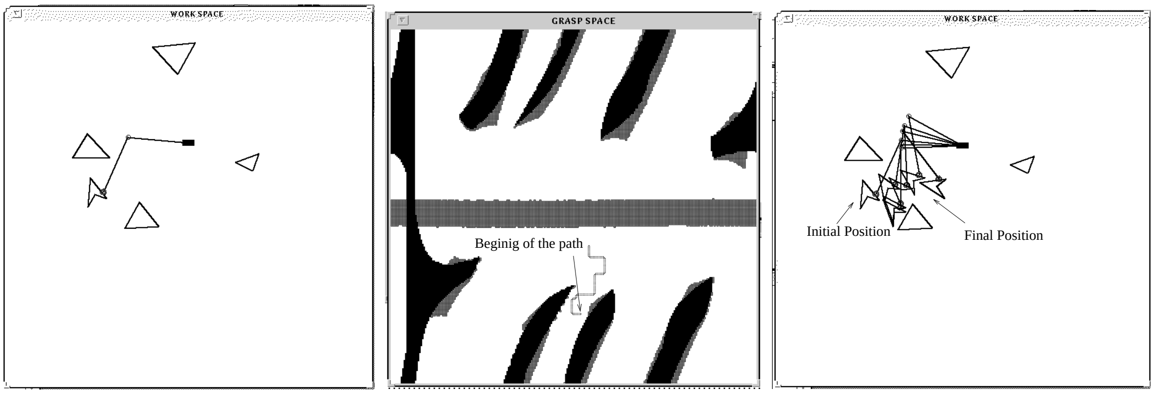


Figure 2: An example of sub-manifold $\mathcal{C}_{\hat{p}}$ and a transit path in it.

With the final grasp shown in Figure 2c, a transfer path is shown in Figure 3. The grasp in Figure 3a is the same as the grasp in 2c. The sub-manifold (formally defined later) corresponding to the grasp shown in 3a is shown in 3b and is the set of all (collision-free) robot configurations with $\mathcal{M}$ being held in the same grasp. The grey shaded areas correspond to just $\mathcal{M}$ (in the given grasp) colliding with other obstacles. A transfer path is also shown in 3b. The corresponding manipulator path in the workspace is shown in 3c. Note that the



a b c

Figure 3: An example of sub-manifold $\mathcal{C}_{\hat{g}}$ and a transfer path in it.

start point of this path is exactly the same as the end point of the transit path in Figure 2b. The concatenation of the path in 2b with that of 3b forms a *single manipulation path.*

Manipulation Planning, is then, essentially searching a collection of these interconnected sub-manifolds for transit and transfer paths, each sub-manifold corresponding to a given grasp or a given placement of the movable object(s). Figure 4 shows a schematic representation of these sub-manifolds. The sub-manifolds corresponding to all possible placements of movable object(s) are schematically represented by rectangles (call this set $\mathcal{CP}$) and those corresponding to all possible grasps of movable object(s) by ellipses (call this set $\mathcal{CG}$). The number of different grasps reachable from a placement is then the number of ellipses intersecting the square and the number of different placements reachable with a given grasp is the number of rectangles intersecting an ellipse. The set $\mathcal{CP} \cap \mathcal{CG}$, or, the intersection of these sub-manifolds (schematically denoted by intersection between a rectangle and an ellipse) are the places where transit and transfer paths are connected. Note that the intersection between the sub-manifolds will be a finite number of configurations for a non-redundant robot, and an infinite number of configurations for a redundant robot.

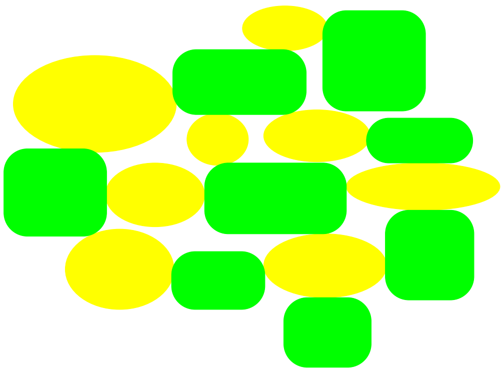


Figure 4: A schematic representation of the topology of the manipulation planning problem.

1.2 Previous Work

Earlier implemented planners for manipulation planning assumed a finite number of configurations in this intersection. For example, [3, 4] presented implemented planners in simple domains: one for the case of a 2-D polygonal mobile robot in a polygonal environment, and the other for a 3-*dof* planar manipulator. A manipulation graph with each node corresponding to a configuration in the intersection of these sub-spaces is then constructed and searched to find a path. This approach breaks down for redundant robots, i.e., if the possible configurations of the robot for a given grasp are infinite. Alami *et al.* [4] present an exact cell decomposition for the case when the set of grasps is infinite (for a polygonal robot with one movable object). Such exact decompositions are primarily of theoretical interest, and are impractical for realistic problems with manipulators in 3-dimensional environments.

Recently, [17] addressed the dual-arm manipulation planning in 2-D environments. A finite number of grasp points are defined for the movable object and a randomized planning technique [5] is used to obtain a free path of the movable object. During the search, the planner makes sure that at least one grasp exists for each of the robots (using inverse kinematics). The robots follow the trace of the grasp point trying to keep the same type of grasp. When the trace can not be followed continuously, a transit path is executed to change the grasp. The same approach was extended to manipulation tasks in sparse 3-dimensional environments with multiple PUMA 560 manipulators with impressive results [18]. The approach, however, assumes that a closed form inverse kinematics solution is available for each of the arms.

Handey [22], considered a key work, integrated a path planner with a grasp planner. The system was capable of re-grasping an object [27]. However, it was limited in that the grasp planner was for a simple parallel jaw gripper and the path planner was limited to manipulators with 3 or 4-*dof* . Nevertheless, it touched upon several important issues in integrating path planning with grasp planning.

One of the emerging paradigms (although there are differences in specific details) in solving classical motion planning problems is to capture the connectivity of the configuration space using a finite (but possibly large) set of landmarks (or nodes) in it [2, 8, 15, 23, 24]. [2, 23] presented an algorithm, called *Ariadne's Clew Algorithm* (*ACA*) that can be used to search for a path in a continuous domain. It is composed of two sub-algorithms : $EXPLORE$ and $SEARCH$, executed in an interleaved manner. The $EXPLORE$ algorithm "explores" the reachable space from a given initial point by iteratively placing landmarks in it. The landmarks are placed so that a path from the initial position to any landmark is known. The $SEARCH$ algorithm is a local planner that verifies if the goal configuration can be reached from a newly placed landmark. Figure 5 shows a schematic representation of the process. Each $\bullet$ denotes a landmark L_i, with L_0 being the start configuration. The continuous paths show the trajectories used by $EXPLORE$ to place a landmark and the dotted ones show the attempts by $SEARCH$ to reach the goal position. The tip of the arrows shows how far the local planner $SEARCH$ could proceed. In the schematic, $SEARCH$ succeeds from L_{11}. The final trajectory will be then constructed using the paths leading to L_0,L_2,L_9,L_{11} and the path to the goal point found by $SEARCH$. Both algorithms are expressed and solved as optimization problems using a particular set of paths – the *Manhattan paths* – and the objective function is the Euclidean distance between robot configurations. The resolution completeness of this algorithm (for classical motion planning) has been shown in [2].

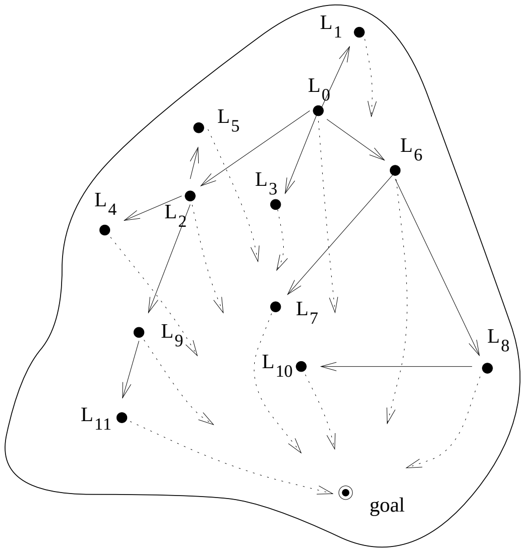


Figure 5: Schematic representation of the Ariadne's Clew Algorithm.

2 Our Approach

In this paper, we extend this paradigm to manipulation planning problem. Our approach may be viewed as *ACA* framework being applied hierarchically at different levels. We call our algorithm ACA_MANIP. At the top level, it is composed of *E_MANIP* (*EXPLORE*) and S_MANIP (*SEARCH*) respectively. *E_MANIP* tries to spread landmarks in $\mathcal{CP} \cap \mathcal{CG}$ from the initial configuration. These landmarks represent (robot configurations at) grasps that are reachable from the initial configuration. By construction, a new landmark is reachable from at least one of the previously placed landmarks via a *single manipulation path*. Each time we obtain a new landmark, we use S_MANIP to try to reach the goal configuration from the current landmark via a free *transfer path*; if it fails, another landmark is placed by *E_MANIP*. Essentially, a tree of landmarks is formed with initial configuration as the root. Figure 6 shows the tree and the configurations where the landmarks have been placed. Each of the landmarks, drawn with a $\bullet$, is connected to its parent by a single manipulation path (continuous line). In order to give an idea of the reachable configurations in $\mathcal{CP} \cap \mathcal{CG}$ from a particular landmark, we have used dashed rectangles to show the reachable configurations from the root, shown as $+$.

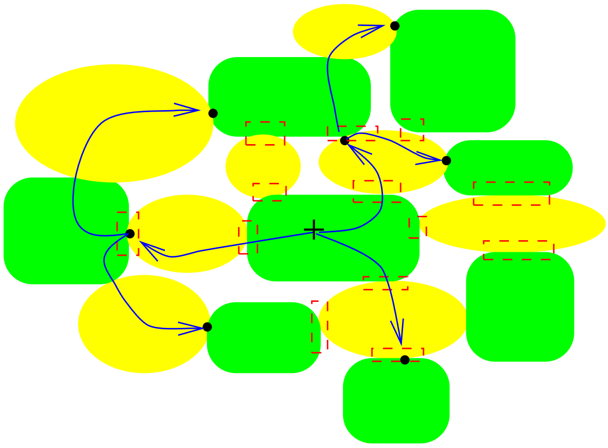


Figure 6: Schematic representation of the placement of landmarks for manipulation planning.

In the process of placing these landmarks, note that *E_MANIP* needs to find collision-free transit and transfer paths. Each of these sub-problems, essentially classical motion planning problems are also solved using *ACA* paradigm – the corresponding algorithms being called *ACA_TRANSIT* and ACA_TRANSFER, respectively – albeit in different spaces. The hierarchical nature of *E_MANIP* is clearly illustrated in Figure 7. The landmarks placed by *E_MANIP* via the use of single manipulation paths are indicated by $\bullet$. *ACA_TRANSIT* and ACA_TRANSFER (used by *E_MANIP*), in turn, also place "sub-landmarks" ([] and $\circ$, respectively) in the corresponding sub-manifolds. *Rather than having fixed points in the composite configuration space (as was the case in the previous works), our approach characterizes the corresponding sub-spaces by placing landmarks.* Both S_MANIP and *E_MANIP* are expressed as optimization problems (as in the standard *ACA* paradigm), however, the objective function to be optimized in our case is a suitable distance between end-effector frames rather than the robot configurations.

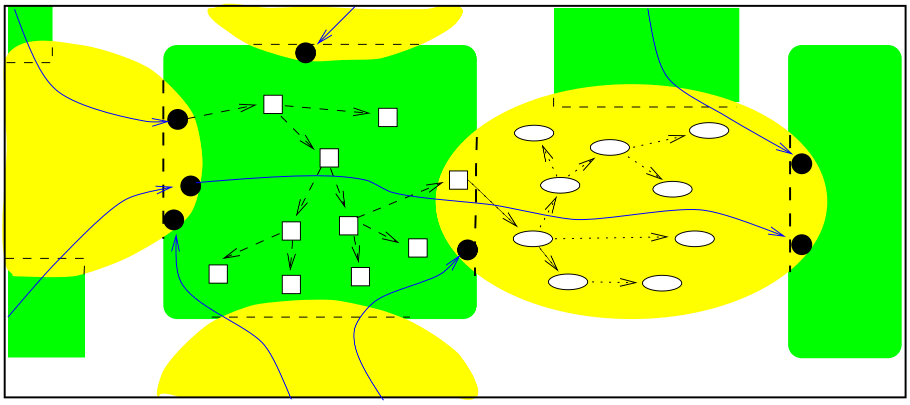


Figure 7: "sub-landmarks" placed by *ACA_TRANSIT* and ACA_TRANSFER.

The advantage of our approach is that we can now easily deal with the case when the cardinality of the set $\mathcal{CP} \cap \mathcal{CG}$ is infinite. As mentioned above, this occurs for redundant manipulators. Furthermore, it does not assume that a closed form inverse kinematics solution for the manipulator is available. We also demonstrate that our approach is practical for realistic problems in 3-dimensional environments with fairly large number of degrees of freedom manipulator arms.

Our initial implementation was for the simple case of a 2-*dof* planar arm among polygonal obstacles. This simplified domain allowed us to better understand the approach. The example in Figure 1 was obtained with this implementation. We have now implemented this approach for a 7-*dof* manipulator arm in realistic 3-D environments with one movable object (the payload). Figures 22 and 23 show two examples of manipulation paths found by the planner (see Section 8 for more details). Note that the manipulator carries out a number of grasp changes before it is able to place the payload in the desired configuration. For these examples, 10 pre-defined grasps were assumed around the periphery of the payload. We also assumed that all collision-free placements are stable.

The paper is organized as follows. We give some basic definitions in Section 3. In Section 4, an overview of *ACA* as a general algorithm for classical motion planning is given, and in Section 5 we describe planners for transit and transfer paths using the *ACA* paradigm. In Section 6, we describe our approach to manipulation planning in detail. In Section 7, we describe how genetic algorithms have been used to solve the optimization routines at the core of our approach. Section 8 discusses some experiments and performance of our approach and Sections 9 and 10 discuss some extensions and conclusions.

3 Basic Definitions and Problem Statement

3.1 Notation

Let $\mathcal{A}$ denote the robot, $\mathcal{G}$ denote its gripper, $\mathcal{M}$ denote the movable object, $\mathcal{B}_{i=1,2,3,\ldots}$ denote the static objects (obstacles). $\mathcal{F}_\mathcal{W}$ denotes the world frame, $\mathcal{F}_\mathcal{M}$ denotes a frame attached to $\mathcal{M}$, and $\mathcal{F}_\mathcal{G}$ denotes a frame attached to the gripper $\mathcal{G}$. A configuration of the robot in $\mathcal{F}_\mathcal{W}$ is completely specified by $\hat{q}_\mathcal{A} = (x_1, x_2, \ldots, x_n)$. Let ${}^\mathcal{W}\hat{q}_\mathcal{M} = (y_1, y_2, \ldots, y_m)$ denote the configuration of $\mathcal{M}$ w.r.t. the world frame $\mathcal{F}_\mathcal{W}$. Let $T({}^\mathcal{W}\hat{q}_\mathcal{M}) =^\mathcal{W} T_\mathcal{M}$ denote the homogenous transformation matrix that represents frame $\mathcal{F}_\mathcal{M}$ corresponding to the configuration ${}^\mathcal{W}\hat{q}_\mathcal{M}$. $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ denote the configuration space of $\mathcal{A}$ and $\mathcal{M}$, respectively. We will make the assumption that $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ are compact sets. The configuration space of the entire system is then $\mathcal{C} = \mathcal{C}_\mathcal{A} \times \mathcal{C}_\mathcal{M}$ and its dimension is $(n+m)$. Let $\mathcal{C}_{free}$ denote the collision-free configuration space (where none of the objects are in collision), and $\mathcal{CB}$ denote its complement. The projections of $\hat{q} \in \mathcal{C}$ in $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ are denoted by $\pi_\mathcal{A} \ : \hat{q} \in \mathcal{C} \rightarrow \hat{q}_\mathcal{A} \in \mathcal{C}_\mathcal{A}$, and $\pi_\mathcal{M} \ : \hat{q} \in \mathcal{C} \rightarrow \hat{q}_\mathcal{M} \in \mathcal{C}_\mathcal{M}$, respectively.

For our purposes it suffices to assume that a grasp is essentially defined by ${}^\mathcal{G}\hat{q}_\mathcal{M}$; this can be thought of as a pre-shaping configuration [6] and is assumed to be given by a grasp planner. When the pre-shaping configuration is reached, it is assumed that a fine motion planner is used to execute the final grasp. When the robot is moving with the movable object grasped in its gripper, ${}^{\mathcal{G}}\hat{q}_{\mathcal{M}}$ remains constant and ${}^{\mathcal{W}}\hat{q}_{\mathcal{M}}$ is changing. Let $\Phi(\hat{q}_{\mathcal{A}}, {}^{\mathcal{W}}\hat{q}_{\mathcal{M}})$ be the function that transforms the configuration of $\mathcal{M}$ from $\mathcal{F}_{\mathcal{W}}$ to the gripper's reference frame $\mathcal{F}_{\mathcal{G}}$ [1]. For brevity, we will simply denote a grasp ${}^{\mathcal{G}}\hat{q}_{\mathcal{M}}$ by $\hat{g}$, and a placement ${}^{\mathcal{W}}\hat{q}_{\mathcal{M}}$ by $\hat{p}$.

3.2 Transit, Transfer and Manipulation Paths

Let $\hat{p} = {}^{\mathcal{W}}\hat{q}_{\mathcal{M}}$ be a placement (a stable configuration) of $\mathcal{M}$ described w.r.t. $\mathcal{F}_{\mathcal{W}}$. The sub-manifold $\mathcal{C}_{\hat{p}} \subset \mathcal{C}_{free}$ is defined as the set of all (free) robot configurations with the movable object at the given placement, $\hat{p}$. This sub-manifold was schematically denoted by a rectangle in Figure 6 and was illustrated in Figure 2b. Formally, $\mathcal{C}_{\hat{p}} = \{\hat{q} \in \mathcal{C}_{free} | \pi_{\mathcal{M}}(\hat{q}) = \hat{p}\} = (\mathcal{C}_{\mathcal{A}}, \hat{p})$.

Definition 1 *A transit path*[2] *is a continuous map* $\hat{\varphi} : [0,1] \rightarrow \mathcal{C}_{\hat{p}}$.

For a given stable grasp $\hat{g}$, the sub-manifold $\mathcal{C}_{\hat{g}} \subset \mathcal{C}_{free}$ is defined as the set of all robot configurations holding the movable object at grasp $\hat{g}$. Note that the dimension of this sub-manifold is n (the number of *-dof* of $\mathcal{A}$). This sub-manifold was schematically denoted by an ellipse in FIgure 6 and for the simple planar example was illustrated in Figure 3b. Formally, $\mathcal{C}_{\hat{g}} = \{\hat{q} \in \mathcal{C}_{free} | \Phi(\hat{q}) = \hat{g}\}$.

Definition 2 *A transfer path is a continuous map* $\hat{\tau} : [0,1] \rightarrow \mathcal{C}_{\hat{g}}$.

Definition 3 *A single manipulation path* $\hat{\sigma}$ *is defined as a concatenation [19]*[3] *of a transit path* $\hat{\varphi}$ *and a transfer path* $\hat{\tau}$, $\hat{\sigma} = \hat{\varphi} * \hat{\tau}$. *A manipulation path of order* ℓ, *denoted by* $\hat{\sigma}^{\ell}$ *is defined as a concatenation of* ℓ *single manipulation paths, i.e.,* $\hat{\sigma}^{\ell} = \hat{\sigma}_1 * \hat{\sigma}_2 * \ldots * \hat{\sigma}_{\ell}$

The concatenation of the path in 2b with that in 3b forms a single manipulation path. Let $P = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_k\}$ be a set of stable placements of $\mathcal{M}$ in $\mathcal{W}$. The entire sub-manifold generated by P is then,

$$\mathcal{CP} = \mathcal{C}_{\hat{p}_1} \cup \mathcal{C}_{\hat{p}_2} \cup \ldots \cup \mathcal{C}_{\hat{p}_k}$$

Similarly, let $G = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_m\}$ be the set of stable grasps. The sub-manifold generated by $\mathcal{G}$ is then,

$$\mathcal{CG} = \mathcal{C}_{\hat{g}_1} \cup \mathcal{C}_{\hat{g}_2} \cup \ldots \cup \mathcal{C}_{\hat{g}_m}$$

Our problem can now be stated as follows :

[1] Note that $\Phi(\hat{q}_{\mathcal{A}}, {}^{\mathcal{W}}\hat{q}_{\mathcal{M}})$ is computed by first computing ${}^{G}T_{\mathcal{M}} = {}^{G}T_{\mathcal{W}} \cdot {}^{\mathcal{W}}T_{\mathcal{M}}$, and then extracting the configuration space vector ${}^{\mathcal{W}}\hat{q}_{\mathcal{M}}$.

[2] We imply *collision-free* transit paths. For brevity, in all our definitions, the word *free* or *collision-free* has been left out and is assumed known by context.

[3] The concatenation of two paths $\hat{\varphi}$ and $\hat{\tau}$ is defined iff $\hat{\varphi}(1) = \hat{\tau}(0)$.

Manipulation Planning Problem: Given an initial configuration $\hat{q}_\circ \in \mathcal{C}_{free}$ and a desired final configuration $\hat{p}_\bullet$ of $\mathcal{M}$, a set of stable placements, P, and a set of grasps, G, find a manipulation path $\hat{\sigma}^k$ connecting the initial configuration to the final one, i.e., find a $\hat{\sigma}^k$ such that $\hat{\sigma}^k(0) = \hat{q}_\circ$ and $\pi_{\mathcal{M}}(\hat{\sigma}^k(1)) = \hat{p}_\bullet$.

The two main sub-problems, that of planning transit paths and transfer paths, respectively, are now stated formally. *E_MANIP* and S_MANIP will use solution to these sub-problems to place landmarks and search from them.

Transit Path Planning Problem: Given an initial configuration $\hat{q}_\circ \in \mathcal{C}_{\hat{p}}$, and a desired final grasp $\hat{g}_\bullet$, find a transit path $\hat{\varphi}$ that starts at $\hat{q}_\circ$ and ends in any configuration in the goal region $\mathcal{C}_{\hat{g}_\bullet}$.

Figure 2 shows the situation graphically for the 2-*dof* robot. Note that the goal, for redundant manipulators in general, corresponds to a region – the set of all robot configurations that satisfy the given grasp – $\mathcal{C}_{\hat{g}_\bullet}$.

Transfer Path Planning Problem: Given an initial configuration $\hat{q}_\circ \in \mathcal{C}_{\hat{g}}$ and a desired final placement $\hat{p}_\bullet$, find a transfer path $\hat{\tau}$ that starts at $\hat{q}_\circ$ and ends in any configuration in the goal region $\mathcal{C}_{\hat{p}_\bullet}$.

Again, Figure 3 shows the situation graphically for the 2-*dof* robot.

4 ACA_GEN: An overview

We now first give a description of the *ACA* as a general path planning technique in continuous spaces.

4.1 High Level Description of the *ACA*

Let $X \subset I\!R^n$ be a given continuous space. Given an initial point $\hat{x}_\circ \in X$, and a goal region $X_\bullet$ such that $X_\bullet \cap X \neq \emptyset$, the aim of the *ACA* is to find a path $\hat{\rho}$ (if it exists) that connects $\hat{x}_\circ$ to $X_\bullet$, that is, $\hat{\rho}(0) = \hat{x}_\circ$, and $\hat{\rho}(1) \in X_\bullet$.

As discussed in section 1.2, the two sub-algorithms *SEARCH* and *EXPLORE*, are expressed as optimization problems over the space of suitably parameterized paths. In our discussion, we will assume that we have an algorithm *GA_MAX*(D, f) (*GA_MIN*, respectively)[4] that returns a maximum (minimum, respectively) of a function $f : D \rightarrow I\!R$, where D denotes the domain of the function f.

Let $\Omega(X, \hat{x})$ denote the set of paths in X beginning at $\hat{x} \in X$. Let $c(\hat{x}, X_\bullet)$ denote a cost function that indicates how "far" is $\hat{x}$ from the goal region $X_\bullet$.[5] The *SEARCH* algorithm essentially carries out the following optimization:

[4] As we will see subsequently, we have used genetic algorithms, however, other optimization techniques could be used.

[5] It will subsequently become clear that c is not necessarily a metric in X, in general. The main property required is that $\hat{x} \in X_\bullet \Leftrightarrow c(\hat{x}, X_\bullet) = 0$.

$$\begin{cases} \text{Minimize } c(\hat{\rho}(1), X_\bullet) \\ \hat{\rho} \in \Omega(X, \hat{x}) \end{cases} \tag{1}$$

or, stated in pseudo-code form:

$SEARCH(X, \hat{x}, X_\bullet)$
begin
    define $f(\hat{\rho}) = c(\hat{\rho}(1), X_\bullet)$ `[define the optimization function]`
    $\hat{\rho} = GA\_MIN(\Omega(X, \hat{x}), f)$ `[minimize` f `over` $\Omega(X,\hat{x})$`]`
    return$(f(\hat{\rho}))$ `[return ''distance'' to` $X_\bullet$`]`
end

Note that if $SEARCH$ succeeds, then $SEARCH(X, \hat{x}, X_\bullet) = 0$. $SEARCH$ is a local planner and therefore it searches some subset, Ω_ℓ of Ω, however, for brevity we will not make that distinction. In order to build a complete planner, the second algorithm, called *EXPLORE*, is used. While the purpose of $SEARCH$ was to directly look for a path from $\hat{x}$ to $X_\bullet$, the purpose of *EXPLORE* is to compute an approximation of the accessible region from $\hat{x}_\circ$ by placing landmarks all over it. The landmarks are placed by *maximizing* the distance from the current set of landmarks. Again, this is posed as an optimization problem over a suitable space of paths, as follows.

Let $\mathcal{L}_m$ denote the set of existing landmarks at step m, and Σ_m the set of paths which start from a point corresponding to one of the landmarks L_i, i.e., $\Sigma_m = \bigcup_{i=1}^{m} \Omega(X, L_i)$. We initialize the set of landmarks to $\mathcal{L}_1 = \{L_1\}$ where $L_1 = \hat{x}_\circ$. Let $\hat{\rho}_1$ be the path such that its endpoint is the farthest from $\mathcal{L}_1$, i.e.,

$$\hat{\rho}_1 : \max_{\hat{\rho} \in \Sigma_1} dist(\hat{\rho}(1), L_1)$$

The endpoint of $\hat{\rho}_1$ becomes the second landmark, i.e., $L_2 = \hat{\rho}_1(1)$ and $\mathcal{L}_2 = \mathcal{L}_1 \cup \{L_2\}$. For the next iteration, the third landmark is placed by maximizing the distance from $\mathcal{L}_2$ over the space of paths starting from either landmark in the set[6],i.e.,

$$\hat{\rho}_2 : \max_{\hat{\rho} \in \Sigma_2} dist(\hat{\rho}(1), \mathcal{L}_2) = \max_{\hat{\rho} \in \Sigma_2} \min_{L_i \in \mathcal{L}_2} dist(\hat{\rho}(1), L_i)$$

More generally, for each iteration m we denote

$$\begin{cases} \text{Maximize } dist(\hat{\rho}(1), \mathcal{L}_m) \\ \hat{\rho} \in \Sigma_m \end{cases} \tag{2}$$

and we obtain the $(m+1)^{th}$ landmark by seting $L_{m+1} = \hat{\rho}_m(1)$. *EXPLORE* can then be written as follows:

[6]Note that this implies a distance between a point and a set: $dist(\hat{x}, Y) = \min_{\hat{y} \in Y} dist(\hat{x}, \hat{y})$

```
EXPLORE(m)
begin
    define f(ρ̂) = dist(ρ̂(1), 𝓛_m)   [define the optimization function]
    ρ̂_m = GA_MAX(Σ_m, f)            [maximize f over Σ_m]
    L_{m+1} = ρ̂_m(1)                [set the (m+1)^th landmark ]
    return(f(ρ̂_m))                  [return the distance to 𝓛_m ]
end
```

At a given iteration, the minimum distance between any two landmarks is defined as the resolution. It has been shown in [2] that if there exists an $\hat{x} \in X_\bullet$ such that $\hat{x}$ is reachable from $\hat{x}_\circ$, then for a desired finite resolution ϵ, there exists a finite n such that $EXPLORE$ will place a landmark within a distance ϵ from $X_\bullet$ at the n^{th} iteration. ACA is thus resolution complete.

Now, using $SEARCH$ and $EXPLORE$, we define ACA.

```
ACA_GEN(X, x̂_∘, X_•, ε)
begin [parameters initialization]
 L_1 = x̂_∘;
 𝓛 = {L_1};
 m = 1;
 path_found = false;
 repeat [place landmarks until solution found or the resolution reaches ε]
    if (SEARCH(X, L_m, X_•) == 0) [a path has been found]
       path_found = true;
    else                          [place a new landmark]
       ε_m = EXPLORE(m);
       𝓛 = 𝓛 ∪ {L_{m+1}};
       m = m + 1;
 until( ε_m < ε or path_found);
 if (path_found)                  [a solution has been found]
    return(history_of(L_m));
 else
    return(Null)                  [no trajectory exists with resolution ε]
```

The "$history\_of$" function is simply the concatenation of the paths between the ancestors of L_m and the path found by $SEARCH$.

4.2 Optimization with Manhattan Paths

4.2.1 Manhattan Paths

The optimization problems in $SEARCH$ and $EXPLORE$ are solved with a particular class of paths – the *Manhattan paths* – which consist of moving along a single dimension of $I\!R^n$ at a time. The main motivation in considering Manhattan paths is that (i) they can be represented by a vector of $I\!R^n$, (note that in general a path is represented by a function and not a vector), (ii) they define a naturally redundant search space which is well suited for posing the path planning problem as an optimization problem in $I\!R^n$, and (iii) the class is resolution complete in that if a path exists from one point $\hat{x}_\circ$ to another point $\hat{x}_\bullet$ and the minimal distance of this trajectory to the boundary of X is $\varepsilon > 0$, then there exists a Manhattan path in X from $\hat{x}_\circ$ to $\hat{x}_\bullet$ [2]. Intuitively this corresponds to approximating the trajectory (in $I\!R^n$) with a combination of piecewise constant and piecewise linear segments.

We define a single (of order 1) Manhattan path in X starting from $\hat{x} = (x_1, x_2, ..., x_n)$ as the function $\hat{\gamma} : [0,1] \to X$ where for $\alpha \in [0,1]$:

$$\hat{\gamma}(\alpha) = (\gamma_1(\alpha), \gamma_2(\alpha), \ldots, \gamma_n(\alpha)) \in X \quad \text{and}$$
$$\gamma_i(\alpha) = \begin{cases} x_i & \text{for} \quad 0 \le \alpha \le \frac{(i-1)}{n} \\ x_i + \Delta_i(n\alpha - i + 1) & \text{for} \quad \frac{(i-1)}{n} \le \alpha \le \frac{i}{n} \\ x_i + \Delta_i & \text{for} \quad \frac{i}{n} \le \alpha \le 1 \end{cases}$$

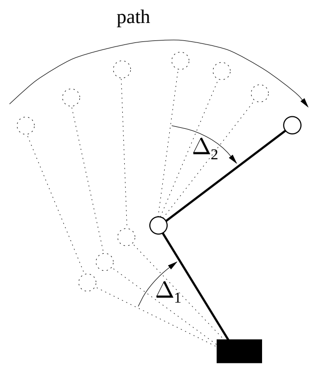


Figure 8: A single Manhattan path (Δ_1, Δ_2) for a 2-*dof* robot.

The motion, $\gamma_i(\alpha)$, of i^{th} component i is composed of three parts. In the first part, component i stays constant (with value x_i) while the previous components $1, 2, .., i-1$ are in motion one after the other. In the second part, component i linearly moves from x_i to $x_i + \Delta_i$ in duration $\frac{1}{n}$ while other components stay constant. Finally, in the third part, component i stays constant with value $x_i + \Delta_i$ while components $i+1, i+2, ..., n$ are in motion one after the other. $\hat{\gamma}$ is therefore completely defined by $\hat{\gamma} = (\Delta_1, \Delta_2, ..., \Delta_n)$ with $\Delta_i \in I\!R$. The semantic of this path is "move in dimension 1 a distance Δ_1" followed by "move in

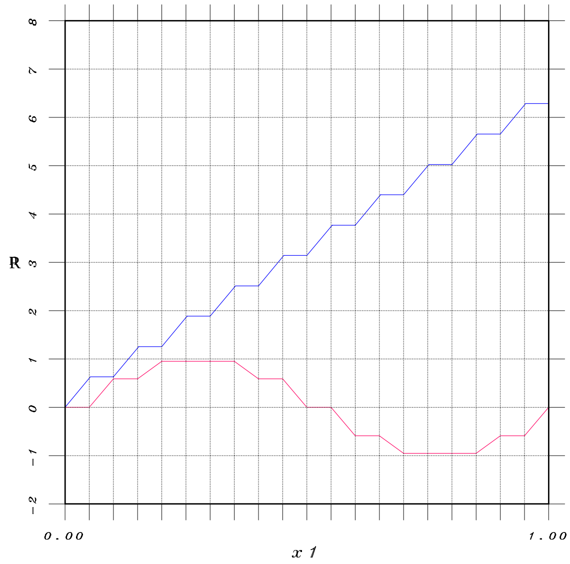


Figure 9: A Manhattan path of order 10 in 2-D. The two components γ_1 and γ_2 are shown here.

dimension 2 a distance Δ_2", and so on. For a robot with n degrees of freedom, it consists of moving one robot joint – starting from joint one and ending at the end effector joint – at a time. Figure 8 shows a single Manhattan path for the 2 -*dof* robot. The concatenation of k single Manhattan paths is a Manhattan path of order k, denoted by $\hat{\gamma}^k$. Figure 9 shows γ_1 and γ_2 for a Manhattan path of order 10 in $\mathbb{R}^2$. Finally, $\hat{\gamma}^k$ is represented as a vector with dimension nk, an element of $\mathbb{R}^{nk}$.

From now on, we will use Ω to denote the *set* of Manhattan paths. $\Omega(\mathbb{R}^n, \hat{x}, k)$, therefore, denotes the space of all manhattan paths in $\mathbb{R}^n$, starting at $\hat{x}$ and of order $\leq k$. The space $\Omega(\mathbb{R}^n, \hat{x}, k)$ is formed by two complementary subsets, the set of admissible (or collision-free) paths that lie completely within the admissible region X, $\Omega_{free} = \Omega(X, \hat{x}, k)$ and its complement, the set of paths that intersect with the inadmissible (or obstacle) regions, $\Omega\mathcal{B} = \Omega(\mathbb{R}^n, \hat{x}, k)/\Omega_{free}$, where / denotes set difference. Note that Ω_{free} (as does $\Omega\mathcal{B}$) depends on X, $\hat{x}$ and k, however, for brevity we are not showing it explicitly in our notation. For the configuration $\hat{q}_\circ$ of the 2-*dof* robot shown in Figure 1a, $\Omega(\mathcal{C}_{free}, \hat{q}_\circ, 1)$ is then nothing but a subset of $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$. The $\Omega\mathcal{B}$ (dark region) and Ω_{free} (white region) corresponding to this configuration are shown in Figure 10.

We can now express the optimization problem *GA_MIN* (similarly for *GA_MAX*) over the set of (collision-free) Manhattan paths as follows:

$$GA\_MIN(\Omega(X, \hat{x}, 1), f)$$

It returns a (free) manhattan path of order 1, (starting from $\hat{x}$), that is best in the sense that its extremity is closest to $X_\bullet$. The graph of function f (defined precisely in Section 5.1)

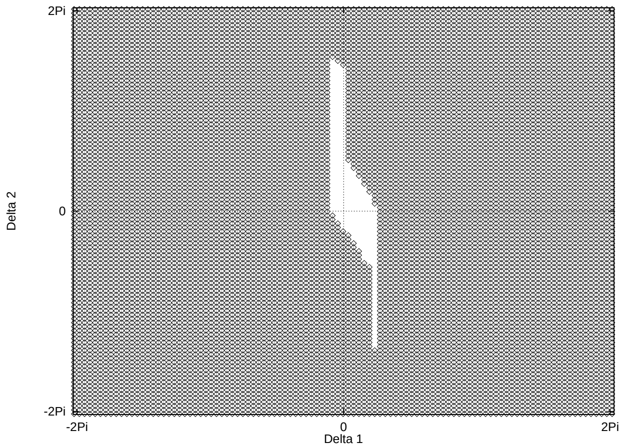


Figure 10: $\Omega\mathcal{B}$ and Ω_{free} for Manhattan path space of order 1.

for Ω_{free} shown in Figure 10 is depicted in Figure 11. *GA_MIN* will find a local minimum of this surface. To reach the goal, however, it may require a path of greater order. This can be achieved by successively calling *GA_MIN* and concatenating the resulting paths. The local planner *SEARCH* is now stated as follows :

$SEARCH(X, \hat{x}_\circ, X_\bullet)$
begin
  define $f(\hat{\gamma}) = c(\hat{\gamma}(1), X_\bullet)$
  $\hat{x}_1 = \hat{x}_\circ$
  $i := 0$
  do
    $i = i + 1$
    $\hat{\gamma}_i = GA\_MIN(\Omega(X, \hat{x}_i, 1), f)$
    $distance = f(\hat{\gamma}_i)$
    $\hat{x}_i = \hat{\gamma}_i(1)$
  while$((distance \neq 0)$ and $(\hat{x}_i \neq \hat{x}_{i-1}))$
  if $(distance = 0)$
    $\hat{\gamma} = (\hat{\gamma}_1 * \hat{\gamma}_2 * ... * \hat{\gamma}_i)$
  else
    $\hat{\gamma} = null\_path$
return$(distance)$
end

In a similar manner, *EXPLORE* can be implemented using Manhattan paths.

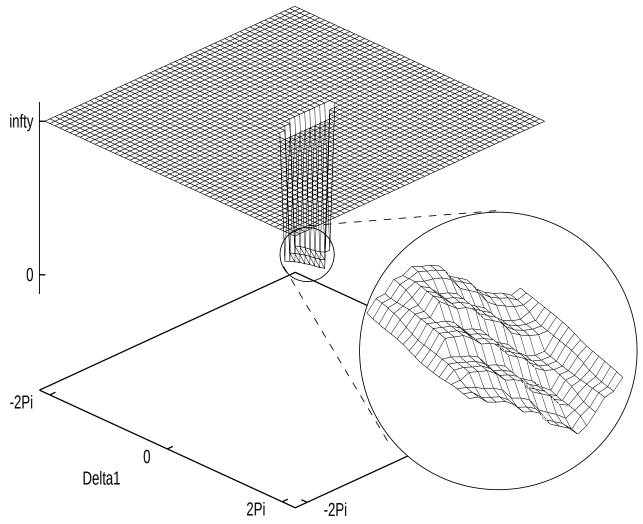


Figure 11: The graph of cost function f defined over Manhattan path space of order 1 (shown in Figure 10) represented as a surface.

4.2.2 Searching Ω_{free}: The Bouncing Technique

The actual search space for the minimization problem(s) is Ω_{free} and its complement region $\Omega\mathcal{B}$ may be quite large. Therefore, it would be computationally efficient to restrict the search explicitly to Ω_{free}. It is clear that any path of order 1 – this assertion is also true for manhattan paths of order > 1 – lying in Ω_{free} can be represented by a vector $\hat{x} \in I\!R^n$, but the converse is not always true. Not all $\hat{x} \in I\!R^n$ may represent a path in Ω_{free}, since it could code an invalid value such that the path $\hat{\gamma}$ goes into obstacle region.

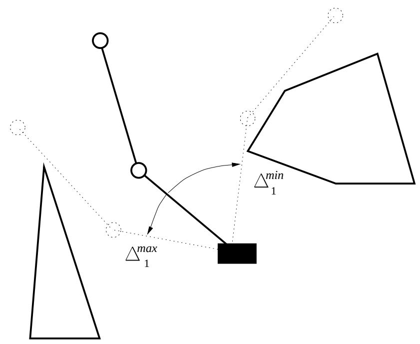


Figure 12: Illustration of collision-free interval $[\Delta_1^{min}, \Delta_1^{max}]$.

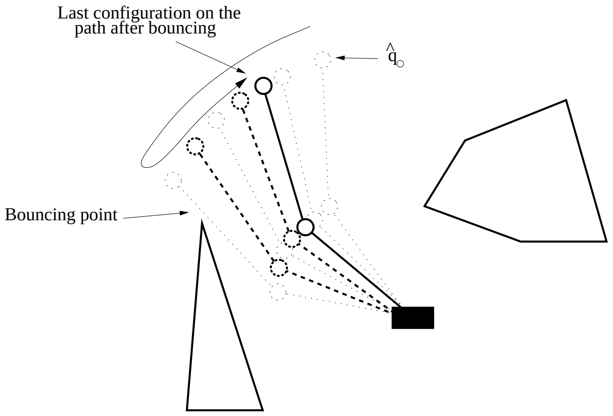


Figure 13: A physical illustration of bouncing technique.

However, collision-free intervals $[\Delta_i^{min}, \Delta_i^{max}]$ can be easily obtained (as in [21]) such that $\forall \Delta_i \in \hat{\gamma} \in \Omega_{free}$, $\Delta_i \in [\Delta_i^{min}, \Delta_i^{max}]$, and conversely, for any $\Delta_i \in \hat{\gamma} \in \Omega$, if $\Delta_i \leq \Delta_i^{min}$ or $\Delta_i \geq \Delta_i^{max}$ then $\hat{\gamma} \in \Omega\mathcal{B}$ (see Figure 12). Note that the range of joint $(i+1)$ depends on the moves of joint $1 \ldots i$, i.e., Δ_{i+1}^{min} and Δ_{i+1}^{max} are functions of Δ_j for $j = 1, 2, \ldots, i$. The computation of these intervals permits us to rewrite the functions $\gamma_i \in \hat{\gamma}^k$ so that the resulting paths are always collision-free. The basic idea is to "bounce off" the obstacle as shown in Figure 13. Below we give an elegant mathematical representation of this bouncing in terms of a periodic function.

Let $a = |\Delta_i^{max} - \Delta_i^{min}|$. The chosen periodic function is a triangular wave as shown in Figure 14 with amplitude a and period $2a$. Note that any value x (along the horizontal axis) can now be mapped to range $[\Delta_i^{min}, \Delta_i^{max}]$ along the vertical axis. We use this to redefine the i^{th} component, γ_i as follows:

$$\gamma_i(\alpha) = \begin{cases} x_i & \text{for} \quad 0 \leq \alpha \leq \frac{(i-1)}{n} \\ x_i + Triang(\Delta_{min}, \Delta_{max}, \Delta_i)(n\alpha - i + 1) & \text{for} \quad \frac{(i-1)}{n} \leq \alpha \leq \frac{i}{n} \\ x_i + \Delta_i & \text{for} \quad \frac{i}{n} \leq \alpha \leq 1 \end{cases}$$

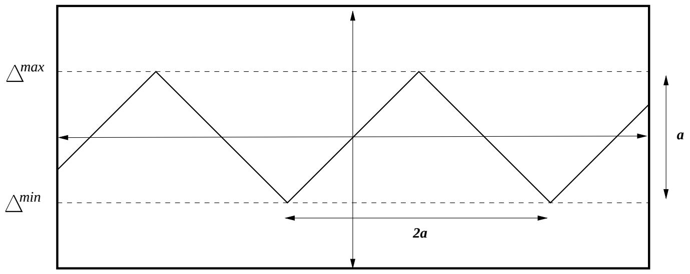


Figure 14: The periodic function $Triang$ used in bouncing technique.

A main advantage of this technique is that it permits us to explicitly code only feasible paths (the ones in Ω_{free}); there is no way to code a path belonging to $\Omega\mathcal{B}$, thereby making it very efficient by avoiding collision-checking in the interior of $\mathcal{CB}$. Effectively, the surface (obstacle regions shown in black) corresponding to the cost function f in Figure 11 is transformed to a periodic surface in Figure 15. With bouncing technique, *GA_MIN*, finds a local minimum of this surface.

This completes our discussion of the fundamental concepts underlying ACA in a general setting. In the next few sections, we present how ACA is used for manipulation planning.

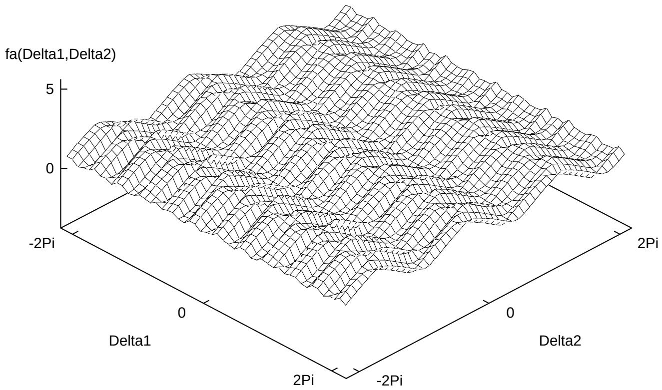


Figure 15: The *Triang* function used in bouncing technique effectively transforms the surface (the graph of cost function f) in Figure 11 to a periodic surface. Note that any path on this surface is collision-free.

5 Planning Transit and Transfer Paths

As discussed before, *transit* and *transfer* paths are the key components in manipulation planning. In this section we show how ACA_GEN is used to plan these paths.

Recall (Section 3.2) the problem of finding transit paths: given an initial configuration of the system $\hat{q}_\circ \in \mathcal{C}_{\hat{p}}$, and a desired final grasp $\hat{g}_\bullet$, find a transit path $\hat{\varphi}$ that connects the two. The goal region, $X_\bullet$, corresponds to the set of all robot configurations that satisfy the given grasp, i.e., $\mathcal{C}_{\hat{g}_\bullet}$. The search space X corresponds to the set of all robot configurations corresponding to the particular placement, i.e., $\mathcal{C}_{\hat{p}}$. Hence, a transit path planner is implemented with ACA_GEN as follows:

$$ACA\_TRANSIT(\hat{q}_\circ, \hat{g}_\bullet) \; = ACA\_GEN(\mathcal{C}_{\hat{p}}, \hat{q}_\circ, \mathcal{C}_{\hat{g}_\bullet}, \varepsilon)$$

The problem of finding a transfer path is: given an initial configuration $\hat{q}_\circ \in \mathcal{C}_{\hat{g}}$ and a desired final placement $\hat{p}_\bullet$, find a transfer path $\hat{\tau}$ that connects the two. The goal region now corresponds to the set of all robot configurations with the object placed at $\hat{p}_\bullet$, i.e., $\mathcal{C}_{\hat{p}_\bullet}$. The search space, X, is the set of all robot configurations corresponding to the grasp $\hat{g}$, i.e., $\mathcal{C}_{\hat{g}}$. A transfer path planner, is therefore, obtained with:

$$ACA\_TRANSFER(\hat{q}_\circ, \hat{p}_\bullet) = ACA\_GEN(\mathcal{C}_{\hat{g}}, \hat{q}_\circ, \mathcal{C}_{\hat{p}_\bullet}, \varepsilon)$$

Note that in Figure 6, *ACA_TRANSIT* corresponds to constructing a representation of the connectivity of a particular square while searching for a path that leads to a particular ellipse. Similarly, ACA_TRANSFER corresponds to constructing a representation of the connectivity of a particular ellipse while searching for a path that leads to a particular square.

5.1 The Cost Function and the Local Planners

We now define the precise cost function used in the optimization routine of *S_TRANSFER* (*SEARCH* routine for ACA_TRANSFER). For simplicity, we illustrate it with a simple 2-D example (Figures 17 and 16). The extension to 3-D is straightforward.

Consider transfer paths. One could use the Euclidean distance between the configuration vector corresponding to $\mathcal{M}$ at the end of the transfer path, $\hat{\tau}(1)$, and that corresponding to the desired final placement of $\mathcal{M}$, $\hat{p}_\bullet$. In 2-D, the vectors would be $\pi_{\mathcal{M}}(\hat{\tau}(1)) = (x_1, y_1, \theta_1)$, and $\hat{p}_\bullet = (x_2, y_2, \theta_2)$. It is well known that this metric mixes translational and rotational components and there is no natural way of doing it [19]. Instead, we have used a different cost function defined directly in terms of the coordinate frames (homogeneous transformation matrices).

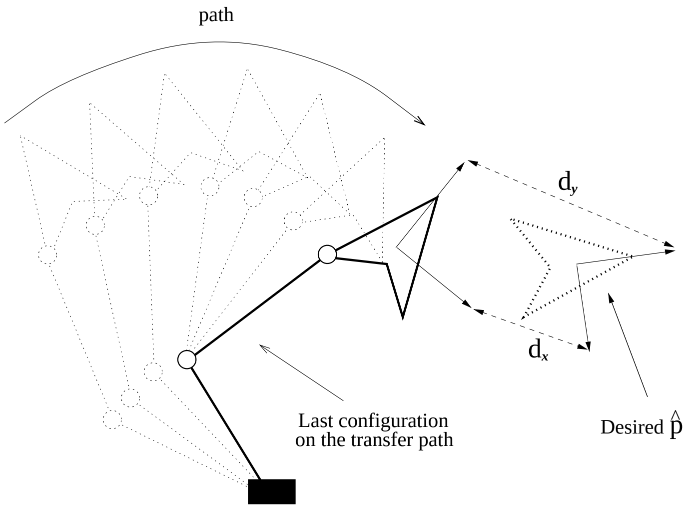


Figure 16: Cost function for *S_TRANSFER*.

We first define a metric d between two coordinate frames, say $\mathcal{F}_1$ and $\mathcal{F}_2$, (represented

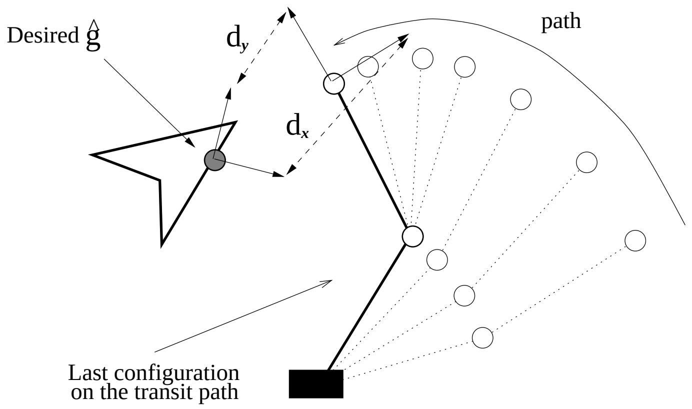


Figure 17: Cost function for *S_TRANSIT*.

w.r.t. $\mathcal{F}_{\mathcal{W}}$, by homogenous matrices ${}^{\mathcal{W}}T_1$ and ${}^{\mathcal{W}}T_2$) as the sum of distances between the corresponding unit vectors along the co-ordinate axes.

For the 2-D case,

$$d({}^{\mathcal{W}}T_1, {}^{\mathcal{W}}T_2) = d_x + d_y$$

where d_x and d_y (d_z in 3-D) are the distances between the unit vectors along the x and y axes, respectively (see Figure 16). Recall that $T(\hat{p})$ denotes the matrix ${}^{\mathcal{W}}T_{\mathcal{M}}$. The cost function c is then:

$$c(\hat{\sigma}(1), \mathcal{C}_{\hat{p}_\bullet}) = d(T(\pi_{\mathcal{M}}(\hat{\sigma}(1))), T(\hat{p}_\bullet))$$

Substituting this cost function in the generic *SEARCH* algorithm yields *S_TRANSFER*. Figure 16 shows an example of the computation of this distance. The cost function for *S_TRANSIT* is similar and is shown in Figure 17.

6 The Manipulation Algorithm: ACA_MANIP

As mentioned earlier, ACA_MANIP may be viewed as *ACA* framework being applied hierarchically at different levels. At the top level, it may be viewed as composed of *E_MANIP* and S_MANIP respectively. Recall that *E_MANIP* places landmarks in $\mathcal{CP} \cap \mathcal{CG}$ by searching the space of single manipulation paths, and S_MANIP verifies if a given configuration is reachable from a landmark via a transfer path. The algorithm ACA_MANIP is written as follows :

$$ACA\_MANIP(\hat{q}_\circ, \hat{p}_\bullet) = ACA\_GEN(\mathcal{C}_{free}, \hat{q}_\circ, \mathcal{C}_{\hat{p}_\bullet}, \varepsilon)$$

with the implicit assumption that the search space used by *E_MANIP* is restricted to the set of single manipulation paths in $\mathcal{C}_{free}$. This restriction guarantees that all landmarks are placed in $\mathcal{CP} \cap \mathcal{CG}$. The search espace of S_MANIP is restricted to the set of transfer paths.

6.1 *E_MANIP* and S_MANIP: Implementation

In this section, we explain how to build *E_MANIP*. As explained earlier, it must involve a mechanism to generate *single* manipulation paths, which, in turn are composed of a *transit* path followed by a *transfer* path. A single manipulation path $\hat{\sigma}$ starting at $L_i \in \mathcal{L}_m$ is generated by a concatenation of the transit path $\hat{\varphi}$ and a transfer path $\hat{\tau}$ as follows. For each L_i, m_1 grasps are generated and $ACA\_TRANSIT(L_i, \hat{g}_j)$ is used to generate m_1 corresponding transit paths $\hat{\varphi}_j,\ j = 1, m_1$. A similar scheme – choose m_2 placements and use ACA_TRANSFER to compute a transfer path $\hat{\tau}$ to it – may be used to obtain a transfer path for each of these transit paths. However, in our implementation, we assume that the movable object can be placed in *any* collision-free configuration, i.e., the set $\mathcal{CP}$ is the free configuration space $\mathcal{C}_{\mathcal{A}free}$ of the robot. In this case, a transfer path is obtained as follows. *E_MANIP* randomly generates m_2 transfer paths with the grasp corresponding to the transit path, i.e., the sub-manifold where $\hat{\tau}$ is executed depends on $\hat{\varphi}(1)$. Remember that the algorithm *ACA_TRANSIT* returns a null path if no path has been found for the grasp $\hat{g}_j$. In this case, the obtained manipulation path will keep the grasp that corresponds to L_i (i.e. there is no change of grasp). Therefore, in all, $m_1 \times m_2$ manipulation paths are generated from each landmark. The manipulation path $\hat{\sigma}$ such that $\hat{\sigma}(1)$ is the farthest from $\mathcal{L}_m$ is the one which is selected and $\hat{\sigma}(1)$ gives the $(m+1)^{th}$ landmark.

The algorithm *E_MANIP* can be written as follows :

E_MANIP(m)
begin
    define $f(\hat{\sigma}) = dist(\hat{\sigma}(1), \mathcal{L}_m)$
    for i = 1 to m
        for j = 1 to m_1
            choose a grasp $\hat{g}_j \in \mathcal{G}$
            $\hat{\varphi}_{i,j} = ACA\_TRANSIT(L_i, \hat{g}_j)$
            for k = 1 to m_2
                $\hat{\tau}_{i,j,k} = random\_manhattan\_path\_with(\hat{\varphi}_{i,j}(1))$
                $\hat{\sigma}_{i,j,k} = \hat{\varphi}_{i,j} * \hat{\tau}_{i,j,k}$
    $\hat{\sigma} = \max_{\hat{\sigma}_{i,j,k}} f(\hat{\sigma}_{i,j,k}(1))$
    return$(f(\hat{\sigma}))$
end

As mentioned earlier, we use genetic algorithms to solve the above maximization problem in *E_MANIP* (see Section 7). The above explanation is a high-level description of this process. In particular, m_1 and m_2 are related to the parameters used in the genetic optimization process.

S_MANIP essentially searches, from the newly placed landmark, for a path – a transfer path – to reach the desired goal configuration for the movable object. This is nothing but *ACA_TRANSFER*.

$$S\_MANIP(\hat{q}_\circ, \hat{p}_\bullet) = ACA\_TRANSFER(\hat{q}_\circ, \hat{p}_\bullet)$$

6.2 Completeness

As mentioned before, ACA_GEN is known to be resolution complete for the classical motion planning problem [2]. [**?**] has shown that for manipulation planning, for the case of discrete placements and grasps, it suffices to search for *single* manipulation paths.

Using the above two key properties, it is fairly straightforward to show that ACA_MANIP is resolution complete, since *ACA_TRANSIT* and ACA_TRANSFER are themselves resolution complete algorithms within the corresponding sub-spaces.

7 Implementing *GA_MIN* and *GA_MAX* with Genetic Algorithms

7.1 The Principle of Genetic Algorithms

Genetic Algorithms (G.A.) are stochastic optimization techniques introduced by Holland [14] in the 70's. Inspired by the natural process of selection and evolution, G.A. has been successfully used to solve different optimization problems [20]. In the last few years, this optimization technique has also been applied to solve different robotic problems [1, 2, 9, 10, 11].

Five main steps are involved in the execution of a G.A. : *initialization, evaluation, selection, reproduction and replacement*. Given a function $f : D \to I\!R$, also called the evaluation function, the goal of G.A. is to find an optimum of f over D using a discrete set of vectors S representing D. Therefore one needs to define a function $g : S \to D$ so that the composite function $f \circ g : S \to I\!R$ can be computed.

The most traditional set S used by a G.A. is the binary set $S = \{0, 1\}^n$ i.e. the set of binary strings of length n. Nevertheless, any set with $N \in I\!N$ symbols can be used to represent the domain of f. In this work, we use sets of finite binary strings to obtain a discrete representation of the domains of our functions to optimize. The principle of G.A. is as follows :

- **Stage 1 (Initialization)** : A random set $P = \{s_1, s_2, ..., s_m\} \subset S$, called the initial population, is generated. An element $s_i \in P$ is called an individual.
- **Stage 2 (Evaluation) :** The function $f(g(s))$ is used to evaluate all the individuals of P and order them from the best to the worst.
- **Stage 3 (Selection) :** Select m pairs $(s^j_{i_1}, s^j_{i_2})$ for $j = 1, 2, ..., m$ of individuals from P such that each $s^j_{i_1}$ has been selected with a probability proportional to the rank

obtained in the Evaluation Stage and $s^j_{i_2}$ has been selected with a uniform probability that is $1/m$. Note that over the m pairs of individuals each member of P can be selected more than once.

- **Stage 4 (Reproduction) :** Each pair $(s^j_{i_1}, s^j_{i_2})$ is used to generate a new individual $s_j{}'$ using an operation of combination. Many operations of combination has been defined [20]. However, the most traditional are **cross-over** and **mutation**. The cross-over consists in selecting a random position $k \in [0, 1, 2, ...n]$ (where n is the length of the individual) and then combining the left side of $s^j_{i_1}$ (from the k^{th} position) with the right side of $s^j_{i_2}$ obtaining $s^{j\prime}$ (see Figure 18).

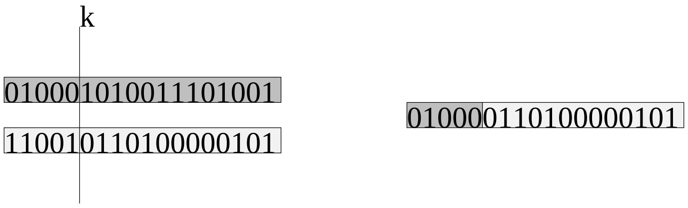


Figure 18: Cross-over between two individuals.

The mutation consists in flipping randomly one of the n possible values of an individual from 1 to 0 or 0 to 1 (see Figure 19). The principal goal of the mutation is to escape from a local minima.

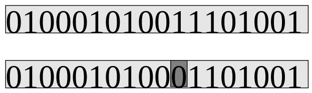


Figure 19: Mutation of an individual.

- **Stage 5 (Replacement) :** The population P is replaced by the new individuals $P = \{s_1{}', s_2{}', ..., s_m{}'\}$ obtained in the Reproduction Stage.

- **Stage 6 :** Goto stage 2 until a predefined condition is satisfied or a (global or local) minimum is found. In every generation, the best individual of the population is known. A local minimum is deemed to reach if the best individual doesn't change from one generation to another. Another termination condition may be to run the G.A. for a fixed number of generations.

7.2 Implementing *GA_MIN* and *GA_MAX* with G.A.

We now describe how *GA_MIN* (for *SEARCH*) and *GA_MAX* (for *EXPLORE*) are implemented. Our explanation is for transit paths, the case for transfer paths being very similar. The domain of the cost function c (see section 4.1) used by *GA_MIN* is the set of (free) Manhattan paths (of order 1), represented by $\hat{\gamma} = (\Delta_1, \Delta_2, ..., \Delta_{n-1}, \Delta_n)$, with

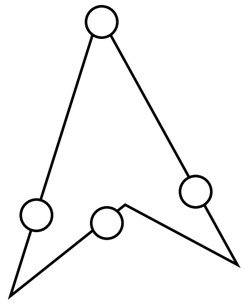


Figure 20: Distribution of the grasp points in 2-D.

$\Delta_{i=1,2,...,n} \in [-2\pi, 2\pi]$. This interval is discretized using N_1 bits, thereby, using $N_2 = N_1 n$ bits for representing a single Manhattan path. The discretized domain of c, consequently, is $S_1 = \{0,1\}^{N_2}$. The minimization problems solved by GA_MIN is then :

$$\min_{s \in S_1} c(g_a(s), \mathcal{C}_{\hat{g}_\bullet})$$

where g_a represents the decoding of binary strings $s \in S_1$ to $\hat{\gamma} \in \Omega(\mathcal{C}_{\hat{p}}, \hat{q}_\circ, 1)$.

The search space Σ_m for $E\_TRANSIT(m)$ is the set of Manhattan paths ($\in \mathcal{C}_{\hat{g}_j}$) of order ℓ, starting at any of the landmarks, L_i. This space is represented by $[1,2,...,m] \times [-2\pi, 2\pi]^{n\ell}$, where $[1,2,...,m]$ represents the set of landmarks and $[-2\pi, 2\pi]^{n\ell}$ represents the set of Manhattan paths. Assume that N_3 bits are used to code the *id* of a landmark (the point $\hat{q}$ corresponding to the landmark), allowing a maximum of 2^{N_3} landmarks. $N_4 = N_2\ell$ bits code a Manhattan path of order ℓ. The set of paths Σ_m is then represented by $S_2 = \{0,1\}^{(N_3+N_4)}$. The maximization problem solved by GA_MAX is then rewritten as :

$$\max_{s \in S_2} f(g_c(s), \mathcal{L}_m)$$

where g_c represents the decoding the function of a binary string $s \in S_2$ to $\hat{\gamma}^\ell \in \Omega(\mathcal{C}_{\hat{p}}, L_i, \ell)$.

In our implementation, 30 generations are used in the G.As. for minimizing or maximizing the corresponding functions.

7.3 Implementing *E_MANIP* with G.A.

The search space Σ_m for *E_MANIP* is the set of single manipulation paths which start from the configuration corresponding to one of the landmarks L_i and terminate at a configuration $\hat{q} \in \mathcal{CG} \cap \mathcal{CP}$. Using Manhattan paths of order ℓ, Σ_m is represented by :

$$[1,2,...,m] \times [1,2,...,k] \times [-2\pi, 2\pi]^{n\ell}$$

where $[1,2,...,m] \times [1,2,...,k]$ represents the set of transit paths from the landmarks in $\mathcal{L}_m$ to the set of grasps G (with cardinality k), and $[-2\pi, 2\pi]^{n\ell}$ represents the set of Manhattan transfer paths of order ℓ executed after the transit path.

We use N_3 bits to code the *id* of the landmark, and N_5 bits for representing the grasps, (2^{N_5} being the maximum number of grasps). These grasps are distributed on the edges of the movable object for the 2-D example (see Figure 20). For the 3-D example, the grasp points are distributed over the faces of the (polyhedral) movable object (see Figure 21). An individual s for GA_MAX is then of the form $s = (s_1, s_2, s_3)$ where s_1 codes a landmark in $\mathcal{L}_m$ using N_3 bits, s_2 codes a grasp of $\mathcal{M}$ using N_5 bits and s_3 codes a transfer path using N_4 bits. The corresponding search space, Σ_m for GA_MAX is therefore, $S_3 = \{0,1\}^{(N_3+N_5+N_4)}$.

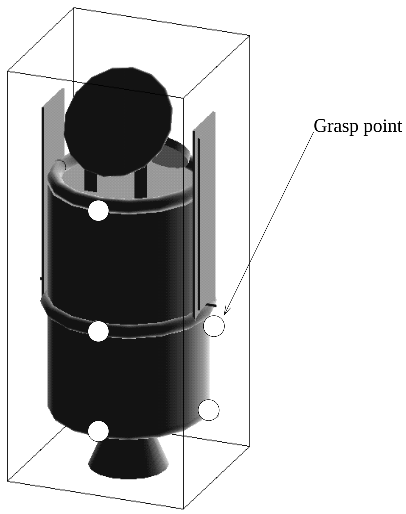


Figure 21: Distribution of the grasp points in 3-D.

The evaluation of an individual $s \in S_3$ is executed as follows :

- use $ACA\_TRANSIT$ to plan a transit path $\hat{\varphi}$ from landmark L_i coded by s_1 to the grasp $\hat{g}_j$ coded by s_2. If no such path exists, the same grasp given by L_i is kept (i.e. $\hat{\varphi}$ is the null path).
- Generate a transfer path $\hat{\tau}$ using the Manhattan path coded by s_3. The singe manipulation path $\hat{\sigma}$ beginning at L_i is then equal to $\hat{\varphi} * \hat{\tau}$.
- Use expression 2 to evaluate $\hat{\sigma}$, i.e., obtain the distance of $\hat{\sigma}(1)$ to the existing landmarks.

If no path has been found after 2^{N_3} landmarks have been placed, the algorithm shows the best solution found and returns failure, and reports the resolution that has been reached i.e. the minimum distance between the landmarks.

Table 1 summarizes the values for various parameters in our implementation for the 2-D and 3-D examples. The maximum number of landmarks used by $ACA\_TRANSFER$, $ACA\_TRANSIT$ and ACA_MANIP is $2^8 = 256$ and the sizes of the search spaces in the G.A's are summarized in Table 2.

Parameter	2-D	3-D
N_1 (bits for $[-2\pi, 2\pi]$)	9	9
N_2 (bits for $\hat{\gamma}^1$)	18	63
N_3 (bits for Landmark id)	8	8
N_4 (bits for $\hat{\gamma}^\ell : N_2\ell, \ell = 5$))	90	315
N_5 (bits for grasps)	3	4

Table 1: Various parameters for the Genetic Algorithms.

Algorithm	Search space	
	2-D	3-D
S_TRANSIT (TRANSFER)	$\{0,1\}^{18}$	$\{0,1\}^{63}$
$E\_TRANSIT(TRANSFER)$	$\{0,1\}^{98}$	$\{0,1\}^{323}$
E_MANIP	$\{0,1\}^{101}$	$\{0,1\}^{326}$

Table 2: The search spaces for the Genetic Algorithms.

8 Experimental Results

The example shown in Figure 1 was solved using ACA_MANIP. 4 grasp points, as shown in Figure 20 were defined around the periphery of the payload. It took about 4 minutes on an IPX Sparcstation. About seven landmarks were needed for this example.

The manipulation planner has also been implemented (on an SGI Indigo II) for a 7-*dof* manipulator arm in 3-dimensional environments for one movable object (payload). Figure 22 and Figure 23 show two examples of manipulation paths generated by the planner. Polyhedral CAD models were used for both the robot and the environment. The collision detection scheme is based on the algorithm in [21] that efficiently computes a collision-free interval for a given robot link as it rotates, keeping all other degrees of freedom fixed. These collision-free intervals are then used in the bouncing technique to convert a Manhattan path into a collision-free Manhattan path. Several standard techniques – simplified and hierarchical representations for manipulator links and obstacles, etc. – were employed for efficient collision detection. The robot links, the fixed obstacles, and the payload are each represented by a parallelopiped. For the examples shown in Figures 22 and 23, the bouncing technique for a single Manhattan path took about 0.31 seconds.

In the first example (Figure 22), the task for the manipulator is to pick up a mock satellite (a) and place it in a desired configuration (h) which is tightly constrained by stick like obstacles that form a “cage”. There are 10 pre-defined grasps distributed around the periphery of the satellite (Figure 21). The manipulator first executes a transit path to grasp the satellite (b); followed by a transfer path to place the satellite in an intermediate placement (c); it then changes grasp by executing a transit path (d); with this new grasp, it moves the object – via a transfer path – to another placement (e); changes grasp – a transit path – again (f); using this new grasp, it is able to move the satellite to the desired final configuration (g). The same configuration is shown in (h) with the manipulator moved back to its initial configuration. Average run times for this example were around 4 minutes and 6 landmarks (at the manipulation level) were placed before finding a solution.

In the second example (Figure 23) – much more difficult than the first one – the task for the manipulator is to re-orient the satellite inside the cage. Both the initial and final configuration of the satellite are quite tightly constrained in the cage. Average run times for this example were around 15 minutes and 33 manipulation landmarks were placed before finding a solution. For both examples, the number of "sub-landmarks" placed by *ACA_TRANSIT* varied between 3 and 15; and the ones placed by ACA_TRANSFER varied between 8 and 23.

Considering the complexity of the problem, our results compare favourably with the run times reported in the literature for other types of motion planning problems [8, 13, 18]. Recall that no other manipulation planning approaches for redundant manipulators have been reported in the literature. It is indeed possible to construct examples where it may require a large amount of computation. However, as observed in [19], in typical practical problems, the solution space is also fairly large, so that the algorithm will typically find a solution much before it has to explore the entire space. Moreover, we have observed that typically it is easier to get out of a constrained region than getting into it. In Figure 22, if the initial and goal positions were swapped, the planner finds the solution much faster. Running the planner bi-directionally (simultaneously, perhaps in an interleaved manner) would be one strategy to solve this problem [8].

9 Extensions

Our approach can be made more efficient. The key property to be exploited is that for any configuration $\hat{p} \in P$ and $\hat{g} \in G$, the (free) sub-manifolds $\mathcal{C}_{\hat{p}}$ and $\mathcal{C}_{\hat{g}}$ are subsets of $\mathcal{C}_{\mathcal{A}free}$, the free configuration space of the manipulator. A representation of $\mathcal{C}_{\mathcal{A}free}$ can be constructed in advance (pre-processed) from a number of initial points by using $EXPLORE$. This representation is essentially a set of landmarks $\mathcal{L}_m{}'$ in the free space of $\mathcal{C}_{\mathcal{A}}$ and tree-structure (with the associated paths) that encodes the connectivity of the landmarks.

This is illustrated for a 2-*dof* manipulator in Figure 24. *EXPLORE* is used to construct $\mathcal{L}_m{}'$ until a given resolution ε. The associated set of paths is $\{\hat{\rho}_1, \hat{\rho}_2, ..., \hat{\rho}_{m-1}\}$ where $\hat{\rho}_i$ is the path used for generating the landmark L_{i+1} in $\mathcal{L}_m{}'$. This representation of $\mathcal{C}_{\mathcal{A}free}$ may then be used to plan transit and transfer paths as follows. The landmarks remaining in $\mathcal{C}_{\hat{p}}$ are used to plan transit paths and those remaining in $\mathcal{C}_{\hat{g}}$ are used to plan transfer paths (See Figure 25). Note that using $\hat{\rho}_i$, it is easy to verify if a given landmark is still reachable, since $\hat{\rho}_i$ is collision-free with respect to the static obstacles. In the case of transit paths, the validity of $\hat{\rho}_i$ in $\mathcal{C}_{\hat{p}}$ is easily checked by computing the collisions between the robot as it moves along $\hat{\rho}_i$, and $\mathcal{M}$ which remains static at $\hat{p}$ during the motion. Similarly for transfer paths, the validity of $\hat{\rho}_i$ in $\mathcal{C}_{\hat{g}}$ is verified by computing the collisions of only $\mathcal{M}$ (as it moves with the robot) and the static obstacles. If a given landmark $L_i \in \mathcal{L}_m{}'$ is not reachable, then its descendants may not be in the same connected component, i.e., $\mathcal{L}_m{}'$ is essentially transformed into a forest (Figure 25). *EXPLORE* can be generalized for this case.

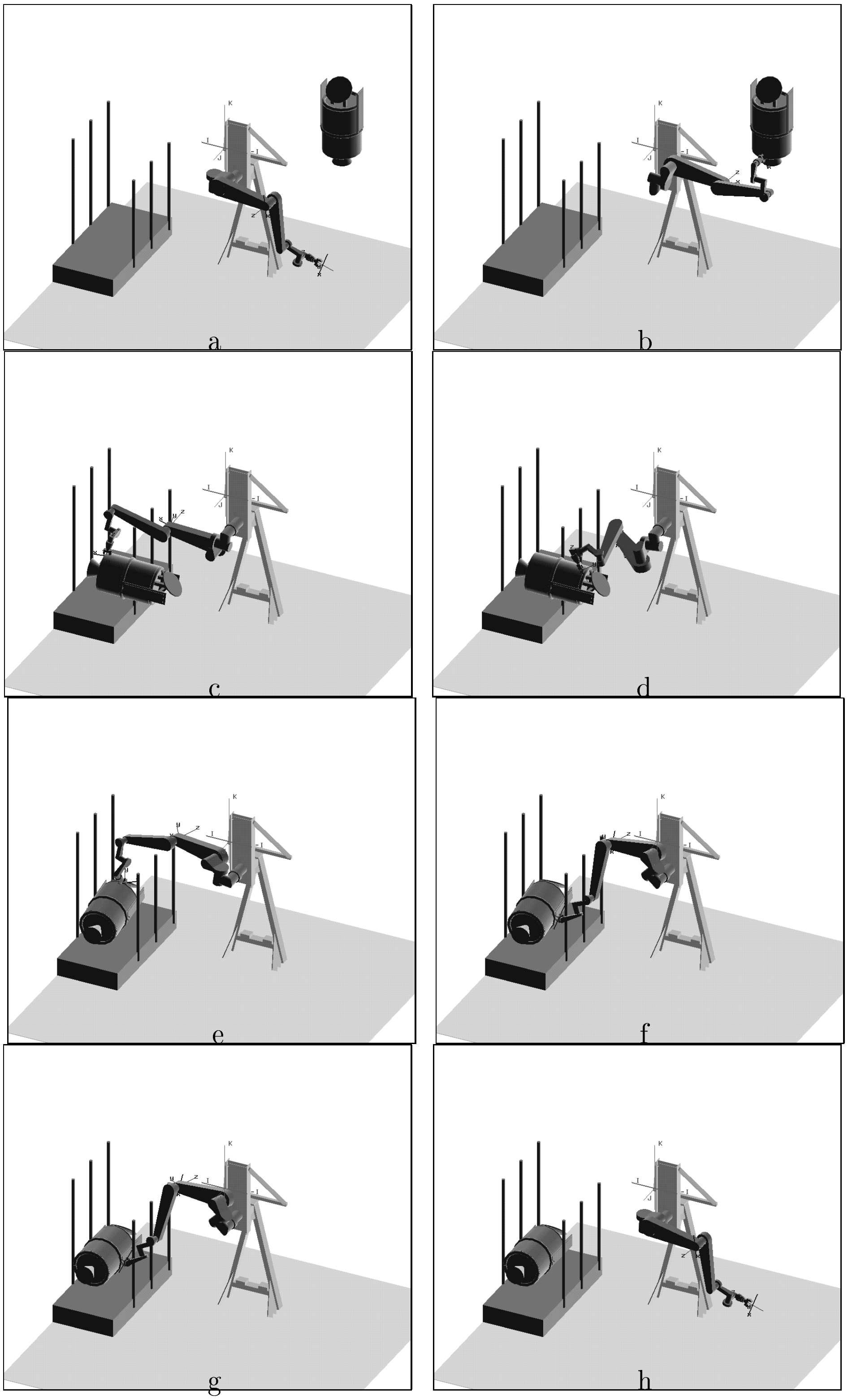


Figure 22: An example of a manipulation task for a 7-*dof* manipulator in 3-D environments.

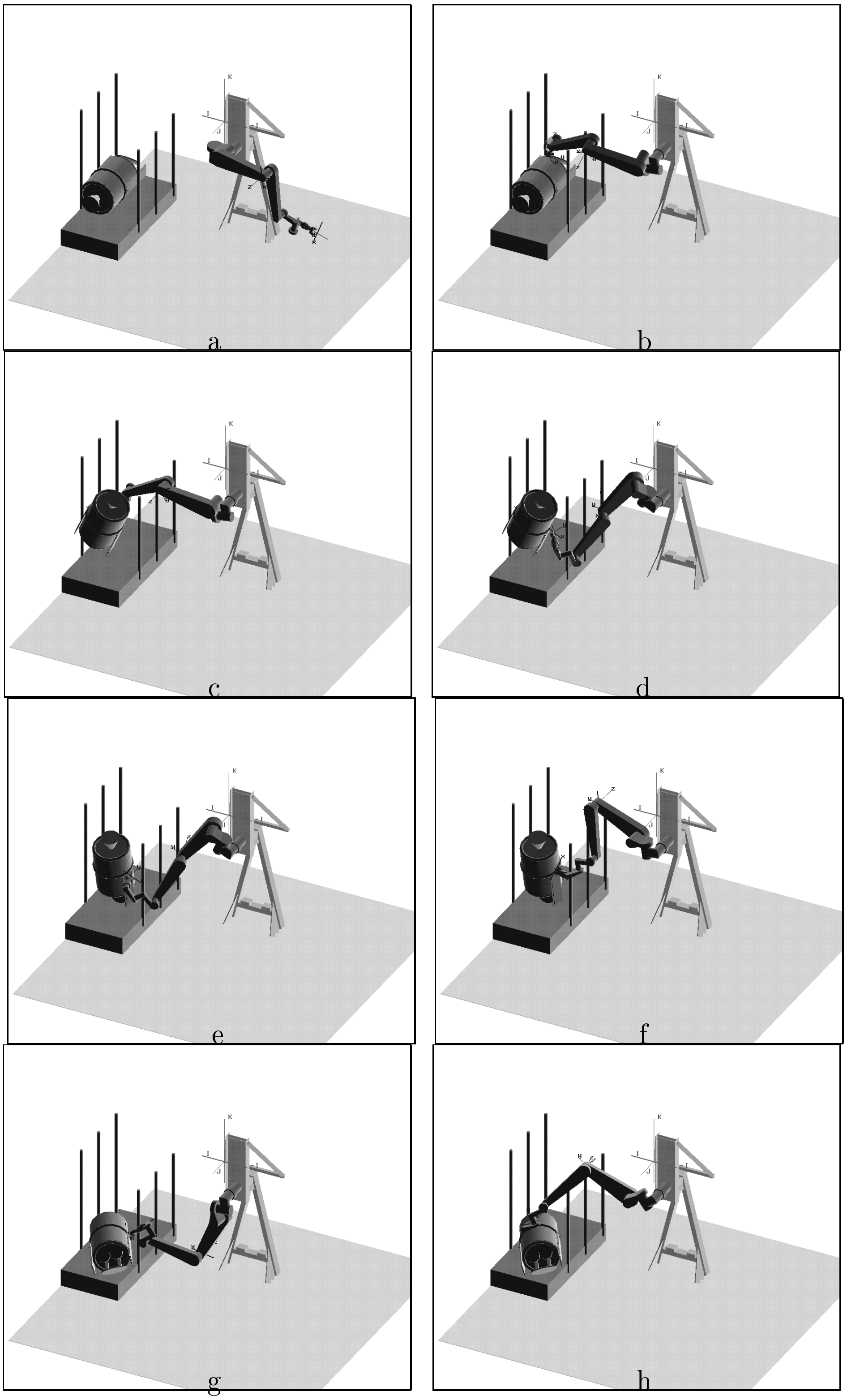


Figure 23: A difficult manipulation task: both the initial and the final configurations of the payload are tightly constrained by obstacles.

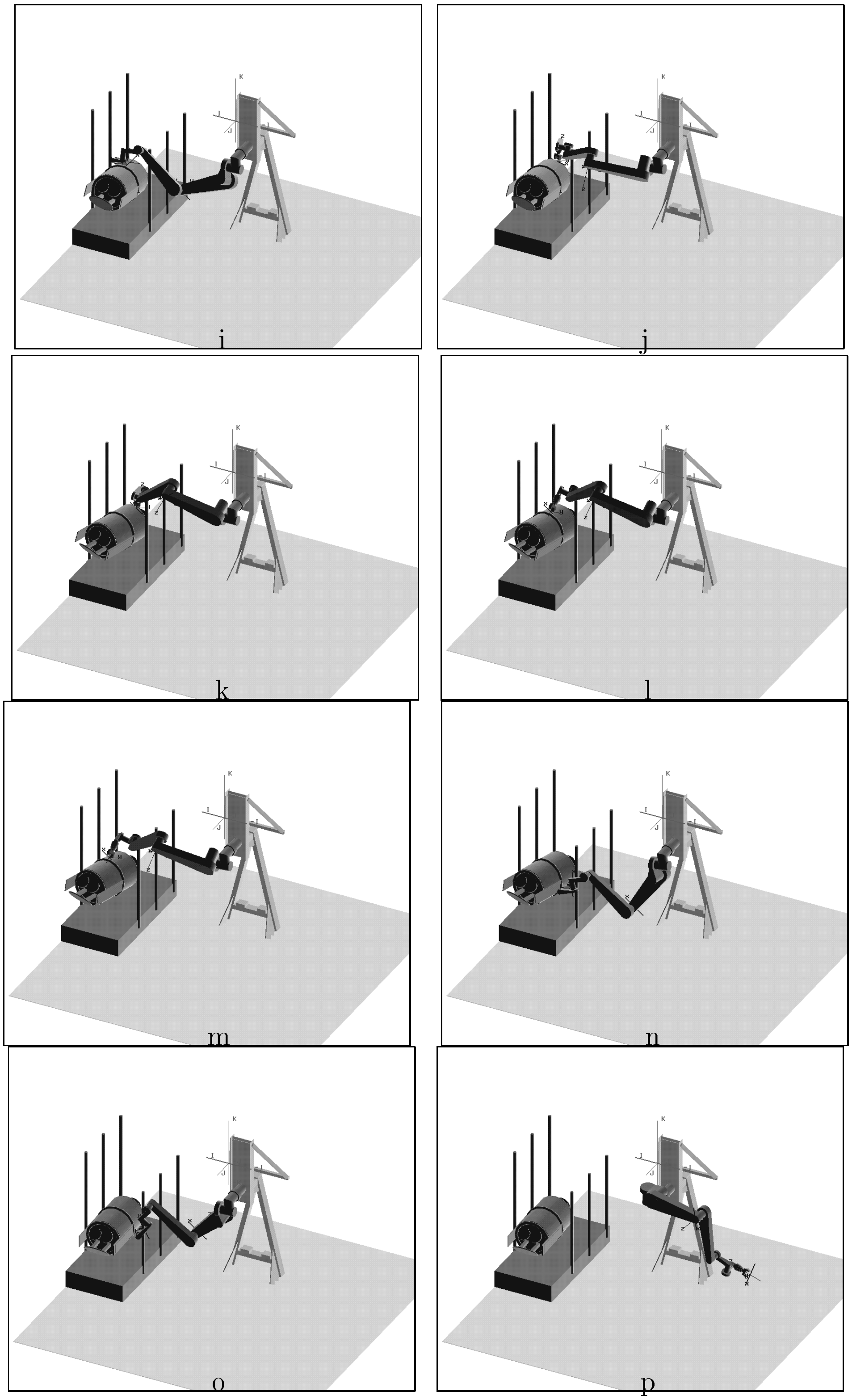


Figure 23: Continued from previous page.

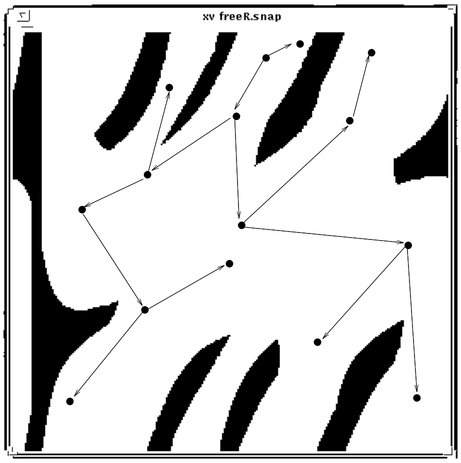


Figure 24: $\mathcal{C}_{\mathcal{A}free}$ for a 2-*dof* planar robot. $\bullet$ are the set of landmarks and arrows symbolize the corresponding $\hat{\rho}_i$. A subset of these landmarks may be used to generate more efficient planners for transit and transfer paths.

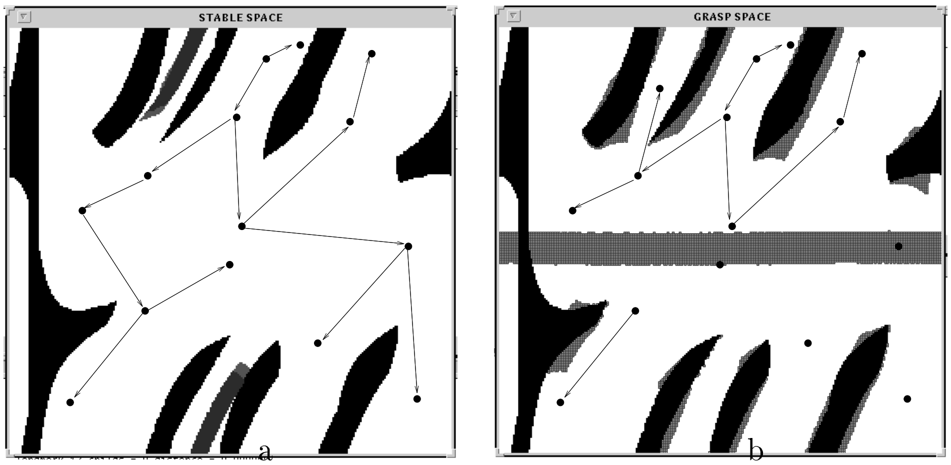


Figure 25: $\mathcal{C}_{\hat{p}}$ is shown in (a) and $\mathcal{C}_{\hat{g}}$ is shown in (b). Note that the free space in each is a subset of $\mathcal{C}_{\mathcal{A}free}$ shown in Figure 24. Some of the landmarks present in Figure 24 are eliminated. Also, the connectivity of the landmarks changes.

10 Conclusions

One of the emerging paradigms in solving motion planning problems is to capture the connectivity of the configuration space by using a finite (but possibly large) set of landmarks (or nodes) in it. In this paper, we extend this paradigm to manipulation planning problem. Our approach offers the following advantages over the previous methods for manipulation planning (i) it does not assume a finite number of robot configurations for each grasp, i.e., it is directly applicable to redundant manipulators, and (ii) it does not implicitly assume that an inverse kinematics algorithm for the manipulator is available.

We have implemented this approach for a 7-*dof* manipulator in 3-dimensional environments. In our preliminary experiments, run times for environments of moderate complexity are of the order of a few minutes to few tens of minutes (from 3 to 20). These results indicate that our approach is practical and could be applied in real industrial applications. There are several directions for future research. One would be to combine the manipulation planner developed here with a grasp planner for a multi-fingered hand (e.g., Salisbury Hand [26]). With more sophisticated paradigms for motion planning recently emerging, coupled with the increase in computational power of a standard workstation, such integrated systems would be of great potential application in several other disciplines ranging from computer animation to ergonomics [18].

References

[1] J.M. Ahuactzin, E. Mazer, P. Bessière, E.G. Talbi. *Using Genetic Algorithms for Robot Motion Planning*, European Conference on Artificial Intelligence, Vienna, Austria, August 1992.

[2] J.M. Ahuactzin. *Le Fil d'Ariane. Une méthode de planification générale. Application à la planification automatique des trajectoires*, Ph.D. Thesis. Institut National Polytechnique de Grenoble. Grenoble, France 1994.

[3] R. Alami, T. Siméon and J.P. Laumond. *A Geometrical Approach to Planning Manipulation Tasks : The Case of Discrete Placements and Grasps*, in Robotics Research 5, The MIT press, Cambridge, 1990.

[4] R. Alami, J.P. Laumond and T. Siméon. *Two manipulation planning algorithms*, in Algorithmics Foundations of Robotics, AK Peters, 1995.

[5] J. Barraquand, J.C. Latombe. *Robot Motion Planning: A Distributed Representation Approach*, International Journal of Robotics Research, 10(6), 1991, pp. 628-649.

[6] C. Bard. *Interaction sensori-motrice en robotique : Application a la prehension automatisée pour une main articulée a plusieurs doigts*, Ph.D. Thesis. Institut National Polytechnique de Grenoble. Grenoble, France 1994.

[7] R. Cerf. *Une Théorie Asymptotique des Algorithmes Génétiques* Thèse de l'Universiteé de Montpellier, Mars 1994

[8] C. Chen, Y. K. Hwang. *SANDROS: A Motion Planner with Performance Proportional to Task Difficulty*, Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France - May 1992.

[9] Y. Davidor. *Genetic Algorithms and Robotics. A Heuristic Strategy for Optimization*, World Scientific Series in Robotics and Automated System-Vol. 1. 1991.

[10] E. Falkenauer and S. Bouffouix, *A Genetic Algorithm for Job Shop*, Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, California-April 1991.

[11] E. Falkenauer, A. Delchambre. *A Genetic Algorithm for Bin Packing and Line Balancing* Proceedings of the IEEE International Conference on Robotics and Automation Nice, France - May 1992.

[12] K. K. Gupta, Z. Guo. *Motion Planning for Many Degrees of Freedom Sequential Search With Backtracking*, Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France - May 1992.

[13] K. K. Gupta and X. Zhu. *Practical Global Motion Planning for Many Degrees of Freedom: A Novel Approach Within Sequential Framework*, Journal of Robotic Systems, John Wiley & Sons, Inc., 1995.

[14] J.H. Holland. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[15] L. Kavraki and J.C. Latombe. *Randomized Preprocessing of Configuration Space for Path Planning : Articulated Robots*, IEEE/RSJ International Conference on Intelligent Robotics and Systems, Munich, Germany, 1994.

[16] L. Kavraki, P. Ŝvestka, J.C. Latombe and M. Overmars. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Space*, Stanford University, Technical Report 1995.

[17] Y. Koga and J.C. Latombe. *Experiments in Dual-Arm Manipulation Planning*, Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France - May 1992.

[18] Y. Koga. *On Computing Multi-Arm Manipulation Trajectories*, Ph.D. Thesis, Department of Mechanical Engineering, Stanford University, 1994.

[19] J.C. Latombe. *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[20] D. Lawrence (Editor). *Handbook of genetic algorithms* VNR Computer Library, Van nostrand Reinhold Pub. New York, 1991.

[21] T. Lozano-Pérez. *A Simple Motion-Planning Algorithm for General Robot Manipulators*, IEEE Transactions on Robotics and Automation, June 1987.

[22] T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O'Donnell. *HANDEY : A Robot Task Planner*, Massachusetts Institute of Technology, U.S.A. 1992.

[23] E. Mazer, J.M. Ahuactzin, E. Talbi, P. Bessière, T. Chatroux. *Parallel Motion Planning with The Ariadne's Clew Algorithm*, Third International Symposium on Experimental Robotics, Kyoto, Japan, October 1993.

[24] M. H. Overmars. *A Random Approach to Motion Planning*, Technical Report Utrecht University, Department of Computer Science, The Netherlands, 1992.

[25] N. S. Pollard and T. Lozano-Pérez. *Grasp Stability and Feasibility for an Arm with Articulated Hand*, Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, 1991.

[26] J. Salisbury. *Kinematic and Force Analysis of Articulated Hands*. Ph.D. Thesis, Department of Mechanical Engineering, Stanford University, 1982.

[27] P. Tournassoud, T. Lozano-Pérez and E. Mazer. *Regrasping*, Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh, NC, March 1987.