Robot Motion Planning: A Distributed Representation Approach

Article *in* The International Journal of Robotics Research · December 1991

DOI: 10.1177/027836499101000604

2 authors, including:

Jerome Barraquand
Thinking Engines
22 PUBLICATIONS **3,481** CITATIONS

SEE PROFILE

Robot Motion Planning: A Distributed Representation Approach

Jérôme Barraquand and Jean-Claude Latombe

The International Journal of Robotics Research 1991 10: 628
DOI: 10.1177/027836499101000604

Robot Motion Planning: A Distributed Representation Approach

Jérôme Barraquand
Jean-Claude Latombe
Robotics Laboratory
Department of Computer Science
Stanford University
Stanford, California 94305

Abstract

We propose a new approach to robot path planning that consists of building and searching a graph connecting the local minima of a potential function defined over the robot's configuration space. A planner based on this approach has been implemented. This planner is considerably faster than previous path planners and solves problems for robots with many more degrees of freedom (DOFs). The power of the planner derives both from the "good" properties of the potential function and from the efficiency of the techniques used to escape the local minima of this function. The most powerful of these techniques is a Monte Carlo technique that escapes local minima by executing Brownian motions. The overall approach is made possible by the systematic use of distributed representations (bitmaps) for the robot's work space and configuration space. We have experimented with the planner using several computer-simulated robots, including rigid objects with 3 DOFs (in 2D work space) and 6 DOFs (in 3D work space) and manipulator arms with 8, 10, and 31 DOFs (in 2D and 3D work spaces). Some of the most significant experiments are reported in this article.

1. Introduction

In this article we propose a new approach to robot path planning that is based on the systematic use of a hierarchical bitmap to represent the robot work space. This "distributed" representation, which strongly differs from the "centralized" semialgebraic representations used in most path-planning algorithms so far, makes it possible to define simple and powerful numeric potential field techniques. Using such techniques, we implemented a planner that is considerably faster than previous path planners and solves problems for robots with many more degrees of freedom (DOFs).

The principle of our approach is to construct *collision-avoiding attractive potential fields* over the work space. Each of these potentials applies to a selected point in the robot, called a *control point*, and pulls this point toward its goal position among the obstacles. The work space potentials are then combined into another potential function defined over the configuration space of the robot. This new potential attracts the whole robot toward its goal configuration.

Each work space potential is computed over the bitmap representation of the work space (at some resolution) in such a way that it has no other local minimum than the desired final position of the robot's control point it applies to. Therefore it can be regarded as a numeric navigation function as introduced in Koditschek (1987). Such a "perfect" potential tends to keep the robot outside the work space concavities created by the obstacles. When the work space potentials are combined together (i.e., when they are applied concurrently to the various control points), the resulting configuration space potential may have (and indeed has) local minima other than the goal. However, it is usually possible to define the combination in such a way that either the number of minima or their domains of attraction remain relatively small. The idea is then to build a graph connecting the local minima and to perform a search of this graph until the goal is attained. If the goal cannot be achieved, the planner repeats the whole process at a finer level of resolution in the work space pyramid. It stops when either a path has been found (success) or the maximal resolution has been attained (failure).

In order to build the local minima graph, our approach requires efficient techniques for escaping

Barraquand is now with the Paris Research Laboratory of Digital Equipment Corporation (DEC), 85 Avenue Victor Hugo, 92563 Rueil-Malmaison Cedex, France.

The International Journal of Robotics Research,
Vol. 10, No. 6, December 1991,

the local minima. In this article we describe two such techniques. One is a brute force technique that exhaustively explores the local minimum wells in the discretized configuration space. The other is a Monte Carlo technique that escapes local minima by executing Brownian motions. The second technique turns out to be very powerful and has solved tricky path-planning problems for robots with many DOFs. Furthermore, it is highly parallelizable. However, for robots with few degrees of freedom, the first technique is faster on a sequential computer. In addition, it is *deterministically resolution complete*, whereas the Monte Carlo technique is only *probabilistically resolution complete*.

We have implemented the planner in a program written in C language and run on a DEC 3100 MIPS-based workstation. (All the running times given in this article refer to this implementation.) We have experimented with the implemented planner using several computer-simulated robots, including rigid objects ("mobile robots") with 3 DOFs (in two-dimensional work spaces) and 6 DOFs (in three-dimensional work spaces) and articulated objects ("manipulator arms") with 8, 10, and 31 DOFs (in two- and three-dimensional work spaces). Some of the most significant experiments are reported in this article. Our planner demonstrated the following capabilities:

- *It is much faster than any previous planner.* For instance, it generates paths for a holonomic 3-DOF mobile robot in nontrivial work spaces in about 1 s of computation, as opposed to minutes or even tens of minutes for many other planners.
- *It generates paths for robots with many DOFs.* In particular, within a few minutes of computation, it constructs complex paths for a 10-DOF non-serial manipulator arm with both revolute and prismatic joints.
- *It solves path-planning problems for multiple robots.* For example, without domain-specific heuristics, it can generate the coordinated paths of two 3-DOF mobile robots in a work space made of narrow corridors.

In addition, the algorithms are highly parallelizable. This allows us to envision an implementation of the planner using specific hardware for generating paths in real time, even for robots with many DOFs. Another advantage of the planner is that it accepts goals defined by specifying the desired positions of one or several points in the robot. This feature is essential when robots have many DOFs, as specifying the goal configuration of the robot (i.e., a collision-free placement of the various bodies of the robot) is a difficult task in itself. It also allows the easy handling of any kind of redundancy of the robot. Finally, a version of the planner (not described in this article) generates paths for nonholonomic robots—i.e., robots with nonintegrable kinematic constraints such as a car and a car towing a trailer; this version of the planner is described in Barraquand and Latombe (1989a; 1990).

This article is organized as follows. In section 2 we relate our work to previous research. In section 3 we describe the distributed representations of the work space and the configuration space and propose various techniques for computing the work space and configuration space potentials. In section 4 we present a simplified version of the planner that is applicable to robots with few DOFs (four or less). In section 5 we show how the use of Brownian motions for escaping local minima allows us to extend the planner and solve path-planning problems with robots having many DOFs.

2. Relation to Other Work

Much research has been devoted to robot motion planning during the past 10 years (Latombe 1990). Most of this research has focused on path planning (i.e., the geometric problem of finding a collision-free path between two given configurations of a robot). Today the mathematic and computational structures of the general problem (when stated in algebraic terms) are reasonably well understood (Schwartz and Sharir 1983b; Canny 1988). In addition, practical algorithms have been implemented in more or less specific cases (Brooks and Lozano-Pérez 1983; Gouzènes 1984; Laugier and Germain 1985; Faverjon 1986; Lozano-Pérez 1987; Faverjon and Tournassoud 1987; Barraquand et al. 1989; Zhu and Latombe 1989).

One of the most widely studied path-planning approaches is the "cell decomposition" approach (Schwartz and Sharir 1983a; Brooks and Lozano-Pérez 1983). It consists of first decomposing (exactly or approximately) the set of free configurations of the robot into a finite collection of cells and then searching a connectivity graph representing adjacency relation among these cells. However, in this approach, the number of cells to be generated is a function of (1) the number of polynomial constraints used to model the robot and the obstacles and (2) the degree of these constraints. This function also grows exponentially with the number of DOFs (n) of the robot, as the volume of the configuration space (locally diffeomorphic to $\mathbf{R}^n$) increases exponentially

with n. Thus the approach is intractable even for reasonably small values of n. To our knowledge, no effective planner has been implemented using this approach with $n > 4$. In fact, this is also true of the other so-called "global" methods—e.g., retraction (Ó'Dúnlaing et al. 1983)—which also represent the connectivity of free space in the form of a graph before actually starting the search for a path.

Some approximate cell decomposition methods proceed hierarchically by decomposing the configuration space into rectangloid cells organized at several levels of resolution. For example, Faverjon (1984) uses an "octree" to represent a three-dimensional configuration space. Each cell is labeled as EMPTY if it intersects no configuration space obstacle, FULL if it lies entirely in configuration space obstacles, and MIXED otherwise. Although there may be a loose resemblance between such a tree and the hierarchical bitmap representations used in our planner, the two approaches are very different. In particular, our planner does not attempt to explicitly approximate the configuration space obstacles as collections of cells.

Because the intractability of the cell decomposition approach—and more generally of the other global path-planning methods—is caused in part by the precomputation of a connectivity graph representing the "global" topology of the robot's free space, "local" methods to path planning have been developed for handling more DOFs, and some successful systems have been implemented (Donald 1984; Faverjon and Tournassoud 1987). A local path-planning method consists of placing a grid (at some resolution) over the robot configuration space and searching this grid. Heuristics computed from partial information about the geometry of the configuration space are used to guide the search. Thus a local method requires no expensive precomputation step before starting the search of a path. In favorable cases, it runs substantially faster than any global method. However, because the search graph (i.e., the grid) is considerably larger than the connectivity graph searched by global methods, it may require much more time than global methods in less favorable cases.

In order to deal efficiently with the large size of the grid, local methods need powerful heuristics to guide the search. However, such known heuristics have the drawback of eventually leading the search to dead-ends from which it is difficult to escape. For example, a widely used heuristics consists of guiding the robot along the negated gradient of an artificial potential field (Khatib 1986). However, this technique may get stuck at local minima of the potential and provides no systematic way to escape these minima. The problem of defining an analytic "navigation function" (i.e., a potential field with a unique minimum at the goal configuration in the connected component of the free space containing the goal configuration of the robot) has been investigated with only limited success so far. Solutions have been proposed only in Euclidean configuration spaces when all the configuration space obstacles are spherical or star-shaped objects (Rimon and Koditschek 1989). Furthermore, if such a navigation function could be defined in the general case, its computation would probably be expensive and would constitute a precomputation step before search, similar in drawback to the construction of the connectivity graph in the cell decomposition approach. Along another line of research, paths for an 8-DOF manipulator have been generated with a variant of the potential field method, called the *constraint method* (Faverjon and Tournassoud 1987). Although impressive, this result has been obtained in specific work spaces where all the obstacles are vertical cylindrical pipes, with interactive human assistance for moving the robot out of the encountered local minima.

Recently, we have developed a potential-based approach using a numeric valley-tracking algorithm (Barraquand et al. 1989) to escape the local minima of the potential function. The planner was able to generate paths for a 10-DOF manipulator arm with a nonserial kinematic chain. (An example with the same arm will be given in this article.) However, the planner implemented using this approach was quite slow and not very reliable. In particular, it failed to solve several problems that the planner described in the present article has been able to solve. Nevertheless, the approach described later derives from this earlier work.

The problem of generating paths for multiple robots has attracted some attention (Schwartz and Sharir 1983c; Kant and Zucker 1986; Erdmann and Lozano-Pérez 1986; O'Donnell and Lozano-Pérez 1989). Implemented systems rely on a simple paradigm with multiple variants—e.g., "velocity tuning" and "prioritizing"—which allows one to consider the individual robots separately or sequentially. This paradigm makes it possible to build planners whose time complexity is "only" exponential in the maximum of the numbers of DOFs of the individual robots, rather than in their sum. However, the paradigm is incomplete and is unable to solve problems in which robots "strongly" interact. An example of such a problem is when two mobile robots have to

interchange their positions in a work space made of narrow corridors. We successfully run our planner on several examples of this kind.

Since Reif's early paper (Reif 1979), the computational complexity of path planning has been analyzed by many authors when the problem is stated in semi-algebraic form (Schwartz and Sharir 1988). The bitmap representations used in our planner may open new perspectives on some computational complexity aspects. The worst-case complexity of our planner is still exponential in the number of DOFs (i.e., the dimension of the configuration space). However, although planners using semi-algebraic representations are time polynomial in the number of polynomial constraints and their maximal degrees, our planner is polynomial in the inverse of the maximal resolution of the bitmap description.[1] One may argue that, unlike semi-algebraic models, the bitmap representation is not "exact." However, when compared with the real world, neither of these representations is exact, and both can be made as precise as one wishes by increasing the resolution of the bitmap, for one representation, and the number and degrees of the semi-algebraic constraints, for the other representation.

3. Distributed Representation and Potential Fields

3.1. Work Space Representation

Let $\mathcal{A}$ denote the robot, $\mathcal{W}$ its work space, and $\mathcal{C}$ its configuration space. A configuration of the robot (i.e., a point in $\mathcal{C}$) completely specifies the position of every point in $\mathcal{A}$ with respect to a coordinate system attached to $\mathcal{W}$ (Lozano-Pérez 1983). Let n be the dimension of $\mathcal{C}$. We represent a configuration $q \in \mathcal{C}$ by a list of n parameters $(q_1, \ldots, q_n)$, with appropriate modulo arithmetic for the angular parameters (Latombe 1990). The subset of $\mathcal{C}$ consisting of all the configurations where the robot has no contact or intersection with the obstacles in $\mathcal{W}$ is called the *free space* and is denoted by $\mathcal{C}_{free}$.

The work space $\mathcal{W}$ is modeled as a multiscale pyramid of bitmap arrays, each of which is N-dimensional, with $N = 2$ or 3 being the dimension of $\mathcal{W}$. At any given resolution level, the array is defined by the following function BM:

$$BM : \mathcal{W} \rightarrow \{1, 0\}$$
$$x \mapsto BM(x)$$

in such a way that the subset of points x such that $BM(x) = 1$ represents the work space obstacles, and the subset of points x such that $BM(x) = 0$ represents the empty part of the work space. We write: $\mathcal{W}_{empty} = \{x \mid BM(x) = 0\}$. Figure 1 displays the bitmap representation of a particular two-dimensional work space at the 256^2 resolution (1 = black; 0 = white).

For each point $p \in \mathcal{A}$, one can consider the geometric application that maps any configuration $q = (q_1, \ldots, q_n) \in \mathcal{C}$ to the position $x \in \mathcal{W}$ of p in the work space. This map:

$$X : \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{W}$$
$$(p, q) \mapsto X(p, q) = x$$

is called the *forward kinematic map*.

The work space representation is given to the planner at the finest level of resolution, typically 512^2 or 256^2 for a two-dimensional work space and 128^3 for a three-dimensional work space. The other levels are automatically derived from it in a conservative fashion. The scaling factor between two successive levels of resolution is 2, but a different factor could have been chosen.

The planner iteratively considers each level of resolution in the pyramid, from the coarsest to the finest, until it generates a path or exits with failure. At each level of resolution, it computes the various potential functions in the same fashion, using the bitmap array at the current level of resolution. We now describe these computations.

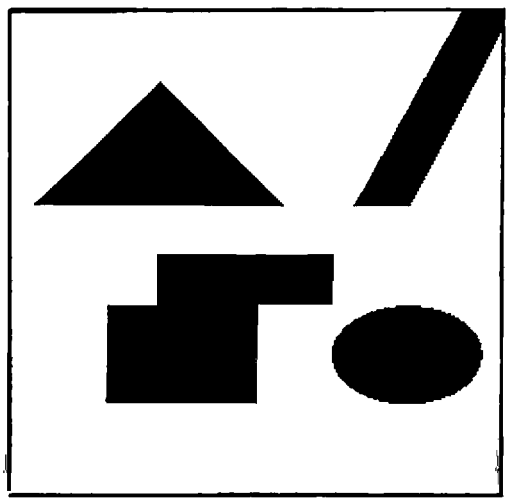

Fig. 1. 256^2 bitmap representation of a work space.

1. Different complexity measures with nonalgebraic models were previously given in Lumelsky (1987).

3.2. *Extraction of the Work Space Skeleton*

Let $p_1, \ldots, p_s$ be s given points in $\mathcal{A}$, called *control points*. For each point p_i we construct a function:

$$V_{p_i} : x \in \mathcal{W}_{empty} \mapsto V_{p_i}(x) \in \mathbf{R}$$

called the *work space potential*. Next, we combine these potentials into another potential function U defined over the configuration space:

$$U : q \in \mathcal{C}_{free} \mapsto U(q)$$

$$= G(V_{p_1}(X(p_1, q)), \ldots, V_{p_s}(X(p_s, q))) \in \mathbf{R}.$$

U is called the *configuration space potential*. The construction of the V_{p_i}s is described in this subsection and the next. The construction of U is described in section 3.5.

In constructing the work space potentials V_{p_i}, we have two goals:

1. We want each function V_{p_i} to have a single minimum at the goal position of the point p_i. This is a major heuristic step toward the construction of a configuration space potential with few or small spurious local minima.
2. We want the path obtained by following the negated gradient of V_{p_i}, from any initial position of p_i, to lie as far away as possible from the obstacles in order to maximize the maneuvering space of the robot.

To achieve these goals, we compute each work space potential in two steps. First, we compute the discrete L^1 (Manhattan) distance $d_1(x)$ from every point $x \in \mathcal{W}_{empty}$ to the obstacles and we simultaneously extract a subset $\mathcal{S}$ of $\mathcal{W}$ of co-dimension 1. We call this subset the *work space skeleton*. In two dimensions, it is a network of one-dimensional curves similar to the skeleton widely used in mathematic morphology (Serra 1982) and to the generalized Voronoi diagram (Lee and Drysdale 1981). Second, we compute the potential functions V_{p_i} using both the d_1 map and the skeleton $\mathcal{S}$. The first step is detailed later in this section. The second is described in the next subsection.

The distance $d_1(x)$ between every point $x \in \mathcal{W}_{empty}$ and the obstacles is computed as follows: First, the points in the boundary of the obstacles are identified, and the value of d_1 at these points is set to zero (we also include the points in the frame bounding the bitmap as boundary points). Then, starting from these boundary points, we apply a wavefront expansion procedure that recursively labels all the points in $\mathcal{W}_{empty}$. More precisely, the values of d_1 at all the neighbors of the boundary points are set to 1; the value of d_1 at the neighbors of these points, if not yet computed, is set to 2; etc. The procedure is repeated until all the points in $\mathcal{W}_{empty}$ have been attained. Figure 2 displays contours of the d_1 map thus computed for the work space shown in Figure 1.

In parallel, we compute the work space skeleton $\mathcal{S}$ as the set of points where the "waves" issued from the boundary points of $\mathcal{W}_{empty}$ meet. This is done by propagating not only the values of d_1, but also the points in the boundary of $\mathcal{W}_{empty}$ that are at the origin of the propagation. Figure 3 displays the skeletons computed in several work spaces, including the work space of Figure 1. Every connected component of the work space yields a connected component of the skeleton $\mathcal{S}$.

The computation of both d_1 and $\mathcal{S}$ is not local and therefore must be done prior to the execution of the rest of our path-planning algorithm. However, the time complexity of the algorithm is linear in the number of points in $\mathcal{W}$ and constant in the number and the shape of obstacles. Its implementation is quite fast (a fraction of a second for a 256^2 bitmap array).

From a conceptual point of view, neither the choice of the L^1 metric nor the precise definition of the skeleton is very important for the rest of our path-planning approach, as the potential functions are only used as heuristics. Instead, we could have used the more classic L^2 distance and computed the generalized Voronoi diagram for that metric. However, as mentioned earlier, the construction of the work space potentials over $\mathcal{W}_{empty}$ is a necessary preliminary step before the rest of our path-planning

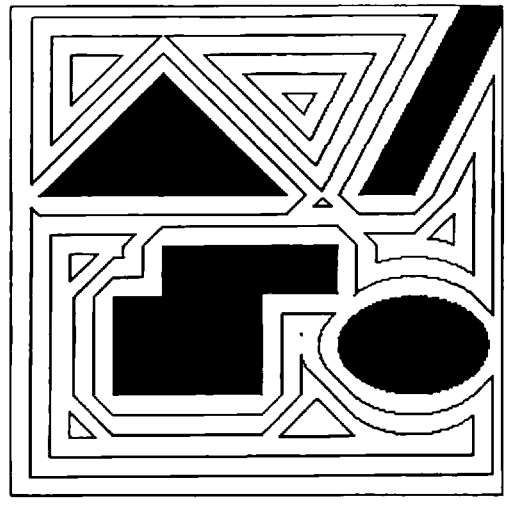

Fig. 2. Contours of the d_1 *map in the work space of Fig. 1.*

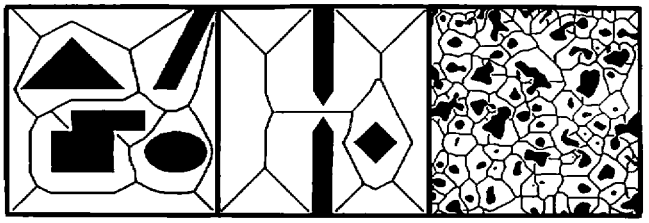

Fig. 3. Examples of work space skeletons.

algorithm can be executed. The computation of the L^1 distance is faster than the computation of the L^2 distance. Notice also that unlike the earlier computation of $\mathcal{S}$, the time complexity of constructing the algebraic description of the L^2 Voronoi diagram of a polygonal work space increases with the number of vertices of the obstacles.

3.3. *Work Space Potential Fields Without Local Minima*

The bitmap description of the work space allows us to compute a numeric navigation function—i.e., a collision-avoiding attractive potential field defined over the work space bitmap that has no other local minima than the goal. Such a navigation function happens to be very helpful for avoiding concavities of the work space obstacles.

We propose two algorithms, NF1 and NF2, for computing numeric navigation functions. NF1 is simpler and does not make use of the map d_1 and the skeleton $\mathcal{S}$. NF2 is slightly more involved but has the advantage of producing work space potentials that keep the control points as far away as possible from the obstacles.

NF1 computes the work space potential V_p for every control point p by using a wavefront expansion technique starting at the goal position x_{goal} of p. The value of V_p is first set to 0 at x_{goal}. Next, the value of V_p at the neighbors of x_{goal} in $\mathcal{W}_{empty}$ is set to 1, the value of V_p at the neighbors of these neighbors in $\mathcal{W}_{empty}$ is set to 2 (if not previously computed), etc. This procedure is recursively repeated until the connected subset of $\mathcal{W}_{empty}$ containing x_{goal} has been completely explored. The complexity of NF1 is linear in the number of points of the bitmap description and constant in the number and shape of the obstacles. Equipotential contours of the resulting work space potential field for the two-dimensional work space of Figure 1 are displayed in Figure 4. This computation was performed in a fraction of a second.

A property of the function V_p thus computed is that by following the flow of the negated gradient

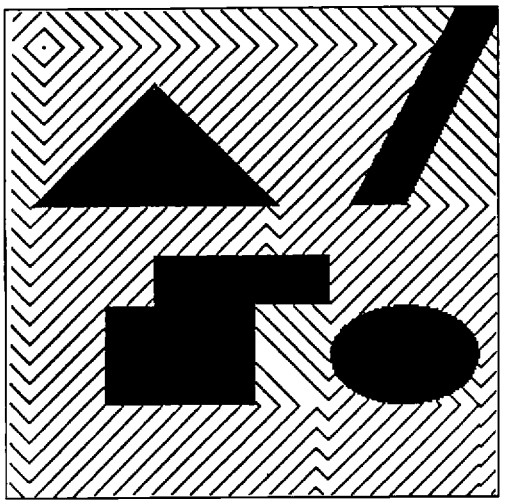

Fig. 4. Equipotential contours of the work space potential computed by NF1.

from any initial point x_{init}, we obtain a path between x_{init} to x_{goal} (if one exists) that is the shortest one for the L^1 distance (at the resolution of the bitmap array) over all the paths connecting x_{init} to x_{goal}. In a three-dimensional work space, this computation may be preferable to exact methods, as the problem of computing the exact shortest distance in a polyhedral space is known to be NP-hard in the number of vertices under any L^p metric (Canny 1988).

Notice that NF1 computes V_p only in the connected subset of $\mathcal{W}_{empty}$ that contains the goal position x_{goal}. Hence if the initial position x_{init} is not in the same connected component as the goal, the value of V_p is not computed at this point, and we can immediately infer that there is no collision-free path for the robot.

However, a significant drawback of this work space potential is that it induces paths that typically graze the obstacles in the work space (i.e., it only achieves the first of the two goals stated in section 3.2). Because several potentials will have to be combined into a configuration space potential U attracting the robot toward a goal configuration, the individual work space potentials may compete in such a way that they produce local minima of U. To reduce the risk of such a competition and to enlarge the maneuvering space of the robot, our planner makes use of a more involved work space potential function V_p computed by the algorithm NF2.

NF2 computes V_p in three steps:

1. The first step consists of tracing a line σ following the gradient of the distance map d_1 from the goal point x_{goal} to the nearest point in the

skeleton $\mathcal{S}$. Once this line is computed, we include it in $\mathcal{S}$, yielding the "extended skeleton" $\mathcal{S}_p = \mathcal{S} \cup \sigma$.

2. The second step of the algorithm consists of labeling all the points of $\mathcal{S}_p$ starting from x_{goal}. The label 0 is assigned to x_{goal}, the label 1 is assigned to its neighbors in $\mathcal{S}_p$, and these neighbors are inserted into a list Q. The point x_1 in Q that is at maximum distance d_1 from the obstacles is considered next and removed from Q; the label $l(x_1) = U(x_1) + 1$ is assigned to its neighbors in $\mathcal{S}_p$ that have not been labeled yet, and these neighbors are inserted into Q. This operation is repeated until the list Q is empty (i.e., the subset of $\mathcal{S}_p$ accessible from x_{goal} has been completely explored). At each step of the recursion, the list of points in Q is stored in a balanced tree sorted according to d_1 so that each insertion of a new point and extraction of a point at maximum distance from the obstacles takes logarithmic time in the size of Q (Aho et al. 1983). At the end of the second step, all the points $x \in \mathcal{S}_p$ connected to x_{goal} in $\mathcal{S}_p$ have a label $l(x)$.
3. The third step is a wavefront expansion starting from the subset S of $\mathcal{S}_p$ labeled at the previous step. NF2 first gives the label $l(x) + 1$ to every neighbor of every point $x \in S$. It then computes the unlabeled neighbors of these neighbors and increments the labels iteratively by 1 until the connected component of $\mathcal{W}_{empty}$ containing x_{goal} has been completely explored.

Figure 5 shows equipotential contours of the work space potential computed by NF2 for the work space of Figure 1. This potential has no other local minima than the goal. Following its negated gradient from an initial position leads to generating a path that lies as far away from the obstacles as possible by following the safest portion on the work space skeleton. Like NF1, NF2 only computes V_p over the connected subset of $\mathcal{W}_{empty}$ that contains x_{goal}.

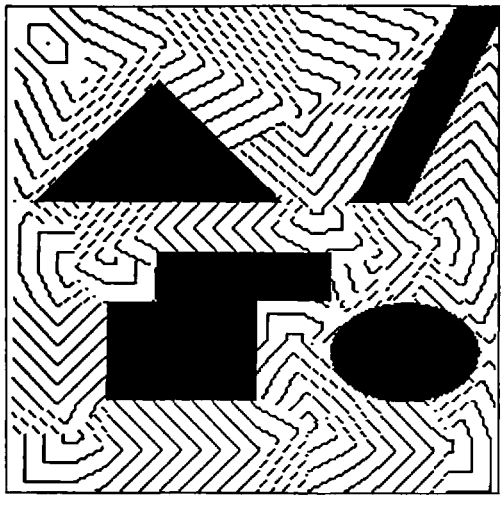

Fig. 5. Equipotential contours of the work space potential computed by NF2.

The complexity of NF2 is slightly higher than that of NF1. Let a be the number of points in the bitmap array and b the number of points in the augmented skeleton $\mathcal{S}_p$. The complexity of NF2 (including the cost of computing d_1 and the skeleton) is $O(a + b \log b)$, instead of $O(a)$. For reasonable work spaces, however, because the skeleton has co-dimension 1 in the work space, we have

$$b \propto a^{N-1/N},$$

with $N = 2$ or 3 being the dimension of the work space. For these work spaces, the complexity of NF2 reduces to

$$O(a + a^{N-1/N} \log a)$$

and hence is linear in a. For the 256^2 work space of Figure 1, it took about 2s for our implementation of NF2 to compute the work space potential shown in Figure 5. This time includes the computation of d_1 and $\mathcal{S}$, which does not have to be repeated if several work space potentials are computed.

Variants of NF2 can easily be imagined to compute work space potentials with slightly different properties. For example, in the third step, we could increment the potential at each iteration by $1/d_1$, rather than by 1, and obtain a potential V_p that becomes infinite in the boundary of $\mathcal{W}_{empty}$. We will not detail the cosmetics of the computation of numeric potential fields further. Our point is simply to show that a large family of potential functions with various properties can be built within our distributed representation approach.

3.4. *Discretization of the Configuration Space*

Because we represent the work space as a pyramid of bitmap arrays, it is consistent to discretize the configuration space $\mathcal{C}$ into a multiresolution grid pyramid. The configuration space pyramid has as many levels of resolution as the work space pyramid, and the resolutions at each level of the two pyramids are tightly related, as described later.

Let δ denote the distance between two adjacent points along the same coordinate axis in a work space bitmap. In the work space pyramid, δ varies between δ_{min} and δ_{max}. For example, if the work

space is represented by a pyramid of arrays whose sizes are ranging between 16^2 and 512^2 and if distances are measured in percentage of the work space diameter ("normalized" distances), we have $\delta_{min} = 1/512$ and $\delta_{max} = 1/16$.

We can define the resolution of a grid as the logarithm of the inverse of the distance between two discretization points in the base defined by the scaling factor between two successive resolution levels (2 in our implementation). Hence, in our example, the resolution r varies between $r_{min} = -\log_2(\delta_{max}) = 4$ and $r_{max} = -\log_2(\delta_{min}) = 9$.

We represent the configuration space $\mathcal{C}$ as a rectangloid subset of the n-dimensional Cartesian space $\mathbf{R}^n$ by representing a configuration q as a list $(q_1, \ldots, q_n)$ of n independent parameters. For any given work space resolution, say $r = -\log_2(\delta)$, the corresponding resolution $R_i = -\log_2(\Delta_i)$ of the discretization along the q_i axis of $\mathcal{C}$ is chosen in such a way that a modification of q_i by $\Delta_i = 2^{-R_i}$ generates a "small motion" of the robot in the work space. By "small motion" we mean that any point $p \in \mathcal{A}$ moves by less than $nbtol \times \delta$, where $nbtol$ is a small number (typically, 1 or 2).

The relation between the position of a robot point in the work space and a robot configuration is given by the forward kinematic map $X(p, q)$. For every point $p \in \mathcal{A}$, a modification of q_i ($i \in [1, n]$) by Δ_i results in a modification of each coordinate x_j ($j \in [1, N]$) of p by:

$$\frac{\partial x_j}{\partial q_i}(p, q)\Delta_i.$$

If we impose the work space motions to be less than $nbtol \times \delta$, we must have:

$$\Delta_i = nbtol \times \delta / \sup_{p \in \mathcal{A}, q \in \mathcal{C}, j \in [1,N]} \left(\frac{\partial x_j}{\partial q_i}(p, q) \right)$$
$$= nbtol \times \delta / J^i_{\text{sup}}.$$

For a given robot, the numbers J^i_{sup} are generally straightforward to compute. This leads us to compute the resolution R_i as:

$$R_i = r + \log_2(J^i_{\text{sup}}) + \log_2(nbtol).$$

For example, consider a bar of length L moving freely in a two-dimensional work space. A configuration of the bar can be represented as (x_G, y_G, θ), with x_G and y_G being the coordinates of the center of mass of the bar in the fixed Cartesian coordinate system embedded in the work space and θ being the orientation of the bar relative to an axis of this system. Let us normalize x_G, y_G, and θ so that their values range between 0 and 1. We have:

$$J^{x_G}_{\text{sup}} = J^{y_G}_{\text{sup}} = 1 \quad \text{and} \quad J^{\theta}_{\text{sup}} = \pi L/2.$$

If we set $nbtol$ to 2, we get:

$$R_{x_G} = R_{y_G} = r - 1 \quad \text{and}$$
$$R_\theta = r + \log_2(\pi L/2) - 1.$$

This means that we need $2^1 = 2$ times less samples for x_G and y_G than for the work space representation at each level of resolution and $1/(\pi L)$ less samples for θ. In our implementation, the J^i_{sup}s are input by hand.

3.5. *Configuration Space Potential*

In our planner, the goal configurations of a robot are specified by the goal positions of one or several points in the robot. By definition, the robot is at a goal configuration whenever all the control points are at their goal positions. For instance, if $\mathcal{A}$ is a two-dimensional object that can both translate and rotate in the plane (three-dimensional configuration space), the specification of the goal positions of two points uniquely determines the goal configuration of the robot. If $\mathcal{A}$ is, say, a 10-DOF manipulator arm, then specifying the desired positions of some points in the end effector determines a goal region in configuration space.

It is important that a path planner allows specification of a goal region in configuration space. Indeed, for many tasks, the goal configuration is incompletely specified. Arbitrarily selecting one would possibly result in a more difficult or impossible path-planning problem. Furthermore, if the robot has many degrees of freedom, specifying a unique goal configuration is a difficult task in itself, as it requires collision-free placement of the various bodies of the robot to be found.

The points used to specify the goal configurations of a robot are exactly those that are later used by the planner as the control points. Let $p_1, \ldots, p_s$ be these points. The configuration space potential U is defined as a combination:

$$U(q) = G\{V_{p_1}[X(p_1, q)], \ldots, V_{p_s}[X(p_s, q)]\}$$

of the work space potentials V_{p_i}, $i = 1, \ldots, s$, defined for the s control points p_i. This combination concurrently attracts the different points p_i toward their respective goal positions. G is called the *arbitration function*. This terminology reflects the facts that the various control points may compete to attain their goal positions and that the function G arbitrates this competition.

In most of the previous collision-avoidance systems using artificial potential fields, G was chosen

as a linear combination of the work space potentials (Khatib 1986); i.e.,

$$G(y_1, \ldots, y_s) = \sum_{i=1}^{i=s} \lambda_i y_i.$$

This simple choice seems natural, because it does not favor one control point over the others. However, precisely for that same reason, it tends to increase the number of conflicts among the control points, thus producing numerous undesired local minima.

The choice of the function G is important, as it highly influences the number of local minima of the potential U. With our "perfect" work space potentials, the work space concavities do not directly create local minima. It is the concurrent attraction of the different control points toward their respective goal positions that creates these local minima.[2] This results from the fact that these points do not move independently. As suggested earlier, the function G precisely defines the way in which the competition between the different points is to be regulated.

The choice of G that seems to minimize the number of local minima is:

$$G(y_1, \ldots, y_s) = \min_{i=1}^{i=s} y_i.$$

Indeed, this competition function favors the attraction of the point that is already in the best position to reach its goal. However, when one point has reached its goal position, the potential field is identically zero, and it does not attract the other points toward their goal positions. A solution to avoid this problem is simply to add another term to the arbitration function:

$$G(y_1, \ldots, y_s) = \min_{i=1}^{i=s} y_i + \varepsilon \max_{i=1}^{i=s} y_i \qquad (1)$$

where ε is a small real number. In our experiments with robots with few degrees of freedom (see section 4), we obtained the best results with $\varepsilon = 0.1$. However, the best value of ε may depend on the robot.

Another choice for G is:

$$G(y_1, \ldots, y_s) = \max_{i=1}^{i=s} y_i. \qquad (2)$$

This choice tends to increase the number of competitions between the control points and, therefore, the number of local minima. However, it can be a good choice for robots with many DOFs. As a matter of fact, the number of local minima is not the only measure for the quality of the potential, as it might be much more difficult to escape a local minimum with a small attractive well than a minimum with a large well. The above competition function increases the number of local minima, but experiments show that in general it also reduces their volumes. This is the function that gave the best results for planning the paths of manipulator arms with many DOFs and multiple control points (see section 5).

Unlike the work space potentials V_{p_i}, the configuration space potential U does not have to be precomputed before actually searching for a path. In fact, in high-dimensional configuration spaces, this precomputation would be intractable. The function U is only computed during the search of a path at those configurations that are attained by the search algorithm.

4. Fast Path Planning for Robots With Few DOFs

We have implemented two versions of our planner, called the *Best-First Planner* (BFP) and the *Randomized Path Planner* (RPP). These two versions[3] differ mainly in the way the local minima of the configuration space potential function are escaped. In this section we present the simplest version of the planner, BFP. This version is fast for robots with few DOFs (less than five), but it is only applicable to these robots.

BFP iteratively considers each level of resolution in the work space pyramid, from the coarsest to the finest. It terminates with success as soon as it has generated a path. It terminates with failure if, after having considered the finest bitmap, it still has not generated a path.

At each level, BFP performs a best-first search (Nilsson 1980) of the collision-free subset $\mathcal{C}_{free}$ of the configuration space grid using the potential U defined in formula (1) as the heuristic function. If r is the resolution of the work space bitmap, $R_i = r - \log_2 (J^i_{sup}) - \log_2 (nbtol)$ is the resolution of the discretization along each q_i axis in $\mathcal{C}$, for every $i = 1$ to n. The successors of a configuration in the search graph are all its neighbors lying in $\mathcal{C}_{free}$. In our implementation, we chose the n-neighborhood, which means that two configurations are neighbors iff 1 to n of their coordinates differ by a single incre-

2. As we will make explicit later, local minima may also appear in the boundary of $\mathcal{C}_{free}$ at configurations where the gradient of the potential function U is not zero.

3. In fact, the two versions are blended in a single program that offers two options. In this article we distinguish between BFP and RPP to make the presentation clearer.

ment, the others being the same. Hence each configuration may have up to $3^n - 1$ neighbors. The size of this neighborhood is reasonable because, for other reasons, n has to be small.

As long as the best-first search process does not reach a local minimum of the function U, the search reduces to following the negated gradient of the potential (fastest descent procedure). When a local minimum is reached, the search algorithm simply fills up the well of the local minimum until it reaches a saddle point. Then the search proceeds again along the negated gradient of U. It stops when the goal configuration is attained.

At this stage, there is an important aspect of the algorithm that we must make precise. The use of the potential U to guide the search does not guarantee that there will be no collision of the robot with the obstacles. Therefore whenever the planner considers a new configuration q in $\mathcal{C}$, it should check that it lies in the free space. Because the planner does not represent the configuration space obstacles explicitly, the verification is done in the work space using a simple divide-and-conquer technique. To illustrate the idea, let the robot be a line segment of length L in a two-dimensional work space. We assume that the motion increments in the configuration space grid are small enough so that if the robot is in free space at one discretized configuration, it cannot lie entirely inside an obstacle at a neighboring configuration. Using the precomputed d_1 map, we obtain the distances $d_1(begin)$ and $d_1(end)$ of the two end points of the segment representing the robot at the configuration q. If the minimum of these two distances is smaller than the length L of the segment, then we are certain that the robot does not hit any of the obstacles at q. Otherwise, we divide the segment into two segments of equal lengths and repeat the computation recursively for the two segments. In two-dimensional work spaces, this computation can easily be generalized to robots whose boundaries consists of straight edges and/or curve segments of higher degrees (e.g., circular and elliptical arcs) by treating each segment separately. In three-dimensional work spaces, the computation can be extended as follows to robots made up of three-dimensional bodies: Assume first that the robot is a triangle whose vertices are v_1, v_2, and v_3. If the maximum of the distances $d_1(v_1)$, $d_1(v_2)$, and $d_1(v_3)$ is greater than the maximum of the lengths of the edges of the triangle, then we are certain that the configuration is collision-free. Otherwise, we divide every edge of the triangle in two equal segments and repeat the test recursively for the four triangles whose vertices are v_1, v_2, v_3, and the edge mid-points. This computation is extended to robots described as collections of polyhedra by triangulating the faces of the polyhedra and considering the generated triangles separately.

Remark. One may argue that the need for collision checking would be eliminated if we used a potential tending toward infinity in the boundary of $\mathcal{C}_{free}$. However, it must be noticed that the computation of such a potential at any configuration q includes the computation of the shortest distance between the robot at q and the obstacles. Hence it includes the computational cost of making the above collision test.

□

Note here that two types of local minima can be attained: the natural minima of U (where the gradient is zero) and the minima located in the boundary of $\mathcal{C}_{free}$ (where the gradient is nonzero in general). Both types of minimum are escaped in the same fashion.

We have implemented BFP in a program written in C language and run on a DEC 3100 MIPS-based workstation. We have experimented with this program using a "mobile robot" with two DOFs of translation and one DOF of rotation—namely, a long rectangle in a two-dimensional work space. Figure 6 shows a path generated by the planner that demonstrates the ability of the planner to produce complex maneuvers. In this example, the configuration space potential was computed using formula (1) with two control points located at the two extremities of the bar. The path was generated in a 256^2 work space bitmap. The running time of the planner

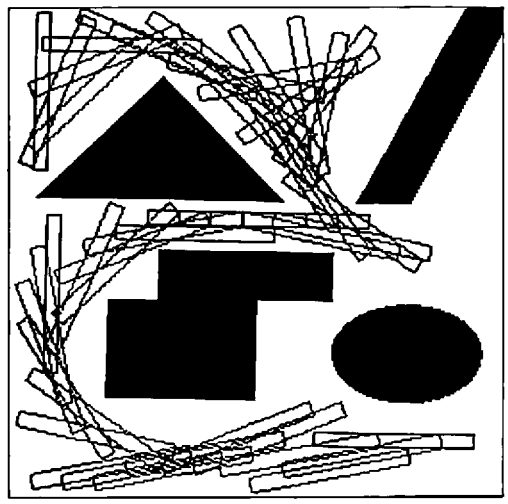

Fig. 6. Path generated for a 3-DOF mobile robot.

was 1 s. This is three orders of magnitude faster than the running times reported in Brooks and Lozano-Pérez (1983) for similar (though apparently simpler) path-planning problems. Roughly one order of magnitude is a result of the faster computer that we used. The other two seem to be the product of our algorithm.

Figure 7 shows another path generated by BFP for the same mobile robot. The path was generated in less than 5 s within a 512^2 bitmap representing a work space with more than 70 randomly constructed obstacles of arbitrary shapes. This example would be very difficult (at best) to run with a planner using a semi-algebraic representation of the work space and illustrates the power of the distributed representation approach used in our planner.

BFP is practical only for robots with a small number n of DOFs—typically, $n < 5$—because the number of discrete configurations in a local minimum well increases exponentially with the dimension of the configuration space. For such robots, it has two major advantages:

1. As illustrated earlier, it is extremely fast. Simpler but nontrivial problems than those of Figures 6 and 7, requiring less complex maneuvers, were solved in about $\frac{1}{5}$ s (at a coarse level of resolution), which can almost be considered real time.
2. It is deterministically resolution complete (i.e., the algorithm generates a path to the goal whenever a solution exists at the maximal resolution and returns failure if there is no solution). In both cases, BFP returns control within some bounded amount of time.

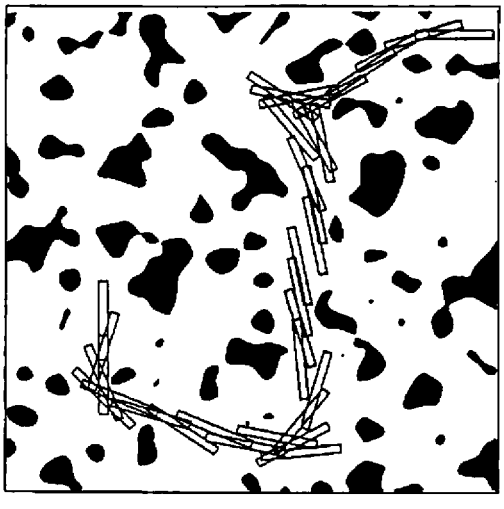

Fig. 7. Path generated among randomly distributed obstacles.

At first sight, the experimental efficiency of such a brute force technique is surprising. In fact, the potential functions computed in the work space are designed to be "perfect" potential fields for a point robot, in the sense that they have no other minimum than the goal. Therefore complex-shaped obstacles do not directly create local minima of the potential. Local minima occur only when the work space is so cluttered that the solution path has to come very close to the obstacles. However, in such cases, because the work space potentials computed by NF2 guide the robot along a path where the maneuvering space is maximized, the local minima usually have a reduced domain of attraction—i.e., the number of discrete configurations contained in the local minimum wells is usually small. The best-first search algorithm therefore fills the wells very quickly.

Remark. Whenever the planner fails to find a path at some resolution level, it forgets about it and considers the next level of resolution. Because the time required to explore a coarser grid is small relative to the time needed for a finer grid, this naive coarse-to-fine approach is adequate. Nevertheless, one could imagine another approach where the work done at a resolution level would be used to guide the work at the next resolution level. However, the solution of a path-planning problem is often so versatile with respect to the resolution of the representation that the information passed from one resolution level to the next might be more misleading than helpful. Moreover, having no interaction among the searches at the various resolution levels makes it possible to perform them concurrently on a parallel machine. □

5. Path Planning for Robots With Many DOFs

The BFP planner presented in the previous section cannot efficiently plan motions for robots with many DOFs. On the other hand, human beings are able to solve motion-planning problems with a high number of DOFs. Sometimes redundancy among the DOFs even seems to simplify planning. Hence there is a "divorce" between the exponential-time complexity of the general path-planning problem and our everyday life experience. In order to design an efficient path planner applicable to robots with many DOFs, it seems that we have to drop the completeness requirement. In this section we describe the RPP version of our planner that has demonstrated its ability to solve many complex planning problems,

some being nontrivial for humans. The new algorithm differs from the one of the previous section in the way it escapes local minima. Rather than filling up a local minimum, it applies a Monte Carlo procedure that consists of generating Brownian motions until the minimum is escaped. The resulting planning algorithm is probabilistically (rather than deterministically) resolution complete. For robots with many DOFs, we used the configuration space potential defined by formula (2) in our experiments, rather than the one defined by formula (1). However, the Monte Carlo procedure itself does not depend on the particular potential function that is used.

5.1. Overview

Starting from the initial configuration q_{init} of the robot, RPP first applies a best-first algorithm (i.e., it descends the potential U until it reaches a local minimum q_{loc}; see section 5.2). We call such a motion a *gradient motion*. Let $U_{loc} = U(q_{loc})$. If $U_{loc} = 0$, the problem is solved, and the planner returns the constructed path. Otherwise, it attempts to escape the local minimum by executing a series of *random motions* issued from q_{loc}. These random motions are approximations of Brownian motions described in section 5.3.

At the terminal configuration of every random motion, the algorithm executes a gradient motion until it reaches a (hopefully) new local minimum. From each local minimum, if none of them is the goal, it performs another series of random motions. The graph of the local minima is thus incrementally built, the path joining two "adjacent" local minima being the concatenation of a random motion and a gradient motion. A randomized depth-first search of this graph is performed (see section 5.4) until the goal configuration is reached or the planner gives up. If the search terminates successfully, the generated path is transformed into a smoother path (section 5.5).

An interesting property of this planning algorithm is that all the random motions starting from a given local minimum can be performed concurrently on a parallel machine, as there is no need for communication among the different processing units.

Because the algorithm uses a random procedure to build the graph of the local minima, it is not guaranteed to find a path whenever one exists. In other words, the algorithm is not complete. However, the properties of Brownian motions make it possible to prove that when the number of Brownian motions executed from every local minimum is unbounded (the computation time may then tend toward infinity), the probability to reach the goal converges toward 1. Hence we say that the algorithm is *probabilistically resolution complete*. However, this convergence-in-distribution property, which is well known for the so-called "simulated annealing" algorithms (Geman and Hwang 1986), is a very weak one. Indeed, the totally uninformed algorithm that executes a Brownian motion from the initial configuration q_{init} and terminates when it enters a small neighborhood of the goal configuration is also probabilistically resolution complete! Despite the weakness of this theoretical result, our experiments show that RPP is quite efficient.

We describe the various components of RPP in more detail in the following sections. We also give experimental results obtained with the implemented planner.

5.2. Gradient Motions

The planner operates in the configuration space grid over which U is defined. In principle, a gradient motion consists of searching this grid in a best-first fashion as described in section 4. However, when the dimension n of the configuration space becomes large, say $n \geq 6$, the n-neighborhood used in section 4 to determine the direction of motion is much too large to be fully explored at each step. For example, if $n = 10$, a configuration has approximately 60,000 n-neighbors; if $n = 31$, it has over 600 thousands of billions n-neighbors.

One way to deal with this difficulty is simply to use a smaller neighborhood—for example, the 1-neighborhood (two configurations are 1-neighbors iff only one of their coordinates differs and if it differs by a single increment of the grid). Then each configuration has only $2n$ neighbors. However, experimentations showed that this solution is not very good, because the crude discretization of the neighborhood of a configuration often results in the detection of a fictitious local minimum.

Another technique consists of using the full n-neighborhood and checking only a small number of configurations randomly selected in this neighborhood. At each step of a gradient motion, the n-neighborhood of the current configuration q is partially explored as follows. A neighbor q' of q is chosen randomly using a uniform distribution law. If it is a free configuration (the test is carried out in the work space as described in section 4) and if $U(q') < U(q)$, q' is taken as the successor of q along the path of the gradient motion. (Hence the path may only follow a rough approximation of the negated gradient flow.) If q' is not free or if $U(q') \geq U(q)$,

another neighbor q' is randomly chosen. The number of iterations is limited to a few tens to a few hundreds (depending on the value of n). If none of them generates a successor of q, q is considered to be a local minimum.

Both techniques mentioned above have been implemented, and the second one gave much better results, generating almost no fictitious minima. As a matter of fact, if the curvature of the surface $U = U(q)$ at q is small and if q is not a minimum of U, then at every iteration, the probability of guessing a configuration q' such that $U(q') < U(q)$ is approximately equal to 0.5. The probability of finding such a configuration q' in 10 independent guesses is of the order of 0.999. Although the number of guesses may be increased with n, it does not have to be increased proportionally.

5.3. *Random Motions*

When the algorithm reaches a local minimum of the potential field U, we consider that there is no more information that we can extract locally from U in order to guess the direction of motion that will lead us to the goal. Then *if we do not make any assumption on the statistics of the obstacle distribution*, we have no additional information for helping us to reach the goal. RPP continues the search by executing random motions issued from the current local minimum q_{loc}.

The most uninformed type of motion is known to be the Brownian motion (Papoulis 1965). Because a Brownian motion is a continuous stochastic process, the random motions performed by RPP are approximations of Brownian motions and are defined as discrete random walks. A random walk in the configuration space consists of executing a certain number t of steps (the "duration" of the random walk). Each step corresponds to a "unit" of time and projects into every q_i axis, $i \in [1, n]$, as an increment $+v_i$ or $-v_i$ of fixed amplitude, each with the constant probability 0.5 (hence independent of the previous steps). The amplitude of the increment, v_i, is the "velocity" of the walk along the q_i axis. This random walk is known to converge almost surely toward a Brownian motion when the amplitude v_i of every increment tends toward 0 (Papoulis 1965).

Without lack of generality, let us take the current local minimum, q_{loc}, as the origin of the coordinates of $\mathcal{C}$. The configuration attained by a Brownian motion of duration t and velocity v_i along each q_i axis is a random variable $Q(t) = (Q_1(t), \ldots, Q_n(t))$ with the following properties (Papoulis 1965):

- The density p_i of $Q_i(t)$ is the Gaussian distribution given by:

$$p_i(q_i, t) = \frac{1}{v_i\sqrt{2\pi t}} \exp\left(-\frac{q_i^2}{2v_i^2 t}\right).$$

- The standard deviation $D_i(t)$ of the difference $Q_i(t + t_0) - Q_i(t_0)$ increases proportionally to the square root of t; i.e.,

$$D_i^2(t) = E\{[Q_i(t_0 + t) - Q_i(t_0)]^2\} = v_i^2 t.$$

- The two processes $Q(t') - Q(t)$ and $Q(t'') - Q(t')$ are independent, for any (t, t', t'') such that $t < t' < t''$.

The Brownian motion (also called the *Wiener-Levy process*) is well defined as long as it does not encounter any obstacle in configuration space. When the process $Q(t)$ hits the boundary of an obstacle, the Brownian motion has to be adapted so that it remains in the free space. The classic generalization of a Brownian motion when the space is bounded consists of reflecting the motion that would have taken place in the absence of boundary, symmetrically to the tangent hyperplane of the boundary at the collision configuration (Brownian motion with "reflective boundary"). The mathematic consistency of this adaptation is discussed in detail in Anderson and Orey (1976). Our planner, which does not construct an explicit representation of the configuration space obstacles, does not know the orientation of the tangent hyperplane at the collision configuration. Hence whenever a random motion step leads to colliding with an obstacle, instead of reflecting the motion on the boundary, the planner guesses another random step and substitutes it for the previous one. Collisions along a Brownian motion are checked in the work space using the divide-and-conquer technique presented in section 4.

We still have to select the velocities v_i and the duration t of every random motion. Because we approximate a Brownian motion as a random walk in a grid where the increment along each q_i axis is Δ_i, we would like the standard deviation of each step to be equal to Δ_i. This leads to choosing $v_i = \Delta_i$. Regarding the duration t, we should choose it such that the generated random motion take the robot out of the current local minima of U. Let us define the *attractive radius* $a_{R_i}(q_{loc})$ of any local minimum q_{loc} of U along the q_i axis as the distance along q_i between q_{loc} and the nearest saddle point of U in that direction. In order to escape the local minimum q_{loc}, the minimum distance that the robot must travel in each direction q_i from q_{loc} is $a_{R_i}(q_{loc})$.

If we were able to estimate the statistics of a_{R_i}, the property $D_i(t) = \Delta_i\sqrt{t}$ would give us a clue for choosing t. The duration of the motion would then be:

$$t \approx \max_{i\in[1,n]} \left(\frac{a_{R_i}(q_{loc})}{\Delta_i}\right)^2. \quad (3)$$

However, as we make no assumption on the obstacle distribution, we cannot infer any strong statistical property about U and a_{R_i}. However, in general, we may assume that the distance a_{R_i} for each parameter q_i does not exceed the distance that would provoke a motion of the robot longer than the work space diameter (defined by the input bitmap). This diameter being equal to 1 (with the normalized L^1 distance previously used), we obtain the following estimate of a_{R_i} for any local minimum q_{loc}:

$$a_{R_i}(q_{loc}) \approx 1/J^i_{\text{sup}}.$$

On the other hand, we have $\Delta_i \approx \delta/J^i_{\text{sup}}$, where δ is the distance between two consecutive points along the same coordinate axis of the work space. Combining these two formulas with (3), we obtain:

$$t \approx \frac{1}{\delta^2}.$$

We could take t equal to this value. However, this choice would mean that we implicitly assume that all the attraction radii are the same, which is not the case. Instead, we take a_{R_i} as the value of a strictly positive random variable A_{R_i} whose expected value is $1/J^i_{sup}$. The most uninformed distribution (i.e., the one that maximizes entropy) of a positive random variable of given expected value is the truncated Laplace distribution. Therefore we define the density of A_{R_i} as:

$$p(a_{R_i}) = J^i_{\text{sup}} \exp(-J^i_{\text{sup}} a_{R_i}).$$

This leads to choosing the duration t of a random motion as the value of a random variable T. The above density of A_{R_i}, combined with the relation (3), entails the following density for T:

$$p(t) = \frac{\delta}{2\sqrt{t}} \exp(-\delta\sqrt{t}). \quad (4)$$

One can verify that the expected value of this distribution is indeed $1/\delta^2$.

In fact, a value of T gives a maximal duration of the random motion. After each step of the motion, the planner checks the value of the potential at the current configuration against $U(q_{loc})$. If it is smaller, the planner terminates the random motion.

Remark. As mentioned at the beginning of this subsection, executing Brownian motions when the algorithm reaches a local minimum of the potential U other than the goal configuration corresponds to assuming that there is no more local information that we can extract from U in order to guess the direction of motion that will lead us toward the goal. However, higher order derivatives could provide useful additional information. As a matter of fact, in Barraquand et al. (1989), we used the concept of valley (which is based on the first and second derivatives) to escape local minima. The resulting planner was not very reliable, but the idea of tracking valleys for escaping local minima could be reused here to generate more informed random motions.

□

5.4. *Searching the Local Minima Graph*

By combining best-first motions and random motions, RPP can incrementally construct a graph of the local minima of the potential function U. The search of this graph can be done using a best-first strategy. This simply consists of iteratively generating the successors of the pending local minimum having the smallest potential value and limiting the number of random motions issued from the same minimum to a prespecified number M. A drawback of this strategy is that the same minimum may be attained several times, which is difficult and costly to detect. The strategy may also waste time exploring a local minimum well containing smaller local minimum wells imbricated in one another.

Another search strategy is the following depth-first strategy. When a local minimum q_{loc} is attained, a maximum of M random motions are generated. Each random motion is immediately followed by a best-first motion that attains a local minimum q'_{loc}. If $U(q'_{loc}) \geq U(q_{loc})$, then the planner forgets q'_{loc}; otherwise, if q'_{loc} is not a goal configuration (in which case the problem is solved), its successors are generated in the same fashion. If none of the M motions issued from q_{loc} allow the planner to attain a lower local minimum than q_{loc}, the latter is considered to be a dead-end, and the search is resumed at the most recent local minimum whose M successors have not all been generated yet (chronologic backtracking).

This second search strategy gives better experimental results than the first, but it may still waste time exploring imbricated minima. Moreover, if a

low local minimum has been attained (this is the case in the example shown in Figure 20 later), it may have difficulty attaining a lower one from it. (It is not difficult to construct examples where all the free paths between a very low local minimum and a goal configuration are quite long and unlikely to be generated by a single random motion.)

The above strategy can be slightly modified as follows: Rather than memorizing the whole graph G, the planner only memorizes the constructed path τ connecting the initial configuration to the current one (τ is represented at a list of configurations in the configuration space grid). At every local minimum, the planner iteratively generates a maximum of M (typically $M \approx 20$) random/best-first motions as described earlier. As soon as one of them reaches a lower minimum q'_{loc}, it inserts the path from q_{loc} to q'_{loc} at the end of the current path τ and continues the search from q'_{loc}. If the M motions are performed and none of them attains a lower minimum, the planner randomly selects a configuration q_{back} in the subset of τ formed by random motions, using a uniform distribution law over that subset, and backtracks to q_{back}. The search is resumed at q_{back} by executing a best-first motion. Because q_{back} belongs to the path of a random motion, this best-first motion may terminate at a new local minimum that has not been attained so far. (If the first local minimum q_{loc} encountered by the planner turns out to be a dead-end, the above backtracking mechanism cannot be applied. Then RPP randomly selects one of the local minima q'_{loc} attained from q_{loc} and resumes the search with a best-first motion starting at a configuration randomly selected along the path of the random motion leading to q'_{loc}.)

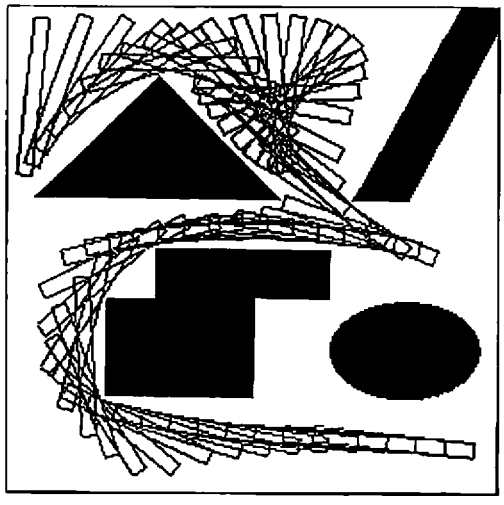

Fig. 8. Path generated by RPP for a 3-DOF rectangular robot.

A more formal expression of the search carried out by RPP is given in the following procedure:

```
procedure RPP-SEARCH;
begin
  τ ← GRADIENT-PATH(q_init); q_loc ← LAST(τ);
  while q_loc ∉ GOAL do
    begin
      ESCAPE ← false;
      for i = 1 to M until ESCAPE do
        begin
          t ← RANDOM-TIME;
          τ_i ← RANDOM-PATH(q_loc, t);
          q_rand ← LAST(τ_i);
          τ_i ← PRODUCT(τ_i, GRADIENT-
            PATH(q_rand));
          q'_loc ← LAST(τ_i);
          if U(q'_loc) < U(q_loc) then
            begin
              ESCAPE ← true;
              τ ← PRODUCT(τ, τ_i);
            end;
        end;
      if ¬ ESCAPE then
        begin
          τ ← BACKTRACK(τ, τ_1, . . . , τ_M);
          q_back ← LAST(τ);
          τ ← PRODUCT(τ, GRADIENT-
            PATH(q_back));
        end;
      q_loc ← LAST(τ);
    end;
end;
```

where:

τ is a list of configurations representing the path constructed so far;
LAST(τ) returns the last configuration in a path τ;
PRODUCT(τ_1, τ_2) returns the list of configurations representing the concatenation $\tau_1 \bullet \tau_2$ of two paths τ_1 and τ_2;
GRADIENT-PATH(q) returns the path generated by a gradient motion starting at q;
RANDOM-PATH(q, t) returns the path generated by a random motion of duration t starting at q;
RANDOM-TIME returns a random duration computed with the distribution defined in formula (4);
BACKTRACK(τ, τ_1, . . . , τ_M) selects a backtracking configuration and returns the path from q_{init} to this configuration; if τ includes a subpath generated by a random motion, then the

returned path is a subpath of τ; otherwise it is a subpath of $\tau \bullet \tau_i$, with i randomly chosen in $[1, M]$.

The above search techniques have all been implemented, and the technique described in the procedure RPP-SEARCH gave the best experimental results.

5.5. *Path Optimization*

The path produced by the search of the local minima graph consists of a succession of gradient and random motions, both of them containing a large spectrum of spatial frequencies. To enable the robot to execute a graceful motion, the resulting path has to be smoothed. The smoothing procedure can be theoretically described as an optimization problem: Given an initial path τ_{init}, find a new path τ in the homotopy class of τ_{init} that minimizes the functional:

$$J(\tau) = \int_0^T K(\dot{\tau}(t))dt,$$

where K is the quadratic form of the kinetic energy under the obstacle avoidance constraint, and the two conditions $\tau(0) = q_{init}$ and $\tau(T) = q_{goal}$. To reduce the amount of computation, we use a simplified diagonal form for K, which corresponds to artificially decoupling the different degrees of freedom. The geodesics of the corresponding Riemannian metric are simply straight line segments in the generalized coordinate system $(q_1, \ldots, q_n)$.

Because any spatial frequency may be present in the initial path, it is highly preferable to use a multiscale technique for minimizing J. Our optimization procedure consists of iteratively modifying the path τ_{init} by replacing subpaths of decreasing lengths with straight line segments in the configuration space. It is necessary to check each of the straight line segments to ensure that it is collision free. The algorithm first checks long segments of the order of the total length of the path, and then smaller and smaller ones until the resolution of the configuration space grid is attained. The final path generated by this algorithm generally lies in the same homotopy class as the initial one.

5.6. *Experimental Results*

We have implemented RPP in a program written in C language that runs on a DEC 3100 MIPS-based workstation. Interestingly, the program only consists of about 1500 lines of code (but it does not include fancy inputs/outputs). We have experimented successfully with RPP using a variety of robot structures. We have also run the planner to generate paths for a PUMA robot in our laboratory and for a dual-arm system in the Stanford Aerospace Robotics Laboratory. We present here some of the most significant experiments. Because the algorithm contains several random components, neither the running time of the planner nor the generated solution are constant across several runs for the same problem. The times given below are typical times. It is not unusual for two running times for the same example to differ by a ratio of 5. The execution times would be both smaller and much more stable on a parallel architecture allowing the concurrent execution of several random motions.

3-DOF Rectangular Robot in 2D Work Space. Figure 8 shows a path generated by the planner for a holonomic rectangular robot in the plane. The resolution of the work space bitmap was 256^2. The computation time for this example was approximately 10 s, whereas the best-first planner (BFP) of section 4 takes only a second to solve this same problem.

6-DOF U-Shaped Robot in 3D Work Space. Figure 9 shows snapshots along a path generated by RPP for a U-shaped robot that can translate and rotate freely in a three-dimensional work space. The obstacles consist of a parallelepipedic block and a lattice. Because the work space is bounded, the robot must "maneuver" among the bars of the lattice.

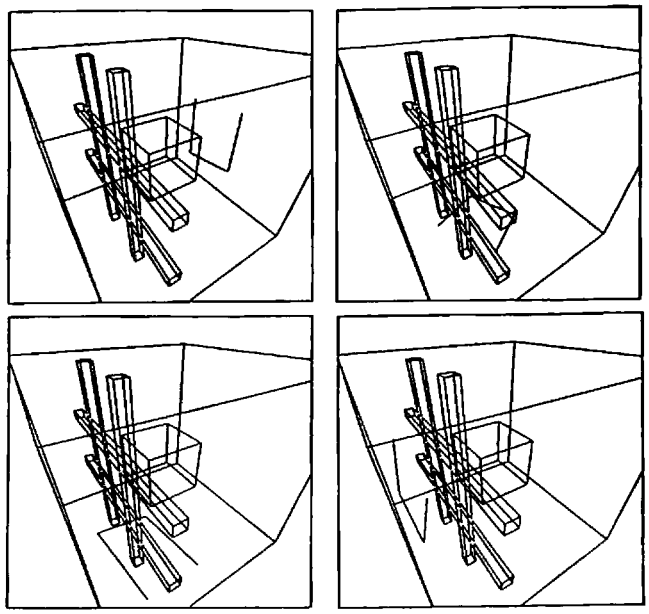

Fig. 9. Path generated by RPP for a 6-DOF U-shaped robot.

8-Revolute-DOF Serial Manipulator Robot in 2D Work Space. We ran RPP with the planar serial manipulator shown in Figure 10, which has eight revolute joints. Figures 11 and 12 show two paths generated by RPP. In both examples, the goal was defined by the position of the end point of the last link of the robot. The potential U was computed using this point as the only control point. The paths were generated in approximately 2 minutes for the first example and 30 seconds for the second, with a 216^2 work space bitmap. Collisions among the links were forbidden and checked by the planner.

10-DOF Nonserial Manipulator Robot in 2D Work Space. We also applied RPP to the planar nonserial manipulator robot shown in Figure 13, which includes three prismatic joints (telescopic links) and seven revolute joints. Figures 14 and 15 illustrate two different paths of the robot. In both examples, we used a potential U computed with two control points located at the end points of the two kinematic chains. Overlapping of the links was forbidden in the first example, but not in the second. The first example was solved in 3 minutes, whereas the second was solved in 2 minutes. Both paths were constructed with a 256^2 work space bitmap. The size of the corresponding configuration space grid is of the order of 10^{20} configurations.

31-DOF Nonserial Manipulator Robot in 3D Work Space. Continuing with our manipulator series, we experimented with RPP using the 31-DOF manipulator illustrated in Figure 16. This manipulator consists of 10 telescopic links connected by 10 spherical joints. The bar at the end of the manipulator is connected to the last link by a revolute joint. A path

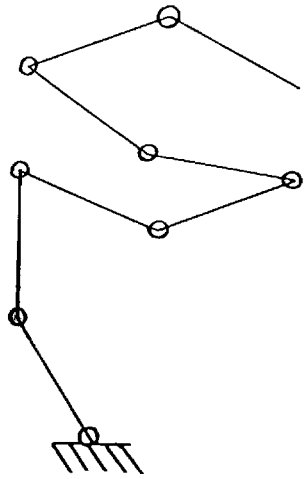

Fig. 10. Structure of the 8-DOF serial manipulator.

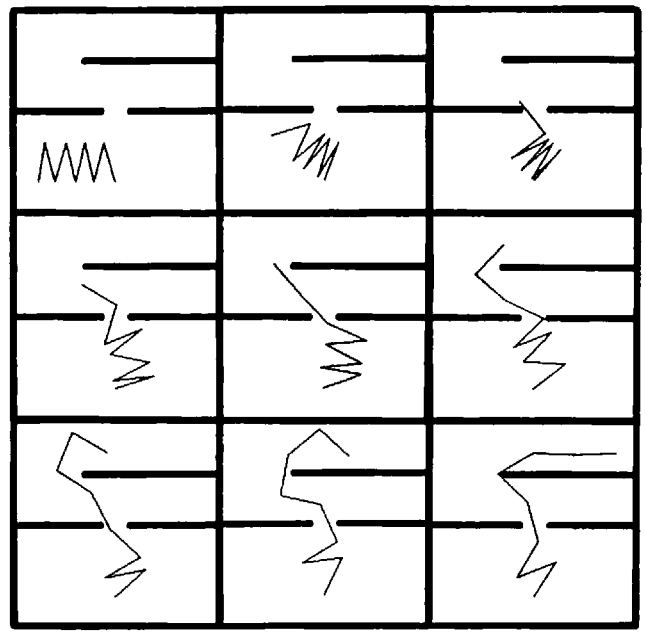

Fig. 11. Path generated by RPP for the 8-DOF serial manipulator (example 1).

generated by RPP is illustrated in Figure 17. The potential was computed with two control points located at the end points of the bar. The computation time was of the order of 15 minutes. The size of the work space bitmap was 128^3. The size of the corresponding configuration space grid is of the order of 10^{62} configurations.

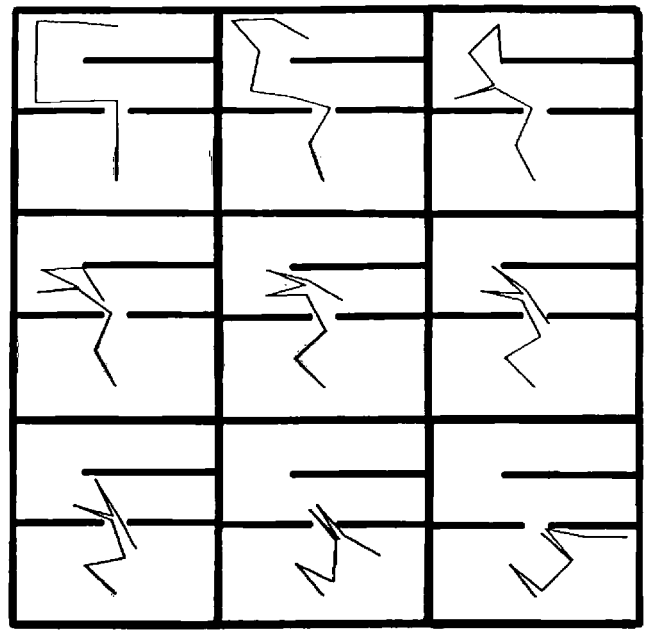

Fig. 12. Path generated by RPP for the 8-DOF serial manipulator (example 2).

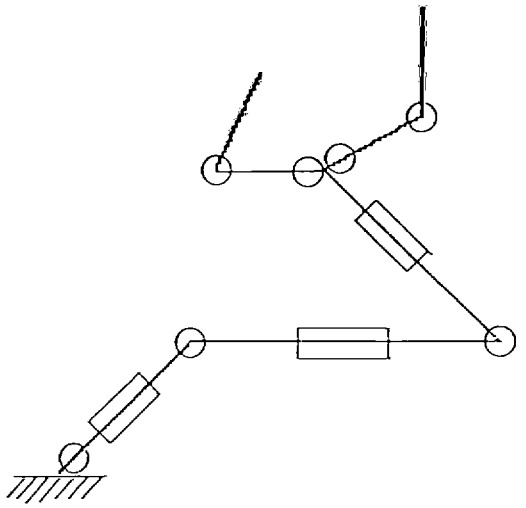

Fig. 13. Structure of the 10-DOF nonserial manipulator robot.

Coordination of Two 3-DOF Mobile Robots. The same planner was applied to problems requiring the coordination of two 3-DOF mobile robots in a two-dimensional work space made of several corridors. These are narrow enough so that the two robots cannot pass each other in the same corridor (Figure 18). The two robots are treated by RPP as a single two-body robot with 6 DOFs. Figures 19 and 20 display two paths generated by RPP for two different problems in the same environment. The second problem is particularly difficult, because the two robots have to interchange their positions in the central corridor; hence both of them must first move to an intermediate position in order to allow the permutation. Notice that in the initial configuration, both robots are rather close to their respective goal configurations, despite the fact that the paths to move there are not short. This example illustrates the

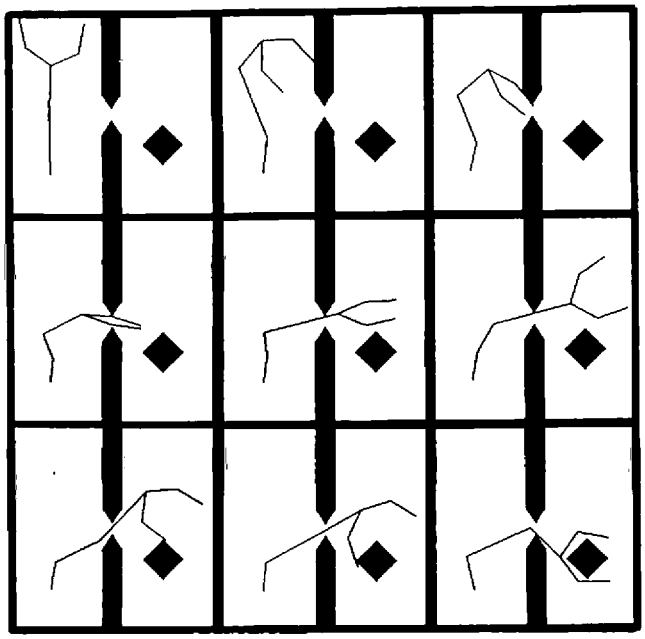

Fig. 14. Path generated by RPP for the 10-DOF nonserial manipulator (example 1).

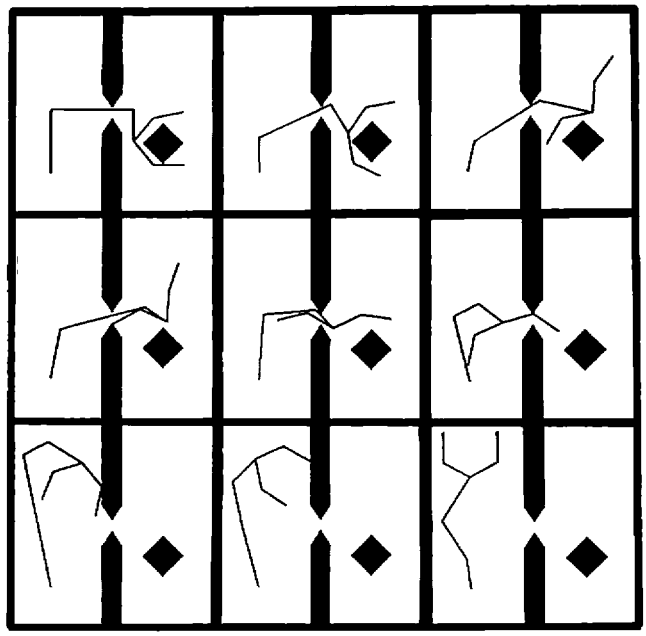

Fig. 15. Path generated by RPP for the 10-DOF nonserial manipulator (example 2).

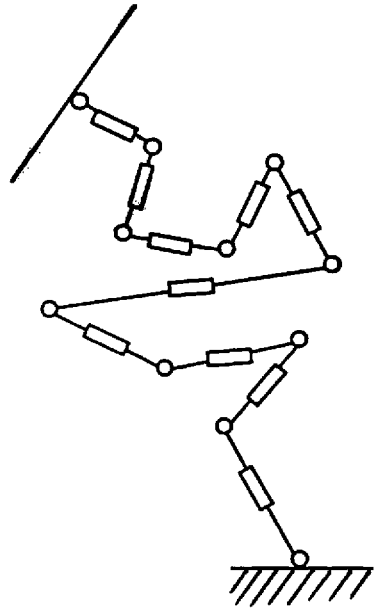

Fig. 16. Structure of the 31-DOF serial manipulator robot.

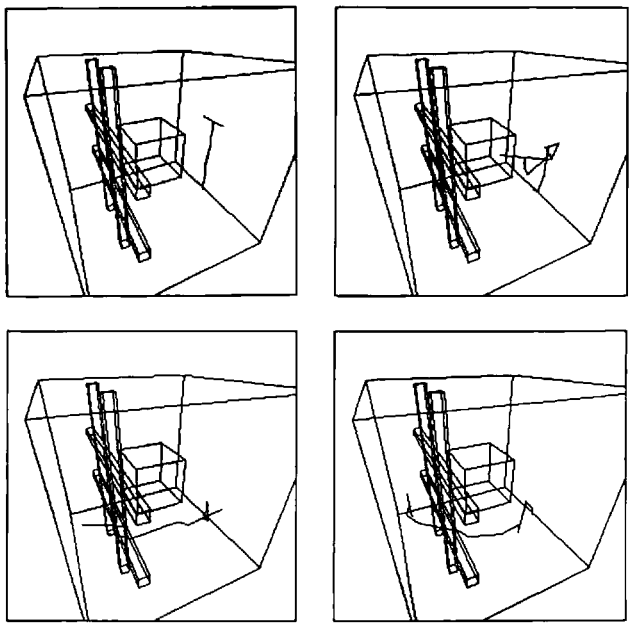

Fig. 17. Path generated by RPP for the 31-DOF serial manipulator.

power of the random techniques used in RPP. The two paths were generated in about 30 s.

5.7. *Discussion*

The experiments with RPP have shown that randomized planning is both efficient and reliable. The efficiency of RPP results from the fact that a typical path-planning problem has many solutions, so that a globally random search procedure can find one if it is well informed most of the time (by the potential function). In fact, similar randomized techniques (e.g., "simulated annealing") have proven to be useful for solving other NP-hard problems—e.g., the traveling salesman problem (Cerny 1985) and VLSI placement and routing (Kirkpatrick et al. 1983; Sechen 1988). In these problems the very large search space is associated with a large number of "good" suboptimal solutions.

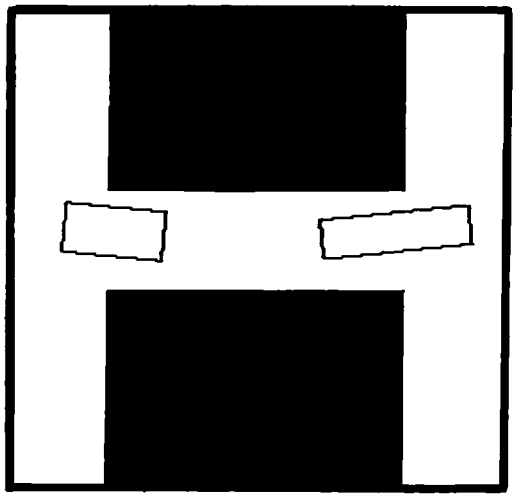

Fig. 18. The corridor problem for two 3-DOF mobile robots.

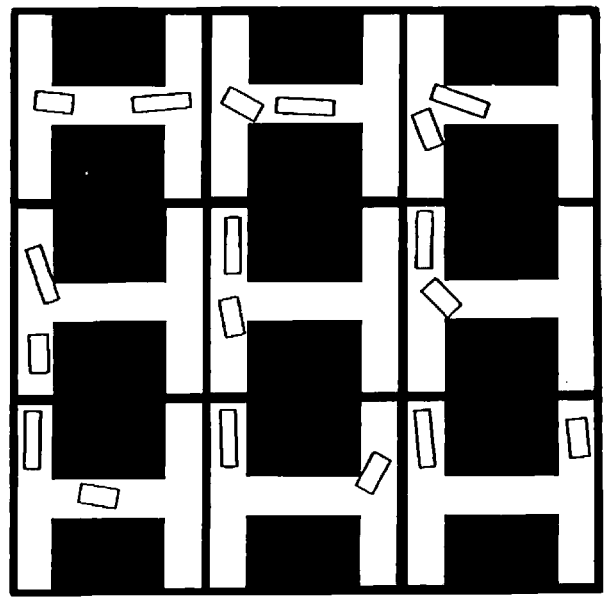

Fig. 19. Coordinated path for the corridor problem (example 1).

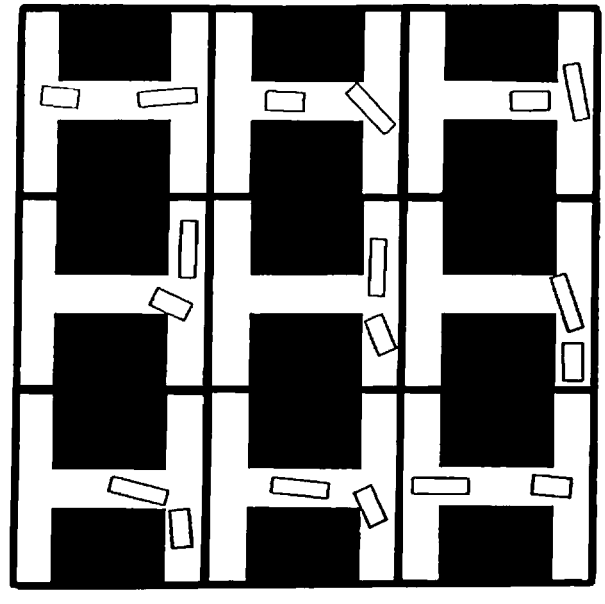

Fig. 20. Coordinated path for the corridor problem (example 2).

Monte Carlo procedures for optimization have also been used more or less successfully in Computer Vision. In Geman and Geman (1984), a simulated annealing approach is applied for restoring images blurred with nonlinear filters. Several authors have implemented edge detection algorithms based on the same paradigm. Recently, simulated annealing has also been applied to higher level problems in Computer Vision. In Barnard (1988) the stereo-matching problem is addressed using a hierarchical pixel-level simulated annealing algorithm. A stochastic optimization approach for the three-dimensional reconstruction of stratigraphic layers and the detection of geologic faults in seismic data is proposed in Barraquand (1988).

Nevertheless, RPP behaves very differently from the classic simulated annealing procedures. On a sequential computer, simulated annealing procedures perform a kind of breadth-first search of the graph of local minima of the function to be optimized, whereas RPP performs a depth-first search of this graph using the potential U as the heuristic function. See Barraquand and Latombe (1989b) for a more detailed comparison of BFP and simulated annealing.

Randomized planning has some drawbacks, however. The planner typically generates different paths if it is run several times with the same problem, and the running time varies from one run to another. Furthermore, if the input path-planning problem admits no solution, the planner usually has no way to recognize it, even after a large amount of computation. Hence a limit on the running time of the algorithm has to be imposed. However, if this limit is attained and no path has been generated, there is no guarantee that no paths exist. However, in practice, experiments have shown that it is not difficult, for a given class of problem (e.g., an object moving in a three-dimensional work space) and a given size of the configuration space grid, to determine a time limit (through a series of preliminary trials) such that if no path has been found by the end of the time limit, there is little chance that one actually exists.

In Barraquand and Latombe (1989b), we proved that the randomized planning algorithm implemented in RPP is probabilistically resolution complete. This result is based on a general property of the Wiener-Levy process: Whenever the free space is connected and relatively compact, the probability for a Brownian motion ω with reflective boundary starting at any initial configuration $q_{init} \in \mathcal{C}_{free}$ to reach any given open subset B of $\mathcal{C}_{free}$ at least once during the interval of time $[0, t]$ converges toward 1 when the duration t tends toward infinity. We can choose B so that it contains at least one goal configuration and is small enough that it does not contain a local minimum (other than the goal minimum). From within B, a gradient motion achieves a goal configuration. See Barraquand and Latombe (1989b) for more details.

6. Conclusions

In this article we described a new approach to robot path planning. This approach essentially consists of (1) discretizing both the work space and the configuration space of the robot into a hierarchical bitmap and grid; (2) computing numeric navigation functions over the robot's work space and combining them into a "good" potential function in the configuration space; and (3) building and searching the graph of the local minima of the configuration space potential using an efficient technique to escape local minima. We proposed several techniques for constructing the potential function and two techniques for escaping local minima. The most powerful of these two techniques, which is applicable to robots with many DOFs, is a stochastic process technique based on the execution of Brownian motions with reflective boundary.

We have implemented our approach and the various techniques presented in this article in two programs, BFP and RPP, which we ran successfully on many different examples. BFP has solved 3-DOF robot problems also solved by previous planners, but several orders of magnitude faster. It has also solved problems with may obstacles of arbitrary shapes that were never attempted before. On the other hand, RPP has solved a large variety of problems that fall far outside the range of the capabilities of any other previous planner (e.g., problems with 8-DOF, 10-DOF, and 31-DOF robots and multirobot problems).

The algorithms implemented in RPP are highly parallelizable. A preliminary investigation of the parallelization of the planner has been conducted in Barraquand and Latombe (1989b) and in Métivier and Urbschat (1990). We envision an implementation of RPP using a specific hardware system, which should permit real-time path planning. Such a system would open new perspectives on some key issues in robotics related to the interaction of planning and execution in partially known and dynamically changing environments.

In addition to implementing a real-time planner, we currently conduct research aimed at using the randomized planning approach for planning manipulation tasks (Alami et al. 1989) involving grasping

and regrasping operations on movable objects with multiple robots.

Acknowledgments

This research was funded by DARPA contract DAAA21-89-C0002 (U.S. Army), DARPA contract N00014-88-K-0620 (Office of Naval Research), SIMA (Stanford Institute of Manufacturing and Automation), CIFE (Center for Integrated Facility Engineering), and Digital Equipment Corporation. The authors also thank Professor J. M. Harrison (Stanford Graduate School of Business) and J. Chang (Stanford Statistical Consulting Service) for their helpful advice on Brownian motions with reflective boundaries. Bruno Langlois also provided useful suggestions.

References

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

Alami, R., Siméon, T., and Laumond, J. P. 1989. A geometrical approach to planning manipulation tasks—the case of discrete placements and grasps. In Miura, H., and Arimoto, S. (eds.): *Robotics Research 5*. Cambridge, MA: MIT Press, pp. 453–463.

Anderson, R. F., and Orey, S. 1976. Small random perturbations of dynamical systems with reflecting boundary. *Nagoya Math. J.* 60:189–216.

Barnard, S. T. 1988. Stochastic stereo matching over scale. *Int. J. Computer Vision*. 2(4).

Barraquand, J. 1988. Markovian random fields in computer vision: Applications to seismic data understanding. Ph.D. dissertation. Dept. of Computer Vision, INRIA, Sophia-Antipolis, France. In French.

Barraquand, J., Langlois, B., and Latombe, J. C. 1989. Robot motion planning with many degrees of freedom and dynamic constraints. In Miura, H., and Arimoto, S. (eds.): *Robotics Research 5*. Cambridge, MA: MIT Press, pp. 435–444.

Barraquand, J., and Latombe, J. C. 1989a. On nonholonomic mobile robots and optimal maneuvering. *Revue d'Intelligence Artificielle* 3(2):77–103. (Also in *Proc. of the 4th IEEE Int. Symp. on Intelligent Control*, Albany, NY: pp. 340–347.)

Barraquand, J., and Latombe, J. C. 1989b. *Robot Motion Planning: A Distributed Representation Approach*. Report no. STAN-CS-89-1257. Dept. of Computer Science, Stanford University.

Barraquand, J., and Latombe, J. C. 1990. Controllability of mobile robots with kinematic constraints. Rep. no. STAN-CS-90-1317. Dept. of Computer Science, Stanford University.

Brooks, R. A., and Lozano-Pérez, T. 1983 (Karlsruhe). A subdivision algorithm in configuration space for findpath with rotation. *Proc. of the 8th Int. Joint Conf. on Artificial Intelligence*, pp. 799–806.

Canny, J. F. 1988. The Complexity of Robot Motion Planning. Cambridge, MA: MIT Press.

Cerny, V. 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Optimization Theory Applications*. 45(1):41–51.

Donald, B. R. 1984. Motion planning with six degrees of freedom. Tech. rep. 791. Artificial Intelligence Laboratory, MIT.

Erdmann, M., and Lozano-Pérez, T. 1986. On multiple moving objects. AI memo no. 883. Artificial Intelligence Laboratory, MIT.

Faverjon, B. 1984 (Atlanta). Obstacle avoidance using an octree in the configuration space of a manipulator. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 504–512.

Faverjon, B. 1986 (San Francisco). Object level programming of industrial robots. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1406–1412.

Faverjon, B., and Tournassoud, P. 1987 (Raleigh, NC). A local based approach for path planning of manipulators with a high number of degrees of freedom. *Proc. of the IEEE Int. Conf. on Automation and Robotics*, pp. 1152–1159.

Geman, D., and Geman, S. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Analysis Machine Intelligence* PAMI-6:721–741.

Geman, S., and Hwang, C. R. 1986. Diffusions for global optimization. *SIAM J. Control Optimization* 24(5).

Gouzènes, L. 1984. Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *Int. J. Robot. Res.* 3(4):51–65.

Kant, K., and Zucker, S. W. 1986. Toward efficient trajectory planning: Path velocity decomposition. *Int. J. Robot. Res.* 5:72–89.

Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.* 5(1):90–98.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220:671–680.

Koditschek, D. E. 1987 (Raleigh, NC). Exact robot navigation by means of potential functions: Some topological considerations. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1–6.

Latombe, J. C. 1990. *Robot Motion Planning*. Boston: Kluwer Academic Publishers.

Laugier, C., and Germain, F. 1985 (Tokyo). An adaptative collision-free trajectory planner. *Int. Conf. on Advanced Robotics*.

Lee, D. T., and Drysdale, R. L. 1981. Generalization of Voronoi diagrams in the plane. *SIAM J. Computing* 10:73–87.

Lozano-Pérez, T. 1983. Spatial planning: A configuration space approach. *IEEE Trans. Computers* C-32(2):108–120.

Lozano-Pérez, T. 1987. A simple motion-planning algo-

rithm for general robot manipulators. *IEEE J. Robot. Automat.* RA-3(3):224–238.

Lumelsky, V. 1987 (Los Angeles). Algorithmic issues of sensor-based robot motion planning. *Proc. of the 26th IEEE Conf. on Decision and Control*, pp. 1796–1801.

Métivier, C., and Urbschat, R. 1990. Run-time statistical analysis of a robot motion planning algorithm. Internal technical note. Robotics Laboratory, Dept. of Computer Science, Stanford University.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Los Altos, California: Morgan Kaufmann.

O'Donnell, P. A., and Lozano-Pérez, T. 1989 (Scottsdale, AZ). Deadlock-free and collision-free coordination of two robot manipulators. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 484–489.

Ó'Dúnlaing, C., Sharir, M., and Yap, C. K. 1983 (Boston). Retraction: A new approach to motion planning. *Proc. of the 15th ACM Symp. on the Theory of Computing*, pp. 207–220.

Papoulis, A. 1965. *Probability, Random Variables, and Stochastic Processes*. New York: McGraw-Hill.

Reif, J. H. 1979. Complexity of the mover's problem and generalizations. *Proc. of the 20th Symp. on the Foundations of Computer Science*, pp. 421–427.

Rimon, E., and Koditschek, D. E. 1989 (Scottsdale, AZ). The construction of analytic diffeomorphisms for exact robot navigation on star worlds. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 21–26.

Schwartz, J. T., and Sharir, M. 1983a. On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Comm. Pure Applied Math.* 36:345–398.

Schwartz, J. T., and Sharir, M. 1983b. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Adv. Applied Math.* 4:298–351.

Schwartz, J. T., and Sharir, M. 1983c. On the piano movers' problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *Int. J. Robot. Res.* 2(3):46–75.

Schwartz, J. T., and Sharir, M. 1988. A survey of motion planning and related geometric algorithms. *Artificial Intelligence* 37(1–3):157–169.

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

Sechen, C. 1988. *VLSI Placement and Global Routing Using Simulated Annealing*. Boston, MA: Kluwer Academic Publishers.

Serra, J. 1982. *Image Analysis and Mathematical Morphology*. New York: Academic Press.

Zhu, D., and Latombe, J. C. 1989. New heuristic algorithms for efficient hierarchical path planning. Rep. no. STAN-CS-89-1279. Dept. of Computer Science, Stanford University.