Numerical Potential Field Techniques for Robot Path Planning

Jérôme Barraquand, Bruno Langlois, and Jean-Claude Latombe

Abstract—**A new approach is proposed to robot path planning that consists of incrementally building a graph connecting the local minima of a potential field defined in the robot's configuration space and concurrently searching this graph until a goal configuration is attained. Unlike the so-called "global" path planning methods, this approach does not require an expensive computation step before the search for a path can actually start. On the other hand, it searches a graph that is usually much smaller than the graph searched by the so-called "local" methods. A collection of effective techniques to implement this approach is described. These techniques 1) construct "good" potential fields and 2) efficiently escape their local minima (i.e., efficiently build the local-minima graph). They are based on the use of multiscale pyramids of bitmap arrays for representing both the robot's workspace and configuration space. This distributed representation makes it possible to construct potential fields numerically, rather than analytically. A path planner based on these techniques has been implemented. Experiments with this planner show that it is both very fast and capable of handling many degrees of freedom. It has solved a variety of problems, some of which are far beyond the capabilities of previously developed planners.**

I. Introduction

In this paper we describe several *numerical potential field techniques* for robot motion planning. We have implemented these techniques in a path planner that turns out to be both very fast and capable of handling many degrees of freedom (DOF's). In particular, the planner has been able to plan the motions of mobile robots with 3 DOF's (two translations and one rotation) two orders of magnitude faster than most previously existing planners. It has also generated paths of robots with many DOF's in reasonable amount of time. For example, difficult paths for a nonserial manipulator with 10 joints (some revolute, some prismatic) were produced in 1 to 5 min.[1] Paths were also generated in a three-dimensional (3-D) workspace for a manipulator with 31 DOF's in about 15 min of computation time. These results are far beyond the capabilities of previously implemented planners.

Manuscript received October 12, 1989; revised July 9, 1991. This research was funded in part by DARPA contract DAAA21-89-C0002 (Army), in part by DARPA contract N00014-88-K-0620 (Office of Naval Research), in part by CIFE (Center for Integrated Facility Engineering), in part by CIS (Center of Integrated Systems), and in part by DEC (Digital Equipment Corporation).

The authors are with the Robotics Laboratory Computer Science Department, Stanford University, Stanford, CA 94305.

IEEE Log Number 9104559.

[1] Most of the experiments reported in this paper were carried out on a DEC 3100 MIPS-based workstation (14-mips processor) using simulated robots. The programs are all written in the C language.

The problem of generating collision-free paths has attracted considerable interest during the past years [29], [30]. By simplifying, we can say that two extreme approaches have been proposed, the "global" one and the "local" one. The global approach consists of first constructing a concise representation of the connectivity of the set of collision-free configurations of the robot in the form of a "connectivity graph" and then searching this graph for a path. Various techniques have been devised, e.g., exact cell decomposition [28], approximate cell decomposition [5], [10], [12], [32], retraction on a network of one-dimensional curves [23]. The local approach consists of searching a grid placed across the robot's configuration space [8]. Heuristics computed from partial information about the geometry of the configuration space are used to guide the search of the grid. Most proposed heuristics take the form of a potential field guiding the search along the flow of its negated gradient vector field [15], [16], [18].

The drawback of the global approach is that it requires an expensive precomputation step—the construction of the connectivity graph—before the search for a path can actually start. Since the computation time required by this construction is typically exponential in the dimension n of the configuration space (i.e., the number of DOF's), the approach is impractical even for reasonably small values of n. To our knowledge, no effective planner has been implemented using this approach with $n > 4$. Instead, the local approach requires no expensive precomputation step before starting the search of a path. Consequently, in favorable cases, it runs substantially faster than any global method. But, since 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 unfavorable cases. In order to deal with this difficulty, local methods need powerful heuristics to guide the search. But known such heuristics have the drawback of eventually leading the search to dead-ends, for instance local minima of the potential field. One may think of constructing a potential field with no other local minimum than the goal configuration (in the connected subset of the free space containing the goal configuration), but the analytical definition of such a potential turns out to be difficult (e.g., see [26]). Futhermore, even if a definition was available, it is likely that its computation would constitute an expensive precomputation step before path generation, similar in drawback to the construction of a connectivity graph.

There are many reasons motivating the development of fast path planners capable of dealing with many DOF's. For instance, a (semi-)autonomous robot will have to generate its

paths on-line, based on its current model of the world, and to react rapidly to contingencies. Although one may envision a robot that learns about its workspace and memorizes a variety of typical paths, it is by far more appropriate to have a fast "real-time" path planner.[2] Robots will typically combine one or several arms mounted on a mobile vehicle; for instance, space robots may consist of several manipulator arms attached to a free-flying platform [27]. Planning the paths of such robots will require to be able to handle many DOF's, specially if cooperation among them is needed. One may argue that, most of the time, at every instant, the planner has to worry only about a subset of these DOF's. But determining which DOF's are important at every instant is also part of planning and hence should be handled by the planner. Fast path planning capabilities may also be extremely useful for off-line programming of industrial and construction robots, and for the automatic generation of animated scenes on a graphic workstation [25].

We propose a new approach to path planning that attempts to combine the advantages of both the local approach (avoid expensive precomputation) and the global approach (search a concise graph). This approach consists of incrementally building a graph connecting the local minima of a potential function defined over the configuration space of the robot and concurrently searching this graph until a goal configuration is attained. The graph of local minima plays a role similar to that of the connectivity graph in the global approach. The major difference, however, is that it is constructed in an incremental fashion during the search. Hence, our approach does not require an expensive precomputation step, although it definitely searches a much smaller graph than the discretization grid placed across configuration space. In this paper we describe a collection of specific techniques that allows the engineer various implementations of this path planning approach.[3] The purpose of these techniques is to (1) construct "good" potential fields and (2) efficiently escape their local minima (i.e., efficiently build the local-minima graph). They are based on the use of multiscale pyramids of bitmap arrays for representing both the workspace and the configuration space of the robot. These representations allow us to construct potential fields *numerically*, rather than *analytically*, in relation to other efficient numerical techniques (e.g., collision checking, valley tracking).

In Section II we describe the hierarchical bitmap representations of the workspace and the configuration space of a robot. In Section III we propose several ways of constructing numerical potential fields in the robot's configuration space. In Sections IV–VII we present four path planning techniques—respectively called "best-first motion," "random motion," "valley-guided motion," and "constrained motion" techniques—which are based on different ways of escaping the local minima of these potential fields. We have implemented all these techniques and, for each one, we give experimental results. Each technique admits many straightforward variants and, in the course of our experimentation, we ran several variants successfully. Hence, within some limits, the algorithms presented in this paper may be adapted so that they better fit the characteristics of a specific application domain.

[2] To that respect, it is interesting to remember that not so many years ago, it was proposed to compute the inverse kinematics of a manipulator by storing the numerical values of the inverse Jacobian matrix of the forward kinematic map at many configurations of the manipulator. Today, the computation of the inverse kinematics is routinely done in real-time without having to store inverse Jacobian matrices.

[3] Some of these techniques were previously presented in [3], [4].

II. Bitmap Representations

A. Basic Terminology and Notations

We denote the robot by $\mathcal{A}$ and its workspace by $\mathcal{W}$. A Cartesian coordinate system, denoted by $\mathcal{F}_{\mathcal{W}}$, is embedded in $\mathcal{W}$. A *configuration* of $\mathcal{A}$ is a specification of the position of every point in $\mathcal{A}$ with respect to $\mathcal{F}_{\mathcal{W}}$. The *configuration space* of $\mathcal{A}$ is the space, denoted by $\mathcal{C}$, of all the possible configurations of $\mathcal{A}$. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration $\boldsymbol{q}$ is denoted by $\mathcal{A}(\boldsymbol{q})$.

The workspace contains a finite number of obstacles denoted by $\mathcal{B}_i$, with $i = 1, \cdots, r$. We denote the region $\mathcal{W} - \bigcup_{i=1}^{r} \mathcal{B}_i$ by $\mathcal{W}_{\text{empty}}$. Each obstacle $\mathcal{B}_i$ maps into $\mathcal{C}$ to the set $\mathcal{CB}_i$ of configurations, called *C-obstacle*, where the robot and the obstacles intersect, i.e., :

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

The subset of configurations where the robot and the obstacles have no intersection, that is:

$$\mathcal{C}_{\text{free}} = \mathcal{C} - \bigcup_{i=1}^{r} \mathcal{CB}_i,$$

is called the *free space*. A *collision-free path* (more simply, a *free path*) between two configurations $\boldsymbol{q}_{\text{init}}$ and $\boldsymbol{q}_{\text{goal}}$ is any continuous map $\tau : [0,1] \rightarrow \mathcal{C}_{\text{free}}$, such that $\tau(0) = \boldsymbol{q}_{\text{init}}$ and $\tau(1) = \boldsymbol{q}_{\text{goal}}$.

The configuration space $\mathcal{C}$ is an n-D manifold [19]. For example, a mobile robot can usually be modeled as a two-dimensional (2-D) object $\mathcal{A}$ that can both translate and rotate in the plane. Then, $\mathcal{C}$ is the 3-D manifold $\boldsymbol{R}^2 \times S^1$, where S^1 is the unit circle. In the case of a manipulator arm with n revolute joints, $\mathcal{C}$ is the n-D manifold $\left(S^1\right)^n$, or a subset of that space when the motion of each joint is limited by mechanical stops.

Throughout this paper, we represent a configuration $\boldsymbol{q}$ of $\mathcal{A}$ by a list of n independent parameters, $(q_1, \cdots, q_n)$, where n is the dimension of $\mathcal{C}$. This parameterization may have to be augmented with modular arithmetic for some of the angular parameters. Hence, we represent $\mathcal{C}$ as an n-D Cartesian space. For example, $\mathcal{C} = \boldsymbol{R}^2 \times S^1$ may be represented as $\boldsymbol{R}^2 \times \boldsymbol{R}/2\pi Z$. Any configuration $\boldsymbol{q}$ is then parameterized by (x, y, θ), where $(x, y) \in \boldsymbol{R}^2$ and $\theta \in [0, 2\pi)$ with modulo 2π arithmetic. Similarly, $\mathcal{C} = \left(S^1\right)^n$ may be represented as $(\boldsymbol{R}/2\pi Z)^n$. Any configuration is parameterized by $(q_1, \cdots, q_n)$, where $q_i \in [0, 2\pi)$ with modulo 2π arithmetic, for every $i \in [1, n]$.

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

$$\begin{array}{rccc} X & : & \mathcal{A} \times \mathcal{C} & \rightarrow & \mathcal{W} \\ & & (p, \boldsymbol{q}) & \mapsto & \boldsymbol{x} = X(p, \boldsymbol{q}) \end{array}$$

is called the *forward kinematic map*.

1) For every $\boldsymbol{x} \in \mathcal{GW}_{\text{empty}}$, set $d_1(\boldsymbol{x})$ to infinity (i.e., a large number M).
2) Scan $\mathcal{GW}$ and identify every point $\boldsymbol{x}$ such that $BM(\boldsymbol{x}) = 1$ and one of its neighbors is in $\mathcal{GW}_{\text{empty}}$. Set L_0 to the list of these points. Include the points forming the frame boundary of $\mathcal{GW}$ in L_0. For every point $\boldsymbol{x}$ in L_0, set $d_1(\boldsymbol{x})$ to 0.
3) For $i = 0, 1, 2, \cdots$, until L_i is empty, do: initialize L_{i+1} to the empty list; for every point $\boldsymbol{x}$ in L_i, for every neighbor $\boldsymbol{y}$ of $\boldsymbol{x}$ in $\mathcal{GW}_{\text{empty}}$, if $d_1(\boldsymbol{y}) = M$ then set $d_1(\boldsymbol{y})$ to $i+1$ and insert $\boldsymbol{y}$ at the end of L_{i+1}.

Fig. 1. Algorithm computing the d_1 map.

B. Workspace Bitmap

In the following, we make the reasonable assumption that $\mathcal{W}$ is a bounded subset of $\boldsymbol{R}^d$, with $d = 2$ or 3. $\mathcal{W}$ is modeled as a multiscale pyramid of d-D bitmap arrays. At each resolution level, the array is represented by a function:

$$\begin{array}{rccc} BM & : & \mathcal{W} & \rightarrow & \{1,0\} \\ & & \boldsymbol{x} & \mapsto & BM(\boldsymbol{x}) \end{array}$$

so that the subset of points $\boldsymbol{x}$ such that $BM(\boldsymbol{x}) = 1$ represents the workspace obstacles and the subset of points $\boldsymbol{x}$ such that $BM(\boldsymbol{x}) = 0$ represents the empty part of the workspace, i.e., $\mathcal{W}_{\text{empty}}$.

The distance between the centers of two consecutive cells in the same line or the same column of an array is constant. Hence, at any level of resolution, the midpoints of the cells of the bitmap array form a regular grid denoted by $\mathcal{GW}$. The subset of the grid where *BM* evaluates to 0 is denoted by $\mathcal{GW}_{\text{empty}}$. For any integer $k \in [1, r]$, the k-neighborhood of a point $\boldsymbol{x}$ in a grid of dimension r is defined as the set of points in the grid having at most k coordinates differing from those of $\boldsymbol{x}$, the amount of the difference (if any) along any axis being the discretization step along that axis. In $\mathcal{GW}$, we always use the 1-neighborhood, except if it is noticed otherwise (e.g., to track a curve or a surface). In a 2-D workspace grid, this means that each point $\boldsymbol{x} = (i, j) \in \mathcal{GW}$ has a maximum of 4 neighbors: $\{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$. In a 3-D workspace grid, each point has up to six neighbors.

Most of the experiments reported in this paper were carried out in a 2-D workspace ($d = 2$). In those experiments, the coarsest level of resolution in the pyramid was 16^2 and the finest 512^2. The workspace representation is given to the planner at the finest level of resolution. The other levels are automatically derived from it in a conservative fashion, so that $\{\boldsymbol{x} \in \mathcal{W} \ / \ BM(\boldsymbol{x}) = 0\} \subseteq \mathcal{W}_{\text{empty}}$. The scaling factor between two successive levels of resolution is 2, but a different factor could have been chosen.

In preparation to other algorithms to be described later, the planner computes the discretized L^1 distance $d_1(\boldsymbol{x})$ of every point $\boldsymbol{x} \in \mathcal{GW}_{\text{empty}}$ to the obstacles. This computation is performed according to the following "wavefront expansion" algorithm.[4] First, the points in the boundary of the obstacles are identified and the value of d_1 at these points is set to zero (the points in the contour of the bitmap are also included as boundary points). Next, the value of d_1 is set to 1 at all the neighbors of the boundary points in $\mathcal{GW}_{\text{empty}}$; to 2 at the neighbors of these new points (if not yet computed); etc. The algorithm terminates when all $\mathcal{GW}_{\text{empty}}$ has been explored. A more formal description of the algorithm is given in Fig. 1. The time complexity of this algorithm is linear in the number of points in the grid $\mathcal{GW}$.

[4] In this algorithm we normalize the distance between two neighbors in the grid to 1.

C. Configuration Space Bitmap

Since we discretize the workspace in a hierarchical fashion, it is consistent to discretize the configuration space $\mathcal{C}$ as well, by constructing another multiresolution grid pyramid. This pyramid has as many levels of resolution as the workspace pyramid and the resolutions at each level of the two pyramids are tightly related, as developed below. At each level of resolution, we denote by $\mathcal{GC}$ the grid representing the configuration space and by $\mathcal{GC}_{\text{free}}$ the subset of the grid lying in the free space $\mathcal{C}_{\text{free}}$.

Let us denote by δ the distance between two adjacent points in a workspace grid $\mathcal{GW}$. In the workspace pyramid, δ varies between $\delta_{\min}$ and $\delta_{\max}$. For example, let us assume that we have a workspace represented by a pyramid of arrays whose sizes are ranging between 16^2 and 512^2. If the distance is measured in percentage of the workspace diameter, we have $\delta_{\min} = 1/512$ and $\delta_{\max} = 1/16$.

Let us 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 (two in our implementation). In our example, the resolution r hence varies between $r_{\min} = -\log_2(\delta_{\max}) = 4$ and $r_{\max} = -\log_2(\delta_{\min}) = 9$.

Remember that we represent the configuration space $\mathcal{C}$ as a subset of an n-D Cartesian space with $\boldsymbol{q} = (q_1, \cdots, q_n)$. For any given workspace resolution, say $r = -\log_2(\delta)$, the corresponding resolution $R_i = -\log_2(\Delta_i)$ of the discretization of $\mathcal{C}$ along the q_i axis 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 workspace. 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, 2).

The relation between positions of robot points in the workspace and robot configurations is given by the forward kinematic map $X(p, \boldsymbol{q})$. For every point $p \in \mathcal{A}$, a modification of q_i by Δ_i results in a modification of each coordinate x_j of p ($j \in [1, d]$) by

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

If we impose all workspace motions to be less than $nbtol \times \delta$, we must have

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

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, let us consider a bar of length L moving freely in a 2-D workspace. We can represent a configuration of the bar by (x_G, y_G, θ), where x_G and y_G are the coordinates of the bar's midpoint and θ is the orientation of the bar. Let us normalize x_G, y_G, and θ so that their values range between 0 and 1. We have

$$J^{x_G}_{\text{sup}} = J^{y_G}_{\text{sup}} = 1$$

and

$$J^{\theta}_{\text{sup}} = \pi L.$$

If we set $nbtol = 2$, we get

$$R_{x_G} = R_{y_G} = r - 1$$

and

$$R_\theta = r + \log_2(\pi L) - 1.$$

This means that we need $2^1 = 2$ times less samples for x_G and y_G than for the workspace representation at each level of resolution, and $2/\pi L$ less samples for θ.

The planning techniques described below do not construct and/or use a complete representation of the configuration space grids, nor of the C-obstacles, since this would be in general too time and space consuming. Indeed, while the configuration space grids implicitly define the search space of the planner, in most practical cases, only a very small subset of the grids will be explored. It is clear, however, that in the worst case the grids would have to be exhaustively explored, requiring exponential time computation as any other existing path planning method.

III. Potential Field Construction

A. Overview

We describe below several ways of constructing numerical potential fields in the configuration space of a robot. They all consist of two major computing steps. First, potential fields, called *W-potentials*, are computed in the robot workspace $\mathcal{W}$. Each W-potential applies to a selected point in the robot, called *control point*, and pulls this point toward its goal position. The W-potentials at the various control points are then combined into another function, called *C-potential*, which is defined over the robot's configuration space.

More formally, let p_i, $i = 1, \cdots, s$, denote the control points in the robot. Each W-potential is a function:

$$\boldsymbol{V}_{p_i} : \boldsymbol{x} \in \mathcal{W}_{\text{empty}} \mapsto \boldsymbol{V}_{p_i}(\boldsymbol{x}) \in \boldsymbol{R}.$$

The C-potential is defined as

$$\boldsymbol{U}(\boldsymbol{q}) = G(\boldsymbol{V}_{p_1}(X(p_1, \boldsymbol{q})), \cdots, \boldsymbol{V}_{p_s}(X(p_s, \boldsymbol{q})))$$

where G is called the *arbitration* function.

The rationale behind this two-step computation is to use the workspace, whose dimension is small, as a source of low-cost information for constructing a "good" C-potential in the configuration space, whose dimension is usually big. As a matter of fact, the W-potentials computed by the algorithms of the next subsection are free of local minima. This allows us to construct C-potentials that avoid the robot to get trapped into simple cavities formed by the obstacles. However, the resulting C-potential may still have spurious local minima. One reason for that is that the information contained in the W-potentials does not completely characterize the connectivity of the free space; indeed, the W-potentials are computed for a finite (usually small) number of control points. The other reason is that the combination of several local-minima-free W-potentials does not guarantee that the C-potential is without local minima. Indeed, the effect of the C-potential is to concurrently attract the various control points, which are related by fixed or variable kinematic constraints, toward their respective goal positions. Thus, the control points are competing among themselves to attain these positions, and this competition may create undesired minima. The role of the function G is to arbitrate in this competition, hence the name of the function. Various arbitration functions are possible, resulting in more or less numerous, more or less deep local minima (Section III-C).

The only precomputation that is required by our planning approach is that of the W-potentials. The C-potential is defined by the arbitration function and is computed according to the needs of the planner during the search for a path. The precomputation of the W-potentials is necessary because we want these potentials to be free of local minima; hence, they cannot be computed locally. We could avoid this precomputation by not requiring the W-potentials to be local-minima-free, but we think that this condition is critical to the construction of a "good" C-potential. On the other hand, the computation of the W-potentials occurs in the workspace bitmap, whose dimension is only 2 or 3. As we will see below, it can be made very fast.

B. Computation of W-Potentials

We propose two techniques for computing local-minima-free W-potentials. Other similar techniques can easily be developed. In the rest of the paper, we will refer to the two W-potentials defined below as the "simple W-potential" and the "improved W-potential," respectively.

Simple W-Potential: Let p be a control point in the robot and $\boldsymbol{V}_p(\boldsymbol{x})$ the corresponding W-potential. We want $\boldsymbol{V}_p$ to be free of local minima, i.e., to have a single minimum at the goal position of p in the connected subset of $\mathcal{W}_{\text{empty}}$ containing this goal position. Indeed, as mentioned previously, we think that this is a major heuristic step toward the construction of a C-potential with few or small local minima.

Let $\boldsymbol{x}_{\text{goal}}$ be the goal position of p in $\mathcal{W}$. The simple W-potential[5] $\boldsymbol{V}_p$ is computed as follows. First, the value of $\boldsymbol{V}_p$ is set to 0 at $\boldsymbol{x}_{\text{goal}}$. Next, the value of $\boldsymbol{V}_p$ is set to 1 at the neighbors of $\boldsymbol{x}_{\text{goal}}$ in $\mathcal{GW}_{\text{empty}}$; to 2 at the neighbors of these new points (if not yet computed), etc. The algorithm terminates when all $\mathcal{GW}_{\text{empty}}$ has been explored. A more formal description of the algorithm is given in Fig. 2. At every point $\boldsymbol{x} \in \mathcal{GW}_{\text{empty}}$, the resulting W-potential is equal to the L^1 length of the minimal-length path connecting $\boldsymbol{x}$ to $\boldsymbol{x}_{\text{goal}}$ through $\mathcal{GW}_{\text{empty}}$. It has no other local minimum than $\boldsymbol{x}_{\text{goal}}$.

[5] A similar potential has previously been proposed in [13].

1) For every $\boldsymbol{x} \in \mathcal{GW}_{\text{empty}}$, set $\boldsymbol{V}_p(\boldsymbol{x})$ to infinity (i.e., a large number M).
2) Set $\boldsymbol{V}_p(\boldsymbol{x}_{\text{goal}})$ to 0 and the list L_0 to $(\boldsymbol{x}_{\text{goal}})$.
3) For $i = 0, 1, \cdots$, until L_i is empty, do: initialize L_{i+1} to the empty list; for every point $\boldsymbol{x}$ in L_i, for every neighbor $\boldsymbol{y}$ of $\boldsymbol{x}$ in $\mathcal{GW}_{\text{empty}}$, if $\boldsymbol{V}_p(\boldsymbol{y}) = M$ then set $\boldsymbol{V}_p(\boldsymbol{y})$ to $i+1$ and insert $\boldsymbol{y}$ at the end of L_{i+1}.

Fig. 2. Algorithm computing the simple W-potential.

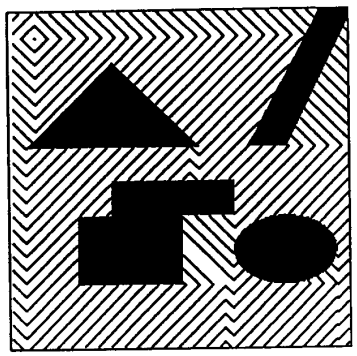


Fig. 3. Contours of the simple W-potential.

This algorithm is similar in structure to the algorithm computing d_1. Its time complexity is also linear in the number of points of $\mathcal{GW}$ and constant in the number and the shape of the obstacles. Fig. 3 displays contours of $\boldsymbol{V}_p$ for a 256^2 2-D workspace, with $\boldsymbol{x}_{\text{goal}}$ located in the upper-left corner of the bitmap array. (This computation takes a fraction of a second on a 14-mips workstation.)

A property of the $\boldsymbol{V}_p$ function computed as before is the following. By tracking the flow of the negated gradient vector field $-\vec{\nabla}\boldsymbol{V}_p$ from any initial point $\boldsymbol{x}_{\text{init}}$, we obtain a path that connects $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$. In addition, this path is the shortest path for the L^1 distance (at the resolution of the bitmap array). In a 3-D workspace, this computation may be preferable to exact methods, since 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 [6].

Notice that the algorithm given previously computes $\boldsymbol{V}_p$ only in the connected subset of $\mathcal{GW}_{\text{empty}}$ that contains the goal position $\boldsymbol{x}_{\text{goal}}$. Hence, after the algorithm has been executed, if the initial position $\boldsymbol{x}_{\text{init}}$ of p is such that $\boldsymbol{V}_p(\boldsymbol{x}_{\text{init}}) = M$, we can immediately return that there is no path connecting $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$. Step 2 of the algorithm can easily be modified to accommodate the case where the goal of p is a subset of $\mathcal{W}_{\text{empty}}$.

Improved W-Potential: A significant drawback of the simple W-potential is that it induces paths that typically graze obstacles in the workspace. In order to both reduce the risk of creating local minima in the C-potential function and enlarge the maneuvering space of the robot, so that, when local minima are created, they can be more easily escaped, we propose an improved W-potential. This potential, like the previous one, is free of local minima. But its negated gradient pulls the control point p toward its goal position along a path of nonminimal length, which stays as far away as possible from the obstacles.

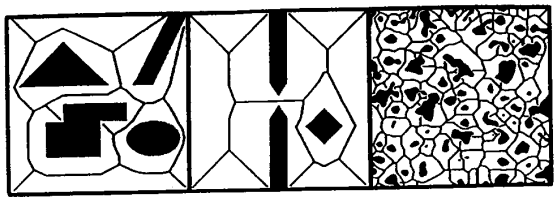


Fig. 4. Examples of workspace skeletons.

1) (*Initialization.*)
For every $\boldsymbol{x} \in \mathcal{GW}_{\text{empty}}$, set $\boldsymbol{V}_p(\boldsymbol{x})$ to infinity (i.e., a large number M).
2) (*Connection of the goal to the skeleton.*)
Set $\boldsymbol{x}$ to $\boldsymbol{x}_{\text{goal}}$. While $\boldsymbol{x} \notin \mathcal{S}$, include $\boldsymbol{x}$ in $\mathcal{S}$, select a neighbor $\boldsymbol{y}$ of $\boldsymbol{x}$ having the largest value of d_1, and set $\boldsymbol{x}$ to $\boldsymbol{y}$.
3) (*Computation of the W-potential in the augmented skeleton.*)
Set $\boldsymbol{V}_p(\boldsymbol{x}_{\text{goal}})$ to 0, Q to $(\boldsymbol{x}_{\text{goal}})$ and L_0 to the empty list. (*Q is a queue of points sorted by decreasing values of d_1.*)
Until Q is empty, do: remove the first element $\boldsymbol{x}$ of Q and insert it at the end of L_0; for every d-neighbor $\boldsymbol{y}$ of $\boldsymbol{x}$ in $\mathcal{S}$, if $\boldsymbol{V}_p(\boldsymbol{y}) = M$ then set $\boldsymbol{V}_p(\boldsymbol{y})$ to $\boldsymbol{V}_p(\boldsymbol{x}) + 1$ and insert $\boldsymbol{y}$ in Q. (*At the end of this step, L_0 contains all the points in $\mathcal{S}$ accessible from $\boldsymbol{x}_{\text{goal}}$.*)
4) (*Computation of the W-potential in the rest of $\mathcal{GW}_{\text{empty}}$.*)
For $i = 0, 1, 2, \cdots$, until L_i is empty, do: initialize L_{i+1} to the empty list; for every point $\boldsymbol{x}$ in L_i, for every neighbor $\boldsymbol{y}$ of $\boldsymbol{x}$ in $\mathcal{GW}_{\text{empty}}$, if $\boldsymbol{V}_p(\boldsymbol{y}) = M$ then set $\boldsymbol{V}_p(\boldsymbol{y})$ to $\boldsymbol{V}_p(\boldsymbol{x}) + 1$ and insert $\boldsymbol{y}$ at the end of L_{i+1}.

Fig. 5. Algorithm computing the improved W-potential.

This improved W-potential is computed in three stages.

First, the discrete L^1 distance $d_1(\boldsymbol{x})$ of every point $\boldsymbol{x} \in \mathcal{GW}_{\text{empty}}$ to the obstacles is computed and a $(d-1)$-D subset $\mathcal{S}$ of $\mathcal{GW}_{\text{empty}}$ is concurrently extracted. This subset is called the *workspace skeleton.* Second, the function $\boldsymbol{V}_p$ is computed in the skeleton $\mathcal{S}$. Third, $\boldsymbol{V}_p$ is computed in the rest of $\mathcal{GW}_{\text{empty}}$.

The distance $d_1(\boldsymbol{x})$ is computed using the algorithm given in Fig. 1. The workspace skeleton $\mathcal{S}$ is extracted during the computation of d_1 as the set of points where the "waves"—the lists L_i in the algorithm—issued from the boundary points of $\mathcal{GW}_{\text{empty}}$ meet. This is done by propagating not only the values of d_1, but also the points in the discretized boundary of $\mathcal{GW}_{\text{empty}}$ that are at the origin of the propagation. For each point $\boldsymbol{x}$ in $\mathcal{GW}_{\text{empty}}$, let $O(\boldsymbol{x})$ denote this point. Notice that $d_1(\boldsymbol{x})$ is the L^1 distance from $\boldsymbol{x}$ to $O(\boldsymbol{x})$. At step 2, we set $O(\boldsymbol{x})$ to $\boldsymbol{x}$ for every $\boldsymbol{x}$ in L_0. At step 3, let $\boldsymbol{x}$ be the point currently considered in L_i and $\boldsymbol{y}$ one of its neighbors. If $d_1(\boldsymbol{y}) = M$, we set $O(\boldsymbol{y})$ to $O(\boldsymbol{x})$; otherwise, if the L^1 distance between $O(\boldsymbol{y})$ and $O(\boldsymbol{x})$ is greater than some threshold (typically, 2) and $\boldsymbol{x} \notin \mathcal{S}$, we include $\boldsymbol{y}$ in $\mathcal{S}$.

Fig. 4 displays the skeletons computed for several 2-D workspaces. Each of these skeletons is a kind of generalized Voronoi diagram of $\mathcal{W}_{\text{empty}}$ [21] for the L^1 distance. It is also similar to the skeleton extracted from a region in a digitized image using techniques from *Image Analysis and Mathematical Morphology* [31].

The improved W-potential in $\mathcal{S}$ is computed as follows (see Steps 2 and 3 in Fig. 5). First, the goal position $\boldsymbol{x}_{\text{goal}}$ is connected to $\mathcal{S}$ by a path following the gradient of d_1 and

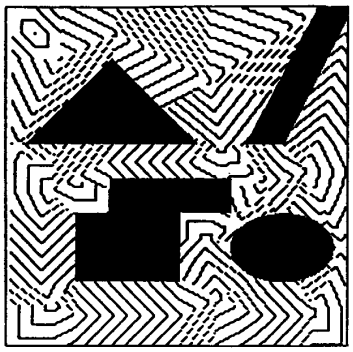


Fig. 6. Contours of improved W-potential.

the configurations along this path are included in $\mathcal{S}$ (Step 2). The new $\mathcal{S}$ is called the augmented skeleton. Then, the W-potential is computed in the subset of $\mathcal{S}$ accessible from $\boldsymbol{x}_{\text{goal}}$, using a wavefront expansion algorithm starting at $\boldsymbol{x}_{\text{goal}}$ and guided by the map d_1 as follows. The potential 0 is given to $\boldsymbol{x}_{\text{goal}}$, and $\boldsymbol{x}_{\text{goal}}$ is inserted in a queue Q whose elements are points of $\mathcal{S}$ sorted by decreasing values of d_1 (Q is initially empty). Next, until Q is empty, the first element of Q — call it $\boldsymbol{x}$ — is removed from Q; every d-neighbor[6] $\boldsymbol{y}$ of $\boldsymbol{x}$ in $\mathcal{S}$ whose potential has not been computed yet receives a potential value equal to $\boldsymbol{V}_p(\boldsymbol{x}) + 1$ and is inserted in Q. The algorithm terminates when Q is empty, i.e., all the points in the augmented skeleton accessible from $\boldsymbol{x}_{\text{goal}}$ have been given a potential value. The queue Q is represented as a height-balanced tree [1], so that both the insertion of a new point and the removal of a point maximizing d_1 are done in logarithmic time.

The W-potential at all the other points in $\mathcal{GW}_{\text{empty}}$ is computed using a wavefront expansion algorithm (see Step 4 in Fig.5) starting from $\mathcal{S}$, similar to the algorithms computing the d_1 map and the simple W-potential. The W-potential at each 1-neighbor $\boldsymbol{y}$ of every point $\boldsymbol{x}$ in $\mathcal{S}$ is set to $\boldsymbol{V}_p(\boldsymbol{x})+1$. The W-potential at the neighbors of these neighbors is iteratively incremented until all the points in $\mathcal{GW}_{\text{empty}}$ accessible from $\boldsymbol{x}_{\text{goal}}$ have been explored.

Fig. 6 shows the equipotential contours of the resulting W-potential for the same workspace as in Fig. 3. This W-potential generates no stable equilibrium state. Following its steepest descent from any initial position $\boldsymbol{x}_{\text{init}}$ produces a path that first connects $\boldsymbol{x}_{\text{init}}$ to $\mathcal{S}$, then stays as far as possible from the obstacles by following the safest curve of $\mathcal{S}$, and finally connects $\mathcal{S}$ to $\boldsymbol{x}_{\text{goal}}$. The previous algorithm only computes $\boldsymbol{V}_p$ over the connected subset of $\mathcal{GW}_{\text{empty}}$ that contains $\boldsymbol{x}_{\text{goal}}$. It can be adapted to the case where the goal of p is a region in $\mathcal{W}$.

The complexity of computing the improved W-potential is slightly higher than for the simple one. Let a be the number of points in the bitmap array, and b the number of points in the augmented skeleton $\mathcal{S}$. The complexity of the algorithm is $O(a + b \log b)$. For reasonable workspaces, however, we have $b \propto a^{d-1/d}$ since the skeleton is a (d-1)D subset of the workspace. For such workspaces, the complexity is $O(a + a^{d-1/d} \log a)$, hence is linear in a. For the 256^2 workspace of Fig. 6, the computation takes about 2 s (including the computation of d_1 and $\mathcal{S}$) on a 14-mips workstation.

Variants of the aforementioned W-potential are easy to define and compute. For example, in step 4 of the algorithm, we could compute $\boldsymbol{V}_p(\boldsymbol{y}) = \boldsymbol{V}_p(\boldsymbol{x}) + 1/d_1(\boldsymbol{x})$ and obtain a W-potential that becomes infinite on the boundary of $\mathcal{GW}_{\text{empty}}$. We will not detail the cosmetics of the computation of numerical W-potentials further. We just want to point out that W-potentials with various properties can be built within our framework.

C. Computation of C-Potentials

The C-potential $\boldsymbol{U}$ is computed as a combination:

$$\boldsymbol{U}(\boldsymbol{q}) = G(\boldsymbol{V}_{p_1}(X(p_1, \boldsymbol{q})), \cdots, \boldsymbol{V}_{p_s}(X(p_s, \boldsymbol{q})))$$

of the W-potentials $\boldsymbol{V}_{p_i}$, $i = 1, \cdots, s$, defined for s ($s \geq 1$) distinct control points p_i. This combination concurrently attracts the different points p_i toward their respective goal positions. $\boldsymbol{U}(\boldsymbol{q})$ attains its minimal value, i.e., 0, when all the control points are at their goal positions. At any other configuration, it is strictly positive.

The control points are those used to input the description of the goal configuration of the robot. (By definition, the robot is at a goal configuration whenever all the control points are at their goal locations.) The goal configuration may, or may not be uniquely defined. For instance, if $\mathcal{A}$ is a 2-D object that can both translate and rotate in the plane (3-D configuration space), the specification of the goal positions of two control points uniquely determines the goal configuration of the robot. If $\mathcal{A}$ is, say, a 10-DOF manipulator arm, specifying the desired positions of some points in the end-effector determines a goal region in configuration space. In all cases, we denote by $\mathcal{C}_{\text{goal}}$ the subset of goal configurations.

It is important that a path planner allows us to define a goal configuration by specifying the goal positions of a small number of points in the robot. Indeed, in many tasks, the goal configuration is incompletely determined by the task constraints. Arbitrarily selecting one configuration among the various possible ones may result in a more difficult, and perhaps even impossible, path planning problem. Furthermore, if the robot has many DOF's, specifying a unique goal configuration is known to be a difficult placement problem.

In order to precisely define the C-potential, we must now specify the arbitration function G. In most of the previous systems using the artificial potential field approach, G has been chosen as the sum of the W-potentials [15]:

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

This simple choice seems natural because it does not favor one control point over the others. However, precisely for that reason, conflicts among the points tend to be frequent, producing numerous local minima.

The choice of the function G is specially important because it highly influences the number and the depths of the

[6] Here, we exceptionally use the d-neighborhood in the workspace grid, so that we can track the augmented skeleton reliably.

local minima of U. With local-minima-free W-potentials, the workspace cavities do not directly create local minima of U. But, as mentioned previously, the concurrent attraction of the different control points toward their respective goal positions may create local minima. The function G determines how the competition between the different points is to be regulated.

The choice of G, which seems to minimize the number of local minima is

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

Indeed, this arbitration 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 the other points are not attracted toward their goal positions. A way to avoid this shortcoming is to add another term to the arbitration function, yielding:

$$G(y_1, \cdots, y_s) = \min_{i=1}^{i=s} y_i + \epsilon \max_{i=1}^{i=s} y_i \quad (1)$$

where ϵ is a small real number. In our experiments, we used $\epsilon = 0.1$. However, the best value of ϵ may depend on the robot.

Another choice for G is:

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

This arbitration function favors the attraction of the control point that is the furthest away from its goal (along the path determined by the W-potential). It tends to increase the number of competitions among the control points and, therefore, the number of local minima. However, it can be a good choice for robots with many DOF's. As a matter of fact, the number of local minima is not the only measure for the quality of the C-potential. Another critical factor is the depths of these minima. Indeed, in some cases, it may be preferable to have several small local minima, rather than a single, very deep one. This is specially true when the robot has many DOF's, since the number of discretized configurations contained in a local-minimum well of given depth increases exponentially with the dimension of the configuration space. Various experiments with the previous arbitration function indicate that in general it increases the number of local minima, but reduces their volumes.

Obviously, many other arbitration functions can be imagined. In some of our experiments, we used other C-potentials, as indicated further in this paper.

We now describe four path planning techniques incorporating the general planning approach presented in the introduction. All these techniques iteratively consider each level of resolution in the workspace and configuration space pyramids, from the coarsest to the finest. They terminate with success as soon as a path has been generated. They return failure if, after having considered the finest bitmaps, they still have not generated a path. Below, we only describe the algorithms executed at each level of resolution. We denote by $\mathcal{GW}$ and $\mathcal{GC}$ the current workspace and configuration space grids.

IV. Best-First Motion Technique

A. Description

We start with a very simple path planning algorithm.[7] This algorithm essentially performs a best-first search [22] of $\mathcal{GC}$ using the C-potential U as the cost function. It iteratively constructs a tree T whose nodes are configurations in $\mathcal{GC}_{\text{free}}$. The root of T is the initial configuration q_{init}. At every iteration, the algorithm examines the k-neighbors (for some $k \in [1, n]$) of the leaf of T, which has the smallest C-potential value, retains the neighbors in $\mathcal{GC}_{\text{free}}$ that are not already in T, and installs them in T as successors of the currently considered leaf. The algorithm terminates when the goal configuration q_{goal} has been attained (success) or when the subset of $\mathcal{GC}_{\text{free}}$ accessible from q_{init} has been fully explored (failure).

A configuration in $\mathcal{GC}$ has up to $2n$ 1-neighbors, $2n^2$ 2-neighborhoods, ..., and $3^n - 1$ n-neighbors. In our implementation of the best-first search algorithm, we use $k = n$. The size of the neighborhood thus increases exponentially with n, but as we will see below, the planning technique is intrinsically limited to problems involving few DOF's (typically, $n \leq 4$), so that the size of the maximal neighborhood remains reasonable.

As long as the algorithm does not reach a local minimum of the C-potential, the search reduces to following an approximation of the negated gradient of the C-potential (fastest descent procedure). When a local minimum is reached, the algorithm naturally fills up the local minimum well until a saddle point is reached. Then, it proceeds again along the negated gradient of U. It stops when a goal configuration ($U = 0$) is attained. Hence, the algorithm basically searches a graph connecting local minima among them, but this graph need not be explicitly represented. Two local minima are adjacent in the graph if they communicates through a saddle point. From a local minimum, the algorithm always goes to the adjacent local minimum that is attainable through the lowest saddle point. When two adjacent minima get filled up, they are implicitly merged into a single minimum.

The algorithm is resolution-complete, i.e., it is guaranteed to reach the goal in a finite amount of time whenever a solution path exists (at the resolution of $\mathcal{GC}$), or return failure when there is no solution. In most cases (virtually all reasonable cases), a very small subset of the grid $\mathcal{GC}$ has to be explored before the algorithm terminates. It is easy, however, to create a problem admitting no solution path, which requires most of $\mathcal{GC}_{\text{free}}$ to be explored before the planner gives up.

At this stage, there is an important aspect of the algorithm that must be made precise. Since the C-potential only depends on the distance of a few control points to the obstacles, best-first search does not guarantee that collisions of the robot with the obstacles will be completely avoided. Therefore, whenever the planner considers a new configuration in $\mathcal{GC}$, it should check that it lies in the free space. Because the planner does not represent the C-obstacles explicitly, the verification is done in the workspace as explained below.[8]

[7] This algorithm is described in more detail in [4].

[8] In a low-dimensional configuration space, as is the case here, an alternative is to precompute a bitmap representation of the C-obstacles, and then to check

As the workspace representation is distributed, it seems natural to represent the robot $\mathcal{A}$ in the same way, i.e., by a bitmap. If we were using a massively parallel computer, this representation would be the simplest and the most efficient to test collisions. For example, we could have one processor per point in the bitmap representation of $\mathcal{A}$; this processor would compute the position of the point in the workspace at the current configuration of $\mathcal{A}$ and determine whether it is contained in an obstacle, or not. However, as we implemented our planner on a sequential computer, we chose to represent the boundary of $\mathcal{A}$ algebraically. The implemented collision-checking technique is a classical "divide-and-conquer" algorithm, which consists of iteratively adjusting the number of points in the robot's boundary that are necessary to check collisions reliably. To illustrate this idea, let us consider a simple robot modeled as a single straight line segment of length L. The function d_1 has already been computed over $\mathcal{W}_{\text{empty}}$. Instead of checking the distance d_1 for each discretized point on the line segment modeling $\mathcal{A}$ and then calculating the minimum of all these distances, we first compute the distances $d_1(begin(\boldsymbol{q}))$ and $d_1(end(\boldsymbol{q}))$ of the two endpoints *begin* and *end* of the segment. If $\min\{d_1(begin(\boldsymbol{q})), d_1(end(\boldsymbol{q}))\} > L$, then we are certain that the robot does not collide with any obstacle. Otherwise, we cut the segment in two half-length segments, and we recursively apply the procedure to the two new segments. If the robot boundary is modeled by a polygon, the same procedure can be applied to each edge of the polygon. The procedure can also be generalized to higher-order shapes of the robot boundary such as circular and elliptical arcs, and to three-dimensional robots by triangulating their boundaries and recursively decomposing the generated triangles into smaller ones. When the robot is far from the obstacles, very few points on its boundary need be checked. When the robot gets closer to an obstacle, an increasing number of points (up to the current resolution of the workspace bitmap) in the boundary segment that is close to the obstacle have to be checked.

The search algorithm only considers the neighbors of a configuration that are collision-free. Hence, there may be two types of local minima: the natural minima of $\boldsymbol{U}$ (where the gradient is zero) and the minima located at the boundary of $\mathcal{GC}_{\text{free}}$ (where the gradient is not zero in general). When an obstacle is hit, the last configuration before the collision (i.e., a configuration in the boundary of $\mathcal{GC}_{\text{free}}$) is taken as the local minimum. Both types of minimum are escaped in the same fashion.

B. Experimental Results

We experimented with the best-first motion planning technique using a planar "mobile robot" with two DOF's of translation and one DOF of rotation, namely a long rectangular bar in a 2-D workspace. Fig. 7 shows an example of path generated by the planner. This path demonstrates the ability of the planning technique to produce complex maneuvers. The running time to produce the path was 1 s.[9] This is three orders of magnitude faster than the running times reported in [5] for similar (though apparently simpler) path planning problems. One order of magnitude is due to the faster computer we used. The other two are actually a product of our algorithm.

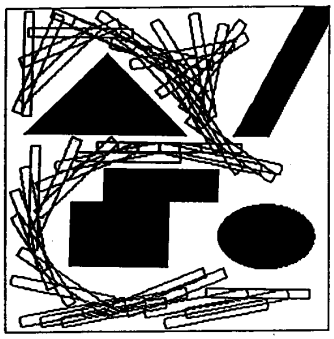


Fig. 7. Path generated for a 3-DOF mobile robot.

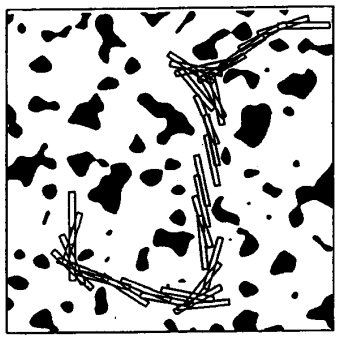


Fig. 8. Path generated among randomly distributed obstacles.

Fig. 8 shows another example of the planning abilities of the algorithm. For the same robot, a path was generated in less than 5 s within a 512^2 bitmap representing a workspace cluttered by more than 70 complex-shaped, randomly generated obstacles. This example would probably be very difficult to solve for a planner using a semi-algebraic representation of the workspace, e.g., an exact cell decomposition planning technique. It demonstrates the power of the "distributed" representations used in our planner relative to the "centralized" algebraic representations.

In both examples, the C-potential was computed by considering two control points located at the midpoints of the two small edges at the extremities of the bar, using the improved W-potential described in Section III-B. The C-potential was computed using the arbitration function defined by (1). The best-first motion technique also works in a satisfactory fashion, but is in average slower, when the simple W-potential is used in place of the improved one.

In practice, the previous planning technique is only applicable to robots with a small number n of DOF's—typically, $n \leq 4$. Indeed, the number of discrete configurations in a local-minimum well increases exponentially with the number

every explored configuration against this bitmap [20].

[9] Since there are many simple variants of the techniques presented in this paper, only the order of magnitude of the given execution times is actually pertinent. In fact, very recent improvements of our algorithm have made it possible to solve planning problems of the complexity of Fig. 7 within a few 1/10s of a second.

of DOF's. Filling up a well would be too time consuming for robots with many DOF's.

V. Random Motion Technique

A. Description

The planning algorithm[10] presented in this section essentially differs from the previous one in the way it escapes local minima. Rather than filling up each encountered local minimum, it applies a Monte-Carlo procedure, which consists of generating random motions until the minimum is escaped.

Starting from the initial configuration, the algorithm first searches the current grid $\mathcal{GC}$ in a best-first fashion. Thus, it follows the negated gradient of the C-potential, until it reaches a local minimum (call it $\boldsymbol{q}_{\text{loc}}$). We call such a motion a *gradient motion*. If $\boldsymbol{q}_{\text{loc}} \in \mathcal{C}_{\text{goal}}$, the planner returns the solution path generated. Otherwise, it attempts to escape the local minimum by generating several *random motions* from $\boldsymbol{q}_{\text{loc}}$. These random motions are symmetrical random walks, which are known to be approximations of Brownian motions. They are described in more detail in the next subsection.

The generation of gradient motions and the recognition of local minima are done using the n-neighborhood in $\mathcal{GC}$. As long as $n \leq 4$, this raises no difficulty. However, when n becomes too big (and we ran the random motion technique with robots having many DOF's), the size of the n-neighborhood becomes too large. In this case, at each step of the gradient motion, a limited (and relatively small) number of configurations in the n-neighborhood of the current configuration $\boldsymbol{q}$ are iteratively considered, until one is found to have a smaller C-potential than $\boldsymbol{q}$. Each iteration consists of randomly selecting a configuration $\boldsymbol{q}'$ in the n-neighborhood of $\boldsymbol{q}$ (using a uniform distribution law). If $U(\boldsymbol{q}') < U(\boldsymbol{q})$, $\boldsymbol{q}'$ is taken as the successor of $\boldsymbol{q}$ along the path of the gradient motion (hence, the motion may only follow a rough approximation of the negated gradient flow). 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 $\boldsymbol{q}$, $\boldsymbol{q}$ is treated as a local minimum. An alternative to this technique would be to use a smaller neighborhood, for instance the linear 1-neighborhood. In fact, we tried several alternatives, and the technique described previously gave the best experimental results. In particular, due to the crude discretization that it entails, a small neighborhood often resulted in the detection of fictitious local minima.

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 it is not in the goal region, it performs another set of random motions, etc. A local-minima graph is thus incrementally built, the path joining two "adjacent" local minima being the concatenation of a random motion and a gradient motion. Using the values of the C-potential at the local minima, a best-first search of this graph is performed until the goal configuration is reached or all the local minima in the graph have been expanded (i.e., a series of random motions have been executed from them). When a solution path is generated, it is smoothed using a classical variational calculus technique. An interesting property of this algorithm is that all the random motions starting from a given local minimum can be performed concurrently on a parallel machine, since there is no need for communication between the different processing units.

Many variants of the best-first search scheme are possible, and we experimented with several ones. The search technique that gave the best results consists of iteratively executing random motions starting at the current local minimum $\boldsymbol{q}_{\text{loc}}$ and, after each one, performing a gradient motion. If the attained local minimum has a lower C-potential than $\boldsymbol{q}_{\text{loc}}$, the iteration is stopped and the search proceeds from this new local minimum, by executing new random motions. The number of random motions starting at each local minimum is arbitrarily bounded. If no random motion from $\boldsymbol{q}_{\text{loc}}$ followed by a gradient motion leads the planner to a better minimum, $\boldsymbol{q}_{\text{loc}}$ is considered as a dead-end. The algorithm then backtracks to a point in the current path connecting the initial configuration to $\boldsymbol{q}_{\text{loc}}$. This point is selected randomly, using a uniform distribution law, over the set of points contained in the current path. The search for a path resumes at that point by executing a gradient motion. Since the backtracking point may belong to the subpath generated by a random motion, the gradient motion may reach local minimum not encountered yet. This search technique has the advantage of not requiring an explicit representation of the local minima graph to be maintained.

As 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 random 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 [4]. 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[11] [11], is a very weak one. Indeed, the totally uninformed algorithm, which executes a Brownian motion from $\boldsymbol{q}_{\text{init}}$ and terminates when it enters a small neighborhood of the goal configuration, is also probabilistically complete!

A weakness of the algorithm with the search technique described previously is that it may be unable to recognize that a planning problem admits no solution path. However, since the algorithm turns out to be relatively fast in most practical cases where there exist solution paths (see Section V-C), one way to deal with this drawback is to arbitrarily limit the computing time allocated to the planner. By simplifying a bit, we can say that we traded decidability and slow computation for semidecidability and fast computation.

B. Random Motions

A random motion is defined as a discrete symmetrical random walk in $\mathcal{GC}$. The duration t of the motion is a

[10] This algorithm is described in more detail in [4].

[11] For a comparison between simulated annealing algorithms and our planning algorithm, see [4].

certain number, say N, of time intervals of length δt. Hence, $t = N \times \delta t$. At each time interval, a motion step is executed, whose projection along each axis q_i, $i = 1, \cdots, n$, is randomly $\Delta_i \times \delta t$ or $-\Delta_i \times \delta t$, each with the same probability 0.5. Therefore, each motion step is independent of the previously executed steps. Such a random walk is known to converge almost surely toward a Brownian motion of duration t when δt tends toward 0 (and, hence, N tends toward infinity) [24]. In our implementation, we take $\delta t = 1$ and we assume that the Δ_i's are small enough to consider the random walk as a reasonably good approximation of the limit Brownian motion. We use this approximation and properties of Brownian motions to select the duration t, as described below.

Without loss of generality, we take the local minimum, $\boldsymbol{q}_{\text{loc}}$, as the origin of the coordinate system in $\mathcal{C}$. The configuration attained by a Brownian motion of duration t defined as before is the value of a random variable $\boldsymbol{Q}(t) = (Q_1(t), \cdots, Q_n(t))$ such that the density $p_i(q_i)$ of $Q_i(t)$, for any $i \in [1, n]$, is [24]:

$$p_i(q_i) = \frac{1}{\Delta_i \sqrt{2\pi t}} \exp(-\frac{q_i^2}{2\Delta_i^2 t}).$$

Hence, the standard deviation D_i of $Q_i(t)$ is: $D_i = \Delta_i \sqrt{t}$.

The Brownian motion is well-defined as long as it does not encounter any obstacle in configuration space. When it hits the boundary of a C-obstacle, the Brownian motion has to be adapted so that it stays in the free space. The classical adaptation of a Brownian motion when the space is bounded consists of reflecting the motion that would take place if there were no boundary, symmetrically to the tangent hyperplane of the boundary at the collision configuration. The mathematical consistency of this adaptation is discussed in [2]. Our planner, which does not construct an explicit representation of the C-obstacles, does not know the orientation of the tangent hyperplane at the collision configuration. Hence, whenever a motion step yields a collision, instead of bouncing symmetrically on the hyperplane tangent to the C-obstacle, the planner guesses another random step and substitutes it for the previous one. The collision is detected in the workspace using the divide-and-conquer technique described in Section IV-A.

The duration t of a random motion should not be too short, since the motion would have then little chance to escape the local minimum. On the other hand, if it is too long, the planner could waste time and loose the opportunity to use the C-potential gradient information when it becomes useful again. We define the *attraction radius* $A_{Ri}(\boldsymbol{q}_{\text{loc}})$ of a local minimum $\boldsymbol{q}_{\text{loc}}$ of $\boldsymbol{U}$ along the axis q_i as the distance along q_i between $\boldsymbol{q}_{\text{loc}}$ and the nearest saddle point of $\boldsymbol{U}$ in that direction. It is the minimum distance that the robot must travel in the direction q_i in order to escape the local minimum $\boldsymbol{q}_{\text{loc}}$. If we were able to estimate the statistics of A_{R_i}, the property $D_i = \Delta_i \sqrt{t}$ would give us a clue for computing t. Indeed, the duration of the Brownian motion could then be taken equal to the value of the function $t(\boldsymbol{q}_{\text{loc}})$ defined by

$$t(\boldsymbol{q}_{\text{loc}}) \approx \max_{i \in [1,n]} \left(\frac{A_{Ri}(\boldsymbol{q}_{\text{loc}})}{\Delta_i} \right)^2. \quad (3)$$

But, as we make no assumption on the obstacle distribution, we cannot infer any strong statistical property about $\boldsymbol{U}$ and the A_{R_i}'s. However, in general, we may consider that the distance A_{Ri} for each parameter q_i does not exceed the distance that would provoke a motion of the robot longer than the workspace diameter itself. Normalizing this diameter to 1, this assumption yields the following estimate of A_{Ri} for any local minimum $\boldsymbol{q}_{\text{loc}}$:

$$A_{Ri} \approx 1/J^i_{\text{sup}}. \quad (4)$$

This estimate, combined with the fact that A_{R_i} is strictly positive, leads us to treat A_{R_i} as a random variable with a truncated Laplace distribution of density:

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

(The truncated Laplace distribution is the less informed distribution, i.e., the one that maximizes entropy, for a positive random variable of given expected value.)

We have $\Delta_i \approx \delta / J^i_{\text{sup}}$ (see Section II-C), where δ is the distance between two adjacent points in $\mathcal{GW}$. Combining this formula with (3) and (4), we obtain

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

One could take t equal to the previous value. However, this choice would implicitly assume that all the attraction radii are the same, which is not the case. Using the previous distribution of A_{R_i}, we choose t as the value of a random variable T with the following density:

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

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

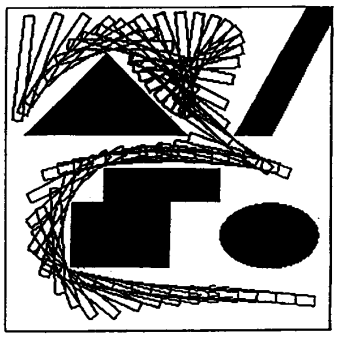


Fig. 9. Path generated for a 3-DOF robot.

C. Experimental Results

We have tested the random motion technique with several different robot stuctures. We now describe some of the obtained results. Since the planning technique contains several random components, the running time for the same planning problem is not constant and depends on the seed given to the random number generator. The times given below are only typical times. It is not unusual that two running times for the same example 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.

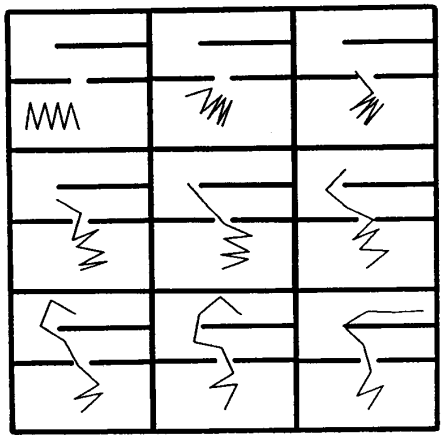


Fig. 10. Path generated for the 8-DOF manipulator.

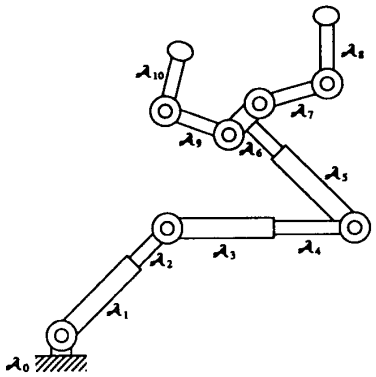


Fig. 11. Structure of the 10-DOF manipulator.

First, we applied the random motion planning technique to the 3-DOF robot example shown in Fig. 7 using the same C-potential as with the best-first motion algorithm. We got the path shown in Fig. 9 (after smoothing). The overall computation time was 10 s, which is about ten times slower than with the pure best-first motion planning technique. However, we think that a parallel implementation of the random motion technique would considerably reduce this computation time.

We experimented with the planner using an 8-DOF serial manipulator with 8 revolute joints and all its links modeled as line segments of the same length. Fig. 10 illustrates a path generated by the planner. The goal region is defined by the position of the endpoint p of the last link of the robot. The C-potential $\boldsymbol{U}$ was computed as the improved W-potential at p, i.e., $\boldsymbol{V}_p(X(p,\boldsymbol{q}))$. The path was generated in 2 min, covering all the computation performed between the input of the robot and workspace models and the display of the generated path.

We also experimented with the planner using a 10-DOF nonserial manipulator arm with three prismatic and seven revolute joints, as depicted in Fig. 11. Fig. 12 illustrates a path generated by the planner. The C-potential was computed by considering two control points located at the endpoints of the two kinematic chains, using the improved W-potential field and the arbitration function defined in (2). The computation time was 3 min. (In this example, the number of configurations in the grid $\mathcal{GC}$ in which the path was found is of the order of $100^{10} = 10^{20}$.)

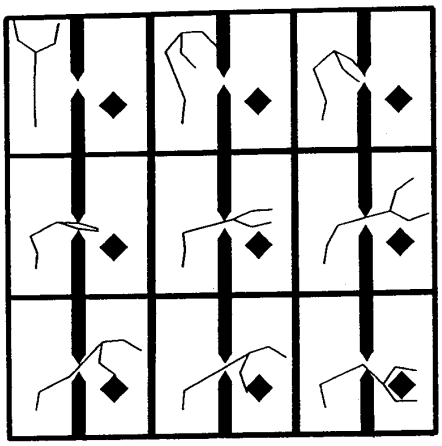


Fig. 12. Path generated for the 10-DOF manipulator.

Finally, we have run the path planning algorithm with a 31-DOF serial manipulator carrying a bar in a 3-D workspace. The manipulator consists of a sequence of 10 identical links, each with three DOF's (one translation and two rotations). The last module provides a third rotation. A path generated by the planner is illustrated in Fig. 13. The C-potential was computed by considering two control points located at the endpoints of the bar, using the improved W-potential field and the arbitration function of (2). The computation time was 15 min. The level of resolution of the workspace representation required to find this path was 128^3. (In this example, the size of $\mathcal{GC}$ is of the order of 10^{62} configurations.)

In all the previous multilink robot examples, we simulated mechanical stops by limiting the range of displacement of every joint. Except with the 31-DOF arm, we also prohibited collisions among the links, by checking collisions between any two links.

Other successful experiments were more recently conducted with two rigid objects moving in the same three-dimensional workspace (12 DOF's), with two manipulator arms carrying the same object (closed-loop kinematic chain of 7 DOF's), and with manipulators mounted on a free-flying platform in a three-dimensional workspace (10 to 20 DOF's).

The efficiency of the algorithm comes from the fact that a typical path planning problem has many solutions, so that a globally random process can find one, providing that it is sufficiently informed most of the time. This efficiency result is not new. Previous research has shown that Monte-Carlo techniques may successfully solve NP-hard problems. For example, Cerny [7] described an algorithm that generates suboptimal solutions to the traveling salesman problem for more than 10 000 towns. Kirkpatrick *et al.* [17] proposed an algorithm for the placement and the routing of VLSI chips, which is better than human experts. In both problems, the very large search space is associated with a large number of "good" suboptimal solutions.

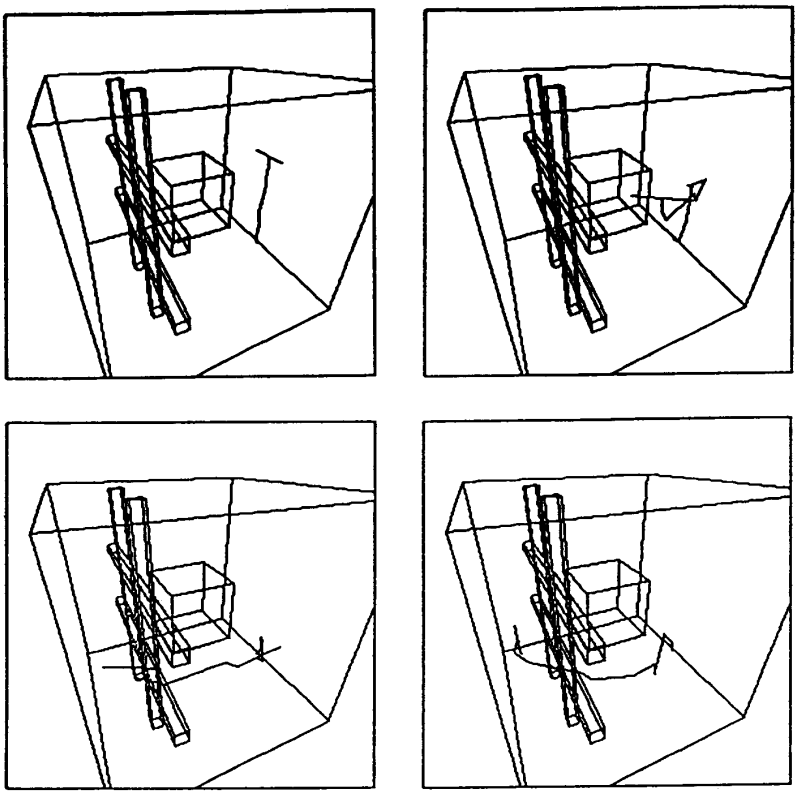


Fig. 13. Path generated for the 31-DOF manipulator.

VI. Valley-Guided Motion Technique

A. Description

The path planning algorithm[12] described in this section differs significantly from the previous two. It consists of searching the "valleys" of a C-potential U in $\mathcal{C}_{\text{free}}$. The set of valley points of U is called the *valley roadmap* and is denoted by $\mathcal{V}$.

The algorithm is similar in structure to a retraction algorithm [23]:

1) Given an initial and a goal configurations, q_{init} and q_{goal}, generate two new configurations, q_i and q_g, local minima of U, by following the negated gradient of U from q_{init} and q_{goal}, respectively.
2) Search the valley roadmap $\mathcal{V}$ for a path connecting q_i to q_g.
3) If step 2 terminates successfully, return the path obtained by concatenating the three paths joining q_{init} to q_i (gradient motion), q_i to q_g (valley-guided motion), and q_g to q_{goal} (gradient motion). Otherwise, return failure.

This algorithm does not require U to be minimum at the goal configuration. However, since q_{goal} is projected in $\mathcal{V}$ by a gradient motion, q_{goal} has to be uniquely defined, which was not the case with the previous two algorithms.[13] Ideally, U should be such that $\mathcal{V}$ is a 1-D subset of the free space $\mathcal{C}_{\text{free}}$. In addition, U should "represent the topology" of $\mathcal{C}_{\text{free}}$, i.e., q_i and q_g should be connected in $\mathcal{V}$ iff q_{init} and q_{goal} are connected in $\mathcal{C}_{\text{free}}$. In [3], we discuss the conditions under which a C-potential satisfies these two properties. Unfortunately, these conditions are quite involved and difficult to verify.

At step 2, $\mathcal{V}$ is searched in a depth-first manner. A decision is made at every crossroad using a simple heuristic function defined as another C-potential U_{heur}. This C-potential is constructed as described in Section III, i.e., by combining local-minima-free W-potentials. In our experiments, this heuristic potential dramatically reduced the computation times.

It now remains to specify how $\mathcal{V}$ is constructed. Unfortunately, as noticed in [3], there has not been much research on the concept of valley reported in the literature so far. One way to get a simple, but *ad hoc*, definition of valley points is to discretize the set of possible directions of the valleys. Namely, we force these directions to be the various q_i coordinate axes. Thus, we only have n possible valley directions. We define a configuration q to be a valley point along the q_i coordinate axis ($i \in [1, n]$) when all the values of the C-potential U

[12] This algorithm is described in more detail in [3].

[13] Actually, one could extend the previous algorithm to handle the case where a goal configuration is any configuration in a subset $\mathcal{C}_{\text{goal}}$ of the configuration space. But this would require each configuration in the discretized boundary of $\mathcal{C}_{\text{goal}}$ to be connected by a gradient motion to $\mathcal{V}$. Except for robots with few DOF's, this is likely to be impractical.

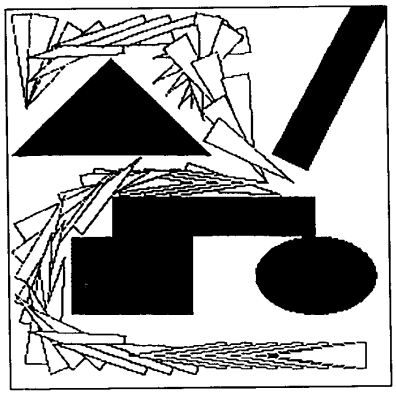


Fig. 14. Path generated for a 3-DOF mobile robot

at the $(2n-2)$ 1-neighbors in the hyperplane orthogonal to this direction are greater than $U(q)$. According to this simple definition, every local minimum of U is also a valley point of U (which is consistent with our intuitive notion of valley), so that the valley roadmap $\mathcal{V}$ can be regarded as a graph connecting the local minima of U. This graph has to be searched for a path between two minima, q_i and q_g.

In order to check whether a point q is a valley point or not, we thus use the simple following algorithm:

1. Compute $U(q)$.
2. Compute the $2n$ values of U at the 1-neighbors of q.
3. For each possible valley direction $i \in [1,n]$ do:
   Compare $U(q)$ to the $2n-2$ values of U at the 1-neighbors in the hyperplane orthogonal to the q_i axis. If $U(q)$ is smaller or equal to these $2n-2$ values, q is a valley point.

The time complexity of this algorithm is $O(n^2)$.

As nondegenerated valleys are 1-D, the safest neighborhood to track them is the n-neighborhood. Unfortunately, the cardinal of this neighborhood is 3^n-1, yielding an exponential time tracking algorithm. For instance, in a 10-D configuration space, using the 10-neighborhood entails performing about 60 000 valley checkings at every increment of the tracking process. Instead, the implemented planner uses the 2-neighborhood, whose size is quadratic in n, allowing each tracking increment to be performed in $O(n^4)$ time. This compromise between time and completeness appeared reasonable in most of the experiments we made. Complex paths have been generated, meaning that valleys can be tracked using the 2-neighborhood. Nevertheless, the choice of the 2-neighborhood is empirical.

Some valleys may be dead-ends. Valley tracking stops when the tracking algorithm is unable to proceed further. It also stops if the valley is running into an obstacle in configuration space (this can be detected by using the collision checking technique described in Section IV-A). In both cases, the planning algorithm backtracks to a crossroad point in the valley roadmap.

B. Experimentation

We experimented with the valley-guided motion technique on a variety of path planning problems.[14] The best results were obtained with the C-potential U defined as follows [3]:

$$U(q) = \sigma \log\Big(\sum_{i=1}^{s} \exp(V_{p_i}(X(p_i,q))/\sigma)\Big),$$

where σ is a small number and the p_i's $(i=1,\cdots,s)$ are the control points.[15] Each W-potential V_{p_i} is computed using the simple definition of Section III-B.

The heuristic potential U_{heur} used in our experiments is defined as the sum of the simple W-potentials computed at a few control points.

We applied the algorithm to 3-DOF mobile robot problems. An example of generated path for a triangular robot is shown in Fig. 14. This example was run using 15 control points equally distributed in the triangle boundary. We ran the algorithm with and without the heuristic potential U_{heur}. When used, U_{heur} was computed by considering a single control point. In both cases, the program found a solution path. However, the heuristic potential considerably reduced the overall computational time. In the example of Fig. 14, the running time was reduced from 2 min to less than 30 s.

We also applied the algorithm to the 10-DOF robot shown in Fig. 11. Fig. 15 shows a path generated by the planner. In this example, the C-potential U was computed using 70 control points equally distributed in the robot and the heuristic potential U_{heur} using two control points located at the endpoints of the two kinematic chains. The use of the heuristic potential was necessary to obtain a path in a reasonable amount of time. (In a 10-D configuration space, each point has 200 2-neighbors. The computation of the next point along a valley takes a few seconds.) The path was generated in less than 1 h.

Notice that the path planning problem shown in Fig. 15 is significantly simpler than the problem shown in Fig. 12. There is more space for the robot to "maneuver." (In Fig. 12, the hole in the wall is narrower and the square object is closer to the wall.) The previous algorithm failed to solve the problem of Fig. 12. (More precisely, we stopped the search after a few hours of computation, but the planner did not explicitly return a failure message.)

The aforemetioned planning technique is slower than the best-first and random motion techniques for robots with few DOF's. It is also slower and definitely less reliable than the random motion technique for robots with many DOF's. The failures of the algorithm may have several causes: the search of the valley roadmap may require exponential-time computation, the valley roadmap may imperfectly represent the connectivity of the free space, it may be locally degenerated (i.e., may not be 1-D), the 2-neighborhood may be insufficient to reliably

[14] We developed the valley-guided motion technique before the other three techniques presented in this paper. It has been implemented on a SUN III workstation, and the experiments described below were carried out on this machine.

[15] The expression defining $U(q)$ is a discrete approximation of $\sigma \log(\int_A \exp(V_p(X(p,q))/\sigma)dp)$, which decreases when the distance of the robot to the obstacles increases and tends toward infinity when this distance tends toward 0. The expression defining $U(q)$ also uniformly converges toward the simpler expression $\sup_{i\in[1,s]} V_p(X(p,q))$ when $\sigma \to 0$. The latter expression, however, is highly degenerated for manipulator arms and produces flat valleys that are difficult to track. Therefore, it cannot be used reliably by the valley-guided planning technique.

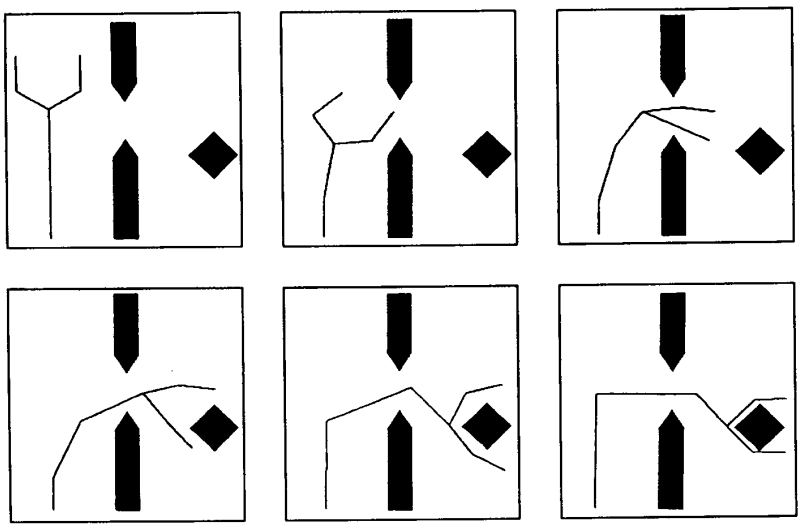


Fig. 15. Path generated for the 10-DOF manipulator

track the valleys, etc . . . We think that these causes might be partly remedied by investigating the concept of valley further (see [3]).

Notice that using the concept of valley might also be useful to improve the random motion technique. Indeed, when a gradient motion reaches a local minimum, that algorithm executes a series of totally uninformed random motions. Instead, valley directions could be extracted around the encountered minimum and the probability laws governing the random motions could be biased so that these motions tend to stay closer to the valley directions than uninformed motions.

VII. Constrained Motion Technique

A. Description

The constrained motion technique rests on a combination of ideas already present in the previous three techniques. On the one hand, like the best-first and random motion techniques, it makes use of a C-potential that is globally minimal (null) in the subset $\mathcal{C}_{\text{goal}}$ of goal configurations and, except when it tries to escape a local minimum, it follows the negated gradient of this C-potential. On the other hand, it uses a concept similar to the notion of valley for escaping encountered local minima.

Starting from the given initial configuration, the algorithm follows the flow of the negated gradient field of the C-potential U until a local minimum, $\boldsymbol{q}_{\text{loc}}$, is attained (gradient motion). If $\boldsymbol{q}_{\text{loc}} \in \mathcal{C}_{\text{goal}}$, the problem is solved. Otherwise, the planner executes a series of *constrained motions*. A constrained motion, denoted by $M_i^+(\boldsymbol{q}_{\text{loc}})$ (resp. $M_i^-(\boldsymbol{q}_{\text{loc}})$), with $i \in [1, n]$, consists of iteratively forcing the ith configuration space coordinate, q_i, to increase (resp. decrease) by the increment Δ_i of the grid $\mathcal{GC}$, until a saddle point of the local-minimum well is reached. Since $\boldsymbol{q}_{\text{loc}}$ is a local minimum of the C-potential, this means that the ith DOF is required to move in the opposite direction of the direction suggested by the C-potential. At each iteration of the constrained motion, the remaining $n-1$ coordinates q_j, $j \neq i$, are selected according to a best-first rule. Hence, if $(q_1, \cdots, q_{i-1}, q_i, q_{i+1}, \cdots q_n)$ is the current configuration, its successor minimizes the C-potential over the set consisting of the configuration $(q_1, \cdots, q_{i-1}, q_i', q_{i+1}, \cdots, q_n)$, where $q_i' = q_i + \Delta_i$ (if $M_i^+(\boldsymbol{q}_{\text{loc}})$ is being executed) or $q_i - \Delta_i$ (if $M_i^-(\boldsymbol{q}_{\text{loc}})$ is being executed), and its 1-neighbors in the hyperplane perpendicular to the q_i axis. The motion thus tracks a kind of valley in the $(n-1)$-dimensional subspace orthogonal to the q_i axis. Hopefully, it ultimately reaches a saddle point of the local-minimum well. The planner recognizes such an event when the C-potential decreases again. Then it terminates the constrained motion and executes another gradient motion.

Assume for an instant that every constrained motion attains a saddle point of the local-minimum well. By interweaving gradient and constrained motions, the planner constructs a graph of local minima. Each minimum, after it has been fully expanded, has $2n$ immediate successors in the graph and is connected to each of them by the concatenation of a constrained motion and a gradient motion. Whenever the planner attains a minimum, it may check if it is a new one, by comparing its coordinates with those of previously generated minima. The search of the local-minima graph may be conducted in a best-first fashion by iteratively selecting a pending local minimum having the smallest value of the C-potential and generating its successors. In order to avoid the systematic generation of all the successors of every selected pending minimum, we have implemented a variant of this strategy. At each iteration, the implemented algorithm selects a minimum $\boldsymbol{q}_{\text{loc}}$ having the smallest value of U among those whose successors have not all been constructed yet, and it generates a new successor $\boldsymbol{q}'_{\text{loc}}$ of $\boldsymbol{q}_{\text{loc}}$. At the next iteration, if $U(\boldsymbol{q}'_{\text{loc}}) < U(\boldsymbol{q}_{\text{loc}})$, the algorithm naturally selects $\boldsymbol{q}'_{\text{loc}}$ and generates one of its successors; otherwise, it constructs another successor of $\boldsymbol{q}_{\text{loc}}$, if any.

However, a constrained motion $M_i^+(\boldsymbol{q}_{\text{loc}})$ (or $M_i^-(\boldsymbol{q}_{\text{loc}})$) may not terminate at a saddle point. Instead, it may hit

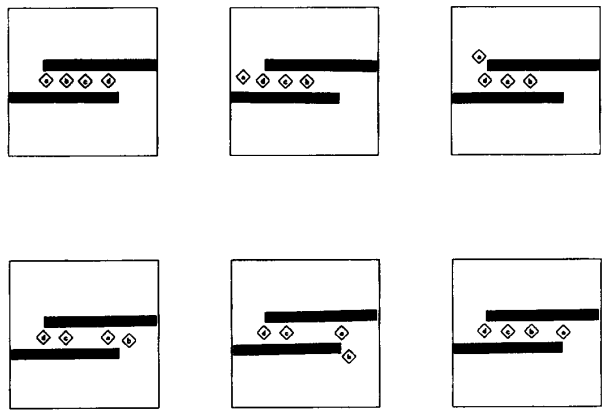


Fig. 16. Coordinated motion of four mobile robots.

an obstacle. In that case, the implemented algorithm merely considers that the local minimum has no successor "in the direction of" increasing (or decreasing) q_i. An alternative would be to treat the last configuration attained by the constrained motion before the collision as a successor of $\boldsymbol{q}_{\text{loc}}$. This successor would not be a local minimum, but it would nevertheless be included as a node of the local-minima graph and then treated as any other node, except that the constrained motions executed from it should not be commanded along the q_i axis. It is rather easy to construct examples where the constrained motion technique succeeds only if these nonlocal-minimum configurations are inserted in the search graph.

Without further assumptions on the C-potential, the previous path planning technique is not complete. Nevertheless, as shown below, it produces interesting experimental results.

B. Experimentation

We initially developed the constrained motion technique with the goal of solving planning problems involving the coordinated motions of several mobile robots in a workspace made of narrow corridors. Indeed, consider the case where a gradient motion leads two mobile robots to roll in opposite directions in a narrow corridor where they cannot pass each other. At some point, they cannot progress further toward their respective goals without colliding. Then, the gradient motion has reached a local minimum of the C-potential defined in the product configuration space of the two robots. A constrained motion corresponds to one of the robots moving backward, the other robot proceeding along the negated gradient of the C-potential, until there is sufficient room for the robots to pass each other.

We applied the technique to plan the motion of four robots in a workspace containing two parallel walls forming a narrow corridor (see Fig. 16). Each robot is a L^1-disc of radius a denoted by $\mathcal{A}_i$, $i = 1, \cdots, 4$, which can only translate in the plane. Its configuration is defined as the coordinates (x_i, y_i) of the center p_i of the square in a workspace coordinate frame.[16] The configuration space of the whole robot system is an 8-D space with each configuration represented by $(x_1, y_1, \cdots, x_4, y_4)$. In the example shown in Fig.16, the robots initially stand in the corridor made by the two walls. In the goal configuration, which is uniquely defined, the robots also stands in the corridor but in the reversed order. Fig.16 shows various intermediate configurations along the path generated by the planner. In this example, the C-potential was constructed as the sum $\sum_{i=1}^{4} \boldsymbol{V}_{p_i}(X(p_i, \boldsymbol{q}))$ of the simple W-potentials computed at the centers of the robots. The collision-checking technique of Section IV-A reduces to the comparison of the L^1 distance $d_1(X(p_i, \boldsymbol{q}))$ to the L^1 radius a. In the same fashion, the collisions between two robots $\mathcal{A}_i$ and $\mathcal{A}_j$ were checked by comparing the L^1 distance between the points p_i and p_j and $2a$. The total planning time was 30 s.

Fig. 17 illustrates another example to which we applied the planning technique. Ten mobile robots, identical to the robots of the previous example, operate in a workspace consisting of two rooms connected by corridors. No two robots can pass each other in any of the corridors, except in the lateral corridors at the four corners, the two intersections with the central corridor, and the locations facing the two doors. The whole robot system has a 20-D configuration space. The initial configuration is shown in the upper-left corner of the figure and the goal configuration in the lower-right corner. The figure displays several intermediate configurations along the path generated by the planner. We first ran the planner using the C-potential $\sum_{i=1}^{10} \boldsymbol{V}_{p_i}(X(p_i, \boldsymbol{q}))$, as in the previous example. The planner was able to construct a path, but this path traversed several tens of local minima. Many of these minima were due to the fact that the robots were moving in a de-organized fashion yielding intricate traffic jams. Then, we ran the planner using a slightly different C-potential obtained by adding to the previous C-potential a term that applies to each robot a force perpendicular to its current direction of motion and pointing toward its right. This additional field, which is analogous to a "magnetic field," essentially reproduces the "drive on the right" traffic rule. In our experiments, it led the various robots to move in a more organized fashion. Using this C-potential, the planner solved the problem shown in Fig. 17 by generating a path traversing only a few local minima. The total path was generated in about 30 s, which is quite satisfactory given the high number of robots.

Since multirobot path planning problems typically involves many DOF's, traditional global path planning methods are often impractical. More specific approaches, e.g., "prioritizing" and "velocity tuning" have been proposed for such problems. The prioritizing approach [9] consists of considering the robots in sequence. When the path of a robot has been constructed, this robot becomes a mobile obstacle for the robots whose

[16] The orientation of the coordinate axes with respect to the walls has no major impact on the operations of the planner.

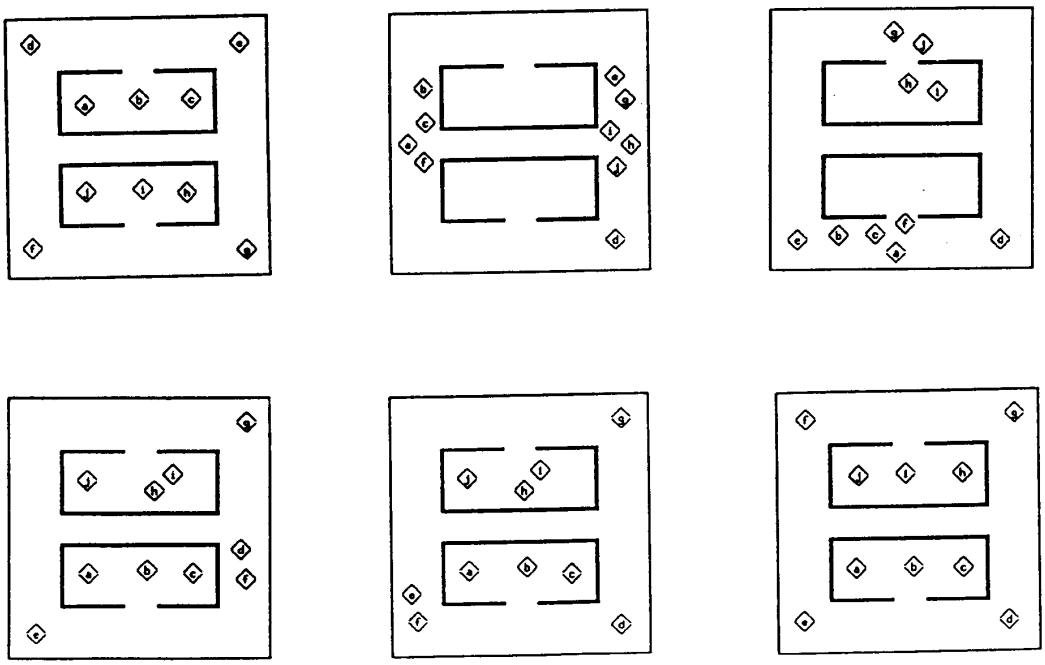


Fig. 17. Coordinated motion of ten mobile robots.

motions have not yet been planned. The velocity tuning approach [14] consists of first planning the motion of each robot as if it was the only robot in the workspace and then tuning the velocities so that no two robots collide. Hence, both approaches first break the planning problem for q robots into q simpler planning problems, each involving a single robot, and then consider the interactions among their solutions. These approaches are unable to solve multirobot problems involving tight interactions among the robots, such as the two problems shown previously.

Although the constrained motion technique seems particularly suitable for planning coordinated motions of multiple mobile robots, we have also experimented with it using the 8-DOF manipulator arm of Fig.10 with the same C-potential as in the experiment reported in Section V-C. In the example of Fig. 10, a solution path was constructed in about 3 min, which is comparable to the result obtained with the random motion technique. However, our experimentation with multijoint manipulator arms has been limited, and the reliability of the constrained motion technique remains to be verified for such robots.

VIII. Conclusion

In this paper we presented a collection of numerical potential field techniques for robot path planning. All these techniques apply the same general approach. They construct a potential field over the robot's configuration space, build a graph connecting the local minima of this potential, and search this graph. The graph is built incrementally and searched as it is built. The only precomputation step is aimed at the construction of the local-minimum-free W-potentials for the various control points in the robot. This step could be avoided, but it is an important one, since it allows us to construct "good" C-potentials. In addition, this precomputation occurs in a space of fixed and small dimension $d = 2$ or 3. In virtually all practical cases, the size of the graph of the local minima is reasonably small, resulting in efficient path searching.

We proposed four techniques for constructing the local-minima graph. These techniques are based on different ways of escaping the encountered local minima of the C-potential. The best-first motion technique is very simple and gives excellent results for robots with few DOF's (less than 5). However, it is impractical for robots with many DOF's. The random motion technique gives very good results for robots with few and many DOF's. In addition, it is highly parallelizable. The valley-guided motion technique gave significantly inferior results. However, it embeds ideas that one might improve in the future by exploring the concept of valleys more thoroughly. The constrained motion technique, although incomplete, gave very good experimental results, specially for planning the coordinated motions of several simple mobile robots. The principles underlying these four planning techniques are somewhat independent of the representation of the robot's workspace and configuration space. However, their efficient implementation was made possible by the use of hierarchical bitmap representations. The planner implementing these techniques was able to solve path planning problems, whose complexity measured by the number of DOF's or the number of obstacles is far beyond the capabilities of previously implemented planners. For simpler problems, our planner is several orders of magnitude faster than most previous planners.

We feel that the techniques presented in this paper and the experimental results obtained with them make it possible

to realistically envision the development of "real-time" path planners. By "real-time," we mean that the planners would be able to produce paths in a very small amount of time (say, a fraction of a second) in almost all practical situations. The construction of such a real-time planner will probably require the use of some dedicated hardware and parallel computing architecture, leading to the notion of "motion engine" just like there exist "geometric engines" for performing graphic operations (e.g., hidden line removal). We believe that the availability of a real-time motion engine would open new perspectives on some important issues in industrial robot programming and autonomous robot control, and enable the construction of efficient robot systems operating in partially known and dynamically changing workspaces. It could also provide a useful tool for automatically generating animated graphic images. Among the four techniques presented in this paper, the random motion technique seems to provide the best combination of qualities (time efficiency, generality, reliability). Since it is also highly parallelizable, it is probably the best candidate for developing such a motion engine. The random motion technique described in the paper makes use of uninformed probability distribution laws. If a very fast implementation of the technique was available, we could envision the inclusion of a learning module. In a specific application domain, this module would collect various pertinent statistics, for instance statistics about the distribution and the radii of the minimum wells, and refine the probability laws used by the planner, yielding even faster and more stable response times.

REFERENCES

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

[2] R. F. Anderson and S. Orey, "Small random perturbations of dynamical systems with reflecting boundary," *Nagoya Math. J.*, vol. 60, pp. 189–216, 1976.

[3] J. Barraquand, B. Langlois, and J. C. Latombe, "Robot motion planning with many degrees of freedom and dynamic constraints," in *Robotics Research*, H. Miura and S. Arimoto, Eds. Cambridge, MA: MIT Press, 1990, pp. 435–444.

[4] J. Barraquand and J. C. Latombe, *Robot Motion Planning: A Distributed Representation Approach*, Report no. STAN-CS-89–1257, Department of Computer Science, Stanford University, 1989, to appear in *The Int. J. Robotics Research*.

[5] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for find-path with rotation," *IEEE Trans. Syst., Man Cybern.*, vol. 15, pp. 224–233, 1985.

[6] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge: MIT Press, 1988.

[7] V. Cerny, "Thermodynamical Approach to the traveling salesman problem: An efficient simulation algorithm," *J. Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[8] B. R. Donald, "A search algorithm for motion planning with six degrees of freedom," *Artificial Intell.*, vol. 31, no. 3, pp. 295–353, 1987.

[9] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," A. I. Memo no. 883, Artificial Intell. Lab., MIT, Cambridge, 1986.

[10] B. Faverjon, "Object level programming of industrial robots," in *Proc. IEEE Int. Conf. Robotics Automation*, San Francisco, CA, 1986, pp. 1406–1412.

[11] D. Geman and S. Geman, "Stochastic relaxation, Gibbs Distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, pp. 721–741, 1984.

[12] L. Gouzènes, "Strategies for solving collision-free trajectories problems for mobile and manipulator robots," *The Int. J. Robotics Res.*, vol. 3, no. 4, pp. 51–65, 1984.

[13] R. A. Jarvis and J. C. Byrne, "An automated guided vehicle with map building and path finding capabilities," in *Robotics Res.*, R. Bolles and B. Roth, Eds. Cambridge: MIT Press, 1988, pp. 497–504.

[14] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: Path velocity decomposition," *The Int. J. Robotics Res.*, vol. 5, no. 1, pp. 72–89, 1986.

[15] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The Int. J. Robotics Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[16] P. Khosla and R. Volpe, "Superquadric artificial potentials for obstacle avoidance and approach," in *Proc. IEEE Int. Conf. Robotics Automat.*, Philadelphia, 1988, pp. 1178–1784.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[18] D. E. Koditschek, "Robot planning and control via potential functions," in *The Robotics Review 1*, O. Khatib, J. J. Craig, and T. Lozano-Pérez, Eds. Cambridge: MIT Press, 1989, pp. 349–367.

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

[20] J. C. Latombe, "A fast path planner for a car-like indoor mobile robot," in *Proc. Ninth National Conf. Artificial Intell.*, Anaheim, July 1991.

[21] D. T. Lee and R. L. Drysdale, "Generalization of Voronoi diagrams in the plane," *SIAM J. Computing*, vol. 10, pp. 73–87, 1981.

[22] N. J. Nilsson *Principles of Artificial Intelligence*. Palo Alto, CA: Morgan Kaufmann, 1981.

[23] C. Ó'Dúnlaing, M. Sharir, and C. K. Yap, "Retraction: A new approach to motion planning," in *Proc. 15th ACM Symp. Theory of Computing*, Boston, 1983, pp. 207–220.

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

[25] C. Puech *et al.*, *Motion Synthesis, Planning and Control*, Tutorial, SIGGRAPH'91, Las Vegas, 1991.

[26] E. Rimon and D. E. Koditschek, "The construction of analytic diffeomorphisms for exact robot navigation on star worlds," in *Proc. IEEE Int. Conf. Robotics Automat.*, Scottsdale, AZ, 1989, pp. 21–26.

[27] S. A. Schneider *Experiments in the Strategic and Dynamic Control of Cooperating Manipulators*, Ph.D. dissertation, Dept. Elec. Eng., Stanford Univ., Stanford, CA, 1989.

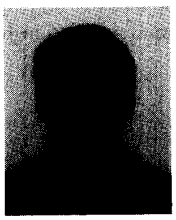
[28] J. T. Schwartz and M. Sharir, "On the 'piano movers' problem—II: General techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, pp. 298–351, 1983.

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

[30] J. T. Schwartz and C. K. Yap, *Algorithmic and Geometric Aspects of Robotics*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1987.

[31] J. Serra, *Image Analysis and Mathematical Morphology*. London: Academic Press, 1982.

[32] D. Zhu and J. C. Latombe, "New heuristic for efficient hierarchical path planning," *IEEE Trans. Robotics Automat.*, vol. 7, pp. 9–20, Feb. 1991.

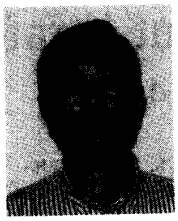


Jerome Barraquand graduated in 1984 from Ecole Polytechnique (MS), Paris, France. From 1984 to 1987, he worked with the Computer Vision Group at the Institut National de Recherche en Informatique et Automatique (INRIA), Sophia-Antipolis, France. He developed 3-D reconstruction techniques for the interpretation of seismic data. He received the Ph.D. degree in computer science in 1988 from University of Nice-Sophia-Antipolis, France.

From 1988 to August 1990, he was a Research Associate with the Computer Science Department at Stanford University, CA. There he worked on computational geometry issues related to robotics, with special interest in the problem of planning collision-free paths for mechanical systems with many degrees of freedom. He joined the Paris Research Laboratory, Digital Equipment Corporation, Paris, France, in September 1990. His current research interests include applications of spatial reasoning to the design of advanced decision support and planning systems.

Bruno Langlois graduated from the Ecole des Mines de Paris (MS), Paris, France, in 1987 where he studied control theory.

From 1987 to 1989 he worked as a research engineer at the Computer Science Department of Stanford University, California. His research was dealing with anticollision path planning focusing on the dynamic constraint aspect. He is now working at Aerospatiale, Paris, France in the field of Computer Integrated Manufacturing and Production Management.



Jean Claude Latombe was born in Pernes-les-fontaines, France, on May 16, 1947. He received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in computer science, in 1969, 1972, and 1977, respectively, from the National Polytechnic Institute of Grenoble, France.

From 1980 to 1984, he was a faculty member at Ecole Nationale Superieure d'Informatique et de Mathematiques Appliquées de Grenoble (ENSIMAG), where he developed new research and education programs in artificial intelligence, robotics, and computer vision. From 1984 to 1987, he was on leave from ENSIMAG and held the position of executive president of Industry and Technology of Machine Intelligence (ITMI), a company that he cofounded in 1982 for commercializing robotic systems and expert systems. From 1984 to 1987, he directed the French national project in Artificial Intelligence supported by the ministry of Research. In April 1987, he joined Stanford University as Associate Professor in the Department of Computer Science. At Stanford, he is the Director of the Computer Science Robotics Laboratory. His current interests include motion and task planning, autonomous robots, integrated design and manufacturing, and graphic animation. He is the author of the book *Robotic Motion Planning* (Kluwer Academic Publisher, 1991).