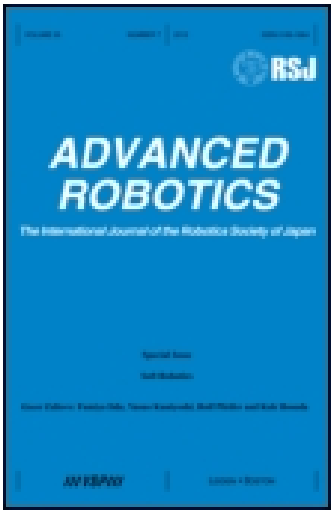


Advanced Robotics
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tadr20

Sensor-based motion planning in three dimensions for a highly redundant snake robot

Dan Reznik [a] & Vladimir Lumelsky [b]
[a] Robotics Laboratory, University of Wisconsin-Madison, Madison, WI 53706, USA
[b] Robotics Laboratory, University of Wisconsin-Madison, Madison, WI 53706, USA
Published online: 02 Apr 2012.

To cite this article: Dan Reznik & Vladimir Lumelsky (1994) Sensor-based motion planning in three dimensions for a highly redundant snake robot, Advanced Robotics, 9:3, 255-280, DOI: 10.1163/156855395X00193

To link to this article: http://dx.doi.org/10.1163/156855395X00193

Advanced Robotics, Vol. 9, No. 3, pp. 255–280 (1995)

Sensor-based motion planning in three dimensions for a highly redundant snake robot

DAN REZNIK and VLADIMIR LUMELSKY
Robotics Laboratory, University of Wisconsin—Madison, Madison, WI 53706, USA

Received for *AR* 6 June 1994

Abstract—A strategy is described for real-time motion planning for a highly redundant snake-like robot manipulator operating in a three-dimensional (3D) environment filled with unknown obstacles of arbitrary shape. The robot consists of many (say 30 or 50) links serially connected by universal joints (such a joint allows a 3D rotation of one link relative to the other). The robot's sensors allow it to sense objects in the vicinity of any points of its body. The task is to move the robot's tip point (its *head*) from its starting position to a specified target position, collision-free for the whole robot's body. To achieve the efficiency necessary for real-time computation, an iterative procedure is proposed which makes use of a unit motion for a single link based on the *tractrix* curve. This choice also results in automatically achieving a motion that is 'natural' (in that the joint displacements tend to 'die out' in the direction from head to tail) as well as locally optimal.

1. INTRODUCTION

We consider the problem of *motion planning with incomplete information* for a special type of robotic device — a highly redundant snake-like kinematic structure (a *snake*, for brevity). The snake's body consists of many (say 30 or 50) links serially connected by universal joints (a universal joint between two links provides three-dimensional (3D) rotation of one link relative to the other via two mutually perpendicular axes [1]). The task of the snake is to move its *head* from the starting position to a given target position in its 3D environment. The rest of the snake's body has to somehow follow the head. When the snake's *tail* is fixed, the robot is called a *snake manipulator*. For reasons that will become clear later, it turns out that from the motion planning standpoint this case includes the case where the snake's tail is free to move. This produces a *free snake* structure which may be of value in itself and is considered below as well.

The environment in which the snake operates may be filled with unknown *obstacles* of arbitrary shapes. The generated motion should involve no collisions with obstacles; other than that, the position (configuration) of the snake's body is not important. Touching an obstacle does not present a collision. The information about the obstacles is provided by the robot's *sensors* which allow it to sense objects at any point of its

body. While the approach considered here will apply to and can take advantage of more complex sensing, such as range sensing or vision, from an algorithmic standpoint it is sufficient to assume simple tactile sensing, similar to that of the skin of real snakes.

Given that the information about the environment is always incomplete and appears in real time from the robot's sensors, the problem at hand calls for a continuous on-line computation and planning. This is in contrast to off-line procedures for motion planning which assume complete information and one-time computation [2–4]. Notice also that, in principle, planning with complete information allows optimal planning, whereas planning with incomplete information rules it out. [To exemplify the difference, imagine a path inside a building of someone who comes routinely to their own office (planning with complete information), as compared to the path of someone else who tries to find the same office for the first time (planning with incomplete information); each task has its own context and rational, although the second path may look absurdly long to the first person.]

A redundant kinematics is one in which the number of degrees of freedom (d.o.f.) available is more than the minimum necessary to perform a general task. If the task is to provide an arbitrary position and orientation at the robot end effector, then the minimum number of d.o.f. is known to be three for the two-dimensional (2D) case and six for the 3D case [5].

Among the many variations of highly redundant manipulators, the snake-like kinematics is perhaps the most prevalent. One reason for that is that from an engineering standpoint the snake structure provides the flexibility necessary in various applications [6–10]. For example, the 8-link, 16-d.o.f. inspection robot built by Asano *et al.* [6] has a slender long body and is flexible enough to operate in the crowded space of a nuclear reactor. On the other hand, the inherent nonlinearities due to the snake's revolute joints and difficulties of dealing with *singularities* (which amount to unrealistically large motions and velocities needed to produce a small displacement at some points of the robot's body [5]), make the snake one of the most general and interesting structures from a theoretical standpoint [11, 12].

There are relatively few works on motion planning for highly redundant manipulators. Almost exclusively complete information and off-line planning is assumed. It has been shown that this problem has exponential complexity in the number of robot d.o.f. [13]; consequently, in considering robots with a large number of d.o.f. researchers have concentrated on heuristic approaches. Specifically, Gupta [14] exploits the serial structure of a planar snake manipulator in a sequential search strategy: the motion is computed iteratively, link by link. Only after a complete collision-free motion has been planned for link i, can link $i + 1$ be processed. If no such motion is found for link i, the search backtracks so as to revise and change the motions planned for the previous links. Chirikjian and Burdick [15] plan collision-free motions of a planar snake by forcing the snake's backbone (a continuous curve which approximates the robot's current position) to pass through certain 'tunnels' placed at strategic locations. This is similar to the strategy by Asano [16] which avoids collisions by requiring each joint of the snake to follow the path traced by the head.

A number of approaches to motion planning for highly redundant robots are based on 'local methods', whereby part of the complete information available is 'suppressed' to speed up the computation. The technique by Barraquand and Latombe [17] for a multi-link planar robot makes use of the potential fields method [18] and reduces the solution space to few randomly generated robot configurations at the next time moment. Faverjon and Tournassoud [19] base their scheme on maximizing the robot's distance to the obstacles. To increase performance, obstacles are represented hierarchically at increasing levels of detail. The approach by Maclean and Cameron [20] is conceptually close to ours, and involves (i) computing a path for the robot's head and then (ii) letting the head 'drag' the snake's body along; obstacles are avoided via repelling potential fields associated with the obstacles.

The sensor-based approach pursued in this work has been so far developed only for manipulators with a small number (up to three) links and joints [21]. Besides an appropriate motion planning algorithm, convergence is a primary concern for those cases. Extending the approach to highly redundant systems would provide a powerful mechanism for handling complex robots in unstructured and time-varying tasks. Given that a redundant robot has an infinite number of configurations for a given position/orientation of its end effector, one price for this power is the loss of guarantee of convergence — the algorithms become heuristic in nature. On the technical side, given the high dimensionality of the associated configuration space and the fact that the surfaces in question are not algebraic, one gives up the common technique of reducing the problem to moving a point in the configuration space; thus the problem is considered directly in the workspace.

Proposed in this work is a heuristic procedure which generates a 'reasonable', though always collision-free, motion. The head 'pulls' the snake's body along a path which is computed on-line, based on sensing information. The links slide along, while avoiding the encountered obstacles. Links' motions are computed by a *propagation* procedure, serially from head to tail and, when necessary, from tail to head. The idea is to find the position of the snake at the next moment such that (i) it provides the desired head position and (ii) it results in the minimal total joint displacements for the rest of the body. For the case of the free snake, only one such head-to-tail propagation is needed. In the case of the snake manipulator, both the head-to-tail and tail-to-head propagations are executed. We address the following questions:

- Given the obvious asymmetry of the task (only the head's position is important; the snake always moves head-on), can some principle of economy of motion be added that would (i) cause the motion to 'die out' towards the tail, (ii) make all links participate in the motion and (iii) result in a locally-minimum motion. In other words, can some kind of 'natural' distribution of motion between links be suggested?
- In robotic we are used to the notion that motors/actuators in the joints are always actively controlled during the motion. Is this really necessary? Observations of real snakes suggest that at a given moment some of the joints/links can be 'passive' and be simply pulled by other links. If realizable, such a scheme would simplify the robot control.

- Still another typical notion is that adding degrees of freedom carries a heavy computational burden. Can information processing be organized so that the extra degrees of freedom would provide freer motion — as they should — while not resulting in a computational punishment?
- Can a computationally inexpensive scheme be offered for dealing with the snake's simultaneous contacts with multiple obstacles, to yield collision-free motion?

The result is a procedure for on-line motion planning for a multi-link 3D snake. The computational complexity of the procedure is linear in the number of snake links. In practical terms, the procedure is quite efficient; the actual computation speed is almost independent of the number of links. The sensing capabilities assumed in our model can take different forms and have already been realized in hardware [22, 23].

The planning procedure is asymmetric and proceeds in small *steps*, say 30 or 50 steps per second, resulting in continuous motion. The snake configuration for the next step is computed as follows: first, the motion is planned for the snake's head; then, a propagation of calculations is done from the head to the tail link so as to produce a collision-free position for the whole body. An essential part in this sequential process is the local minimization of total motion. This comes out automatically due to the fact that a link's 'unit motion' is chosen to follow a *tractrix* curve [24]; this curve also happens to possess interesting optimization properties. In the case of a snake manipulator, the fact that the snake's tail is fixed produces an additional constraint which dictates still another propagation of calculations, this time from tail to head. Once the calculations are complete, the actual step motion is executed, and the process repeats.

Note that with each link being a line segment, two links connected by a joint always lie in a plane. Therefore, the basic procedure for planning the position of a link relative to its neighbor link can be approached as a planar problem. Generalizing the planar motion to 3D then becomes a relatively easy task. Accordingly, most of the development below proceeds in the plane, with the 3D generalization appearing at the end.

The remainder of this paper is organized as follows: Section 2 describes the accepted model. In Section 3 the idea of choosing the tractrix curve for forming a basic unit motion is discussed and its properties are established. Then, in Section 4, three *propagation procedures* are developed for iterative computation of the next step position of the snake. These procedures are then incorporated into an algorithm for sensor-based motion planning of a free planar snake (Section 5) and a snake manipulator (Section 6). The strategy is then generalized for the three-dimensional case (Section 7).

2. THE MODEL

A snake is a sequence of N *links*, l_i, $i = 1, \ldots, N$, serially connected by *universal joints*. A universal joint has two mutually perpendicular intersecting axes and allows the two links that it connects to rotate relative to each other; this is equivalent to two

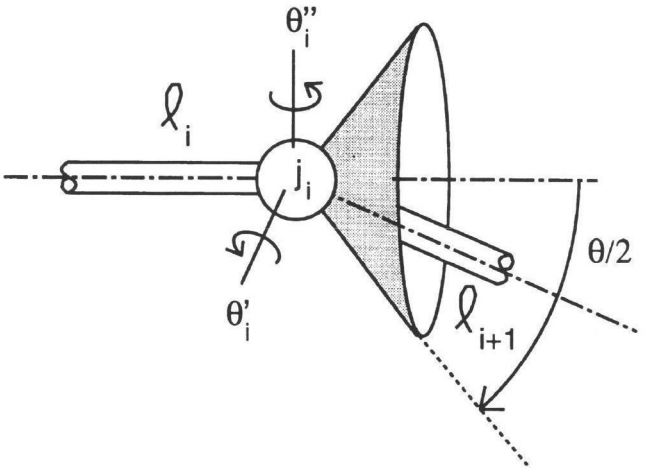


Figure 1. A universal joint: joint j_i has two independently controlled revolute degrees of freedom, about two mutually perpendicular axes θ_i' and θ_i''; at any moment link l_{i+1} lies inside the cone shown.

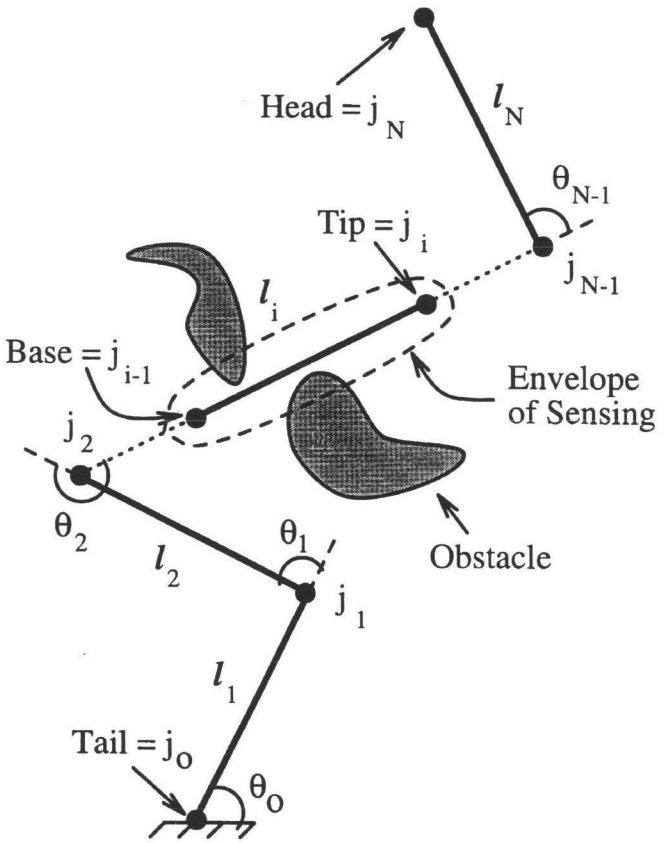


Figure 2. An N-link snake manipulator. Link l_i is shown with its envelope of sensing (dotted line).

independently controlled revolute joints (2 d.o.f.) [1]. Each joint j_i is thus associated with two *joint angles*, θ_i' and θ_i'', or a joint vector $\theta_i = (\theta_i', \theta_i'')$. Each link is a line segment; all links are of equal length L. Therefore, if the end j_i of link l_i is fixed, its other end lies on a sphere of radius L centered at j_i. Each joint angle θ_i is bounded by the same *joint limits* $\theta_{\max}$, $\theta_{\min}$, and so its range of change is $\theta = \theta_{\max} - \theta_{\min}$. Link l_i is connected by a joint j_{i-1} to the previous link, l_{i-1}, and by joint j_i to the next link, l_{i+1} (Fig. 1). Assume that a limit is also imposed on the sum effect of both angles θ_i', θ_i'', such that link l_{i+1} is always inside a cone of angle θ whose apex is at j_i. A snake with N links has therefore $2N$ d.o.f. A planar version of the snake manipulator is shown in Fig. 2.

The instantaneous *configuration* of the snake can be described uniquely either via the set of angles (θ_0', θ_0''), (θ_1', θ_1''), ..., $(\theta_{N-1}', \theta_{N-1}'')$, or in Cartesian terms, $(j_0, j_1, \ldots, j_N)$, with j_i being a 3D vector of Cartesian coordinates. While both presentations are equivalent, the Cartesian presentation is more convenient for the approach considered and is assumed below. The two endpoints of the snake are called *head* and *tail*, respectively (Fig. 2). The head is responsible for initiating and generating motion along a path in free space and along obstacle boundaries. For completeness, the head is also called joint j_N, and the tail — joint j_0.

If joint j_0 is free to move, i.e. the snake motion in the workspace is limited only by obstacles, the structure becomes a *free snake*. If joint j_0 is fixed, the snake's tail link is constrained to rotating about j_0 and the structure becomes a *snake manipulator*. From the standpoint of practical applications and theoretical analysis, both structures have peculiarities of their own. We start below with a free snake and then proceed with a snake manipulator.

3. TRACTRIX AS A BASE UNIT OF MOTION

3.1. Motion in free space

Consider one link, of length L, in the plane (Fig. 3). Assume it is a rigid body, with point P being its *tip* and point Q being its *base*. If one pulls the tip P along the axis (straight line) x and if no additional constraints are imposed on Q's motion, then point Q will move along a curve known as the *tractrix* [24]. Clearly, since the motion of point Q is induced by that of P, the vector of velocity at point Q is always aligned with the link, i.e. with the tangent to the tractrix. (The tractrix has been independently discovered by Leibnitz and Huygens around 1690.) The tractrix equation is derived from the corresponding differential equation for the tangent (Fig. 3):

$$\frac{dy}{dx} = \frac{-y}{\sqrt{L^2 - y^2}}, \tag{1}$$

which after integration produces a more explicit tractrix equation:

$$x = L \cdot \log \frac{y}{L - \sqrt{L^2 - y^2}} - \sqrt{L^2 - y^2}. \tag{2}$$

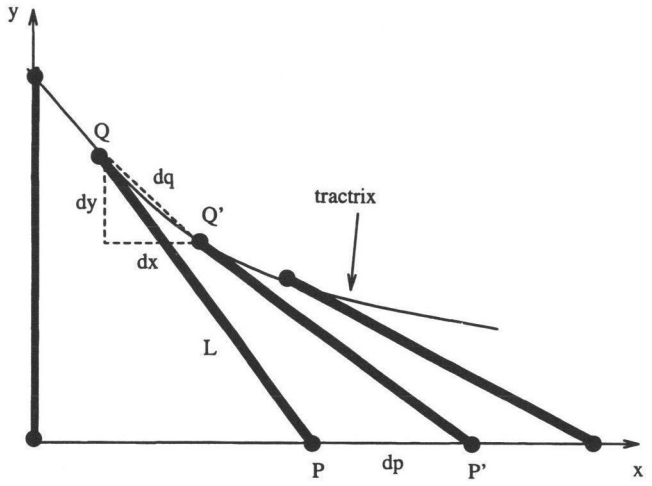


Figure 3. For a single link, if its tip P is pulled along the straight line x, its other end, point Q, moves along the tractrix. Shown are four different positions of the link along its path.

Because the instantaneous velocity at point Q is directed along the link, the following optimality property holds: given an infinitesimal displacement $\mathrm{d}p$ of P along x, the length of the path traversed by Q, denoted by a vector $\mathrm{d}q$, presents a local minimum of all possible paths for Q. Furthermore, the ratio between $\mathrm{d}q$ and $\mathrm{d}p$ obeys an inequality $\mathrm{d}q \leqslant \mathrm{d}p$. To show that, consider the ratio $\mathrm{d}q/\mathrm{d}p$. Denote x and y to be the coordinates of point Q, and p the x-coordinate of point P; then, $\mathrm{d}x$ and $\mathrm{d}y$ are the corresponding x, y-components of $\mathrm{d}q$ (Fig. 3), and

$$p = x + \sqrt{L^2 - y^2}. \tag{3}$$

Present $\mathrm{d}q/\mathrm{d}p$ as

$$\frac{\mathrm{d}q}{\mathrm{d}p} = \frac{\mathrm{d}q}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}p}. \tag{4}$$

Displacement $\mathrm{d}q$ can be written as $\mathrm{d}q = \sqrt{\mathrm{d}x^2 + \mathrm{d}y^2}$. Substituting into this equation $\mathrm{d}y$ from the tractrix equation, obtain

$$\mathrm{d}q = \frac{\mathrm{d}x \cdot L}{\sqrt{L^2 - y^2}}, \tag{5}$$

from which $\mathrm{d}q/\mathrm{d}x$ can be extracted. To obtain $\mathrm{d}x/\mathrm{d}p$, find first its inverse, by differentiating the expression for p above:

$$\frac{\mathrm{d}p}{\mathrm{d}x} = \frac{L^2}{\sqrt{L^2 - y^2}}. \tag{6}$$

Now, by substituting dq/dx and dx/dp into the expression for dq/dp, obtain

$$\frac{dq}{dp} = \frac{\sqrt{L^2 - y^2}}{L} \leqslant 1. \tag{7}$$

This becomes an equality if $y = 0$, which happens only if the whole link lies in the line x of the motion of its tip P. In all other positions of the link relative to the line of motion of its tip P, the displacement at Q is smaller than that at P. That is, if δ_1 is the displacement of the tip between points P and P', and δ_0 is the displacement along the tractrix between the corresponding points Q and Q', then the differential property above implies that always $\delta_0 \leqslant \delta_1$ (Fig. 3).

Note that the motion of the link's base along the tractrix appears 'automatically' — it is simply being pulled by the link's tip. In the robotics context, such motion can be generated solely by an actuator located at the tip.

Now, consider a free snake with N links (see a planar version in Fig. 4), and consider a straight-line motion of the snake's head along the line ST. As before, joint j_{N-1} moves along the tractrix. This motion will pull link l_{N-1}, which will make joint j_{N-2} move along what can be called the *second-order tractrix*, which will make joint j_{N-3} to move along what can be called the *third-order tractrix*, and so on. The described pattern is called *tractrix motion.*

Denote the corresponding displacements at the joints $j_N, j_{N-1}, \ldots$ as $\delta_N, \delta_{N-1}, \ldots$. Given the continuity of motion, as the step size δ_N at the head decreases, the corresponding (tractrix) motion of the base of l_N in the limit approaches a straight line. Since the base of l_N coincides with the tip of l_{N-1}, this means that although the motion of the base of link l_{N-1} is along the second-order tractrix, for the calculation purposes one can use a regular tractrix — the error will be small if δ_{N-1} is sufficiently small. Doing so for all links eliminates the necessity for higher-order tractrices. The argument above produces this conclusion:

LEMMA 1. *Under tractrix motion, when the snake's head moves along a straight line, the corresponding displacements of the snake's joints obey an inequality*

$$\delta_0 \leqslant \delta_1 \leqslant \ldots \leqslant \delta_{N-1} \leqslant \delta_N, \tag{8}$$

with the equality $\delta_i = \delta_{i-1}$ reached only when the line of motion of joint j_i coincides with link l_i.

Now, given the current position of the N-link snake, its position at the next moment can be computed using the following iterative procedure which operates on the joints' step displacements. Recall that the tip of link l_i coincides with the base of link l_{i+1}, $i = 1, \ldots, N-1$. Assume that the step of the head is $\delta_N = \delta = \text{const}$, and that the line ST and the current positions of all the joints j_i, $i = 0, \ldots, N$, are given. Positions j_i' of the joints at the next moment are computed sequentially, starting with joint j_N and progressing to joint j_0, as follows.

Starting with the head, its next step position becomes j_N' such that $\delta_N = |j_N, j_N'|$, and the position of the base j_{N-1}' of link l_N is computed using the tractrix equation

above, with its displacement being $\delta_{N-1} = |j'_{N-1}, j_{N-1}|$. Here $|a, s|$ is the Euclidean distance between points a and b. In general, the next step position of the tip j_i of link l_i is computed as

$$\delta_i = |j'_i, j_i|. \tag{9}$$

The position of its base, j'_{i-1}, is computed using the tractrix equation and, finally, the corresponding displacement of this base is

$$\delta_{i-1} = |j'_{i-1}, j_{i-1}|. \tag{10}$$

As described, the computation of the free snake's position at the next step consists of a number of calculations related to individual links and joints. Only after this process is over, does the actual step move take place. All these calculations can be done fast enough to produce real-time motion.

Figures 4–6 present examples of motion of free snakes with four and 20 links, respectively, in an obstacle-free 2D space. For convenience, the paths produced by all joints are also shown. In Fig. 4, the snake has been first requested to move from the initial position S to point T_1 (Fig. 4a), then from T_1 to T_2 (Fig. 4b) and so on until point T_6. The motion shown in Fig. 5 is generated in a similar manner. Observe the effect to decreasing displacements at the joints from the head to the tail, as suggested by Lemma 1. The examples in Fig. 5, while making it difficult to discern the motion of individual links, show the area swept by the snake's body during the motion. Figure 6 shows the motion of the same 20-link snake between two points, S and T, where S corresponds to a rather complex 'folded' configuration of the snake; to provide more detail, few intermediate positions are also shown.

3.2. Motion amongst obstacles

Consider a single link, denoted PQ, whose endpoints are P and Q (Fig. 7a). The link's length is $|PQ|$. Assume a unit motion by the link is contemplated (as calculated by some operator called *Unit*), which would move the link's tip along a straight line to point P' and the link's base along a tractrix to point Q'.

Suppose this unit motion from PQ to $P'Q'$, $Q' = Unit(PQ, P')$, would result in a collision (intersection) with an obstacle O_1, as shown in Fig. 7(b). This intersection can be cleared by rotating the link about P' until the overlap with O_1 is eliminated. A unit motion followed by an optional rotation is combined in a single operation called *UnitRot*, defined as follows:

$$UnitRot(PQ, P', \delta) = R$$

such that

$$|RQ'| = \min_X \left(|XQ'|:\ P'X \cap O_s = \phi \quad \text{and} \quad |XQ| \leqslant \delta\right),$$

where

$$Q' = Unit(PQ, P').$$

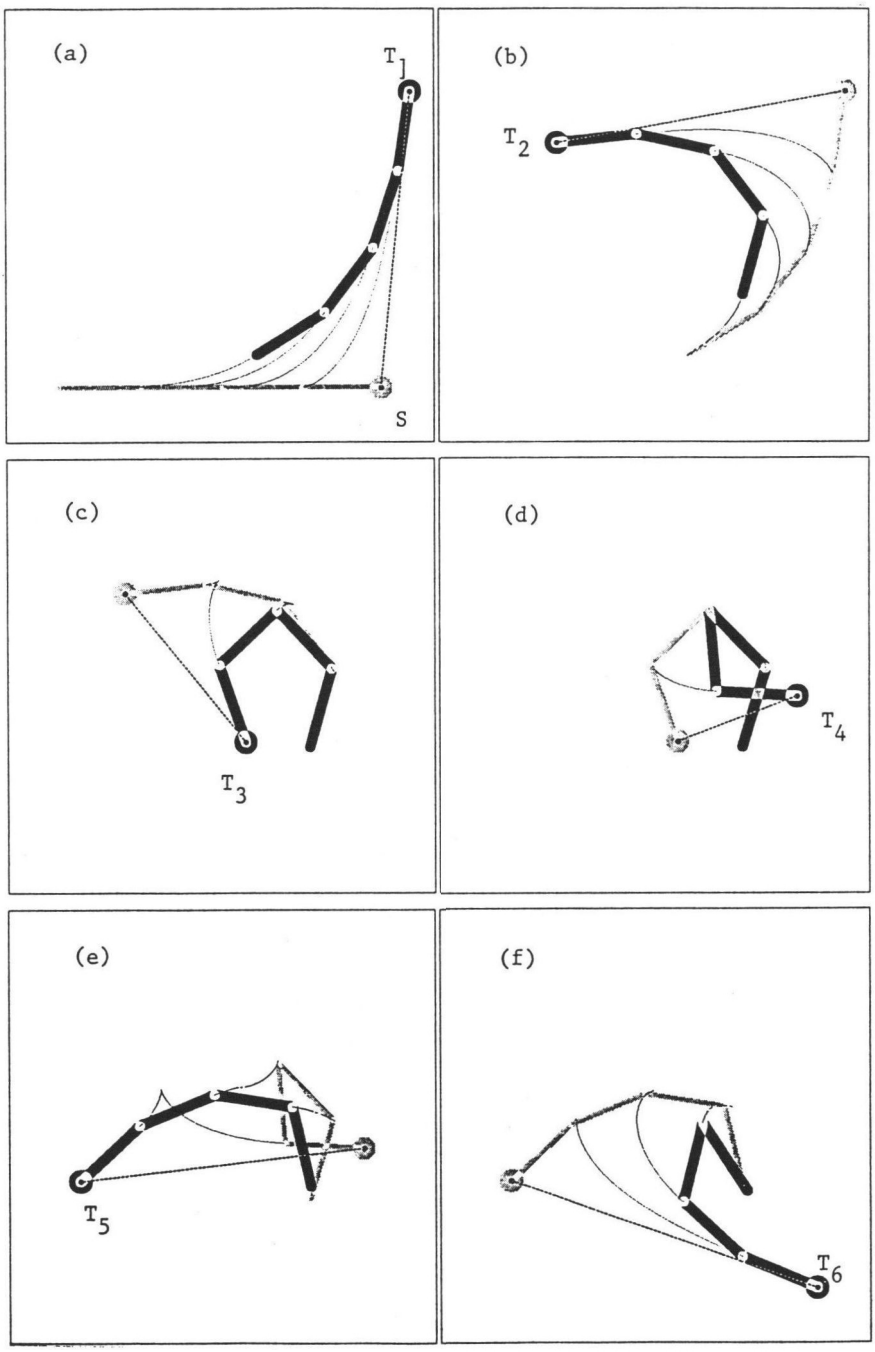


Figure 4. Motion of a 4-link snake in an obstacle-free environment. Also shown are the trajectories of all joints. In (a), the snake starts in the horizontal position, with the links stretched from the tail to the head in S, and moves to the target position T_1. Similarly, in (b)–(f), the snake starts at the position that has been the target position in the previous figure, and proceeds to the next target position, $T_2, \ldots, T_6$.

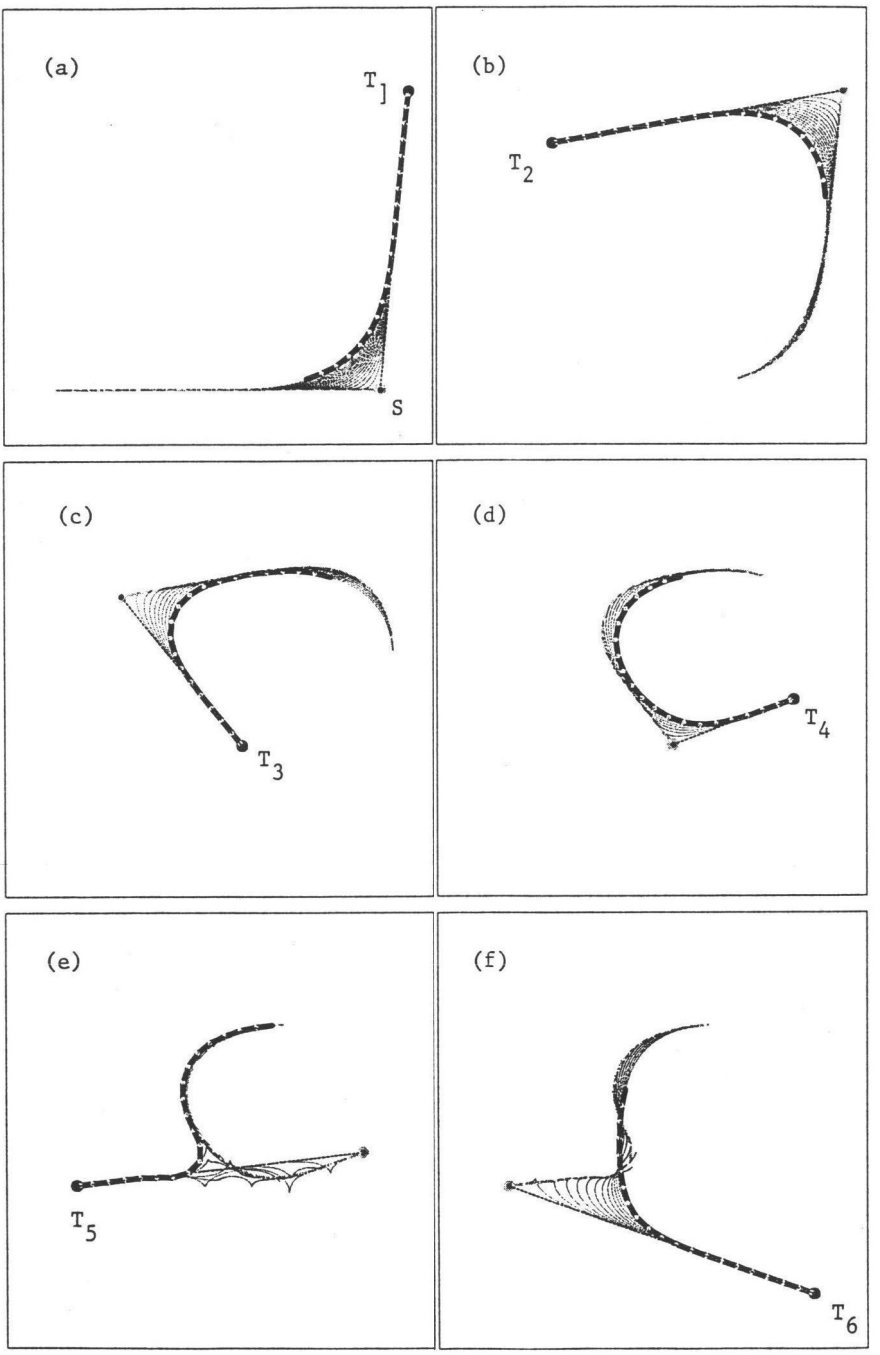


Figure 5. Motion of the 20-link snake in an obstacle-free environment. The target positions, $T_1, \ldots, T_6$, and the order of visiting them are the same as in Fig. 4.

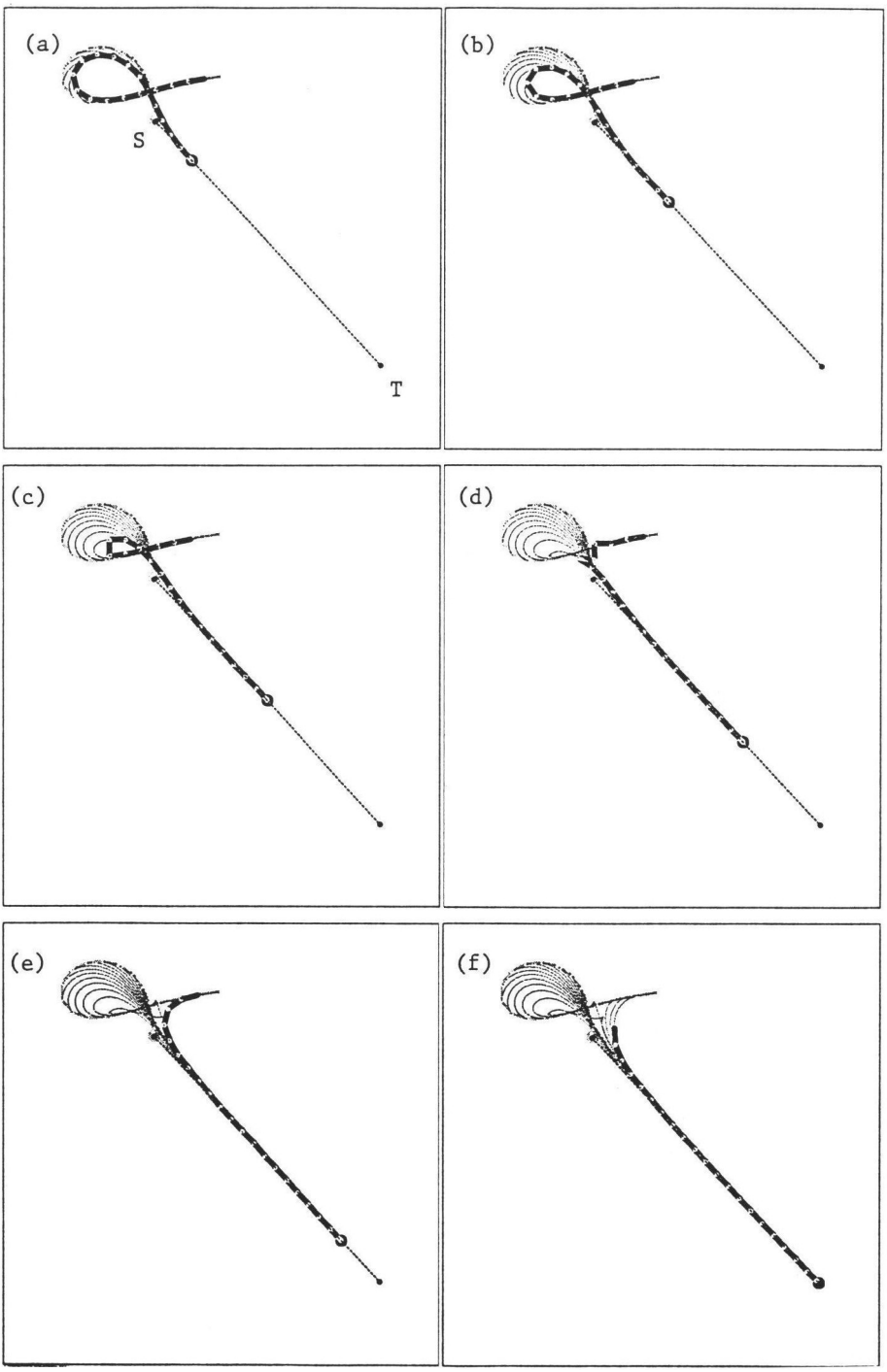


Figure 6. Motion of the 20-link snake in an obstacle-free environment. The snake starts in a folded starting position, and moves to point T. For better detail, few intermediate positions are also shown.

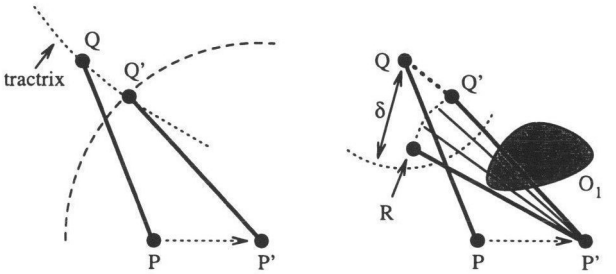


Figure 7. (a) A unit motion of a link: given a straight line motion of one endpoint, the other endpoint moves along the tractrix curve. (b) If a unit motion causes an interference with a nearby obstacle, the link is rotated about its tip until the conflict is eliminated.

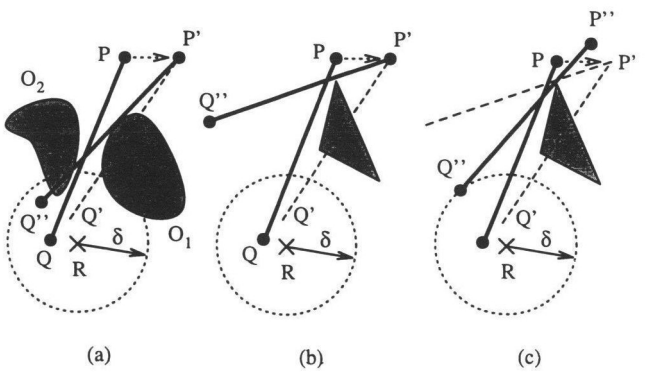


Figure 8. In (a) and (b), the operator *UnitRot*(PQ, P', δ) fails to find a point R within distance δ from point Q which results in a collision-free position of the link. In (a), obstacle O_2 is encountered during a rotation intended to clear obstacle O_1; in (b), the required rotation would be excessive, $|Q'Q''| > \delta$. In (c), the *NewDir* operator finds a new point P'' closest to P such that *UnitRot* does not fail.

Here O_s represents all obstacles sensed by link PQ. The parameter δ specifies a maximum value for $|RQ|$ and serves two purposes: (i) to prevent the link from rotating outside the pre-specified range of sensing and (ii) to preserve the motion attenuation property even after the rotation. If a point R cannot be found within distance δ from Q such that the link at $P'R$ avoids all intersections, then *UnitRot* is said to fail. Figure 8 illustrates two situations where this happens: in Fig. 8(a) the failure is caused by the presence of two nearby obstacles and in Fig. 8(b) it is caused by a single obstacle.

If the operator *UnitRot* fails, an additional operation, *NewDir*(PQ, P', δ), is evoked, which limits the step at Q by planning a shorter step at P. Namely, it chooses a new point P'' closest to P' such that *UnitRot*(PQ, P'', δ) does not fail, see Fig. 8(c). *NewDir* is defined as follows:

$$NewDir(PQ, P', \delta) = P''$$

such that

$$|P'P''| = \min_X \big(|XP'|:\ UnitRot(PQ, X, \delta) \text{ does not fail}\big).$$

4. THE PROPAGATION PROCEDURES

Consider a vector $M = (j_0 \dots j_N)$ containing the coordinates of all the joints in the snake; j_0 is the tail and j_N is the head. The basic propagation procedure $Prop(M, i, P)$ computes a new vector $M' = (j'_0 \dots j'_N)$ for the next step position of the snake. The procedure emulates a 'pull' towards the next position P planned for a joint j_i, which is then *propagated* towards both j_0 and j_N, through iterative applications of *UnitRot*. Namely, the procedure involves two iterations over the elements of M: (i) from j_i down to j_0 and (ii) from j_i up to j_N. If $UnitRot(j_a j_b, Q, |j_i P|)$ fails at any point, e.g. due to an unresolved conflict with an obstacle, the procedure stops and reports an error (a, b, Q). The distance $|j_i P|$ is used as the δ parameter in *UnitRot* to ensure that upon completion of *Prop*, $j_k \leqslant |j_i P|$, $\forall k$. Figure 9(a) illustrates a propagation $M' = Prop(M, 3, P)$ applied to the joint j_3 of a 5-link manipulator in an obstacle-free workspace. The procedure $Prop(M, i, P)$ operates as follows:

Proc.: $Prop(M, i, P)$
Inputs: A cartesian vector $M = (j_0 \dots j_N)$;
An integer i, $0 \leqslant i \leqslant N$;
A point P.
Output: A cartesian vector $M' = (j'_0 \dots j'_N)$ or else
an error vector (a, b, Q), where a, b, are
integers and Q is a point.

Step 1: Set $j'_i = P$.
Step 2: Set $k = i$.
Step 3: While $k > 0$ do:
3.1: Set $j'_{k-1} = UnitRot(j_k j_{k-1}, j'_k, |j_i P|)$;
3.2: If 3.1 fails, abort with error $(k, k-1, j'_k)$;
3.3: Set $k = k - 1$.
Step 4: Set $k = i$.
Step 5: While $k < N$ do:
5.1: Set $j'_{k+1} = UnitRot(j_k j_{k+1}, |j_i P|)$;
5.2: If 5.1 fails, abort with error $(k, k+1, j'_k)$;
5.3: Set $k = k + 1$.
End.

We now introduce two other propagation procedures, a *soft propagation* and a *hard propagation*, which form the motion planning algorithm proper. The *soft propagation* procedure, $SoftProp(M, i, P)$, operates on the vector M of the joints' coordinates and is defined as follows. First, a normal propagation, $M' = Prop(M, i, P)$, is attempted.

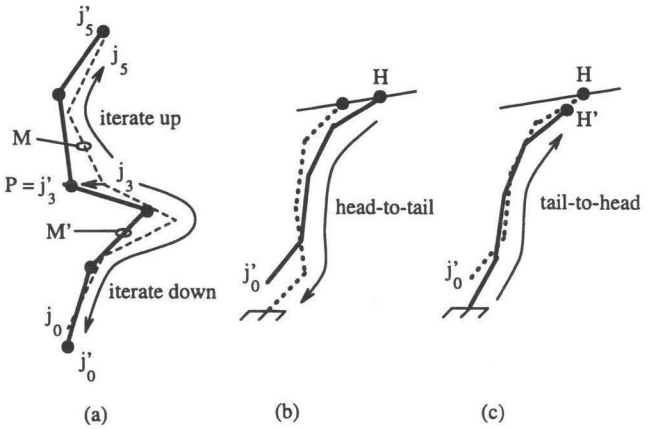


Figure 9. (a) A propagation applied to link j_3 of a 5-link manipulator in an obstacle-free workspace. The original position of the manipulator is shown with dashed lines. Links are traversed starting at j_3 both down and up the structure. (b) The generation of every new configuration consists of a head-to-tail propagation, which causes the tail to drift, followed by a (c) tail-to-head propagation, which re-establishes the original position of the tail but produces a small error at the head.

If it is successful, M' is simply returned and the process stops. Otherwise, if *Prop* returns an error of the form (a, b, Q), then a second propagation of $Prop(M, a, P')$ is attempted, where P' is computed with $NewDir(j_a j_b, Q)$. The process continues until *Prop* returns no errors. We call this propagation soft since upon completion it does not guarantee that joint j_i will arrive at point P, $j_i = P$. (Note that $j_i = P$ only if the first propagation is a success; this condition may be unimportant if it only defines a desired direction of motion and does not reflect any physical constraint.) Procedure $SoftProp(M, i, P)$ operates as follows:

Proc.: $SoftProp(M, i, P)$
Inputs: A cartesian vector $M = (j_0 \ldots j_N)$;
An integer i, $0 \leqslant i \leqslant N$;
A point P.
Output: A cartesian vector $M' = (j'_0 \ldots j'_N)$.

Step 1: Set $M' = Prop(M, i, P)$.
Step 2: While an error (a, b, Q) reported, do:
2.1: Set $Q' = NewDir(j_a j_b, Q)$;
2.2: Set $M' = Prop(M, a, Q')$.
End.

In contrast, the *hard propagation* procedure, $HardProp(M, i, P)$, does guarantee that joint j_i will arrive at point P, $j_i = P$, upon completion. This is achieved by calling *SoftProp* repeatedly, until the equality is satisfied. Procedure $HardProp(M, i, P)$ operates as follows:

Proc.: $HardProp(M, i, P)$
Inputs: A cartesian vector $M = (j_0 \ldots j_N)$;

An integer i, $0 \leqslant i \leqslant N$;
A point P.
Output: A cartesian vector $M' = (j'_0 \ldots j'_N)$.

Step 1: Set $M' = SoftProp(M, i, P)$.
Step 2: While $j'_i \neq P$ do:
2.1: Set $M' = SoftProp(M', i, P)$.
End.

5. 2D MOTION PLANNING FOR A FREE SNAKE

Generating one step for a free snake simply amounts to a single execution of the soft propagation procedure, $SoftProp(M, i, P)$. Since at the end of the step the tail does not have to satisfy any special constraints, it is positioned wherever it appears at the end of the step calculation. The procedure then iterates for the next step and so on, until point T is reached by the snake's head.

This computation is linear in the number of the snake's links. For a given set of obstacles, the procedure is guaranteed to work if the link length L is sufficiently small. In the limit, when L approaches zero and the snake becomes a rope, the solution is feasible for any set of obstacles. Inversely, for a given L, the solution is feasible for some and not feasible for other sets of obstacles. In the latter case the described single propagation of calculation along the snake would not be sufficient. Such situations, while of some interest for the case of a free snake, become especially important in motion planning for the snake manipulator (see below). The described algorithm is called $\mathcal{A}_f$ (f for 'free').

Figures 10 and 11 present examples of motion of a 20-link snake in a 2D environment with obstacles. For illustration purposes, the paths produced by all joints are also shown. As in Fig. 4, the snake has been first requested to move from the initial position S to point T_1 (Fig. 10a), then from T_1 to T_2 (Fig. 10b) and so on until point T_6. Figure 11 'summarizes' the motion shown in Fig. 10: the trajectories of the joints indicate the total area swept by the snake during its motion between the positions $S, T_1, \ldots, T_6$.

6. 2D MOTION PLANNING FOR A SNAKE MANIPULATOR

This motion planning algorithm, called $\mathcal{A}_m$ (m for 'manipulator'), is built out of the procedures developed above. It also makes use of the operator $HeadStep(P)$ which, starting with the current position P of the head, computes its next step position P', using some maze-searching algorithm (see, e.g., [25, 26]). It is convenient to present vector $M = (j_0 \ldots j_N)$ in terms of Cartesian coordinates of the snake's joints in the workspace. The actual control of motors at the robot joints may require the corresponding vector of the joint angles $C = (\theta_0 \ldots \theta_{N-1})$. Translating M into C takes a simple trigonometric operation which results in a unique solution.

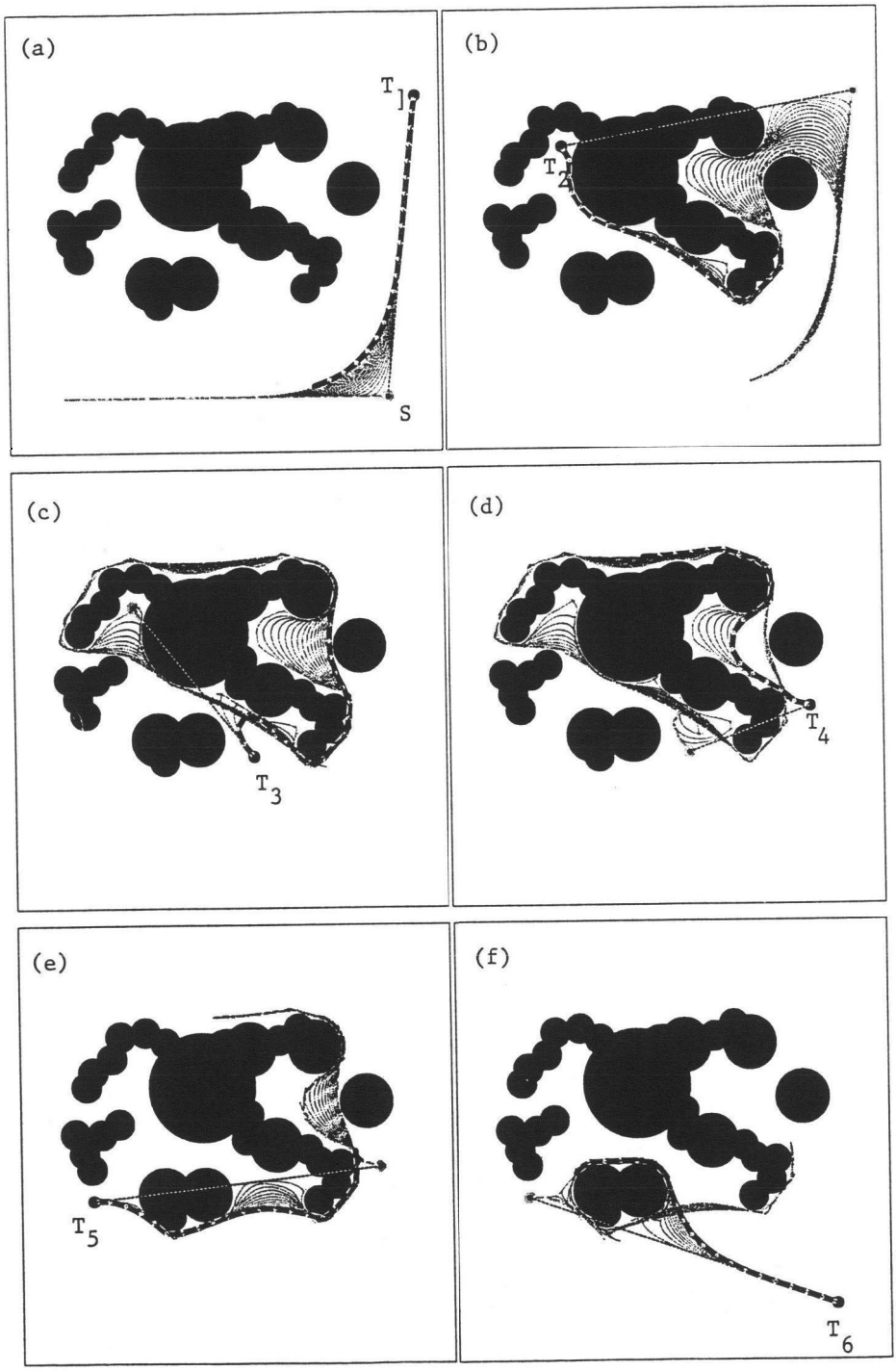


Figure 10. Motion of the 20-link snake in an environment with obstacles. The target positions, $T_1, \ldots, T_6$, and the order of visiting them are the same as in Fig. 4.

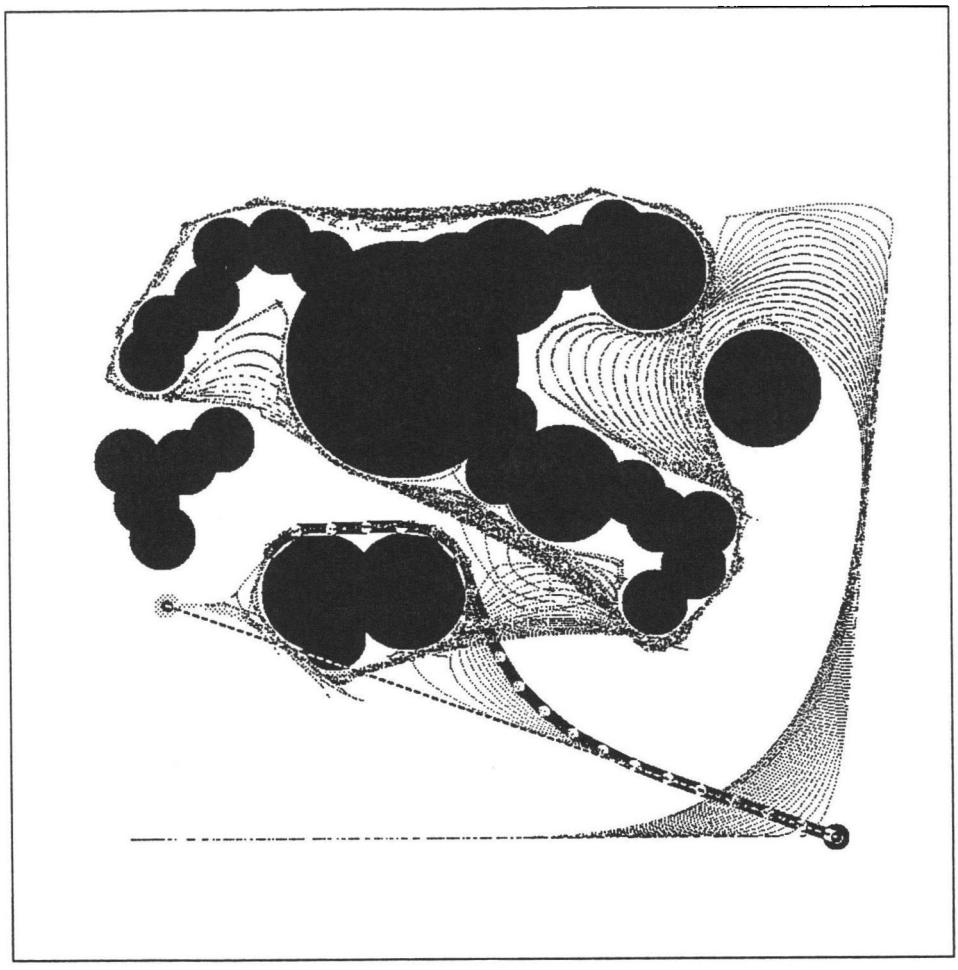


Figure 11. A 'summary' of the motion of the 20-link snake shown in Fig. 10. Trajectories of the joints indicate the total area swept by the snake during its motion between the positions $S, T_1, \ldots, T_6$.

The algorithm is composed of a single loop which iterates until the head of the manipulator arrives at point T. At the end of every iteration, a new collision-free configuration for the manipulator is produced and sent to the joint motors. Each iteration begins with the computation of a next position H for the head, performed by the operator *HeadStep*. This is used to trigger the computation of the next configuration starting with a head-to-tail soft propagation. One side effect of this is a (small) displacement of the tail. To restore the tail to its original position, a tail-to-head hard propagation is executed. This causes the final position of the head to deviate from H by some amount, which is acceptable since no physical constraints are imposed at this endpoint. Note that due to the tractrix attenuation property the amount of displacement at the last joint in the propagation is small (and is guaranteed to be less than the step size δ). The intermediate joint positions following a head-to-tail soft propagation and a tail-to-head hard propagation in an obstacle-free environment are

illustrated in Fig. 9(b and c). The whole algorithm $\mathcal{A}_m$ can now be summarized as follows:

Proc.: $\mathcal{A}_m(M, T)$
Inputs: The manipulator configuration $M = (j_0 \ldots j_N)$;
The target point T.
Output: A sequence of collision-free configurations
taking the head to T.

Step 1: While $j_N \neq T$ do:
1.1: Set $H = HeadStep(j_N)$;
1.2: Set $M' = SoftProp(M, N, H)$;
1.3: Set $M = HardProp(M', 0, j_0)$;
1.4: Send M to joint motors;
End.

A simulated example of the algorithm's performance is shown in Fig. 12. The manipulator operates in a complex unknown environment. Its task is to move from the position S to the target position T (Fig. 12a). Snapshots of the manipulator during its motion are shown in Fig. 12(b–f). The algorithm used here (in the *HeadStep* operator) to generate the motion of the head is *VisBug*21 [25], which uses range sensing within a limited radius (in this example equal to the link length) to produce locally

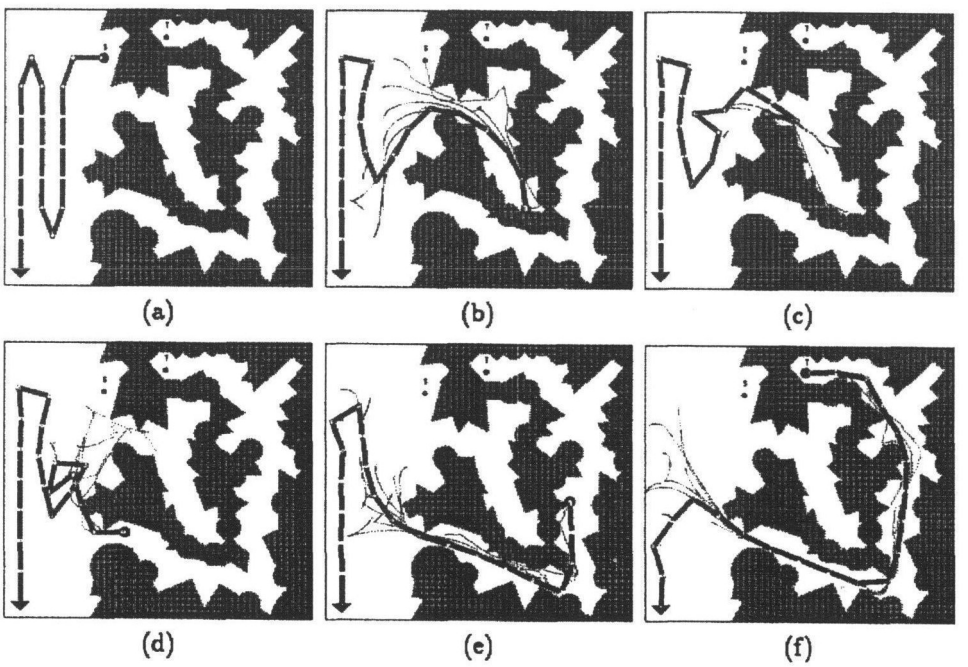


Figure 12. Example of the algorithm's performance for a 20-link manipulator immersed in a complex 2D environment. The head starts at position S and moves to point T selected by the user as the target. Six snapshots of the motion are shown. Also shown are the trajectories of all joints between the previous and current snapshots.

optimal paths. Note that during the operation the snake does not collect information about the environment and does not remember its previous path.

7. 3D MOTION PLANNING

In principle, the 3D version of the problem at hand requires exhaustive search. When on its way to the target the (3D) snake encounters an obstacle, there is an infinite number of directions it can use to pass it around. Subsequently, a rather simple-minded heuristic approach is taken in the procedure below. As above, assume that besides sensing at its body, the head has a range sensor that allows it to see obstacles within some distance (depth of vision).

Imagine that at some moment the head, while at a position H, sees an obstacle in the direction of the line HT, where T is the target position. Say, the line HT intersects the obstacle at point P (Fig. 13). A simple strategy then for passing around the obstacle would be this: while staying at H, the robot inspects the obstacle radially around P until it sees a 'clearance' in the obstacle that suggests a possible way around. If no such clearance is found, the head moves toward the farthest point visible in a prespecified direction, to pass around the obstacle. Otherwise, a clearance with the smallest distance from point P, as seen in the plane perpendicular to line HT and passing through point H, is chosen (see plane Π, Fig. 13), and the head moves in the corresponding direction.

A good approximation of such a radial visual inspection of the obstacle is a scan that starts at point P and follows an unwinding spiral in the plane, Π, perpendicular to the line HT, until a clearance is found (Fig. 13).

Assume that, given the current output of the head's range sensor, an operator *GetDist*(Q) computes the distance to the obstacle measured along the line HQ, Q — an arbitrary point. The head's new desired position H' is obtained as follows: (i)

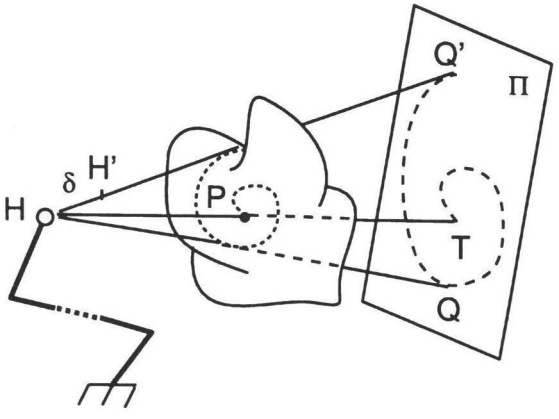


Figure 13. A 3D obstacle prevents the snake head H from moving directly towards the target point T. To plan the next position H' for the head, the obstacle is scanned along an unwinding spiral starting at P, until a clearance, or a visible edge of the obstacle, is spotted. The head then makes a step towards the point found, and the process repeats.

if the target is visible from the head, i.e. $GetDist(T) = |HT|$, then plan H' directly toward the target, with $|HH'| = \delta$ and (ii) if line HT is crossed by an obstacle, $GetDist(T) < HT$, plan a step toward a point identified as described above.

In the algorithm, the procedure $GetDist(Q)$ performs a spiral-like scan of the obstacle until a point Q appears with an abrupt increase of the measured distance $|HQ|$. This abrupt increase is used as an indication of a clearance. The operator *HeadStep* is defined so that it takes as input the head's current position H and the location T of the target and outputs H', the new desired position for the head, such that $|HH'| = \delta$, as follows:

Proc.: $HeadStep(H, T)$
Input: H : the head's current position;
T: the target's position.
Output: A new position H' planned for the head.

Step 1: Set $Q = T$;
Step 2: Set $d = GetDist(Q)$;
Step 3: If $d < |HT|$ then:
3.1: Set $\hat{u}$ = unit vector $\perp (T - H)$;
3.2: Set $\hat{v}$ = unit vector $\perp (T - H)$ and $\hat{u}$;
3.3: Set $t = 0$;
3.4: Repeat:
a. Set $d' = d$;
b. Set $Q = T + k_1 t[\hat{u}\cos(k_2 t) + \hat{v}\sin(k_2 t)]$;
c. Set $d = GetDist(Q)$;
d. Set $t = t + \delta t$;
3.5: Until $|d - d'| > d_{\max}$.
EndIf;
Step 4: $H' = H + \delta \frac{(Q-T)}{|QT|}$;
End *HeadStep*.

In lines 1, 2 and 3, the procedure attempts to plan a step directly toward T. Namely, if $GetDist(T) = HT$, then there is no blocking obstacle, and the procedure goes directly to line 4 where a step toward T of length δ is planned. If, however, an obstacle appears between H and T, two mutually perpendicular vectors $\hat{u}$ and $\hat{v}$ are computed (lines 3.1 and 3.2) which lie in the plane Π passing through T and perpendicular to segment HT. Let a planar spiral curve [24] be parameterized by a single parameter t, initially set to zero (line 3.3). Then the procedure enters a loop (line 3.4) which repeats until there is an abrupt change (e.g. above a threshold $d_{\max}$) in the distance toward the obstacle. Specifically, in line 3.4(a) the last distance measured is stored in an auxiliary variable d'. In 3.4(b) a new point Q along the spiral is computed, based on the orthogonal directions $\hat{u}$ and $\hat{v}$ — constants k_1 and k_2 regulate, respectively, the spiral's radial and angular velocities with respect to t (their values should be fixed according to the accuracy desired for the scan). In 3.4(c) the distance to the new Q is measured and in 3.4(d) the parameter d is incremented by a small step. Finally, the loop test in 3.5 compares the value of the previously measured

distance d' to the distance d just measured in line 3.4(c); the loop is aborted if the difference $d - d'$ is above a threshold $d_{\max}$, i.e. the segment HQ' is tangent to the obstacle. Finally, line 4 plans a step of length δ toward the last position Q was set to. A simulated example shown in Figs 14–16 illustrates the performance of the 3D motion planning algorithm described. The simulation was done on a Silicon Graphics workstation; the motion was calculated and executed in real time, thus producing an animated movie. The robot's workspace (Fig. 14) presents a structure with multiple vertical columns (somewhat similar to the fuel rods of a nuclear reactor), limited from the top and the bottom by a ceiling and a floor. The connecting beams that enforce some columns add to the space clutter and make it more difficult for the robot to find an appropriate opening. The robot manipulator is a 15-link, 30-d.o.f. snake; its base is positioned as shown in Fig. 15. The robot's task is to move from the starting position S to the target T (Fig. 15a). The robot has no knowledge about the objects in its environment and plans its motion in real time, based on sensing information. Its sensors allow it to sense objects in the vicinity of any point of its body (see Section 2). Each pair of pictures in Fig. 15(a–d) shows the scene from two different directions and relates to the same time moment: Fig. 15(a) corresponds to the starting position, Fig. 15 (b and c) to two intermediate positions, and Fig. 15(d) to the final (target) position. For better visualization, Fig. 16 shows the same final position with the ceiling removed. Note that on its way to the target the robot has to identify relatively

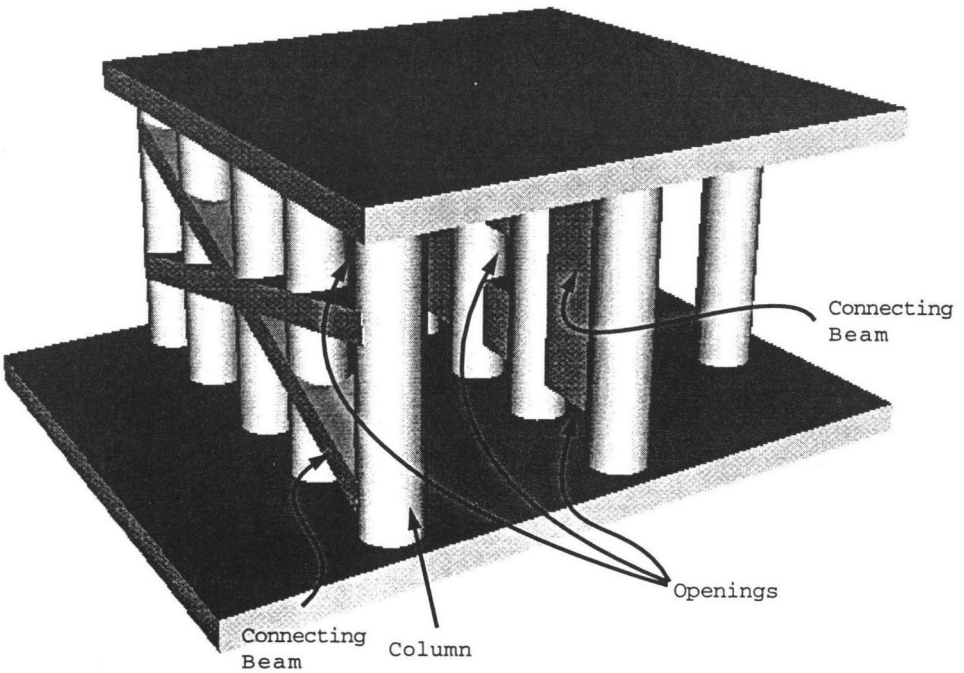


Figure 14. In this example, the robot's workspace includes multiple vertical columns (somewhat similar to the fuel rods of a nuclear reactor), limited from the top and the bottom by the ceiling and the floor. The connecting beams that enforce some columns add to the space clutter and make it harder to find an appropriate opening.

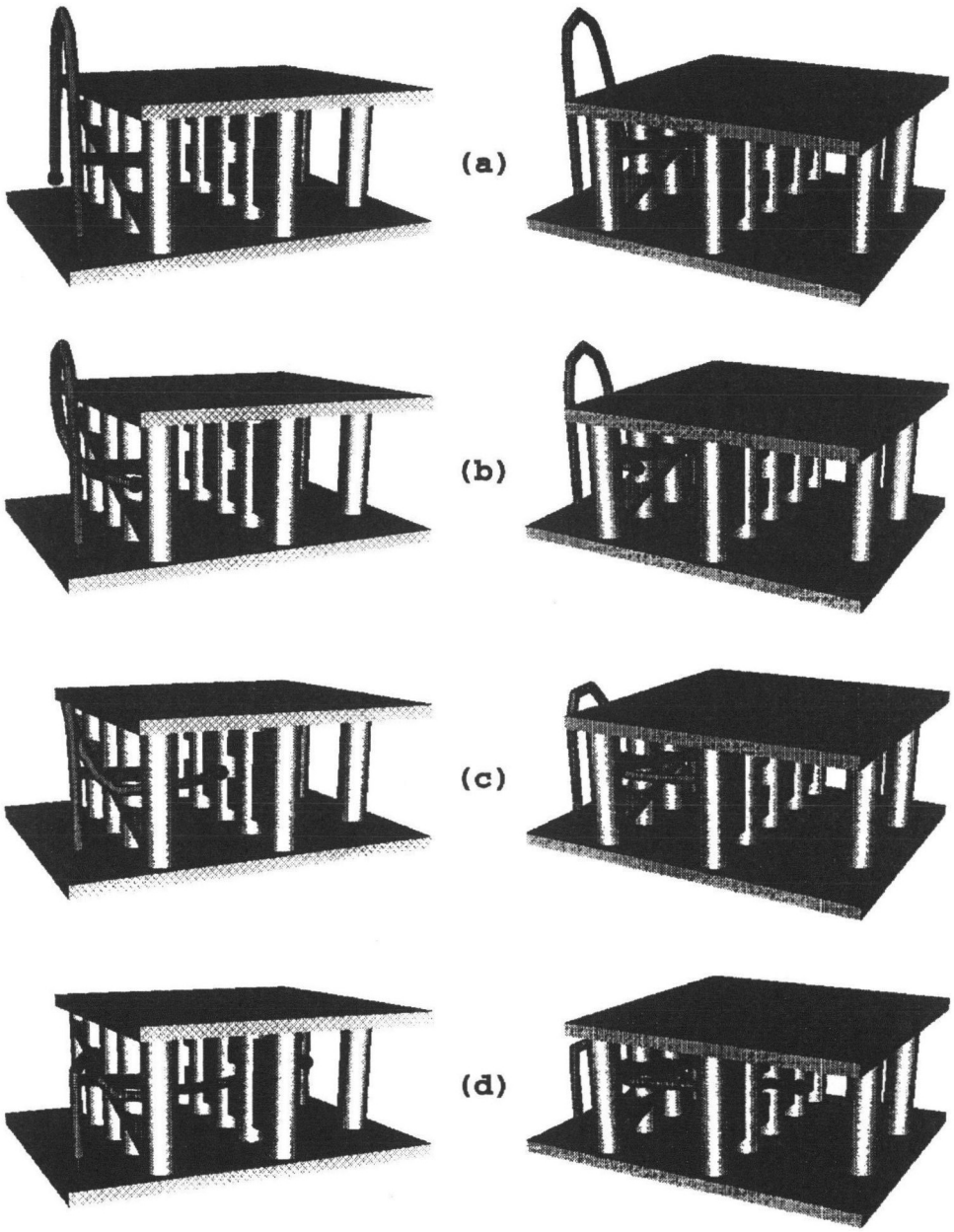


Figure 15. Four time snapshots of the robot's motion in the environment of Fig. 14. The robot manipulator is a 15-link, 30-d.o.f. snake; its tail is positioned in the workspace as shown. Each pair of pictures, (a)–(d), shows the scene from two different directions and relates to the same time moment: figure (a) corresponds to the starting position, (b) and (c) correspond to two intermediate positions, and (d) corresponds to the final (target) position.

small openings in this highly crowded environment, which produces its 'very 3D' shape in the final configuration.

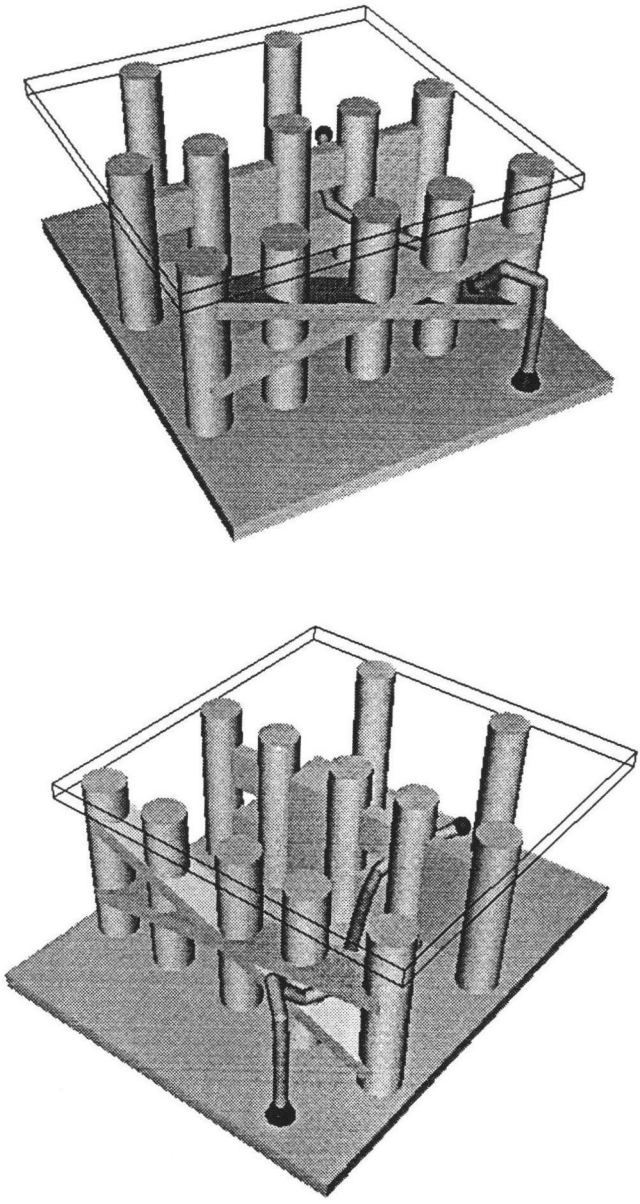


Figure 16. Two alternative views of the final position of the robot (same as in Fig. 15d). For better visualization, the workspace ceiling is made 'transparent'. Note the 3D curve produced by the robot body in the course of its motion.

Acknowledgements

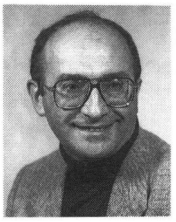
Supported in part by the National Science Foundation Grant IRI-9220782 and by the DoE (Sandia Laboratories) Grant 18-4379C.

REFERENCES

1. J. Craig, *Introduction to Robotics, Mechanics and Control.* Reading, MA: Addison-Wesley, 2nd edn, 1989.
2. T. Lozano-Pérez, "A simple motion planning algorithm for general robot manipulators," *IEEE J. Robotics Automat.*, vol. RA-3, no. 3, pp. 224–238, 1987.
3. J. Schwartz and M. Sharir, "On the 'piano movers' problem. II: General techniques for computing topological properties of real algebraic manifolds," *Adv. Appl. Math.*, vol. 4, pp. 298–351, 1983.
4. J. Canny, "A new algebraic method for robot motion planning and real geometry," in *28th Annu. IEEE Symp. on Foundations of Computer Science*, Los Angeles, CA, 1987, pp. 39–48.
5. R. Paul, *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge, MA: MIT Press, 1981.
6. K. Asano, M. Obama, Y. Arimura, M. Kondo and Y. Hitomi, "Multi-joint inspection robot," *IEEE Trans. Ind. Electron.*, vol. IE-30, no. 3, pp. 277–281, 1983.
7. W. Clement and R. Iñigo, "Design of a snake-like manipulator," *Robotics Autonomous Syst.*, vol. 6, no. 3, pp. 265–282, 1990.
8. S. Hirose and A. Morishima, "Design and control of a mobile robot with an articulated body," *Int. J. Robotics Res.*, vol. 9, no. 2, pp. 99–114, 1990.
9. Y. Shan and Y. Koren, "Design and motion planning of a mechanical snake," *IEEE Trans. Syst., Man, Cybernet.*, vol. 23, no. 4, pp. 1091–1100, 1993.
10. G. Chirikjian and J. Burdick, "Design and experiments with a 30-DOF robot," in *IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, 1993.
11. J. Burdick, "On the inverse kinematics of redundant manipulators: characterization of the self-motion manifolds," in *IEEE Int. Conf. on Robotics and Automation*, Scottsdale, AZ, 1989.
12. R. Baker and C. Wampler, "On the inverse kinematics of redundant manipulators," *Int. J. Robotics Res.*, vol. 7, no. 4, pp. 3–21, 1988.
13. J. Reif, "Complexity of the mover's problem and generalizations," in *20th Annu. IEEE Symp. on Foundations of Computer Science*, San Juan, PR, 1979, pp. 421–426.
14. K. Gupta, "Fast collision avoidance for manipulator arms: a sequential search strategy," *IEEE Trans. Robotics Automat.*, vol. 6, no. 5, pp. 522–532, 1990.
15. G. Chirikjian and J. Burdick, "An obstacle avoidance algorithm for hyper-redundant manipulators," in *IEEE Int. Conf. on Robotics and Automation*, Cincinnati, OH, 1990.
16. K. Asano, "A snake robot arm manipulator," *Toshiba Reviews*, vol. 8, 1977.
17. J. Barraquand and J.-C. Latombe, "Robot motion planning: a distributed representation approach," *Int. J. Robotics Res.*, vol. 10, no. 6, pp. 628–649, 1991.
18. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robotics Res.*, vol. 5, no. 1, pp. 90–98, 1986.
19. B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," in *IEEE Int. Conf. on Robotics and Automation*, Raleigh, NC, 1987.
20. A. Maclean and S. Cameron, "Snake-based path planning for redundant manipulators," in *IEEE Int. Conf. on Robotics and Automation*, Altanta, GA, 1993.
21. V. Lumelsky and K. Sun, "Three-dimensional motion planning in an unknown environment for robot arm manipulators with revolute or sliding joints," *Int. J. Robotics Automat.*, vol. 9, no. 5, pp. 89–104, 1994.
22. E. Cheung and V. Lumelsky, "Real-time path planning procedure for a whole-sensitive robot arm manipulator," *Robotica*, vol. 10, pp. 339–349, 1992.
23. K. Roberts, "Robot active touch exploration: constraints and strategies," in *IEEE Int. Conf. on Robotics and Automation*, Cincinnati, OH, 1990.
24. G. Korn and T. Korn, *Mathematical Handbook.* New York: McGraw-Hill, 1968.
25. V. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Trans. Syst., Man, Cybernet.*, vol. 20, no. 5, pp. 1058–1069, 1990.
26. O. Ore, "Path problems," in *Theory of Graphs*, vol. XXXVIII, New York: American Mathematical Society, 1962.

ABOUT THE AUTHORS

Dan Reznik was born in Rio de Janeiro, Brazil in 1967. He received a Bachelor's degree with distinction in Computer Engineering from the University of Kansas, Lawrence, KS, in 1990, and a Master's degree in Computer Science from the University of Wisconsin—Madison, in 1992. He is currently conducting his PhD research on sensor-based motion planning for highly-redundant robots at the Robotics Laboratory of the University of Wisconsin—Madison. Mr Reznik's main interests include kinematics of redundant manipulators, highly-redundant grasping, visualization of 3D configuration spaces and computer simulation of motion planning algorithms.



Vladimir Lumelsky is Consolidated Paper Professor of Engineering and Computer Science at the University of Wisconsin—Madison. Prior to this, he held academic and research positions, starting in 1968, with the Institute of Control Sciences of the Russian National Academy of Sciences in Moscow, Russia; at Ford Motor Research Laboratories; General Electric Corporate Research Center; and Yale University. His current research is in robotics and industrial automation, and spans theoretical as well as experimental work; his professional interests also include computational geometry, motion planning algorithms, sensor-based intelligent systems, control theory, computer vision, pattern recognition and kinematics. He has served on the Editorial Board of the *IEEE Transactions on Robotics and Automation*; the Board of Governors of the IEEE Robotics and Automation Society; as Chairman of this Society's Technical Committee on Robot Motion Planning and Control; Chairman of IFAC Working Group on Robot Motion, Sensing and Planning; Program Chairman of the 1989 IEEE International Conference on Intelligent Robots and Systems (IROS'89) in Tokyo; and Guest Editor for two special issues of the *IEEE Transactions on Robotics and Automation* (1987 and 1989). He is an IEEE Fellow, and a member of ACM and SME.