gorithm used for *HeadStep* here happens to be *VisBug*21 [4], which uses vision within a limited radius (in our case equal to the link length) to produce locally-optimal paths. Note that during the operation the snake does not collect information about the environment and does not remember its previous path.

With this choice of a head planning procedure, the whole algorithm can be thought of as an emulation of the behavior of a real snake: the sensing at the head is equivalent to vision, allowing the head to foresee obstacles and locally optimize its motion; the sensing at the rest of the body emulates tactile sensing, allowing the snake to slide along obstacles.

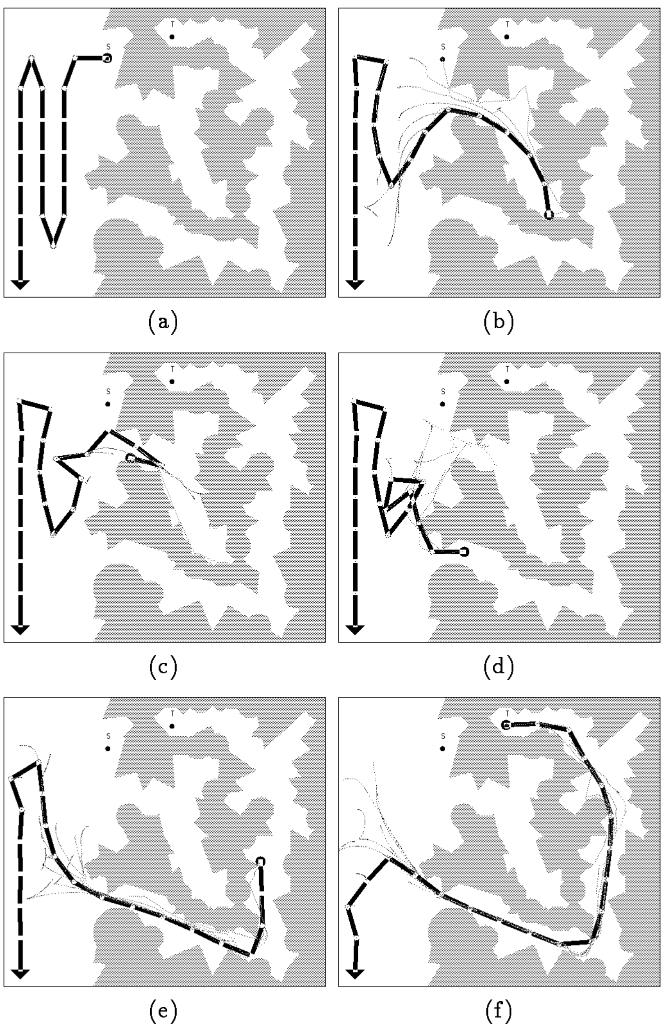

Fig. 5. Example of the algorithm's performance for a 20-link manipulator immersed in a complex environment. The head starts at position S and the user selects T as the target. Six snapshots of the motion are shown. Also shown are the trajectories of all the joints between the previous and current snapshots.

References

[1] D. Reznik, V. Lumelsky, Motion Planning with Uncertainty for Highly Redundant Kinematic Structures: I. "Free Snake" Motion. *1992 IEEE International Conference on Robotics and Automation*, Raleigh, North Carolina.

[2] R. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, 1981.

[3] J. T. Schwartz, M. Sharir, On the "Piano Movers" Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. *Advances in Applied Mathematics*. Academic Press. No.4 (298-351), 1983.

[4] V. J. Lumelsky, T. Skewis, "Incorporating Range Sensing in the Robot Navigation Function", *IEEE Transactions on Systems, Man, and Cybernetics*. Vol. 20, No. 5, September 1990, 1058-1069.

[5] O. Khatib, Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *International Journal of Robotics Research*, 5(1), 1986, 90-98.

[6] J. Barraquand, B. Langlois, J.-C. Latombe, Numerical Potential Field Techniques for Robot Path Planning. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 2, March 1992, 224-241.

[7] G. S. Chirikjian, L. W. Burdick, An Obstacle Avoidance Algorithm for Hyper-Redundant Manipulators, *Proc. 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, May 1990.

[8] G. S. Chirikjian, L. W. Burdick, A Geometric Approach to Hyper-Redundant Manipulator Obstacle Avoidance, *Proc. ASME Mechanisms Conference*, Chicago, September 1990.

[9] K. Gupta, Fast Collision Avoidance for Manipulator Arms: A Sequential Search Strategy, *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 5, October 1990, 522-532.

[10] K. Asano, A Snake Robot Arm Manipulator, *Toshiba Reviews*, 1977.

[11] K. Sun, V. Lumelsky, Motion Planning for Three-Link Robot Arm Manipulators Operating in an Unknown Three-Dimensional Environment, *Proc. 30th IEEE International Conference on Decision and Control*, Brighton, England, December 1991.

[12] G. Korn and T. Korn, *Mathematical Handbook*, McGraw Hill, New York, 1968.

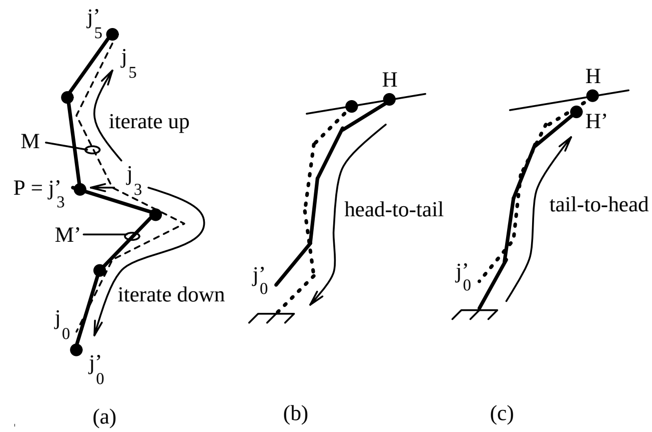

Fig. 4. (a) A propagation applied to link j_3 of a 5-link manipulator in an obstacle-free workspace. The original position of the manipulator is shown with dashed lines. Links are traversed starting at j_3 both down and up the structure. (b) The generation of every new configuration consists of a head-to-tail 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.

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

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* $HardProp(M, i, P)$ is a procedure that does guarantee $j_i = P$ upon completion. This is achieved by calling $SoftProp$ repeatedly, until the equality is satisfied:

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

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.

III. The motion planning algorithm

The final motion planning algorithm, $\mathcal{A}_m$, is built out of the procedures developed above. It also makes use of the operation $HeadStep(P)$ which computes the next point P' of the path for the head (assuming the head is currently at P), using some maze-searching algorithm. It is convenient to present vector $M = (j_0 \cdots j_N)$ in terms of cartesian coordinates of the arm joints in the workspace. The actual control of motors at the robot joints may require the corresponding vector of joint angles $C = (\theta_0 \cdots \theta_{N-1})$. Translating M into C takes a simple trigonometric operation which results in a unique solution. We now define $\mathcal{A}_m$, the motion planning algorithm for an N-link manipulator:

Proc.: $\mathcal{A}_m(M, T)$
Inputs: The manipulator configuration $M = (j_0 \cdots j_N)$;
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.

$\mathcal{A}_m$ is composed of a single loop which iterates until the head of the manipulator is placed 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 $HeadStep$. 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 motion 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 a small amount, which is acceptable since no physical constraints are imposed at this endpoint. 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 Figure 4 (b) and (c), respectively.

IV. Example

A simulated example of the algorithm's performance is shown in Figure 5. Here, the manipulator is immersed in a complicated environment unknown to the robot. Its task is to move from the position S to the target position T, Figure 5(a). Snapshots of the manipulator during its motion are shown in Figures 5(b-f). The head motion al-

O_1, as shown in Figure 2(b). This intersection can be cleared by rotating the link about P' until the overlap with O_1 disappears. A unit motion followed by an optional rotation is combined in a single operation called *UnitRot*, defined as follows:

$$UnitRot(PQ, P', \delta) = R \text{ such that}$$

$$|RQ'| = \min_X(|XQ'| : P'X \cap O_s = \phi \text{ and } |XQ| \leq \delta)$$

$$\text{where } Q' = Unit(PQ, P')$$

Here O_s represents all obstacles around (i.e., sensed by) PQ. The parameter δ specifies a maximum value for $|RQ|$ and serves two purposes: (1) to prevent the link from rotating outside the prespecified range of sensing, and (2) to preserve the motion attenuation property even after the rotation. If an R within distance δ from Q such that the link at $P'R$ avoids all intersections cannot be found, then *UnitRot* is said to *fail.* Figure 3 illustrates two situations where this happens: in (a) the failure is caused by the presence of two nearby obstacles and in (b) it is caused by a single obstacle.

If the operator *UnitRot* fails, an additional operation, $NewDir(PQ, P', \delta)$, 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 $UnitRot(PQ, P'', \delta)$ does not fail, see Figure 3(c). *NewDir* is defined as follows:

$$NewDir(PQ, P', \delta) = P'' \text{ such that}$$

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

B. The propagation procedure

Consider a vector $M = (j_0 \cdots j_N)$ containing the coordinates of all the joints in the manipulator. The propagation

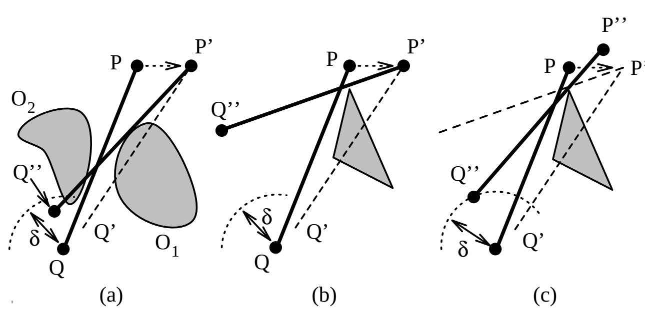

Fig. 3. In (a) and (b) $UnitRot(PQ, P', \delta)$ fails to find a point R within distance δ from Q which results in a collision-free position of the link. In (a) obstacle O_2 is encountered during a rotation intended to clear O_1; in (b) the required rotation would be excessive ($|Q'Q''| > \delta$). In (c) the *NewDir* operator finds a new point P'' closest to P such that *UnitRot* does not fail.

$Prop(M, i, P)$ computes a new vector $M' = (j'_0 \cdots j'_N)$ according to the following procedure:

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

The procedure emulates a "pull" towards P applied to j_i, which is then *propagated* towards both j_0 and j_N, through iterative applications of *UnitRot*. Namely, the procedure involves two iterations on the elements of M: (1) from j_i down to j_0 and (2) 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 a cluttered workspace, the procedure stops and reports an error (a, b, Q). Note that the distance $|j_i P|$ is used as the δ parameter in *UnitRot* to ensure that upon completion of *Prop*, $j_k \leq |j_i P|, \forall k$. Figure 4(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.

We now introduce two procedures, a *soft propagation* and a *hard propagation*. The *soft propagation*, $SoftProp(M, i, P)$, 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. 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 a *soft* propagation since upon completion it does not guarantee that $j_i = P$. (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).

the zero-motion constraint. Hence, a second, tail-to-head propagation is initiated, which produces zero motion at the tail and a small deviation from the contemplated position for the head; since the latter does not violate any physical constraints, it can be tolerated. Once the second propagation is complete, the physical move to the calculated configuration takes place, and the whole process repeats.

A conflict may arise within a given propagation if a link's endpoint, e.g., joint j_i, cannot be brought to a prescribed position while (1) avoiding possible collisions, and (2) obeying the constraints that no point of the link body should move further than the maximum step δ allowed. In such a case, the current propagation is aborted and joint l_i in question becomes a temporary "head" within a new propagation which proceeds from l_i in both directions, toward the tail and toward the head.

For simplicity, the snake's links are allowed full rotation about their joints – which can result in self-intersections. The latter problem, not addressed here, can be handled in a number of ways, one being that each link treats other links as obstacles.

Another issue not addressed here for lack of space relates to the links' length and shape. Though our links are straight line segments of the same length and zero width, the algorithm allows links to be of different length and of different shape (see, e.g. [11]).

C. *Related work*

Between the two possible models of motion planning – *motion planning with complete information* and *motion planning with incomplete information* – this work belongs to the latter group. As to the former group, the existing work has considered both redundant and nonredundant systems. The problem is stated as follows: given the geometry and positions of the robot and the obstacles (usually assumed to be analytical, such as polyhedra), the problem is to find a path between the starting and target positions. Given the complete knowledge, in principle an optimal path is feasible (for more details and a general formulation, see, for example, [3]). One variation within this model includes the potential field approach which looks for special desired features of the generated paths [5]. An intermediate situation, where part of the (complete) information is being "suppressed" to improve the algorithm efficiency is considered in [6]. Obstacle-avoidance techniques for highly redundant snake-like systems with complete information have been considered in [7, 8, 9]. On the applications side, an interesting snake-link robot manipulator with 18 links, intended for repair work in a nuclear reactor, has been built in the 1970s by K. Asano in Toshiba Corp. [10].

Although from the engineering standpoint motion planning that relies on real-time sensor data is highly desirable, such systems have been slow to come, due largely to the theoretical and algorithmic difficulties. As a result, the existing work on motion planning with incomplete information has almost exclusively concentrated on "minimal", nonredundant systems. The problem is usually reduced to moving a point in an appropriately chosen two-dimensional subspace of the corresponding configuration space [11]. Given the inherently local nature of such algorithms, plan optimality is ruled out; instead, a "reasonable" behavior is sought. To our knowledge, the work in [1], on a somewhat simpler case of a free-moving snake robot, is the first attempt of this kind for redundant systems.

II. Components of the Algorithm

A. *Step planning for a single link*

Consider a single link, denoted PQ, whose endpoints are P and Q, Figure 2(a). Also, consider a *unit motion* of link PQ towards a point P', which places the link at a new position $P'Q'$ such that the distance $|QQ'|$ is minimized over Q'. The optimal Q' is computed by the *Unit* operator defined as

$$Unit(PQ, P') = Q' \text{ such that}$$

$$|QQ'| = \min_X(|QX| : |P'X| = |PQ|)$$

When $|PP'| \to 0$, the locus of points Q' coincides with a curve called the *tractrix* [12]. One property of the tractrix is that $|QQ'| \leq |PP'|$, see Figure 2(a); This is called the tractrix *motion attenuation property* and is used extensively in the algorithm below.

Suppose the link unit motion from PQ to $P'Q'$, $Q' = Unit(PQ, P')$, results in an intersection with an obstacle

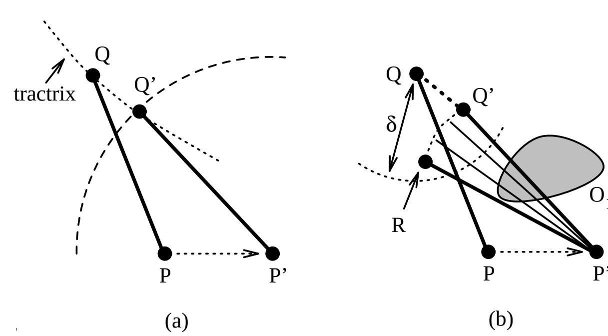

Fig. 2. (a) A unit motion of a link: given a rectilinear motion of an endpoint, the other endpoint moves along the *tractrix* curve. (b) If a unit motion causes an interference with a nearby obstacle, the link is rotated about its tip until the conflict is eliminated.

at a starting position S, and the head's target position T, generate continuous motion, collision-free for every point of the robot body, which takes the head from point S to point T. The orientation of the manipulator links during the motion or at T is not important. The motion will consist of a sequence of *steps*, which are computed and executed in real time (say, 50 steps per second) thus resulting in smooth motion. Each step involves in general all joint motors; to preserve continuity of motion, within one step no point in the robot's body should move further than the prescribed maximum, $\delta > 0$.

B. *Overview of the motion planning strategy*

Designing a strategy for sensor-based motion planning for a highly redundant kinematic structure forces one to address these two problems: (1) algorithm convergence and (2) the choice between the infinite number of possible link configurations at each motion step. The two issues are interrelated. In turn, the convergence question is two-fold: first, given the uncertainty of unknown arbitrary obstacles in the environment and the local nature of sensing, is it possible to design a provably-correct algorithm? Second, within one step the multiplicity of links makes a closed-form solution for the robot configuration unlikely; so, can one design an acceptable finite-time iterative procedure? Since the problem of provably-correct motion planning for even a point automaton in three-dimensional space is known to be intractable, the first question is not likely to have a positive answer. Thus, in this paper we consider a heuristic procedure. The convergence issue is being addressed partially, by choosing a provably-correct procedure for the arm head. Similarly, the number of iterations in the calculation of one configuration is bounded by the requirement that the iterations should not produce cycles in the links processed. In practice, the process stops very quickly.

The main question that we address here is the potential explosion of the number of kinematic solutions (configurations) due to redundancy. Rather than to struggle with the immense problem of inverse kinematic transformations with redundancy [2], we attempt to turn the problem around and to capitalize on the additional freedom offered by link multiplicity. In doing so, we choose to compute the next step configuration using a link-by-link propagation of "local" computations. Namely, the next step position of each link is computed based only on the obstacles around the link and the contemplated position of the previous link, without any knowledge about the distant links and/or obstacles. This simplifies the computation dramatically and produces a strategy that can be easily implemented in real time.

The effect of distant links is further reduced by the choice of a unit motion for a single link whereby the prescribed motion at one end of the link produces, except in degenerate cases, a smaller motion at its other end. Still, conflicts between distant links may occur. As an example, envision a real snake trying to move its head some place while a part of its body near its tail is stuck between two rocks.

Our strategy is based on the algorithm for free-snake motion planning [1], called here $\mathcal{A}_f$ (f for "free"). $\mathcal{A}_f$ attempts to emulate a passive reaction of the free snake body to a continuous "pull" at the head (or, in general, at any joint) and to the surrounding obstacles. To compute head motion, any maze-searching provably-correct algorithm can be used (see, e.g., [4]). We make use of an operator *HeadStep*, based on one such algorithm, which determines the next desired motion of the head. The output of $\mathcal{A}_f$ is a real-time step-by-step sequence of collision-free configurations of the free snake that eventually take it to the target. The input information for computing each step is the desired step motion at the head (given by *HeadStep*) and the sensing data about the surrounding environment. The computation involves a *propagation* procedure, which amounts to a serial head-to-tail calculation of each link's *unit motion*.

The direction of the unit motion vector is based on a specially-chosen curve called the *tractrix* [12], which has some desired locally-optimal properties. In particular, for a single link, the distance δ_b along the tractrix traversed by the base of the link due to a step motion δ_t at its tip obeys the relation $\delta_b \leq \delta_t$. That is, the tractrix guarantees a *monotonic attenuation* of motion. For the multi-link snake this means that a step motion at the head causes a much smaller offset at the tail. In other words, for a given motion at the head, no point at the robot body generates a larger motion.

The situation becomes more complex in the presence of obstacles. To assure collision avoidance, for every link's unit motion a check is made to determine if the contemplated position of the link interferes with any obstacles; if so, that link is rotated until the conflict is resolved. After the whole propagation calculation is complete, the snake moves to the new step configuration, and the process repeats.

In spite of the tractrix attenuation property, the $\mathcal{A}_f$ algorithm cannot be used directly for the arm manipulator since, in general, any motion of the head causes some motion at the tail. To account for the constraint of zero motion at the tail, we propose a new algorithm, $\mathcal{A}_m$ (m for "manipulator"), which consists of two basic propagation processes executed at each step. First, a head-to-tail propagation procedure similar to $\mathcal{A}_f$ takes place. It produces an arm configuration which places the head at the position contemplated by *HeadStep*, but it also results in a prospect of some motion at the tail, which violates

Sensor-Based Motion Planning for Highly Redundant Kinematic Structures: II. The Case of a Snake Arm Manipulator

Dan Reznik and Vladimir Lumelsky
University of Wisconsin-Madison
Madison, Wisconsin 53706, USA

Abstract—**This is a continuation of our work on sensor-based motion planning for highly-redundant kinematic structures. In [1], we considered a planar, snake-like robot freely moving amidst obstacles of arbitrary shape. Here, we assume that the tail of the snake is fixed, i.e., the snake is a redundant arm manipulator. The manipulator is capable of sensing obstacles in the vicinity of any point of its body. The task is to move the head of the manipulator from a starting position to a target one while avoiding collisions with obstacles. We present a procedure which avoids the computational explosion due to link multiplicity by emulating a passive reaction of the manipulator's body to a continuous "pull" at the head and to the surrounding obstacles. This results in a local link-by-link (instead of a global closed-form) processing, and produces an efficient real-time algorithm. A computer simulation showing robust motion planning in an obstacle-filled environment is presented.**

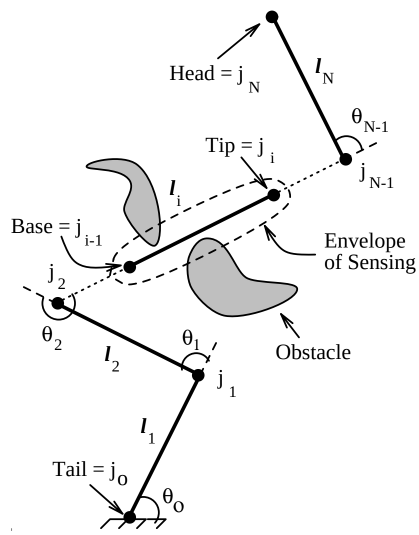

Fig. 1. An N-link snake arm manipulator and its workspace. A link l_i is shown with its envelope of sensing (dotted line).

I. Introduction

A. Overview of the problem

We consider the problem of motion planning with *incomplete information* for a highly redundant kinematic structure called a *snake arm manipulator*, shown in Figure 1. The manipulator is immersed in a planar *workspace* populated by unknown stationary obstacles of arbitrary shapes.

The manipulator is composed of a serial chain of *links* $l_i, i = 1 \ldots N$, connected to each other through *revolute joints* $j_i, i = 0 \ldots N-1$. Each link is a straight segment of length L. A link's endpoints are called *base* and *tip*; for link l_i, these coincide with joints j_{i-1} and j_i respectively. In particular, the tip of link l_N is called the *head* and the base of link l_1 is called the *tail*, denoted j_N and j_0, respectively. The head can carry, for example, a gripper or a camera. The tail endpoint is fixed in the workspace. Apart from this constraint at the tail, the arm manipulator is identical to the *free snake* studied in [1], which serves as the basis for this work.

The robot's input information comes from its *sensors* which are capable of acquiring accurate information about obstacles within a fixed radius from every point of the robot body (this is a good model of, for example, sonar or infrared sensors). This kind of sensing effectively provides every link with an *envelope of sensing* within which the environment is fully known. Apart from sensing, no information about the workspace is available.

Each joint $j_i, i = 0 \ldots N-1$, is associated with an angle θ_i measured between links l_i and l_{i+1}. Angle θ_0 is defined as the angle between link l_1 and the horizontal. The N-tuple $C = (\theta_0 \cdots \theta_{N-1})$ of joint angles uniquely defines the manipulator *configuration*. The latter can also be uniquely described by a vector of the cartesian coordinates of the joints, $M = (j_0, \ldots, j_N)$. Depending on the implementation, the robot control system can prefer one or the other representation.

We consider the following *motion planning task*: given an initial configuration of the manipulator, with its head