Λ ~ - - ''10 - -12

SAND--91-1871C

DE92 000229

SANDROS: A Motion Planner with Performance Proportional to Task Difficulty*

Pang C. Chen Yong K. Hwang
Sandia National Laboratories
Albuquerque, NM 87185

September 10, 1991

Abstract

To address the need of a practical motion planner for manipulators, we present an efficient and resolution-complete algorithm that has performance commensurate with task difficulty. The algorithm uses SANDROS, a new search strategy that combines hierarchical, nonuniform-multi-resolution, and best-first search to find a near-optimal solution in the configuration space. This algorithm can be applied to any manipulator, and has been tested with 5 and 6-degree-of-freedom robots, with execution time ranging from 20 seconds to 10 minutes on a 16 MIPS workstation.

1 Introduction

Every robotic system requires a motion planner for manipulators to execute tasks without colliding with objects in the workspace. Motion planning of a manipulator refers to finding a short and collision-free path from the start to goal position of the manipulator. At present, manipulators are either controlled by human operators [TeVTS91] or restricted to follow trajectories that have been pre-computed through hours of off-line programming [Loza87]. Human operators, however, have a fatigue factor, and need expensive safety equipment to work in hazardous environments. On the other hand, off-line programming can only be used in a highly structured and well-controlled environment with little tolerable positional errors and no unexpected object encounter.

*This work has been performed at Sandia National Laboratories and supported by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

Automation of motion planning offers a number of advantages over the existing alternatives. It relieves human workers of the continual burden of detailed motion design and collision avoidance, and allows them to concentrate on the robotic tasks at a supervisory level. Robots with an automatic motion planner can accomplish tasks with fewer, higher-level operative commands. Robotic tele-operations can then be made much more efficient, as commands can be given to robots at a coarser time interval. With an automatic motion planner and appropriate sensing systems, robots can adapt quickly to unexpected changes in the environment, and be tolerant to modeling errors of the workspace.

Unfortunately, it is unlikely [Reif79] to have a uniformly fast motion planner capable of finding a collision-free motion within a few seconds for all tasks. Hence, the next best alternative is to have an algorithm with performance commensurate with task difficulty. In this paper, we present an efficient, resolution-complete motion planner with exactly this characteristic. It can be used with any manipulator, and can solve "easy" problems very quickly and "difficult" problems systematically with a gradual increase of computation time. For typical problems and robots with 5 or 6 degrees of freedom (dof), the running time ranges from 20 seconds to 10 minutes. The overall performance should become real-time with faster computers in the near future.

In the following, we review previous work in Section 2, and describe our algorithm in Section 3. We illustrate the algorithm with examples in Section 4, and evaluate its performance in Section 5. We assume a polyhedral representation of the robot and its environment, and use the terms manipulator and robot interchangeably to refer to the ensemble of the robot and the objects being handled.

2 Previous Work

Planning motions for a general 6-dof manipulator is very difficult [Reif79], and there are presently no practical motion planners that guarantee solutions. Motion planners can be classified into two categories: exact or heuristic. Exact motion planners are slow but sure in that they require long computation time (typically hours for 6 dof) [Loza87, PaMF89, BrNe90, Cann88], but can either find a solution if there is one, or prove that there is none. Heuristic motion planners are fast (typically under 10 minutes), but they often fail to find a solution even if there is one.

There are five known exact motion planners implemented at present, and they all require long computation times except that in [Kond91]. The motion planner in [Loza87] takes about 15 minutes to plan motions for a typical 4-dof manipulator problem, and hours for a 6-dof manipulator. It should be noted that the computation times for 5 and 6-dof robots are important, because they are useful for general purpose manipulation. (A 6-dof robot is required to arbitrarily position and orient an object within the robot's workspace.) Other exact motion planners have been discussed in the research literature, but have not been implemented on 5 or 6 degree of freedom robots due to their extensive computation time requirements. Paden et al. [PaMF89] have presented an exact algorithm which takes 35 seconds on a 1 MIPS computer for a 2-dof robot. Its estimated computation times for 5 and 6 degrees of freedom robots are also too long to be practical. Branicky et al. [BrNe90] have developed a motion planner that works fast for robots with up to three degrees of freedom. This approach cannot be implemented for robots with higher degrees of freedom because of huge memory requirements and long search time. In the Roadmap algorithm and its variant [Cann88, CaLi90], a collision-free motion is found from a one-dimensional skeleton of the free space in the configuration space. They are not implemented for 6-dof robots, but expected to take a long time. Kondo [Kond91] has reported an exceptionally fast exact algorithm for 6-dof robots. We believe, however, that this algorithm would take substantially more time for harder yet realistic problems. (See Section 3.3 for more discussion.)

There are many heuristic motion planners [FaTo87, HaTe88, BaLa90] with computation times less than 10 minutes. However, they often fail to find a collision-free motion even if there exists one. These motion planners are typically developed for specific applications, and cannot be used for general situations.

3 Algorithm

We have developed an efficient, resolution-complete algorithm capable of planning motions for arbitrary robots operating in cluttered environments. (Our notion of efficiency and completeness will be discussed in Section 3.3.) We plan a motion of a robot in the *configuration space.* The configuration space is a representation of possible robot motions through the use of robot joint positions and angles. For a robot manipulator with six joints, i.e., six motors, the configuration space is six dimensional. The following design decisions are crucial in developing an effective motion planner without sacrificing completeness.

First, we use the distance algorithm in [GiJK88] to compute collision-free parts of the configuration space. One can use contact conditions between the robot and obstacles to get a range of collision-free joint values as done in [Loza87]. However, the distance information between the robot and the obstacles at a particular configuration allows us to select "safer" configurations for the robot to follow.

Second, we use a two-level hierarchical planning scheme to reduce memory requirement as done in [FaTo87]. It is difficult to store all the collision-free points even if we could compute them all, since the configuration space typically has an enormous number of points even at a coarse resolution. We circumvent this problem by planning at two levels using a a global and local planner. The global planner keeps track of reachable, unreachable, and potentially reachable portions of the configuration space, and the local planner is used to check the reachability of a portion of the space from a point. If a portion of space is reachable from a point, then the corresponding collision-free motion need not be stored as in [CaLi90], since they can be readily recovered by the local planner at any later stage of the algorithm. Although this scheme requires re-computing the solution path, it is more efficient than storing hundreds of path segments generated during the search process.

Third, we use a multi-resolution approach to reduce search time. The high dimensionality of the configuration space has hindered the development of a fast, complete motion planner. An exhaustive search for a collision-free motion is prohibitive because of the enormous size of the configuration space. Yet, heuristic algorithms that do not examine the entire space are inevitably incomplete. To achieve both time efficiency and completeness, we use the global planning module to first search promising portions of the configuration space at a coarse resolution. It increases the resolution to finer levels only if a solution is not found at the coarse level, and only in promising portions of the configuration space. The planner searches the space both heuristically and systematically so that each motion planning problem can be solved in time according to its difficulty.

These design decisions are embodied in a new search strategy called SANDROS, which stands for *Selective And Non-uniformly Delayed Refinement Of Subgoals.* Given two points s and t representing the start and goal configurations of a robot, the planner maintains a set of subgoals to be used by the robot as guidelines in moving to the goal configuration. Subgoals represent portions of the configuration space that have relatively large clearances to obstacles, and hence correspond to configurations that are easy for the robot to reach using the local planner. Initially, the planner maintains only a small number of "big" subgoals, each of which represents a large portion of the configuration space. Because these subgoals are big, they provide only coarse guidelines for the robot to follow. A collision-free motion can be found very quickly with only these coarse guidelines if the problem is easy. However, if the robot cannot find a collision-free path with these subgoals, then some of the subgoals are broken down to several smaller, heuristically selected subgoals to provide more specific guidelines. The process of subgoal refinement is delayed as much as possible, and is performed in a non-uniform fashion.

At the highest level, the SANDROS planner uses a *generate-and-test* strategy to plan motions. It has two main modules: a *global planner* $\mathcal{G}$ that generates a plausible sequence of subgoals to guide the robot, and a *local planner* $\mathcal{L}$ that tests the reachability of each subgoal in the sequence. If $\mathcal{L}$ succeeds in reaching each subgoal through the tested sequence, then a collision-free path is found. If $\mathcal{L}$ fails to reach a subgoal due to collisions, then $\mathcal{G}$ would first try to find another sequence without any subgoal refinement. If no sequence is available, then the current set of subgoals would be refined repeatedly according to the SANDROS strategy until either a sequence becomes available, or no further refinement is possible.

It should be noted that the global planner is completely independent of the local planner. However, there is a trade-off between the local planner's simplicity and range of effectiveness. If the local planner is as complicated as one of the exact algorithms mentioned above, then the global planner will never generate any subgoals because the local planner's range of effectiveness encompasses the entire configuration space. At the other extreme, if the local planner is a simple algorithm like connect-with-straight-line, then the burden of planning rests almost completely on the global planner in that it will have to generate many subgoals before a solution can be found. We have empirically found that the overall computation time is the shortest when the local planner implements a hill-climb algorithm (see Section 3.2). The global and local planners are explained in the following subsections, and illustrated with an example in Section 4.

3.1 Global Planning

Global planning takes place in three stages: sequence generation, sequence verification, and node refinement. In the sequence generation stage, the global planner $\mathcal{G}$ finds a "good" sequence of subgoals by searching through a dynamic graph G containing s and t with additional nodes representing subgoals. For a robot with n degrees of freedom, a node at (refinement) level k is an n-vector with only the first k components specified. A point is an n-vector with every component specified. Thus, the totally unspecified node v_0 at level 0 represents the entire configuration space, and a fully specified node represents a single point. A node v is considered reachable from a point p if $\mathcal{L}$ is able to find a collision-free path from p to a point q in v.

The nodes of G are divided into three sets: U, the reachable nodes, V, nodes not yet reachable, and P, the points that are reached when a node is declared reachable by $\mathcal{L}$. Set P is further divided into P_s and P_t representing points reachable from s and t, respectively. The edges of G establish three types of connections: those between P_s and V, P_t and V, and those within $U \cup V$. The edge cost between two nodes is defined as the sum of the differences between the coordinates that are specified in both nodes (a modified Manhattan metric). Nodes are connected by an edge only if the edge cost does not exceed a certain threshold T_e. For each point in P, we also store a point cost indicating the cost of reaching it from s or t. We initialize G by setting $P_s = \{s\}$, $P_t = \{t\}$, $U = \{\}$, $V = \{v_0\}$, and connecting s to v_0 and v_0 to t with edges. To control the subgoal refinement process, we also maintain a node queue Q, initialized to the empty set.

To generate a plausible sequence, we simply apply Dijkstra's shortest-path algorithm [AhHU74] on $G \setminus U$ with source nodes P_s and sink nodes P_t. We define the cost of a sequence with end points in P and intermediate nodes in V as the sum of the edge costs plus the costs of the end points. This cost serves only as an estimate of the actual length of a solution going through the subgoals. To restrict the number of possible sequences through a subgoal, we adopt the principle that no other ways of reaching a subgoal would be considered once it is declared reachable. We implement this principle by associating exactly one point in P with every node in U. Allowing more than one point per node is also possible, but we favor the one-point-per-node rule for its conceptual and implementational simplicity. (We will allow a reached node to be reached again at other points when the node is split into smaller nodes through refinement.)

If Dijkstra's algorithm produces a sequence, then we enter the sequence verification stage. In this stage, we determine the connectability of the sequence by searching bi-directionally with $\mathcal{L}$. Let $s' \in P_s$ and $t' \in P_t$ be the end points of this sequence. Let $d(q)$ be the minimum Euclidean distance between the robot at configuration q and the obstacles. We begin by choosing the search direction using $d(s')$ and $d(t')$ as a guide: If the $d(t')$ is smaller than $d(s')$, then we will search backward by starting at t'; otherwise, we will search forward by starting at s'. Heuristically, the point chosen (s' or t') should have less clearance from the obstacles, and hence should be extricated first to constrain search. Let p be the point chosen and p' be the other point. To search forward, we call $\mathcal{L}$ to check the reachability of the first node v from point p; to search backward, we call $\mathcal{L}$ to check the reachability of the last node v from point p. Either way, if v is reachable, then the corresponding point reached, q, has to agree with v on all of its specified coordinates. Once v is *focused* onto q in this fashion, we

1. swap v from V to U,
2. insert q into P,
3. store a pointer $A(q) = v$,
4. connect q to the original neighbors of v in V with new edges, and
5. store a back pointer $B(v) = p$, so that a path from s or t to q can be retraced.

Then, we continue the verification stage by checking the connectability between p' and q.

The verification stage ends when either the sequence is connected, or a node v is found unreachable from a point p. In the former, we retrace a path from s to t through G to yield a motion for the robot. In the latter, however, we disconnect p from v, push both $A(p)$ and v into Q, and return to generating another sequence.

Continuing with the sequence generation process, if Dijkstra's algorithm produces no candidate sequence, then we enter the node refinement stage. In this stage, we modify G continually by refining the subgoals in Q until either a candidate sequence becomes available, or Q becomes empty. We pop off every node v in Q at the minimum refinement level k, and refine each as follows: First, we use the specified components of v as the first k joint values for the robot. Then, using only the links whose positions are totally specifiable by the first $k+1$ joints, we compute the distance between the links and other objects in the workspace for all possible $(k+1)^{\text{th}}$ joint value at a prescribed resolution. The process is similar to that reported in [Loza87]. The resulting set of nodes with $k+1$ specified components is then filtered into a list of nodes $C(v)$ using a *dominate-and-kill* method. In this method, the process of selecting a node v with the maximum distance value, and removing each node whose $(k+1)^{\text{th}}$ component is within λ number of nodes of v is repeated until every node is considered. The idea is to condense the set of possible nodes into a sparse collection of heuristically good subgoals, each having a maximal clearance based on its specified links.

After refining v into $C(v)$, the next step is to modify G to reflect the change in V. We augment G with $C(v)$ by inserting $C(v)$ into V, and connecting every neighbor node of v with every node of $C(v)$. Also, if $A(B(v))$ has not been refined already, then we push $A(B(v))$ into Q to ensure the eventual chance that every node of U gets refined.

3.2 Local Planning

The local planner simulates robot movements in the configuration space in small steps. A step is defined as a configuration change where each joint is changed by a preset amount, which represents the resolution. This preset amount of change in each joint is indicated by a number called ***stride***. The stride in each joint is normalized so that the maximum distance traveled by any point on the robot is about the same for each stride. A point that is within one step of another is a neighbor of that point. Collision checking is done after the robot takes a step. It is thus possible for the robot to collide with the objects while taking a step, although it does not collide before and after taking the step. We will assume, however, that given small enough strides, such collisions are very minor and can be ignored.

The local planner $\mathcal{L}$ checks the reachability of a node v from a point p by moving the manipulator from p to a point q in the subspace of v. Recall that a point has all the coordinates specified, and a node has partially specified coordinates. The iterative procedure of moving p toward v is as follows: First, we make progress toward v by sampling uniformly for a number (n^2) of the $3^n - 1$ neighbors of p that are closer to v, and picking among them the point p' that has the maximum clearance $d(p')$. Next, we slide in the unspecified dimension by sampling for an additional number (n^2) of neighbors of p' with the specified coordinates unchanged, and picking among them the final point q with the maximum $d(q)$. If no progress can be made, then $\mathcal{L}$ would report a failure; otherwise, the procedure is repeated until v is reached.

3.3 Completeness and Efficiency

To recapitulate, our algorithm searches for a solution by repeating the process of finding a promising sequence of nodes in a graph G, verifying its feasibility with $\mathcal{L}$, and modifying G with the refinement procedure. The efficiency of our algorithm comes from the fact that we use a non-trivial λ in the refinement procedure, and that we delay the refinement of nodes until there is no sequence in the current G.

It is of course possible to refine every node of G down to a point first, and then plan a path based on the resulting network of points $\bar{G}$. In fact, if we were to use $\lambda = 0$, then the resulting network is simply the discretized map of the free configuration space. However, it would be terribly inefficient to find the shortest plausible sequence of subgoals because of the size of $\bar{G}$. For such a $\bar{G}$, $\mathcal{L}$ only needs to check whether an adjacent point is collision-free. Non-trivial λ allows us to combine similar subgoals into one single subgoal so that G contains only a small number of subgoals that are also representative of all regions of the free configuration space. We therefore reduce the size of $\bar{G}$ by using a larger λ, and utilize the power of $\mathcal{L}$.

To further minimize the number of nodes in G, we do not search for a solution from $\bar{G}$, the completely refined graph. Rather, we refine G only if we cannot find a solution with G at the current refinement level. It remains to show that SANDROS' interleaving process of search and refinement will eventually find a solution if there is a solution in the completely refined graph $\bar{G}$.

Theorem 1 *Suppose that a task of moving from s to t is solvable by first refining the configuration space into a network of points $\bar{G}$, and then planning a path through $\bar{G}$ using $\mathcal{L}$ to connect s to t. Then our algorithm is complete in that it can also solve the same problem, but with possibly less node refinement.*

Proof If the problem of moving from s to t is solvable through total refinement, then there must be a loopless sequence

$$\Gamma_0 = (s = v_0, v_1, \ldots, v_n = t)$$

with $v_i \in \bar{G}$ such that $\mathcal{L}$ is able to connect v_i with v_{i+1}) for all $i < n$. Suppose that our algorithm fails to solve the same problem, and terminates with a partially refined $G \neq \bar{G}$. Then consider the sequence

$$\Gamma_1 = (\varphi(v_0), \ldots, \varphi(v_n)),$$

where $\varphi(v_i)$ denotes the node of G that dominates v_i in that the specified coordinates of $\varphi(v_i)$ match that of v_i. By contracting any loop of this sequence repeatedly, we can obtain a loopless (but not unique) sequence of the form

$$\Gamma_2 = (s = w_0, w_1, \ldots, w_m = t)$$

with $m \leq n$, and each w_i in G. Since Γ_2 contains no repetition of nodes and has cost less than that of Γ_0, it must be a sequence verified by $\mathcal{G}$ to be infeasible. Now, since Γ_0 is loopless, every fully expanded node w_i in Γ_2 must correspond to a unique v_j in Γ_0. Further, for such $i < n$, w_{i+1} must dominate v_{j+1}. Hence, for Γ_2 to be infeasible, there must exist a smallest $k \geq 1$ such that w_k strictly dominates v_k in that w_k has at least one unspecified coordinate. On the other hand, such k cannot exist for the following reason. If w_k were reachable by the end of our algorithm, then w_k would eventually be pushed into Q. (See the last portion of Section 3.1.) If w_k were not reachable, then w_k would have been pushed into Q immediately after this determination. Either way, w_k would have been expanded eventually, implying that it is actually a point, and hence cannot strictly dominate v_k. Therefore, our algorithm must have also succeeded by contradiction. ∎

Motion Planner	2 dof	4 dof	5 dof	6 dof
Lozano-Pérez	6 sec	15 min	3 hr*	40 hr*
Paden	4 sec	5 min*	2 hr*	50 hr*
SANDROS	5 sec	2 min	5 min	10 min

Table 1: Time comparisons scaled to a 10 MIPS computer
(* indicates estimated times)

We now examine experimentally the efficiency of our algorithm by comparing it with other algorithms. Two things must be considered when comparing the performances of motion planners: what kind of problems they can solve, and how fast they can solve them. Exact planners are by definition better than heuristic planners since they guarantee a solution. However, a fast heuristic planner that finds solutions for many realistic problems has an advantage in that it can be applied to a problem before applying an exact planner, which is typically computationally expensive. Heuristic algorithms are harder to compare as they tend to be application specific. From the practical point of view, efficiencies of motion planners should be compared with the average-case analysis rather than the worst-case theoretical complexity. Comparing the performances of motion planners has been difficult, since there is no set of bench-mark problems that represent *realistic* and *non-pathological* motion planning problems. We urge future motion planning researchers to include at least one example satisfying the following criteria. First, the number of obstacles should be between 5 and 10, and some obstacles should be concave. Second, the solution path should be non-trivial and utilize all dofs available. Third, there should be a narrow space at some point along the solution path so that the problem cannot be solved with a coarse discretization of joint variables. Fourth, there should be a large but not artificial trap in the space so that some back-tracking is required to find a solution. In Section 4, we show an example that meets all these requirements.

Since our goal is to develop an exact motion planner for 6-dof general manipulators in a three-dimensional world, we compare the performance of our algorithm with only such algorithms [Loza87, PaMF89, Kond91]. Because the actual computation times of the algorithms in [Loza87, PaMF89] for 5 and 6-dof problems are not available, they are estimated conservatively based on a resolution of 30 points per dimension and linear extrapolation. We should only consider the orders of magnitude of these times, because of the sparsity of examples and the different examples and computers used. As can be seen in Table 1, our algorithm performs much better for 5 or 6 dof robots, which are common in real applications. Kondo's algorithm in [Kond91] uses an average of 7000 collision detections to solve 7 different 6-dof problems, as opposed to 10000 collision detections in our algorithms. His examples, however, use rather a coarse discretization (5 degrees as compared to 2 degrees in our examples), and we expect the algorithm to use many more collision detections and have memory limitation problems when a finer resolution is used. Kondo suggests the use of 2^n-tree to solve the memory

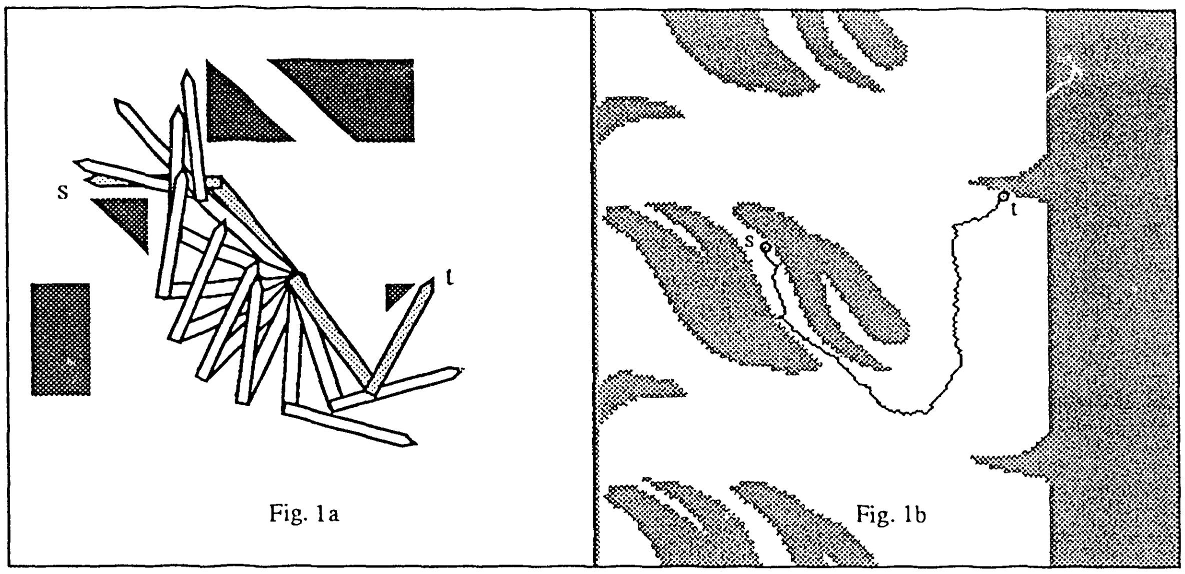

Figure 1: A 2-link robot example with SANDROS' solution shown in both world (a) and configuration (b) space.

problem, but our experience tells us that such a hierarchical representation has a lot of overhead associated with keeping track of adjacency relationship among the cells as in [PaMF89].

4 Examples

We have tested our planner with a planar 2-link robot (RR), Puma (RRRRRR), Adept (SCARA robot, RRPRR) and a gantry robot (PPPRRR with a long wrist) in computer simulations, and actually implemented it on the Adept robot. A 16 MIPS Silicon Graphics workstation is used to implement the SANDROS planner. For the planar 2-link robot, the computation times are all less than 30 seconds for problems with 5 to 7 obstacles. For the example in Figure 1a, it takes 3 seconds to find the solution. The configuration space obstacles are shown in Figure 1b.

Figures 2a through 2f show the intermediate stages of the search performed in the configuration space. Initially, the global planner $\mathcal{G}$ invokes the local planner $\mathcal{L}$ to connect the start configuration s and the goal configuration t directly. Since $d(t) < d(s)$, the robot chooses to search backward from t to s. Because of the greedy nature of $\mathcal{L}$, it was not able to avoid collision at the point marked by a triangle (Figure 2a).

Thus, the whole configuration space is refined into several smaller subgoals corresponding to having the first joint angle θ_1 specified. Since the refinement process includes a filtering stage, the resulting "good" subgoals with only θ_1 specified (shown

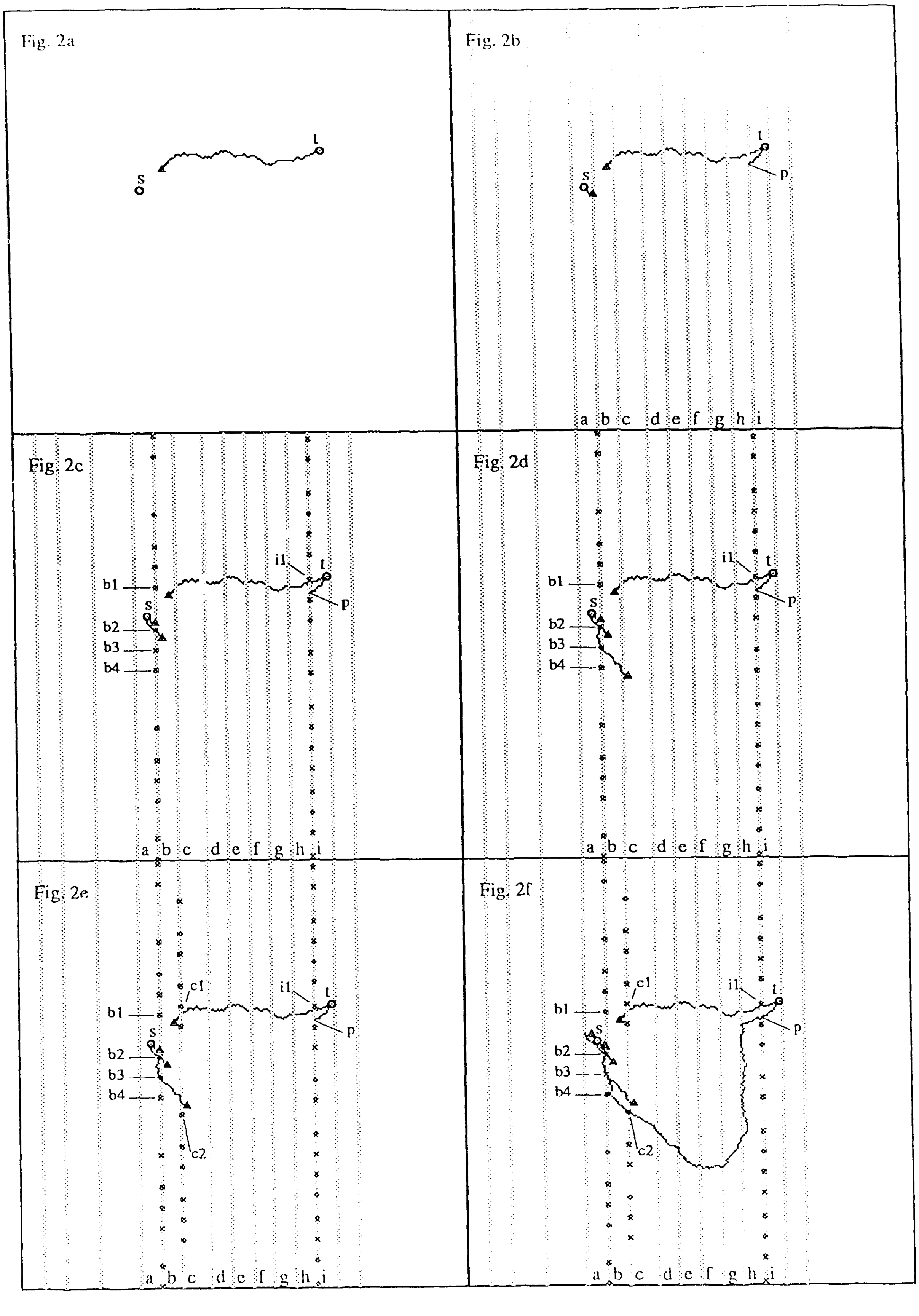

Figure 2: Search stages of the SANDROS planner.

with vertical bars in Figure 2b) have large clearances between the first link and the obstacles.

At this stage, the shortest sequence of subgoals between s and t is s-b-c-d-e-f-g-h-i-t. The local planner succeeds in connecting t to node i at point p, and s to node b, but fails to connect node b to node c. Since there is no other sequence of subgoals available, the reached subgoals farthest from s and t, which are nodes b and i, are refined to generate smaller subgoals. These new subgoals have the second joint angle θ_2 specified as well as θ_1. These subgoals (shown with dark squares in Figure 2c) correspond to positions of the second link with large clearances to the obstacles. After augmenting the graph of subgoals with these new nodes, the process of finding and verifying a shortest sequence continues.

The shortest sequence is now s-$b2$-c-...-p-t. but $\mathcal{L}$ fails to connect b_2 to c. The next shortest sequence is s-$b2$-$b3$-c-...-p-t, but $\mathcal{L}$ also fails between c and d. These failures are shown in Figure 2d.

Since there is no sequence of subgoals, node c is refined (Figure 2e). The shortest sequence is now s-a-$b1$-$c1$-d-...-$i1$-t. When the robot reaches node a, it is too far from $b1$, so $\mathcal{L}$ considers them unconnectable.

After two more sequences are tried unsuccessfully, a solution is finally found (Figure 2f) by following sequence s-$b2$-$b3$-$b4$-$c2$-d-...-p-t. The total number of points, bars, and squares approximately corresponds to the number of distance computations performed. Only a small portion of the configuration space is examined in finding the solution, hence the increase in performance.

For the three-dimensional problems tested, we model the gantry, Puma, and Adept robots with 5 to 8 polyhedra in environments filled with 5 to 7 polyhedral obstacles. Figure 3 shows the Adept robot pulling an L-shaped object out of a wicket, simulating a disassembly task. This particular example takes 3 minutes of computation. In other examples tested, the computation time ranges from 20 seconds to 10 minutes, depending upon the complexity of the task and environment.

5 Conclusions

We have presented a general motion planner for manipulators that has performance commensurate with task difficulty. The efficiency of our algorithm can be attributed to the following design features. We use a bi-directional, two-level search scheme to guide the robot out of tight places and reduce memory requirement. We use large subgoals to provide coarse guidelines and limit search, and only when finer resolutions become necessary do we selectively and sequentially refine these subgoals. We use the heuristic of best-first in selecting both promising parts of the configuration space for increased resolutions, and promising sequence of subgoals to be tested with the local planner. Finally, we use a local planner of an empirically optimal complexity and power so that the total computation time spent by the global and local planners are minimized.

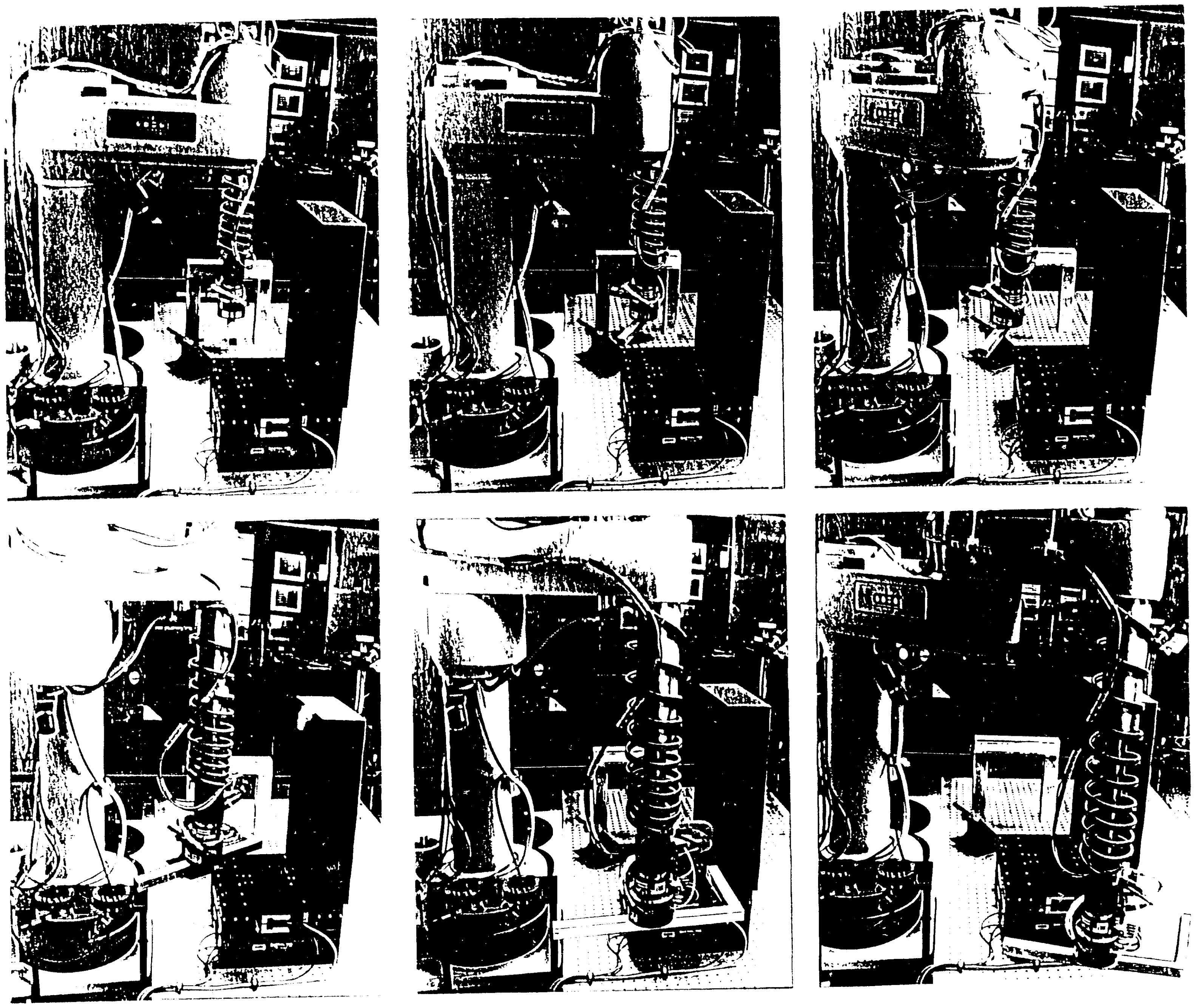

Figure 3: An Adept robot problem and solution.

One drawback of our algorithm is that it is a gross motion planner, and cannot be used for fine motion planning, e.g., peg-in-a-hole. One should also note that although our algorithm can be used for the classical mover's problem, it may not be as efficient as for the manipulator case. A single rigid body in three dimensions must have at least the position of its reference specified in order to do any partial collision detection. Hence, the largest subgoals we can have are three dimensional. As a result, our algorithm has to start the search with a large number of three dimensional subgoals, thereby having less potential savings than that for manipulators.

In summary, our algorithm is a judicious combination of several important ideas developed in motion planning over the past decade [FaTo87, Loza87, GiJK88]. To obtain a planner significantly faster than ours would seem to require using massively parallel machines, or taking a totally different approach utilizing sensors or additional knowledge. The next line of research would be to develop a fully functional motion planner by incorporating our algorithm with a fine motion planner.

Acknowledgement

We are grateful to Professor E.G. Gilbert of University of Michigan for providing the software for computing the distance between polyhedra. We also thank B.K. Christiansen, C.S. Loucks and L.P. Ray of Sandia National Laboratories for helping us execute our algorithm's solutions on the robots.

References

[AhHU74] Aho, A., Hopcroft, J., and Ullman, J. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[BaLa90] Barraquand, J. and Latombe, J., "A Monte-Carlo algorithm for path planning with many degrees of freedom," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1712-1717, 1990.

[BrNe90] Branicky, M. S. and Newman, W. S., "Rapid computation of configuration obstacles," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 304-310, 1990.

[CaLi90] Canny, J. F. and Lin, M. C. "An Opportunistic Global Path Planner," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1554-1559, 1990.

[Cann88] Canny, J. F., *The Complexity of Robot Motion Planning*, MIT Press, 1988.

[FaTo87] Faverjon, B. and Tournassoud,, P., "A local approach for path planning of manipulators with a high number of degrees of freedom," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 1152-1159, 1987.

[GiJK88] Gilbert, E. G., Johnson, D. W. and Keerthi, S. S., "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193-203, April 1988.

[HaTe88] Hasegawa, T. and Terasaki, H., "Collision avoidance: divide-and-conquer approach by space characterization and intermediate goals," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 18, no. 3, pp. 337-347, May/June 1988.

[Kond91] Kondo, K., "Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration", *IEEE Transactions on Robotics and Automation,* vol. 7, no. 3, pp. 267-277, June 1991.

[Loza87] Lozano-Pérez, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation,* vol. RA-3, no. 3, pp. 224-238, June 1987.

[PaMF89] Paden, B., Mees, A. and Fisher, M., "Path Planning Using a Jacobian-Based Freespace Generation Algorithm," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 1732-1737, 1989.

[Reif79] Reif, J. H., "Complexity of the Mover's Problem and Generalizations," *Proceedings of 20th IEEE Symposium on Foundations of Computer Science,* pp. 421-427, 1979.

[ScSh83] Schwartz, J. T. and Sharir, M., "On the Piano Movers' Problem: I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Communications on Pure and Applied Mathematics*, vol. 34, pp. 345-398, 1983.

[TeVTS91] Tendick, F., Voichick, F., Tharp, G. and Stark, L., "A Supervisory Telerobotic Control System Using Model-Based Vision Feedback," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 2280-2285, 1991.

DATE
FILMED
4 16 92

I