MAR 12 1997

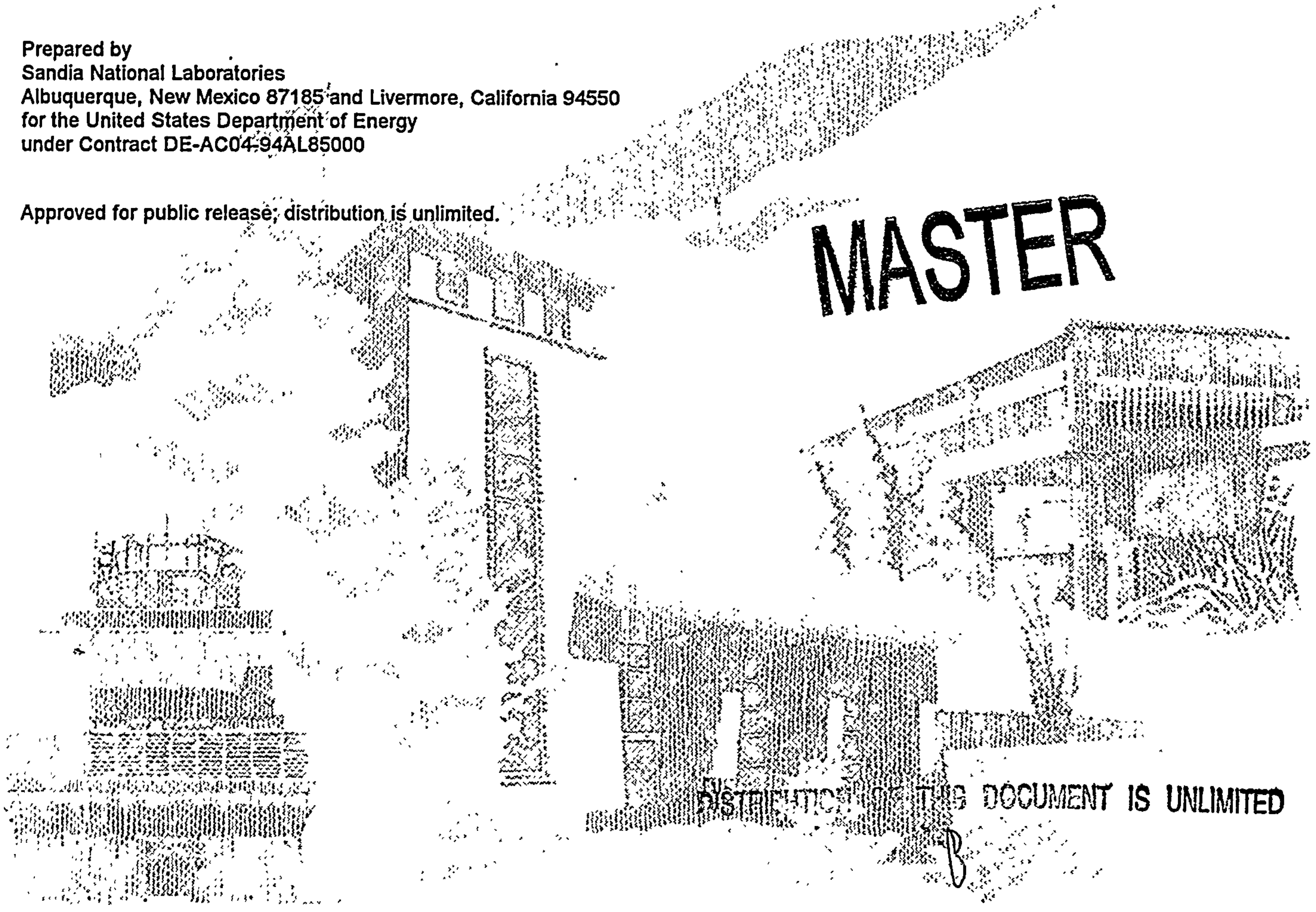
SAND--97-0367

SANDIA REPORT
SAND97–0367 • UC-~~705~~ 905
Unlimited Release
Printed February 1997

General Techniques for Constrained Motion Planning

Yong K. Hwang, Peter A. Watterberg, Pang C. Chen, Christopher L. Lewis

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000



SF2900Q(8-81)

SAND 97-0367
Unlimited Release
Printed February 1997

Distribution
Category UC-~~705~~605

General Techniques for Constrained Motion Planning

Yong K. Hwang, Peter A. Watterberg,
Pang C. Chen, and Christopher L. Lewis
Intelligent Systems Principles Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1008

Abstract

This report presents automatic motion planning algorithms for robotic manipulators performing a variety of tasks. Given a task and a robot manipulator equipped with a tool in its hand, the motion planners compute robot motions to complete the task while respecting manipulator kinematic constraints and avoiding collisions with objects in the robot's work space. To handle the high complexity of the motion planning problem, a sophisticated search strategy called SANDROS is developed and used to solve many variations of the motion planning problem. To facilitate systematic development of motion planning algorithms, robotic tasks are classified into three categories according to the dimension of the manifold the robot tool has to travel: visit-point (0 dimensional), trace-curve (1 dimensional) and cover-surface (2 dimensional) tasks. The motion planner for a particular dimension is used as a sub-module by the motion planner for the next-higher dimension. This hierarchy of motion planners has led to a set of compact and systematic algorithms that can plan robot motions for many types of robotic operations. In addition, an algorithm is developed that determines the optimal robot-base configuration for minimum cycle time.

The SANDROS search paradigm is complete in that it finds a solution path if one exists, up to a user specified resolution. Although its worst-case time complexity is exponential in the degrees of freedom of the manipulator, its average performance is commensurate with the complexity of the solution path. Since solution paths for most of motion planning problems consist of a few monotone segments, the motion planners based on SANDROS search strategy show approximately two-orders of magnitude improvements over existing complete algorithms.

1 Introduction

In manufacturing environments manipulators are used to perform repetitive tasks or physically demanding tasks unfit for human workers. Manipulators are also used in harsh, unstructured environments where it is expensive to use human workers for safety reasons. Moving the joints of a manipulator in a coordinated manner to perform a given task turns out to be a difficult problem, since the movement has to respect the manipulator kinematic constraints, task-related constraints on the tool paths, and avoid collisions with objects in the work space. Due to the non-linearities and complexities of these constraints, manual path planning and control is a long, tedious process.

Currently, robot motions for manufacturing tasks are usually planned manually using a teach pendant or simulation software [39, 40]. The teach-pendant method involves moving the actual robot with the teach pendant and recording robot joint angles. This method is labor intensive, taking about a week to plan motions to paint the interior of a car body. Simulation software provides a way to program and preview robot motions in graphical simulation. The robot and parts are modeled as graphical objects, and the robot motions are programmed by placing a sequence of graphical coordinate axes in the simulated workcell. The sequence of the coordinate axes specify the positions and orientations of the robot tool tip, and the programmed motion can be simulated by the graphical robot for reviewing. Although this method takes about a day, it is hard to plan 3-dimensional motions from a 2-dimensional computer screen and thus requires a long training period. Since different motions have to be planned for different tasks, the prolonged motion programming times of the above methods make it expensive to use robots in small-batch manufacturing.

In on-line, interactive applications where human operators are controlling robots directly, it takes a long time to train operators due to the difference between the kinematics of robot and human arms. Even after a rigorous training, human operators are prone to errors and accidents due to the tedious nature of motion control, i.e., an operator needs to specify the motion trajectory continuously in time to perform a task while avoiding robot-object collisions. Both off-line and on-line applications can benefit from an automatic motion planner for the reasons stated above.

Automatic motion planning is difficult because of the complexity of the collision avoidance problem and nonlinear robot kinematics. The high dimensionalities of robot joint space (often 6 or more) and the tool tip space (6, 3 for position, 3 for orientation) make it difficult to use a brute force search to find a solution. For example, several hours are required just to compute the set of joint angles that do not cause collisions between robot links and objects, let alone find a path in the collision-free set of joint angles. The nonlinearities of robot forward and inverse kinematics also make it computationally intensive to map the constraints described above from the world space to the joint space and vice versa.

The motion planning algorithms in this paper perform a global search in the joint space to find a near-optimal solution path that satisfies all three types of constraints: task constraints, kinematic constraints and collision constraints. Robotic tasks are classified into four classes according to the dimension of the manifold the robot tool has to cover: visit points (0-dimensional tasks), trace-curves (1D tasks), cover-surfaces (2D tasks) and build-volumes (3D task). The 3D tasks were realized during the development of the 0 to 2D motion planners, and only a brief sketch on how to extend our algorithm is described in this paper. Some example tasks for each class are

0 dimensional tasks (*visit-point*): sampling, marking and spot welding.
1 dimensional tasks (*trace-curve*): sheet-metal cutting, seam welding, edge deburring and caulking.
2 dimensional tasks (*cover-surface*): spray painting, polishing, surface inspection.
3 dimensional tasks (*build-volume*): rapid prototyping.

Based on this classification, a hierarchy of motion planning algorithms has been developed based on a novel search strategy called SANDROS [10, 19]. The motion planners are hierarchical in that the motion planner for a particular-dimensional task decomposes the task into a series of lower-dimensional tasks and invokes the motion planners for lower-dimensional tasks. This paper also presents a method to select the optimal robot-base position and orientation that results in the shortest path for the robot manipulator.

The main contribution of this paper is the systematic classification of robotic tasks and the development of motion planners according to this classification. The efficiencies (computation times) and powers (success rates of finding a solution path) of our motion planners have shown to be one of the best at the present time. Our motion planners are expected to make great impacts on virtual and agile manufacturing by allowing automatic planning and programming of robots, and verification of feasible motions for manufacturing operations in simulation. For on-line, interactive applications, our motion planners perform detailed motion path generation so that robot operators can concentrate on the tasks at a supervisory level.

For the balance of this paper, we use the terms *robot* and *manipulator* interchangeably. We call the joint space of a manipulator the *configuration* space (C-space), since a set of joint angles completely determines the position of every point of the robot. This paper is organized as follows. Section 2 reviews the previous work related to motion planning. In the next section, two basic motion planners are presented: a point-to-point motion planner and a motion planner for tracing a single curve. These planners are used in Section 4 to develop motion planners for 0-2D robotic tasks, and their performances are demonstrated in simulation. Finally, conclusions and future work are discussed in Section 5.

2 Related Work

There are many research results in point-to-point motion planning and dual-/multi-arm coordination. In point-to-point motion planning, a collision-free path between the start and the goal configurations of a manipulator is typically found by searching the joint space of the manipulator (see [18, 25] for surveys of various approaches). Since the joint space usually has many dimensions, *selective* search algorithms are required to perform the computation in a practical time limit. The algorithms presented in [10, 16, 19, 21] represent some of the most efficient (short computation time) and powerful (a high success rate) algorithms to date. We briefly review different approaches for motion planning here. A skeleton approach computes a one-dimensional skeleton of the free space, and a solution is found by computing a path from the start to a point on a skeleton, a path from the goal to the skeleton, and then connecting the two points on the skeleton. This approach is used in [8] to show that the motion planning problem has an upper bound that is single exponential in the number of degrees of freedom (dof). A randomized planning [21] is probabilistically complete in the sense that if it runs long enough, it will find a solution. In this approach, the robot moves toward the goal using a planner based on a potential field. If the robot cannot get to the goal, the robot takes a random walk, and then tries to approach the goal from there. Hierarchical multi-resolution search algorithms are used in [16] to develop motion planners that solve problems in time commensurate to the problem difficulty.

The work on motion planning for curve tracing is mostly done in redundant-robot research. The redundancy is used to optimize secondary objectives such as link collision avoidance [9, 27, 29, 34, 36],singularity avoidance [23, 29], cyclic (drift-free) joint motions for cyclic tool paths [9, 34], or manipulability measures [3, 38]. Most of the work concentrates on the method of solving the inverse kinematics for numerical stability, computational efficiency, and handling of kinematic and algorithmic singularities. The task prioritization approach is used in [27] to get a path that best satisfies the path-tracking and collision-avoidance requirements, whereas compact quadratic programming is used in [9]. Singular value decomposition is used in [22] to improve computational efficiency, and the extended Jacobian method is used to map algorithmic singularities in [23]. All of the above algorithms are, however, local methods; they use a greedy approach to avoid collisions and singularities while tracing the curve with the tool tip. These algorithms do not backtrack during search, and cannot solve problems requiring global-space knowledge such as that in Figure 1. The only global algorithms for curve tracing that we are aware of are presented in [20, 36]. These algorithms compute inverse kinematic solutions at the points along the curve, and find a connected sequence of robot joint angles that make the robot tool tip trace the curve.

There are relatively fewer efforts on motion planning for cover-surface tasks. Automatic painting path generation for slightly curved surfaces is done in [35], where the robot operates in space free of object clutter. Minimization of the thickness variation of spray coats along a specified path is considered in [2] by optimizing the robot speed along the path. Since there are many process parameter constraints besides geometric constraints in cover-surface tasks such as painting, most robots performing these tasks are manually programmed at present.

There has been some work done on workcell design and robot placement with different emphases or methods. A semi-automatic method of generating an assembly workcell layout is presented in [33]. A workcell designer first generates a rough layout of robots and other production resources. The visit points for the robot tool are generated from an assembly plan and part locations, then reachability of these points are checked. A numerical optimization routine then assesses the the quality of the layout, and computes a new layout. This process is iterated until a satisfactory layout is found. Paths between points are found after the layout is determined. This work is aimed at developing a layout algorithm for an overall assembly workcell. In contrast, our algorithm specifically determines optimal robot base configurations, with the optimality being the *actual, not estimated,* and *collision-free* path length of the robot. Determining an optimal robot placement for a robot performing a transfer movement between two points is considered in [14]. This algorithm uses a gradient descent rather than a brute-force search to find an optimal robot placement. This algorithm minimizes the robot execution time by taking into account the joint acceleration, whereas ours minimizes the length of the *collision-free* path that visits *multiple* points. Work on selecting robots and deploying them to proper work cells for a CIM system is presented in [11], but it is concerned with layout design of a whole manufacturing line from an industrial-engineering perspective.

3 Basic Motion Planners

The motion planners for 0 to 2D tasks are based on two basic motion planners: a point-to-point motion planner (PTP planner) and a motion planner for tracing a single curve (TSC planner). The PTP planner reported in [10] computes a collision-free path from one robot configuration to another, and is used by the visit-point (VP) planner to plan a path that visits multiple points. The TSC planner reported in [20] computes a robot motion that makes the robot tool tip trace a cürve while avoiding collisions between robot links and obstacles. This planner along with the PTP planner are used by the trace-curve planner (TC planner) to plan a path that traces multiple curves. We now describe the PTP and TSC planners.

3.1 PTP Planner

There are many motion planners that are either complete (guaranteed to find a solution if one exists) or efficient (short computation time), but only a few have both characteristics. The SANDROS planner in [10] is one of them, and will be used in this paper. We plan robot motions in the C-space, and make the standard assumption that if a task is solvable, a solution path can be represented by a sequence of unit movements in the C-space, discretized to a preset resolution. To handle the high dimensionality of the C-space and search complexity, we make the following design decisions without sacrificing completeness.

First, we determine whether a given point in the C-space is collision-free by computing the distance [15] between the robot and obstacles. Contact conditions can be used [12, 26], but equations describing contact boundaries are nonlinear, and detecting intersections of these boundaries is computationally expensive. The use of distance makes our algorithm independent of object models, and enables the selection of 'safer', i.e., larger clearance, configurations for the robot.

Second, we use a two-level hierarchical planning scheme to reduce memory requirement as done in [13]. It is difficult to store all the collision-free points even if we could compute them all, since the C-space typically has an enormous number of points even at a coarse resolution. We circumvent this problem by planning at two levels using a global and a local planner. The global planner keeps track of reachable, unreachable, and potentially reachable portions of the C-space, and the local planner checks 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 needs not be stored as in [6], 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. An exhaustive search for a collision-free motion is prohibitive because of the enormous size of the C-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 C-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 C-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, we maintain a set of subgoals to be used by the robot as guidelines in moving to the goal configuration. Subgoals represent portions of the C-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, we maintain only a small number of 'big' subgoals, each of which represents a large portion of the C-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 these coarse guidelines if the problem is easy. If a collision-free path cannot be found with these subgoals, 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 to minimize the number of nodes.

At the highest level, 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 a subset of the current subgoals would be refined repeatedly according to SANDROS strategy until either a sequence becomes available, or no further refinement is possible.

$\mathcal{G}$ is completely independent of $\mathcal{L}$, and that there is a trade-off between the simplicity of $\mathcal{L}$ and the guiding effort of $\mathcal{G}$. If the local planner is as complicated as a complete algorithm then the global planner will never generate any subgoals because the local planner's range of effectiveness encompasses the entire C-space. At the other extreme, if the local planner is a simple and inflexible algorithm like connect-with-straight-line, then the burden of planning rests heavily on the global planner in that it will have to generate many subgoals before a solution can be found. Finding the optimal subdivision of labor between $\mathcal{L}$ and $\mathcal{G}$ is difficult. However, in principle, $\mathcal{L}$ should implement some 'greedy' algorithm with a capability to 'slide' around simple obstacles, so easy problems can be solved without generating too many subgoals. Finally, we remark that the dynamic graph searching framework of SANDROS can be tailored to various kinds of robots by specifying the subgoal representation and refinement strategy in $\mathcal{G}$, and the sliding strategy in $\mathcal{L}$.

3.1.1 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 configuration points s and t with additional points and nodes representing subgoals. A point represents a single configuration in the C-space, while a node represent a subset.

The points of G are all points reachable from s or t through applications of $\mathcal{L}$. This set of points P is divided into P_s and P_t representing points reachable from s and t, respectively. A point cost is stored indicating the cost of reaching it from s or t. Each node v of G is classified as either reachable (in U) or not-yet-reachable (in V), depending on whether a reachable point p in P has been found with which $\mathcal{L}$ is able to find a collision-free path from p to a point q in v. Although G may be changing, we maintain the invariant that the nodes of G not intersect each other. Thus, each reachable point p has an unique node $A(p)$ of G that contains p.

There are two types of edge connections in G: (node)-to-(node) and (reachable point)-to-(not-yet-reachable node). Each edge has a cost estimate of traversing from a terminal (node or point) to another. We initialize G by setting $P_s = \{s\}$, $P_t = \{t\}$, $U = \{\}$, $V = \{\text{root}\}$, and connecting s to root and root to t with edges. The root denotes the entire C-space. 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 a shortest-path graph algorithm [1] on the subgraph of G induced by V and P, with source vertices P_s and sink vertices 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 the graph algorithm does produce a sequence with end points in P and intermediate nodes in V, then we enter the sequence verification stage. In this stage, we use $\mathcal{L}$ to determine the connectability of the end points through the sequence of nodes. 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 any point, say q, of v is reachable from p with $\mathcal{L}$, we would

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 $B(q) = p$, so that a path from s or t to q can be retraced.

Then, we would continue the verification stage by checking the connectability between p' and q. The verification stage ends when either the entire sequence is connected, or a node v is found unreachable from a point p. In the former, we would retrace a path from s to t through G to yield a motion for the robot. In the latter, however, we would 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 the graph algorithm produces no candidate sequence, then we would 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 deemed to have the least amount of refinement and refine it into more clear, defined subgoals. After refining v into children $C(v)$, 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 would push $A(B(v))$ into Q to ensure the eventual chance that every node of U gets refined.

3.1.2 Node Representation and Refinement

Although subgoal representation is independent of the search framework thus presented, it is nevertheless an important factor in determining the efficiency of the eventual search. For a manipulator with n dof, a node at (refinement) level k is a n-vector with only the first k components specified. Thus, the totally unspecified node v_0 at level 0 represents the entire C-space, and a fully specified node represents a single point. 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.

In the node refinement stage, nodes at the highest level are refined: A node v at level k is refined 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 values at a prescribed resolution. The resulting set of nodes with $k+1$ specified components and positive distance 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_c with the maximum distance value, and removing each node whose $(k+1)^{\text{th}}$ component is within λ number of nodes of v_c is repeated until every node is considered. The idea is to condense the set of possible nodes into a sparse collection of subgoals having maximal clearances based on its specified links. Refinement stops when all nodes are fully specified.

3.1.3 Local Planning

The local planner simulates robot movements in the C-space in small steps. A step is defined as a configuration change where a dof is changed by a preset amount, which represents the resolution. This preset amount of change is indicated by a number called *stride*. The stride in each dimension 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 possible for the robot to collide with the objects during a step, although it does not before and after the step. The Jacobian method [31] can be used to dynamically control the stride sizes and ensure no inter-step collisions, but we have decided to use small fixed strides for implementation efficiency.

The local planner $\mathcal{L}$ checks the reachability of a node v from a point p by moving the robot from p to any point q in v. The iterative procedure of moving from p toward v is as follows: First, we make progress toward v by considering all neighboring points of p that are one step closer to v than p is, and move to the point p' that has the maximum clearance $d(p') > 0$. (Depending on the node representation, the distance between point p and node v is measured by the Euclidean distance between p and the closest point q in v.) Next, we slide repeatedly by considering sequentially in each dimension, the two neighboring points one step away from either direction. If there is a point p'' closer to v than p is, while having a larger clearance $d(p'')$, we would move from p' to p'' and set p' to p''. If no progress can be made, then $\mathcal{L}$ would report a failure; otherwise, the move-toward-and-slide procedure is repeated until v is reached.

3.1.4 Experiments

The PTP planner is run on both easy and hard problems, and its performance is illustrated using the following numbers. The computation times are run times on a 200MHz SGI Indigo2 workstation. The number of nodes on the graph generated by the PTP planner and the number of local moves invoked by the global planner give ideas of the problem difficulty the planner's efficiency.

To get a meaningful and fair assessment of our performance, the following measure is developed. First, we use the number of collision detection (or distance computation) between the robot and its environment rather than computation time, since it is independent of geometric complexities of the robot and obstacles. Second, consider a trivial planner which discretizes the C-space into N_{grid} points, performs collision detection at each point, and finds a path among the collision-free points. This planner will run in $O(N_{\text{grid}})$ time. The ratio of the number of distance, N_{dist}, computed by the planner to the N_{grid} of the underlying grid in the C-space gives a good measure of the planner's efficiency. In Figure 1, $N_{\text{dist}}/N_{\text{grid}}$ for the examples that follow are plotted in log scale against dof, and this empirically shows the efficiency of our algorithm for high-dof problems. The apparent 'log-linearity' exhibited in the plot indicates an exponential time improvement of the PTP planner over the trivial planner, even though its performance may still be exponential in dof.

We have tested our planner with 2,4 and 9-dof planar manipulators, a 5-dof Adept, and 6-dof Puma manipulators in 3D. The number of dofs, computation times, number of nodes in the graph, number of local planner invoked, number of distance computations, and $N_{\text{dist}}/N_{\text{grid}}$ are shown in Table 1.

Figure 2 shows 3 problems of increasing difficulty with a 2-link planar robot. Solution motions and the subgoal graphs in the C-space are shown along with the C-space obstacles (Figure 2(d)). The size of the subgoal graph increases with the problem difficulty. The real power of the PTP planner shows in solving problems with higher dofs. Figure 3 shows 4-dof problems, which are solved in 8 and 14 seconds. Planning motion of a 5-dof AdeptOne robot to move an L shape out of a wicket to the left side of the table takes 22 seconds (Figure 4). Pulling a stick from behind two vertical posts with a 6-dof Puma robot is shown in Figure 5 (5 sec.). Finally, the 9-dof problem shown in Figure 6(c) takes about 15 minutes, showing the gradual increase of computation time with dof. In the examples above where obstacles and robots have about 10 to 40 faces, the time for one distance computation was 1 millisecond. For typical 5 and 6-dof problems of moderate difficulties (Figure 4 and 5), the PTP planner shows near real-time performances. If the geometric complexities of robots and environments increase, or for very difficult problems, the PTP planner will need a 10 to 100 times faster computer for a near real-time performance.

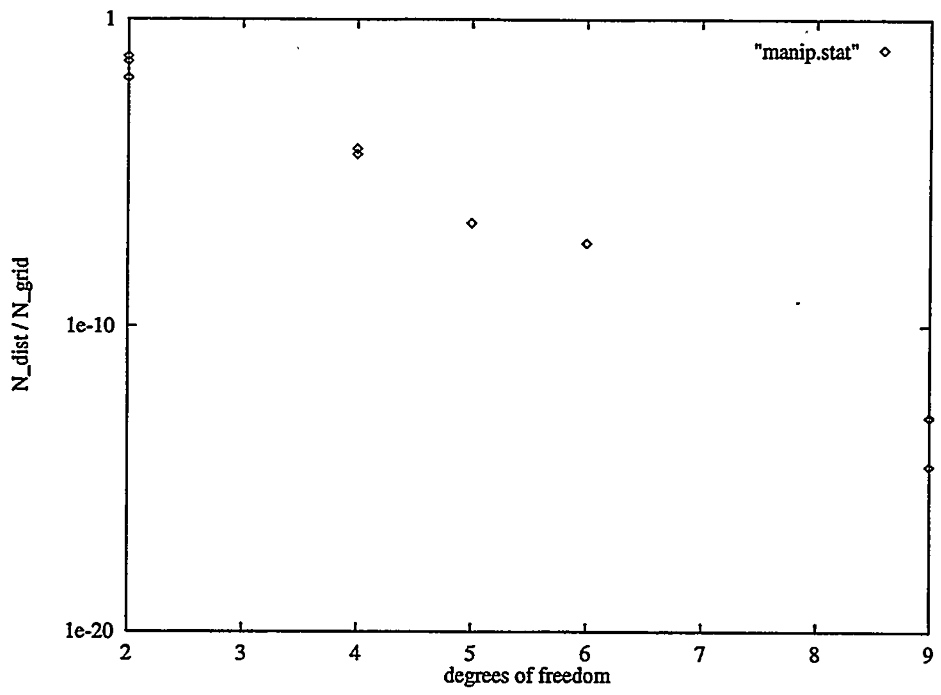
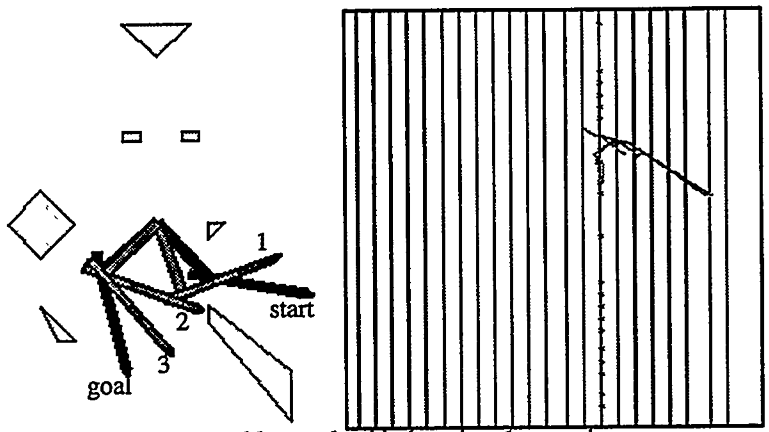


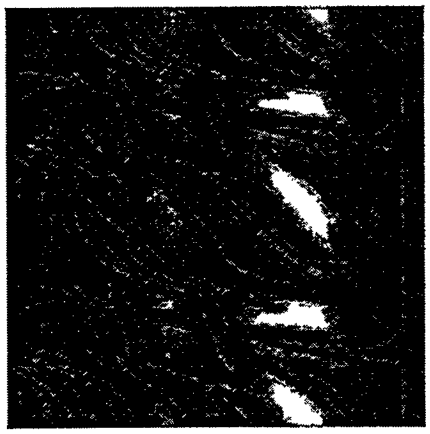
Figure 1: Ratio of the number of collision-check points to the total number of points in the C-space, plotted in log scale against the degrees of freedom of the robot(s).

example	dof	time(sec)	N_{node}	N_{local}	N_{dist}	$N_{\text{dist}}/N_{\text{grid}}$
2link, easy	2	1	45	15	1588	1.25e-2
2link, medium	2	2	235	41	5469	4.31e-2
2link, hard	2	3	323	97	7949	6.23e-2
4link, easy	4	8	266	98	10165	4.15e-5
4link, hard	4	14	271	164	14986	6.12e-5
AdeptOne	5	22	182	94	10079	2.37e-7
Puma260	6	5	46	35	2129	5.13e-8
9link, easy	9	10	17	9	7482	2.49e-15
9link, medium	9	544	1186	714	285931	9.53e-14
9link, hard	9	940	2114	1823	308292	1.03e-13

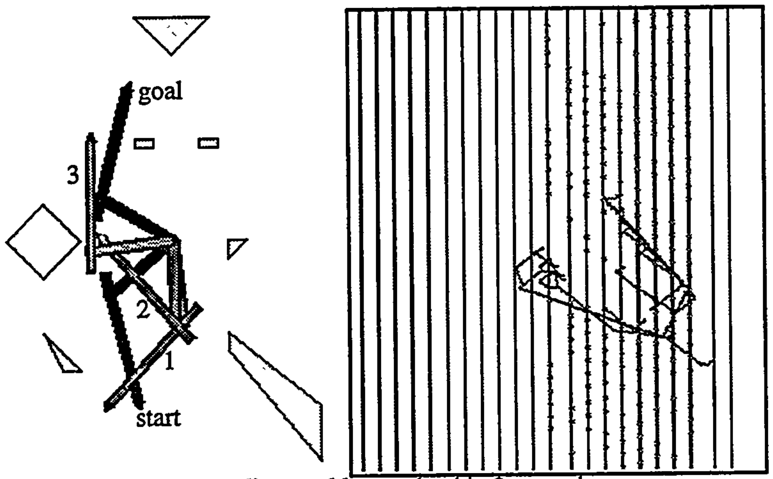
Table 1: Performance statistics of the PTP planner.



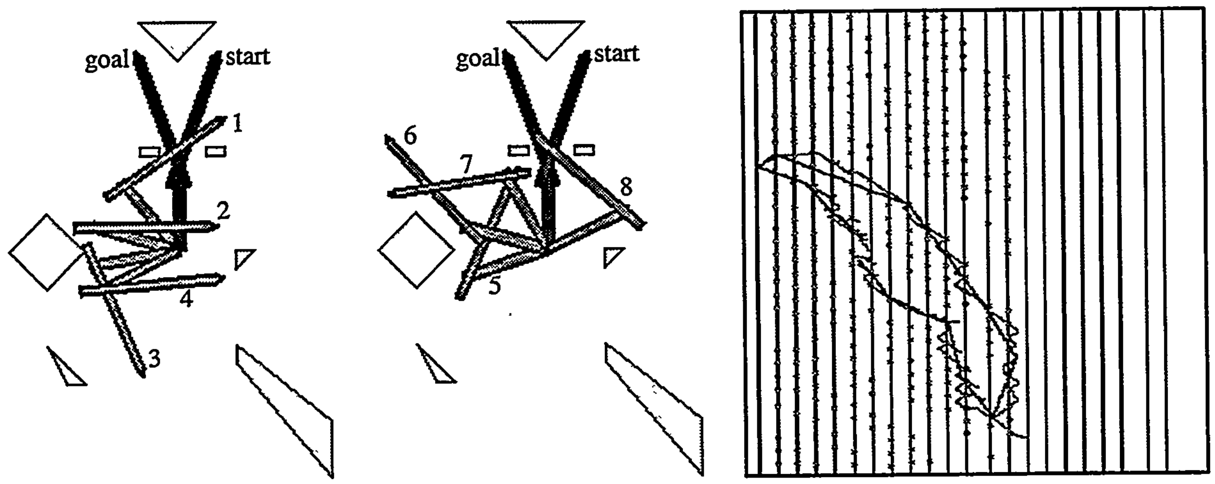
(a) Easy problem, solved in less than 1 second.



(d) C–space obstacles.



(b) Intermediate problem, solved in 2 seconds.



(c) Hard problem, solved in 3 seconds.

Figure 2. A planar 2–link manipulator among polygonal obstacles. Three problems with diferent levels of difficulty are solved in times proportional to the difficulty. The graphs of subgoals in the C–space are also shown. The vertical bars are 1–dimensional subgoals, and the small squares denote 0–dimensional subgoals. The graph sizes reflect the problem difficulties. The curves represent the traces of the local planner, and the dark straight lines are post–processed solution paths. The upper right figure shows the C–space obstacles (uniformly shaded regions) and the varying darkness between the obstacles shows the distance between the robot and obstacles.

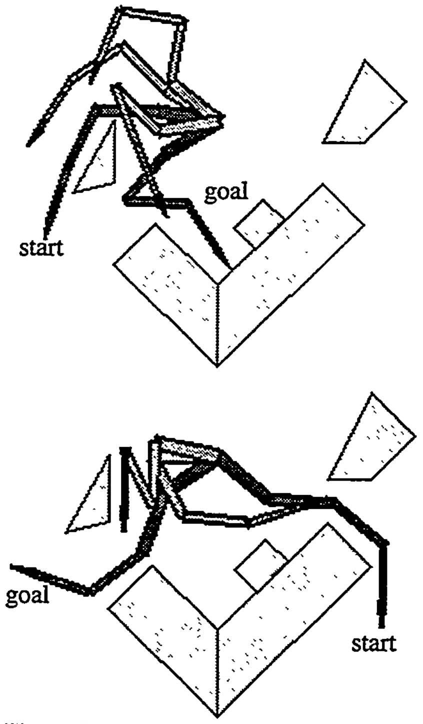


Figure 3. Two 4–dof problems solved in 8 (top) and 14 (bottom) seconds.

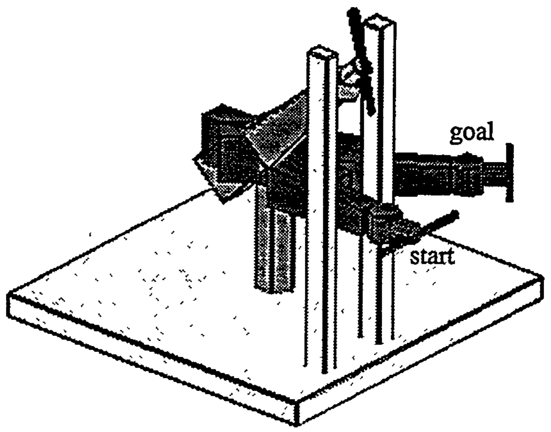


Figure 5. It took 5 seconds to find the motion to move a stick through the two vertical posts.

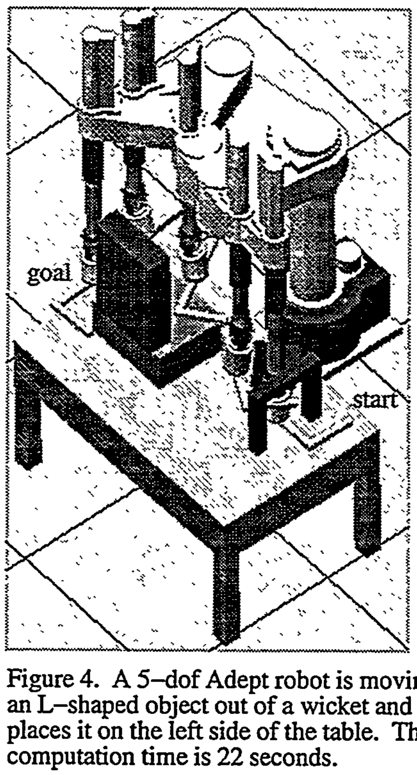


Figure 4. A 5–dof Adept robot is moving an L–shaped object out of a wicket and places it on the left side of the table. The computation time is 22 seconds.

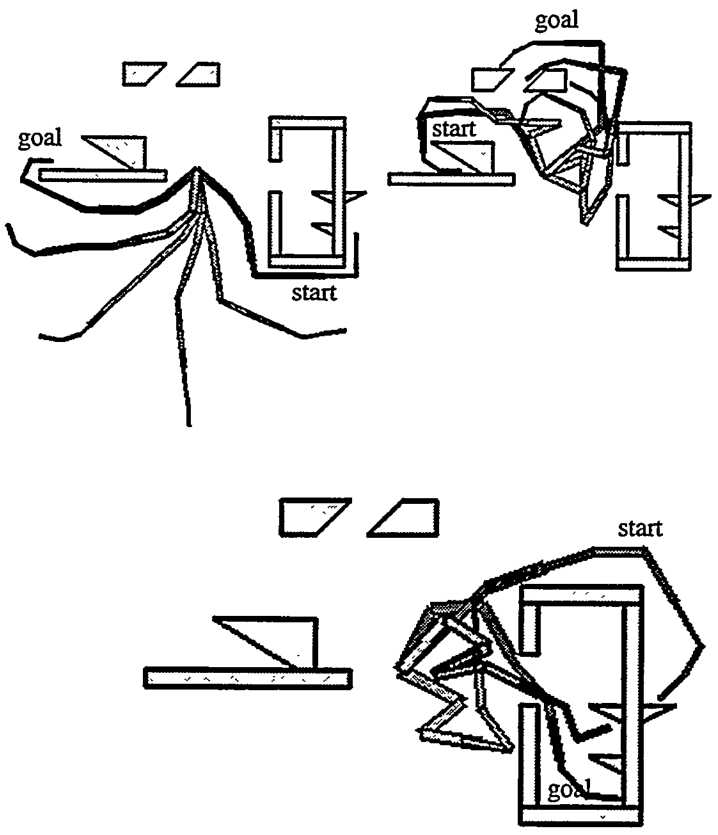


Figure 6. Easy, intermediate and hard problems involving a 9–link planar robot. It took 10 seconds (top left), 10 minutes (top right) and 16 minutes (bottom) to solve these problems, respectively.

3.2 Trace-Single-Curve Planner

Our TSC planner is basically a structured search algorithm that examines the solution space without building an explicit representation of the feasible motion set. Note that building an explicit representation is computationally expensive for robots with high degrees of freedom. Our planner works roughly as follows. Given a curve to be traced by the robot tool tip, we first identify points on the curve that are in cluttered space. These points are called ***critical locations***, and include both the starting point and ending point of the curve by definition. We then find at each critical location a set of inverse kinematic solutions that do not cause collisions between robot links and objects. We call these inverse kinematic solutions ***subgoals***. Next, a sequence of subgoals, one for each critical location, from the starting to the ending point is selected as a candidate path. Finally, a local planner is used to verify the existence of a collision-free joint motion from one subgoal to the next in the sequence, until the goal is reached. Because inverse kinematic solutions are computed only for critical points, we gain efficiency and thus can handle more redundancies. The way our algorithm handles the case requiring reconfiguration of the manipulator in the midst of tracing the curve is explained below. We divide our planner into a global planner and a local planner; the global planner keeps generating a candidate sequence of subgoals, while the local planner finds actual joint motions connecting subgoals. The global and local planners are completely separate of each other, and can be modified independently. For example, one of the local planners cited in Section 2 can be used in our planner. We now describe the global and the local planner in detail.

3.2.1 Global Planner

Given a curve, the global planner is responsible for generating a set of critical locations, finding subgoals for each critical location, and generating candidate sequences of subgoals that will be examined by the local planner.

Setting up critical locations on the curve

Given one piece c of the curve C, and the length r of robot tool tip, we first build a cylinder of radius r whose axis coincides with c. We then compute the intersection I of the cylinder with each object O in the workspace using ACIS routines. Next, we project I back onto the curve c to obtain line segments L_i, which denote the portions of c on which the robot needs to cleverly maneuver itself to avoid object O. Finally, we construct the set of critical locations from the endpoints of the L_i's using the following filter. When two critical locations are closer than a preset threshold, we delete the one that is farther from the starting point of curve c. (This step reduces computational complexity without degrading solution quality.) Figure 7 shows an example of critical locations.

Computing subgoals

At each critical location l_j, we solve for inverse kinematic solutions that are collision free. For redundant robots, there are usually an infinite number of solutions and computing all of them is itself a research problem. A brute-force method is used in [36] to compute all inverse kinematic solutions for each critical location. This method discretizes the redundant degrees of freedom with a grid and solves $\Delta x = J\Delta q$ for Δq with additional equations setting the redundant degrees of freedom equal to the joint values at each grid point. This method is, however, exponential in the number of redundant degrees of freedom, and gives us unnecessarily many solutions. Ideally for our algorithm, we would like to get one solution from each *aspect* [36]. An aspect of a manipulator is a connected region of the joint space in which the manipulator Jacobian remains full rank. (This means, roughly, that we want a small number of samples uniformly distributed over the set of inverse kinematic solutions.) When obstacles are present, one aspect might be divided into several regions by the configuration space obstacles, requiring us to find a solution for each connected region of each aspect. Since computing aspects is not the main focus of this paper, we leave this for future work and use the following heuristic approach.

Given the position of the tool tip, we first find a set of collision-free orientations of the tool tip. This specifies a set of tool tip configurations. We then find a set of joint angles that achieve each of the tool tip configurations as follows. We define a set of initial manipulator configurations uniformly distributed over the joint space. From these initial configurations we make the robot converge to a configuration that achieves a given tool tip configuration. We use the local planner in Section 3.2 for the converging movement except we do not include the collision avoidance. If we incorporate the collision avoidance in this step, the robot tends to stay away from the objects, and we may not compute a collision-free robot configuration that places the robot in a tight space. Such a configuration may be essential in generating a global collision-free motion tracing the curve.

The resulting manipulator configuration, therefore, may involve collisions with objects. If so, we move the manipulator to a nearby collision-free configuration using a greedy search algorithm that minimizes the amount of overlap between the manipulator and the objects. We restrict the movement in the null space with respect to the position of the tool tip so that the tool tip stays at the corresponding critical location. The amount of overlap between two objects is measured by the minimum distance one of the objects has to translate in order to separate them. This measure has also been called the *negative distance* [5]. From the current configuration, the greedy search algorithm moves the robot to one of the adjacent configurations with a smaller overlap. The search is continued until the current configuration has the minimum overlap, or the robot is in a collision-free configuration. We then select only those configurations that are collision-free as subgoals.

We use the following scheme to define a set of initial robot configurations. We divide the range of each joint into two equal intervals, and use the center value of each of the intervals as a possible joint value. The scheme is equivalent to representing the joint space with a one-level deep 2^{dof}-tree and defining the center of each cell as one of the initial configurations. The selection process is roughly equal to getting one initial configuration from each of the aspects defined in [36], and results in an approximately uniform sampling in the joint space.

Finding the shortest sequence

Once we have computed the subgoals, i.e., inverse kinematic solutions, q_{jk} for each critical location l_j, we construct a graph G whose nodes are q_{jk}. The edges of G are between two subgoals q_{jk} and $q_{(j+1)k'}$ in the adjacent critical locations whose distance in the joint space is less than a preset number Λ times the distance between the adjacent critical locations l_j and l_{j+1} in the operational (world) space. The heuristic is that the joint angle should not change much when the tool tip is tracing a small segment of the curve. The edge cost is set to the Euclidean distance between the subgoals in the joint space. We then use dynamic programming to find the shortest sequence from any of the subgoals at the starting location to any of the subgoals at the ending location of the curve. The sequence with the smallest total edge cost is selected as the candidate sequence, and the existence of a collision-free path via the subgoals in this sequence is verified by the local planner. As the local planner finds a collision-free motion between two subgoals, the corresponding edge cost is replaced by the actual length of the collision-free motion in the joint space. If the local planner cannot find a collision-free path between two subgoals, the edge connecting them is deleted from the graph. Since the length of the collision-free motion is always greater than or equal to the straight line distance between two subgoals, our graph search will examine all sequences that can potentially result in a shorter path than the current solution path. This process of selecting and verifying sequences is repeated until there is no sequence with a smaller estimated cost than the actual cost of the shortest path found so far. Figure 8 shows a graph and a candidate sequence of subgoals.

Reconfiguration of manipulator

Let s and t be the starting and ending location of the curve. Given a graph G of subgoals, define *s-reachable* (*s-unreachable*) subgoals to be those that can (cannot) be reached along G using the local planner from any of the subgoals at the first critical location. Define a critical location to be s-reachable if one of its subgoals is s-reachable; otherwise, s-unreachable. Similarly, define the corresponding terms for final location t. It may be the case that at a particular critical location, some subgoals are s-reachable, some t-reachable, but none are both s-reachable and t-reachable. In such a case, the curve cannot be traced completely without taking the tool tip off the curve, i.e., the manipulator has to be reconfigured in the midst of tracing. Our algorithm reconfigures the manipulator as follows.

First, start from the subgoals at s, and trace the curve as far as we can (called *forward planning*), by continually generating a candidate sequence and verifying it with the local planner. If we can reach any of the subgoals at t, then we have succeeded in tracing the whole curve. Otherwise, there exists a critical location v that is not s-reachable, but its predecessor u is. In this case, insert a new critical location w halfway between u and v, by computing the subgoals at w and updating G accordingly. We repeat the process of forward planning and inserting a new critical location until the distance between u and v is smaller than a preset distance D_{min}. At this time, we reconfigure the manipulator at u. We use the PTP motion planner to move the manipulator from an s-reachable subgoal u_0 to an s-unreachable subgoal u_1. The PTP motion planner almost always moves the tool tip off the curve, and the generated motion corresponds to a reconfiguration motion. If it succeeds, then we add the edge (u_0, u_1) into G and continue with forward planning. Otherwise, the planner fails as v is declared s-unreachable via G with reconfiguration.

Notice that our algorithm always reconfigures at the last s-reachable critical location u, even though other critical locations may be possible. Let T_{ab} (T'_{ab}) be a curve-tracing motion from critical location a to b (b to a). Let R_a be a reconfiguration motion between two subgoals in critical location a. We now show that if there is a curve-tracing motion that includes a reconfiguration, then there is a curve-tracing motion with a reconfiguration motion at the last s-reachable critical location, given that the following two conditions are satisfied. First, the PTP motion planner is complete, i.e., finds a solution if there is one. Second, for a collision-free, curve-tracing motion $T_{a_i b_j}$ between subgoal a_i at critical location a and subgoal b_j at critical location b, there must be a collision-free motion $T^*_{a_i b_j}$ between a_i and b_j along which the tool tip does not touch the curve. The $T^*_{a_i b_j}$ is easily obtained by perturbing $T_{a_i b_j}$ by a small amount to move the tool tip away from the curve.

Suppose that there is a curve-tracing motion, $T_{sw} + R_w + T_{wt}$, which includes a reconfiguration motion R_w (Figure 9). Suppose that u is the last s-reachable critical location that is farther from s than w. Then there exists a reconfiguration motion at u, namely, $T'^*_{wu} + R_w + T^*_{wu}$, which will be found by a *complete* PTP motion planner. Thus, the choice of the critical location for reconfiguration does not affect the completeness of our algorithm.

3.2.2 Local Planner

The local planner moves the robot from one subgoal to the next while tracing the curve and keeping an optimal orientation of the tool tip. Optimal orientations are specified by the robotic task at hand. For example, a caulking operation might require the tip of the caulk to maintain 45 degrees from the edge. The tool tip orientation is compromised only to avoid collision between the robot and the objects in the workspace. We can also compromise the tool tip orientation to avoid kinematic singularities, but this is currently not implemented. The local planner never moves the tool tip off the curve in any case, since it severely degrades the quality of robot performance in most tasks.

Our local planner is a modified version of the algorithm in [27]. We first solve $\Delta x = J(q)\Delta q$ to move the tip along the curve using singular value decomposition. We then use a null-space movement to change the tool tip orientation as close to the optimal value as possible as long as the distance between the robot links and the objects are greater than a preset threshold D_{danger}. If the distance is smaller than D_{danger}, we use the null space movement to increase the distance. The null space movement is achieved by moving the robot joint angles along the basis vectors of the null space of $J(q)$, which are computed from the singular value decomposition. We also limit the number of the null space movement so as not to exceed joint velocity limits. If the robot collides with an object or the tool tip orientation goes out of the acceptable range, the local planner declares that there is no feasible motion between the two subgoals. Figure 10 illustrates the local planner.

3.2.3 Completeness and Efficiency

Our algorithm gains computational efficiency by computing inverse kinematic solutions at several critical locations rather than at all points on the curve. We also compute only a small number of inverse kinematic solutions at each critical location to gain further efficiency. If we had computed the set of all inverse kinematic solutions at every point along the curve and searched for an optimal motion in that set, our algorithm would be complete. The resulting computation time, however, would be too long for practical applications. Instead, our algorithm relies on the local planner to find motions between subgoals at the adjacent critical locations. It is difficult to analyze the complexity of our overall planner precisely, but it does have the following characteristics. If the local planner is a sophisticated algorithm, the critical locations can be far apart and the global planner does less work. If the local planner is a simple algorithm such as *moving straight in the joint space*, the global planner has to do more work by computing more subgoals at more critical locations. In the global planner, the initial path lengths of the edges between subgoals are the straight-line distances in the joint space, and thus are under-estimates. This satisfies the *admissibility* condition of A* search, and the global planner is guaranteed to find the shortest path in the graph G of the subgoals. The optimal path in G is close to the optimal solution to the problem as demonstrated in the next section. Moreover, we can always further optimize the optimal path in G around its neighborhood using a numerical technique [31].

3.2.4 Experiments

We have tested our algorithm with a 4-dof planar manipulator, and the planned motion is shown in Figure 11. We have chosen the examples so that a reconfiguration motion is needed in tracing a curve. It took less than 2.5 minutes to solve the problem on a 200-MIPS workstation. Our algorithm computes the distance between robot links and objects on the order of 10^{3-5} times, and this has necessitated the use of the fast distance routine in [15].

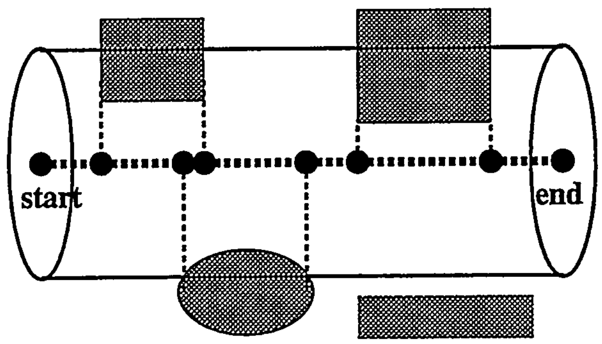


Figure 7. Critical locations shown in black circles are the beginning and end of the intervals in the curve where objects are close by. The fourth critical location will be deleted as it is close to the third critical location.

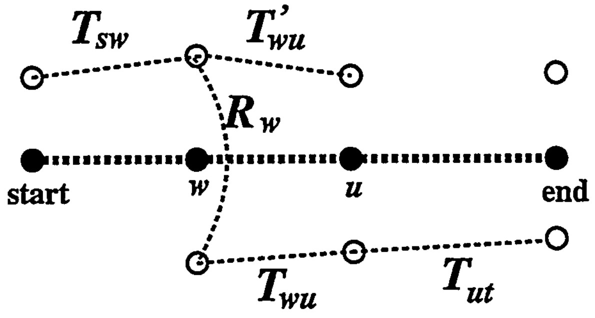


Figure 9. If there is a curve–tracing motion which includes a reconfiguration motion, the reconfiguration motion can always be done at the last critical location u reachable from the start of the curve.

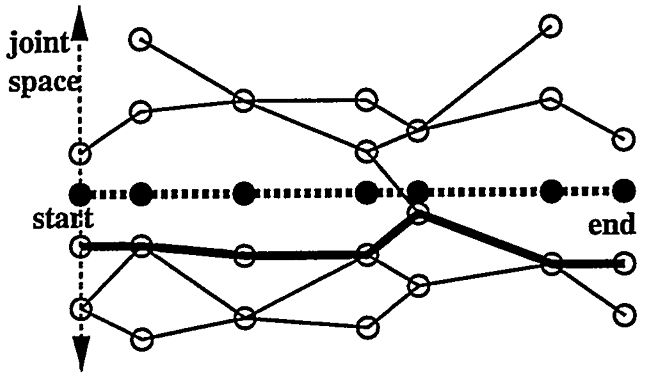


Figure 8. Subgoals shown with white circles at the critical locations represent inverse kinematic solutions. A graph is constructed by connecting close subgoals at the adjacent critical locations. The thick line shows the shortest sequence of subgoals from the start to the end of the curve.

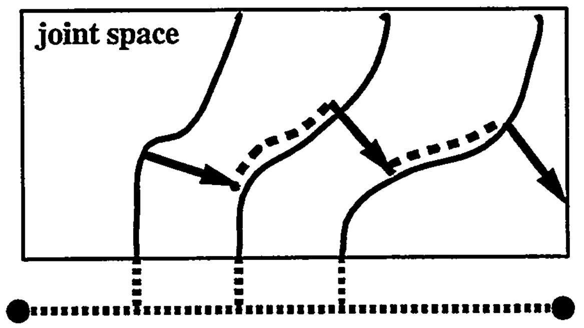


Figure 10. Each curve in the joint space denotes all inverse kinematic solutions that place the tool tip at the corresponding point on the curve. The local planner first moves the tool tip along the curve (arrows), and then uses the null space movement to avoid collisions and optimize the tool tip orientation (dotted lines).

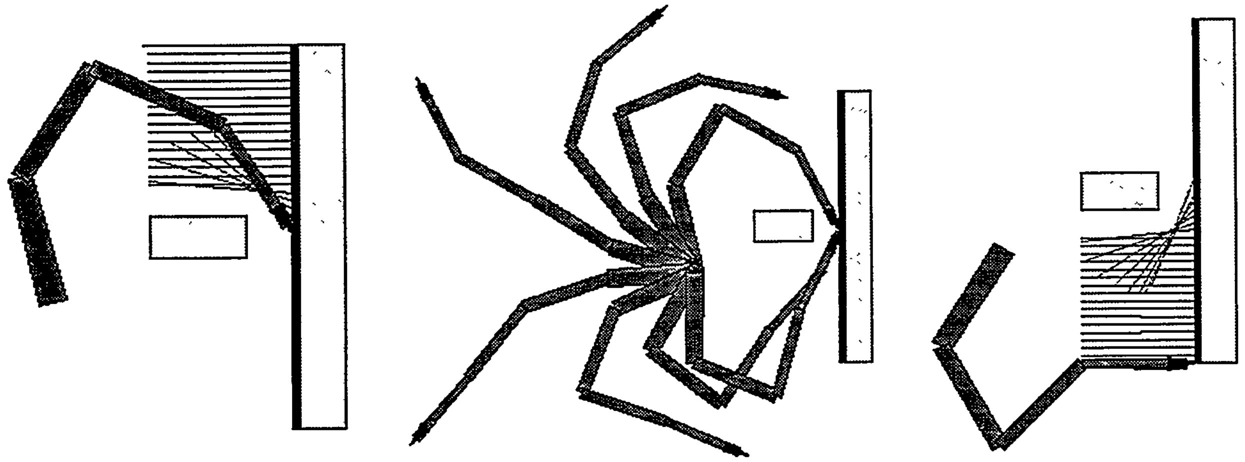


Figure 11. A planar 4–link robot needs a reconfiguration to trace the bold edge. The tool tip orientation deviates from the edge normal only if necessary to avoid collisions.

4 Motion Planners for Robotic Tasks

The visit-point (VP) planner and the trace-curve (TC) planner are generalizations of the PTP planner and the TSC planner. To get a path that visits multiple points or tracing multiple curves, the VP and TC planners employ an algorithm for the traveling salesman problem (TSP) to find the optimal order of the points or the curves. The TSP algorithm in the robot joint space is more complicated than the traditional TSP algorithms, and the differences are describes in Section 4.1. The cover-surface (CS) planner partitions the surfaces into a set of stripes, and the TC planner is used to trace mid-lines of the stripes in the optimal sequence.

4.1 Visit-Point Task

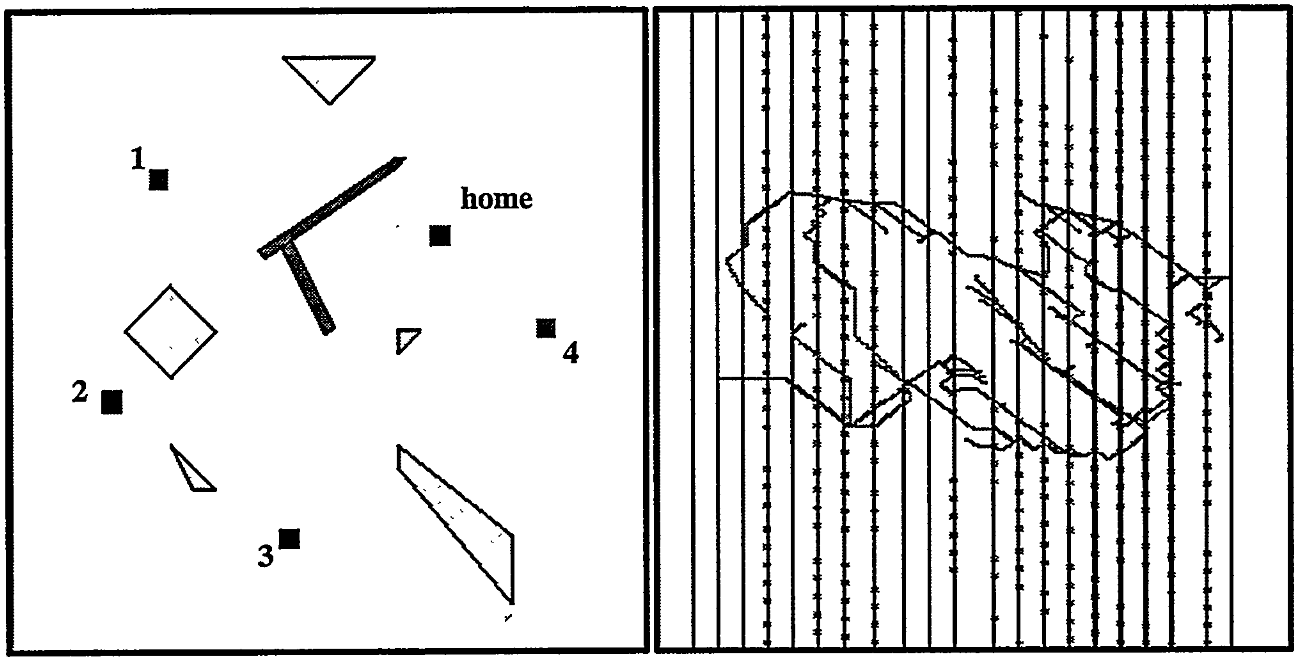
The visit-point task for robot manipulators are different from the traditional traveling salesman problem in two respects. First, many robots have more than one way to reach a point in the world space. In fact, for a redundant manipulator, there are an infinite number of solutions to the inverse kinematics. The visit-point task requires that a point be visited using any one of the inverse kinematics solutions. If there are n ways to reach each of the w points, we would have to solve n^w regular TSP problems to get the shortest path. Second, the actual path length between two points are not known until a collision-free path in the configuration space is computed using the PTP planner. Although computing the actual path lengths between all pairs of points would yield the shortest path, the required computation resource would be prohibitive for many practical problems.

The following heuristic algorithm is developed to get an approximately shortest path for the visit-point problem, based on the nearest insertion algorithm for the regular TSP problem [24]. Let say the robot is to visit w points in the world space. A group of points in the robot's configuration space that place the robot tool tip at each of the world points is computed. If there are an infinite number of configuration points as for a redundant manipulator, a finite number of them are sampled. The optimal robot motion is then the shortest tour in the configuration space that includes one configuration point from each group.

The VP planner has two stages: tour generation and cost update. In the tour generation stage, we assume the distance between two configuration points are the straight-line Euclidean distance between them. To generate a tour, we pick a point in the configuration space that has not been included in the tours previously generated. Call this the seed point. We then select the closest configuration point to the seed point that has a different corresponding world point from that of the seed point. We keep on inserting the closest configuration point to the current tour whose world point is not yet visited by the current tour. When inserting, we find the best place to insert to minimize the tour length. We repeat this process until every configuration point is included in one of the tours. Although the straight-line distances of these tours are correlated to the actual path lengths, they can be quite different when the robot has to go around an obstacle. At the cost-update stage, the PTP planner is used to compute the collision-free motion between pairs of points adjacent in all of the tours. After replacing the straight-line distances with the actual lengths of the collision-free motions, the tour with the shortest actual length is selected as the best path. A more expensive, sophisticated TSP algorithms are possible with more computing resources.

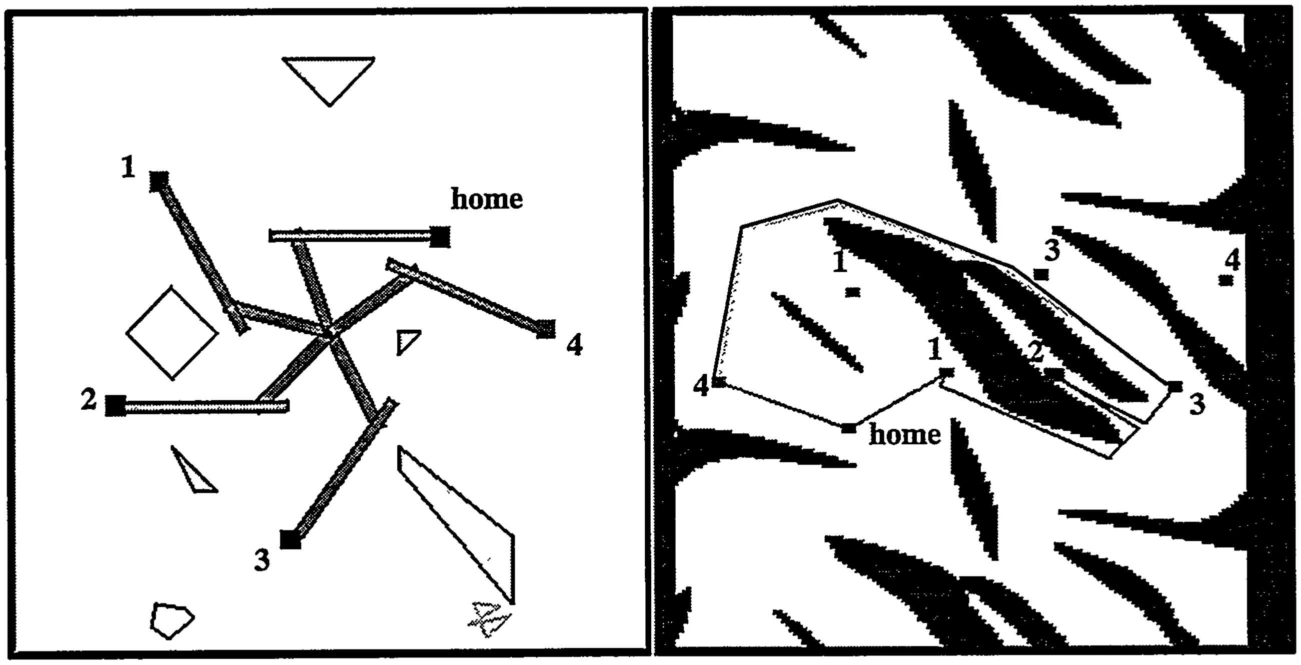
Experiments

Figure 12(a) shows a 2-link robot, polygonal obstacles and 5 squares to be visited by the robot's tip. Figure 12(b) shows the search tree explored by the PTP planner. Notice that the VP planner does not build the complete description of the configuration obstacles shown in Figure 12(d), i.e., the set of all joint angles that cause collisions between the robot and the obstacles. In Figure 12(c), the robot can reach Point 1, 3 and 4 in 2 configurations, one elbow up, one elbow down. The home position has only one configuration by definition. Only those configurations that are used by the shortest tour are shown. The shortest tour found by the VP planner is shown in the configuration space in Figure 12(d), where the shaded regions represent the configuration obstacles. In this example the VP planner is called 10 times and it takes 20 seconds of computation time on a modern workstation.



(a) World space

(b) Search in the configuration space



(c) Solution tour in the world space

(d) Solution tour in the configuration space

Figure 12. A planar 2–link robot is to visit Point 1 through 4 starting from the home position (a). The VP planner searches for collision–free paths in the configuration space (b), without building the configuration obstacles shown in (d) explicitly. The solution tour found by the VP planner is shown in (c) and (d).

4.2 Trace-Curve Task

The trace-curve (TC) planner is similar to the VP planner, except it uses the TSC planner to trace along each curves and the PTP planner to move between curves. In general, there are several robot motions that trace the same curve, and a finite number of curve-tracing motions are computed by the TSC planner. While tracing a curve, the TSC planner keeps the tool tip orientation at a specified optimal value, and allows deviations only to avoid collisions or singularities. We will impose the restriction that once the robot starts to trace a curve, it has to finish tracing the curve before it starts on another. This is a practical assumption, since many tasks such as caulking and metal cutting require the robot to finish the curve it has started to trace. In the tour-generation stage the TC planner uses the straight-line distance between the end points of the motions tracing the curves. The nearest insertion method is also used to generate these tours, and at least one of the tours contains each robot motion tracing each curve. In the cost-update stage, the tour lengths are converted to actual lengths of the collision-free paths using the PTP planner, and the minimum-length path is selected as the best path.

Experiments

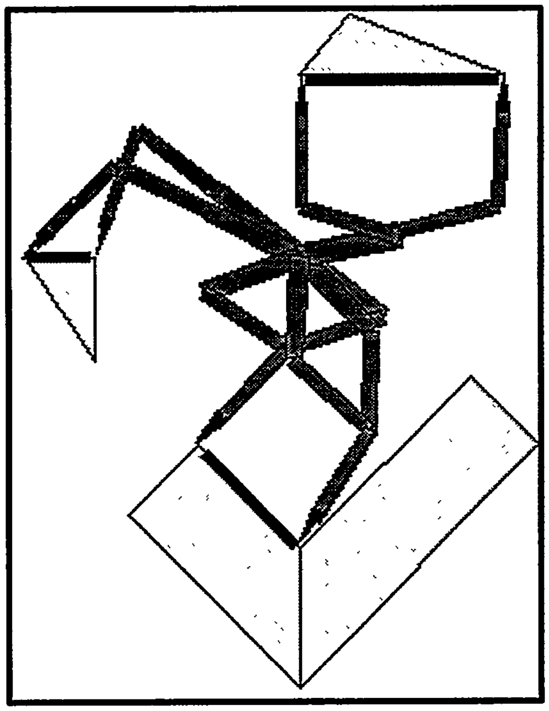
Figure 13 shows a 3-link robot tracing 3 edges drawn in bold lines. The robot configurations that place the robot's tool tip at the start and end points of the curves are shown in Figure 13(a). The robot tries to keep the tool orientation as close to the edge normals as possible. The motions tracing 3 edges are shown in Figure 13(b); the motions from one edge to another are not shown to avoid cluttering.

4.3 Cover-Surface Task

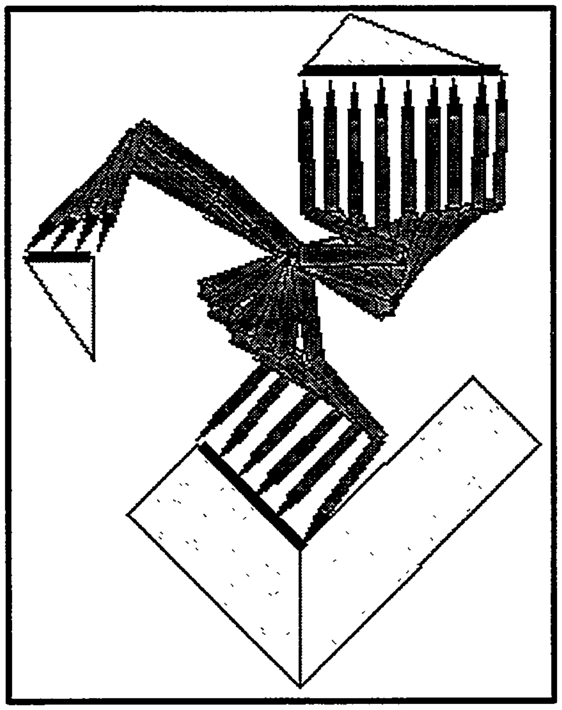
This section describes the cover-surface (CS) planner specialized for painting applications. The CS planner partitions the surfaces into a set of stripes whose widths are equal to the effective width of the spray cone. The stripe direction is chosen along the longest edge of a surface by default (this tends to minimize the number of the stripes), but other criteria could be used. Once the partitioning is done, the TC planner can be used to compute the motion of the robot holding a spray gun to trace the center lines of all the stripes. Since there are usually a hundred or more stripes in a typical painting application, a straight forward use of the TC planner would take a long time. We use an important piece of painting knowledge to improve painting quality and to reduce the computation time. When painting, an adjacent stripe is sprayed right after the currently sprayed stripe while the paint is still wet so that good bonding occurs at the stripe boundary. Using this knowledge, the CS planner finishes covering a surface once it gets started on it. This also tends to minimize the wasted motions for traveling between stripes. Rather than applying the TC planner directly to the stripes, we first apply the TC planner on the stripes of each surface to get an optimal ordering of painting the stripes. We then connect with a straight line the starting and the finishing points of painting the surface to get a *super-curve* (see Figure 14). We then apply the TC planner to the super-curves to get the ordering of the surfaces. Once the surface ordering is determined, the TSC planner is used to find a motion to spray each stripe and the PTP planner to find a collision-free motion to move the robot from one stripe to the next.

Experiments

Figure 15 shows the performance of the CS planner on the task of painting a wall with two boxes attached on it. The surfaces are partitioned into stripes, which are in the direction of the longest edges of the surfaces. For this problem, it took on a workstation 30 seconds for partitioning and TSP algorithms, and 10 minutes for computing the motions tracing the center lines of the stripes.



(a) Robot configurations at the start and end points of the curves.



(b) Robot motions tracing the three curves. Motions between the curves are not shown.

Figure 13. The robot motions tracing the three curves drawn in bold lines.

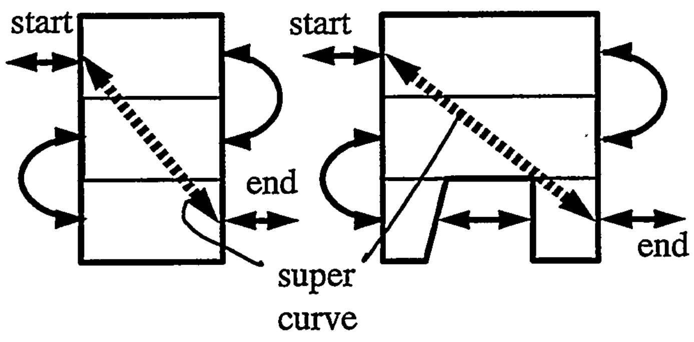


Figure 14. The stripes of a surface are painted together. The TC planner determines the order of the stripes of each surfaces, and forms a *super–curve* connecting the entry and exit point on the surface.

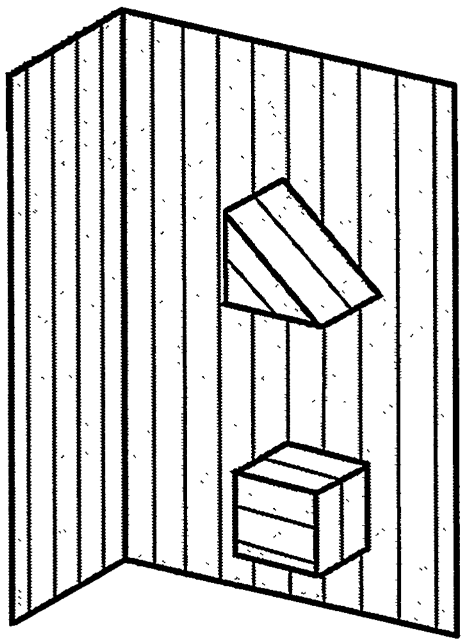


Figure 15. The surfaces of a wall and two boxes are partitioned into stripes in the direction of the longest edges.

4.4 Build-Volume Task

The build-volume planner can be developed by either 1) partitioning volumes into a set of surfaces and using the CS planner, or 2) partitioning volumes into a set of rod shapes and using TSC planner. It is, however, left for future work.

4.5 Robot Base Placement

The robot placement (RP) algorithm is based on the VP planner in Section 4.1. The RP algorithm places the robot base in all possible configurations given by the user, and runs the VP algorithm to get the optimal motion visiting the points in any order. The robot base configuration that results in a path with the smallest length in the joint space is selected as the optimum. The RP planner provides an algorithmic way to shorten the robot cycle time with respect to the robot base configuration.

The user first specifies the degrees of freedom of the robot base, and the associated range for each dof. The dof is usually 3 (x, y, θ) or 4 (x, y, z, θ). Assuming we discretize each dof with 10 points, there are 10^3 or 10^4 possible robot base configurations. The 10-point discretization is not coarse, since the user usually has a rough idea of the optimal robot-base location.

A grid is formed over the space of possible robot-base configurations. The robot is placed at each of the grid points, and inverse kinematic solutions are computed for each of the task points. The base configurations with at least one inverse kinematic solution for each of the task points will be labeled as feasible. For each feasible base configuration, the shortest collision-free path that visits all the task points is computed using the VP planner. Finally, the robot base configuration with the shortest path length is selected as the optimum.

Experiments

We have used our algorithm for a robot engaged in a simulated spot welding task. A CRS-465 robot in Figure 16 is to perform 4 spot welds on a car body at the points marked with circles. Two filled circles are to be welded from under the roof, and the two empty circles are to be welded from above the roof. The robot is place at a nominal position and its joint angles are set to the home position as shown in Figure 16. A simplified L-shaped weld gun is used to avoid clutter in showing motions. When computing collision-free paths, we used a simplied model of the robot shown in Figure 17 to speed up the computation of distances between the robot and obstacles. We note here that fast distance routines for complex objects have been developed in [7, 37], and will be integrated in our algorithm in the near future. The CRS-465 robot can have up to 8 inverse kinematic solutions for a given tool configuration. We used only half of the solutions since the other half amounts to rotating the robot shoulder 180 degrees and swinging the first and second links of the arm to the other side. (Such motions are time consuming and are not used in practice.) Because the spot weld process is symmetric with respect to the roll about the axis of the welding-gun tip, we generated inverse kinematic solutions at a 15 degree roll interval. The number of collision-free inverse kinematic solutions for each spot-weld point ranges from 2 to 8. The robot base is moved in (x, y, θ) space on a grid of 5x7x5 points. The computation time to examine 175 robot base locations was 60 hours on an SGI Indigo 2 workstation, with codes that include graphics. The time should be reduced at least by a factor of 3 when running without the graphics. The shortest path length corresponding to each base configuration is shown in Table 2. The optimal base configuration has an offset of $(0.0, -10.0, -5.0)$ from the nominal base configuration in Figure 16. Figures 18(a)-(d) show the robot base configuration minimizing the length of the spot-welding path, the inverse kinematic solutions used at the weld points, and the order of spot welding. A super computer or a network of workstations will be needed to get a similar run time for a realistic problem containing complex objects and many more weld points.

Table 2: Tour length verses Robot base configuration. The '-' denotes that one of the weld points is not reachable. The 'f' denotes that the point-to-point planner cannot find a collision-free path either because there is no solution or because it could not find one in 100 second time limit. The minimum path is achieved at the base configuration of (0,-10,-5).

	x	-10	-5	0	+5	+10
y	θ					
-20	-10	-	-	-	-	-
	-5	-	-	-	-	-
	0	-	-	-	-	-
	+5	-	-	-	-	-
	+10	-	-	-	-	-
-15	-10	-	-	-	-	-
	-5	-	-	-	-	-
	0	-	-	-	-	-
	+5	-	-	-	-	-
	+10	-	-	-	-	-
-10	-10	21.20	-	-	-	-
	-5	20.04	20.02	20.00	20.29	22.13
	0	f	20.32	20.28	f	f
	+5	20.45	20.40	20.34	f	f
	+10	20.62	21.60	21.05	21.04	22.01
-5	-10	21.16	f	f	f	f
	-5	20.08	20.05	20.78	20.38	21.55
	0	f	f	f	f	26.71
	+5	21.21	f	f	f	f
	+10	20.67	22.14	21.22	20.84	21.50
0	-10	31.06	f	f	f	f
	-5	20.85	21.39	20.88	20.43	21.61
	0	f	f	f	f	f
	+5	21.35	f	f	f	f
	+10	20.87	20.65	20.96	20.79	21.50
+5	-10	-	f	f	f	f
	-5	-	20.98	20.69	20.52	21.37
	0	-	21.94	f	f	f
	+5	-	f	f	f	f
	+10	-	f	20.54	20.36	21.81
+10	-10	f	f	f	f	f
	-5	28.57	29.36	30.15	32.39	32.34
	0	22.17	f	21.72	f	f
	+5	f	f	f	f	f
	+10	20.90	f	20.59	21.94	21.87

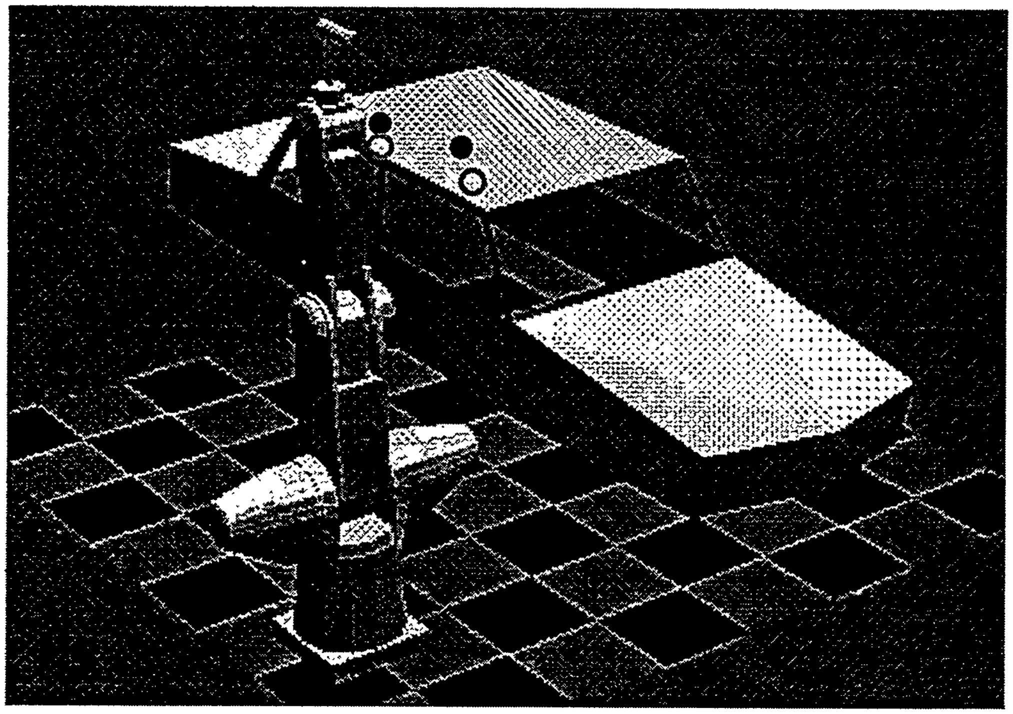


Figure 16. Planning a CRS–465 robot to spot weld a car body. The weld points are shown as circles on the roof. The filled circles are to be welded from under the roof, while the empty circles are to be welded from above the roof.

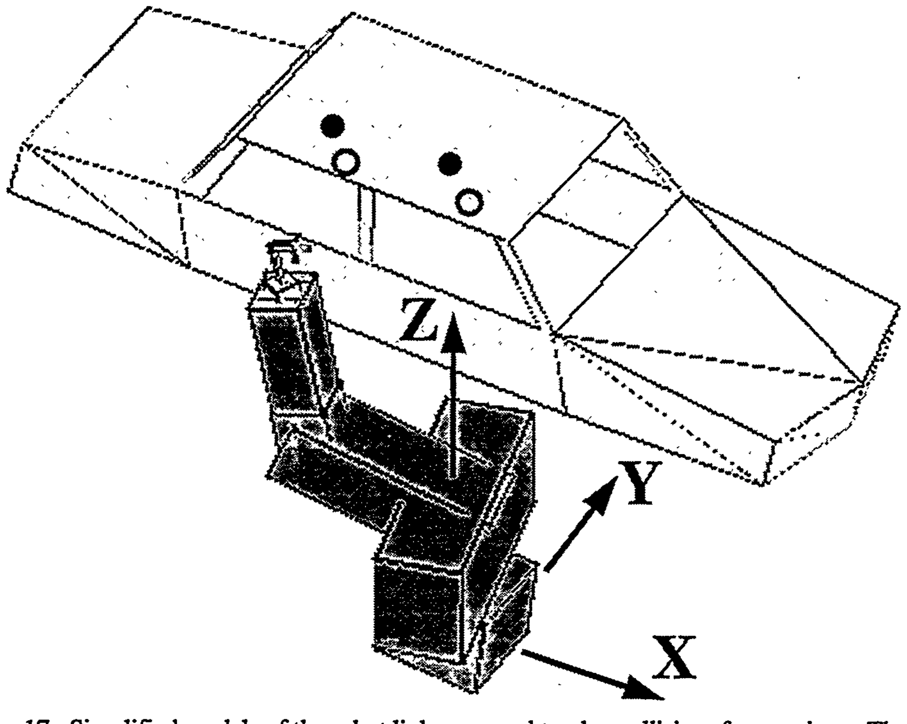


Figure 17. Simplified models of the robot links are used to plan collision–free motions. The robot base is varied in x, y, θ to find the optimal base configuration for the spot welding task. The ranges are ([–10, 10], [–20, 10], [–10, 10]), and the step sizes are (5,5,5), respectively. For reference, the forearm of the robot is 330 units long.

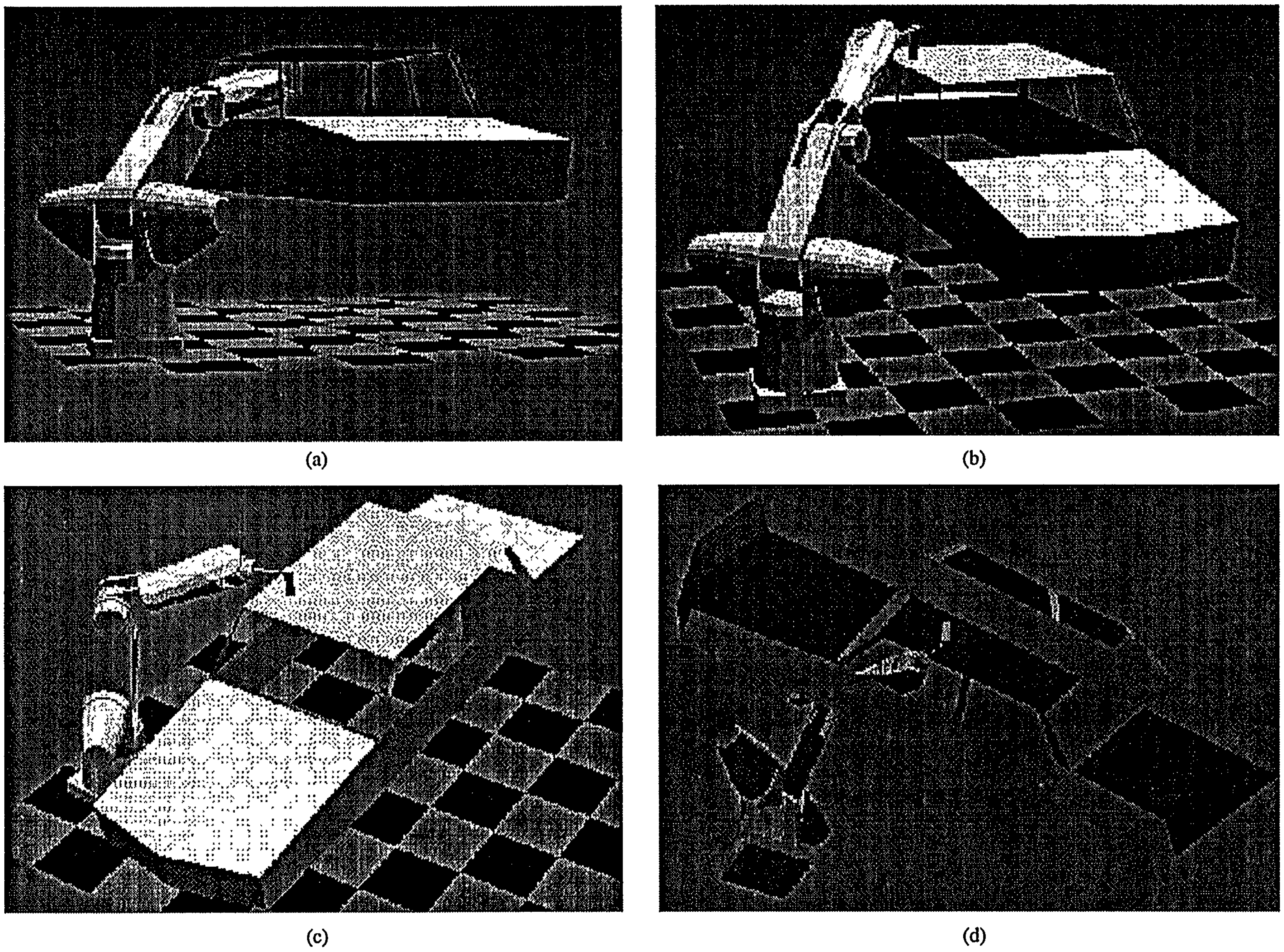


Figure 18. The optimal base configuration has a (0.0, –10.0, –5) offset from the nominal configuration in Figure 2. Figures (a) through (d) show the robot configuration used for each weld point as well as the order of spot welding.

5 Conclusions and Future Work

We have presented a classification of robotics tasks according to the dimension of the manifold the robot tool has to cover, and developed a hierarchy of motion planners based on the novel search strategy called SANDROS. Two basic planners, the point-to-point motion planners and the motion planner for tracing a single curve, are developed first. They are then used to plan motions that visit multiple points, trace multiple curves or cover multiple surfaces. A traveling salesman problem is formulated in the manipulator joint space, and an algorithm is developed to determine the optimal order of tool paths. In addition, the robot-base configuration that minimizes the cycle time can be determined using these planners. These planners provide a compact and systematic approach to motion planning for a variety of robotic tasks, rather than developing a specific motion planner for each task.

Our planners enable automatic generation of collision-free motions for robots in both manufacturing and unstructured environments, and thus make it easier and cheaper to use robots. Manufacturing engineers or robot operators can concentrate on higher-level tasks instead of tedious motion planning and programming. Virtual and agile manufacturing also benefits from our results by verifying feasible motions in simulation in the manufacturability study.

Future work includes improvements of several components of the motion planners such as the TSP algorithm, the local planners in the VP and TSC planners, and faster distance computation. Extension of our planners to 3-dimensional tasks, e.g., rapid prototyping, is more challenging due to the constraints relating to material deposition/removal processes.

References

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

[2] Antonio, J.K., "Optimal trajectory planning for spray coating," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 2570-2577, San Diego, CA, 1994.

[3] Baillieul, J., "A constraint oriented approach to inverse problems for kinematically redundant manipulators," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1827-1833, Raleigh, NC, March-April 1987.

[4] J. Barraquand and J-.C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1712–1717, Sacramento, CA, 1990.

[5] Buckley, C.E. and Leifer, L.J., "A proximity metric for continuum path planning," *Proc. of 9th IJCAI,* pp. 1096-1102, 1985.

[6] J.F. Canny and M.C. Lin, "An Opportunistic Global Path Planner," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1554-1561, Cincinnati, OH, 1990.

[7] Canny, J.F. and Lin, M., "A Fast Algorithm for Incremental Distance Computation," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1008-1014, Sacramento, CA, 1991.

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

[9] Cheng, F.T., Chen, T.H., Wang, Y.S. and Sun, Y.Y, "Obstacle avoidance for redundant manipulators using the compact QP method," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 262-269, Atlanta, GA, 1993.

[10] P.C. Chen and Y.K. Hwang, "SANDROS: A Motion Planner with Performance Proportional to Task Difficulty," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 2346-2353, Nice, France, 1992.

[11] Cook, J.S. and Han, B. T., "Optimal Robot Selection and Work Station Assignment for CIM System," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 2, pp. 210-219, April 1994.

[12] B. Donald, "Motion planning with six degrees of freedom," Massachusetts Institute of Technology Artificial Intelligence Laboratory, AI-TR-791, Massachusetts Institute of Technology, Cambridge, MA, 1984.

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

[14] Feddema, J.T., "Kinematically Optimal Robot Placement for Minimum Time Coordinated Motion," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 3395-3400, 1996.

[15] E.G. Gilbert, D.W. Johnson and S.S. Keerthi, "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.

[16] K.K. Gupta, "A 7-dof practical motion planner based on sequential framework: theory and experiments," *Proc. of IEEE Int. Symp. on Assembly and Task Planning*, pp. 213-218, Pittsburgh, PA, 1995.

[17] Hartmann, K., Krishnan, R., Merz, R., Neplotnik, G., Prinz, F.B., Shultz, L., Terk, M. and Weiss, L.E., "Robot-assisted shape deposition manufacturing," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 2890-2895, San Diego, CA, 1994.

[18] Y.K. Hwang and N. Ahuja, "Gross Motion Planning - A Survey," *ACM Computing Surveys* vol. 24, no. 3, pp. 219-292, September 1992.

[19] Hwang, Y.K. and Chen, P.C., "A Heuristic and Complete Planner for the Classical Mover's Problem," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 729-736, Nagoya, Japan, May 1995.

[20] Hwang, Y.K., Chen, P.C., Maciejewski, A.A. and Neidigk, D.D., "A global motion planner for curve-tracing robots," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 662-667, San Diego, CA, 1994.

[21] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 2138-2145, San Diego, CA, 1994.

[22] Kircanski, M.V., "Symbolical singular value decomposition for a 7-dof manipulator and its application to robot control," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 895-900, Atlanta, GA, 1993.

[23] Klein, C., Chu-Jenq, C. and Ahmed, S., "Use of an extended Jacobian method to map algorithmic singularities," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 632-637, Atlanta, GA, 1993.

[24] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B., *The Traveling Salesman Problem*, New York: John Wiley & Sons, 1985.

[25] J.-C. Latombe, *Robot Motion Planning*, New York: Kluwer Academic Publishers, 1991.

[26] T. Lozano-Pérez, "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.

[27] A.A. Maciejewski, and C.A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. Journal of Robotics Research*, vol. 4, no. 3, pp. 109-117, 1985.

[28] A. Mohri, M. Yamamoto and G. Hirano, "Cooperative path planning for two manipulators," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 2853-2858, Minneapolis, MN, 1996.

[29] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *International Journal of Robotics Research,* vol. 6, no. 2, pp. 3-15, Summer 1987.

[30] K. Nagai and T. Yoshikawa, "Grasping and manipulation by arm/multifingered-hand mechanisms," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1040-1047, Nagoya, Japan, 1995.

[31] B. Paden, A. Mees and M. Fisher, "Path Planning Using a Jacobian-Based Freespace Generation Algorithm," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1732-1737, Scottsdale, AZ, 1989.

[32] P.I. Ro, B.R. Lee and M.A. Seeihammer, "Two-arm kinematic posture optimization for fixtureless assembly," *Journal of Robotics Systems,* 12(1):55-65, John Wiley & Sons, 1995.

[33] Robgoderer, U. and Woenckhaus, C., "A Concept for Automatical Layout Generation," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 800-805, Nagoya, Japan, 1995.

[34] Seereeram, S. and Wen, J.T., "A global approach to path planning for redundant manipulators," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 283-288, Atlanta, GA, 1993.

[35] Suh, S.H., Woo, I.K. and Noh, S.K., "Development of An Automatic Trajectory Planning System (ATPS) for Spray Painting Robots," *Proceedings of IEEE International Conference on Robotics and Automation* pp. 1948-1955, Sacramento, CA, 1991.

[36] Wenger, P., Chedmail, P. and Reynier, F., "A global analysis of following trajectories by redundant manipulators in the presence of obstacles," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 901-906, Atlanta, GA, 1993.

[37] Xavier, P.G., "A Generic Algorithm for Constructing Hierarchical Representations of Geometric Object," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 3644-3651, Minneapolis, MN, 1996.

[38] T. Yoshikawa, "Dynamic manipulability of robot manipulators," *J. of Robotic Systems,* vol. 2, no. 1, pp. 113-124, January 1985.

[39] *IGRIP User's Manual,* Deneb Robotics, Inc., Auburn, MI, 1995.

[40] *CimStation User's Manual,* Silma, Inc., Cupertino, CA, 1995.

DISTRIBUTION:

1	MS	0188	LDRD Office, 4523	
1		9018	Central Technical Files, 8940-2	
5		0899	Technical Library, 4414	
2		0619	Review & Approval Desk, 12690	For DOE/OSTI
1		1110	Pang Chen, 9223	
1		1003	Chris Lewis, 9611	
20		1008	Peter Watterberg, 9621	