A Global Motion Planner for Curve-Tracing Robots *

Yong K. Hwang, Pang C. Chen, Anthony A. Maciejewski† and David D. Neidigk
Sandia National Laboratories
Albuquerque, New Mexico 87185
†Purdue University
West Lafayette, Indiana 47907

Abstract

We present a global motion planner for tracing curves in three dimensions with robot manipulator tool frames. This planner generates an efficient motion satisfying three types of constraints: constraints on the tool tip for curve tracing, robot kinematic constraints and robot-link collision constraints. Motions are planned using a global search algorithm and a local planner based on a potential-field approach. This planner can be used with any robots including redundant manipulators, and can control the trade-offs between its algorithmic completeness and computation time. It can be applied in many robotic tasks such as seam welding, caulking, edge deburring and chamfering, and is expected to reduce motion programming times from days to minutes.

1 Introduction

A significant number of tasks in manufacturing requires robots to trace a curve with their tool tips. These tasks include seam welding, caulking, edge deburring and chamfering. Motions of these robots are currently programmed with either teach pendants or graphical simulation software, and require long programming time. The teach pendant method involves moving the actual robot with the teach pendant and recording robot joint angles. This method takes days of programming time and the programmed motion cannot be modified easily when there is a design change. Simulation software [18, 19] provides a means of programming and reviewing robot motions on a graphics workstation. Although this method significantly reduces programming time, planning 3-dimensional motions from a 2-dimensional computer screen is difficult and thus requires a long training period.

Motion planning for curve tracing is complicated due to three types of constraints: task constraints, robot kinematic constraints and collision constraints. Task constraints are typically constraints on the position and orientation of the robot tool tip needed to perform the task assigned to the robot. Robot kinematic constraints are the physical relationships among robot links and the limits on the ranges of robot joint angles. Collision constraints arise from the need to avoid collisions between robot links and objects during motion. All three types of constraints are nonlinear for robots with rotary joints, and there is no compact way to represent the set of robot configurations, i.e., the set of joint angles, satisfying these constraints. The high dimensionality of the robot configuration space (usually 6 or more) also makes it impractical to use any brute-force type of search on a grid representation.

A key observation on manipulator motion planning is that there are numerous near-optimal solutions for most realistic problems. It is the small set of pathological problems that have impractical worst-case time complexity. Based on this observation, we have developed an efficient motion planning algorithm that solves most realistic problems in a short time (minutes), and requires gradually more computation time as the problem difficulty increases.

2 Previous Work

Most work in manipulator motion planning has been done for the point-to-point problem, i.e., the problem of moving the manipulator from one configuration to another while avoiding obstacles [6, 10]. In these planners, there are no constraints placed on the robot tool tip, and the robot motion is computed to minimize the path length or traveling time [15]. Another body of work involves tracing a curve with a redundant robot. The redundancy is used to optimize secondary objectives such as link collision avoidance [3, 11, 12, 14, 16], singularity avoidance [8, 12], cyclic (drift-free) joint motions for cyclic tool paths [3, 14], or manipulability measures [1, 17]. 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 [11] to get a path that best satisfies the path-tracking and collision-avoidance requirements, whereas compact quadratic programming is used in [3]. Singular value decomposition is used in [7] to improve computational efficiency, and the extended Jacobian method is used to map algorithmic singularities in [8]. All of the above algorithms are, however, local methods; they use a greedy (hill-climb) 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 algorithm for curve tracing that we are aware of is presented in [16]. This algorithm finds *all* joint motions that make the robot tool tip trace a given curve, and if impossible computes portions of the curve that cannot be traced. It computes all inverse kinematic solutions at each point on the curve by discretiz-

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

ing the redundant degrees of freedom, and representing the solutions using a quadtree (octree). Because of the computational burden due to discretization, this algorithm is implemented for at most 2 redundant degrees of freedom.

This paper concentrates on the development of an off-line, global planner for curve-tracing robots in manufacturing environments, rather than a real-time, local planner. Our motion planner computes several near-optimal joint motions that trace a given curve with the robot tool tip, instead of giving *all* solutions as done in [16]. Moreover, our algorithm can be applied to robots with higher degrees of redundancy, and generate better solutions as the computation resource increases. When solving a problem known to be computationally difficult, it is important for algorithms to have the capability of generating solutions with quality commensurate with the available resource. Additionally, our algorithm can handle the case where reconfigurations of the manipulator are necessary to trace a curve (Figure 1). Our planner can also incorporate singularity avoidance, which is explained in Section 3.2.

In this paper, we use the terms *robot* and *manipulator* interchangeably. We define *robot configuration* to be the set of robot joint angles, and *tool tip configuration* to be the position and orientation of the robot tool tip. We define *dof* to be the number of degrees of freedom of the robot.

3 Curve Tracing Algorithm

Our curve-tracing algorithm 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 a satisfactory sequence is found. 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 in Section 3.1. 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.

3.1 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

We model our objects using the ACIS solid modeler due to its open architecture and our need to compute intersections between solids. In the current implementation, our algorithm works only for tracing a continuous, piecewise linear curve among polyhedral objects. 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 2 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 [16] 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 and many solutions. Ideally for our algorithm, we would like to get one solution from each *aspect* [16]. 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* [2]. 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 [16], 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 3 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 a point-to-point motion planner to move the manipulator from an s-reachable subgoal u_0 to an s-unreachable subgoal u_1. The point-to-point motion planner almost always moves the tool tip off the curve, and the generated motion corresponds to a reconfiguration motion. We use the Sandros motion planner in [4] for the point-to-point motion planner. 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 point-to-point motion planner is complete, i.e., guarantees 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 4). 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 the *complete* point-to-point motion planner. Thus, the choice of the critical location for reconfiguration does not affect the completeness of our algorithm.

3.2 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 [11]. We first solve $\Delta x = J(q)\Delta q$ to move the tip along the curve using singular value decomposition. We then use the 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 5 illustrates the local planner.

3.3 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 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 [13].

4 Examples

Our curve-tracing algorithm has been implemented with as much generality as possible. We use the Denavit-Hartenberg parameters to compute forward and inverse kinematics for manipulators, and objects are modeled in the ACIS solid modeler. We have tested our algorithm with a 3 and 4-dof planar manipulators, and the planned motions are shown in Figure 1 and 6, respectively. We have chosen the examples so that a reconfiguration motion is needed in tracing a curve. It took less than 5 minutes to compute the motions in the examples on a 100MIPS workstation. We attempted using robot models in the commercial simulation packages IGRIP and CimStation, but the overhead of calling the distance computation routines was excessive. Our algorithm computes the distance between robot links and objects on the order of 10^{4-5} times, and this has necessitated the use of the fast distance routine in [5].

5 Conclusions

We have developed a global motion planner for curve-tracing robots engaged in operations such as welding, cutting, caulking, deburring and chamfering. This planner is capable of computing near-optimal, collision-free robot motions by taking into account of the global geometric information of objects. This planner can be used with any type of robot including non-redundant and highly redundant robots. The optimality of a computed motion by our planner is commensurate with the available computing resource. Our planner is currently implemented for polyhedral objects. Extending our planner to curved objects will require distance computation between curved objects. Our planner also does not take into account velocity constraints, which are crucial in tasks such as welding and caulking. We plan to extend our planner to eliminate these limitations and integrate it with the commercial simulation software, enabling direct use of models of robots and workcells in the simulation software.

References

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

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

[3] 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.

[4] Chen, P.C. and Hwang, Y.K., "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.

[5] 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, 1988.

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

[7] 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.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] Paden, B., Mees, A. and Fisher, M., "Path planning using a Jacobian-based freespace generation algorithm," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1732-1737, Scottsdale, AZ, 1989.

[14] 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.

[15] Shiller, Z. and Dubowsky, S., "On computing the global time-optimal motions of robotic manipulators in the presense of obstacles," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 6, pp. 785-797, Dec. 1991.

[16] 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.

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

[18] *IGRIP User's Manual*, Deneb Robotics, Inc., 1992.

[19] *CimStation User's Manual*, Silma, Inc., 1992.

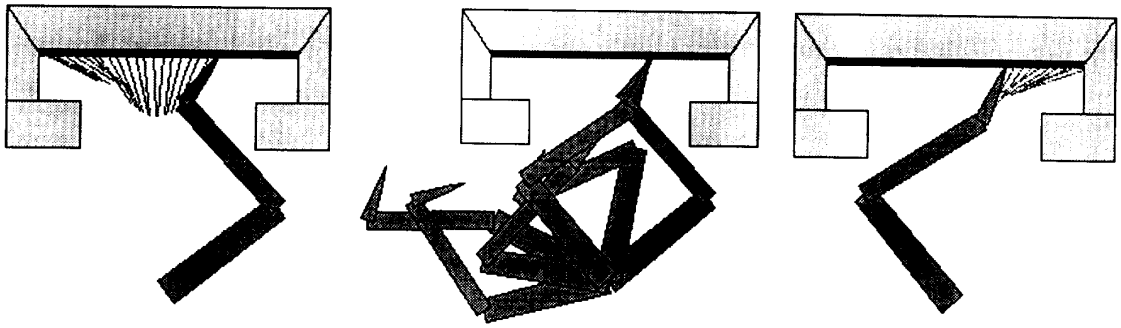

Figure 1. The robot has to reconfigure itself in the middle of tracing the bold edge. The traces of the tool tip are shown.

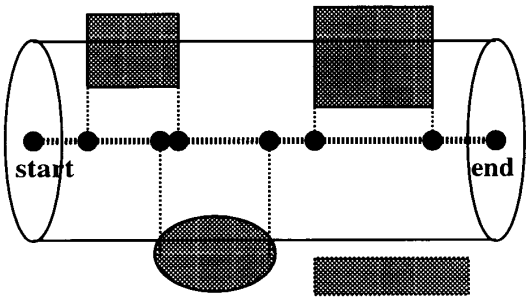

Figure 2. Critical locations shown in black circles are the beginning and end of the intervals in the curve where objects are close by. The fourth critical location will be deleted as it is close to the third critical location.

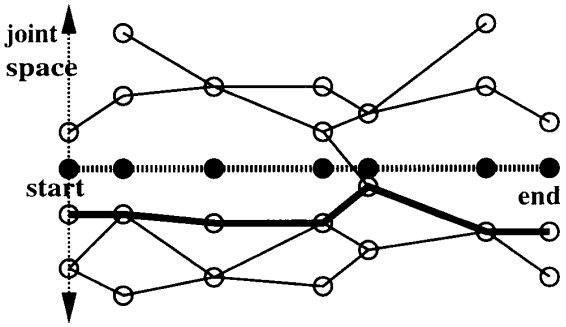

Figure 3. Subgoals shown with white circles at the critical locations represent inverse kinematic solutions. A graph is constructed by connecting close subgoals at the adjacent critical locations. The thick line shows the shortest sequence of subgoals from the start to the end of the curve.

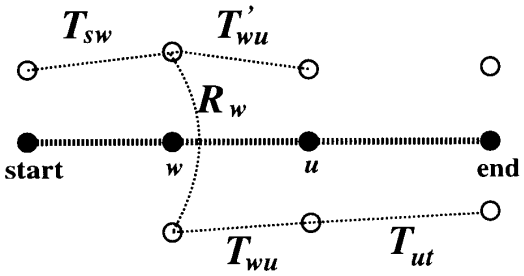

Figure 4. If there is a curve–tracing motion which includes a reconfiguration motion, the reconfiguration motion can always be done at the last critical location u reachable from the start of the curve.

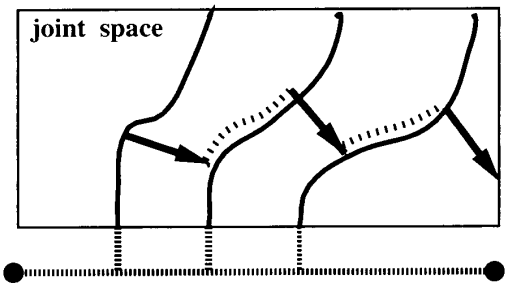

Figure 5. Each curve in the joint space denotes all inverse kinematic solutions that place the tool tip at the corresponding point on the curve. The local planner first moves the tool tip along the curve (arrows), and then uses the null space movement to avoid collisions and optimize the tool tip orientation (dotted lines).

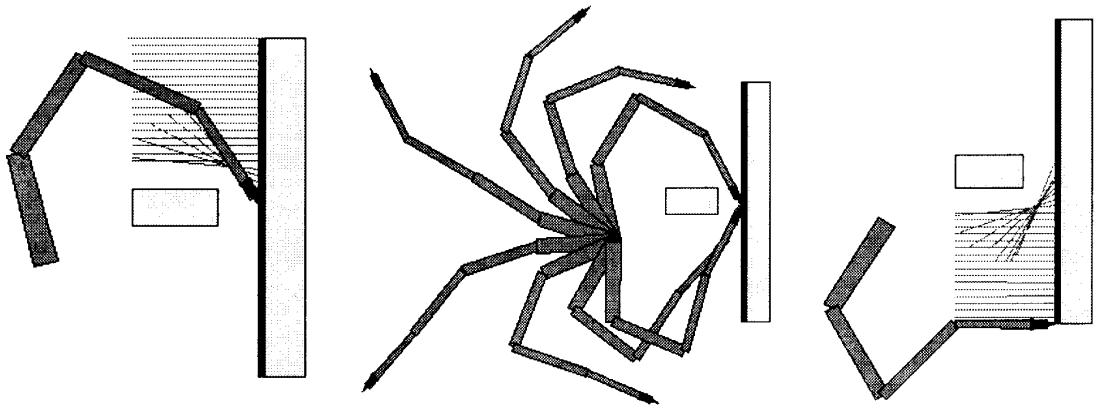

Figure 6. A planar 4–link robot needs a reconfiguration to trace the bold edge. The tool tip orientation deviates from the edge normal only if necessary to avoid collisions.