Proceedings of the 1995 International Conference on Robotics and Automation.

A Heuristic and Complete Planner for the Classical Mover's Problem*

Yong K. Hwang **Pang C. Chen**
Sandia National Laboratories
Albuquerque, NM 87185-0951, USA

Abstract

We present a motion planner for the classical mover's problem in three dimensions that is both resolution-complete and efficient in that it has performance commensurate with task difficulty. It is based on the SANDROS search strategy, which uses a hierarchical, multi-resolution representation of the configuration space along with a generate-and-test paradigm for solution paths. This planner can control the trade-offs between the computation resource and algorithmic completeness/solution path quality, and thus can fully utilize the available computing power. It is useful for navigation of mobile robots, submarines and spacecraft, or part motion feasibility in assembly planning.

1 Introduction

The classical mover's problem is the problem of finding a path for a rigid object between the start and target configurations while avoiding obstacles. The solution is usually given as a sequence of positions and orientations of the object. This problem is the most basic motion planning problem, and people solve this problem every day in moving furniture and assembling mechanical parts. If robots are to act autonomously at the skill level of people, they must possess the power to solve the classical mover's problem. Even when the robots are tele-operated, various levels of automatic motion planning will greatly speed up the overall robotic operation by allowing the operator to issue higher-level commands at a coarser time interval.

Although people solve the classical mover's problem quite easily and efficiently, the problem turns out to have a high complexity. At present the most efficient motion-planning algorithm has a complexity that is polynomial in the number of obstacles and exponential in the number of degrees of freedom of the robot. For the two dimensional (2D) case with three degrees of freedom (two translational and one rotational), a brute force search can be used to solve most problems in a few minutes. For the three dimensional (3D) case, however, the 6 dimensional configuration space is too large for a brute force search even when geometry is simple, and a more powerful search strategy is needed. In spite of many approaches developed for motion planning, there is no planner to this date for the 3D classical mover's problem that is both complete (guarantees a solution) and runs in a practical time limit (less than an hour for off-line and less than a few minutes for on-line applications).

In our pursuit of a heuristically efficient and complete motion planner, we have noted that there are numerous feasible solutions for most realistic problems. It is the small set of pathological problems that have impractical worst-case time complexities. 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. This planner is based on the SANDROS search strategy that we have used for manipulator path planning in [5], and is observed to be quite efficient for the classical mover's problem in many examples.

The major difference between the previous and new SANDROS planners is as follows. For a manipulator, each degree-of-freedom (dof) controls the configuration of a subset of links (usually one) and the subgoal selection can be done sequentially, i.e., by specifying the value of the first joint, and then second, etc. This gives a natural way to generate *big* subgoals, e.g., we can select good (large clearance from obstacles) values for the first joint angle by placing the first link in wide free space. For a single rigid object, however, all dofs control the configuration of the single object, and all dofs must be specified to do a collision check or distance computation. This forces us to have only *point subgoals* as opposed to *big* subgoals necessary for the hierarchical and multi-resolution search paradigm. Developing an efficient SANDROS planner with point subgoals is the major contribution of this paper, and this shows that SANDROS is a widely applicable search strategy.

In this paper, we refer to the object being moved as the (mobile) *robot*. The set of all robot poses is referred as the *configuration space* or C-space. A robot configuration defines a robot pose by specifying its position and orientation. In general, it is 3-dimensional (2 translational/1 rotational) for 2D environments, and 6-dimensional (3 translational/3 rotational) for 3D environments. The next section reviews the previous work done on the classical mover's problem, and Section 3 describes our motion-planning algorithm. The performance of our planner is shown in Section 4, and the conclusions and future work are discussed in Section 5.

2 Previous Work

There has been a lot of work, both theoretical and experimental, on the classical mover's problem since late 70's. On the theoretical side, the lower bound on the complexity

*This work has been performed at Sandia National Laboratories and supported by the U.S. Department of Energy under Contract DE-AC04-94AL85000.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

of a generalized mover's problem is shown to be PSPACE-complete [21]. The first upper bound is derived with an algorithm that is doubly exponential in dof and polynomial in the number of objects [23]. This algorithm is based on Collins' cylindrical decomposition method [6]. The result is later improved to a single exponential algorithm [8] using one-dimensional curves at the boundaries of the C-space obstacles. Since these theoretical algorithms are likely to require days of running time if implemented, many approximate or heuristic algorithms have been developed.

For the 2D classical mover's problem, the C-space is only three dimensional, and a brute force search algorithm can be used. A brute force search would discretize the 3-dimensional configuration space with about a million points(100x100x100), check whether there is a collision at each of these points, and find a connected sequence of collision-free points in the C-space. Although there are many algorithms for 2D based on skeletons [19, 15], cell decomposition [3, 20] or potential fields [12], it is our view that the brute-force search runs fast enough on present workstations. In fact, such a brute-force algorithm has been implemented in [18, 22, 14]. We will, however, show a 2D version of our algorithm for illustrative purposes.

For the 3D problems, we describe five algorithms that are implemented and representative of four different approaches. The first implemented algorithm for 3D is a grid search algorithm [9]. It sets up a grid of points in the configuration space, and collision checking for a point is done by evaluating the boundary equations of the C-space obstacles. It uses several heuristics to guide the search direction so as to find a solution path in a short time. 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. In [7], the free space skeleton in the C-space is computed by computing points with maximal distance from obstacles. A cell decomposition approach [20] is developed using a hierarchical cell decomposition and the robot Jacobian. The bound on the Jacobian gives the maximum distance traveled by any point on the robot when the robot configuration is varied inside a unit sphere. Thus, the distance between the robot and objects divided by the bound on the Jacobian gives the radius of the collision-free sphere in the C-space. This algorithm works well for round robots; if the robot is elongated, the bound on the Jacobian is large and only a small sphere of free space is obtained per distance computation. All of the algorithms above guarantee solutions if exist, and we estimate their average running times at a few to several hours on today's computers.

A randomized planner [2] is probabilistically complete in the sense that if it runs long enough, it will find a solution. In this algorithm, the robot approaches the goal using a simple planner. If the robot cannot get to the goal, the robot takes a random walk, and then tries to approach the goal from there. This algorithm runs fast on most problems, but will take a long time to find a solution for hard problems. The main difference between this planner and our algorithm is that our algorithm keeps a search record so that all parts of the C-space is eventually searched. Finally, a potential field approach in [12] performs efficiently when the robot is approximately convex shaped. It finds a path for a point robot first, and then moves robot's reference point along this path while aligning robot's longest axis tangentially to the path to minimize the swept volume. If the robot collides with an object, it generates feasible robot orientations in the collision region, and uses these as intermediate subgoals. This algorithm sacrifices completeness for efficiency (runs in less than an hour), but fails on problems with highly concave robots like that in Figure 3. Our algorithm is motivated by the efficiency of heuristic planners and the completeness of exact algorithms. It works like a heuristic planner initially, but it searches systematically so that it will eventually exhaustively search the entire C-space if necessary. Incidentally, if there are more than one problem to be solved, then learning [4] or preprocessing [13] can be used to amortize the cost of each problem.

3 Algorithm

We have developed an efficient, resolution-complete algorithm capable of planning motions for a rigid object navigating in 2D or 3D environments. (Our notion of completeness will be discussed in Section 3.3.) We plan a motion of a robot in the C-space. To plan, we make the standard assumption that if a task is solvable, then it has a solution path representable by a sequence of unit movements in the C-space, discretized to a preset resolution. To handle the potentially high dimensional 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 [11] between the robot at the given configuration and the environment. One can use contact conditions between the robot and obstacles to get a set of collision-free configurations as done in [9]. However, equations describing contact boundaries are nonlinear, and detecting intersections of these boundaries is computationally expensive. Instead, we compute the distance between the robot and obstacles to check collisions, as well as to get a moving direction away from the obstacles.

Second, we use a two-level hierarchical planning scheme to reduce memory requirement as done in [10]. It would be 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 a global and 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 [7], 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 thousands of path segments generated during the search process.

Third, we use a multi-resolution approach to reduce search time. The high dimensionality of the C-space has hindered the development of a fast, complete motion planner. An exhaustive search for a collision-free motion is prohibitive because of the enormous size of the 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 general search strategy called SANDROS, which has been previously used to develop a motion planner for robot manipulators [5]. The acronym stands for *Selective And Non-uniformly Delayed Refinement Of Subgoals,* which indicates the nature of the search strategy. 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 only these coarse guidelines if the problem is easy. However, if a collision-free path cannot be found with these subgoals, then some of the subgoals are broken down to several smaller, heuristically selected subgoals to provide more specific guidelines. The process of subgoal refinement is delayed as much as possible, and is performed in a non-uniform fashion.

To recapitulate, the SANDROS planner 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 the SANDROS strategy until either a sequence becomes available, or no further refinement is possible. It should be noted that $\mathcal{G}$ 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 powerful as one of the exact algorithms mentioned above, then the global planner will never generate any subgoals because the the local planner's range of effectiveness encompasses the entire configuration 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 almost completely on the global planner in that it will have to generate many subgoals before a solution can be found. Finding the optimal division of labor between $\mathcal{L}$ and $\mathcal{G}$ is difficult. However, in principle, $\mathcal{L}$ should implement some 'greedy' algorithm with 'sliding' capability, so that it can quickly solve most problems without requiring much memory.

The main difference between the present SANDROS planner for mobile robots and the previous SANDROS planner for manipulators lies in the specifications of $\mathcal{G}$ and $\mathcal{L}$. In particular, there are subgoal representation/refinement differences in $\mathcal{G}$ and sliding strategy difference in $\mathcal{L}$.

3.1 Global Planning

Before starting the global planning process, we first make sure that the local planner $\mathcal{L}$ is indeed incapable of solving the task without additional subgoals; otherwise, we are done. 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 points s and t with additional points representing subgoals and nodes representing subspaces of subgoals. A node v is simply a cuboid cell of the C-space. (This representation is different than that for manipulators [5], which uses hyperplanes to represent partially specified subgoals.) It can either be *focused*, or unfocused, depending on whether a collision-free *target* point p inside the cell has been found. A focused node is further classified as *reachable* or not yet reachable, depending on whether $\mathcal{L}$ declares the associated point reachable from other reachable points. Although G may be changing, we maintain the invariant that the nodes of G form a partition of the C-space. Thus, nodes are divided into three sets: the reachable nodes U, the not-yet-reachable nodes V, and the unfocused nodes W. The points of G are either reachable (in P) or not-yet-reachable (in P'). Since the nodes of G forms a partition of the C-space, each point p has an unique node $A(p)$ of G that contains p. The set P is further divided into P_s and P_t representing points reachable from s and t, respectively. For each point in P, a point cost is stored indicating the cost of reaching it from s or t.

There are two types of edge connections in G: (node)-to-(node) and (reachable point)-to-(not-yet-reachable node). Two nodes are connected with an edge iff they are adjacent in that their contact area is nonzero. The edge cost between two nodes is defined as the Euclidean distance between the two associated points if they exist, infinity otherwise. A point p in P is connected to a node v in V iff the node u containing p is adjacent to v. We initialize G by setting $P_s = \{s\}$, $P_t = \{t\}$, $U = \{root\}$, and the rest to empty set. Here *root* denotes the root node that represents the entire C-space. To control the node refinement process, we also maintain a node queue Q, initialized to U.

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.

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 sequence of points associated with 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, let q be the point associated with v. If $\mathcal{L}$ is able to connect p to q, then v is declared reachable, and we would swap v from V to U, insert q into P, connect q to the original neighbors of v in V with new edges, and 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 nodes in Q until either a candidate sequence becomes available, or Q becomes empty. We pop off every node u in Q with the largest volume, and refine each cell by cutting the longest side in half. Moreover, if u contains a point p in P, then $A(B(p))$ is pushed into Q to ensure the eventual chance that every node of U gets refined. Refinement stops when the longest side of u is less than a preset length $\lambda \geq 1$.

After refining u into two children $C(u)$, we need to modify G to reflect the replacement of u by $C(u)$. First, we determine the node type for each child v. If v contains a point in P, then insert v into U; if v contains a point in P', then insert v into V; otherwise, sample for a collision-free point p in the cell of v. If the sampling is successful, then insert p into P' and v into V; otherwise, insert v into W. (For our work, the sampling is uniform, and the number of samples taken is proportional to the longest side of the cell.) Next, we disconnect u and connect each child v to its adjacent nodes. Finally, if v belongs to V, then every point of P in nodes adjacent to v also needs to be connected to v.

3.2 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 degree of freedom 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 with some minimum distance threshold is done after the robot takes a step. We assume that the strides are set sufficiently small to guarantee collision-free motion for the entire step.

Given two configuration points p and q, the local planner $\mathcal{L}$ attempts to move the robot from p to q. The iterative procedure is as follows: First, we move toward q by considering all neighboring points of p that are one step closer to q than p is, and move to the point p' that has the maximum clearance $d(p') > 0$. 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 q than p is (under the L_∞-norm), 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 q is reached.

3.3 Completeness

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

It is of course possible to refine every node of G down to a point first, and then plan a path based on the resulting network of points $\bar{G}$. In fact, if we were to use $\lambda = 1$, then the resulting network is simply the discretized map of the free C-space. However, it would be terribly inefficient to find the shortest plausible sequence of subgoals because of the size of $\bar{G}$. For such a $\bar{G}$, $\mathcal{L}$ only needs to check whether an adjacent point is collision-free. Non-trivial λ allows us to combine similar subgoals into one single subgoal so that G contains only a small number of subgoals that are also representative of all regions of the free configuration space. We therefore reduce the size of $\bar{G}$ by using a larger λ, and utilize the power of $\mathcal{L}$.

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

that the algorithm is again complete for mobile robots with nodes represented by cuboid cells rather than hyperplanes. Note that theorem depends on both the invariant that the nodes of G form a partition of the C-space and the fact that new target points are uniquely determined for every node, regardless of when the node is generated.

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

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

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

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

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

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

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

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

4 Examples

Our algorithm has been implemented for 2D and 3D environments, and tested with both easy and hard examples. The examples were run on a typical (100MIPS) workstation. Figure 1(a) shows an L-shaped object moving from the center of the workcell to the left of the triangle via the subgoal at the top. Two lightly shaded subgoals are also generated but not used for the solution path. Figure 1(b) show the solution path (dark curves) and other path the planner explored during the search (light curves). Note that for the orientation dimension (z axis), top and bottom surface represent the same orientation. The computation time was 1 second. Figure 2a shows a moderately hard problem of unhooking a clip from a nail, and the subgoals used are shown. The computation time was less than 1 second. Figure 3 shows the same clip, but there are two nails this time. This problem with many traps in the C-space was generated to test the completeness of our algorithm. It took 219 local planning calls and 15 seconds to solve this problem, generating 116 point subgoals (Figure 3(c)). The C-space resolution was 120x120x240 = 3.5 million points. We include these examples to illustrate our algorithm, even though in 2D a brute-force search is sufficient.

The real power of our algorithm shows in 3D environments, where the size of the C-space is much larger. Figure 4 shows an L-shaped object turning in an L-shaped hallway (the two walls in the front are not drawn). Although this problem is not trivial since the robot has to make two rotation moves at the corner, our planner was able to solve this problem with just one local planning call. The computation time was less than 1 second, and it illustrates the problem-solving capability of our local planner. The next two problems in Figure 5 and 6 show how to get a T and a Pi (Greek letter) out of a jail cell. These problems are motivated by pipe removals in wastesite cleanup. The T took 4 point subgoals and 10 seconds to get out, while the Pi took 32 point subgoals and 135 seconds. The C-space resolution was 100x100x75x250x250x157 = 7.4 trillion points. All of the examples above show the computation time of our algorithm is correlated with the problem complexity.

5 Conclusions and Future Work

We have developed a global motion planner for the classical mover's problem. This planner is resolution complete, and at the same time runs in a practical amount of time. Our planner can continue the search after generating a solution to get a more optimal path. Our planner can be parallelized easily either by computing the distance between the robot and all obstacles in parallel, or by verifying a sequence of subgoals in parallel. It can also scale the amount of computation according to the available computing resource.

Our current planner has lots of room for improvement. The probability of reaching a node needs to be estimated so that the global planner can generate a test sequence of subgoals with a higher chance of success. Two local planners can be used instead one: a fast one that favors moving toward the goal over obstacle avoidance and a safer but slower one that favors moving away from obstacles. By applying the latter after applying the first, the ratio of the success rate to the computation time can be increased. The goal of the local planner can be either a point (singleton) as done in this paper, or can be any point in a node (set) as done for our SANDROS planner for manipulators [5]. It is our impression that the local planner will have a higher chance of success when a bigger goal is given. If the goal is a set, however, we do not have a completeness proof, and the planner cannot be parallelized (since we do not know which point in a node the local planner will reach). Further

work is needed in this area.

Our planner is applicable to the important task of navigating mobile robots, submarines and spacecraft. It can also be used in object manipulation; one usually plans the path for the object to be moved and then plans the motions of manipulators, especially when multiple robots are involved [16]. Still another application is in assembly planning to check the existence of a part motion to its assembled position. We expect that our planner will play a fundamental role in many robotic applications.

References

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

[2] Barraquand, J. and Latombe, J.C., "A Monte-Carlo Algorithm for Path Planning with Many Degrees of Freedom," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1712-1717, Cincinnati, OH, 1990.

[3] Brooks, R.A. and Lozano-Perez, T., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," *International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, 1983.

[4] Chen, P.C., "Improving Path Planning with Learning," *Proc. of Ninth Int. Conf. on Machine Learning,* pp. 55-61, Aberdeen, Scotland, 1992.

[5] 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.

[6] Collins, G., "Quantifier elimination for real closed fields by cylindrical algebraic decomposition," *Second GI Conference on Automata Theory and Formal Languages*, 33, pp. 134-183, New York: Springer-Verlag, 1975.

[7] Canny, J.F. and Lin, M.C., "An opportunistic global path planner," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1554-1561, Cincinnati, OH, 1990.

[8] Canny, J.F., "A New Algebraic Method for Robot Motion Planning and Real Geometry," *Proceedings of 28th Annual Symposium on Foundations of Computer Science*, pp. 39-48, 1987.

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

[10] Faverjon, B. and Tournassoud, P., "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, Raleigh, NC, 1987.

[11] 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.

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

[13] Kavraki, L. and Latombe, J.-C, "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.

[14] Kavraki, L., "Computation of Configuration-Space Obstacles Using the Fast Fourier Transform," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 255-261, Atlanta, GA, 1993.

[15] Kedem, K. and Sharir, M., "An Automatic Motion Planning System for a Convex Polygonal Mobile Robot in 2-D Polygonal Space," *Proceedings of 4th Annual ACM Symposium on Computational Geometry*, pp. 329-340, Urbana, IL, 1988.

[16] Koga, Y, L. and Latombe, J.-C, "On multi-arm manipulation planning," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 945-952, San Diego, CA, 1994.

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

[18] Lengyel, J., Reichert, M., Donald, B.R. and Greenberg, D.P., "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware," *Computer Graphics*, vol. 24, no. 4, pp. 327-335, August 1990.

[19] Lozano-Perez, T. and Wesley, M.A., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560-570, 1979.

[20] 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.

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

[22] Ressler, E., "Planning 2D Solid Robot Motion on A Discrete Grid," *Course Notes for "Robotics and Machine Vision"*, Cornell University Department of Computer Science, Ithaca, New York, 1992.

[23] Schwartz, J.T. and Sharir, M., "On the Piano Movers' Problem: II. Techniques for Computing Topological Properties of Real Algebraic Manifolds," Courant Institute of Mathematical Sciences, Report No. 39. (also in *Advances in Applied Mathematics*, 1983, no. 4, pp. 298-351).

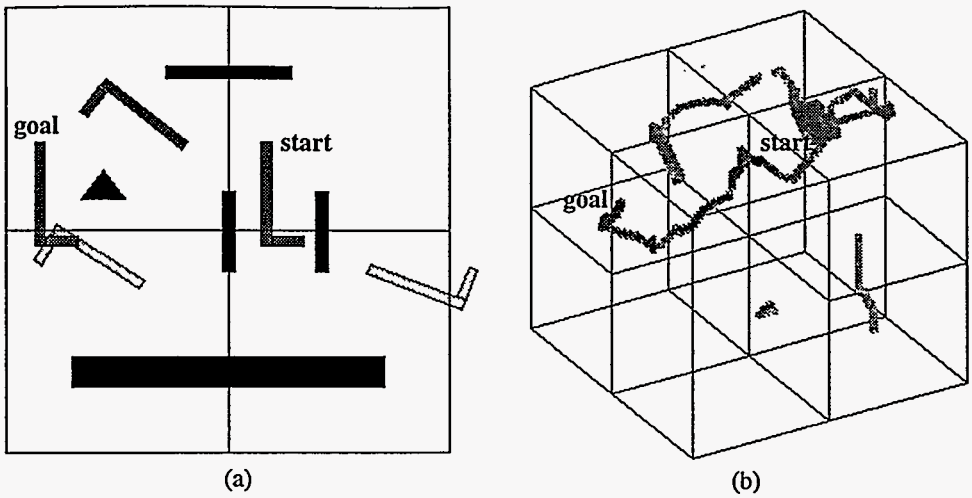


Figure 1: An L–shaped object in planar environment (a) with C–space (b).

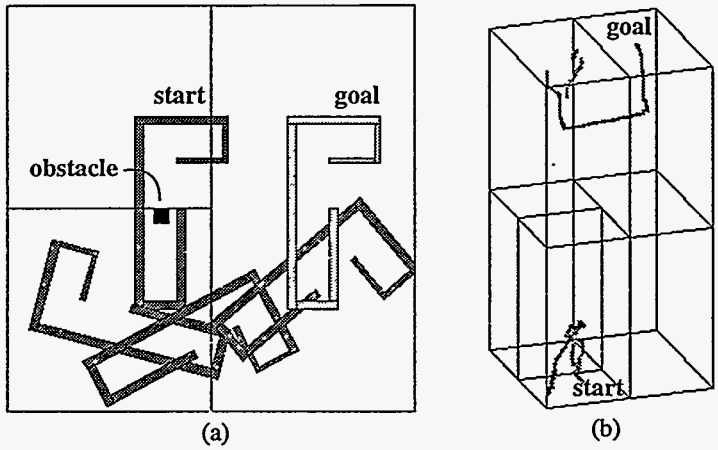


Figure 2: A clip with one nail problem in planar environment (a) with C–space (b).

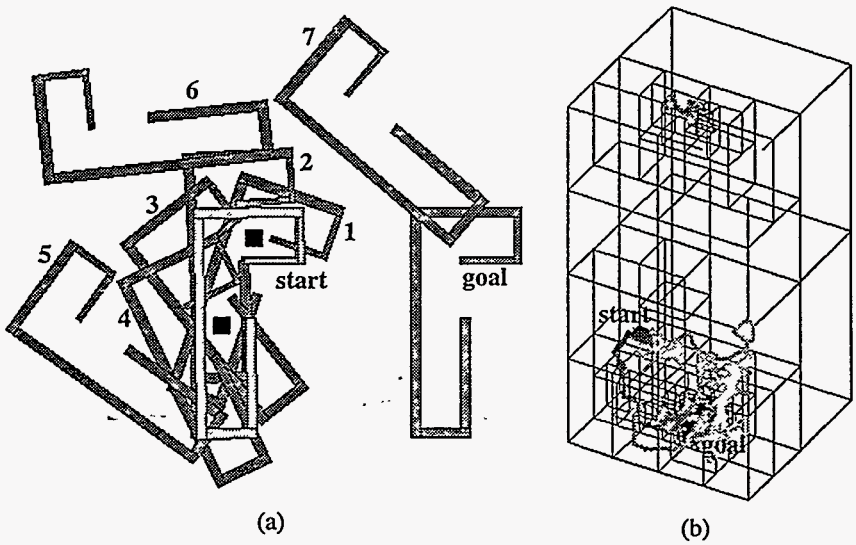


Figure 3: A clip with two nails problem in planar environment (a) with C–space (b).

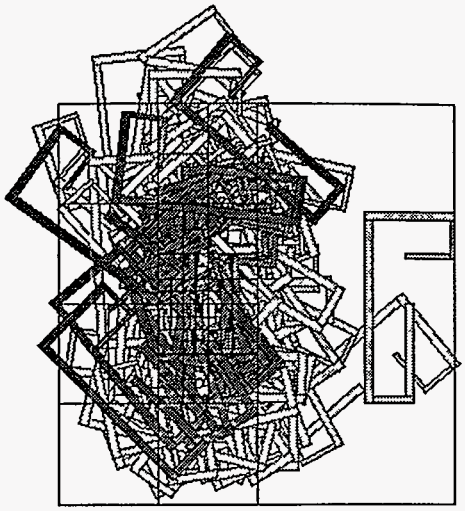


Figure 3c: Subgoals generated for the clip–with–2–nails problem.

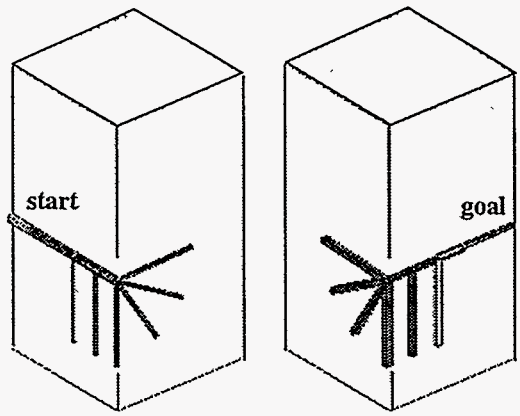
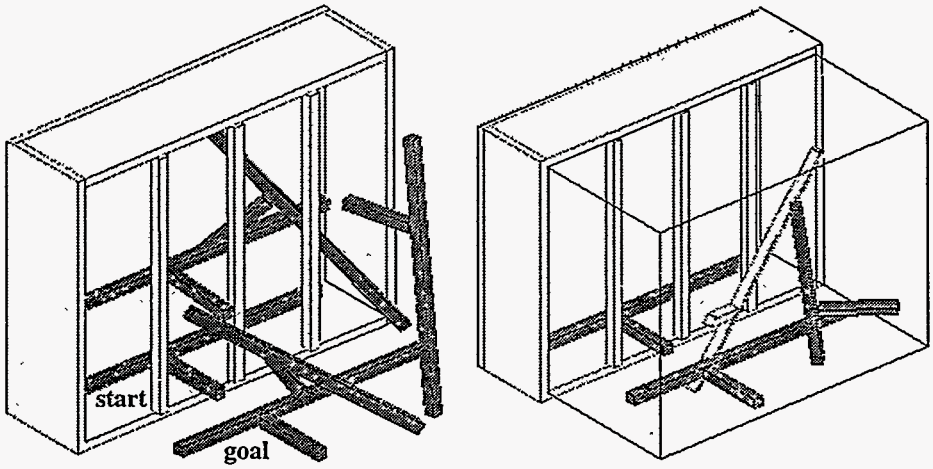
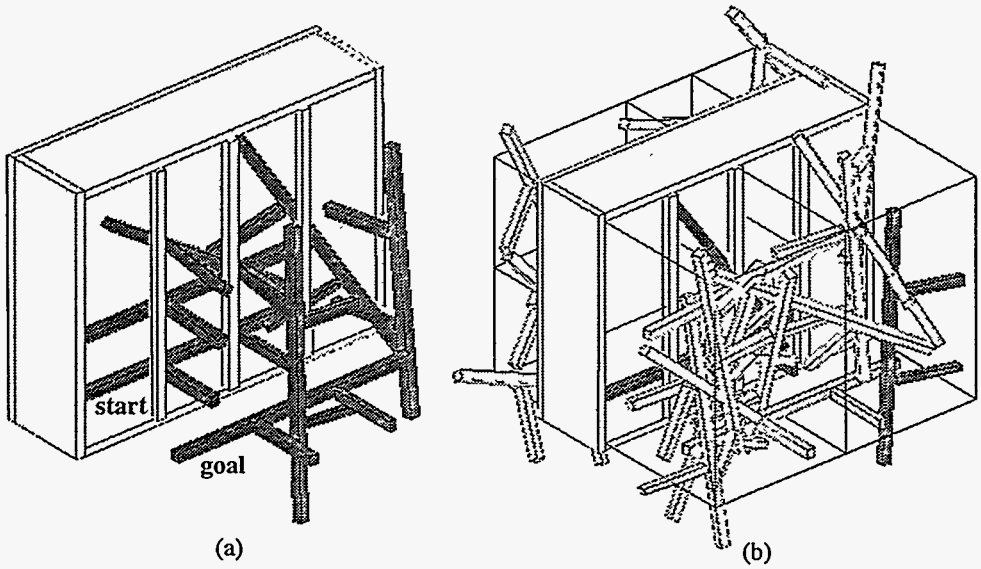


Figure 4: An L–shaped object moving around a block.



(a) (b)

Figure 5: A T–shaped object moving out of a jail box in (a) with generated subgoals in (b).



(a) (b)

Figure 6: A Pi–shaped object moving out of a jail box in (a) with generated subgoals in (b).

DISCLAIMER