MOTION PLANNING IN INDUSTRIAL DESIGN

A Senior Thesis

By

Darek VanToorn

1996-97 University Undergraduate Research Fellow

Texas A&M University

Group: ELECTRICAL ENGINEERING/COMPUTER SCIENCE

Motion Planning In Industrial Design

- Preliminary Version -

To be updated with final version in Fall 1997

Darek VanToorn
University Undergraduate Research Fellow, 1996-97
Texas A&M University
Department of Computer Science

APPROVED

Undergraduate Advisor [illegible]

Exec. Dir., Honors Program [illegible]

Motion Planning In Industrial Design. Darek VanToorn (Dr. Nancy Amato), Computer Science, Texas A&M University.

Automation of the design and manufacturing process of complex mechanical systems could save countless man-hours and reduce costs. The development of practical motion planning algorithms could aid this automation by increasing the power of computer-aided design tools so that they can be used to perform manufacturability and maintainability studies. In this paper we present a new motion planning method called single-shot obstacle-based probabilistic roadmap method, or ssOBPRM. This new method is based on previous motion planning methods but addresses observed weaknesses in these methods for the industrial applications we consider. This new method is aimed at efficiency, only performing enough processing to obtain a collision-free path from one robotic configuration to another or else to determine that a path cannot be computed in a reasonable amount of time. A description of this algorithm is provided along with insight into its performance and possible future improvements and directions upon this project. From our experimental results we hope to establish that this new planning method is indeed successful and more efficient than existing ones, and therefore a viable motion planning algorithm for use in computation intensive design and manufacturing studies necessary in industry.

Table of Contents

Implementation of Motion Planning in Industrial Design

1. Introduction

Companies involved in the design and manufacture of complex mechanical systems (e.g., passenger jets, spacecraft, automobiles, and engines) could save countless man-hours if motion plans could be automatically generated from geometric, kinematic, and dynamic models of the system. In particular, the development of practical planning algorithms could lead to the automation of assembly and other fine motion tasks and to the design of part-removal and other maintenance operations to be carried out in highly cluttered environments (e.g., under the hood of a car, inside the walls of an aircraft fuselage).

For example, suppose a maintenance requirement for a mechanical system is that a certain part can be removed from the system without disassembly. In the past, such requirements were often tested by building a physical mock-up of the system, and then attempting to remove the part manually. More recently, as computer-aided design (CAD) tools have gained wide-spread use, attempts have been made to test such maintainability requirements using CAD models. In this situation, the engineer will usually specify a potential removal path for the part, and then automated techniques can be used to determine whether the removal path is feasible, e.g., collision-free. In addition to reduced costs, CAD-based maintainability studies are superior to physical mock-up since, once it is determined, the removal path can be archived for use in developing, e.g., maintenance procedures. However, there are still problems with the CAD-based approach. Most importantly, the engineer is required to determine the removal path to be tested (at least roughly), and the automated system can only test the feasibility of this path. In mechanical systems, however, the determination of such a path can be very difficult since, for space efficiency, the designs are often very compact and crowded. Thus, automated motion planning methods for finding a removal path, if one exists, are needed.

Motivated by the needs described above, our research focuses on automated methods for analyzing product designs for manufacturability and maintainability. In particular, we are studying how motion planning methods developed in the field of robotics can be adapted to perform such tasks. While there are many similarities between robotic path planning and the industrial design problems we are considering, there are also a number of differences which require new techniques. For example, many robotic path planning methods spend a large amount of computation pre-processing the environment so that later, planning queries can be answered quickly. However, such methods are most useful for static environments, which is not the case for these applications since each design change corresponds to a new environment. Another difference is that mechanical systems are typically more crowded than environments encountered in robotic path planning. Thus, although our research builds on robotic motion planning, these methods must be modified and new techniques developed that are suitable for analyzing designs of mechanical systems. Before describing our research in more detail, a review of related work in robot motion planning is necessary.

2. Preliminaries and Previous Related Work on Motion Planning

The basic motion planning problem can be stated as follows: it must be determined whether there exists a path such that a robot (a single rigid object or a number of rigid objects connected by joints which can move independently) can be moved in a two or three dimensional workspace from a start position and orientation to some goal position and orientation without colliding with any objects, and if so, to find one such path [2,4].

2.1 Complexity of Motion Planning

Recent research in motion planning has generated numerous methods, most of which work only in specific cases and have a number of limitations. Part of the difficulty in finding an efficient algorithm is that most planning problems are at least PSPACE-hard. A problem is said to be in P if there is a polynomial-time algorithm to solve it. A problem is in NP (non-deterministic polynomial time) if its solution can be checked by an algorithm in polynomial time. An NP-hard problem is at least as difficult as any NP problem. The NP-complete problems are those that are NP-hard and contained in NP. It is possible that P=NP, but nobody knows since no exponential lower bound has been found for an NP problem [8]. A problem is in PSPACE if it requires space (for memory) that is polynomial in the problem size. Definitions are similar for

PSPACE-hard/complete problems. The motion planning problem for a robot of polyhedral parts connected by joints in a 3-D workspace of polyhedral obstacles is PSPACE-hard [6]. The problem for a planar arm robot with non-extensible links serially connected by revolute joints in a plane with polygonal obstacles is also PSPACE-hard [5]. Furthermore, the problem of finding the shortest path between two points in a space of polyhedral obstacles is NP-hard; however, in the plane this problem is simply polynomial [2]. So it can be seen that these few seemingly simple problems actually have high complexity.

2.2 C-Space

Before continuing, the notion of Configuration Space (C-Space) must be defined. The basic idea is to represent a robot as a point in the appropriate space, called C-Space, in which workspace obstacles have been mapped into objects called C-obstacles. A simple example is shown in Figure 1 below. The C-Space is the set of all possible configurations of the robot, each configuration being represented by a d-tuple in which each variable specifies one of the d degrees of freedom (dof) of the robot [1]. The dof of a robot is the number of parameters necessary to uniquely specify a configuration of the robot. For example, a point in space would require three dof: X, Y, and Z coordinates, while a solid object in space requires six dof: X, Y, and Z coordinates as well as Θx, Θy, and Θz to uniquely specify its location *and* orientation.

This transformation from the actual workspace to C-Space changes the problem of planning for a multi-dimensioned robot into that of simply planning the motion of a point [2]. We represent the start and goal configurations as two points (s and g) in C-Space and try to find a path that connects s and g while avoiding C-obstacles. This approach makes the constraints on the robot more explicit, yet there are still difficulties with the method. First, C-Space will often have high dimension and complexity for even simple problems, due to the relation to the number of dof of the robot. C-obstacles also tend to have very irregular surfaces which are difficult to accurately map into C-Space. Further, the C-Space is not a Euclidean space and therefore joint angles of a robot may wrap around the C-Space, like connecting opposite edges of a piece of paper to form a continuous cylinder.

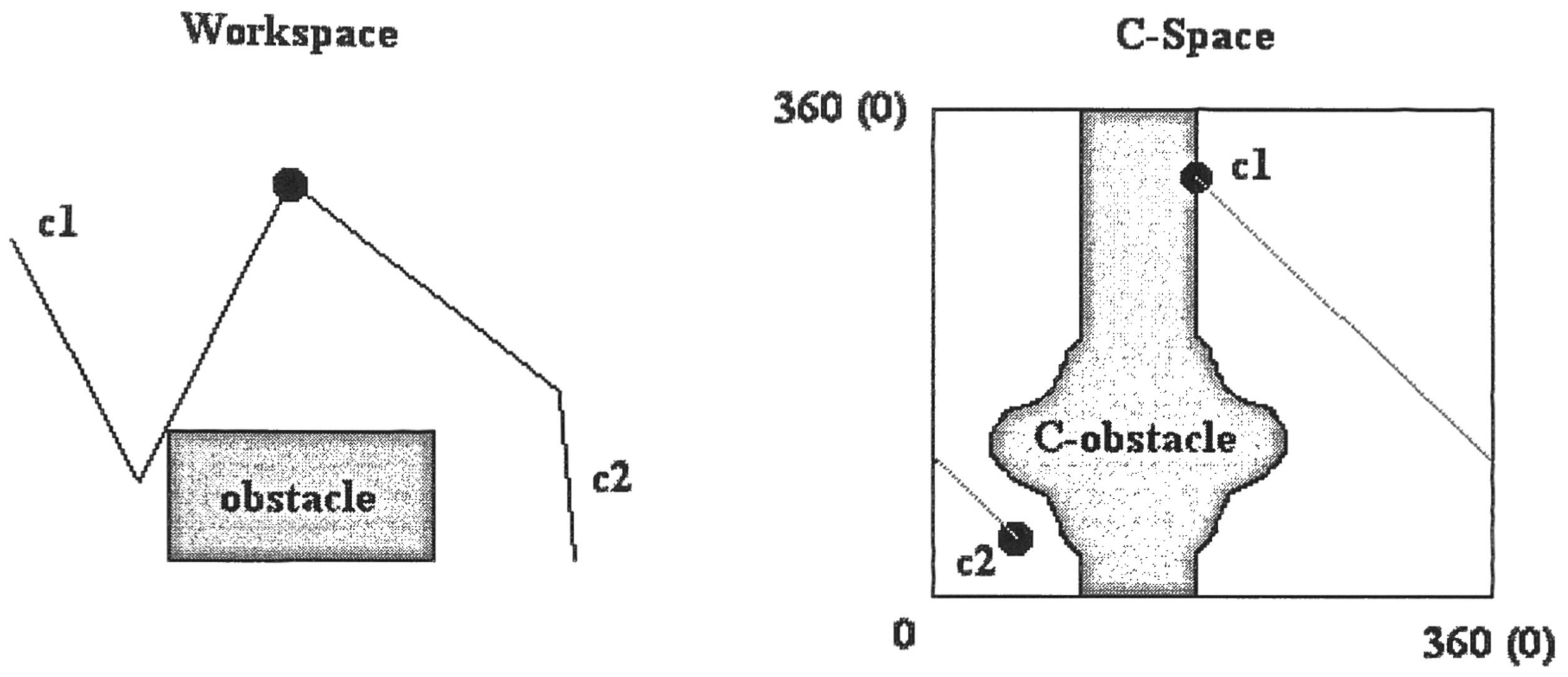

Figure 1: A sample mapping from a Workspace to a C-Space

2.3 Planning Methodologies

There is no general solution to the motion planning problem, however many methods have been developed. Most of these are for specific cases with certain limitations, and can be categorized into several basic approaches to the problem [2,3]: cell decomposition, potential field, mathematical programming, and skeleton or roadmap methods.

The cell decomposition approach breaks the free C-Space, where the robot and obstacles do not overlap, into a set of disjoint cells [2]. Adjacency relationships between cells are found, then a path of cells is found between those containing the start and goal configurations. In an obstacle-dependent (exact) decomposition of the C-Space, boundaries of C-obstacles are used as cell boundaries, resulting in irregularly shaped and sized cells. The set of cells is hard to compute, but it exactly covers the free C-Space. An obstacle-independent (approximate) decomposition uses a grid to partition the cells [2]. Those cells that are not collision-free are further partitioned, and this process continues to a desired resolution of cells. This approach is easier to compute and the cells are simpler with more regular boundaries, but the entire free C-Space is not guaranteed to be covered. Finer global partitioning could be used, but this requires extra time in computation and collision detection. In theory this method can be resolution complete; if the resolution is not limited, then cells may be infinitely partitioned and ultimately approach an exact decomposition of the C-Space.

A potential field method constructs a vector function, called the potential function, such that its value decreases monotonicly toward the goal, where a minimum value exists, but it increases monotonicly as obstacles approach [2]. The robot begins at the starting configuration and moves along a decreasing gradient towards the goal. Planning becomes an iterative process in that the force is computed from the potential function at the current configuration, a small step is taken in the direction indicated by the force, then the process is repeated until the goal configuration is reached. The main difficulty with this method is defining a potential function for which the global minimum is also the only local minimum. The attractive goal force must be global; conic, sphere, and hybrid functions are often used for this. Yet the obstacle repulsive forces must be limited to act only near those obstacles so that local minima do not occur; local functions inversely proportional to the distance between the robot and obstacle are often used.

Mathematical programming methods represent boundaries of obstacles as a system of equations or inequalities. The problem becomes one of mathematical non-linear optimization, finding a curve between the start and goal configurations that minimizes some scalar quantity. Numerical methods are usually required to generate optimal solutions for this technique [1].

In skeleton or roadmap methods the connectivity of the robot's free C-Space is represented by a graph or network. Examples of skeletons include the visibility graph, the Voronoi diagram, Canny's silhouette, and the subgoal network [1, 2]. Connectivity information about the free C-Space is extracted and placed into a graph of one-dimensional curves. Planning is carried out by connecting the start and goal configurations to the network, and then a simple graph search determines if a solution exists. Since representatives of all topologically distinct feasible paths in C-space must be contained in the network for it to be complete, complete methods are not feasible for high-dimensional C-space. Probabilistic methods, however, often generate acceptable results. Below we describe two such methods, PRM [7] (Probabilistic Roadmap Method) and OBPRM [1] (Obstacle-Based Probabilistic Roadmap Method), on which our work is based.

2.3.1 Probabilistic Roadmap Method (PRM)

The Probabilistic Roadmap Method (PRM) proposed by Kavraki, Latombe, Overmars, and Svestka [7] avoids explicitly constructing C-obstacles by merely sampling the C-Space. The two-step method used is a basic subgoal network approach. This strategy provides a small response time after initial preprocessing of the environment. The first step (preprocessing) includes node generation and roadmap connection. A set of roadmap candidate nodes is randomly generated, for example, by using points on a uniform grid over C-Space as in Figure 2. Collision checking is used to determine those points which are in free C-Space, and those points are retained as vertices of the roadmap. Valid points are interconnected to form a roadmap or network using a simple, fast, deterministic local planner, an example being a straight line between points in C-Space that is checked for collision at discrete intervals. Processing on the roadmap, such as analysis of connectivity, also occurs in this stage. After preprocessing, a roadmap is constructed and ready for planning. The second step involves planning queries in which a feasible path between the start and goal configurations is sought. It is carried out as in other roadmap methods. Namely, a local planner connects the start and goal configurations to a connected component of the roadmap if possible, then a graph search finds a path in the roadmap between these two connection points.

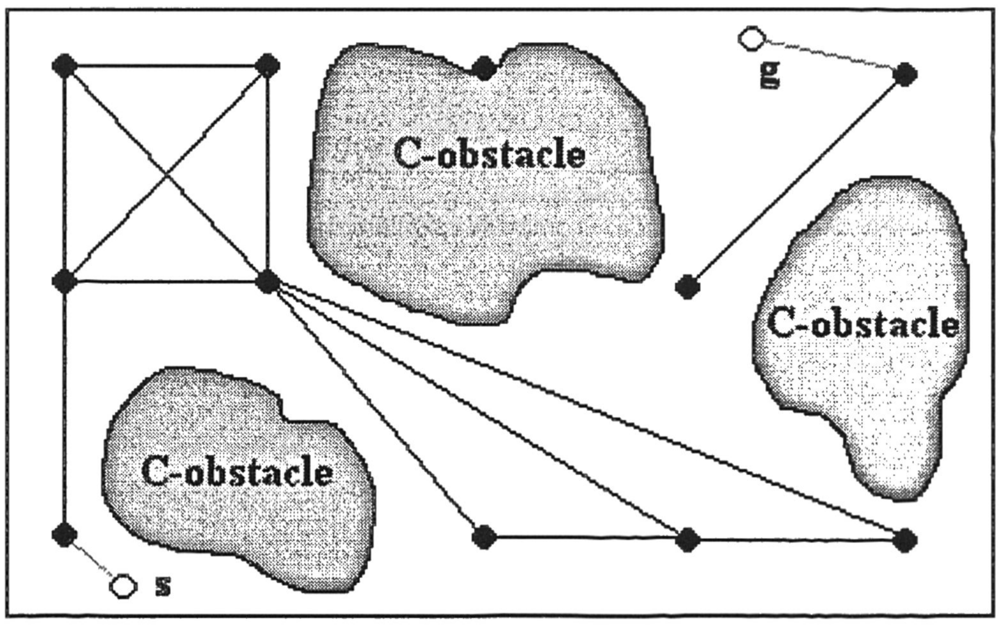

Figure 2: A roadmap in C-Space that might be obtained using PRM

Probabilistic methods, unlike many others, are applicable to high-dimensional C-Space and are practical. They are also probabilistically complete, meaning they would be complete if run for a long enough time (during preprocessing). Running time depends on the application, but preprocessing consumes the majority of the time, ranging from several minutes to many hours.

By adjusting and fine tuning the parameters of the method this time can be altered. On the other hand, it generally takes less than a second to answer a planning query, so it can be seen that the preprocessing step deserves greater attention. This method operates very fast in many situations, but difficulties arise in cluttered C-Spaces and long, narrow passages between C-obstacles, and planning must be done on C-obstacle surfaces for contact tasks.

2.4 Object Based Probabilistic Roadmap Method (OBPRM)

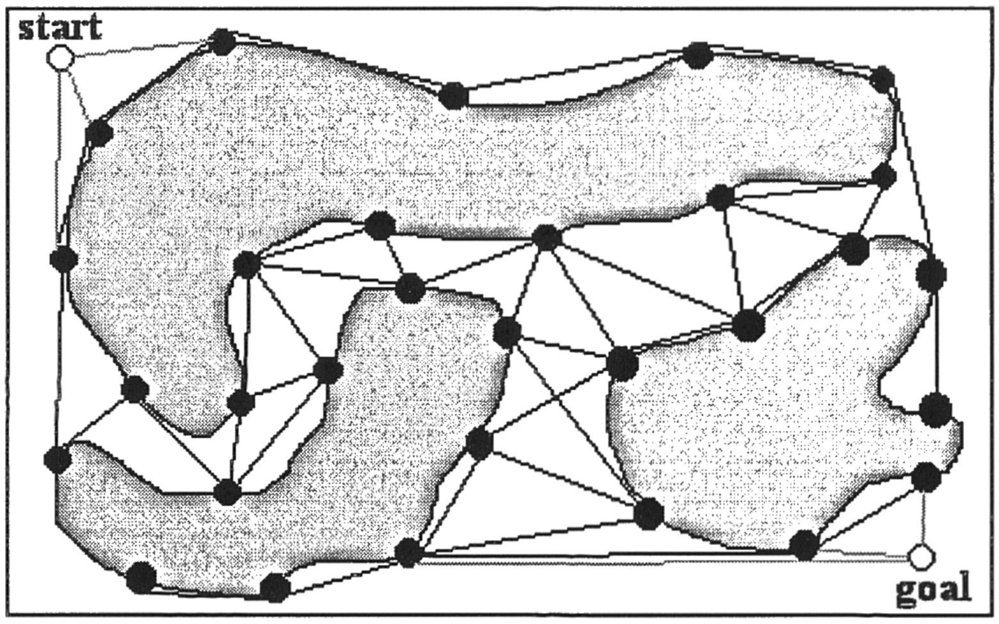

Figure 3: A roadmap in C-Space that might be obtained using OBPRM

In an attempt to overcome those difficulties faced by PRM, a new method was proposed by Amato and Wu [1]. Here follows an overview of OBPRM as developed for motion planning for robots with many dof. The approach follows PRM and traditional roadmap methods. A roadmap (graph) is built in C-space during preprocessing. Each vertex of the graph corresponds to a collision-free configuration of the robot, and vertices are connected if a path between them can be found by a simple, local planner. Planning consists of connecting the initial and goal vertices to the roadmap, then finding a path between these two connection points. The new idea is how roadmap candidate points are generated. As shown in Figure 3, points are randomly distributed on the surface of each obstacle. In this way, high quality roadmaps can be obtained even in a crowded C-Space with long, narrow passages.

The general strategy of the node generation process is to construct a set of candidate nodes for each object such that each point lies on the surface of an obstacle and is not contained in the interior of any other obstacle. The set of roadmap candidate nodes is the union of all nodes computed for each object. For a high quality roadmap, it is desirable to distribute nodes uniformly on the surfaces that delineate the robot's free space; for simplicity, Euclidean distance in C-space is the metric used. For each object a set of points is computed that is randomly distributed on its surface, and then all points internal to any other C-object are discarded. The number of points generated depends on the size and shape of each object, and thus may vary from object to object. Collision detection determines which points must be removed from the set. To avoid computing the constraint surfaces, the following point generation heuristic is used: Determine a point inside an object. Select a number (depending on the size and shape of an object) of rays with origin at that point, and directions randomly distributed. For each ray, use a binary search to locate a point of the ray on the object boundary. This method works best when objects are roughly spherical. Oblong shaped objects and objects which fold over themselves can cause problems, but this problem is being addressed. Figure 4 below demonstrates this point generation scheme.

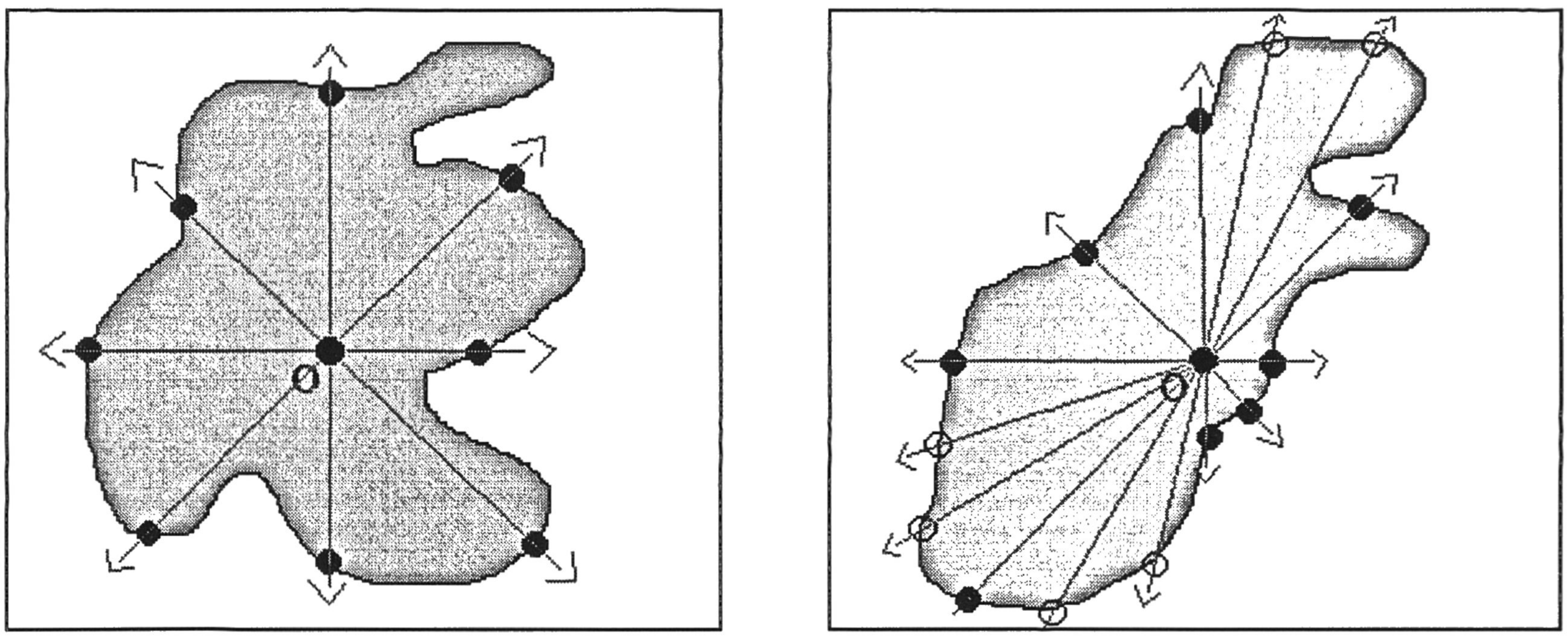

Figure 4: Generating points uniformly on a C-object in two-dimensional C-Space. Note on the oblong object how additional points (non-solid) improve the uniform distribution on its surface.

Now consider how to connect the candidate nodes to create a roadmap. The idea is to use simple, fast, deterministic, local planners to connect pairs of nodes. In trying to connect only those nodes which have a high probability of connection, the planner attempts to connect the closest roadmap nodes. To save space, paths found in this stage are not recorded since they can be re-generated quickly. The method uses two common local planners to find and connect nodes so they are not explained in detail here. It is hoped that the roadmap constructed includes paths through all corridors in C-space. The paths obtained at this point are not smooth, so standard smoothing techniques are applied to the paths. For example, mathematical functions can be used to extract a smooth curve from a straight line path. It is necessary to keep in mind that manipulation planning requires contact between the robot and objects and thus restricts the smoothing techniques that can be applied.

This new randomized roadmap method for motion planning is applicable for both collision-free path planning and manipulation planning of contact tasks. The concept was tested by implementing the method for path planning of various segmented robots in 2-D environments and CAD problems in 3-D. The method was shown to perform well, even in crowded C-space. Additional optimizations and experiments are being planned and tested, including methods by which nodes are generated on surfaces of C-obstacles, use of various local planners, and improving the speed of the planner mainly through a single shot planning method. Our work has concentrated on this last point.

3. Single Shot OBPRM

Computation intensive preprocessing only makes sense if an environment is static, or non-changing, and one expects to perform multiple motion planning queries. Traditional methods and PRMs work well in these instances. In those methods a roadmap is initially generated for an environment, and then many different queries may be performed on the one roadmap. However, for our intended uses for manufacturability and maintainability studies in industrial CAD applications, changes often occur as products are altered and improved, designed incrementally, and components are replaced or repaired. Each change in design corresponds to a new environment and designs are altered too frequently for the preprocessing requirements on each update to be acceptable. Also, multiple queries are seldom performed for industrial purposes. Designs are usually only checked once for feasibility then either implemented or altered. A method by which new environments could be analyzed as quickly as possible is very desirable for this type of implementation. In order to address these weaknesses, there is a need for this new Single Shot OBPRM. By single shot, it is meant that this method is performed only once for a particular start-goal pair, and only enough processing is done, and time consumed, to find a path or determine that one cannot be computed in a reasonable amount of time. There will not be multiple queries upon the same roadmap for multiple start-goal pairs.

The need for the single shot planning method arises from the nature of the two-step planning method: nearly all the execution time of the algorithm is consumed in the preprocessing step. The Single Shot OBPRM eliminates the preprocessing time by only generating enough nodes as is necessary to obtain a path, rather than generating enough nodes to form a well connected roadmap. Since a path is to be found for only one start-goal pair, the path may be found with fewer nodes that are specifically generated for this pair. This new method does not explicitly do any preprocessing as in PRM or OBPRM, but rather actively performs all operations for a start-goal configuration pair on-line.

3.1 Overview of ssOBPRM

As stated above, the single shot planner is intended to only perform enough processing as necessary to find a path between specified start and goal configurations. The driving idea is to find a path as fast as possible with as little computation as possible. Following this strategy, this method begins by attempting to find a straight-line path between the start and goal configurations. In this simple case, the path is very quickly found and returned, and the process is finished. Most environments, however, will have numerous obstacles between the two specified configurations. In this case, ssOBPRM determines whether there are multiple obstacles, or only one, blocking the straight-line path between start and goal. If only one obstacle is in the path, nodes are generated on it, as in OBPRM, and the planner attempts to find a path around it. Points are only generated on this one obstacle in order to keep the number of roadmap points low and to try to find the straightest possible path. If there are multiple obstacles blocking the desired path, the planner will generate roadmap points on those outermost obstacles along the straight path between start and goal. If a direct path can be found between these two outer obstacles, then it is returned, along with the path from start to its closest obstacle and goal to its closest obstacle, and the process is finished. When no direct path is possible between those two outer obstacles, the algorithm will attempt to find a path around each obstacle, then it will recurse upon the space between the two obstacles. This process of recursion continues until the path is completed or else some predefined threshold is reached. A detailed description of this algorithm follows.

3.2 Detailed Description of ssOBPRM

The single shot planner first attempts to find a straight-line path in C-Space between the start (S) and goal (G) configurations of the robot using a simple local planner, and returns the path if one is found. This is an attempt to find a path very quickly in the event that there are no obstacles directly in the path from S to G, which embodies the basic principle of the single shot planner in that a path is to be found quickly with as little processing as possible. The exclusion of preprocessing eliminates the bulk of the time required in the PRM and OBPRM methods.

In the event that the first attempt at a path fails, a more complex local planner is used. This planner also uses the straight-line path between S and G, but it determines where this path intersects the outermost obstacle(s), the obstacle(s) nearest S and G that is also in the path. This process is illustrated in Figure 5. We say obstacle(s) because it is possible for one or more objects to intersect the straight path from S to G. This more complex planner begins an iterative process: beginning at S and G, it finds feasible parts of a path and some new points along the path, and then repeats the process with those new points. The process works inward from S and G. The complex planner determines the points (S' and G') where the straight-line path, between S and G, intersects the outer edges (with respect to S and G) of the obstacles nearest S and G respectively. The path from S to S' and from G to G' is saved, as well as the points S1 and G1. S1 and G1 are also points on the outermost obstacles along the straight-line path from S to G, but on the side of the obstacle opposite S' and G'.

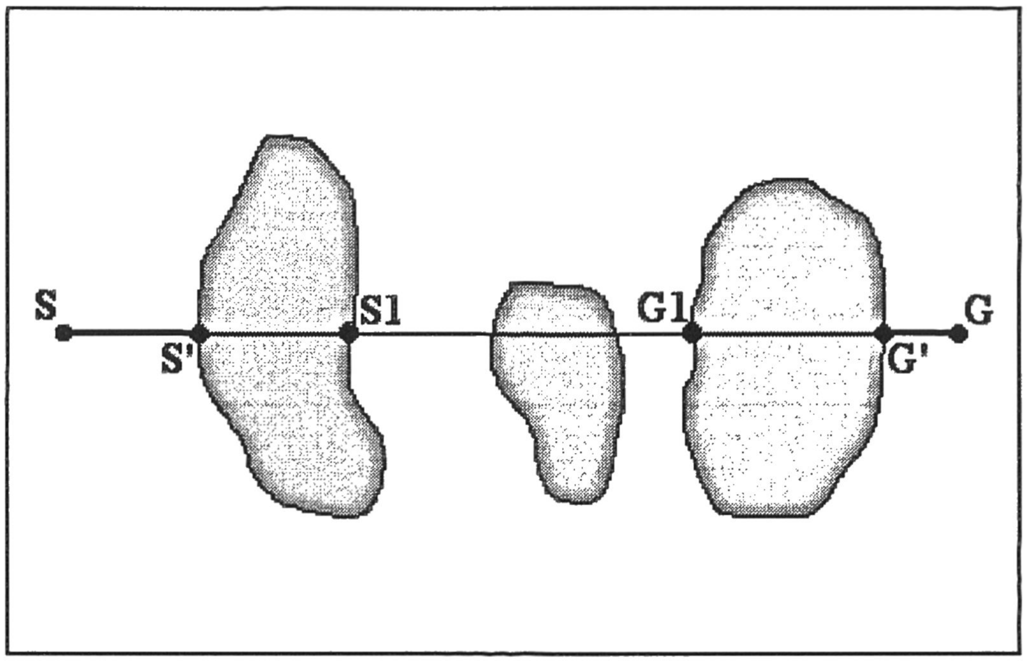

Figure 5: ssOBPRM attempts to connect S and G with a straight-line path in C-Space

3.2.1 Planning on a Single C-Obstacle

At this point it is determined whether or not the points generated are on the same object, indicating only one object between S and G. It is very easy to test this, because S' and G1 will correspond to the same point, as will G' and S1 as shown in Figure 6. In the event that there is only one object between S and G, but this object folds over itself or is shaped such that S' and G1 do not represent the same point, nor G' and S1, then the object is treated as if there were multiple objects intersecting the path. If it is found that there is only one object intersecting the straight-line path between S and G, then nodes are generated on the surface of this obstacle using the same method as in the OBPRM planner. Once a set of nodes on the obstacle is available, a roadmap is constructed and a path is found, as in the PRM, that traverses around the object from S' to G'. If a path is found at this point then it is returned and the query is over. Otherwise a random walk is performed from points S' and G' and the process is attempted again to try to find a path around the object. This random walk process will repeat until a path is found around the object or else a specified amount of time or number of repetitions has passed. If a path is still not found then the algorithm returns failure.

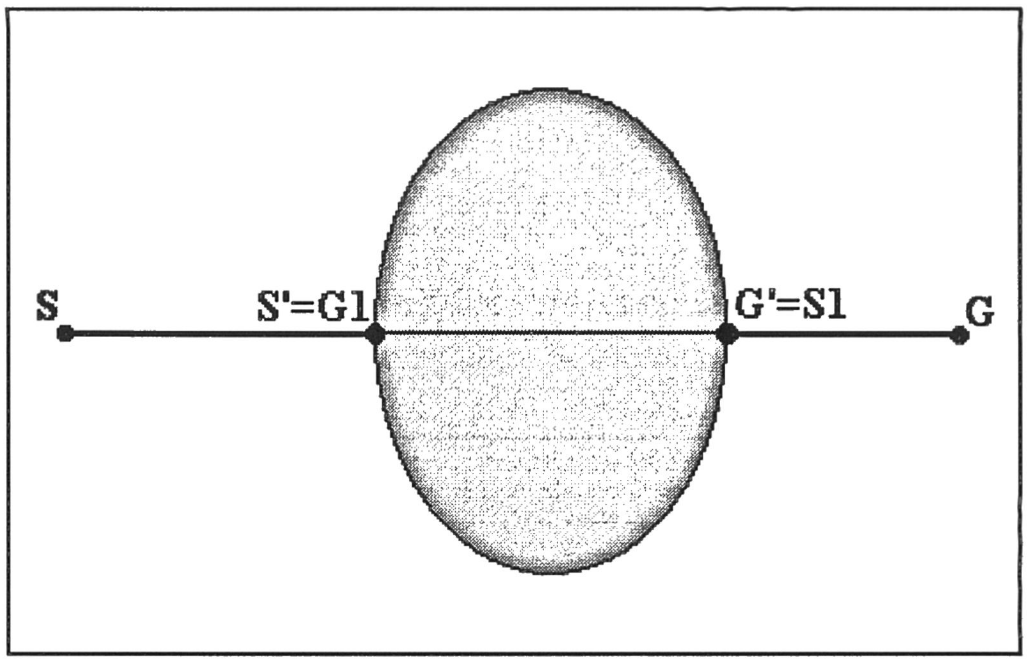

Figure 6: ssOBPRM when only one object lies between S and G

3.2.2 Planning on Multiple C-Obstacles

If it is determined that those points S' and G' are not on the same object, there is more than one object between S and G. This is the case when S' and G1 do not represent the same point, and the same for G' and S1. Figure 7 demonstrates this instance. Two sets of nodes are then generated, one set on the surface of each of the two obstacles, by the same method as the OBPRM planner. Once these two sets of nodes are generated, we attempt to find a path connecting the two, which connects the two objects. This process is similar to a bipartite search in which we try to find any connection between the two sets. Attempting to connect these two objects addresses the possibility that a path may be found that avoids other objects between the two current ones. This attempts to reduce the amount of processing and thus time required for the algorithm. There may be other objects that lie along the straight-line path from S to G, yet are small or shaped such that a path may be found between two outer objects that avoids those internal ones. If a connection is found between the two outer objects, those two points which connected are returned as S" (on the object nearest S) and G" (on the object nearest G), and the path between those two points is saved. In this case the planner attempts to connect S' to S", and G' to G", exactly as on the single object above when connecting S' and G' to move around the obstacle. Upon connection of S' to S" and G' to G", the planning is complete and the path is returned.

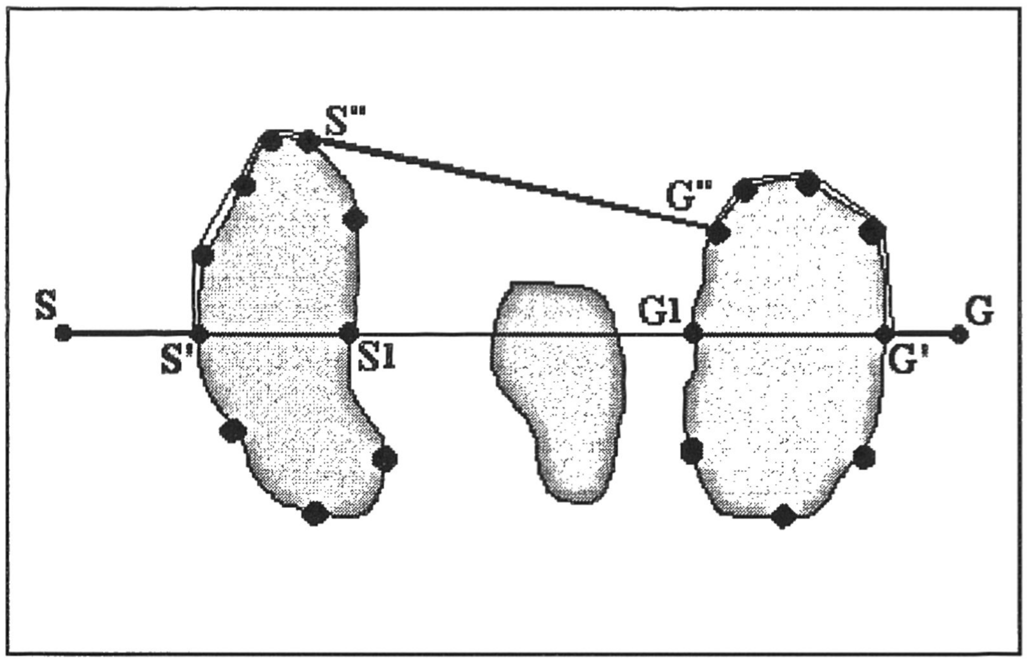

Figure 7: ssOBPRM when multiple obstacles lie between S and G

If it is the case that no connection is found between the two outer obstacles, meaning there is no possible direct connection between the two, then the planner attempts to connect S' to S1, and G' to G1 as on the single object previously. In this case there are objects between the outer objects that are large enough or positioned such that they block any straight-line path between the two as seen in Figure 8. As in the case of the single object, upon failure to connect around either obstacle, a random walk is performed and the planner again tries to move around the object until a path is found or a specified limit of time or repetitions is reached. Provided the planner succeeds in connecting around each obstacle, the paths found from S to S', S' to S1, G to G', and G' to G1 are saved, and the algorithm recurses on itself.

When the algorithm recurses, it now uses the current points S1 and G1 as the next S and G respectively. The algorithm repeats, this time operating on the interior portion of the path. Basically, the method begins with the outermost obstacles, finds a path around them if possible, then recurses on the interior obstacles to find a path around them. If allowed to run long enough a path will eventually be found or else some predefined threshold will be reached and the planner will halt. We use the idea that a straight-line path will be close to, if not, the most efficient path between a start and goal configuration. Also, an attempt to avoid interior obstacles along the path improves the straightness of the path while reducing the processing required to deal with more objects. The result is a new algorithm that builds on PRM and OBPRM to eliminate their shortcomings, especially for industrial and CAD purposes. A pseudocode outline of the algorithm follows, along with brief descriptions of the subroutines used.

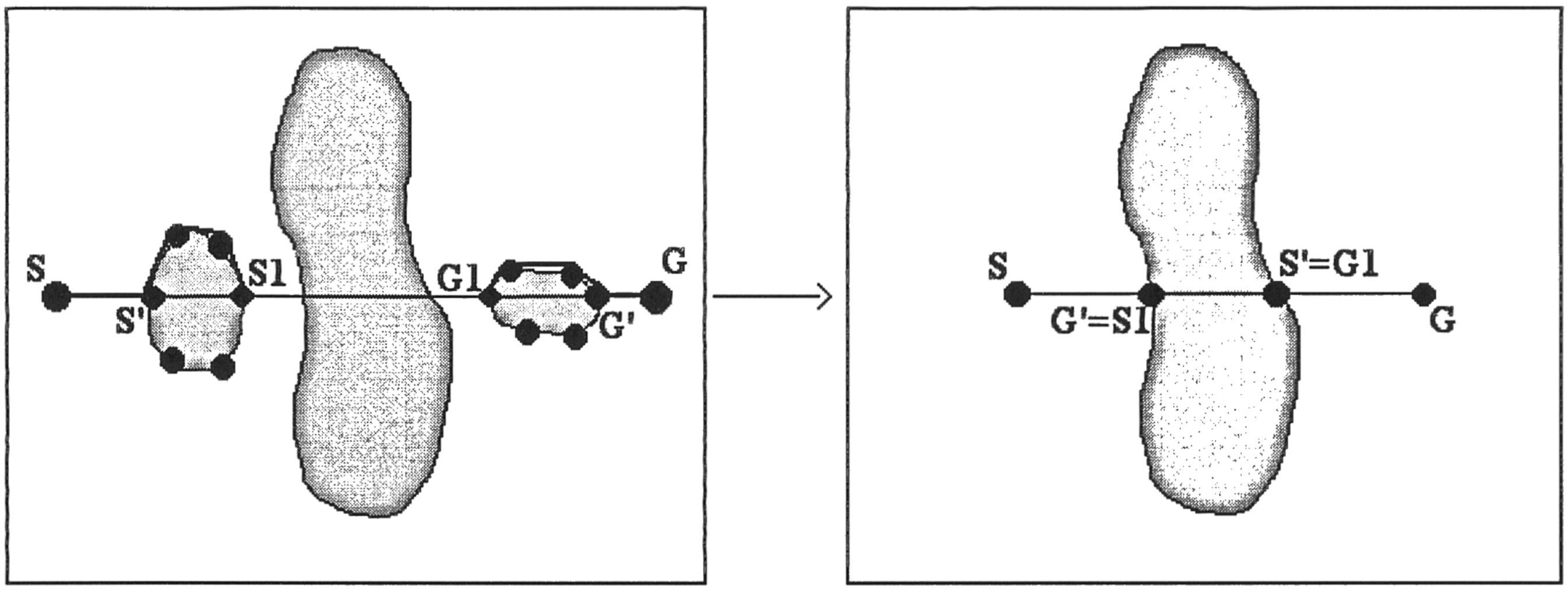

Figure 8: Recursion of ssOBPRM on the interior path

3.3 Pseudo-Code Description of ssOBPRM

```
FindPath  (S, G, path);                /* main routine */


        int success                     /* success = 1 if a path is found from S → G */
Start
                                        /* S1 & G1 = path points on opposite side of obstacle */
   success:=0;                          /* S' & G' = path points on S & G side of obstacle */
                                                /* path is a datatype with path information */
   connected-simple (S, G, success,path);       /* try to find a path from S → G with local
planner*/
   if (not success)                             /* but don't save the path...easily regenerated */
        begin
           connected-complex (S, G, S1, G1, S', G', path); /* returns path info. and points */
           if (S1 = G')                         /* if points are on the same object */
               then
                  node (object-of-S1)                  /* generate points on object */
                  connect1 (S', S1, success, path)     /* connect S' & G' on same object */
                  (if connect fails, perform a random walk and repeat up to specified limit)
                  if not success
                       then return (failure-to-connect)
                       else return (path);             /* return entire path around object */
               else                                    /* points are on different objects */
                  node (object-of-S1)                  /* generate points on object near S */
                  node (object-of-G1)                  /* generate points on object near G */
```

```
                connect2 (S1, G1, success, path)  /* try to connect the 2 objects */
                if not success
                    then  connect1 (S', S1, success, path)          /* get around object */
                       if not success
                           then return (failure-to-connect)
                           else connect1 (G', G1, success, path) /* get around object */
                                  if not success
                                     then return (failure-to-connect)
                                     else FindPath (S1, G1, path);        /* recurse */
                    else connect1 (S', S1, success, path)            /* 2 objects did connect */
                       if not success                 /* get around objects to connection path */
                           then return (failure to connect)
                           else connect1 (G', G1, success, path)
                              if not success
                                  then return (failure-to-connect)
                                  else return (path)        /* path found between 2 objects - */
        end;                                                /* return the entire path */
End.
```

At every connection attempt, failure is returned (path = nil) or success is returned along with the new parts of the path, so that path = path + newpart-of-path (path = path + connect*i*(…..)).

Connected-simple (S, G, success, path)

Input: S and G of type node, they represent the start and goal configurations
Success is a boolean type indicating a path is found or not
Path is a list of nodes making up a path if one is found

Output: Success, true if a path is found
Path, the list of nodes in the path

A dumb local planner tries to find a straight-line path between S & G, and returns success if one is found. Does not return path because path is easily regenerated.

Connected-complex (S, G, S1, G1, S', G', path)

Input: S, G, S1, G1, S', G' of type node, represent various configurations
Path, the list of nodes in the path

Output: Returns S..G', necessary path points to find a path

A more complex local planner returns part of path from S & G to first intersected obstacles from each point. It also returns in S'/G' the points where the straight-line path intersects the obstacles on the side of S/G, and in S1/G1 the points where the straight-line path intersects the obstacles on the side opposite of S/G.

Node (object)

Input: Object, a pointer to an object

Output: A list of nodes generated on the surface of the object

Generates points on the object surfaces to be used for connection points.

Connect1 (S', S1, success, path)

Input: S' and S1, two configurations
Success, variable that indicates if path is found
Path, list to return a path in

Output: Success, true if S' and S1 are connected
Path, the list of nodes in the path connecting S' and S1

Tries to connect the points S' & S1 on the same object using the points generated. It's just trying to connect two points using some generated nodes. If they connect, success = true and the path is returned, otherwise success = false and the returned path = nil.

Connect2 (S1, G1, success, path)

Input: S1, G1, two robot configurations
Success, variable that indicates if path is found
path, list to return a path in

Ouput: Success, true if S1 and G1 are connected
Path, the list of nodes in the path connecting S1 and G1

Tries to connect two objects using points generated on the surfaces of each object, it is a bipartite search where one set consists of the points on one object, and the other set consists of those points on the other object. If a possible connection is found, S1 and G1 return the endpoints of the path connecting the two objects so that the planner can try to connect this portion with the rest of the path. If no connection is found, then the original S1 and G1 are returned unchanged so the planner can try to get around each object and recurse on FindPath.

4. Implementation and Experimental Results

This algorithm will be further developed, details added, new subprograms developed and existing ones altered as needed, and eventually its performance tested. This will involve updating and re-testing of the algorithm as it is discovered which methods work best. New or altered subprograms will be tested and developed also, for example different node generation techniques and local planners within the planning algorithm. Our implementation and experimental results will be discussed in detail in the final version of this thesis.

5. Conclusion and Future Research

Implementation of ssOBPRM will identify future problems and guide future directions of this project. We believe our experimental results will show ssOBPRM to be a faster and more efficient planning method that may be implemented in industry and any other use of robotic motion planning.

Future improvements envisioned at this stage include, but are not limited to, the use of fast algorithms to compute the convex hull (cone) of the outermost objects in the single shot method (those closest to S and G), and using those points to try to connect the outer objects. Again, this is another attempt to find a better path quicker by avoiding possibly smaller inner objects that may intersect the straight-line path from S to G, but do not intersect a path between the convex hulls of the outermost objects. Additionally, points may be generated just off the surfaces of objects in order to use more free space in trying to find a better path. This will work fine for motion planning, but manipulation tasks require contact between robots and obstacles and therefore points for manipulation must be generated on object surfaces.

6. Acknowledgments

Our general research goal is to develop an automated method for analyzing product designs for manufacturability and maintainabililty. This work is being carried out with the help of Dr. Nancy Amato (Department of Computer Science, Texas A&M University) and her students. We plan to implement this new method within Product Vision, a CAD package developed by researchers at GE's Corporate Research Center. This project has benefited from collaboration with GE. In particular, Dr. Hsuan Chang, formally of GE, has provided general guidance and a strong base for our work. Additionally, GE will provide tested applications (e.g., assembly maintainability studies) for our method.

References

[1] Nancy Amato and Yan Wu, A Randomized Roadmap Method for Path and Manipulation Planning. *Proceedings of 1996 IEEE International Conference on Robotics and Automation* (ICRA), pages 802-827, 1996.

[2] Jean-Claude Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[3] Hsuan Chang and Tsai-Yen Li, Assembly Maintainability Study with Motion Planning. *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, pages 1012-1019, 1995.

[4] Yong K. Hwang and Narendra Ahuja, Gross Motion Planning-A Survey. *ACM Computing Surveys*, pages 219-291, Vol. 24, No.3, Sept. 1992.

[5] D.H. Joseph and W.H. Plantiga, On the Complexity of Reachability and Motion Planning Questions. *Proceedings of the ACM Symposium on Computational Geometry*, pages 62-66, 1985.

[6] J. Reif, Complexity of the Mover's Problem and Generalizations. *Proceedings of the 20th Annual Symposium on the Foundations of Computer Science*, pages 421-427, 1979.

[7] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, pages 566-580, 1996.

[8] Sara Baase, *Computer Algorithms*, Addison Wesley Publishing, 1988.