Planning Smooth Paths for Mobile Robots

Paul Jacobs and **John Canny***
Department of Electrical Engineering and Computer Science and
The Electronics Research Laboratory
University of California
Berkeley, CA 94720

Abstract

A mobile robot which is built on a wheeled platform is unable to follow an arbitrary path. In this paper, we present an algorithm for planning paths for a mobile robot subject to the nonholonomic constraints imposed. Our technique is to define a set of canonical trajectories which satisfy the constraints. A configuration space can be constructed for these trajectories in which there is a simple characterization of the boundaries of the obstacles generated by the workspace obstacles.

We describe a graph search algorithm which divides the configuration space into sample trajectories. The characterization of the boundaries enables us to calculate an approximate path in time $\mathcal{O}(\frac{n^2}{\delta}\log n + A\log(\frac{n}{\delta}))$, where n is the number of obstacle vertices in the environment, A is the number of free trajectories, and δ describes the robustness of the generated path and the closeness of the approximation. We also describe a plane sweep for computing the configuration space obstacle for a trajectory segment. We use this to generate robust paths using a quadtree based algorithm in time $\mathcal{O}(n^4\log n + \frac{n^2}{\delta^2})$.

1 Introduction

Research in path planning for robots has traditionally been focused on finding ways to ensure that a fixed arm will not collide with objects in its workspace as it performs its tasks. However, to be truly flexible, manipulators will someday be mounted upon mobile platforms, giving them not just the ability to manipulate objects in a fixed region, but to interact with their environment in more complex ways, much as humans do.

The reigning paradigm of path planning has been the piano movers problem, which concerns moving an object through a set of obstacles without any constraints on the type of movements which can be made. But mobile robots are likely to be based upon carts with wheels for steering and locomotion. This will introduce restrictions on the way they will be able to change directions. A wheeled robot must be able to account for the motion it undergoes as turns around corners. It is unrealistic to plan paths for mobile robots which force it to change direction instantaneously. Even for those robots which have a capability to turn on a spot, the dynamics of this task will force them to decelerate sharply and stop at each turn.

In this paper, we consider the problem of planning paths for a robot which has a minimum turning radius. This is a first step towards accurately modeling a robot with the kinematics of a car. The approach we have taken is to define a set of trajectories which satisfy the constraints and to show that considering only these trajectories is sufficient to find a path if one exists.

*This research was supported by the Defense Advanced Research Projects Agency (DoD), monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292.

1.1 Statement of the Problem

The problem we consider in this paper is to find a path of minimal length, if one exists, which satisfies the following conditions.

Conditions *Given IP, FP, U, V* $\in \mathbf{R}^2$ *with* $\|U\| = \|V\| = 1$. *Let* $R > 0$. *Let* $\Omega \subset \mathbf{R}^2$ *be a (closed) set of polygons. Let C=C(n,IP,U,FP,V,R,*Ω*) be the collection of all curves X defined on the interval [0,L], where L=L(X) varies with X, such that* $X(s) \in \mathbf{R}^2 \setminus int(\Omega)$, $0 \le s \le L$; $\|X'\| = 1$; $\|X'(s_1) - X'(s_2)\| \le R^{-1}\|s_1 - s_2\|\ \forall s_1, s_2 \in [0, L(X)]$; $X(0) = IP$, $X'(0) = U$, $X(L) = FP$, $X'(L) = V$.

We have used int(Ω) to denote the interior of the set Ω. This general formulation of the problem is due to Dubins [3]. Theorem 1 (section 5) asserts that the problem is well-posed. Henceforth, we refer to the angles associated with U and V as θ_i and θ_f, respectively.

1.2 Previous Work in the Field

Planning motion subject to non-holonomic constraints is a relatively new area of research. The particular problem addressed in this paper has been examined in [1,2]. In Laumond [1], the environments consist of closed curves which are not necessarily polygons. The solution presented there is not guaranteed to find a path, however. In Fortune and Wilfong, [2], the question is to decide if a path exists under given conditions. The algorithm is exact, but does not generate the path in question. This algorithm runs in time and space which is exponential ($2^{O(poly(m,n))}$), where n is the number of corners in the environment, m is the number of bits used to specify the positions of the corners, and poly(m,n) is a polynomial in n and m.

In Dubins [3], the minimal path problem is considered, but without obstacles. We discuss the extension to this result as it applies to the problem of planning collision free trajectories.

2 Preliminary Discussion

2.1 Robustness of Paths

We can find a path approximating a minimal length path in the case that is sufficiently robust that it allows for small changes in orientations at points along the path without causing collisions. This is a reasonable constraint since practical mobile robot systems are not able to follow a planned trajectory perfectly. There is an explicit tradeoff between the approximate path found and the minimal length robust path in terms of the desired robustness.

2.2 The Canonical Trajectories

We will show in section 5 that it is enough to consider paths which consist of a circular arc segment of maximum curvature followed

by a straight line segment followed by another such arc segment where the initial arc segment begins at an obstacle vertex or an obstacle edge or IP, and the final arc segment ends at an obstacle vertex or edge or FP, or a trajectory consisting of three such arc segments. Thus we can reduce the problem to separately finding trajectories which can pass from point or edge to point or edge, then pasting them together to form the complete trajectory from (IP,θ_i) to (FP,θ_f).

For a given location and orientation, there are two oriented circles of maximum curvature along which the robot can effect a change in orientation. One corresponds to a left hand turn, the other a right hand turn. For a path which consists of an arc followed by a straight line motion followed by an arc, there are then four possible combinations of turns at the two endpoints. We denote these paths as LL,LR,RL,RR, where the L and R denote left and right hand turns, and the ordering denotes the end of the path at which the turn is made. In the case that the trajectory is of the three arc type, there are only two possibilities, LRL or RLR. To resolve ambiguities, we require that the middle arc of the three arc type path have length $\geq \pi R$, where R is the minimum radius of curvature. Thus there are six types of paths. These we will call the *turn types*, and denote them $\mathcal{T} = \{LL, LR, RL, RR, LRL, RLR\}$. There are also four possibilities for the type of contact that the trajectory has with obstacles at the start and finish of the path segment. It can start and finish either at an edge or a vertex of an obstacle. The trajectory to be followed is completely determined by the initial and final orientations, turn type, initial and final locations, and maximum curvature of the path.

We can parameterize all the paths of the allowed types which pass between two locations in the environment using two parameters for each of the turn types. If a trajectory contacts an obstacle edge, then the curve must be tangent to that edge, which fixes the robot's orientation, but allows the initial (or final) position to be anywhere along that edge. If a robot is situated at a point, then its location is fixed, but its orientation is not. Therefore, for every motion of a given turn type between pairs of edges or vertices, there is one parameter each which specifies the initial and final circular arcs traversed.

2.3 Geometry of the Problem

The solution which we have devised depends strongly on the relative simplicity of the geometry of the circle and the straight line. In order to determine the proper tangents and directions of travel along the trajectories the only geometrical operations needed are to solve for the intersections between circles, lines and line segments. These can all be done in closed form.

The circles which contribute the arc portions of the trajectory are called *t-circles*. The straight-line portion of the path is called the *t-line*. In some cases, we will use this term to refer to the infinite line which supports the t-line.

It is conceptually easier to solve for the centers of the t-circles instead of computing the tangents directly. This technique brings with it another set of lines and circles, which are related to the t-circles and t-lines. For a given fixed point, p, in the plane, the locus of points defined by the centers of all the t-circles passing through p is also a circle, which we will call the *circle of curvature center, or CCC*. Associated with the t-line are two lines parallel to it at a distance equal to the fixed radius of curvature. The lines are called the *lines of curvature center or LCC's*. There are two LCC's to account for the direction of the turn on each end of the t-line, whether right hand or left hand. In figure 1, the CCC's are shown as dashed circles. The endpoints of the trajectories are at the centers of the CCC's. The t-line is shown

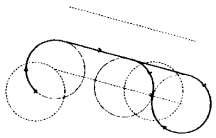

Figure 1: Encoding trajectories by intersections

as a thick line and the associated LCC's are indicated by the parallel dashed lines. We see how the intersections of the CCC's of the trajectory endpoints with the LCC's completely describe the possible trajectories which move along the same t-line. Two of the four trajectories which pass along the t-line are shown.

We will often look for the centers of circles which either lie tangent to an obstacle edge or pass through its endpoints. For a fixed radius R of the circles, we define the *tube of radius R about an edge* to be the locus of all centers of circles of radius R satisfying these constraints. See figure 2 for a picture of the tube.

3 Constraints in Configuration Space

3.1 Mapping the Obstacles to C-Space

Because we can parametrize the canonical trajectories in a unique fashion, we can construct a configuration space for each pairing of vertex and edges and turn type. For example, for a vertex to vertex path, the configuration space consists of the product of the sets of initial and final orientations with the set of turn types, $S^1 \times S^1 \times \mathcal{T}$. The other cases are analogous, where the point along an edge is parameterized by a normalized distance along that edge. The obstacles in the work space can then be mapped to the configuration space. If the trajectory represented by a point in C-space enters the interior of an obstacle, then the point is in the C-space obstacle. The complete configuration space for the environment consists of the product of the configuration spaces for all pairings of obstacle vertices, edges, IP and FP.

In this section, we discuss the techniques for computing the configuration space obstacles. We define curves which cut out regions in the configuration space with the following properties.

1. Any trajectory can be continuously deformed into any other trajectory in the same region.
2. If any trajectory in a region intersects with an obstacle edge, all trajectories do.

We define the number of intersections for a region to be the number of intersections along any trajectory in the region. This number is well-defined for each region where the trajectories exist. We discuss below the regions in which the trajectory does not exist. A region is part of the configuration space obstacle if the associated number of intersections is non-zero.

There are exactly four types of constraints which can form the boundary of a configuration space obstacle for a curve of type LL,RR,LR, or RL:

Type A These constraints are associated with the positions of the t-circle for which it either passes through the endpoint of a line segment or lies tangent to it. Notice that the angles are given by the intersection of the CCC with the tube of radius R around the obstacle edge, as shown in figure 2.

Type B These constraints account for the fact that the deformation of the curves is not continuous as the initial and final angles are varied. The discontinuity occurs as the tangent point to the t-circle crosses the point where the robot is located (which indicates that the robot could proceed straight

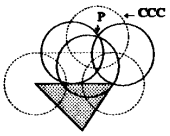

Figure 2: Type A constraints

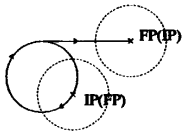

Figure 3: Type B constraints

ahead in that direction to or from the endpoint generating the constraint).

Type C This constraint defines the trajectories for which the t-line passes through a particular vertex of an obstacle polygon between the points where it is tangent to the t-circles through IP and FP.

Type D For all of the turn types except LL and RR, there are conditions under which no path of the type will exist. For LR and RL paths, this happens if the t-circles overlap. The boundary condition occurs when the t-circles are just touching.

A superset of these curves is described in Fortune[2]. For a curve of type LRL or RLR, there is no straight-line path, and hence no Type C constraint. There is an additional constraint which we will call **Type C′**. It corresponds to the curve in configuration space generated by the pairings of the angles for which the middle arc of the three arc path goes through an edge or vertex. See figure 5. No trajectory of these two types exists when the t-circles are separated by more than 4R. The associated constraint is analogous to type D. To distinguish the two, we call it **Type D′**.

The constraint curves and their method of generation are similar for an edge or vertex contact. Here we discuss a vertex to vertex trajectory. See figure 3.1 for an example. The type A constraints have to do only with the t-circle at one end of the trajectory, and thus they hold for a fixed angle on one axis and all angles on the other. The type A constraints indicate when it is possible for the trajectory to intersect an obstacle edge along a circular arc segment and when it is possible for the number of intersections along an edge to change. As a type B boundary is crossed,the path traveled along the corresponding t-circle goes from being the entire perimeter of the circle to being a very small portion of it (or vice-versa depending upon the direction that the tangent moved across the point). Thus, if the t-circle intersects an obstacle, on one side of the constraint the trajectory will pass through the portion of the t-circle which intersects the obstacle edge; on the other side it may not. The type C constraints correspond to the robot being able to pass by the edge of an obstacle. Changing either the initial angle or the final angle in one direction or the other causes the t-line to pass through the polygon or pass completely clear of it. The type D boundary cuts C-space into two regions, one in which the trajectories exist, and one in which they do not.

It is useful for the plane sweep algorithm described later in this paper to define a number of intersections even for the portions of the configuration space corresponding to unrealisable trajectories. The technique we use is to define a *modified trajectory*

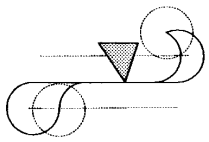

Figure 4: Type C constraints

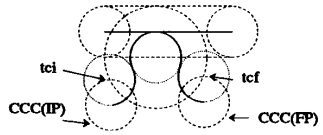

Figure 5: Type C′ constraints

consisting of two parts: the path traversing the initial t-circle from the starting point until it enters the final t-circle, and the path traversing the final t-circle from the point it exits the initial t-circle until it reaches the final point of the trajectory. Strictly speaking, it is not a trajectory since it does not connect the two endpoints of the path, however it goes to the true trajectory at the two extreme cases where the two t-circles have rotated such that they just touch. In order to cut out regions for which the number of intersections is invariant, we define some additional constraints.

4 Search Algorithms

4.1 A Grid Based Approach

In order to find an approximate path through the obstacles without regard to robustness of the planned path, a grid-based method is fast and easily implemented. It is possible to check each potential trajectory for collisions, since only a finite number of possible paths are considered. We present a faster approach which relies on the characterization of configuration space presented earlier.

The algorithm is based upon a graph search such as Dijkstra's algorithm [5, pp. 203 – 208]. Each graph node represents a fixed orientation and position, or *pose*. The orientations are uniformly spaced by δ for those poses which represent the robot lying at an obstacle vertex. The positions are uniformly spaced by $\epsilon(\delta)$ for those poses along obstacle edges. Graph nodes are adjacent if there exists a collision free trajectory which starts at the first position and orientation and connects to the second. The cost associated with each link is the length of the corresponding path.

We find all of the neighbors of a given node by examining the configuration spaces for each pairing of the associated initial position with one of the set of possible final positions. The set of possible final orientations in the configuration space represents a line, called the *sweepline*. We utilize a sweep algorithm to generate the final poses of the robot which can be reached along a collision-free trajectory. We present the case in which both the initial and final locations of the robot are obstacle vertices.

4.1.1 Link Generation Algorithm

1. Set current final orientation to some fixed initial value.
2. Determine the number of intersections of the trajectory connecting the initial pose with the current final pose. Calculate the number of intersections of the initial and final t-circles with the obstacles as well.
3. Find the intersection of all of the constraint curves with the straight line representing all possible final orientations of the

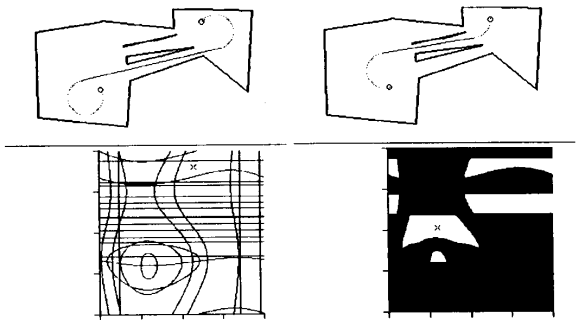

Figure 6: The two figures above show the configuration space generated by the obstacles as shown in the upper half of the picture. The initial and final endpoints of the path segment are shown as small circles. Two sample trajectories are shown. The figure on the left shows the constraint arcs mapped into configuration space. The figure on the right shows the configuration space obstacle.

robot for the given initial orientation. Sort them in increasing order starting from the current final orientation.

4. If the number of intersections of the trajectory with the obstacles is zero, report the current final pose as a neighbor.
5. Increment the current final orientation by δ. If we have examined all of the final orientations, stop.
6. If we have crossed any constraints, update the count of intersections for each one.
7. Go to 4.

4.1.2 Complexity of the Algorithm

For complexity analysis, we need to know the number of constraint curves which the sweepline can intersect. By examining each fixed t-circle to determine the number of possible ways the constraint types can be satisfied, we can show that there are at most $\mathcal{O}(n)$ crossings, where n is the number of obstacle edges in the environment. Using this we can calculate the time required for each step of the algorithm.

1. $\mathcal{O}(1)$.
2. Check for intersection of a trajectory with each obstacle edge. $\mathcal{O}(n)$.
3. There are $\mathcal{O}(n)$ intersections of the constraint curves with the vertical sweep line. To sort them takes time $\mathcal{O}(n \log n)$.
4. $\mathcal{O}(1)$.
5. $\mathcal{O}(1)$. We perform this step $\mathcal{O}(\frac{1}{\delta})$ times.
6. $\mathcal{O}(1)$ per constraint. We cross at most $\mathcal{O}(n)$ constraints during the entire algorithm.

In order to generate the neighbors for a given node requires performing this algorithm $\mathcal{O}(n)$ times, once for each possible final position. Hence the complexity of finding the neighbors of a node is $\mathcal{O}(n^2 \log n + \frac{n}{\delta})$. The brute force method of checking every trajectory individually would take $\mathcal{O}(\frac{n^2}{\delta})$.

Complexity of the entire Dijkstra's algorithm is then $\mathcal{O}(\frac{n^2}{\delta^2} + \frac{n^3}{\delta} \log n)$. This is using an implementation which normally runs in time $\mathcal{O}(n^2)$ (n = number of nodes) without having to generate the neighbors.

The link generating algorithm can be modified to check only free regions for links, yielding a running time $\mathcal{O}(n^2 \log n + a)$ where a is the number of actual neighbors of the given node. The overall running time is then $\mathcal{O}(\frac{n^3}{\delta} \log n + A \log(\frac{n}{\delta}))$, where A is the total number of actual links in the graph.

4.2 A Quadtree Based Algorithm

Using the description of the constraint curves in the configuration space, we can perform a plane sweep to generate the obstacle in configuration space. By partitioning the free regions into quadtree cells we can perform a graph search in which the nodes represent ranges of orientations at the various endpoints. Each free quadtree cell represents the cartesian product of a range of orientations at each endpoint of the trajectory segment, where all of the final orientations can be reached from any of the initial orientations. This guarantees that we find a path which is robust to orientation perturbations, which was not done in the previous algorithm.

4.2.1 Constructing the C-space Obstacle

In order to determine the C-space obstacle it is necessary to find which parts of the constraint curves actually form its boundaries. By performing a plane sweep the constraints which do not contribute to the boundary of the C-space obstacle can be eliminated. It is a standard technique in computational geometry to determine the regions formed by an arrangement of lines. See Mehlhorn[4, pp. 147 – 160] for details of a plane sweep algorithm which works on straight lines. It can be easily extended to the case of arbitrary curves provided that we can generate the points in C-space which mark the "horizontal" extent of a region. Fortunately, these *critical points* of the constraint curves in C-space have geometric significance which can be exploited. During the plane sweep we keep a tally of the number of intersections with the obstacles for each region intersected by the sweepline. Because each critical point is associated with a small number of edges in the environment, we can compute the changes in the number of intersections between adjacent regions in constant time.

4.2.2 Complexity of the Algorithm

By enumerating the critical points, we can show that there are $\mathcal{O}(n^2)$ of them. Because we must sort them, the plane sweep algorithm takes $\mathcal{O}(n^2 \log n)$, where n is the number of corners in the environment, for each piece of configuration space. Generation of the $\mathcal{O}(n^2)$ pieces of configuration space then takes time $\mathcal{O}(n^4 \log n)$.

The path represented by the center of each free quadtree block is analogous to a grid point of the grid-based algorithm. We can again perform a search using Dijkstra's algorithm. Since there are at most $\mathcal{O}(\frac{n^2}{\delta^2})$ quadtree blocks, this is the complexity of the search once the blocks have been generated by the plane sweep. Therefore, the search requires time $\mathcal{O}(n^4 \log n + \frac{n^2}{\delta^2})$.

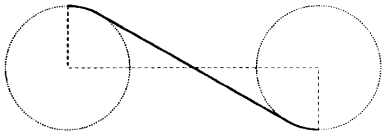

Figure 7: The path used to bound the length of an RL or LR path is the dark line. The two dashed circles are the t-circles.

4.3 Closeness of the Approximation

By assumption, we will always have a free grid point with initial and final orientation within δ of the shortest robust path. In order to determine the amount by which the approximate path is stretched with respect to the shortest path, we need only to consider the difference in path length for a path which differs from another by δ in one orientation.

We assume that there is a continuous homotopy between the two paths parameterized by θ_i, *wlog*. Therefore we need not consider multiples of 2π in the orientations. We assume that $\delta \in [0, \pi]$. The minimum radius of curvature is given by R. The initial position and final position of the path segment are given by IP and FP, respectively. The center of the initial t-circle is denoted *tci* and the final *tcf*. For example:

$$tci(\theta_i) = IP + R\begin{pmatrix} cos\theta_i \\ sin\theta_i \end{pmatrix} \tag{1}$$

First we bound the difference

$$\|tci(\theta_i + \delta) - tci(\theta_i)\| = R\sqrt{2(1 - cos\delta)} \tag{2}$$

using the law of cosines. Furthermore, $1 - cos\theta$ is monotone increasing over $[0, \pi]$, so equation 2 gives the maximum over the range $[\theta_i, \theta_i + \delta]$.

Using this estimate, we can bound the change in the distance traveled. As an example of these calculations, we present the result for LR and RL turns between two vertices. The other cases are similar.

4.3.1 LR and RL Turns between Vertices

The form of the length of the path in this case is complicated. In order to simplify the calculations, we include a portion of the path which is along the t-circles, in determining the length of the t-line. The path is shown in figure 7. The arclength, p, of the path is related to the distance $\|tci(\theta_i) - tcf(\theta_f)\|$, which we denote d, by

$$\frac{p}{d} = \frac{2R}{d} sin^{-1}(\frac{2R}{d}) + \sqrt{1 - \frac{4R^2}{d^2}} \tag{3}$$

Since $d \geq 2R$, we have that this function is monotonically decreasing for $\frac{d}{2R} \geq 1$. This can be seen by taking the derivative of this function (where we change variables by using $x = \frac{d}{2R}$).

$$\frac{d}{dx}\left(\frac{1}{x} sin^{-1}(\frac{1}{x}) + \sqrt{1 - \frac{1}{x^2}}\right) = -x^2 sin^{-1}(\frac{1}{x}) \tag{4}$$

Since $x \geq 1$, $\pi \geq sin^{-1}(\frac{1}{x}) > 0$, since we are taking the principal value of the arcsin. Hence the derivative is strictly negative, implying that the function takes its maximum at $d = 2R$ where it equals $\frac{\pi}{2}$. Now as the t-circle rotates the path goes from different points along the t-circle. However, we can bound the change of the locations of the endpoints of this path with respect to any fixed point on the non-rotating t-circle and the IP for the rotating t-circle. The angle at which the line perpendicular to the line joining the two t-centers can change by is at most δ. This is because the tci(θ_i) and tci($\theta_i + \delta$) must be at least 2R from tcf(θ_f). Along the rotating t-circle the angle to IP also changes by an additional δ. Together, these yield:

$$\Delta p \leq 3R\delta + \frac{\pi}{2}\sqrt{2(1 - cos\delta)} \tag{5}$$

4.3.2 Deriving a Multiplicative Bound

It is necessary to express the change in path length as a stretching factor in order to get a bound which is independent of the length of the minimum path. The bound on the change in path length derived for LR and RL is largest and therefore we divide that by the length of the shortest path segment which can be part of a globally minimum length path. A path of type LRL or RLR is at least πR in length, if it is a minimal path [3, p. 513]. For paths which travel between two separate vertices, the minimum length path is a straight line connecting the two. For a path which goes to the same vertex, the minimum length path is at least $2\pi R$ long. For a path between two non-intersecting edges, the path length must be larger than the minimum distance separating them. This leaves only the length of a path which passes between two edges which are adjacent. In the case that the corner is not convex, robustness of the path indicates that the path can be pulled free of the wall. This will minimize its length. Thus this case will not occur as part of a minimizing robust trajectory. In the case that the corner is convex, the path must travel at least πR in distance. The minimum length of a path segment, p_{min}, can be given.

$$p_{min} = min\left(\pi R, \min_{e_i \cap e_j = \emptyset} \text{dist}(e_i, e_j)\right) \tag{6}$$

where in equation 6 the second term bounds the distance between any two object edges, e_i, e_j which don't share a vertex. Then we can finally bound the stretch which occurs by the generated path having orientation off by at most δ from the global minimum path (with length $\hat{p}$).

$$p_{generated} \leq \left(1 + 2\frac{3R\delta + \frac{\pi}{2}\sqrt{2(1 - cos\delta)}}{p_{min}}\right)\hat{p} \tag{7}$$

We are guaranteed that this path segment, or one closer to the global minimum will be chosen by the action of Dijkstra's algorithm. Thus the algorithm will generate a path whose length differs from an optimal path by this same factor.

5 Proof of Completeness

Theorem 1 ([3]) *Given the conditions of section 1.1. if C contains an X with $L(X) \leq \infty$, then there exists an $X \in C$ of minimal length.*

A remark on theorem 1. The restriction of the curves to lie outside of an open set is not part of the proposition proven in [3]. However, it is shown that there exists a sequence of curves in C which converges to a minimum length path. The sequence of curves all lie outside the open set, int(Ω), therefore the limit must also lie outside int(Ω).

Theorem 2 ([3]) *Every minimum length planar curve satisfying the conditions of section 1.1 with $\Omega = \emptyset$ is necessarily a C^1 curve which is either*

1. *an arc of a circle of radius R. followed by a line segment. followed by an arc of a circle of radius R*
2. *a sequence of three arcs of circles of radius R*
3. *a subpath of a path of either of these two types*

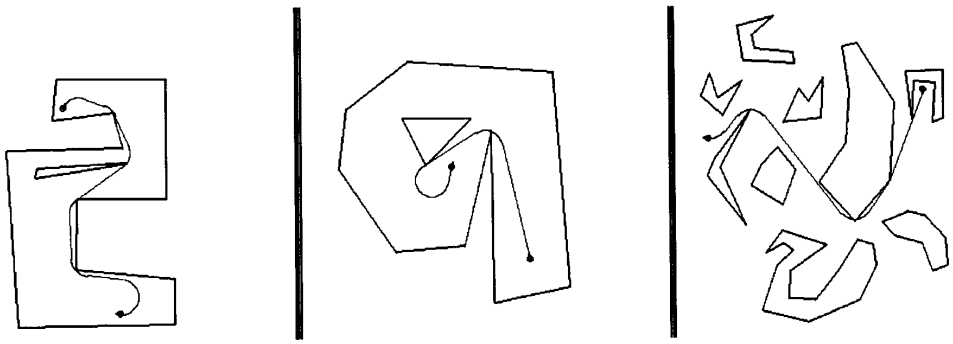

Figure 8: Paths generated by the algorithm.

Corollary Let Ω above be a set of polygons in the plane with boundary $\partial\Omega$. Every minimum length planar curve satisfying the conditions of theorem 1 is composed of path segments of the given types which pass through $\partial\Omega$ finitely many times. By "pass through finitely many times" we mean the curve is partitioned into finitely many intervals $[s_i, s_{i+1}]$ for which $X(s) \subset \partial\Omega$, $\forall s \in [s_i, s_{i+1}]$ and $X(s) \cap \partial\Omega = \emptyset$, $\forall s \in (s_{i+1}, s_{i+2})$

Sketch of Proof Either the path from (u,U) to (v,V) does not touch $\partial\Omega$, in which case it is obviously of the types given above, or it touches finitely many times. It can touch only finitely many times because a path of minimum length is necessarily finite and for the path to leave an edge, return and satisfy the curvature constraints requires a finite distance to be traveled. In between the contacts with $\partial\Omega$, the curve must be of a type given above. It is shown in [3] that the three arc type paths have length greater than πR. X is necessarily a straight line for the intervals $[s_i, s_{i+1}]$ with $s_i \neq s_{i+1}$ and $X(s) \subset \partial\Omega$, $\forall s \in [s_i, s_{i+1}]$.

Remark This corollary establishes that the algorithm allows paths of sufficient generality to include a minimal length path if one exists. We need look only for paths which touch the edges of the obstacles and have the form specified in the theorem. The algorithm is guaranteed to find an approximation to a minimal length path which is robust. This path may be longer than the minimal path.

6 Further Areas of Research

Various problems and extensions to this work are under preparation or work is in progress. The immediate focus of our attention is on the following list, which is by no means complete.

Different Robot Models This involves extending the algorithm described in this paper to plan paths for a robot which is larger than a point. We have solved one such case. A paper describing the extension of this algorithm to the case of a robot which is described as a disk is currently under preparation.

Planning Optimal Paths The grid and quadtree based algorithms only approximate the free space region. This may cause an optimal path to be overlooked. However, eliminating the sample trajectories will necessitate a completely different approach to searching free space.

Planning Robust Paths The use of quadtree partitioning of the free space allows the orientation of the robot at the via points to be in a neighborhood of the specified orientation for the path generated by the algorithm without causing a collision with an obstacle. This suggests that if robustness with respect to perturbation of the location of the via points can also be achieved, then the path will be very robust to errors in controlling the robot to follow a specified trajectory.

Planning C^r Paths, r>1 Practical systems will not be able to track a path which requires instantaneous changes in acceleration. Although the algorithm presented in this paper relies very heavily on the geometry of the class of trajectories considered, it may be possible to generalize to other types of trajectories without sacrificing complexity.

Many open problems remain to be solved which fall under the category of planning motion subject to non-holonomic constraints.

Acknowledgments

The authors would like to thank Professor Shankar Sastry, Greg Heinzinger, Jeff Mason, Niklas Nordstrom, Saman Behtash, and Richard Murray for their helpful discussions and aid in producing this paper.

References

[1] J. Laumond, "Finding collision-free smooth trajectories for a non-holonomic mobile robot," in *IJCAI 87 Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, (Milan), pp. 1120–1123, IJCAI, Inc., August 1987.

[2] S. Fortune and G. Wilfong, "Planning constrained motion," in *STOCS*, (Chicago IL), pp. 445–459, Association for Computing Machinery, May 1988.

[3] L. E. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[4] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Berlin: Springer-Verlag, 1984.

[5] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*. Reading MA: Addison-Wesley, 1983.