# An Opportunistic Global Path Planner[1]

**John F. Canny[2] and Ming C. Lin[3]**

**Abstract**

In this paper we describe a robot path planning algorithm that constructs a global skeleton of free-space by incremental local methods. The curves of the skeleton are the loci of maxima of an artificial potential field that is directly proportional to distance of the robot from obstacles. Our method has the advantage of fast convergence of local methods in uncluttered environments, but it also has a deterministic and efficient method of escaping local extremal points of the potential function. We first describe a general roadmap algorithm, for configuration spaces of any dimension, and then describe specific applications of the algorithm for robots with two and three degrees of freedom.

**Key Words:** Obstacle Avoidance, Global Path Planner, Roadmap Algorithm, Robot Motion Planning, Artificial Potential Field.

---

[1]This research was supported by a David and Lucile Packard Foundation Fellowship and by NSF Presidential Young Investigator Grant number IRI-8958577

[2]Department of Computer Science, 543 Evans Hall, University of California, Berkeley, CA 94720, USA. email: jfc@robotics.berkeley.edu

[3]Department of Electrical Engineering and Computer Science, 211 Cory Hall - Box #79, University of California, Berkeley, CA 94720, USA. email: mlin@robotics.berkeley.edu

# 1 Introduction

There have been two major approaches to motion planning for manipulators, (i) local methods, such as artificial potential field methods [1], which are usually fast but are not guaranteed to find a path, and (ii) global methods, like the first Roadmap Algorithm [2], which is guaranteed to find a path but may spend a long time doing it. In this paper we present an algorithm which has characteristics of both. Our method is an incremental construction of a skeleton of free-space. Like the potential field methods, the curves of this skeleton locally maximizes a certain potential function that varies with distance from obstacles. Like the Roadmap Algorithm, the skeleton, computed incrementally, is eventually guaranteed to contain a path between two configurations if one exists. The size of the skeleton in the worst case, is comparable with the worst-case size of the roadmap.

Unlike the local methods, our algorithm never gets trapped in local extremal points. Unlike the Roadmap Algorithm, our incremental algorithm can take advantage of a non-worst-case environment. The complexity of the roadmap came from the need to take recursive slices through configuration space. In our incremental algorithm, slices are only taken when an initial search fails and there is a "bridge" through free space linking two "channels". The new algorithm is no longer recursive because bridges can be computed directly by hill-climbing . The bridges are built near "interesting" critical points and inflection points. The conditions for a bridge are quite strict. Possible candidate critical points can be locally checked before a slice is taken. We expect few slices to be required in typical environments.

In fact, we can make a stronger statement about completeness of the algorithm. The skeleton that the algorithm computes eventually contains paths that are homotopic to all paths in free space. Thus, once we have computed slices through all the bridges, we have a complete description of free-space for the purposes of path planning. Of course, if we only want to find a path joining two given points, we stop the algorithm as soon as it has found a path.

The tracing of individual skeleton curves is a simple enough task that we expect that it could be done in real time on the robot's control hardware, as in other artificial potential field algorithms. However, since the robot may have to backtrack to pass across a bridge, it does not seem worthwhile to do this during the search.

For those readers already familiar with the Roadmap Algorithm, the following description may help with understanding of the new method: If the configuration space is $\mathbb{R}^k$, then we can construct a hypersurface in $\mathbb{R}^{k+1}$ which is the graph of the potential function, i.e. if $P(x_1, \ldots, x_k)$ is the potential, the hypersurface is the set of all points of the form $(x_1, \ldots, x_k, P(x_1, \ldots, x_k))$. The skeleton we define here is a subset of a roadmap (in the sense of [2]) of this hypersurface.

This work builds on a considerable volume of work in both global motion planning methods [2] [3], [4], [5], and local planners, [1]. Our method shares a common theme with the work of Barraquand and Latombe [6] in that it attempts to use a local potential field planner for speed with some procedure for escaping local maxima. But whereas Barraquand and Latombe's method is a local method made global, we have taken a global method (the Roadmap Algorithm) and found a local opportunistic way to compute it.

Although our starting point was completely different, there are some other similarities with [6]. Our "freeways" resemble the valleys intuitively described in [6]. But the main difference between our method and the method in [6] is that we have a guaranteed (and reasonably efficient) method of escaping local potential extremal points and that our potential function is computed in the configuration space.

The paper is organized as follows: Section 2 contains a simple and general description of roadmaps. The description deliberately ignores details of things like the distance function used, because the algorithm can work with almost any function. Section 3 gives some particulars of the application of artificial potential fields. Section 4 describes our incremental algorithm, first for robots with two degrees of freedom, then for three degrees of freedom.

# 2 A Maximum Clearance Roadmap Algorithm

We denote the space of all configurations of the robot as $CS$. For example, for a rotary joint robot with $k$ joints, the configuration space $CS$ is $\mathbb{R}^k$, the set of all joint angle tuples $(\theta_1, \ldots, \theta_k)$. The set of configurations where the robot overlaps some obstacle is the configuration space obstacle $CO$, and the complement of $CO$ is the set of free (non-overlapping) configurations $FP$. As described in [2], $FP$ is bounded by algebraic hypersurfaces in the parameters $t_i$ after the standard substitution $t_i = \tan(\frac{\theta_i}{2})$. This result is needed for the complexity bounds in [2] but we will not need it here.

A roadmap is a one-dimensional subset of $FP$ that is guaranteed to be connected within each connected component of $FP$. Roadmaps are described in some detail in [2] where it is shown that they can be computed in time $O(n^k \log n(d^{O(n^2)}))$ for a robot with $k$ degrees of freedom, and where free space is defined by $n$ polynomial constraints of degree $d$ [7]. But $n^k$ may still be too large for many applications, and in many cases the free space is much simpler than its worst case complexity, which is $O(n^k)$. We would like to exploit this simplicity to the maximum extend possibly. The results of [6] suggest that in practice free space is usually much simpler than the worst case bounds. What we will describe is a method aimed at getting a minimal description of the connectivity of a particular free space. The original description of roadmaps is quite technical and intricate. In this paper, we give a less formal and hopefully more intuitive description.

## 2.1 Definitions

Suppose $CS$ has coordinates $x_1, \ldots, x_k$. A slice $CS|_v$ is a slice by the hyperplane $x_1 = v$. Similarly, slicing $FP$ with the same hyperplane gives a set denoted $FP|_v$. The algorithm is based on the key notion of a channel which we define next:

> A *channel-slice* of free space $FP$ is a connected component of some slice $FP|_v$.

The term channel-slice is used because these sets are precursors to channels. To construct a channel from channel slices, we vary $v$ over some interval. As we do this, for most values of $v$, all that happens is that the connected components of $FP|_v$ change shape continuously. As $v$ increases, there are however a finite number of values of $v$, called *critical values*, at which there is some topological change. Some events are not significant for us, such as where the topology of a component of the cross-section changes, but there are four important events: As $v$ increases a connected component of $FP|_v$ may appear or disappear, or several components may join, or a single component may split into several. The points where joins or splits occur are called *interesting critical points*. We define a channel as a maximal connected union

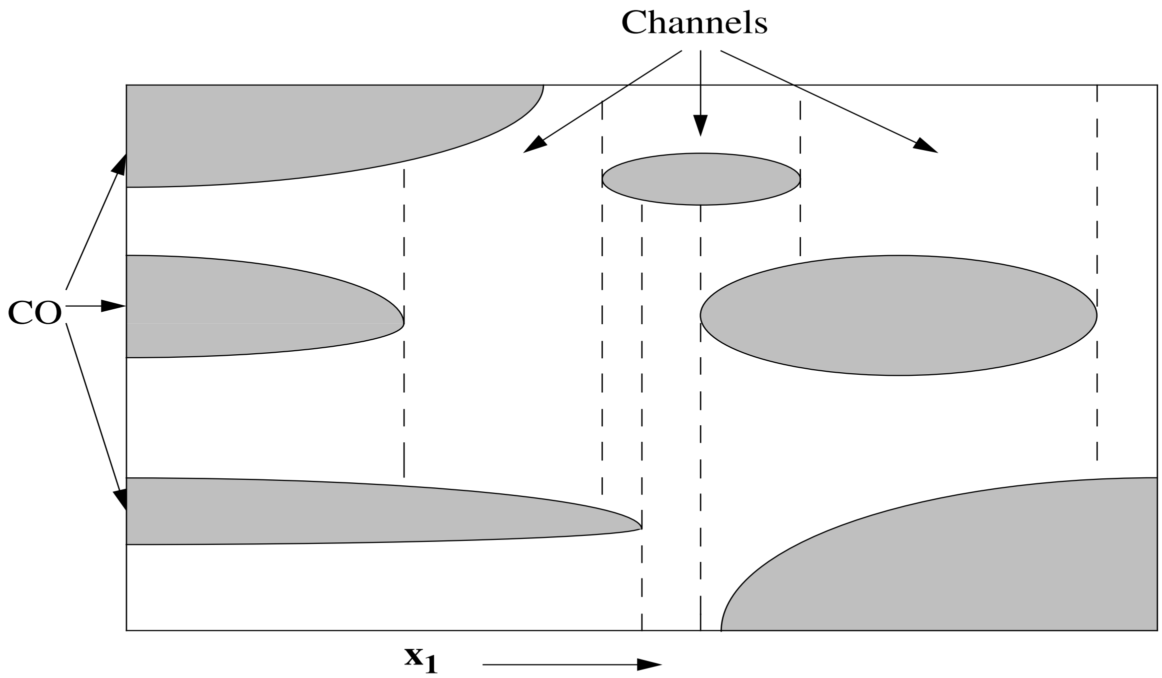



Figure 1: A schematized 2-d configuration space and the partition of free space into $x_1$-channels.

of cross sections that contains no image of interesting critical points. We use the notation $FP|_{(a,b)}$ to mean the subset of $FP$ where $x_1 \in (a,b) \subset \mathbb{R}$.

A *channel* through $FP$ is a connected component of $FP_{(a,b)}$ containing no splits or joins, and (maximality) which is not contained in a connected component of $FP_{(c,d)}$ containing no splits or joins, for $(c,d)$ a proper superset of $(a,b)$. See Fig. 1 for an example of channels.

The property of no splits or joins can be stated in another way. A maximal connected set $C|_{(a,b)} \subset FP|_{(a,b)}$ is a channel if every subset $C|_{(e,f)}$ is connected for $(e,f) \subset (a,b)$.

## 2.2 The General Roadmap

Now to the heart of the method. A roadmap has two components:

(i) Freeways (called silhouette curves in [2]) and

(ii) Bridges (called linking curves in [2]).

A freeway is a connected one-dimensional subset of a channel that forms a backbone for the channel. The key properties of a freeway are that it should span the channel, and be continuable into adjacent channels. A freeway *spans* a channel if its range of $x_1$ values is the same as the channels, i.e. a freeway for the channel $C|_{(a,b)}$ must have points with all $x_1$ coordinates in the range $(a, b)$. A freeway is *continuable* if it meets another freeway at its endpoints. i.e. if $C|_{(a,b)}$ and $C'|_{(b,c)}$ are two adjacent channels, the $b$ endpoint of a freeway of $C|_{(a,b)}$ should meet an endpoint of a freeway of $C'|_{(b,c)}$. (Technically, since the intervals are open, the word "endpoint" should be replaced by "limit point")

In general, when a specific method of computing freeway curves is chosen, there may be several freeways within one channel. For example, in the rest of this paper, freeways are defined using artificial potential functions which are directly proportional to distance from obstacles. In this case each freeway is the locus of local maxima in potential within slices $FP|_v$ of $FP$ as $v$ varies. This locus itself may have some critical points, but as we shall see, the freeway curves can be extended easily past them. Since there may be several local potential maxima within a slice, we may have several disjoint freeway curves within a single channel, but with our incremental roadmap construction, this is perfectly OK.

Now to bridges. A bridge is a one-dimensional set which links freeways from channels that have just joined, or are about to split (as $v$ increases). Suppose two channels $C_1$ and $C_2$ have joined into a single channel $C_3$, as shown in Fig. 2. We know that the freeways of $C_1$ and $C_2$ will continue into two freeway curves in $C_3$. These freeways within $C_3$ are not guaranteed to connect. However, we do know that by definition $C_3$ is connected in the slice slice $x_1 = v$ through the critical point, so we add linking curves from the critical point to *some* freeway point in each of $C_1$ and $C_2$. It does not matter which freeway point, because the freeway curves inside the channels $C_1$ and $C_2$ must be connected within each channel, as we show in Appendix I. By adding bridges, we guarantee that whenever two channels meet (some points on) their freeways are connected.

Once we can show that whenever channels meet, their freeways do also (via bridges), we have shown that the roadmap, which is the union of freeways and bridges, is connected. The proof of this very intuitive result is a simple

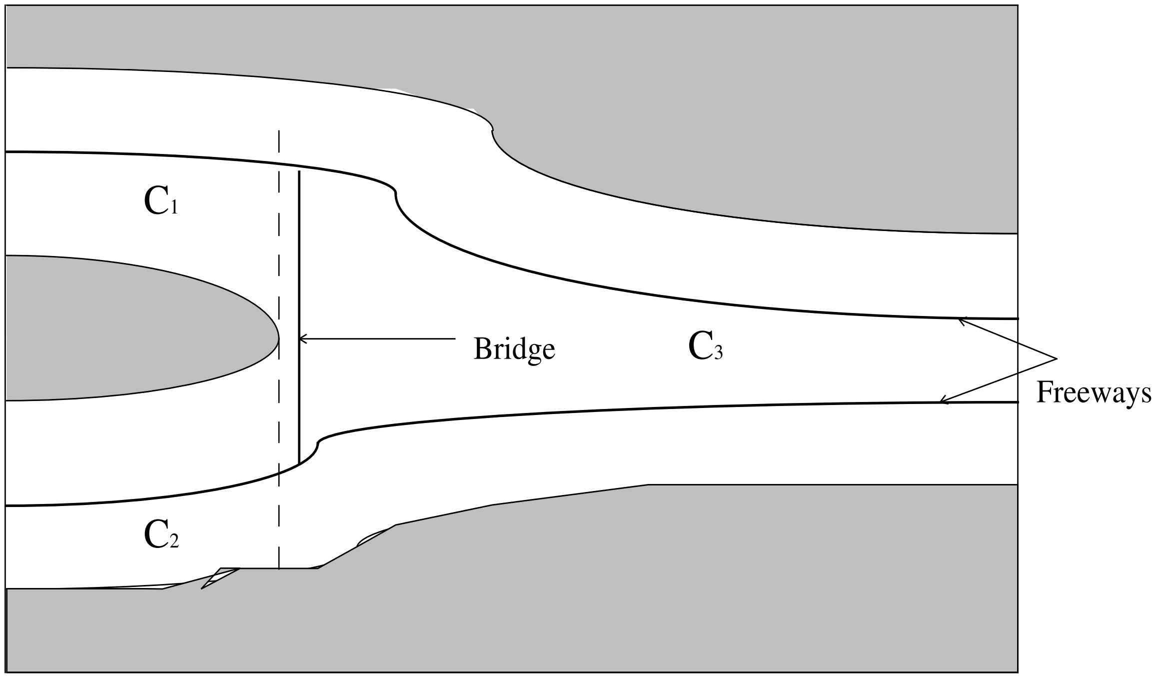



Figure 2: Two channels $C_1$ and $C_2$ joining the channel $C_3$, and a bridge curve in $C_3$.

inductive argument on the (finite number of) channels, given in Appendix I.

The basic structure of the general Roadmap Algorithm follows:

1. Start tracing a freeway curve from the start configuration, and also from the goal.

2. If the curves leading from the start and goal are not connected, enumerate a split or join point, and add a bridge curve "near" the split or join ($x_1$-coordinate of the slice slightly greater than that of the joint point for a join, slightly less for a split).

3. Find all the points on the bridge curve that lie on other freeways, and trace from these freeways. Go to step (2).

The algorithm terminates at step (2) when either the start and goal are connected, in which case the algorithm signals success and returns a connecting path, or if it runs out of split and join points, in which case there is no path connecting the start and goal. This description is quite abstract, but in later sections we will give detailed description of the approach in two- and three-dimensional configuration spaces.

Three things distinguish our new algorithm from the previous Roadmap Algorithm. The most important is that the new algorithm is not recursive. Step 2 involves adding a bridge curve which is two pieces of curve found by hill-climbing on the potential. In the original roadmap algorithm, linking curves had to be defined recursively, because it is not possible to hill-climb to a maximum with an algebraic curve. Another difference is that the freeways do not necessarily lie near the boundary of free space as they did in [2]. In our present implementation we are in fact using maximum clearance freeways. But the most important difference is that we now only enumerate *true* split or join points. For a robot with $k$ degrees of freedom and an environment of complexity $n$, it can be shown that there are at most $O(n^{(k-1)})$ potential split or join points. (Please refer to Appendix II for the proof on the upper bound for the maximum number of interesting critical points.) But many experiments with implemented planners in recent years have shown that the number of true splits or joins in typical configuration spaces is much lower. In our new algorithm, we can make a purely local test on a potential split or join point to see if it is really qualified. The vast majority of candidates will not be, so we expect far fewer than $O(n^{(k-1)})$ bridges to be required.

**Definition**

A point $p$ in $\mathbb{R}^{k+1}$ is an *interesting critical point* if for every neighborhood $U$ of $p$, one of the following holds:

(i) The intersection $U \cap x_1^{-1}(x_1(p)+\epsilon)$ consists of several connected components for all sufficiently small $\epsilon$. This is a generalized split point.

(ii) The intersection $U \cap x_1^{-1}(x_1(p)-\epsilon)$ consists of several components for all sufficiently small $\epsilon$. This is a generalized join point.

We will assume the environment is generic, i.e. there is no special topology such that a small perturbation will change the clearance of the paths. This is true for almost all practical situations: most of obstacles have a reasonably large interior that a small perturbation will not affect much of the obstacle configuration space. Based on the transversality condition of general position assumptions in [2], the interesting critical points can be computed as follows. The set $S$ is defined by inequalities, and its boundary is a union of surfaces of various dimensions. Let $S_\alpha$ be such a surface; it will be defined as the intersection of several configuration space constraint surfaces. Each of these is given by an equation of the form $f_i = 0$. To find the critical points of such a surface w.r.t. the function $x_1(.)$, we first define a polynomial $g$ as follows:

$$g = \sum_{i=1}^{l} f_i^2 \tag{1}$$

and then solve the system

$$g = \epsilon, \quad \frac{\partial}{\partial x_2} g = 0 \quad \cdots \quad \frac{\partial}{\partial x_k} g = 0 \tag{2}$$

where $l$ is the number of equations which are zero on $S_\alpha$, the $x_2, \ldots, x_k$ are coordinates which are orthogonal to $x_1$, and $\epsilon$ is an infinitesimal that is used to simplify the computation (see [2]).

It can be shown [8] that the solutions of interest can be recovered from the lowest degree coefficient in $\epsilon$ of the resultant of this system. This normally involves computing a symbolic determinant which is a polynomial in $\epsilon$ [9]. But a more practical approach is to recover only the lowest coefficient in $\epsilon$ by using straight line program representations and differentiating [10].

To enumerate all the interesting critical points is computationally expensive, since we have to solve $O(n^{(k-1)})$ systems of non-linear equations. Thus, we also plan to experiment with randomly chosen slice values in some bounded ranges, alternating with slices taken at true split or join points. The rationale for this is that in practice the "range" of slice values over which a bridge joins two freeways is typically quite large. There is a good probability of finding a value in this range by using random values. Occasionally there will be a wide range of slice values for a particular bridge, but many irrelevant split and join points may be enumerated with values outside this range. To make sure we do not make such easy problems harder than they should be, our implementation alternates slices taken near true split and join points with slices taken at random $x_1$ values.

# 3 Defining the Distance Function

The idea of our approach is to construct a potential field which repels the point robot in configuration space away from the obstacles. Given a goal position and a description of its environment, a manipulator will move along a "maximum potential" path in an "artificial potential field". The position to be reached represents a critical point that will be linked by the *bridge* to the nearest maximum, and the obstacles represent repulsive surfaces for the manipulator parts.

Let $CO$ denote the obstacles, and $x$ the position in $\mathbb{R}^k$. The artificial potential field $U_{art}(x)$ induces an artificial repulsion from the surface of the obstacles. $U_{art}(x)$ is a non-negative function whose value tends to zero as any part of the robot approaches an obstacle. One of the classical analytical potential fields is the Euclidean distance function.

Using the shortest distance to an obstacle $O$, we have proposed the following potential field $U_{art}(x)$:

$$U_{art}(x) = \min_{ij}(D(O_i, L_j(x)))$$

where $D(O_i, L_j(x))$ is the shortest Euclidean distance between an obstacle $O_i$ and the link $L_j$ when the robot is at configuration $x$. $D(O_i, L_j(x))$ is obtained by a local method for fast computation of distance between convex polyhedra [11].

Notice that the proposed $U_{art}(x)$ is not a continuously differentiable function as in many potential field methods. $U_{art}(x)$ is *piecewise* continuous and differentiable. This is perfectly all right for the application in our Roadmap algorithm. In fact it will be a lower envelope of smooth functions. This is all the better because it means that local maxima that do not occur where the function is smooth are all the more sharply defined. The graph of the distance function certainly has a *stratification* into a finite number of smooth pieces [12]. Its maxima will be the union of certain local maxima of these smooth pieces. So we can still use the system of equations defined earlier to find them.

With this scheme, a manipulator moves in such a way to maximize the artificial potential field $U_{art}(x)$. But like any local method, just following one curve of such maxima is not guaranteed to reach the goal. Thus, the need for bridges.

# 4 Algorithm Details

The algorithm takes as input a geometric description of the robot links and obstacles as convex polyhedra or unions of convex polyhedra. It also takes the initial and goal configurations, and the kinematic description of the robot, say via Denavit-Hartenberg parameters. The output is a path between the initial and goal configurations represented as a sequence of closely spaced points (more closely than the C-space distance to the nearest obstacle at that point), assuming such a path exists. If there is no path, the algorithm will eventually discover that, and output "NO PATH".

The potential function is a map $U_{art} : CS \rightarrow \mathbb{R}$. The graph of the function is a surface in $CS \times \mathbb{R}$. Let $u$ and $v$ denote two coordinate axes, the Roadmap Algorithm fixes $v$ and then follows the extremal points in direction $u$ as the value of $v$ varies. But, the new algorithm differs from the original roadmap algorithm[2] in the following respects:

- It does not always construct the entire roadmap
- In the new algorithm, $v = x_i$, where $x_i$ is one of the CS coordinates while $u = h$, where $h$ is the height of the potential function. Yet, in the original, $v = x_i$ and $u = x_j$ where $x_i$ and $x_j$ are *both* CS coordinates.

- The original Roadmap algorithm fixes the $x_i$ coordinate and follows extremal points (maxima, minima and saddles) in $x_j$, to generate the silhouette curves. On the other hand, the new algorithm fixes $x_i$, and follows only *maxima* in $h$.
- The new algorithm is not recursive. Recursion was necessary in the original because there is no single algebraic curve that connects an arbitrary point to an extremum in $u$. But the new algorithm uses numerical hill-climbing which has no such limitation.

## 4.1 Freeways and Bridges

A roadmap has two major components – freeways and bridges. They are generated as following:

**Freeway Tracing** is done by tracking the locus of local maxima in distance within each slice normal to the sweeping direction. Small steps are made in the sweep direction, and the local maxima recomputed numerically. Freeway tracing continues in both directions along the freeway until it terminates in one of two ways:

(a) The freeway runs into an inflection point, a point where the curve tangent becomes orthogonal to the sweep direction. It is always possible to continue past these points by adding a bridge.

(b) The freeway runs into an obstacle. This is a normal termination, and the tracing simply stops and the algorithm backtracks.

**Bridges** begin always at inflection points or critical points and terminate always at freeway points within the same slice. The algorithm simply follows the gradient of the potential function from the start point within the slice until it reaches a local maximum, which must be a freeway point.

**Enumeration of Critical Points**

Critical points are calculated as in Section 2. But most of these critical points will not be interesting. We can check locally among all the critical points to see if they qualify to be a "split" or "join". This test checks if the point has a neighborhood that is "saddle-like". It is based on the orientations of the CSpace boundary surface normals at the critical point.

**Random Slicing**

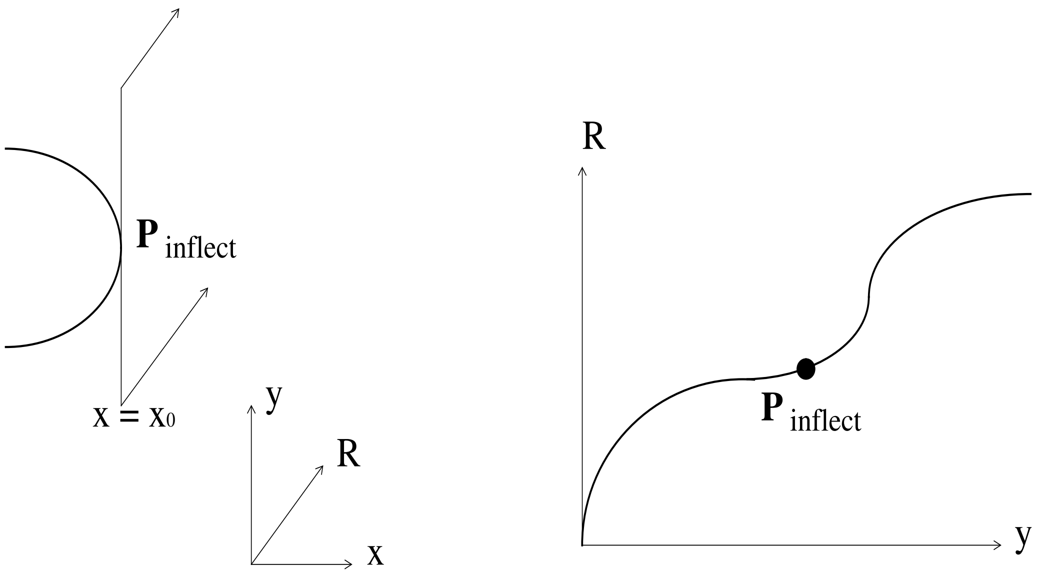
Optionally, the user may wish to add roadmaps of randomly chosen slices, rather than calculating many critical points (or rather than calculating them at all, but then of course, completeness will be lost). This *is* a recursive procedure, and involves choosing a $v$ value at random, making a recursive call to the algorithm on this $v$-slice.

Random slicing may also be used within the slice itself, and so on, which leads to a depth-$k$ recursion tree. If this is done, some search heuristics must be added to guide the choice of where in the tree the next slice (and hence the next child) should be added. The heuristic also needs to trade off random slicing and critical point enumeration. The goal of the heuristic is to enumerate enough random slices that the algorithm will have a good chance of success on "easy" environments (intuitively where there are large passages between channels) without having to explore too many critical points. Yet it should still find its way around a difficult environment using the critical slices without having wasted most of its time taking random slices.

Given the general outline of our algorithm, we will now give an instantiation on 2-D and a detailed description of how it can be applied to 3-D.

## 4.2 Two-Dimensional Workspace

Starting from the initial position $p_{init} \in CS$, we first fix one of the axes of $CS$ and then take the $x$ coordinate of a slice to be the $x$ coordinate of $p_{init}$. Then we search this slice to find the nearest local maximum. (This local maximum is a freeway point.) Next, we build a *bridge* between the point $p_{init}$ and this local maximum. At the same time, we begin tracing a freeway curve from the goal. If the goal is not on the maximum contour of the potential field, then we must build a *bridge* to link it to the nearest local maximum. Afterwards, we trace the locus of this local maximum as $x$ varies until we reach an endpoint. If the current position $p_{loc}$ on the curve is the goal $G$, then we can terminate the procedure. Otherwise, we must verify whether $p_{loc}$ is a "dead-end" or an inflection point of the slice $x = x_0$. (See Fig. 3.) If $p_{loc}$ is a point of inflection, then we can continue the curve by taking a slice at the neighborhood of the inflection point and hill-climbing along the gradient direction *near* the inflection point. This search necessarily




● **Portion of silhouette curve in CS x R** ● **Slice projection at x = $x_0$ in R-y plane**

Figure 3: A pictorial example of an inflection point in $CS \times \mathbb{R}$ vs. its view in $\mathbb{R} \times y$ at the slice $x = x_0$

takes us to another local maximum.

Fig. 4 demonstrates how the algorithm works in 2-d $CS$. This diagram is a projection of a constructed potential field in $CS$ x $\mathbb{R}$ onto the $x$-$y$ plane of the 2-d $CS$. The shaded area is the $CO$ in the configuration space. The solid curves represent the contour of maximum potential, while the dashed curves represent the minima. Furthermore, the path generated by our planner is indicated by arrows. In addition, the vertical lines symbolize channel slices through the interesting critical points and inflection points. When this procedure has been taken to its conclusion and both endpoints of the freeway terminate at dead-ends, then at this point it is necessary to take a slice at some value of $x$. Our planner generates several random $x$-values for slices (at a uniformly spaced distribution along the span of the freeway), interweaving them with an enumeration of all the interesting critical points. If after a specified number of random values, our planner fails to find a connecting path to a nearby local maximum, then it will take a slice through an inter-

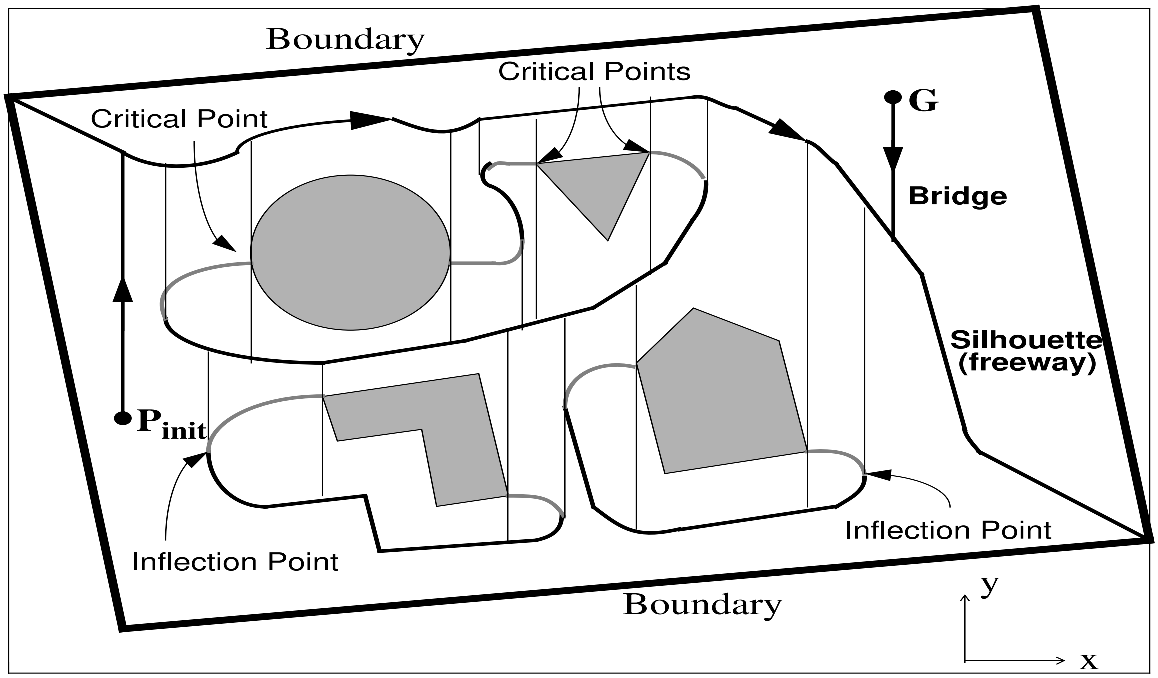



Figure 4: An example of the algorithm in the 2-d workspace

esting critical point. Each slice, being 1-dimensional, itself forms the bridge curve (or a piece of it does). We call this procedure repeatedly until we reach the goal position $G$ or have enumerated all the interesting critical points.

The algorithm is described schematically below:

## • Algorithm

Procedure FindGoal (Environment, $p_{init}$, G)

if ($p_{init} \neq G$)
  then Explore($p_{init}$) and Explore($G$)
  else return(FoundGoal);
even := false;
While ( CritPtRemain and NotOnSkeleton(G) ) do
  if (even)
    then x := Random (x-range)
    else x := x-coord(next-crit-pt());

```
        TakeSlice(x);
        even := not even;
    end(while);

End(FindGoal);

Function Explore(p)
% Trace out a curve from p

    q := search-up&down(p);
    % To move up & down only in y, using gradient near p
    if new(q) then
    % new() checks if q is already on the curve
        begin(if)
        <e1,e2> := trace(q);
        % trace out the curve from q, output two end points
        if inflection(e1) then Explore(e1);
        if inflection(e2) then Explore(e2);
        % inflection(p) checks if p is an inflection point
        end(if);

End(Explore);

Function TakeSlice(x-coordinate(p))
% This function generates all points on the slice and explore
% all the maxima on the slice.

    old-pt := find-pt(x-coordinate);
    % find-pt() find all the points on the x-coordinate.
    % It moves up&down until reaches another maximum.
    new-pt := null;
    For (each pt in the old-pt) do
        <up,down> := search-up&down(pt);
        % <up,down> is a pair of points of 0,1,or2 pts
        new-pt := new-pt + <up,down>;
        For (each pt in the new-pt) do
            Explore(pt);

End(TakeSlice);
```

## 4.3 Three-Dimensional Workspace

For a three-dimensional workspace, the construction is quite similar. Starting from the initial position $p_{init}$ and the goal $G$, we first fix one axis, $X$. We trace from the start point to a local maximum of distance within the $Y$-$Z$ plane containing the start point. Then we follow this local maximum by taking steps in $X$. If this curve terminates in an inflection point, we can always reach another maximum by following the direction of the potential gradient *just beyond* the inflection point in $X$. Eventually, though, we expect to terminate by running into an obstacle.

When we wish to enumerate a critical point, the bridge curve is the same as the first segment that we used to get from $p_{init}$ onto the freeway. That is, we trace from the critical point along the direction of the gradient within the current $Y$-$Z$ slice. There will be two directions outward from the critical point along which the distance increases. We follow both of these, which gives us a bridge curve linking freeways of two distinct channels.

If we decide to use random slicing, we select a slice $FP|_x$ normal to the $x$-axis and call the algorithm of the last section on that slice. We require it to produce a roadmap containing any freeway points that we have found so far that lie in this slice. This algorithm itself may take random slices, so we need to limit the total number of random slices taken before we enumerate the next interesting critical point (in 3-D), so that random slicing does not dominate the running time.

## 4.4 Path Optimization

After the solution path is obtained, we plan to smooth it by the classical principles of variational calculus, i.e. to solve a classical two points boundary value problem. Basically we minimize the potential which is a function of both distance and smoothness to find a *locally* optimal (smooth) path.

Let $s$ be the arc that is the path refined from a sequence of points between $a$ and $b$ in space, $r$ be the shortest Euclidean distance between the point robot and the obstacle, and $\kappa$ be the curvature of the path at each point. The cost function for path optimization that we want to minimize is:

$$f(s, r, \kappa) = \int_a^b (\frac{A}{r^2} + B\kappa^2) ds$$

where $r$, $\kappa$, and $s$ are functions of a point $P_i$ in a given point sequence, and $A, B$ are adjustment constants. Taking the gradient of this function with respect to each point $P_i$ gives us the direction of an improved, *locally* optimal path.

This can be done in the following manner: given a sequence of points $(P_1, P_2, \cdots, P_k)$, we want to minimize the cost function

$$g(P_i) = \sum_i \frac{A}{r(P_i)^2}|\Delta S_i| + B\kappa(P_i)^2|\Delta S_i| \tag{3}$$

where $\Delta S_i$ and $\kappa(P_i)$ are defined as:

$$\Delta S_i = \frac{P_{i+1} - P_{i-1}}{2}$$

$$\kappa(P_i) = \frac{\angle P_{i-1}, P_i, P_{i+1}}{|\Delta S_i|}$$

Now, taking the gradient w.r.t. $P_i$, we have

$$\nabla g(P_i) = \sum_i \frac{-2A}{r(P_i)^3}\nabla r(P_i)|\Delta S_i| + 2B\kappa(P_i)\nabla\kappa(P_i)|\Delta S_i| \tag{4}$$

The most expensive procedure in computing the above gradient is to compute the distance at each point. By using the incremental distance calculation algorithm described in [11], we can compute the distance between the robot and the closest obstacle in constant time. Since we have to do this computation for a sequence of points, the computation time for each iteration to smooth the curve traced out by our planner is linear in the total number of points in a given sequence. After several iterations of computing the gradient of the summation in Eqn.4, the solution path will eventually be smooth and *locally* optimal.

# 5 Complexity Bound

Since our planner probably does not need to explore all the critical points, this bound can be reduced by finding only those *interesting* critical points where adding a bridge helps to reach the goal. If $n$ is the number of obstacle features (faces, edges, vertices) in the environment and the configuration space is $\mathbb{R}^k$, then the number of "interesting critical points" is at most $O((2d)^k n^{(k-1)})$. As mentioned earlier, the algorithm is no longer recursive in calculating the critical points and linking curves (bridges) as in [2], the complexity bound calculated in [2] does not apply here. (Please refer to Appendix II for more details.)

# 6 Summary and Discussion

By following the maxima of a well-designed potential field, and taking slice projections through critical points and at random values, our approach builds incrementally an obstacle-avoiding path to guide a robot toward the desired goal. The techniques proposed in this paper provide the planner with a systematic way to escape from these local maxima that have been a long standing problem with using the potential field approach in robot motion planning.

Our algorithm, computed from local information about the geometry of configuration space, requires no expensive precomputation steps as in most global methods developed thus far. In a two dimensional space, this method is comparable with using a Voronoi Diagram for path planning. In three-dimensional space, however, our method is more efficient than computing hyperbolic surfaces for the Voronoi diagram method. In the worst case, it should run at least as fast as the original roadmap algorithm. But, it should run faster in almost all practical cases.

# Appendix I: Proof of Completeness for an Opportunistic Global Path Planner

Careful completeness proofs for the roadmap algorithm are given in [2] and [13]. These proofs apply with very slight modification to the roadmap algorithm that we describe in this paper. The roadmap of [2] is the set of extremal points in a certain direction in free space. Therefore it hugs the boundary of free space. The roadmap described in this paper follows extrema of the *distance function*, and therefore stays well clear of obstacles (except at critical points). But in fact the two are very similar if we think of the *graph of the distance function* in $\mathbb{R}^n$. This is a surface $S$ in $\mathbb{R}^{(n+1)}$ and if we follow the extrema of distance on this surface, the roadmap of this paper is exactly a roadmap in the sense of [2] and [13].

The silhouette curves of [2] correspond to the freeway curves of this paper, and the linking curves correspond to bridges. Recall the basic property required of roadmaps:

**Definition**

A subset of $R$ of a set $S$ satisfies the *roadmap condition* if every connected component of $S$ contains a single connected component of $R$.

For this definition to be useful, there is an additional requirement that any point in $S$ can "easily" reach a point on the roadmap.

There is one minor optimization that we take advantage of in this paper. That is to trace only *maxima* of the distance function, rather than both maxima and minima. This can also be applied to the original roadmap.

For those readers not familiar with the earlier papers, we give here an informal sketch of the completeness proof. We need some notation first.

Let $S$ denote the surface in $\mathbb{R}^{(n+1)}$ which is the graph of the distance function. $S$ is an $n$-dimensional set and is semi-algebraic if configuration space is suitably parametrized. This simply means that it can be defined as a boolean combination of inequalities which are polynomials in the configuration space parameters.

One of the coordinates in configuration space $\mathbb{R}^n$ becomes the *sweep direction*. Let this direction be $x_1$. Almost any direction in $CS$ will work, and heuristics can be used to pick a direction which should be good for a particular application. When we take slices of the distance surface $S$, they are taken normal to the $x_1$ coordinate, so $S|_a$ means $S \cap (x_1 = a)$. Also, for a point $p$ in $\mathbb{R}^{(n+1)}$, $x_1(p)$ is the $x_1$-coordinate of $p$.

The other coordinate we are interested in is the distance itself, which we think of as the height of the distance surface. So for a point $p$ in $\mathbb{R}^{(n+1)}$, $h(p)$ is the value of the distance at this configuration.

For this paper, we use a slightly different definition of silhouettes, taking only local maxima into account. We will assume henceforth that the configuration space is bounded in every coordinate. This is certainly always the case for any practical robot. If it is not bounded, there are technical ways to reduce to a bounded problem, see for example [7]. The set of free configurations is also assumed to be closed. The closed and bounded assumptions ensure that the distance function will attain locally maximal values on every connected component of free space.

A *silhouette point* is a locally maximal point of the function $h(.)$ on some slice $S|_a$ of $S$. The *silhouette* $\Sigma(S)$ of $S$ is the set of all such points for all $a$.

The key properties of the silhouette are ([2], [13]):

(i) Within each slice of $S$, each connected component of $S|_a$ must contain at least one silhouette point.

(ii) The silhouette should be one-dimensional.

(iii) The critical points of the silhouette w.r.t the function $x_1(.)$ should include the critical points of the set $S$.

Clearly, using local maxima will satisfy property (i). This is true simply because a continuous function (in this case, a distance function with the value zero on the boundary and positive values in the interior) has a local maximum in a compact set. For property (ii) we require that the directions $x_1$ and $h$ be "generic" (see the earlier papers). This is easily done by picking a general $x_1$, but $h$ may not be generic a priori. However, rather than the true distance $h$, we assume that the distance plus a very small linear combination of the other coordinates is used. This linear combination can be arbitrarily small, and we assume that it is small enough that it does not significantly affect the clearance of silhouette points.

For property (iii), we depart somewhat from the original definition. The critical points of the silhouette curves that we have traced can be discovered during the tracing process (they are the points where the curve tangent becomes orthogonal to $x_1$). But we need to find all (or a sufficient set of) critical points to ensure completeness. All critical points do indeed lie on silhouette curves, but since our algorithm is incremental, we may not discover these other curves unless we encounter points on them. So we need a systematic way to enumerate the critical points of $S$, since these will serve as starting points for tracing the silhouette curves that we need for completeness.

In fact, not all critical points of $S$ are required. There is a subset of them called *interesting critical points* that are sufficient for our purpose.

Intuitively, the interesting critical points are the split or join points in higher dimensions. They can be defined as follows:

**Definition**

A point $p$ in $\mathbb{R}^{k+1}$ is an *interesting critical point* if for every neighborhood $U$ of $p$, one of the following holds:

(i) The intersection $U \cap x_1^{-1}(x_1(p) + \epsilon)$ consists of several connected components for all sufficiently small $\epsilon$. This is a generalized split point.

(ii) The intersection $U \cap x_1^{-1}(x_1(p) - \epsilon)$ consists of several components for all sufficiently small $\epsilon$. This is a generalized join point.

From the definition above, it follows that as we sweep the plane $x_1 = a$ through $S$, the number of connected components of $S|_a$ changes only when the plane passes though interesting critical points.

**Definition**

Now we can define the roadmap of the surface $S$. The roadmap $R(S)$ is defined as follows: Let $P_C(S)$ be the set of interesting critical points of $x_1(.)$ on $S$, $P_C(\Sigma)$ be the set of critical points of $x_1(.)$ on the silhouette, and $P_C$ the union of these two. The roadmap is then:

$$R(S) = \Sigma(S) \cup (\bigcup_{p \in P_c} L(p)) \tag{5}$$

That is, the roadmap of $S$ is the union of the silhouette $\Sigma(S)$ and various linking curves $L(p)$. The linking curves join critical points of $S$ or $\Sigma$ to other silhouette points.

The new roadmap algorithm has an advantage over the original in that it is not restricted to algebraic curve segments. This is because the original was formulated to give precise algorithmic bounds on the planning problem, whereas the new algorithm approximates the silhouette by tracing. Tracing is just as easy for many types of non-algebraic curves as for algebraic ones.

This allows us to do linking in a single step, whereas algebraic linking curves must be defined recursively. We generate linking curves in the present context by simply fixing the $x_1$ coordinate and hill-climbing to a local maximum in $h(.)$. The curve traced out by the hill-climbing procedure starts at the critical point and ends at a local maximum (which will be a silhouette point) of the distance within the same $x_1$ slice. Thus it forms a linking curve to the silhouette. If we are at an interesting critical point, there will be two opposing directions (both normal to $x_1$) along which the distance function increases. Tracing in both directions links the critical point to silhouette points on both channels that meet at that critical point.

**Theorem** $R(S)$ satisfies the roadmap condition.

**Proof** Let $a_1, \ldots, a_m$ be the $x_1$-coordinates of the critical points $P_C$, and assume the $a_i$'s are arranged in ascending order. The proof is by induction on the $a_i$'s.

Our inductive hypothesis is that the roadmap condition holds to the "left" of $a_{i-1}$. That is, we assume that $R(S)|_{\leq a_{i-1}} = R(S) \cap x_1^{-l}(x_1 \leq a_{i-1})$ satisfies the roadmap condition as a subset of $S|_{\leq a_{i-1}}$.

The base case is $x_1 = a_1$. If we have chosen a general direction $x_1$, the set $S|_{a_1}$ consists of a single point which will also be part of the roadmap.

For the inductive step we start with some basic results from Chapter 2 in [12], which state that we can smoothly deform or retract a manifold (or union of manifolds like the surface $S$) in the absence of critical points. In this case, it implies that the set $S|_{<a_i}$ can be smoothly retracted onto $S|_{\leq a_{i-1}}$, because the interval $(a_{i-1}, a_i)$ is free of critical values. There is also a retraction which retracts $R(S)|_{<a_i}$ onto $R(S)|_{\leq a_{i-1}}$. These retractions imply that there are no topological changes in $R(S)$ or $S$ in the interval $(a_{i-1}, a_i)$, and if $R(S)|_{\leq a_{i-1}}$ satisfies the roadmap condition, then so does $R(S)|_{<a_i}$.

So all that remains is the transition from $R(S)|_{<a_i}$ to $R(S)|_{\leq a_i}$. Let $p_i$ be the critical point whose $x_1$ coordinate is $a_i$. The roadmap condition holds for $R(S)|_{<a_i}$, i.e. each component of $S|_{<a_i}$ contains a single component of $R(S)|_{<a_i}$. The only way for the condition to fail as $x_1$ increases to $a_i$ is if the number of silhouette curve components increases, i.e. when $p_i$ is a critical point of the silhouette, or if the number of connected components of $S$ decreases, which happens when $p_i$ is a join point. Let us consider these cases in turn:

If $p_i$ is a critical point of the silhouette, the tangent to the silhouette at $p_i$ is normal to $x_1$. By assumption, a new component of the silhouette appeared at $p_i$ as $x_1$ increased to $a_i$. This means that in the slice $x_1 = a_i - \epsilon$ (for $\epsilon$ small enough) there is no local maximum in the neighborhood of $p_i$. On the other hand, there must be a local maximum of distance in this slice, which we can find by hill-climbing. So to link such a critical point, we move by $\epsilon$ in the $-x_1$ direction (or its projection on $S$ so that we remain on the surface) to a nearby point $q_i$. Then we hill climb from $q_i$ in the slice $x_1 = a_i - \epsilon$ until we reach a local maximum, which will be a silhouette point. This pair of curves links $p_i$ to the existing roadmap, and so our inductive hypothesis is proved for $R(S)|_{\leq a_i}$.

At join points, though, the linking curve will join $p_i$ to a silhouette point in each of the two channels which meet at $p_i$. If these two channels are in fact separate connected components of $S|_{<a_i}$, the linking curve will join their respective roadmaps. Those roadmaps are by hypothesis connected within each connected component of $S|_{<a_i}$. Thus the union of these roadmaps and the linking curve is a single connected curve within the connected component of $S|_{\leq a_i}$ which contains $p_i$. Thus we have proved the inductive hypothesis for $a_i$ if $p_i$ is a join point. $\square$

We have proved that $R(S)$ satisfies the roadmap condition. And it is easy to link arbitrary points in free-space with the roadmap. To do this we simply fix $x_1$ and hill-climb from the given point using the distance function. Thus our algorithm is complete for finding collision-free paths.

Note that we do not need to construct configuration space explicitly to compute the roadmap. Instead it suffices to be able to compute the interesting critical points, and to be able to compute the distance function and its gradient. This should not surprise the reader familiar with differential topology. Morse theory has already shown us that the topology of manifolds can be completely characterized by looking locally at critical points.

# Appendix II: Geometric Relations between Critical Points and Contact Constraints

Let $n$ be the number of obstacle features in the environment and the robot has constant complexity. Free space $FP$ is bordered by $O(n)$ constraint surfaces. Each constraint surface corresponds to an elementary contact, either face-vertex or edge-edge, between a feature of the robot and a feature of the environment. Other types of contacts are called non-elementary, and can be viewed as multiple elementary contacts at the same point, e.g. vertex-edge. They correspond to intersections of constraint surfaces in configuration space.

**Definition**

An *elementary contact* is a local contact defined by a single equation. It corresponds to a constraint surface in configuration space. For example, face-vertex or edge-edge.

**Definition**

A *non-elementary contact* is a local contact defined by two or more equations. It corresponds to an intersection or conjunction of two or more constraint surfaces in configuration space. For example, vertex-edge or vertex-vertex. There are $O(n)$ of non-elementary contacts if the robot has constant complexity.

We can represent $CO$ in disjunctive form:

$$CO = \Big( \bigvee_{\substack{e_i \in edges(obstacles) \\ f_j \in faces(robot)}} O_{e_i,f_j} \Big) \vee \Big( \bigvee_{\substack{e_j \in edges(robot) \\ f_i \in faces(obstacles)}} O_{e_j,f_i} \Big)$$

where $O_{e_i,f_j}$ is an overlap predicate for possible contact of an edge and a face. See [2] for the definition of $O_{e_i,f_j}$. For a fixed robot complexity, the number of branches for the disjunctive tree grows linearly w.r.t. the environment complexity. Each $O_{e_i,f_j}$ has constant size, if the polyhedron is preprocessed [11]. Each clause, $O_{e_i,f_j}$, is a conjunction of inequalities. This disjunctive tree structure is useful for computing the maximum number of critical points by combinatorics. The interesting critical points (which correspond to the non-elementary contacts) occur when two or more constraint surfaces lie under one clause.

Using the disjunctive tree structure, we can calculate the upper bound for the maximum number of critical points by combinatorial means (by counting the number of systems of equations we must solve to find all the critical points). Generically, at most $k$ surfaces intersect in $k$ dimensions. For a robot with $k$ degrees of freedom and an environment of complexity $n$, (i.e. $n$ is the number of feature constraints between robot and obstacles) the number of critical points is

$$(2d)^k \begin{pmatrix} n+k \\ k \end{pmatrix} \quad = \quad O((2d)^k n^k)$$

where $d$ is the maximum degree of constraint polynomial equations. This is an upper bound on the number of critical points from [14] and [15]. Therefore, for a given robot (with fixed complexity), there are *at most* $O((2d)^k n^k)$ critical points in terms of $n$, where $k$ is the dimension of configuration space and $n$ is the number of obstacle features. (NOTE: We only use the above argument to prove the upper bound, *not* to calculate critical points in this fashion.)

These $O((2d)^k n^k)$ intersection points fall into two categories: (a) All the contacts are elementary; (b) one or more contacts are non-elementary. When all contacts are elementary, i.e. all contact points are distinct on the object, free space in a neighborhood of the intersection point is homeomorphic to the intersection of $k$ half-spaces (one side of a constraint surface), and forms a cone. This type of intersection point cannot be a split or join point, and does not require a bridge. However if one or more contacts are non-elementary, then the intersection point is a potential split or join point. But because the $O(n)$ non-elementary contact surfaces have codimension $\geq 2$, there are only $O(n^{(k-1)})$ systems of equations that define critical points of type (b), and therefore at most $O((2d)^k n^{k-1})$ possible points. Interesting critical points may be either intersection points, and we have seen that there are $O((2d)^k n^{(k-1)})$ candidates; or they may lie on higher dimensional intersection surfaces, but these are certainly defined by fewer than $k$ equations, and the number of possible critical points is not more than $O((2d)^k n^{(k-1)})$ [15], [14]. Therefore, the number of interesting critical points is at most $O((2d)^k n^{(k-1)})$.

# References


[1] O. Khatib. Real-time obstable avoidance for manipulators and mobile robots. *IJRR*, 5(1):90–98, 1986.

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

[3] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.

[4] J. Reif. *Complexity of the Mover's Problem and Generalizations*, chapter 11, pages 267–281. Ablex publishing corp., New Jersey, 1987.

[5] J.T. Schwartz and M. Sharir. *On the 'Piano Movers' Problem, II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds*, chapter 5, pages 154–186. Ablex publishing corp., New Jersey, 1987.

[6] B. Langlois J. Barraquand and J-C. Latombe. Robot motion planning with many degrees of freedom and dynamic constraints. In *Proceedings 5th ISRR*, pages 74–83, Tokyo, Japan, 1989.

[7] J. Canny. Computing roadmaps of general semi-algebraic sets. In *AAECC-91*, pages 94–107, 1991.

[8] J.F. Canny. Generalized characteristic polynomials. *Journal of Symbolic Computation*, 9(3), 1990.

[9] D. Manocha and J. F. Canny. Efficient teniques for multipolynomial resultant algorithms. *Proceedings of ISSAC'91*, 1991. Bonn, Germany.

[10] Canny and Rege. An efficient algorithm for computing perturbed polynomial systems. In preparation, 1992. University of California, Berkeley.

[11] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. *IEEE ICRA'91 Proceedings*, 2:1008–1014, 1991.

[12] C. G. Gibson K. Wirthmuller and A. A. du Plessis E. J. N. Looijenga. *Topological Stability of Smooth Mappings*. Springer-Verlag, Berlin . Heidelberg . New York, 1976.

[13] J. F. Canny. Constructing roadmaps of semi-algebraic sets I: Completeness. *Artificial Intelligence*, 37:203–222, 1988.

[14] J. Milnor. On the betti numbers of real varieties. *Proc. Amer. Math. Soc.*, 15:275–280, 1964.

[15] R. Thom. Sur l'homologie des varietes algebriques reelles. *Differential and Combinatorial Topology*, pages 255–265, 1965.