Sensor Based Planning and Nonsmooth Analysis

Howie Choset Joel Burdick
Dept. of Mechanical Engineering
Mail Code 104-44, CALTECH, Pasadena, CA 91125

Abstract. This paper describes some initial steps towards sensor based path planning in an unknown static environment. The method is a based on a sensor-based incremental construction of a one-dimensional retract of the free space. In this paper we introduce a retract termed the *Generalized Voronoi Graph*, and also analyze the Roadmap of Canny and Lin's Opportunistic Path Planner. The bulk of this paper is devoted to the application of nonsmooth analysis to the Euclidean distance function. We show that the distance function is in fact nonsmooth at the points which are required to construct the plan. This analysis leads directly to the incorporation of simple and realistic sensor models into the planning scheme.

1. Introduction

"Sensor Based Planning" incorporates sensor information, reflecting the current state of the environment, into a robot's planning process, as opposed to classical planning, which assumes full knowledge of the world's geometry prior to planning. Sensor based planning is important because: (1) the robot often has no a priori knowledge of the world; (2) the robot may have only a coarse knowledge of the world because of limited memory; (3) the world model is bound to contain inaccuracies which can be overcome with sensor based planning strategies; and (4) the world is subject to unexpected occurrences or rapidly changing situations.

There already exists a large body of path planning literature; see [Lat91] and references contained therein. However, many of these techniques are not amenable to sensor based interpretation. It is not possible to simply add a step to acquire sensory information, and then construct a plan from the acquired model using a classical technique, since the robot needs a path planning strategy in the first place to acquire the world model. Instead, an incremental approach is needed. Incremental and sensor based planning algorithms have been developed for two dimensions. See [RI91] for an example of an incremental Voronoi diagram construction technique. Lumelsky's bug algorithm [LS87] is another sensor based planning strategy which is guaranteed to reach a goal in a two-dimensional world, but not in higher dimensions.

In this paper, we describe some initial steps towards path planning in a static environment where there is no a priori knowledge. We develop an incremental method to construct a *Generalized Voronoi Graph* (GVG), which is a 1-dimensional retract of a bounded space. Much of the analysis in this paper can also be applied to "sensorize" other methods based on a retract, such as the Opportunistic Path Planner (OPP) described in [CL93]. In constructing the retract from sensor data, we only assume that the robot has a dead reckoning system and on board sensors that measure distance and direction to nearby obstacles. This planning scheme can be used in two ways. First, it will find a path from an initial location to a goal if such a path exists. Second, the method can be used to construct a 1-dimensional retract of a bounded space.

The principal focus of this paper is not the Generalized Voronoi Graph or any particular retract, such as the Canny Roadmap. Instead, most of this paper is devoted to the application of nonsmooth analysis to the Euclidean distance function. This function is an integral component of many path planners, in addition to the one described in this paper. Prior work has not fully considered the important issues of nonsmoothness when they employ this distance function. We show that this function is in fact *always* nonsmooth at the points which are required for constructing the plan. Furthermore, we show that this analysis leads to the incorporation of simple and realistic sensor models into the planning scheme. We give the first rigorous basis for sensor-based construction of the retract fragments which are required for planning. Simulations demonstrate the validity of the approach. Experiments are currently under way.

2. Relation to Previous Work

Many successful classical motion planning methods are based on the construction of a 1-dimensional retract of the free configuration space, $\mathcal{F}$. For example, we have in mind the "Opportunistic Path Planner" (OPP) of Canny and Lin [CL90], [CL93], which is itself based on Canny's Roadmap Algorithm [Can88]. One-dimensional retracts have the nice properties of *accessibility* and *departability.* That is, the planner can construct a path between any two points in $\mathcal{F}$ by first finding a path onto the retract (accessibility), traversing the retract to the vicinity of the goal, and then constructing a path from the retract to the goal (departability).

As an example of these retract methods, we review the OPP and point out its limitations for sensor based use. Assume the configuration space, $\mathcal{C}$, is $\mathbb{R}^m$, with coordinates $(x_1, \cdots, x_m)$. As defined in [CL90], [CL93], a *slice* is the subset of $\mathcal{C}$ defined by the plane $x_1 = \lambda$. On each slice, a continuous and differentiable artificial potential function is defined. Canny and Lin suggest that the Euclidean distance function between a given point (which represents the robot's configuration) on the slice and the nearest obstacle be used as the potential function. The loci of the maxima of the potential function are termed *ridge curves* [RC94]. These are the maximally safe paths for the robot.

The algorithm works as follows. A path is traced from the start and from the goal onto the nearest ridge curves by gradient ascent on the potential function in the slices which intersect the start and goal positions. The start and goal ridge curves are subsequently constructed by sweeping a slice (slicing) through the environment while tracing the local maxima of the potential function. If the start and goal ridge curves are connected, then the algorithm terminates. In general, the set of ridge curves will not be connected, and paths between neighboring ridge curves must be found. The OPP proposes a method to connect ridge curves with *bridge curves*. The bridge curves are constructed in the vicinity of *interesting critical points*. Interesting critical points occur when c-space *channels* (Figure 1) join or split. If the goal and start ridge curves do not connect, the OPP adds a bridge curve, and the process is repeated until the start and goal curves are connected, thereby finding a path from the start to the goal. If all interesting critical points are explored, and the goal and start do not connect, then no path exists. Note that the union of bridge and ridge curves forms the *skeleton*, or 1-dimensional retract.

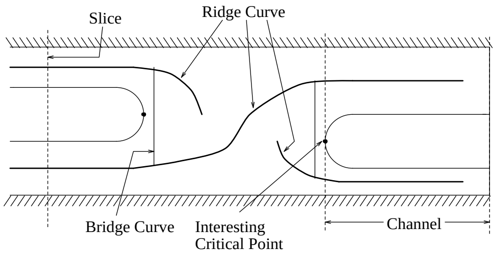

Figure 1: Schematic of the OPP planning scheme

The OPP can not be directly implemented in a sensor-based way because it assumes: (1) prior knowledge of the location of all the interesting critical points; and (2) that ridge curves can be traced backward from the goal. Rimon and Canny [RC94] have recently suggested a way to "sensorize" the OPP algorithm. The principle contribution of [RC94] is a study of the interesting critical points, which are crucial to construction of the retract. They introduce the notion of a "critical point sensor," though they do not suggest how to construct the interesting critical point sensor. Nor do they provide a rigorous way to construct the ridge curve fragments from sensor data. Both works do not consider the issues of nonsmoothness of the distance unction. As we show below, the Euclidean distance function is *not* differentiable on the ridge curves. Furthermore, the interesting critical points, which are crucial to the construction of the retract, are sweep direction dependent–this is an undesirable property.

The Generalized Voronoi Graph (GVG) retract introduced in Section 6 has some advantages over the skeleton of the OPP method. However, we first digress to develop some ideas that are necessary to define the GVG as well as analyze other retracts, such as the one used in the OPP.

3. Review of Nonsmooth Analysis

We show below that the Euclidean distance function is in fact nonsmooth at many points of interest, and therefore does not have a conventional derivative at these points. However, one can build a calculus for such nonsmooth functions from a less restrictive class of assumptions than smoothness: Lipschitz, regularity, and convexity. We review here some essential results from nonsmooth analysis and develop a few useful results. A more comprehensive treatment of nonsmooth analysis can be found in [Cla90]. Throughout this section, assume all $\vec{x} \in X$ where X is a finite dimensional vector space.

Definition 3.1: A function $f(\vec{x})$ is *Lipshitz near* $\vec{x}$ when:

$$\|f(\vec{x}_i) - f(\vec{x}_j)\| \leq K\|\vec{x}_i - \vec{x}_j\| \quad \forall \vec{x}_i, \vec{x}_j \in Nbhd(\vec{x})$$

where K is some scalar.

Definition 3.2: *The generalized directional derivative* ([Cla90], p10) of $f(\vec{x})$ in the direction $\vec{v}$ is:

$$f^{\circ}(\vec{x};\vec{v}) = \limsup_{\vec{y}\to\vec{x},t\to 0+} \frac{f(\vec{y}+t\vec{v}) - f(\vec{y})}{t}$$

Definition 3.3: $f(\vec{x})$ is *regular* ([Cla90], p39) at $\vec{x}$ when: (1) $\forall \vec{v}, f^{'}(\vec{x},\vec{v})$ exists, where $f^{'}$ is the usual one sided derivative; and (2) $\forall \vec{v}, f^{'}(\vec{x},\vec{v}) = f^{\circ}(\vec{x},\vec{v})$.

Definition 3.4: f is a *convex function* if $\forall \vec{x}_i, i = 1..n$ and $\sum_{i=1}^{n} \lambda_i = 1$

$$f\left(\sum_{i=1}^{n} \lambda_i \vec{x}_i\right) \leq \sum_{i=1}^{n} \lambda_i f(\vec{x}_i)$$

All convex functions are regular ([Cla90], p40).

While a Lipschitz function need not be smooth, it does possesses a *generalized derivative*, or *generalized gradient*. This definition is key to the remainder of this paper.

Definition 3.5: *In finite dimensional space, the generalized gradient* ([Cla90], p63) of a Lipschitz function f at $\vec{x}$ is denoted by $\partial f(\vec{x})$ and given by:

$$\partial f(\vec{x}) = co\{\lim_{\vec{x}_i\to\vec{x}} \nabla f(\vec{x}_i) : \vec{x}_i \notin S,\, \vec{x}_i \notin \Omega_f\}$$

where Ω_f is the set of points where f fails to be differentiable, S is any set of measure zero, and co means convex hull. Note that if $f(\vec{x})$ is smooth at $\vec{x}$, then $\partial f(\vec{x})$ reduces to the conventional gradient.

We now introduce some properties that are useful for manipulating the generalized gradient.

Proposition 3.6:

$$-co\{\vec{x}_i : \quad i = 1...n\} = co\{-\vec{x}_i : \quad i = 1...n\}$$

Proof: $-(co\{\vec{x}_1, ..., \vec{x}_n\}) = -\sum_{i=1}^n \lambda_i \vec{x}_i = \sum_{i=1}^n -\lambda_i \vec{x}_i = \sum_{i=1}^n \lambda_i(-\vec{x}_i) = co\{-\vec{x}_1, \cdots, -\vec{x}_n\}$ ■

Proposition 3.7:

$$\partial(sf)(\vec{x}) = s\partial f(\vec{x}) \quad \forall s \in \mathbb{R}$$

In particular, note that $\partial(-f) = -\partial(f)$.

It will be useful in the sequel to analyze functions which are described as the maxima or minima over a set of Lipschitz functions.

Proposition 3.8: Let $\{f_i\}$, $i = 1, \cdots, n$, be a set of functions which are Lipschitz near $\vec{x}_0$. For $\vec{x} \in Nbhd(\vec{x}_0)$, the function

$$f(\vec{x}) = \max_{i=1,...,n} \{f_i(\vec{x})\} \tag{3.1}$$

is also Lipschitz and regular.([Cla90],p47)

Since (3.1) is Lipschitz, we can define its generalized gradient.

Proposition 3.9: *Pointwise Maxima.* For the function $f(\vec{x}) = \max_{i=1,...,n}\{f_i(\vec{x})\}$, where each$f_i(\vec{x})$ is regular, $f(\vec{x})$ has the generalized gradient:

$$\partial f(\vec{x}) = co\{\partial f_i(\vec{x}) : i \in I(\vec{x})\} \tag{3.2}$$

where $I(\vec{x})$ is the set of indices for which $f_i(\vec{x}) = f(\vec{x})$. That is, $f_i(\vec{x})$ attains the maximum $\forall\ i \in I(\vec{x})$.

Proposition 3.10: *Pointwise Minima.* For a set of regular functions $f_i(\vec{x})$, the function

$$g(\vec{x}) = \min_{i=1..n} (f_i(\vec{x})) = -\max_{i=1..n}(-f_i(\vec{x})) \tag{3.3}$$

has generalized gradient:

$$\partial g(\vec{x}) = co\{\partial f_i(\vec{x}) \quad \forall i \in I(\vec{x})\} \tag{3.4}$$

where $I(\vec{x})$ is the set of indices for which $f_i(\vec{x}) = g(\vec{x})$, i.e. where $f_i(\vec{x})$ attain the minimum. Like above, g is regular. The proof is a simple consequence of Props. 3.6 and 3.7.

4. Analysis of the Distance Function

As seen in Section 2, a function which encodes the distance between the robot and nearby obstacles is key to the construction of the skeletons of the OPP retract. It is also key to the definition of the Generalized Voronoi Graph of Section 6. This section defines several important distance functions and uses the contents of Section 3 to analyze them. We assume a point robot operating in an m-dimensional Euclidean space, $\mathcal{W}$, which is populated by obstacles $C_1, \cdots, C_n$, and which can be described as sets of points. We often assume that the obstacles are convex. Non-convex obstacles can be modeled as the union of convex shapes. The case in which the robot is not modeled as a point, but as a set, is not considered in this paper.

First we consider the distance between a point robot and an obstacle in $\mathbb{R}^m$.

Definition 4.1: *Single Object Distance Function.* The distance between a point, $\vec{x}$ and a set C_i is

$$d_i(\vec{x}) = \inf_{\vec{q} \in C_i} \|\vec{x} - \vec{q}\| \tag{4.1}$$

where $\|\cdot\|$ is the 2-norm in $\mathbb{R}^m$.

Recall from Section 2 that in order to incrementally construct the skeleton curve, we wish to extremize the distance function on a *slice*. A slice is a set of points $\{\vec{x} : \alpha(\vec{x}) = \lambda\}$ where λ is a scalar and $\alpha(\cdot)$ is a function which foliates $\mathcal{W}$. For now, we assume that a slice is a hyperplance, and that the coordinates are chosen so that $\alpha(\vec{x}) = x_1$. In this case, we can decompose the physical space coordinates $\vec{x}$ into "slice coordinates $\vec{y}$ and the "sweep coordinate" λ: $\vec{x} = (\lambda,\ \vec{y}^T)^T$.

Definition 4.2: *Single Object Distance constrained to a slice, λ.* The distance between a point, which is constrained to a slice, λ, and a set C_i is:

$$\tilde{d}_i(\vec{y}; \lambda) = d_i(\vec{x})\Big|_{\alpha(\vec{x})=\lambda} = \inf_{\vec{q} \in C_i} \|\vec{y} - \vec{q}\|$$

where $\vec{y} \in \alpha^{-1}(\lambda)$. Hereafter, $\tilde{d}_i$ is shorthand for $\tilde{d}_i(\vec{y}; \lambda)$.

See figure 2 for an example of the distance function plotted along a slice. At each slice point, $\tilde{d}_i$ is computed to the nearest point of the obstacle.

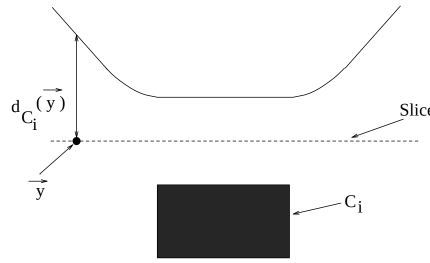

Figure 2: Distance function plotted along a horizontal slice

Typically, the world is populated with multiple obstacles, and thus we define:

Definition 4.3: *Multi-object Distance Function.* The distance between a point and many obstacles is taken as the minimum distance to any obstacle:

$$D(\vec{x}) = \min_{i=1,\cdots,n} d_i(\vec{x})$$

$D(\vec{x})$ can be found with realistic sensors.

Definition 4.4: *Multi-object Distance Function constrained to a slice*

$$\tilde{D}(\vec{y}; \lambda) = \min_{i=1,\cdots,n} \tilde{d}_i(\vec{y}; \lambda) \quad \text{where} \quad \vec{y} \in \alpha^{-1}(\lambda).$$

This is the function which is extremized to generate the ridge curves of the OPP, or the retract fragments of the GVG. See figure 3 for an example of $\tilde{D}$ plotted along a slice.

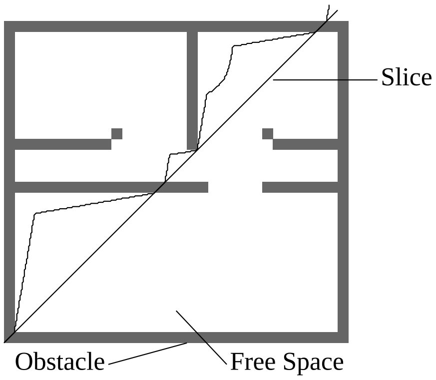

Figure 3: Distance function plotted along a diagonal slice

We now consider how to compute the gradients of these distance functions.

Proposition 4.5: The single object distance function satisfies:

$$\|d_i(\vec{x}) - d_i(\vec{y})\| \leq \|\vec{x} - \vec{y}\|.$$

and so is *Lipschitz* ([Cla90],p51). d_i is a convex function if C_i is a convex set. Since convex obstacles are assumed, d_i is convex, and hence it is regular ([Cla90],p40). And, by Prop. 3.10, D is therefore regular as well.

We now consider how to compute the gradient of d_i. When there is a **unique** closest point $\vec{c}_0 \in C_i$ to $\vec{x} \notin cl(C_i)$, then ([Cla90], p66):

$$\nabla d_i(\vec{x}) = \frac{\vec{x} - \vec{c}_0}{\|\vec{x} - \vec{c}_0\|}. \tag{4.2}$$

That is, ∇d_i is a unit vector emanating from $\vec{x}$ pointing away from $\vec{c}_0$. Thus, if C_i is convex, then d_i is smooth everywhere but on the boundary of C_i. To compute (4.2) from sensor data, one need only know the distance to the nearest point on an obstacle, and direction to that point.

But, d_i *is not smooth* when there are multiple closest points. Such a case would occur if C_i were not convex, and $\vec{x}$ is equidistant from multiple points on the boundary of C_i. If $\vec{x}$ is equidistant to a set of points $\{\vec{c}_j\}$, then by Definition 3.5:

$$\partial d_i(\vec{x}) = co\left\{ \frac{\vec{x} - \vec{c}_j}{\|\vec{x} - \vec{c}_j\|} : \quad \forall \vec{c}_j \right\} \tag{4.3}$$

If C_i is convex, ∂d_i evaluates to ∇d_i. Likewise, since the multi-object distance function is a function of the form (3.3) where each d_i is regular, Prop. 3.10 states that its generalized gradient will have the form:

$$\partial D(\vec{x}) = co\{\nabla d_i(\vec{x}) \quad i \in I(\vec{x})\} \tag{4.4}$$

Recall that $I(\vec{x})$ is the set of indices where $d_i(\vec{x}) = D(\vec{x})$. The physical intuition is:

1. If there is a unique closest point (and hence a unique closest obstacle), $\partial D(\vec{x})$ is a unit vector pointing away from the closest point. In this case, $D(\vec{x})$ is smooth.
2. If there are a set of equidistant closest points, then $\partial D(\vec{x})$ is the convex hull of each of the gradients with respect to each point. In this case, the robot is equidistant to multiple convex obstacles.

5. Creating Ridge Fragments from Sensor Data

To implement a sensor based incremental construction of a retract, we must compute the gradient of our distance measurements directly from sensor data. In particular, for the OPP ridge curves, the local maxima of $\tilde{D}(\vec{y};\lambda)$ needs to be found on each slice. In order to extremize $\tilde{D}(\vec{y};\lambda)$ on a slice, we must compute its gradient with respect to the slice variables, $\vec{y}$. In this section we show how sensor data can be used to compute this gradient, and how to reliably find and differentiate between the different required extremal points on each slice.

We want to compute the generalized gradient of $\tilde{D}(\vec{y};\lambda)$ with respect to the slice variables, $\vec{y}$. However, our sensors give us data which can be used to construct the generalized gradient of $D(\vec{x})$ in the ambient space. This difference can be resolved as follows.

Proposition 5.1: The projection of $\partial D(\vec{x})$ onto the $\vec{y}$ subspace is equal to the partial gradient of $D(\vec{x})$ with respect to $\vec{y}$:

$$\pi_{\vec{y}}(\partial D(\vec{x})) = \partial_{\vec{y}} D(\vec{x})$$

where π_y projects onto the $\vec{y}$ subspace and $\partial_{\vec{y}}$ represents partial differentiation with respect to $\vec{y}$.

Proof: First recall that for smooth functions

$$\nabla_{\vec{x}_1} f(\vec{x}_1, \vec{x}_2) = \pi_{\vec{x}_1}(\nabla f(\vec{x}_1, \vec{x}_2))$$

If there is a unique closest point, then $D(\vec{x})$ is smooth at $\vec{x}$, and the proposition is proved. If there is not a unique closest point, by (4.4):

$$\partial D(\vec{x}) = \sum_{i \in I(\vec{x})} \lambda_i \nabla d_i(\vec{x}) \quad s.t. \sum_{i \in I(\vec{x})} \lambda_i = 1 \quad \lambda_i > 0$$

Now project this generalized gradient onto the $\vec{y}$ coordinates:

$$\begin{aligned}\pi_{\vec{y}}(\partial D(\vec{x})) &= \pi_{\vec{y}}\Big(\sum_{i \in I(\vec{x})} \lambda_i \nabla d_i(\vec{x})\Big) \\ &= \sum_{i \in I(\vec{x})} \pi_{\vec{y}}(\lambda_i \nabla d_i(\vec{x})) = \sum_{i \in I(\vec{x})} \lambda_i \pi_{\vec{y}} \nabla d_i(\vec{x}) \\ &= \sum_{i \in I(\vec{x})} \lambda_i \nabla_{\vec{y}}(d_i(\vec{x})) = \partial_{\vec{y}}(D(\vec{x})) \qquad \blacksquare\end{aligned}$$

Thus, it is quite straightforward to compute $\partial_{\vec{y}}\tilde{D}(\vec{y},\lambda)$ from simple distance sensor data.

A local maximum of $\tilde{D}$ is found as follows. Assume a point robot is located at $\vec{x} = (\lambda, \vec{y})$. $\partial D(\vec{x})$ can be computed using sensor data. These gradients are then projected onto the slice coordinates to yield $\partial_{\vec{y}}\tilde{D}$. If the robot is not located at a maximum or inflection point, then $\tilde{D}$ is smooth and $\partial_{\vec{y}}\tilde{D}$ consists of a single vector gradient. The robot does gradient ascent until it reaches a local maximum. From there, the robot increments forward in the sweep direction, and re-optimizes $\tilde{D}$ on the new slice. In this way, the ridge curves are constructed. The inflection curves are constructed in a similar way.

An important question which must now be addressed is: how do we reliably determine if we are at a maxima, minima, or inflection point on a slice? For the local maxima, some of the local minima, and inflection points, $D(\vec{x})$ and $\tilde{D}(\vec{x})$ are not smooth. Thus, unlike the case of smooth functions, we can not use the vanishing of $\tilde{D}$'s gradient as an indication of an extremal point. However, as the following propositions point out, it is possible to differentiate between the extremals. To our knowledge, these results are new to the nonsmooth analysis literature. They are equivalent to the Hessian, or curvature conditions, which classify the extremal points of smooth functions.

PROPOSITION 5.2: *Local Extrema* ([Cla90], p38) If f attains a local minima or maxima at $\vec{x}$, then $0 \in \partial f(\vec{x})$. Unfortunately, the converse is not always true.

Proposition 5.3: *Let $\vec{x}_* = (\lambda, \vec{y}_*^T)^T \in \mathbb{R}^m$ be equidistant from obstacles $C_1, \cdots, C_n$, where $n \geq m$. That is, $d_1(\vec{y}_*;\lambda) = \cdots = d_n(\vec{y}_*;\lambda) = D(\vec{y}_*;\lambda)$. If $0 \in int(\partial_{\vec{y}}D(\vec{y}_*;\lambda)) = int(\pi_{\vec{y}}\partial D(\vec{y}_*;\lambda))$ then $\vec{y}_*$ is a local maximum.*

Proof: To prove this proposition, we first prove:

Lemma 5.4: If $\vec{0} \in int(co\{\pi_{\vec{y}}\nabla d_i, i = 1,\ldots,n\}) = int(\partial_{\vec{y}}D(\vec{y}_*,\lambda))$, then there exists at least one $i \in \{1,\ldots,n\}$ such that inner product $\langle \vec{v}, -\pi_{\vec{y}}\nabla d_i\rangle > 0$ in an ϵ neighborhood of $\vec{y}_*$ for a fixed and arbitrary $\vec{v} \in \mathbb{R}^{m-1}$

Proof: Assume that there is no i for which $\langle \vec{v}, -\pi_{\vec{y}}\nabla d_i\rangle > 0$. Therefore, $\vec{v}$ defines a half space $H^+ = \{\vec{u} \in \mathbb{R}^m : (\vec{u},\vec{v}) > 0\}$. By the assumption, none of the vectors $\{-\pi_{\vec{y}}\nabla d_i\}$ lies in H^+. Therefore, all vectors $\{-\pi_{\vec{y}}\nabla d_i\}$ are in the same half space. Thus, the interior of the convex hull of the vectors $\{-\pi_{\vec{y}}\nabla d_i\}$ does not contain the origin. This is a contradiction, proving the above lemma. ▼

Since $\langle \vec{v}, -\pi_{\vec{y}}\nabla d_i\rangle > 0$ for at least one i, d_i decreases in the direction of $\vec{v}$ in an ϵ neighborhood. Since this is true for all $\vec{v}$,there is always a d_i that decreases in any direction $\vec{v}$. Therefore, for some $\epsilon > 0$

$$d_i(\vec{y}_* + \epsilon\vec{v}) < d_i(\vec{y}_*) = D(\vec{y}_*)$$

By definition, $D(\vec{y}_* + \epsilon\vec{v}) \leq d_i(\vec{y}_* + \epsilon\vec{v})$, thus

$$D(\vec{y}_* + \epsilon\vec{v}) < D(\vec{y}_*) \quad \forall\epsilon, \vec{v}$$

which implies that $D(\vec{y}_*)$ is a local maxima. ■

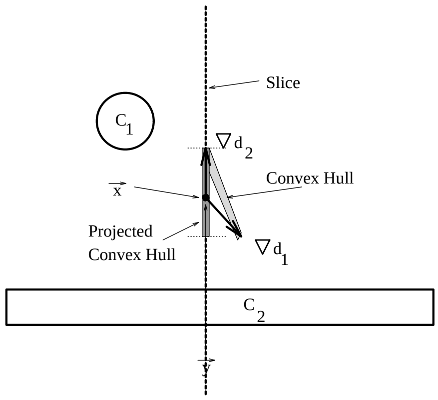

Figure 4: Local Maximum

The conditions for local minima and inflection points are similar, and can be proven in a similar way.

Proposition 5.5: *At an inflection point, $\tilde{D}$ is nonsmooth, and $0 \in boundary(\partial_{\vec{y}}D(\vec{y}_*;\lambda)$.*

Proposition 5.6: *At a local minima of $\tilde{D}$, $0 = \partial_{\vec{y}}D$.*

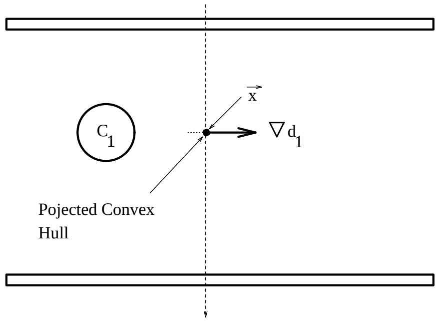

Figure 5: Local Minimum

Figure 5 shows an example of a local minima. Generically, D is smooth at local minima. We term a connected local minima curve a *valley curve*. The valley curves are *not* an essential part of the skeleton system we compute in Section 8, though they are often useful in practice.

The OPP skeleton is the union of ridge curves and bridge curves. Notice that a necessary condition for x to be a local maxima of $\tilde{D}$, is that x must be equidistant to at least m points in an m-dimensional space.

6. Generalized Voronoi Graph

This section defines the *Generalized Voronoi Graph* (GVG), and shows that on it, $D(\vec{x})$ and $\tilde{D}(\vec{y};\lambda)$ is nonsmooth. The GVG is a 1-dimensional retract with many useful properties for sensor based planning Since the ridge curves of the OPP method are fully contained in the GVG, $D(\vec{x})$ and $\tilde{D}(\vec{y};\lambda)$ are nonsmooth on the OPP ridge curves, as well.

Definition 6.1: $\Gamma_{ij} = \{\vec{x} \in \mathbb{R}^m : d_i(\vec{x}) - d_j(\vec{x}) = 0\}$ is the set of points which are equidistant to obstacles i and j

Definition 6.2: $\gamma_{ij} = \{\vec{x} \in \Gamma_{ij} : d_i(\vec{x}) \leq d_k(\vec{x})\, \forall k \neq i,j\}$ is the set of points equidistant to obstacles i and j, such that each point x is closer to i and j than any other obstacle.

Definition 6.3: $\Gamma^2 = \bigcup_{i=1}^{n-1} \bigcup_{j=i+1}^{n} \Gamma_{ij}$

Definition 6.4: $\gamma^2 = \bigcup_{i=1}^{n-1} \bigcup_{j=i+1}^{n} \gamma_{ij}$. It can be shown that γ^2 is the Voronoi Diagram.

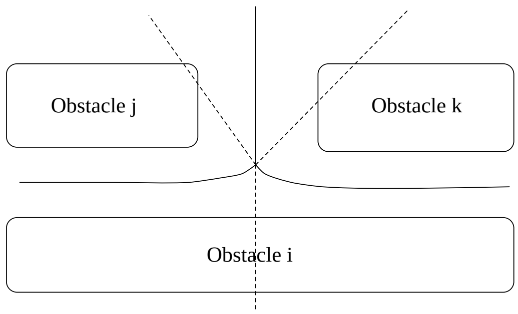

Figure 6: Set of Points Equidistant between any Two Obstacles. The solid lines are the set γ^2, while the union of the solid and dashed lines is the set Γ^2.

Definition 6.5: $\Gamma_{ijk} = \Gamma_{ij} \bigcap \Gamma_{ik} \bigcap \Gamma_{jk}$ is the set of points equidistant to objects i, j and k. By transitivity, we only need two Γ's, so $\Gamma_{ijk} = \Gamma_{ij} \bigcap \Gamma_{ik}$.

Definition 6.6: $\gamma_{ijk} = \gamma_{ij} \bigcap \gamma_{ik} \left(\bigcap \gamma_{jk}\right)$ is the set of points equidistant to objects i, j and k, such that each point is closer to i, j and k than any other object. Again, by transitivity, only two γ's are required to define γ_{ijk}.

Definition 6.7: $\gamma^3 = \bigcup_{i=1}^{n-2} \bigcup_{j=i+1}^{n-1} \bigcup_{k=j+1}^{n} \gamma_{ijk}$

The following definition is a result of taking $m-1$ intersections:

Definition 6.8: *The Generalized Voronoi Graph (GVG) is equal to* γ^m. γ^m is the set of points equidistant to m objects, such that each point is closer to the m objects than any other object. The GVG is a connected 1-dimensional retract with the properties of accessibility and departability (the proof of this claim is beyond the scope of this paper). The ridge curves of the OPP method are contained in γ^m since all ridge curve points are equidistant to at least m objects.

Proposition 6.9: $\tilde{D}$ is nonsmooth on the ridge curves.

Proof: By Definition 4.4 and Proposition 3.10, $\tilde{D}$ is nonsmooth on γ^m. Since the ridge curves are fully contained in γ^m, $\tilde{D}$ is nonsmooth on the ridge curves. ■

Proposition 6.10: The ridge curves are a one dimensional set.

Proof: The above proposition is a result of the following three Lemmas.

Lemma 6.11: γ^2 is co-dimension one in $\mathcal{W}$

Proof: Again, we assume non-intersecting convex obstacles, and note that $(d_i - d_j)$ is smooth by the obstacle convexity assumption. We now show that 0 is a regular value of $(d_i - d_j)(\vec{x})$, which is true if and only if $\nabla(d_i - d_j)(\vec{x})$ is surjective, i.e. $\nabla d_i(\vec{x}) \neq \nabla d_j(\vec{x})$ $\forall\ \vec{x} \in \Gamma_{ij}$. Since $\nabla d_i(\vec{x})$ and $\nabla d_j(\vec{x})$ are each unit vectors, they are only equal when they point in the same direction. However, since $i \neq j$ (i.e. they point at different obstacles non-intersecting), and $\vec{x}$ is a point equidistant to obstacles i and j, $\nabla d_i(\vec{x})$ and $\nabla d_j(\vec{x})$ can never point in the same direction. Therefore $\nabla d_i(\vec{x}) \neq \nabla d_j(\vec{x})$. Otherwise, obstacles i and j would occupy the same physical space.

By the pre-image theorem, since 0 is a regular value of the smooth function $(d_i - d_j)(\vec{x})$, Γ_{ij} is a manifold of co-dimension 1 in $\mathcal{W}$. Since γ_{ij} is a subset of Γ_{ij}, it too is a manifold of co-dimension 1. γ^2 is a set (not necessarily a manifold) of of co-dimension 1 because it is the finite union of sets of co-dimension one. ▼

Lemma 6.11 proves Prop. 6.10 is true in two dimensions. So, the following proposition generalizes the above result to three dimensions, from which we can generalize to m dimensions.

Lemma 6.12: γ^3 is co-dimension 2 in $\mathcal{W}$.

Proof: Another definition is: $\Gamma_{ijk} = \{\vec{x} \in \Gamma_{ij} : d_i(\vec{x}) - d_k(\vec{x}) = 0\}$. It is assumed that $\Gamma_{ij} \neq \Gamma_{ik} \iff i \neq k$. Therefore, 0 is a regular value of $(d_i - d_k)(\vec{x})$ on Γ_{ij}. By the pre-image theorem, Γ_{ijk} has co-dimension 1 in Γ_{ij}, and thus co-dimension 2 in $\mathcal{W}$. γ_{ijk} is a subset of Γ_{ijk} and thus is co-dimension 2 on $\mathcal{W}$. Since γ_3 is the finite union of co-dimension 2 manifolds, it is co-dimension 2 on $\mathcal{W}$. ▼

Lemma 6.13: γ^m is co-dimension $m-1$, that is, one dimensional, in $\mathcal{W}$.

Proof: Generalizing the above assumptions to higher dimensions, by induction, one can show that Γ^m is one dimensional in $\mathcal{W}$ and has co-dimension 1 in Γ^{m-1}. Since γ^m is fully contained in Γ^m, it too is is co-dimension $m-1$ and one dimensional in $\mathcal{W}$. ▼

Since the ridge curves in an m-dimensional space are fully contained in γ_m, they are one dimensional. ■

Definition 6.14: *Meet Points* are elements of γ^{m+1}

That is, the Meet Points are where two or more local fragments of the GVG meet in a point. It can be shown that the Meet Points are "sweep invariant."

7. Detecting the Meet Points

As a result of Prop. 6.12, the ridge curves in an $m-$ dimensional space are locally connected (smooth) segments equidistant to m obstacles. These locally smooth segments (termed *ridge fragments*) meet at points (termed *meet points*) equidistant to $m+1$ obstacles. See figure 7. To incrementally construct the GVG, it is sufficient to trace the local GVG fragments. When Meet Points are reached, the algorithm recursively explores each of the graph fragments that depart from the meet point. Thus, finding these meet points is a critical part of our algorithm. These meet points can be detected as follows. While tracing the ridge curves, the robot's sensors look at the set of m (two in the plane) closest obstacles. Due to sensor noise, and positioning inaccuracies, it is unreasonable to expect the robot to accurately detect that it is equidistant to $m+1$ points at a meet point. However, as the robot passes by a meet point, one of the m closest points changes. In other words, we moved from $\gamma_{i_1 \ldots i_{m-1} j}$ to $\gamma_{i_1 \ldots i_{m-1} k}$. This occurrence is easy to detect and robust. From this, one can compute a good estimate for the location of the meet point.

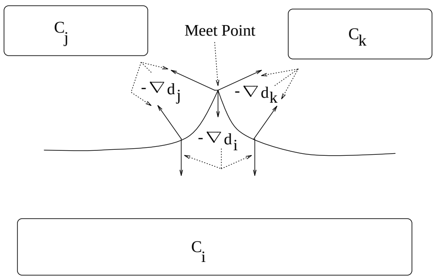

Figure 7: Meet point

In future work, we show that meet points are necessary for the incremental construction of the Generalized Voronoi Graph.

8. Simulation of the Method

We implemented the incremental GVG method in simulation for the planar case. Recall that there are two applications of the algorithm: (1) to find a path to a goal; or (2) to construct a retract of a bounded environment. We focus here on the more general case of retract construction. The method is illustrated by the example in Figures 8, 9 and 10.

The input to the algorithm is an initial sweep direction, and the initial coordinates of the robot. Like the OPP, the robot finds a local maximum (or possibly inflection in some non-generic cases) of $\tilde{D}$ via gradient ascent along the slice which is determined by the starting point and sweep direction. The methods of Section 5 are used to check and verify the extremal conditions. The location of the local maximum and the direction of the two nearest obstacles are stored.

The robot then begins to incrementally construct a retract fragment. Since there is a choice of two directions for tracing the fragment, one direction is arbitrarily chosen. The curve is locally traced by using a continuation method, which can be thought of as a sweeping method similar to the OPP, but where the sweep direction continually changes, and is defined by the tangent to the retract fragement curve. The fragment is traced until the robot reaches a boundary or a Meet Point (it can be shown that one of these two conditions will occur). If the robot reaches a boundary, it returns to the start to trace the retract frament in the other direction. Otherwise, it can mark the meet point and begin to explore the other retract fragments that depart from the Meet Point. In this regard, our algorithm departs significantly from the OPP in that at meet points, a new sweep direction is effectively determined. For the first stage, the robot may return to the starting point to sweep in the negative direction until it reaches a Meet Point or boundary. Figure 8 shows a snapshot of the computer simulation after this stage has been completed. The figure consists of the trace of the local maxima, and three Meet Points. The two on the ends were explored (diagonal lines), and the middle meet point remains to be explored.

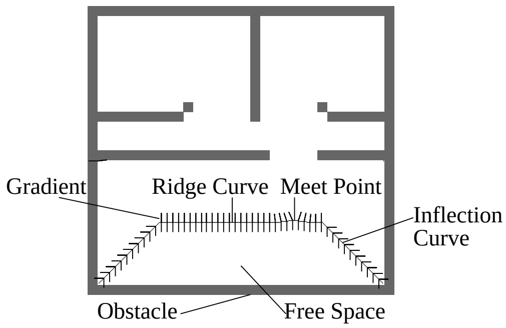

Figure 8: Computer simulation of first phase

The algorithm then continues recursively. The robot returns to a Meet Point with unexplored branches, and traces one of these branches. This tracing procedure continues until another Meet Point is reached (where another level of recursion is initiated), or until a boundary is reached. Figure 9 shows a subsequent snapshot of the simulation after the robot has moved through several Meet Points and reached a boundary.

After tracing a fragment to a boundary, the robot returns to an already detected Meet Point which has unexplored directions. Note that during this process, the robot might reach a previously explored Meet Point by closing a "loop." In such cases, the robot returns to the last previously unexplored Meet Point, and continues. The algorithm terminates when all retract fragments emanating from all Meet Points have been explored. The result is a complete retract of the bounded space which has been iteratively constructed with no a priori knowledge (Figure 10)

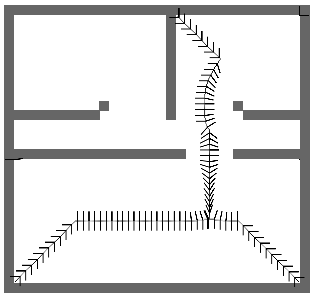

Figure 9: Second Snapshot

The advantage of this method is that the Meet Points are easily detected, and do not depend upon a sweep direction. In addition, as opposed to OPP's method of building bridge curves in the vicinity of the interesting critical points, the resulting GVG retract is maximally far from the obstacles at all times.

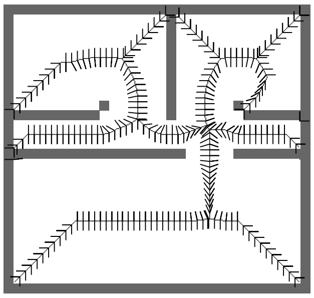

Figure 10: Final retract constructed by the method

9. Summary

The bulk of this paper was devoted to nonsmooth analysis of the Euclidean distance function. In particular, we showed how to obtain the local maxima of a nonsmooth function, entirely from first order information. We showed that this analysis leads naturally to a simple, robust, and rigorous methods to construct local retract curve fragments from sensor data. These result are useful for "sensorizing" other methods which have been proposed in the classical planning literature. In addition, the distance functions introduced in this paper lead naturally to a new 1-dimensional retract, which we call the Generalized Voronoi Graph. This retract has the nice property that its local fragments meet in easily detectable and invariant Meet Points. We proposed a novel method to incrementally construct the Generalized Voronoi Graph from distance sensor data. This method in turn can be used to construct a 1-dimensional retract of an unknown environment based solely on distance measurements. Simulation results validated the approach, and experiments are currently under way.

Acknowledgements The authors grateful acknowledge the support of the Office of Naval Research, Grant # N00014-93-1-0782. The authors would like to thank Jim Ostrowski, Andrew Lewis, Elon Rimon and Andrew Conley for their discussions. In particular, we would like to thank Jim Ostrowski for supplying a proof for Prop. 5.3.

[Can88] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[CL90] J.F. Canny and M.C. Lin. An opportunistic global path planner. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1554–1559, Cincinnati, Ohio, 1990.

[CL93] J.F. Canny and M.C. Lin. An opportunistic global path planner. *Algorithmica*, 10:102–120, 1993.

[Cla90] F. H. Clarke. *Optimization and Nonsmooth Analysis.* Society of Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[Lat91] J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[LS87] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving admist unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[RC94] E. Rimon and J.F. Canny. Construction of c-space roadmaps using local sensory data — what should the sensors look for? In *Proc. IEEE Int. Conf. on Robotics and Automation*, San Diego, CA, 1994. Recently Submitted.

[RI91] N. Rao, N.S.V. Stolzfus and S.S. Iyengar. A retraction method for learned navigation in unknown terrains for a circular robot. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA., 1991. 699-707.