Complete Topological Mapping with Sparse Sensing

Wesley H. Huang Kristopher R. Beevers
Rensselaer Polytechnic Institute
Department of Computer Science
110 8th Street, Troy, New York 12180, U.S.A.
{whuang,beevek}@cs.rpi.edu

Abstract

This paper describes algorithms for a mobile robot with sparse, short-range sensing to create a topological map of an unknown environment. While a limited array of sensors is appealing from the standpoint of having simpler and cheaper hardware, mapping is more difficult because the robot cannot guarantee it will detect obstacles as soon as they enter its sensing range. Thus, the robot's mapping strategy must ensure that all relevant parts of the environment are observed. Our approach constructs topological maps based on the SGVD$_\infty$, a version of the saturated generalized Voronoi diagram defined under the L_∞ distance metric, which is well-suited to robots with sparse sensing. We first describe behaviors that allow a robot with an omnidirectional short-range sensor to trace the SGVD$_\infty$, and then extend these behaviors to the sparse sensing case by introducing a "virtual sensor" that tracks unseen obstacles and emulates the output of the omnidirectional sensor. We show that our behaviors are complete for the problem of mapping an arbitrary rectilinear environment. We have implemented the mapping behaviors and report the results of simulated experiments.

Keywords: topological mapping, saturated generalized voronoi diagram, sensing limitations, sparse sensing

1 Introduction

Mapping is a fundamental capability for mobile robots, allowing them to communicate spatial information about an environment and navigate through that environment using efficient paths. While often done using powerful sensors such as scanning laser rangefinders, it is possible to map and navigate in an environment using a more limited array of sensors. Robots with limited sensing can be made very inexpensively, which qualifies them for applications where a large number of potentially disposable robots would be required, e.g. hazard assessment of a contaminated building, urban reconnaissance, security patrols, rescue operations, etc.

Another reason for our interest in sensing-limited robots is to ultimately address fundamental questions about mapping: what sensing is required for mapping; how limitations on sensing affect the ability to map, the quality of the resulting maps, the efficiency of subsequent navigation; and so on.

In this paper, we consider two types of sensing limitations: limited-range sensing and (additionally) sparse range sensing. We consider sensors such as the popular Sharp IR rangefinders, which have a maximum range of approximately 80 cm. In many office buildings, such a robot would not even be able to simultaneously sense both sides of a hallway! By "sparse sensing," we mean that the range to the closest obstacle is provided only along a small number of directions from the robot. Our robot model has eight range sensors at orientations of $n\frac{\pi}{4}$ for $n = 0, 1, \ldots, 7$.

Our focus is on mapping indoor spaces such as office buildings. We make the assumption that the environment consists of a (possibly nonconvex) rectilinear polygonal boundary and obstacles. Since most buildings are rectilinear in their basic structure, this is a reasonable assumption that enables us to formulate complete algorithms.

We create topological maps that are defined in terms of behaviors: the nodes in the map correspond to points in the environment where the behaviors terminate, and the edges correspond to a behavior (or sequence of behaviors) that moves the robot from one node to another. We have designed our behaviors so that the paths taken by the robot lie on the saturated generalized Voronoi diagram (Acar, Choset, and Atkar, 2001) under the L_∞ distance metric. This construct, which we refer to as the SGVD$_\infty$, contains paths that are either equidistant to two obstacles or are at the "saturation distance" from one obstacle. For this reason, it is well suited for robots using sensory feedback to follow paths. The L_∞ norm results in paths that are easier for a robot with sparse sensing to track and follow than those for the L_2 (Euclidean) norm. We describe the Voronoi diagram, generalized Voronoi diagrams, the SGVD$_\infty$, and the L_∞ norm in Section 3.

We first present mapping behaviors for a robot with a limited-range omnidirectional sensor in Section 4, and then for a robot with sparse limited-range sensing in Section 6. Our behaviors for sparse sensing are very similar to those for omnidirectional sensing. This is possible because of a "virtual sensor," described in Section 5, that monitors the inputs from the sparse sensors and maintains state, enabling it to compute an output that emulates that of the omnidirectional sensor.

Our behaviors are designed for an arbitrary rectilinear en-

vironment, even though real rectilinear environments would be far less complex. We show that these algorithms are complete in a sense analogous to graph search, where a complete algorithm visits every node in a connected component. Our algorithms are guaranteed to trace a connected component of the SGVD_∞. However, the robot is not guaranteed to map every area of the environment — this would be completeness in the "coverage" sense. In fact, no algorithm for coverage of an arbitrary environment by a robot with sparse sensing yet exists.

We have implemented our virtual sensor and sparse sensing behaviors in simulation, and we show several examples. We conclude in Section 7 with some observations on the tradeoffs for sensing limitations on mapping.

1.1 Assumptions

We consider a differential drive robot with a small number of short-range sensors. These sensors return the distance to the nearest obstacle along a straight line; they are "short-range" in that their maximum range is small compared to the dimensions of the environment.

There can be known error in the robot's movement, odometry, and sensing. The main problem that arises because of this error is that of "closing loops" in the map — the robot must recognize when it has returned to a place it has already been. We do not address the loop-closing problem in this paper. However, in previous work (Beevers and Huang, 2005), we have presented an evidential approach for solving the loop-closing problem that can be applied to robots with limited sensing.

We assume an enclosed, static, rectilinear environment. Rectilinearity is a strong assumption, but it removes uncertainty in the robot's orientation, and it allows us to focus on the fundamental problem of mapping with limited sensing information.

2 Related work

There are two traditional paradigms for robotic mapping: metric maps and topological maps. In metric maps, e.g. (Pagac, Nebot, and Durrant-Whyte, 1998; Thrun, 2002), the geometry of the world is explicitly represented, either through exact or approximate representations. In topological maps (Kuipers, 1978), "places" in the world (typically hallway junctions) are represented by vertices in a graph, and paths between places are represented by edges. Metric maps provide a detailed world representation but require more storage and are sensitive to measurement errors. Topological maps offer a more concise representation of the environment that is easy to use for purposes of navigation.

Some researchers have devised methods for extracting topological maps from occupancy grid data (Chatila and Laumond, 1985; Thrun and Bücken, 1996). Others have combined the topological and metric approaches by using "local" metric maps at nodes in topological maps (Duckett and Saffiotti, 2000; Thrun et al., 1998). For example, Tomatis, Nourbakhsh, and Siegwart (2002) use a topological map to represent a network of hallways and use metric maps to represent rooms. We focus primarily on topological mapping with limited-range sensors, which are well-suited to mapping in narrow corridors. However, we also map the perimeters of open spaces in the environment. Our use of the saturated generalized Voronoi diagram (Acar, Choset, and Atkar, 2001) to generate a map of the perimeter of an open area is related to the idea of "coastal navigation" (Roy and Thrun, 1999) which recognizes that areas near walls and obstacles have "high information content" due to the features they produce.

Once the robot has built a map of the boundaries of open areas, it can safely explore their interiors. Though we do not address that problem in this work, we have discussed methods elsewhere (Beevers, 2004; Huang and Beevers, 2004) that allow a robot with short-range sensing to improve the connectivity of its topological map in open spaces by "foraying" through the interiors of open areas.

Most research on topological mapping has focused on the online construction of topological maps (Kuipers and Byun, 1991; Rybski et al., 2003; Shatkay and Kaelbling, 1997; Tovar, LaValle, and Murrieta, 2003). Many researchers base their maps on the idea of "behaviors," simple control strategies that enable a robot to travel between "distinctive places" in an environment (Kuipers and Byun, 1991). In our approach, these distinctive places are meet points in the SGVD_∞ of an environment, which the robot is able to recognize based on several simple conditions. Topological maps constructed using behaviors generally consist of edges representing the execution of behaviors and nodes representing the distinctive places where the behaviors terminate.

Sensing capabilities affect a robot's ability to map an environment, and the mapping and exploration problems have been examined over a wide range of sensing modalities and limitations. Deng, Kameda, and Papadimitriou (1998) discuss an algorithm for observing all visible points in a rectilinear environment with an infinite-range omnidirectional sensor, and Rao (1989) and Rao and Iyengar (1990) have built topological maps based on the Voronoi diagram and the visibility graph using an infinite-range omnidirectional sensor. Grabowski et al. (1999) use "millibots" with short-range SONAR sensors to build metric maps. Choset (1997) and Choset and Nagatani (2001) have built topological maps by tracing Voronoi diagrams using a short-range omnidirectional sensor.

We instead build our maps by tracing the SGVD_∞ using sparse short-range sensing, constructing the corresponding graph as we go. Though Acar, Choset, and Atkar (2001) solve the coverage problem using the SGVD, they do so under the L_2 distance metric and with omnidirectional limited-range sensing. Other researchers have used different techniques to address mapping and exploration with sparse sensing. Butler, Rizzi, and Hollis (2001) describe coverage (equivalent to metric mapping) using robots with only contact sensors but with near-perfect odometry. Doty and Seed (1994) have shown preliminary results in creating a "landmark map" using a robot with four short-range infrared sensors and one long-range (and non-sparse) SONAR sensor, but it is not clear that their algorithm is

suitable for arbitrary environments.

Erdmann (1995) has examined the question of what sensing is required to perform certain tasks. Toward that end, he proposed a "top-down" design method for constructing sensors that provide exactly the information required to perform a task. Though Erdmann's work focuses mainly on manipulation tasks, the questions he asks are relevant to mapping. In this work, we show that mapping can be performed even with significant limitations on the robot's sensing capabilities.

3 Properties of the L_∞ norm and the SGVD$_\infty$

The basic mapping strategy followed by our robots is to trace the saturated generalized Voronoi diagram of the environment using the L_∞ distance metric (the SGVD$_\infty$). Meet points in the diagram constitute vertices in the topological map, and segments in the diagram are edges in the map. In this section, we define the SGVD$_\infty$ and introduce properties of the SGVD$_\infty$ and the L_∞ metric that are important to tracing the SGVD$_\infty$ with sparse, short-range sensing.

3.1 The Voronoi diagram, GVD, and GVD$_\infty$

The *Voronoi diagram* (Aurenhammer, 1991) is a fundamental structure in computational geometry. Given a set of points (called *sites*) in the plane, the Voronoi diagram can be defined as the locus of points equidistant to the two closest sites. Alternatively, the Voronoi diagram can be defined as the union of the boundaries of the *Voronoi regions* of the sites; the Voronoi region of a site is the set of points closer to that site than to any other. The Voronoi diagram can also be defined in higher dimensions, under different topologies, and with different distance metrics (Lee, 1980). When the sites are not points, the result is a *generalized Voronoi diagram* (GVD) (Kirkpatrick, 1979; Lee and Drysdale, 1981).

We will define a version of the GVD under the L_∞ metric that we refer to as the GVD$_\infty$. The GVD$_\infty$ will be defined for axis-aligned rectilinear polygons that are represented by sites that are open line segments. Our GVD$_\infty$ is slightly different than the usual GVD under the L_∞ metric in the way we define bisectors and half planes.

The L_∞ distance metric defines the distance between two points $\mathbf{p}$ and $\mathbf{q}$ as:

$$d(\mathbf{p}, \mathbf{q}) = \max_i |p_i - q_i| \tag{1}$$

Note that under the L_∞ norm, the locus of points equidistant from a given point (a circle under L_2) is an axis-aligned square under L_∞. Bisectors between two points under L_∞ may consist of multiple segments or even include unbounded regions. Our definition of a bisector differs from the usual approach taken in defining a Voronoi diagram under L_∞ because we include all equidistant points, even though this results in a bisector that is not always one dimensional; the usual approach is to choose a one dimensional curve, typically one of the boundary curves. The bisector between two line segment sites under L_∞ has similar properties to that between two points; Figure 1 shows several examples.

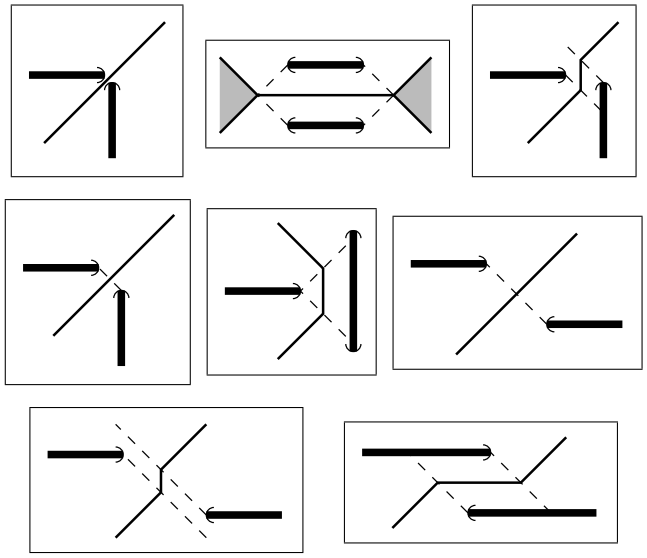
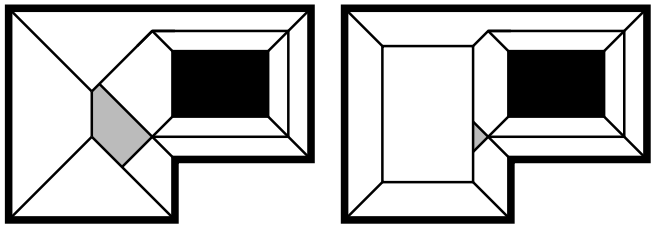


Figure 1: Bisectors of two open-ended line segments under the L_∞ metric.



GVD$_\infty$ SGVD$_\infty$

Figure 2: Examples of the GVD$_\infty$ and SGVD$_\infty$. Shaded areas are regions of points equidistant to two or more walls.

Following (Aurenhammer, 1991), we define a "half plane" for a pair of sites $s_i, s_j \in \mathcal{S}$, where $\mathcal{S}$ is the set of all sites. Our half plane is the *open* set of points on one side of a bisector, i.e.:

$$h(s_i, s_j) = \{\mathbf{r} \in \mathcal{R}^2 \mid d_\infty(\mathbf{r}, s_i) < d_\infty(\mathbf{r}, s_j)\} \tag{2}$$

This is not a half plane in the traditional sense because it is not necessarily bounded by a straight line. Also, the boundary of a half-plane may be a proper subset of a bisector; this only occurs when the bisector is not one-dimensional.

The Voronoi region of a site is the locus of points closer to that site than to any other site: $V(s_i) = \bigcap_{j \neq i} h(s_i, s_j)$. We define the GVD$_\infty$ to be the union of all Voronoi region boundaries: GVD$_\infty(\mathcal{S}) = \bigcup_i \partial V(s_i)$. The GVD$_\infty$ in $\mathcal{R}^2$ is a one-dimensional construct that partitions the plane into the Voronoi regions and regions of equidistant points. It is composed entirely of straight line segments; each edge is on the boundary of a bisector. "Meet points" are junctions of two or more edges; these points are equidistant from the closest three or more sites. Figure 2 shows an example.

3.2 The SGVD$_\infty$

The *saturated generalized Voronoi diagram* (SGVD) (Acar, Choset, and Atkar, 2001) is a modification of the GVD that can be constructed by robots with limited-range sensors. In regions where the distance to the two closest sites is less than the "saturation distance" r_{sat}, the SGVD is identical to the GVD. At points on the GVD that are equidistant to two sites, and where the distance to the sites is equal to r_{sat}, the SGVD "diverges," maintaining the saturation distance to one site. We define the SGVD$_\infty$ in the same manner with respect to the GVD$_\infty$.

More formally, let $\mathcal{M}$ be the Minkowski sum of the sites with a "disk" with a radius equal to the saturation distance. (Under the L_∞ metric, this disk is an axis-aligned square.) Then we can define:

$$\text{SGVD}_\infty = (\text{GVD}_\infty \cap \mathcal{M}) \cup \partial\mathcal{M} \tag{3}$$

The SGVD$_\infty$ consists of a collection of line segments; each segment is either a *saturated segment* from the boundary of $\mathcal{M}$, or an *unsaturated segment*, a part of a bisector. In addition to the type of meet points found in the GVD$_\infty$, the SGVD$_\infty$ contains meet points where the GVD$_\infty$ meets the boundary of $\mathcal{M}$; these are junctions of two or more segments that are equidistant to two or more sites at r_{sat}.

3.3 Orientations of paths

In order for a robot to trace the SGVD$_\infty$, it must be able to compute the orientations of edges locally based on the closest points of equidistant obstacles. We now show that the orientations of SGVD$_\infty$ edges are restricted. This restriction is important to traversing the SGVD$_\infty$ with sparse sensing, since it guarantees a particular set of sensor orientations at all times.

First, we compute the directional derivative of the distance function. Let $f[\alpha]$ be the distance to an obstacle, parameterized by α, the angle to the closest point on the obstacle. In general, there will be multiple equidistant points on an obstacle under L_∞, in which case we choose the closest of them under L_2.

Lemma 3.1 *The directional derivative of $f[\alpha]$ as the robot moves forward in direction θ is:*

$$\frac{df}{ds} = \begin{cases} \cos(\theta-\pi) & \alpha \in (-\frac{\pi}{4}, \frac{\pi}{4}) \\ & \textit{or } (\alpha = -\frac{\pi}{4} \textit{ and } \sin(\theta-\alpha) \geq 0) \\ & \textit{or } (\alpha = \frac{\pi}{4} \textit{ and } \sin(\alpha-\theta) \geq 0) \\ \cos(\theta-\frac{3\pi}{2}) & \alpha \in (\frac{\pi}{4}, \frac{3\pi}{4}) \\ & \textit{or } (\alpha = \frac{\pi}{4} \textit{ and } \sin(\alpha-\theta) \geq 0) \\ & \textit{or } (\alpha = \frac{3\pi}{4} \textit{ and } \sin(\theta-\alpha) \geq 0) \\ \cos(\theta) & \alpha \in (\frac{3\pi}{4}, \pi] \bigcup [-\pi, -\frac{3\pi}{4}) \\ & \textit{or } (\alpha = -\frac{3\pi}{4} \textit{ and } \sin(\theta-\alpha) \geq 0) \\ & \textit{or } (\alpha = \frac{3\pi}{4} \textit{ and } \sin(\alpha-\theta) \geq 0) \\ \cos(\theta-\frac{\pi}{2}) & \alpha \in (-\frac{3\pi}{4}, -\frac{\pi}{4}) \\ & \textit{or } (\alpha = -\frac{\pi}{4} \textit{ and } \sin(\theta-\alpha) \geq 0) \\ & \textit{or } (\alpha = -\frac{3\pi}{4} \textit{ and } \sin(\alpha-\theta) \geq 0) \end{cases} \tag{4}$$

Proof: The directional derivative of $f[\alpha]$ can be written as:

$$\frac{df}{ds}[\alpha](\theta) \quad = \quad \frac{df}{dx}\cos\theta + \frac{df}{dy}\sin\theta \tag{5}$$

When the robot is not on a diagonal to an obstacle vertex, one of $\frac{df}{dx}$ or $\frac{df}{dy}$ must be 0 and the other ± 1 by the definition of the L_∞ norm (Equation 1). This implies that $\frac{df}{ds}$ can always be written in the form $\cos(\theta - n\frac{\pi}{2})$ for some integer n.

The derivative is not continuous across diagonals, so in these cases ($\alpha = \pm\frac{\pi}{4}, \pm\frac{3\pi}{4}$), the directional derivative depends upon the side of the diagonal in which θ lies. The resulting cases, while lengthy, are straightforward to compute, yielding Equation 4. □

With this, we can now show that edge orientations in the GVD$_\infty$ are restricted.

Lemma 3.2 *All edges in the GVD$_\infty$ for a rectilinear environment have orientation $n\frac{\pi}{4}$ for some integer n.*

Proof: Pick a point on an edge of the GVD$_\infty$. To determine the direction of this edge, we consider the directional derivative of the distance to each of the equidistant obstacles. Directions in which the directional derivatives are equal correspond to the orientation of the edge.

We take two obstacles and set the directional derivatives equal: $\frac{df_1}{ds} = \frac{df_2}{ds}$. Using Equation 4, we get:

$$\cos\left(\theta - n_1\frac{\pi}{2}\right) = \cos\left(\theta - n_2\frac{\pi}{2}\right) \tag{6}$$

for some integers n_1 and n_2. This can be simplified to:

$$a\cos\theta + b\sin\theta = 0 \tag{7}$$

where $a = \cos(n_1\frac{\pi}{2}) - \cos(n_2\frac{\pi}{2})$ and $b = \sin(n_1\frac{\pi}{2}) - \sin(n_2\frac{\pi}{2})$ for integral n_1, n_2. Note that $a, b \in \{-2, -1, 0, 1, 2\}$ and when one of a or b is ± 2, the other must be 0. So the solutions of this equation are $\theta \in \{0, \pm\frac{\pi}{2}, \pi\}$ when $a \in \{\pm 2, \pm 1\}$ and $b = 0$ or vice versa, and $\theta \in \{\pm\frac{\pi}{4}, \pm\frac{3\pi}{4}\}$ when $a, b = \pm 1$.

The case where $a = b = 0$ corresponds to either a point in the interior of a region of equidistant points (which cannot be on the GVD$_\infty$) or a point on the boundary of an equidistant region that is on a diagonal to an obstacle vertex. In the latter situation, value of n_1 and n_2 for Equation 7 will only be valid only over a 180 degree range of θ. Since the GVD$_\infty$ contains points only on the boundary of the equidistant region and not in the interior, we must move at the limits of this range, which must occur on the diagonal from the point, which must lie at $\theta \in \{\pm\frac{\pi}{4}, \pm\frac{3\pi}{4}\}$. □

We state the following Lemma without proof:

Lemma 3.3 *The boundary of the Minkowski sum of axis-aligned rectilinear polygons with an axis aligned square consists of edges with orientations of $\theta \in \{0, \pm\frac{\pi}{2}, \pi\}$.*

Since the SGVD$_\infty$ is the union of the Minkowski sum boundary and a subset of the GVD$_\infty$, we have the following:

Theorem 3.4 *The SGVD$_\infty$ contains edges only at $\theta = n\frac{\pi}{4}$ for integral n.*

Corollary 3.5 *All meet points occur on a diagonal equidistant from two of the closest walls.*

Proof: From Theorem 3.4, we know that all SGVD$_\infty$ edges are at orientations that are multiples of $\frac{\pi}{4}$. Therefore, a robot tracing the SGVD$_\infty$ must change its direction of travel discontinuously (by a multiple of $\frac{\pi}{4}$). $\frac{df}{ds}[\alpha]$ is discontinuous only when $\alpha \in \{\pm\frac{\pi}{4}, \pm\frac{3\pi}{4}\}$, i.e. on a diagonal between two walls. Diagonals are generally to the corner between two walls, but if the equidistant walls are perpendicular and do not intersect, the diagonal is to the point of intersection between the lines through the walls. □

4 Tracing the SGVD$_\infty$with an omnidirectional sensor

In this section, we describe behaviors with which a robot can trace the SGVD$_\infty$ using a short-range omnidirectional sensor. In Section 6, we extend these behaviors to the case in which a robot has only sparse sensing.

We assume a differential drive point robot with an omnidirectional limited-range sensor. The point robot assumption is made to simplify the behaviors; it is essentially equivalent to the assumption that there are no walls in the environment smaller than the radius of the robot, so by approaching as close as possible to walls the robot will detect all obstacles. We assume that the robot's range sensor maintains a set $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ of the angles to all closest equidistant obstacles that are within r_{sat}. If there is more than one point on a single obstacle equidistant under L_∞, then the closest point under L_2 is chosen.

4.1 Situating on the SGVD$_\infty$

When the robot is first turned on, it may not be at a location on the SGVD$_\infty$. Therefore, it must "situate" itself on the SGVD$_\infty$ before it can begin mapping. The SITUATE behavior performs the necessary actions. Though the robot may not be at a meet point when the SITUATE behavior finishes, the MEET-POINT behavior is used to determine directions of travel that are valid from the robot's initial location on the SGVD$_\infty$, since the required computations are the same.

Lemma 4.1 *The* SITUATE *behavior moves the robot from any starting location to a point on the SGVD$_\infty$.*

Proof: If the robot sees more than one equidistant obstacle, it may be inside a region of equidistant points, so lines 1–2 move the robot toward the closest point on an obstacle until that obstacle is closest. Otherwise, if there were no visible obstacles to begin with, the robot drives forward until it encounters one (lines 3–5). If it simultaneously encounters more than one equidistant obstacle, then the robot is on the boundary of a equidistant region. Since this point is also on the boundary of a Voronoi region, it must be on the SGVD$_\infty$.

Otherwise there is only one closest obstacle, and the robot must drive away from this obstacle until it first sees more than one obstacle or until it reaches the saturation distance. At this point, the robot will be either on the boundary of a Voronoi region or on a saturated edge, both of which are on the SGVD$_\infty$. □

Behavior: SITUATE
Behavior for situating the robot on the SGVD$_\infty$ from an arbitrary initial starting location.

1: **if** $|A| > 1$ **then** *// get out of regions of equidistant points*
2: Pick $\alpha_i \in A$ arbitrarily and move in that direction until $|A| = 1$
3: **else if** $|A| = 0$ **then** *// nothing in sight*
4: Move forward until $|A| > 0$
5: **end if**
6: **if** $|A| = 1$ **then** *// single obstacle closer than saturation distance*
7: Move in direction $\alpha_1 + \pi$ until one of the following occurs:
(A) Saturation distance is reached
(B) $|A| > 1$
8: **end if**
9: MEET-POINT *// determine direction to move*

Behavior: SATURATED(θ)
Traces a segment of the GVD$_\infty$ at the saturation distance from current closest obstacles

1: Let $n = |A|$
2: Continue moving in direction θ, maintaining saturation distance to obstacles at $\alpha_1 \ldots \alpha_n$
3: Until one of the following occurs:
(A) $|A| > n$
(B) Any $\alpha \in A$ becomes $\pm\frac{\pi}{4}$ or $\pm\frac{3\pi}{4}$ and $\theta \notin \{\alpha, \alpha + \pi\}$
4: MEET-POINT

4.2 Tracing SGVD$_\infty$ segments

The UNSATURATED and SATURATED behaviors are each responsible for following a single edge of the SGVD$_\infty$. They differ slightly in their termination conditions and also in the sensory feedback required to follow the desired segment.

The SATURATED behavior follows a line segment at the saturation distance from an obstacle. This is essentially a wall-following behavior under the L_∞ distance metric. It is possible for there to be multiple obstacles being tracked at the saturation distance, but they must all be on one side of the robot and collinear.

Lemma 4.2 *The* SATURATED *behavior follows an SGVD$_\infty$ edge at the saturation distance and terminates when it reaches a meet point.*

Behavior: UNSATURATED(θ, α_s)
Traces a segment of the GVD_∞ on the boundary of the Voronoi region for obstacle at $\alpha_s \in A$.

1: Let $n = |A|$
2: Continue moving in direction θ, maintaining equidistance from obstacle at α_s to other equidistant obstacles in A
3: Until one of the following occurs:
   (A) $|A| > n$
   (B) Any $\alpha \in A$ becomes $\pm\frac{\pi}{4}$ or $\pm\frac{3\pi}{4}$ and $\theta \notin \{\alpha, \alpha + \pi\}$
   (C) Saturation distance is reached
4: MEET-POINT

Proof: All points on a saturated edge must be at the saturation distance from all the closest equidistant obstacles, so the direction θ satisfies $\frac{df}{ds}[\alpha](\theta) = 0$ for all $\alpha \in A$. This condition is met when called from the MEET-POINT behavior.

The behavior should terminate when it reaches a location where the robot must change direction to maintain $\frac{df}{ds} = 0$, or when new obstacles are encountered and a choice of which obstacles to track must be made. Thus, termination conditions are:

(A) When a new obstacle is detected at the equidistant range, the robot may choose to trace the boundary of the Voronoi region of that obstacle.
(B) When the robot crosses a diagonal ($\alpha \in \{\pm\frac{\pi}{4}, \pm\frac{3\pi}{4}\}$) and is not moving along the diagonal (toward or away from the corner), $\frac{df}{ds}$ is discontinuous. Thus, the robot must change direction instantaneously to maintain the saturation distance offset (see Corollary 3.5), so this is a meet point.

□

The UNSATURATED behavior follows a line segment on the boundary of the Voronoi region for the obstacle at α_s. This segment will be equidistant to the obstacle at α_s and to the other obstacles in A. When $|A| = 2$, the edge is on the bisector between two Voronoi regions. In this case, this behavior is essentially a hall-following behavior — if the robot should veer to one side of this segment, the distance to one obstacle will increase whereas the distance to the other obstacle will decrease. However, the distance of equidistance may increase or decrease (e.g., when following a diagonal separating two perpendicular walls).

When $|A| > 2$, this edge separates the Voronoi region for α_s from a region of equidistant points. If the robot should veer off the segment into the Voronoi region, the distance to that obstacle will decrease and distance to the other obstacles will increase. However, if the robot veers into the equidistant region, the distance to all obstacles represented in A remains equal. Thus, this case is more similar to wall-following than to hall-following in that the robot must maintain the correct offset from a single obstacle at α_s. However, since multiple obstacles are equidistant and the equidistant range is less than r_{sat}, the UNSATURATED behavior is used. The offset from the obstacle may increase or decrease since at less than r_{sat}, the boundaries of equidistant regions are on diagonals.

Behavior: MEET-POINT
Behavior for determining valid outgoing paths at a meet point.

1: Compute W, the set of all pairs (θ, i) where direction θ lies on the boundary of the Voronoi region of the obstacle at α_i. If at the saturation distance, exclude any θ where $\frac{df}{ds}[\alpha_i](\theta) > 0$.
2: **if** at the saturation distance **then**
3: Compute Y, the set of all pairs (θ, i) where $\frac{df}{ds}[\alpha_i](\theta) = 0$ and $\forall_{\alpha_j \in A}\ \frac{df}{ds}[\alpha_j](\theta) \geq 0$
4: Compute C, the set of unique θ in pairs from W and Y
5: Choose an element c from C
6: Start moving in the direction c
7: **if** $(c, *) \in W$ **then**
8: Let s be the index i from any pair $(c, i) \in W$
9: UNSATURATED(c, α_s)
10: **else**
11: SATURATED(c)
12: **end if**

Lemma 4.3 *The* UNSATURATED *behavior follows an SGVD$_\infty$ segment on the boundary of at least one Voronoi region and terminates when it reaches a meet point.*

Proof: Since this edge is on the boundary of a Voronoi region, we know that θ is a direction that maintains equidistance to the closest obstacles, so $\frac{df}{ds}[\alpha_i](\theta) = \frac{df}{ds}[\alpha_j](\theta)$ for any $\alpha_i, \alpha_j \in A$. The termination conditions are the same as those of the SATURATED behavior, except that the robot must also terminate when it reaches the saturation distance. This is a meet point where the GVD_∞ reaches the saturation distance. □

4.3 Meet points

In general, when either of the above SGVD_∞ tracing behaviors terminate, a meet point has been reached. The MEET-POINT behavior determines which directions will take the robot to a new segment of the SGVD_∞. There are two phases to this computation.

First, all directions corresponding to segments for the UNSATURATED behavior are computed. This is done by determining, for the current robot location, which directions correspond to an edge on the boundary of the Voronoi region for each obstacle. These can be found by considering the bisectors between one obstacle and the other obstacles. Figure 3 shows the relevant cases. Each bisector eliminates a range of angles. After overlapping the ranges eliminated by all the bisectors, the angles at the extremes of the remaining range are the directions to follow the boundary of that Voronoi region. In certain cases, the bisectors for an obstacle may eliminate all directions, which indicates that this meet point is not on the boundary of the Voronoi region of that obstacle. Note that if the meet point is at the saturation distance, any direction for which $\frac{df}{ds}(\theta) > 0$ must be discarded since the robot cannot increase the distance of equidistance beyond r_{sat}. Figure 4 depicts an example of the process of determining outgoing directions from meet points

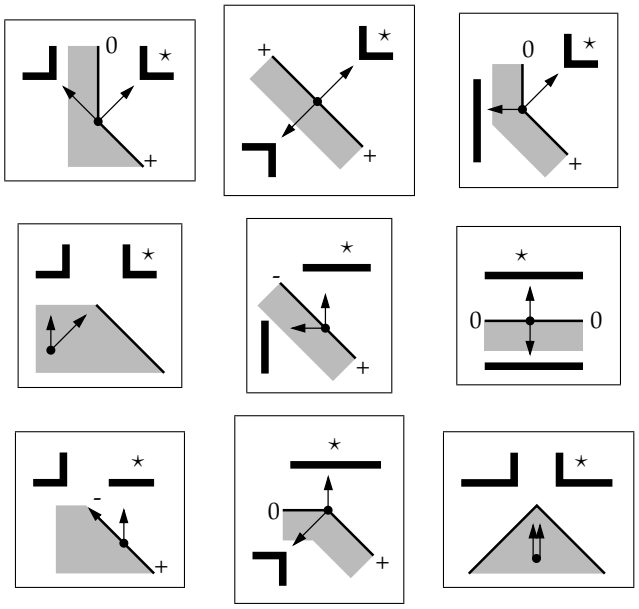


Figure 3: Types of local bisectors between two obstacles. The Voronoi region of the obstacle labeled "⋆" is being computed by eliminating ranges of angles (highlighted side of the bisector). See Figure 4 for a complete example. Moving along the bisector, the range of equidistance may change (+ indicates an increase, - a decrease, 0 no change). Note that in some cases the complete bisector is not known from only local information (e.g., when the robot is inside or on the boundary of a region of equidistant points).

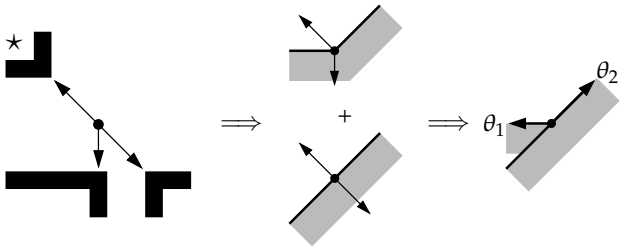


Figure 4: Example of determining outgoing directions from a meet point by examining local bisectors between obstacles detected at the meet point.

using local bisectors.

If the meet point is at r_{sat}, then we also compute the directions that maintain the saturation distance to a subset of the current equidistant obstacles. This is done by solving $\frac{df}{ds}(\theta) = 0$ for values of θ. The allowable values of θ must be directions in which the change in distance to other obstacles will be nonnegative since otherwise those obstacles would become closer than the obstacle being traced at the saturation distance, and the robot would no longer be on the SGVD$_\infty$.

One of the allowable directions is chosen, and the robot moves in that direction and executes the correct behavior. In some cases, there may be only one path (aside from the incoming one) that can be taken from a meet point. In these circumstances, no choice is necessary.

Lemma 4.4 *The* MEET-POINT *behavior generates the directions of all edges leaving a point on the SGVD$_\infty$ and calls the appropriate behavior to trace the chosen edge to the next meet point.*

Proof: Line 1 of the MEET-POINT behavior calculates the directions of edges on the boundaries of Voronoi regions. The only other directions that can be on an edge of the SGVD$_\infty$ are edges at the saturation distance. Lines 2–3 of the algorithm, executed only when the meet point is at the saturation distance, calculate these directions. See the preceding discussion for the details of these calculations.

In lines 4–6, a direction c is chosen, and the robot starts moving in that direction to put the robot on the edge before the appropriate edge-following behavior is called in lines 7–11.

If c is associated with one or more pairs in W, then it lies on the Voronoi region of an obstacle, so the UNSATURATED behavior is called on line 9. Otherwise, c is not on the boundary of any Voronoi region, so it must be a saturated edge (from a pair in Y), and the SATURATED behavior is called on line 11. Note that edges at distance r_{sat} from two non-collinear walls are unsaturated edges under this approach. □

4.4 Completeness

The SITUATE behavior moves the robot to a point on the SGVD$_\infty$ and calls the MEET-POINT behavior to move it to a meet point. Subsequently, the MEET-POINT behavior calls either the SATURATED or UNSATURATED behaviors to move the robot to another meet point. With a suitable strategy for exploration, the robot will traverse all edges of the SGVD$_\infty$.

We note also that there are SGVD$_\infty$ edges that extend into each convex and nonconvex corner. It is not necessary to traverse these edges to explore them (unless there is an opening at an interior corner) because all the relevant area will be sensed from the other segments on the SGVD$_\infty$.

Theorem 4.5 *The* SITUATE, SATURATED, UNSATURATED, *and* MEET-POINT *behaviors are sufficient to guide a robot to trace a connected component of the SGVD$_\infty$ from any initial configuration.*

Proof: The proof follows immediately from the previous Lemmas and discussion. □

5 A virtual sensor for sparse sensing

We now consider the case in which the robot has only sparse short-range sensing, rather than omnidirectional short-range sensing. Specifically, we assume that the robot has eight one-dimensional sensors (e.g. infrared sensors), with one pointing in the forward direction and the rest spaced at 45° intervals around the robot.

In order to trace the SGVD$_\infty$, the robot needs to know about the closest equidistant obstacles. However, with sparse sensing, the robot may not always be able to measure the distance to the closest obstacles. Even worse, the robot may not detect an obstacle until it has gotten too close, i.e., after the robot has strayed off the SGVD$_\infty$.

We address these problems by introducing a "virtual sensor" which keeps track of the distance to obstacles that can no longer be sensed directly by the robot. Still, this does not solve the problem of discovering an obstacle too late, so we present behaviors in Section 6 that handle situations where the robot has strayed off the SGVD$_\infty$ and go back to correct it.

5.1 Basic operation

The main function of the virtual sensor is to keep track of a set of equidistant obstacles and report their angles relative to the robot, in similar fashion to the omnidirectional sensor. The robot will maintain equidistance to these obstacles in order to trace the SGVD$_\infty$. These obstacles are represented by a set T that contains both "unseen obstacles" — representations of obstacles that cannot be directly observed but that have been seen in the past — and sensors that can directly observe an obstacle. Normally, the obstacles represented by T are the closest (known) equidistant obstacles. Let e be the distance to these obstacles. The set T contains all sensors for which the range $d(s) = e$ and all unseen obstacles u for which the distance to that obstacle $d(u) = e$.

When first initialized, T contains only sensors. As the robot moves, there are six kinds of changes related to the obstacles represented by T.

1. The distance of equidistance e may change; this does not have any effect on T.
2. One or more obstacles in T may increase in distance beyond e so they are no longer equidistant; any unseen obstacle or corresponding sensors should be removed from T.
3. Obstacles seen by a sensor that were at a distance greater than e may now be at a distance e; the sensors that can see such obstacles should be added to T.
4. An equidistant obstacle observed by a sensor may "disappear" so that it is not seen by any sensor. An unseen obstacle should be created and added to T, and that sensor should be removed from T.
5. An unseen equidistant obstacle may "appear," i.e., be detected by a sensor. This unseen obstacle should be removed from T, and the appropriate sensor should be added to T.
6. One or more previously undetected obstacles may appear that are closer than distance e; we call the appearance of such obstacles a "K-Appearance" and set a flag to true. This flag remains true as long as there are any obstacles closer than e. The set T is *not* changed in this case.

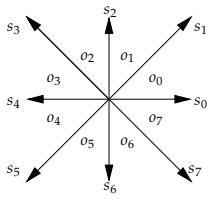


Figure 5: Labels for sensors and octants which are fixed in the world frame.

Variable	Explanation
θ	the robot's current heading
S	the set of sensors; $S \equiv \{s_0, s_1, \ldots, s_7\}$
U	a set of unseen obstacles (o, d) where o is the octant in which it lies and d is the distance to the obstacle
T	the set of "objects" $t \in U \cup S$ (sensors and unseen obstacles) being tracked to follow the SGVD$_\infty$ at any given time
A	a set of pairs (α, t), $t \in T$, of angles to objects being tracked and the corresponding objects
ang$[s]$	absolute angle to the wall observed by sensor s; always a multiple of $\frac{\pi}{2}$ except when a diagonal sensor sees a corner
ang$[u]$	absolute angle to an unseen obstacle u that is being tracked; for $u = (o_i, d)$, ang$[u] \equiv \left\lfloor \frac{i+1}{2} \right\rfloor \frac{\pi}{2}$

Table 1: Variables used by the virtual sensor.

This paradigm maintains a consistent set of obstacles for the robot to track but also indicates when the robot has strayed off the SGVD$_\infty$. After a K-Appearance occurs, the robot's SGVD$_\infty$ must be corrected because a meet point was missed. The robot will turn around and re-trace its path on the (now incorrect) SGVD$_\infty$. The obstacle that caused the K-Appearance will become an unseen obstacle and will eventually become equidistant to the other obstacles represented by T. This causes the flag to be turned off, indicating the robot is at the missed meet point. The handling of K-Appearances is discussed in detail in Section 6.

The virtual sensor is given in the VIRTUAL-SENSOR algorithm, but the main work is done by the VS-TRACK algorithm, described in Section 5.2, which maintains the set T, and the VS-UPDATE algorithm, described in Section 5.3, which computes A, the output of the virtual sensor.

5.1.1 Notation

We will label the robot's eight range sensors s_0 through s_7, as as illustrated in Figure 5. These labels are for absolute sensor orientations in the world frame and do not shift when the robot changes direction. Instead, the physical sensors on the robot have a different labels depending on the direction of the robot

Function	Explanation
$d(s)$	L_∞ distance reported by sensor s; if s detects nothing within the saturation distance r_{sat}, $d(s) = \infty$
$d(u)$	L_∞ distance to an unseen obstacle u
$d^-(s)$	returns the range on sensor s just before a disappearance
ORIENT(s)	absolute orientation of sensor s
EC-DIAG(s)	`#t` if sensor s is on a diagonal to an exterior (convex) corner of an obstacle within r_{sat}
IC-DIAG(s)	`#t` if sensor s is on a diagonal to an interior (concave) corner of an obstacle within r_{sat}
CALC-B/F(s_i, θ)	computes the sensors and octants behind and in front of sensor s_i when the robot moves in direction θ. Returns s_h, s_j, o_h, o_j where s_h is the sensor behind sensor s_i; s_j, the sensor in front; o_h, the octant behind; and o_j, the octant in front.
SET-SAME(s_i, s_j, b)	asserts that sensors s_i and s_j are not seeing the same wall if the boolean b is `#f`.
IS-SAME(s_i, s_j)	returns `#f` if it has been asserted that sensors s_i and s_j are seeing different walls or if $d(s_i) \neq d(s_j)$; accounts for transitivity and commutativity of sensors.
APPEAR(s)	returns `#t` if there has been a discontinuous decrease in range on s
DISAPPEAR(s)	returns `#t` if there has been a discontinuous increase in range on s

Table 2: Utility functions for implementing the virtual sensor.

motion; for example, when the robot moves in direction $\theta = i\frac{\pi}{4}$, its front sensor is s_i. We will sometimes refer to sensors relative to the direction of the robot's motion: front diagonal sensors, side sensors, and rear diagonal sensors. For example, if the robot's direction is $\frac{5\pi}{4}$, s_5 is the front sensor, s_6 and s_4 are the front diagonal sensors, s_7 and s_3 are the side sensors, and s_0 and s_2 are the rear diagonal sensors. We may also refer to the two adjacent sensors (to a given sensor) as the "forward sensor" and "backward sensor," again relative to the direction of the robot's motion. When we refer to "diagonal sensors," this means the sensors at orientations of $\pm\frac{\pi}{4}$ and $\pm\frac{3\pi}{4}$, namely s_1, s_3, s_5, and s_7.

Figure 5 also shows labels for the eight octants, o_0 through o_7, which are also fixed in world frame. Octant o_i includes points at directions from $i\frac{\pi}{4}$ to $(i+1)\frac{\pi}{4}$. Keeping the sensor and octant labels fixed in the world frame simplifies the bookkeeping for the virtual sensor.

In our algorithms, we use the symbols `#t` for true, `#f` for false, $\wedge$ for the boolean AND operation, $\vee$ for OR, and $\neg$ for NOT. We also assume that orientations and angles are compared modulo 2π.

Algorithm: VIRTUAL-SENSOR
Maintain a set of equidistant obstacles for the robot to track.

```
VS-INIT
loop
    VS-TRACK
    VS-UPDATE
end loop
```

Algorithm: VS-INIT
Initialize virtual sensor state

```
Perturb robot θ to determine ang[s] for diagonal sensors
for all s_i ∈ S do
    s_h, s_j, o_h, o_j ← CALC-B/F(s_i, θ)
    SET-SAME(s_i, s_j, d(s_i) == d(s_j) ∧ ang[s_i] == ang[s_j])
end for
Let T = {s ∈ S | d(s) = min_{σ∈S} d(σ)}
```

5.1.2 Variables and utility functions

The variables used by the virtual sensor are listed in Table 1. The "ang" variable is a slightly special variable, used to keep track of the angles to unseen obstacles and to walls observed by the sensors. This information is needed to properly calculate the virtual sensor output. For unseen obstacles and nondiagonal sensors, this angle can be calculated, but for the diagonal sensors, the algorithms must actively determine these angles.

A number of utility functions, listed in Table 2, are used to simplify the presentation of the algorithms. A few functions merit some additional explanation. The CALC-B/F function, used to compute the forward and backward sensors and octants, can be applied to the front sensor, returning two backward (and no forward) sensors and octants. The back sensor is handled similarly.

Knowing whether two adjacent sensors are detecting the same obstacle is necessary to determine when unseen obstacles should be created and to calculate the virtual sensor output. When we know that two sensors are seeing different walls, a false (`#f`) assertion with the SET-SAME function is made. A true (`#t`) assertion, however, means that the sensors *may* be observing the same obstacle. The IS-SAME function checks for false assertions but also compares the distance on the sensors when it is called. For convenience, IS-SAME handles commutativity of the arguments and transitivity of "sameness."

5.1.3 Implementation note

In practical circumstances, the EC-DIAG and IC-DIAG functions would depend on the robot having a sensor range of slightly more than r_{sat} and would look for a local maximum or minimum in the range by observing a time series of readings. This would require the robot to travel a small distance past the diagonal to

Algorithm: VS-TRACK
Track unseen obstacles.

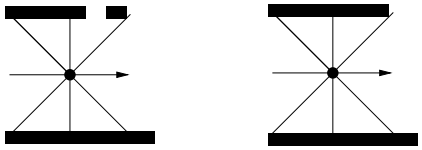
1: Let Δd be the distance moved in direction θ
2: **for all** $u = (o, d) \in U$ **do** *// track unseen obstacles*
3: Let $\alpha = \text{ang}[u]$
4: $d \leftarrow d + \Delta d \frac{df}{ds}[\alpha](\theta)$
5: **end for**
6: **for all** $s_i \in S \mid \text{APPEAR}(s_i) \vee \text{DISAPPEAR}(s_i)$, in order from front to back **do**
7: $s_h, s_j, o_h, o_j \leftarrow \text{CALC-B/F}(s_i, \theta)$
8: **if** $\text{DISAPPEAR}(s_i)$ **then**
9: **if** $\text{IS-SAME}(s_i, s_h)$ == #f $\vee$ $\text{ang}[s_i] \neq \text{ang}[s_h]$ **then**
10: Add $(o_h, d^-(s_i))$ to U
11: **if** $s_i \in T$ **then**
12: Add $(o_h, d^-(s_i))$ to T
13: $\text{SET-SAME}(s_i, s_h, d(s_i) == d^-(s_h))$
14: **end if**
15: **if** $\text{APPEAR}(s_i)$ **then**
16: $\text{SET-SAME}(s_i, s_h, \texttt{\#f})$ *// for both s_h if front sensor*
17: Remove s_i from T
18: **end if**
19: **if** $d(s_i) <= r_{\text{sat}}$ **then**
20: **if** $\text{ORIENT}(s_i) \in \{\pm\frac{\pi}{4}, \pm\frac{3\pi}{4}\}$ **then**
21: **if** $\text{ORIENT}(s_i) == \theta \vee \text{DISAPPEAR}(s_i)$ **then**
22: $\text{ang}[s_i] =$ "?"
23: **else**
24: $\text{ang}[s_i] = \text{ORIENT}(s_j)$
25: **if** $\exists (o, d) \in U \mid o == o_j \wedge d == d(s_i)$ **then**
26: Remove (o, d) from U
27: $\text{SET-SAME}(s_i, s_j, \texttt{\#f})$
28: **else**
29: $\text{SET-SAME}(s_i, s_j, d(s_i) == d(s_j))$
30: **end if**
31: **end for**
32: Let $e = \min_{t \in T} d(t)$
33: $U \leftarrow U - \{u = (o, d) \in U \mid d > e\}$
34: $T \leftarrow \{t \in S \cup U \mid d(t) = e\}$

detect a corner; the robot could then move backwards to the potential meet point.

5.2 Tracking unseen obstacles

The primary function of the VS-TRACK algorithm is to maintain the set T which represents the closest equidistant obstacles. In order to provide information about obstacles that cannot be sensed directly, "unseen obstacles" are created and maintained. An unseen obstacle is a pair (o, d) where o is the octant the obstacle lies in, and d is the distance to the obstacle.

As the robot moves, the distance to unseen obstacles must be updated; this is handled by the first part of the VS-TRACK algorithm (lines 1–5). However, the bulk of the VS-TRACK algorithm (lines 6–31) is devoted to handling the appearance and disappearance of obstacles on the sensors. This may involve creating or removing unseen obstacles as well as updating the virtual sensor state. Finally, the last part of this algorithm (lines 32–34) updates the set T.



(a) (b)

Figure 6: Two examples where an obstacle disappears from sensor s_1 with the same sensor readings. In example (a), an unseen obstacle should be created, but not in example (b).

5.2.1 Disappearances

A "disappearance" is a discontinuous increase in the range reported by a sensor, either to a finite value or to ∞. The main decision that must be made when a disappearance occurs is whether to create an unseen obstacle or not.

Figure 6 shows two disappearances with the same readings on the other sensors; an unseen obstacle should be created in one but not in the other. The key to deciding when to create an unseen obstacle is whether the back sensor is detecting the same wall.

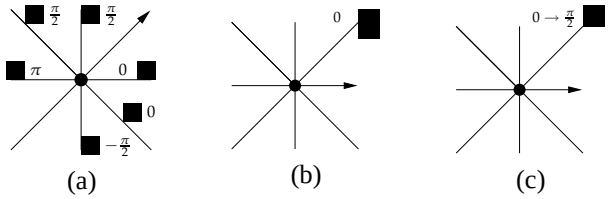
In the general case, the robot may not have seen enough of the environment to determine whether two sensors observe the same wall or not. Because we only keep limited state information about the environment, we assume that two sensors on the same side of the robot with the same distance readings *are* observing the same wall — unless there is some evidence to the contrary (expressed as a false SET-SAME assertion). If we assume two walls are the same and they are not, the robot will discover this in the course of its exploration and correct its map appropriately.

Disappearances are handled on lines 8–14 of VS-TRACK. If it is known that the backward sensor does not see the same wall, an unseen obstacle is created. In addition, this unseen obstacle must be added to T if that sensor was being tracked at the equidistant range.

After a disappearance, the sensor may still detect a wall; we call this a "finite-range" disappearance. This situation is like an appearance because a new obstacle is now seen on that sensor. We need to decide whether this sensor and its back sensor are seeing the same wall. If the distances are different, we know that they see different walls; this is asserted in line 13. Whether or not the forward sensor sees the same wall is evaluated by the last part of this section (lines 19–30), which is executed for both disappearances and appearances.

5.2.2 Appearances

An "appearance" is a discontinuous decrease in the range reported by a sensor, either from ∞ or from a finite value. Appearances are often caused by a corner of an obstacle crossing the ray of a sensor. This means that a wall has started crossing



(a) (b) (c)

Figure 7: Appearance examples, with the angle from the robot to the newly detected wall displayed: (a) appearances at less than r_{sat} where the angle to the wall for diagonal sensors will be the orientation of the forward sensor; (b) a point on a wall appears at r_{sat} on a front diagonal sensor; (c) a corner of an obstacle appears at r_{sat} on a front diagonal sensor; the angle will be set normally (to 0 in this case) and will be changed to $\frac{\pi}{2}$ by the S-MEET-POINT behavior if the robot keeps moving to the right.

from one octant to the next as the robot moves forward. However, an appearance can also be caused by any point of a wall (along the sensor ray) that has decreased in range to r_{sat}. Figure 7 illustrates several different situations.

A single appearance definitively indicates that the backward sensor is not seeing the same wall. In addition, if the sensor was being tracked, it should be removed from T since the wall that appeared is closer (this is a K-Appearance). These two changes, handled in lines 15–18, are unique to appearances (and not to the "appearance" caused by a finite-range disappearance).

The remainder of the appearance handling applies to both appearances and to finite-range disappearances. The first part (lines 20–24) sets the angle to the wall for this sensor; this only needs to be done for the diagonal sensors. Figure 7 indicates this angle for the examples shown; it can easily be seen in these cases that the angle should be set to the orientation of the forward sensor. The angle for a sensor can only change if the robot crosses the diagonal from an interior or exterior corner. This is taken care of by the FIX-ANG algorithm called from line 1 of the VS-UPDATE algorithm. We should note that if the appearance is caused by an exterior corner at r_{sat}(Figure 7c), the angle is not correct but will be fixed immediately during the execution of the VS-UPDATE algorithm.

There are two exceptions to setting the angle to the orientation of the forward sensor: when the appearance happens on the front sensor and when the appearance is due to a disappearance. In these cases, we cannot instantaneously determine the angle, so it is set to an "unknown" value. In the meet point behavior when the actual angle is required, the robot must actively measure it.

The second part of the appearance handling (lines 25–29) makes an assertion about whether this sensor and the forward sensor are seeing the same wall. If there is an unseen obstacle in the forward octant at the same distance, then we assume the unseen obstacle has appeared on this sensor. The unseen obstacle is removed, and we know that the forward sensor cannot be seeing the same wall — an unseen obstacle can only be created when an wall disappears from a sensor and that sensor was the only sensor that detected that wall. Otherwise, we assume that the distances from the two sensors indicate whether or not they see the same wall.

Note that a wall can disappear from one sensor and appear on an adjacent sensor simultaneously. For this reason, the VS-TRACK algorithm processes sensors from front to back (line 6) so that the disappearance is handled first (creating an unseen obstacle), before the appearance (removing the unseen obstacle).

5.2.3 Updating T

The set T is updated in lines 32–34. This update is fairly straightforward, but the key to the correctness of the VS-TRACK algorithm lies in knowing that an appearance or disappearance of an obstacle represented by T will always be detected.

When one obstacle occludes another, a disappearance or appearance may be missed by a sensor. Furthermore, different sensors may miss different events. This could potentially cause "phantom" unseen obstacles — an obstacle that disappears from one sensor but never appears on the sensor behind it and is therefore never removed. Since T contains obstacles at the same distance, none may occlude any other. The only exception is a K-Appearance: when an obstacle appears that is closer than the obstacles represented in T. This obstacle could then occlude obstacles represented by T. We address K-Appearances fully in Section 5.3.1 but offer the following related lemmas in the meantime.

Lemma 5.1 *As long as there is no known obstacle closer than the obstacles represented by T, any unseen obstacle in T will either be removed or will appear on a sensor.*

Proof: An unseen obstacle must initially have a distance of r_{sat} or less. As the robot moves, the distance must either increase, decrease, or remain the same. If the distance decreases the robot will eventually see the unseen obstacle, as the sensors will sweep all points within r_{sat}. If the distance increases, the unseen obstacle will eventually be at a distance of greater than r_{sat} and will therefore be removed. This leaves only the case where the unseen obstacle distance remains the same. From examining Equation 4, we note that this can only occur when the robot is traveling in a direction $n\frac{\pi}{2}$ and the obstacle is in one of the four "side" octants. (For example, if $\theta = 0$, the obstacle must be in o_1, o_2, o_5, or o_6.) However, all points in these octants will be swept by the side or rear diagonal sensors, so the obstacle will appear on a sensor. Note that since there is no known closer obstacle, the appearance of the unseen obstacle cannot be occluded. □

Lemma 5.2 *The set T will represent the closest known equidistant obstacles until a K-Appearance occurs.*

Proof: When the virtual sensor is initialized, T contains only the sensors with the minimum distance reading. As the robot moves about its environment, there are six kinds of changes that can occur, as listed in Section 5.1.

Algorithm: VS-UPDATE
Update the output of the virtual sensor.

1: FIX-ANG(θ)
2: Let $A = \{\}$
3: **for all** $s_i \in S$ **do**
4: **if** EC-DIAG(s_i) $\wedge$ $d(s_i)$ == e **then**
5: ang$[s_i]$ = ORIENT(s_i)
6: **if** IC-DIAG(s_i) $\wedge$ $d(s_i)$ == e **then**
7: $s_h, s_j, o_h, o_j \leftarrow$ CALC-B/F(s_i, θ)
8: **if** IS-SAME(s_i, s_h) == #f **then**
9: Add (ang$[s_h], s_i$) to A
10: **if** IS-SAME(s_i, s_j) == #f **then**
11: Add (ang$[s_j], s_i$) to A
12: **end for**
13: Let $M = \{\}$
14: **for all** $s_i \in S \mid \neg$IC-DIAG(s_i) **do**
15: **if** ($d(s_i)$ == e $\wedge$ $\neg\exists m \in M \mid$ IS-SAME(s_i, m)) **then**
16: Add s_i to M
17: **end for**
18: $A \leftarrow A \cup \{(\text{ang}[t], t) \mid t \in T \cap U \vee t \in M\}$
19: K-Appear = $(\exists r \in S \cup U \mid d(r) < e)$
20: **if** $\exists s_i \mid \{$APPEAR(s_i) $\wedge$ ORIENT(s_i) $\in \{\theta + \frac{\pi}{2}, \theta - \frac{\pi}{2}\}$ $\wedge \neg(\exists (a, t) \in A \mid a$ == ORIENT(s_i))$\}$ **then**
21: S-Appear $\leftarrow$ #f

Algorithm: FIX-ANG(γ)
Fix angles of sensors when they pass over corners.

1: **for all** $s_i \in S$ **do** *// update sensor state for this direction*
2: $s_h, s_j, o_h, o_j \leftarrow$ CALC-B/F(s_i, γ)
3: **if** EC-DIAG(s_i) **then**
4: ang$[s_i]$ == ORIENT(s_h)
5: **else if** IC-DIAG(s_i) **then**
6: ang$[s_i]$ == ORIENT(s_j)
7: **end for**

Case 1 is when the distance of the closest equidistant obstacles represented by T changes; this distance, e, is computed in line 32. Cases 2 and 3 remove obstacles at a distance greater than e and add obstacles now at e; this is handled on line 34. Case 4 addresses obstacles that have disappeared from a sensor; this was discussed in Section 5.2.1. Case 5 covers obstacles that appear on a sensor; this was discussed in Section 5.2.2. Finally, case 6 handles K-Appearances. We note that an obstacle that causes a K-Appearance will be closer than any obstacle represented in T but the sensor that detects it will not be added to T.

These six cases ensure that T is properly updated based on observed appearances and disappearances as well as calculated updates to unseen obstacles. However, one possibility remains: that an obstacle disappears, causing an unseen obstacle to be created, and there is never an observed appearance to remove it. By the previous lemma, it is not possible for this to happen as long as no known obstacle exists closer which could occlude the appearance, and this is what a K-Appearance signals. □

5.3 Generating virtual sensor output

The VS-TRACK algorithm maintains information about the set of nearest equidistant obstacles. In order to emulate the output of an omnidirectional sensor, the virtual sensor must use the tracking information to produce a set A containing angles to those obstacles. The VS-UPDATE algorithm performs the necessary computations.

The first part of the VS-UPDATE algorithm (lines 2–12) handles sensors that lie on a diagonal to an interior or exterior corner. For exterior corners, the closest point is exactly at the corner; for interior corners, there are two closest points, on the walls adjacent to the corner. The next part of the algorithm (lines 13–17) ensures that no two sensors that see the same obstacle contribute to A, since each equidistant obstacle should only be reported once to the meet point and other behaviors. Finally, the output state of the virtual sensor can be constructed. Angle pairs for each equidistant obstacle are added to A. The K-Appear and S-Appear flags are also set if necessary.

5.3.1 K-Appearances and S-Appearances

An obstacle discovered closer than the obstacles represented by T causes a K-Appearance, indicating the robot has strayed off the SGVD_∞. In order for the robot to return to the SGVD_∞, this obstacle must be "ignored" by the virtual sensor for the time being so that the robot can retrace its steps. The set T represents the obstacles that the virtual sensor previously determined to be the closest equidistant obstacles, so this set must be maintained for the robot to find its way back to the SGVD_∞. The K-Appear flag is set whenever a K-Appearance occurs.

There is one type of K-Appearance that must be handled slightly differently. This is when the robot is following the boundary of an obstacle at r_{sat} and a new obstacle appears at r_{sat} on a side sensor (opposite the obstacle the robot has been following). Without any special handling, this would be treated as a meet point. However, in this situation, it means that the robot missed a meet point where it should have switched to an unsaturated behavior. The VS-UPDATE algorithm sets the flag S-Appear to indicate when this situation might have arisen. Line 20 checks whether there has been an appearance on a side sensor and that there is no other obstacle on that side of the robot. The S-SATURATED behavior takes care of the rest.

5.4 Virtual sensor correctness

We have described the background and operation of the virtual sensor, so we now set about establishing its correctness.

Lemma 5.3 *The virtual sensor produces a set A equivalent to the set A produced by the omnidirectional sensor for the obstacles represented by T.*

Proof: By Lemma 5.2, T represents all the closest equidistant obstacles. The set A should contain only one element for each obstacle. Through the SET-SAME assertions, the robot records when sensors are known to be observing distinct

walls, and this is used to only add one element to A per obstacle wall.

An obstacle is represented in the set A by a pair containing the angle from the robot to each obstacle and either an unseen obstacle or a sensor that can see the corresponding wall. The unseen obstacle or sensor is associated so that the behaviors can know which sensor to use or how to access the distance to the unseen obstacle in the virtual sensor state.

The angles produced by the virtual sensor cannot, in general, be identical to those from the omnidirectional sensor. However, the directional derivative of Equation 4 returns the same result if the angles to obstacles (except for angles on the diagonals, i.e., $\pm\frac{\pi}{4}$ and $\pm\frac{3\pi}{4}$) are rounded to the nearest $n\frac{\pi}{2}$. The angles reported for sensors and unseen obstacles are always values of $n\frac{\pi}{2}$ except for diagonal sensors that see exterior corners. Therefore, the set A produced by the virtual sensor is equivalent to that from the omnidirectional sensor, with respect to the known closest equidistant obstacles. □

Theorem 5.4 *The virtual sensor correctly emulates the omnidirectional sensor with respect to known obstacles until a K-Appearance occurs. If the virtual sensor is used to immediately return to the missed meet point, the sensor will thereafter correctly emulate the omnidirectional sensor again.*

Proof: The previous lemmas establish the emulation of the omnidirectional sensor until a K-Appearance is detected. Once one occurs, there is an obstacle closer than any obstacle represented in T, and this obstacle could occlude appearance or disappearances that enable the virtual sensor to maintain T. However, if the robot turns around, this obstacle will become an unseen obstacle and thus unable to occlude any obstacle represented by T. By returning to the missed meet point, this unseen obstacle will be added as an element of T, so the virtual sensor will emulate the omnidirectional sensor with this newly-discovered obstacle. □

Behavior: S-SITUATE
Behavior for situating the robot on the SGVD$_\infty$ from an arbitrary initial starting location and orientation.

1: **if** no sensor sees an obstacle **then**
2: move forward until at least one sensor sees an obstacle
3: Turn the robot until the front sensor sees a local minimum
4: Move forward until the obstacle is reached (zero distance)
5: **if** robot is on a wall **then**
6: follow the wall until a corner is reached
7: Turn to align the robot with the diagonal from the corner
8: Start moving forward
9: Start running VIRTUAL-SENSOR
10: S-UNSATURATED(`#t`)

Behavior: S-SATURATED(k)
Traces a segment of the SGVD$_\infty$ at the saturation distance from current closest obstacles.

1: Let $n = |A|$
2: **repeat**
3: Move in direction θ, maintaining saturation distance to all objects $t \in T | (*, t) \in A$
4: **until** one of the following conditions is true:
(A) $\exists s_i \in S$ such that $(\text{ORIENT}(s_i) \notin \{\theta, \theta + \pi\}) \wedge (\text{EC-DIAG}(s_i) = \texttt{\#t} \vee \text{IC-DIAG}(s_i) == \texttt{\#t})$
(B) `K-Appear` $== k \wedge$ `S-Appear` $==$ `#f`
(C) $|A| > n$
5: **if** (`K-Appear` $==$ `#t` $\wedge$ $k ==$ `#t`) $\vee$ ($|A| > n \wedge$ `S-Appear` $==$ `#t` $\wedge$ `K-Appear` $==$ `#f`) **then**
6: HANDLE-K-APPEARANCE(S-SATURATED)
7: **else**
8: S-MEET-POINT(k)
9: **end if**

6 Tracing the SGVD$_\infty$ with sparse sensing

We now turn to the problem of tracing the SGVD$_\infty$ using the virtual sensor. We begin by introducing sparse sensing versions of the SGVD$_\infty$ tracing behaviors and discussing their differences from the behaviors that assume an omnidirectional sensor. We then show that these behaviors exhibit the same completeness properties as the omnidirectional behaviors, and discuss the results of simulations of the behaviors.

The robot again begins the mapping process by situating itself on the SGVD$_\infty$. The S-SITUATE behavior is somewhat different than the omnidirectional SITUATE behavior. When the robot is first "turned on," the virtual sensor is uninitialized and the robot knows only about what is seen by its real sensors. Though we could take a similar approach to the original SITUATE by using the virtual sensor immediately, there are complications because K-Appearances may occur and because the robot is not necessarily aligned at an orientation of $n\frac{\pi}{4}$. These issues can be overcome, but it is far simpler to take the approach in S-SITUATE and use only the sparse sensors to situate the robot, by approaching a single obstacle until there is guaranteed to be no closer obstacle.

Once the robot is situated on the SGVD$_\infty$, it traverses it in order to create its map. With a virtual sensor that closely emulates the functionality of an omnidirectional sensor, the sparse-sensing analogs to the omnidirectional SGVD$_\infty$ tracing behaviors, S-SATURATED, S-UNSATURATED, and S-MEET-POINT, are mostly equivalent to their omnidirectional counterparts. The main difference is that they must be capable of handling K-Appearances.

6.1 Handling K-Appearances

Recall that a K-Appearance occurs when a new obstacle is detected by one of the sensors at less than the equidistant range, indicating that the robot has strayed from the SGVD$_\infty$. The robot must return to the SGVD$_\infty$ by backtracking until the newly detected obstacle is at the equidistant range.

Behavior: S-UNSATURATED(t, k)
Traces a segment of the SGVD$_\infty$ separating the Voronoi regions of two sites, or on the boundary of the Voronoi region for the site represented by $t \in T$ (on the border of a region of equidistant points).

1: Let $n = |A|$
2: **repeat**
3: Move in direction θ, maintaining equidistance from object t to all other equidistant objects $r \in T \mid (*, t) \in A$
4: **until** one of the following conditions is true:
(A) $\exists s_i \in S$ such that ORIENT(s_i) $\notin \{\theta, \theta + \pi\}$ and (EC-DIAG(s) == #t or IC-DIAG(s) == #t)
(B) $\exists r \in T | d(r)$ == saturation distance
(C) K-Appear == k $\wedge$ S-Appear == #f
(D) $|A| > n$
5: **if** (K-Appear == #t $\wedge$ k == #t) **then**
6: HANDLE-K-APPEARANCE(S-UNSATURATED)
7: **else**
8: S-MEET-POINT(k)
9: **end if**

The mechanics of this operation are relatively simple. Since the virtual sensor computes A as if the newly detected obstacle does not exist (because it is too close), the robot can use the S-SATURATED, S-UNSATURATED, and S-MEET-POINT behaviors to retrace the incorrect SGVD$_\infty$, as long as they terminate when the new obstacle is at the equidistant range. This is accomplished by modifying those behaviors to take an argument k. When the K-Appear flag (set by the virtual sensor) is equal to k, the S-SATURATED and S-UNSATURATED behaviors terminate. During "normal" operation, k == #t, and while the robot is backtracking, k == #f.

When a K-Appearance is detected by the virtual sensor, it sets the K-Appear flag to #t, causing the segment-following behavior in control at the time (either S-SATURATED or S-UNSATURATED) to terminate and call the HANDLE-K-APPEARANCE behavior. HANDLE-K-APPEARANCE initiates backtracking by turning the robot around, and then returns control to the behavior that called it, which terminates when the new obstacle is at the equidistant range, where a meet point should be placed.

While the robot is backtracking to handle a normal K-Appearance, it may encounter previously-discovered meet points. Since they are incorrect (there is a closer obstacle), they must be discarded. The S-MEET-POINT behavior checks for this condition and deletes the meet point. It also enforces the choice of a direction that takes the robot along a previously explored path; since it has been previously explored, there can be no K-Appearances as the robot returns along that path.

Figure 8 depicts this strategy for handling K-Appearances. It shows a scenario with several K-Appearances, and where in one case the robot places multiple incorrect meet points before it encounters a K-Appearance and backtracks.

An S-Appearance is a special type of K-Appearance that only occurs when an obstacle appears at the saturation distance, indicating that the robot used the S-SATURATED behavior to traverse a portion of the SGVD$_\infty$ that should have been traversed using the S-UNSATURATED behavior. The strategy for dealing with an S-Appearance is the same as for a K-Appearance, except that the S-MEET-POINT behavior is responsible for recognizing (by checking values of $\frac{df}{ds}$) when the robot has returned to the meet point where the robot should have switched to using the S-UNSATURATED behavior. The S-MEET-POINT behavior is given this responsibility since the virtual sensor is unable to recognize the termination condition internally, because the new obstacle is already equidistant to the one being tracked.

One other change is necessary to the S-MEET-POINT behavior for dealing with K-Appearances. Recall that once an unseen obstacle is farther than the equidistant range from the robot, the virtual sensor stops tracking it. This means that if the robot returns to the vicinity of the obstacle when navigating later, it

Behavior: S-MEET-POINT(k)
Behavior for determining valid outgoing paths at a meet point.

1: Let p = label of this meet point
2: **if** $(p, *, *) \notin V \vee ((p, A_p, *) \in V \wedge |A| > |A_p|)$ **then**
3: Add $(p, A, (U, T, \text{ang}[\cdot], \text{same}[\cdot]))$ to V
4: **else**
5: Restore virtual sensor state from V
6: **if** $\exists(\alpha, t) \in A \mid \alpha$ == "?" **then**
7: Perturb robot θ to determine unknown α values
8: Compute W, the set of all pairs $(\alpha, t) \in A)$ where direction θ lies on the boundary of the Voronoi region of the object t. If at the saturation distance, exclude any θ where $\frac{df}{ds}[\alpha](\theta) > 0$.
9: **if** at the saturation distance **then**
10: Compute Y, the set of all pairs $(\theta, (\alpha, t) \in A)$ where $\frac{df}{ds}[\alpha](\theta) = 0$ and $\forall_{(\beta,r)\in A | r \neq t}\ \frac{df}{ds}[\beta](\theta) \geq 0$
11: **end if**
12: Compute C, the set of unique θ in pairs from W and Y
13: **if** S-Appear == #t $\wedge$ Y is not empty **then**
14: S-Appear = #f
15: **if** K-Appear == #t **then**
16: Delete this meet point
17: **if** $\exists c \in C | c$ corresponds to an explored edge **then**
18: Choose c
19: **else**
20: S-SITUATE(#t)
21: **else**
22: Choose an element c from C *// direction to move*
23: $k \leftarrow$ #t
24: **end if**
25: FIX-ANG(c)
26: Start moving in the direction c
27: **if** $(c, *) \in W$ **then**
28: Choose t arbitrarily from the pairs $(c, t) \in W$
29: S-UNSATURATED(t, k)
30: **else**
31: S-SATURATED(k)
32: **end if**

Behavior: HANDLE-K-APPEARANCE(B)
Handle the appearance of an obstacle at closer than the equidistant range.

1: Stop and turn 180 degrees
2: Begin moving forward
3: Call B(#f)

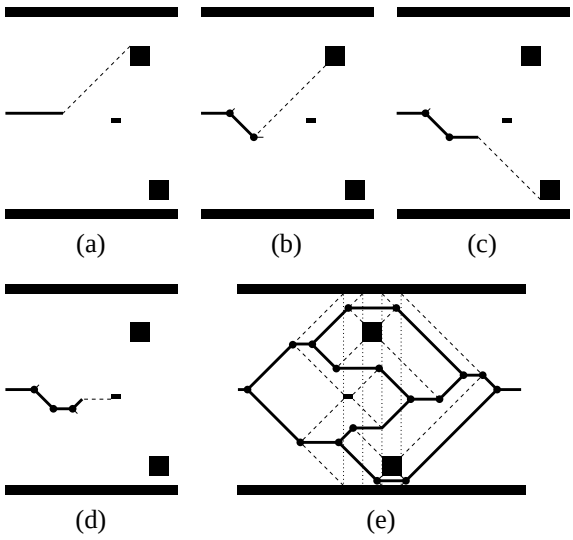


Figure 8: A situation in which the robot must handle K-Appearances. (a) The first K-Appearance — the top obstacle is closer than either of the walls being traced. (b) After handling the first K-Appearance, the robot has placed two incorrect meet points. (c) The second K-Appearance — the bottom obstacle is closer than the bottom wall and the top obstacle. (d) After handling the second K-Appearance, the robot replaces one of the incorrect meet points with another, but then detects another K-Appearance. (e) The final map after all incorrect meet points have been discarded and no more K-Appearances occur.

may cause a K-Appearance, which requires extra work for the robot to return to the SGVD_∞. Lines 1–5 of S-MEET-POINT avoid this by associating information with a meet point about the state of the virtual sensor at the meet point. When the robot returns to a previously-visited meet point, it can update its sensor state to avoid dealing with K-Appearances. This is useful for improving navigation efficiency, but is not strictly necessary for navigation since backtracking is sufficient. However, storing virtual sensor state at meet points *is* necessary to ensure that when the robot is backtracking to handle a K-Appearance, no other K-Appearances occur.

Aside from the modifications to handle K-Appearances, the behaviors for tracing the SGVD_∞ with sparse sensing are the same as the omnidirectional versions, aside from several small syntactical differences that do not change their functionality.

6.2 Completeness

The following lemmas and theorem show the correctness of the sparse-sensing behaviors and establish the completeness of our mapping.

Lemma 6.1 *The* S-SITUATE *behavior moves the robot from any starting location to a location on the correct* SGVD_∞.

Proof: The S-SITUATE behavior begins by moving forward until a sensor detects an obstacle within the saturation distance. The robot then approaches this obstacle until it is exactly upon it, so there can be no closer obstacles. Since the robot does not make use of the virtual sensor while doing this, unseen obstacles, appearances, and disappearances do not affect the robot's approach. By moving until the end of the wall is reached, the robot is guaranteed to be at a location on the correct SGVD_∞, since the SGVD_∞ contains terminal segments into all corners in the environment. Additionally, since the robot is at a corner, it can compute the normal angles of the adjacent walls and orient itself at the correct angle to trace the SGVD_∞ segment it is on. □

Lemma 6.2 *When a K-Appearance occurs, the robot returns to a point either on the correct* SGVD_∞ *or that will be detected as not on the correct* SGVD_∞ *only if another K-Appearance occurs.*

Proof: When a K-Appearance occurs, the HANDLE-K-APPEARANCE behavior is initiated, which causes the robot to stop and return along the (now incorrect) SGVD_∞ segment from which it came. At any meet point that is encountered, the S-MEET-POINT behavior requires the robot to move along a (now incorrect) SGVD_∞ segment that it had previously explored. Therefore no new K-Appearances can occur while the robot re-traces that segment, since they would have been detected on the first traversal. The robot stops backtracking only when all obstacles, both directly observable and unseen, are at the same distance. By Lemma 6.1, this must occur before the robot backtracks over all the previously explored segments in its SGVD_∞ since the robot's initial location after situating is guaranteed to be on the correct SGVD_∞. Thus, once all known obstacles are at the same distance, the robot is on the correct SGVD_∞ unless a future K-Appearance occurs that renders the meet point incorrect. □

Lemma 6.3 *Any portion of the* SGVD_∞ *traced by the robot that is incorrect because there is an obstacle closer than the equidistant range will eventually be discarded because of a K-Appearance.*

Proof: First, we show that the robot is guaranteed to sweep all portions of the environment within the saturation distance of the SGVD_∞ with its sparse sensors. If an obstacle has a wall of length less than the maximum distance between sensors, the robot may fail to detect it while the robot is on the

$SGVD_\infty$, and the obstacle causes a K-Appearance. The maximum distance between sensors occurs at the saturation distance from the robot.

If no such obstacles exist, the robot's sensors sweep all points in the environment within the saturation distance of the $SGVD_\infty$. The robot must move at least the maximum distance between sensors to trace each wall in the environment, because each wall is contained inside a Voronoi region whose boundary is on the $SGVD_\infty$ and is traced by the robot. If there is an obstacle within the saturation distance of the $SGVD_\infty$ that is occluded, there is another obstacle closer than it which determines the $SGVD_\infty$. Since each wall of the occluding obstacle constitutes a site and is traced to follow the corresponding $SGVD_\infty$ segments, the robot eventually circumnavigates the occluding obstacle and "uncovers" the occluded obstacle.

Thus, if obstacles with walls short enough to cause a K-Appearance exist, they will be detected as the robot sweeps portions of the environment close to $SGVD_\infty$ segments corresponding to larger obstacles (or obstacles that caused previous K-Appearances). The obstacles that caused K-Appearances will then be circumnavigated to trace the corresponding $SGVD_\infty$ segments and the space occluded by them will be swept. Therefore, all space within the saturation distance of the correct $SGVD_\infty$ will be swept by the robot's sensors and all obstacles in this area will be discovered.

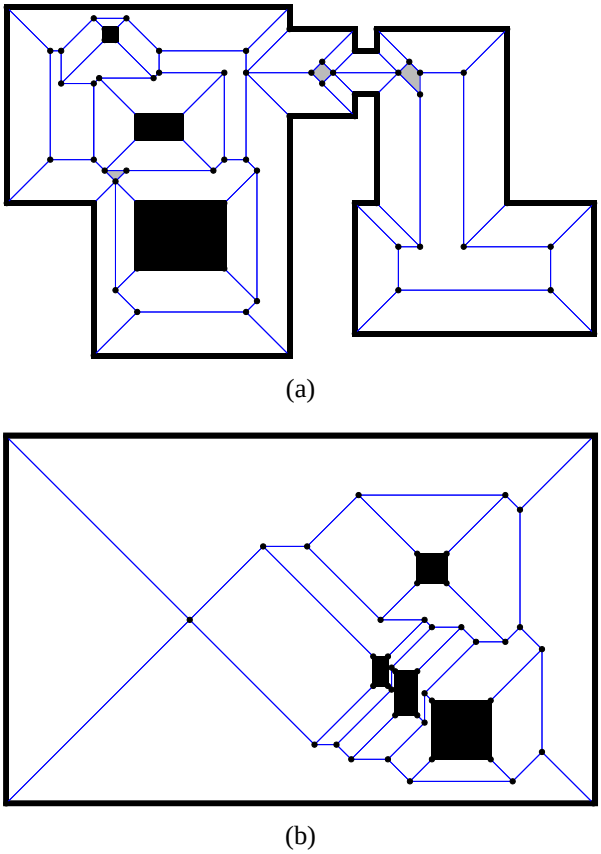
By Lemma 6.2, each K-Appearance causes the robot to return to the correct $SGVD_\infty$, discarding segments that are incorrect because of the new obstacle. No obstacle beyond the saturation distance from the $SGVD_\infty$ can cause a K-Appearance since the saturation distance is the maximum equidistant range. Once every K-Appearance is encountered, no incorrect segments remain in the $SGVD_\infty$. □

Theorem 6.4 *Together with the virtual sensor, the* S-SITUATE, S-SATURATED, S-UNSATURATED, HANDLE-K-APPEARANCE, *and* S-MEET-POINT *behaviors are sufficient to guide a robot with sparse sensing to trace a connected component of the $SGVD_\infty$ from any initial configuration.*

Proof: By Theorem 5.4, the virtual sensor correctly emulates an omnidirectional sensor except in K-Appearance situations. By Lemma 6.1, the S-SITUATE behavior is guaranteed to place the robot on the correct $SGVD_\infty$. And, aside from the handling of K-Appearance situations, the S-SATURATED, S-UNSATURATED, and S-MEET-POINT behaviors are identical to their omnidirectional counterparts. Therefore, except in K-Appearance situations, Theorem 4.5 applies.

By Lemma 6.2, the robot is always either on the correct $SGVD_\infty$ or able to return to it when a K-Appearance occurs. By Lemma 6.3, every obstacle in the environment that can cause a K-Appearance does so.

Therefore, these behaviors enable a robot with sparse sensing to trace a connected component of the $SGVD_\infty$ from any initial configuration. □



(a)

(b)

Figure 9: $SGVD_\infty$ maps produced in simulation by the behaviors from Section 6, using a virtual sensor implemented as described in Section 5.

6.3 Experimental results

The virtual sensor and $SGVD_\infty$ tracing behaviors have been implemented in simulation and tested in a variety of environments. The simulations assumed a robot with no sensor or actuator error and were primarily focused on verifying the correctness of the behaviors experimentally and were used mostly to test the behaviors in difficult scenarios such as K-Appearances.

Figures 9 and 10 show several maps produced in simulation using a robot with sparse, limited-range sensing. Figure 9(a) is a fairly simple scenario aside from the one small obstacle at the top, which causes a K-Appearance for large enough saturation distances. There are also several regions of equidistant points in the environment. Figure 9(b) shows the $SGVD_\infty$ of a difficult scenario in which multiple K-Appearances occur, similar to that discussed in Section 6.1.

Figure 10 shows how the $SGVD_\infty$ changes as the saturation distance (i.e. the sensing range) is increased. In these experiments, the robot initially starts near the left wall, facing left. With a small saturation distance, as in Figure 10(a), the robot encounters the wall and traces it at the saturation distance, but never discovers any of the obstacles interior to the environment since the are outside its sensing range. With a slightly increased sensing range, as in Figure 10(b), the robot encounters some of

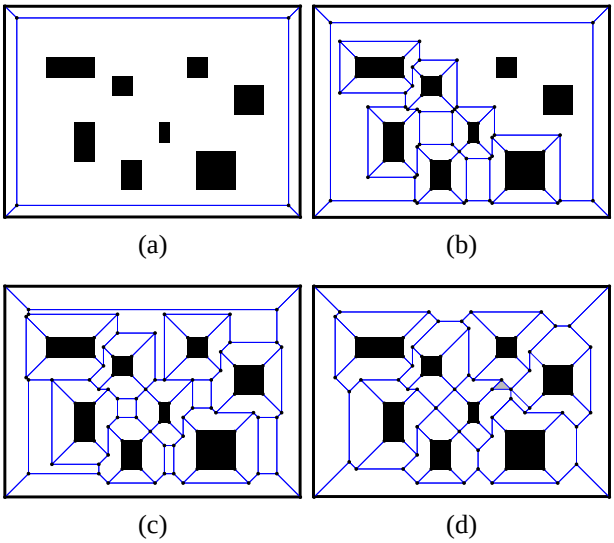


Figure 10: The effect on the SGVD_∞ of varying the saturation distance.

the obstacles and "jumps" between obstacles that are within the saturation distance of each other, but it still cannot find all of the obstacles in the environment. Increasing the sensing range further, the robot discovers every obstacle, as in Figure 10(c). Finally, with an "infinite" saturation distance, the robot produces the GVD_∞ of the environment, shown in Figure 10(d), since no obstacles are ever outside the saturation distance. All of the maps in Figures 9 and 10 were produced using the virtual sensor.

7 Conclusions

In this paper, we have presented algorithms that enable a robot capable of only sparse, short-range sensing to create a topological map of an enclosed rectilinear environment. The map is based on the saturated generalized Voronoi diagram of the environment, constructed under the L_∞ distance metric (the SGVD_∞). We have shown that the SGVD_∞ exhibits properties that are well-suited to tracing by a robot with sparse sensing. In particular, in a rectilinear world and under the L_∞ metric, the SGVD_∞ paths consist of straight line segments at orientations of $n\frac{\pi}{4}$. A robot with sparse sensors spaced at $n\frac{\pi}{4}$ is therefore able to detect the meet points where two or more SGVD_∞ segments come together.

We first developed behaviors for tracing the SGVD_∞ with a short-range omnidirectional sensor, and then extended these behaviors to the sparse sensing case. This was accomplished by introducing a "virtual sensor" that emulates the output of the omnidirectional sensor by tracking unseen obstacles, i.e., obstacles that were detected but that later disappeared from the sensors. The virtual sensor enables the robot to properly trace SGVD_∞ segments once the obstacles with Voronoi regions bisected by the segments have been detected. With sparse sensing, however, it is still possible for the robot to miss meet points when a small obstacle is observed too late by the robot's sensors. When such an obstacle is detected, our behaviors backtrack to return the robot to the SGVD_∞ and correct its map. We have shown both the omnidirectional and sparse sensing behaviors to be complete in that they will trace all portions of the SGVD_∞ reachable from the robot's initial location on the SGVD_∞. We believe these are the first such algorithms for robots with sparse sensing in arbitrary rectilinear environments.

Our virtual sensor could be viewed as maintaining a local map of the area around the robot that is updated using odometry information. However, the virtual sensor maintains relatively little state information when compared to approaches that keep detailed local maps about the area near the robot, e.g. by storing occupancy grid data or other detailed metric information.

Creating maps of an unknown environment takes longer when using a robot with less powerful sensing. One reason is that the robot can miss meet points on the SGVD_∞ and must backtrack to correct them. We also note that the virtual sensor requires odometry information, whereas the omnidirectional sensing behaviors do not. Potentially this makes the sparse sensing behaviors more fragile, but it should be duly noted that the robot is still able to trace exactly the same map as a robot with omnidirectional sensing.

8 Acknowledgement

Thanks to Howie Choset for discussions on the GVD. This work was supported by the NSF through award IIS–9983642.

References

Acar, E., Choset, H., and Atkar, P. 2001. Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and Voronoi diagrams. In *Proc. 2001 IEEE/RSJ Intl. Conf. on Intelligent Robots & Systems*, 1305–1311.

Aurenhammer, F. 1991. Voronoi diagrams — A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405.

Beevers, K. R. Topological mapping and map merging with sensing-limited robots. Master's thesis, Rensselaer Polytechnic Institute, Troy, NY, 2004.

Beevers, K. R. and Huang, W. H. Loop closing in topological maps. *2005 International Conference on Robotics and Automation (ICRA 2005)*, to appear, 2005.

Butler, Z. J., Rizzi, A. A., and Hollis, R. L. Complete distributed coverage of rectilinear environments. In Donald, B. R., Lynch, K. M., and Rus, D., editors, *Algorithmic and Computational Robotics: New Directions (WAFR 2000)*, 51–61. A K Peters, 2001.

Chatila, R. and Laumond, J.-P. 1985. Position referencing and consistent world modeling for mobile robots. In *Proc. 1985 IEEE Intl. Conf. on Robotics & Automation*, 138–145.

Choset, H. 1997. Incremental construction of the generalized Voronoi diagram, the generalized Voronoi graph, and the hierarchical generalized Voronoi graph. In *Proc. First CGC Workshop on Computational Geometry*.

Choset, H. and Nagatani, K. 2001. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Transactions on Robotics & Automation*, 17(2):125–137.

Deng, X., Kameda, T., and Papadimitriou, C. 1998. How to learn an unknown environment I: The rectilinear case. *Journal of the ACM*, 45(2):215–245.

Doty, K. and Seed, S. 1994. Autonomous agent map construction in unknown enclosed environments. In *Proc. MLC-COLT Workshop on Robot Learning*, 47–55.

Duckett, T. and Saffiotti, A. 2000. Building globally consistent gridmaps from topologies. In *Proc. 6th Intl. IFAC Symp. on Robot Control (SYROCO)*, 357–361.

Erdmann, M. 1995. Understanding action and sensing by designing action-based sensors. *International Journal of Robotics Research*, 14(5):483–509.

Grabowski, R., Navarro-Serment, L., Paredis, C., and Khosla, P. 1999. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308.

Huang, W. H. and Beevers, K. R. 2004. Topological mapping with sensing-limited robots. In *6th International Workshop on the Algorithmic Foundations of Robotics (WAFR 2004)*, 367–382.

Kirkpatrick, D. 1979. Efficient computation of continuous skeletons. In *Proc. 20th IEEE Annual Symposium on Foundations of Computer Science*, 18–27.

Kuipers, B. 1978. Modeling spatial knowledge. *Cognitive Science*, 2:129–153.

Kuipers, B. and Byun, Y.-T. 1991. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63.

Lee, D. 1980. Two-dimensional Voronoi diagrams in the L_p-metric. *Journal of the Association for Computing Machinery*, 27(4):604–618.

Lee, D. and Drysdale, R. 1981. Generalization of Voronoi diagrams in the plane. *SIAM Journal of Computing*, 10(1): 73–87.

Pagac, D., Nebot, E., and Durrant-Whyte, H. 1998. An evidential approach to map-building for autonomous vehicles. *IEEE Transactions on Robotics & Automation*, 14(4):623–629.

Rao, N. 1989. Algorithmic framework for learned robot navigation in unknown terrains. *IEEE Computer*, 22(6):37–43.

Rao, N. and Iyengar, S. 1990. Autonomous robot navigation in unknown terrains: Incidental learning and environmental exploration. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1443–1449.

Roy, N. and Thrun, S. 1999. Coastal navigation with mobile robots. *Advances in Neural Processing Systems*, 12:1043–1049.

Rybski, P., Zacharias, F., Lett, J.-F., Masoud, O., Gini, M., and Papanikolopoulos, N. 2003. Using visual features to build topological maps of indoor environments. In *Proc. 2003 IEEE Intl. Conf. on Robotics & Automation*, 850–855.

Shatkay, H. and Kaelbling, L. 1997. Learning topological maps with weak local odometric information. In *Proc. 15th Intl. Joint Conf. on Artificial Intelligence*, 920–929.

Thrun, S. and Bücken, A. 1996. Integrating grid-based and topological maps for mobile robot navigation. In *Proc. 13th Natl. Conf. on Artificial Intelligence*, 944–950.

Thrun, S., Gutmann, J., Fox, D., Burgard, W., and Kuipers, B. 1998. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Proc. Natl. Conf. on Artificial Intelligence (AAAI)*, 989–995.

Thrun, S. Robotic mapping: A survey. Technical Report CMU-CS-02-111, Carnegie Mellon University, Pittsburgh, PA, 2002.

Tomatis, N., Nourbakhsh, I., and Siegwart, R. 2002. Hybrid simultaneous localization and map building: closing the loop with multi-hypothesis tracking. In *Proc. 2002 IEEE Intl. Conf. on Robotics & Automation.*

Tovar, B., LaValle, S., and Murrieta, R. 2003. Optimal navigation and object finding without geometric maps or localization. In *Proc. 2003 IEEE Intl. Conf. on Robotics & Automation*, 464–470.