ISRN SICS-T--96/01--SE

View-Invariant Regions and Mobile Robot Self-Localization

Kristian T. Simsarian, Thomas J. Olson, N. Nandhakumar

SICS technical report
T96:01
ISSN 1100-3154

View-Invariant Regions and Mobile Robot Self-Localization

Kristian T. Simsarian[*], Thomas J. Olson[†], N. Nandhakumar[‡]

SICS technical report T96:01

Swedish Institute of Computer Science,
Box 1263, S-164 28 KISTA, Sweden

March, 1996

Abstract

This paper addresses the problem of mobile robot self-localization given a polygonal map and a set of observed edge segments. The standard approach to this problem uses interpretation tree search with pruning heuristics to match observed edges to map edges. Our approach introduces a preprocessing step in which the map is decomposed into *view-invariant regions* (VIRs). The VIR decomposition captures information about map edge visibility, and can be used for a variety of robot navigation tasks. Basing self-localization search on VIRs greatly reduces the branching factor of the search tree and thereby simplifies the search task. In this paper we define the VIR decomposition and give algorithms for its computation and for self-localization search. We present results of simulations comparing standard and VIR-based search, and discuss the application of the VIR decomposition to other problems in robot navigation.

[*]Kristian T. Simsarian is with the Swedish Institute of Computer Science, Box 1263, S-164 28 Kista, Sweden, E-mail: kristian@sics.se

[†]Thomas J. Olson is with the Personal Systems Laboratory, Texas Instruments, PO Box 655474, M/S 238, Dallas, TX 75265, E-mail: olson@csc.ti.com

[‡]N. Nandhakumar is with the Electrical Engineering Dept., University of Virginia, Charlottesville, VA 22903, E-mail: nandhu@virginia.edu

1 Introduction

An autonomous mobile robot that performs tasks in a large workspace typically represents its environment using some type of two-dimensional map. The map specifies the geometric layout of navigable space, the locations of objects of interest, and the robot's own location. It is used for path planning, for monitoring progress along a path, and for many other tasks involving the robot's relationship to its environment. The way in which the map is organized and represented has a major impact on the efficiency with which the robot can carry out these tasks.

In this paper we present a novel map representation and explore its application to a number of problems in robot navigation (this report serves as a more comprehensive explanation of the presentation in [15]). Our primary focus is on the *self-localization* problem, in which the robot seeks to determine its location with respect to its map using data acquired by its sensors. We assume that the robot's environment is represented by a 2D polygonal map, and that the robot is able to extract line segments corresponding to portions of the workspace boundary from sensor data (see Figure 1). The self-localization task then reduces to matching observed segments against map edges to recover the robot's position and orientation parameters (x, y, θ).

Our approach to self-localization is an extension of that of Miller [21] and Drumheller [7], who use interpretation tree search [9] to match range data against edges in a 2D polygonal map. The interpretation tree is a depth-first search tree in which each level of the tree associates an observed feature with a map edge. Heuristics (*e.g.* ordering constraints) can be used to make the search more efficient, but the branching factor of the search tree is of the order of the number of map edges. The search is thus very expensive for large workspaces.

A large part of the cost of interpretation tree search is due to failure to make use of visibility information that is implicit in the map. For example, having matched a map edge A to an observation, standard search methods may try to match edge B against another observation even if there is no position in the map from which both A and B are simultaneously visible. For large workspaces such as office buildings and factories, only a small part of the workspace is visible at any one time. Thus the cost of searching and rejecting inherently implausible matches may dominate the cost of search.

In order to reduce the cost of interpretation tree search, we introduce a preprocessing step during which the implicit visibility information is extracted and represented explicitly. This is done as part of an off-line initialization process, *e.g.* during map construction. The preprocessing involves decomposing the map into *view-invariant regions* (VIRs), a set of disjoint polygons characterized by the map edges that are visible from points within them. Together with each VIR the ini-

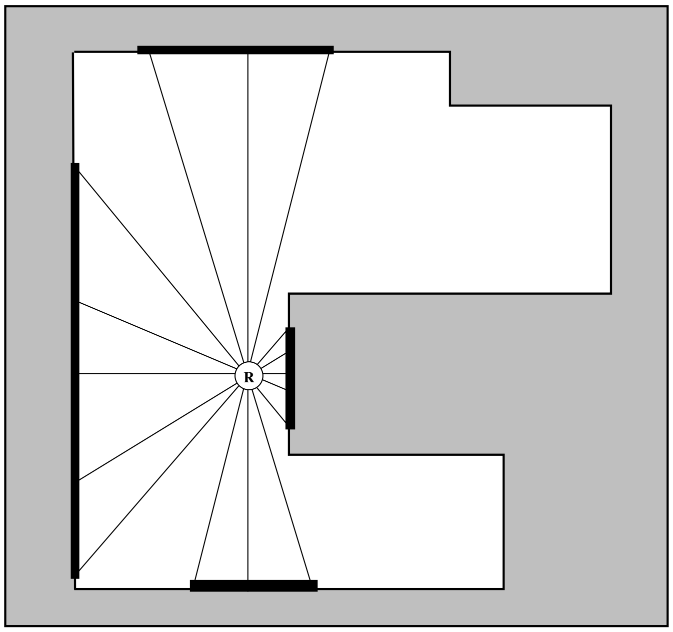

Figure 1: The self-localization problem : the robot (**R**) extracts straight edge segments (bold lines) from its sensor data and seeks to match them against known map edges.

tialization process stores the set of edges visible from within that VIR, and a set of heuristic features that characterize the world as seen from points inside the VIR. During self-localization, the robot uses local feature measurements to index into a set of VIRs with similar features, quickly isolating those that are likely to contain its current position. It then performs an interpretation tree search that is guided by the visibility information in the VIR set. For most maps this greatly reduces the cost of search.

The VIR decomposition has a number of applications beyond accelerating self-localization search. Because it captures information about visibility, it can be used for a variety of tasks involving perception and perceptual planning. These include path planning, searching for landmarks or other objects, updating approximately known positions during navigation, and self-localization in the presence of local ambiguities.

In the remainder of this section we review related work on visibility, mobile robots and the use of maps. Section 2 defines the VIR decomposition and develops algorithms for its construction and its application to self-localization. Section 3 presents simulation results illustrating the effect of the VIR decomposition on the cost of self-localization search, and section 4 describes other applications of the decomposition.

1.1 Related Work

1.1.1 Visibility

Self-localization using view-invariant regions is conceptually similar to object recognition using aspect graphs [27],[31],[10]. In both cases, a region of uncertainty (map location or viewing direction) is partitioned into subregions characterized by topological invariance, and the topology within subregions is then used to constrain search. The algorithms used to construct the two representations are based on the same general principles, though they differ substantially in their details.

The construction of aspect graphs depends on the camera model adopted for the object recognition task. When orthographic projection is used, it is common to tessellate the surface of an imaginary sphere, known as the Gaussian sphere, surrounding the object [13],[14]. Each surface patch of the sphere corresponds to a topologically invariant view. This tessellation may be achieved in one of two ways. (1) The surface is first partitioned into a large number of identical facets. Adjacent facets corresponding to the same view or aspect are merged. This approach, however, is limited by the resolution of the initial partitioning of the surface of the Gaussian sphere. (2) Alternatively, for convex polyhedral objects, each face defines a great circle on the Gaussian sphere. These circles partition the surface into regions corresponding to invariant aspects of the object. The latter approach has been extended for nonconvex polyhedra [11]. Visual events (that correspond to the edges of the aspect graph) are noted to be of only two fundamental types: coincidence of the projections of an edge and a vertex, and intersection of the projections of three nonadjacent edges. A specific event describes a curve on the Gaussian sphere. As before, the intersection of these curves forms a graph which is the dual of the desired aspect graph.

Deriving aspect graphs under perspective projection is more similar to our task of computing view-invariant regions for a planar polygonal map. The geometric incidence lattice method [12] computes the intersections of planes with each other which may be followed by a test to determine which planes are visible from a volume [31]. Alternatively, a *plane sweep* technique may be used to compute the aspect graph [26]. This technique could be applied in a two-dimensional space where line segments may consist of the edges of a polygonal map. The line sweep would provide information useful to the construction of the View-Invariant regions defined above. However, such an approach is less straightforward than the VIR extraction scheme adopted in this paper.

A number of other researchers have used visibility and topological constraints as aids to robot navigation. Levitt and Lawton [20] describe the use of topological information to obtain qualitative descriptions of robot location in landmark-based navigation. Talluri and Aggarwal [33] describe a map representation that is closely related to the VIR decomposition. In their approach the robot is in an outdoor environment. The boundary polygon is fixed at the outer limits of the map and does not generate region boundaries. Objects such as buildings in the environment are partitioned into convex pieces and can generate boundaries similar to those in aspect graphs. Occlusion of objects is taken into account and the 2D map is partitioned into "edge visibility regions" that correspond to unique views of the scene.

1.1.2 Mobile Robot Self-localization

The simplest possible approach to sensor-based self-localization is to add readily detectable and distinguishable beacons to the environment, as in the HILARE project [5]. Given observations of a sufficient number of beacons, the robot can locate itself by triangulation. These methods depend on relatively open environments to guarantee that some landmark or beacon is always visible. They have the advantage of simplicity, and (unlike the methods described below) do not depend on particular representations of the robot workspace.

Many researchers have chosen to represent 2D workspace maps by means of arrays or *occupancy grids* [23], in which the contents of each array cell reflect the robot's certainty that there is an obstacle at the corresponding spatial location. Occupancy grids are most often used with low-resolution, noisy sensors such as sonar, particularly in situations where the workspace map is either not known a priori or changes frequently. This basic idea has been extended to include representation of positional uncertainty [8],[3],[19]. Moravec and Elfes [23] describe a method of self-localization in occupancy grid maps by correlation. Robot location is determined by correlating new occupancy grids made from unknown positions with the occupancy grid that forms the map. Since the tessellation size is usually on the order of 10cm, this method is computationally expensive when applied to large spaces that include many rooms.

Our work follows Miller, Drumheller and others (*e.g.* [21],[7],[17]) in assuming workspace maps consisting of two-dimensional polygons. This type of map can be constructed from CAD representations of the workspace or from architectural drawings. Self-localization with polygonal maps is usually based on the interpretation tree search as described above [9].

1.1.3 Other Related Work

Miller [22] discusses several advantages of decomposing polygonal maps into disjoint regions, and proposes two decompositions. One is based on the Voronoi diagram and the other on the number of constraints on the robot's position that can be inferred from local information. These decompositions facilitate refining the robot's estimate of its own position, provided that it has an approximate position estimation. They do not, however, address the initial self-localization problem. It was this work that stimulated our interest in map decompositions and led to the approach taken here.

2 View-Invariant Regions

The primary contribution of our approach to self-localization is the use of *view-invariant regions* (VIRs) to limit the branching factor of the self-localization tree search. In this section we define the VIR decomposition, give algorithms for its construction, and describe its application to the self-localization problem. In order to simplify the exposition we assume here that the map is a simple polygon without holes. The definitions and algorithms generalize readily to maps with holes, however.

The definition of view-invariant regions is based on the concept of *visibility* in computational geometry (see *e.g.* [24]). A point $\mathbf{y}$ is visible to a point $\mathbf{x}$ if the line segment $\mathbf{xy}$ is interior to the polygon that contains the points $\mathbf{x}$ and $\mathbf{y}$. An edge $\mathbf{E}$ of the polygon is visible from point $\mathbf{x}$ if any point on $\mathbf{E}$ is visible from $\mathbf{x}$. Call the set of edges visible from some point $\mathbf{P}$ in the polygon the *label set of P*. The VIR decomposition of a polygonal map is a set of polygons $v_1, \ldots, v_k$ having the property that all points in any one of the polygons have the same label set, that the relative interiors of the polygons are disjoint, and that the union of the polygons is the original map[1].

Intuitively, what this definition means is that motion within a VIR cannot change the apparent topological structure of the world; the same set of edges and vertices is visible from every point in a given VIR. Only when a VIR boundary is crossed can the set of visible edges change. Figure 2 shows the VIR decomposition of the room shown in figure 1. Notice that crossing a VIR boundary always causes an edge to appear or disappear. For example, crossing from region k to region m causes a portion of edge VI to become visible. Although VIRs are defined by edge visibility, VIR boundaries always correspond to places where one map vertex occludes another; crossing a boundary from one VIR to another always causes one or more vertices to be exposed or occluded. It is often more convenient to think of VIR boundaries in terms of vertex visibility.

2.1 Computing View-Invariant Regions

Our basic strategy for computing view-invariant regions can be summarized as follows: First, we find all *view boundaries* in the map. A view boundary is a line along which one vertex is occluded by another; on one side of the boundary the vertex is visible, while on the other it is not. Second, we split the polygon repeatedly along its view boundaries, ending with a set of disjoint polygons that are not crossed by

[1]For maps with holes, this definition is insufficient to insure that all points sharing the same label set are connected. In this case we add the constraint that points $\mathbf{x}$ and $\mathbf{y}$ are in the same VIR iff all points on the line segment $\mathbf{xy}$ have the same label set.

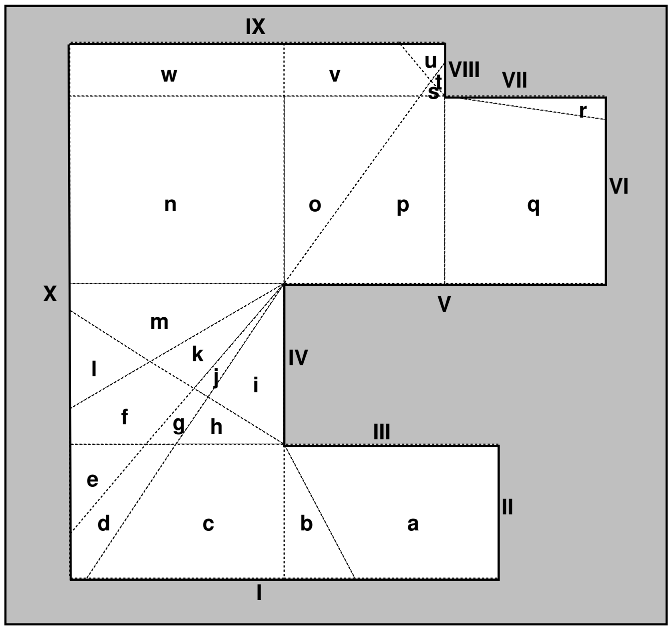

Figure 2: A polygonal map of a robot workplace showing its VIR decomposition. Edges are numbered with roman numerals and VIRs are labeled *a* through *w*. The map contains 10 edges and 23 VIRs.

any view boundary. Finally, we label each polygon with the set of map edges that are visible from all points within it[2]. These polygons are the VIRs.

2.2 The Algorithm

The first step in computing the VIR decomposition is to find all of the view boundaries. This is done by computing the *point visibility polygon* [24] for each vertex in the map. A point visibility polygon is defined relative to some reference point on the boundary or interior of the map, and consists of that portion of the map that is visible from the reference point [see figure 3].

The algorithm to compute the point visibility polygon consists of a traversal around the vertices of the map and will be referred to here as a *point visibility scan* (PVS). The reference point for which the PVS is done will be called the *reference vertex.* A point visibility polygon consists of two types of lines: 1) those that cross the interior of the map boundary, and 2) those that lie on the map boundary.

Lines of the first type correspond to the view boundaries described above. Cross-

[2]For efficiency much of the work of finding the labels is actually carried out in tandem with the splitting, but this does not change the fundamental nature of the algorithm.

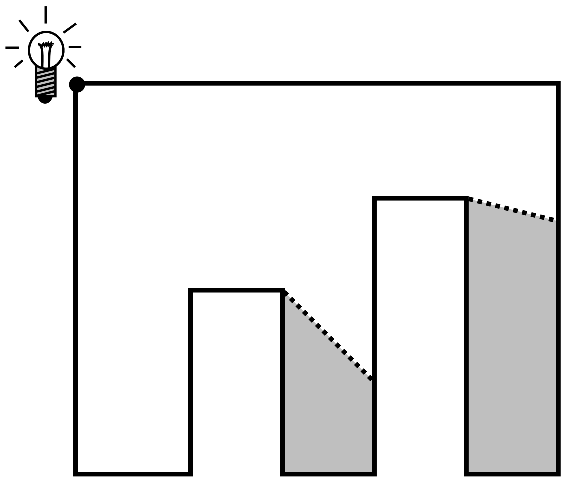

Figure 3: The *point visibility polygon.* The point visibility polygon may be thought of as the area interior to the map that would be lit by a light bulb (point light source) situated at the reference vertex. The portions of the map not included in the point visibility polygon are shown shaded. Dotted lines indicate the view boundaries.

ing a view boundary will cause the occlusion or the appearance of the reference vertex and some map edge. A view boundary divides the part of the map from which the reference vertex is visible from the part of the map from which the reference vertex is not visible. By analogy with figure 3, we call the side of a boundary from which the vertex is visible the *light* side and its opposite, the *dark* side.

Each line in a point visibility polygon that coincides with a map edge contributes its label to the PVS reference vertex's *label set.* At the end of the PVS computation, each map vertex's label set contains the labels of all edges visible from that vertex.

The complete computation to find the set of VIRs is composed of the following four steps:

I. Do a PVS for each map vertex to find view boundaries and vertex label sets, and add view boundaries to the map;

II. Calculate new interior vertices from view boundary intersections;

III. Traverse each view boundary to find label sets for new vertices;

IV. Extract VIRs from the augmented map (which now includes the view boundaries) and compute VIR label sets from the vertex label sets.

The first step of map construction is to perform a point visibility scan (PVS) for each map vertex, using the algorithm given in [24]. The algorithm consists of a single scan of the map edges, and runs in linear time with respect to the number of map vertices. During the PVS for each map vertex three types of information are collected: 1) The coordinates of the view boundary endpoints; 2) The label of the edge obscured by a particular view boundary; 3) The labels of map edges that are part of the point visibility polygon.

The view boundaries found from the PVS are lines that are interior to the map polygon and form the divisions between the different VIRs. At least one of the view boundary endpoints will always be a map vertex.

The reference vertex has two connected map edges. Crossing any view boundary created by this reference vertex will occlude or expose one of those map edges or an edge behind it. The label of the occluded edge is easily calculated during the PVS and is stored as the label for the current view boundary. This label is used during the view boundary traversal in step III (see figure 4). The names of the edges of the point visibility polygon that are not view boundaries are used in step IV to calculate the label sets for the VIRs.

View boundaries that have been added to the map may intersect and create new interior vertices. The process of calculating these intersections and adding them to the map constitutes step II. In the final map there are three types of vertices[see figure 5]. All three vertex types form vertices of the VIRs found in step IV.

A VIR's label set is formed by intersecting the label sets of its vertices. The label sets for the first type of vertices were calculated in step I. Step III calculates the label sets for vertex types 2 and 3. This is done by a view boundary traversal.

Each view boundary is traversed once. The traversal begins at the map vertex end of the view boundary. As the traversal proceeds, labels for new vertices are calculated. If another view boundary is crossed, the label for the other view boundary is added or subtracted from the label set depending on whether crossing the boundary exposes or occludes the reference vertex that generated that boundary. Following the light bulb analogy, this would be crossing to the light or dark side, respectively (see figure 6).

Steps II and III convert the map to a planar directed graph with label sets at each vertex. The final step of the algorithm consists of finding all minimum-length counter-clockwise cycles in the graph. Each such cycle forms the boundary of a VIR, whose label set is the intersection of the label sets of the vertices on its boundary. Finding minimum-length CCW cycles is done by the obvious method: while there is an unused edge V_iV_j in the graph, search forward through the graph until V_i is found, at each vertex turning as sharply as possible to the left.

This completes the discussion of the off-line VIR extraction process. A more detailed description of map construction can be found in [30]. Section 2.3 describes how the VIR decomposition is used at run-time to speed up robot self-localization.

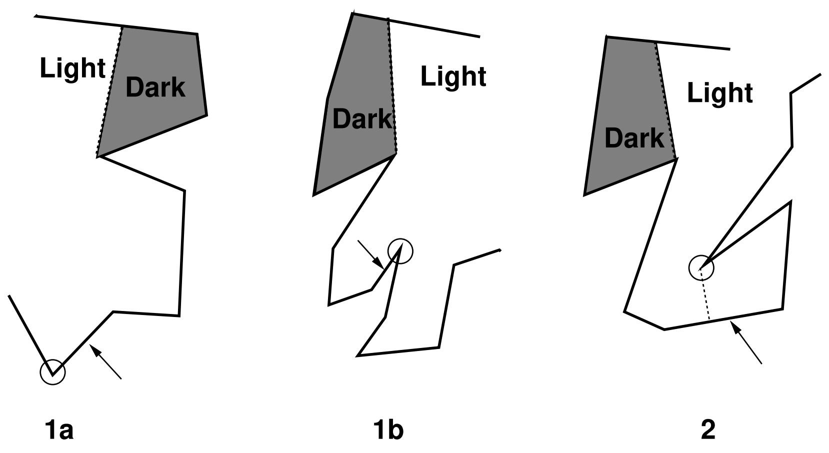

Figure 4: View boundary labels. In each case the reference vertex is circled and the view boundaries appear as dotted lines. Crossing a view boundary will always occlude or expose an edge, whose label becomes associated with that view boundary. In the figure the edges that provide labels for the view boundaries are marked with arrows. How the label is computed for any given boundary depends on whether the reference vertex is positioned as in cases 1a) and 1b) or as in case 2). In the first two cases the occluded or exposed edge is the edge connected to the reference vertex on the occluding vertex side of the view boundary. Case 2 is more complicated. Here a second view boundary is formed for which the roles of the occluding and reference vertices are reversed. The edge that provides the label for the first view boundary is the edge that the second view boundary strikes. Since this information is computed during the PVS from the nominally occluding vertex, handling case 2 requires only minor additional bookkeeping.

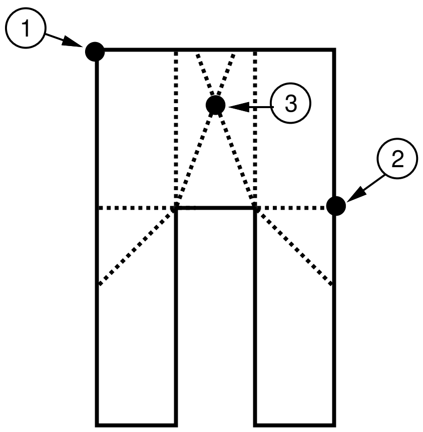

Figure 5: The three vertex types in the map. 1) Map vertices; 2) View boundary endpoints; 3) Interior view boundary crossings.

2.3 Self-localization with view-invariant regions

Self-localization using VIRs is a two-stage process. First, the line segments extracted from range data are used to hypothesize a small set of candidate VIRs in which the robot might be located. Second, observations are matched against map edges in the label sets of the candidate VIRs to constrain the robot's position as much as possible.

The second step is performed using an interpretation tree as described in [9],[21],[7]. The interpretation tree is a search method used to match features in a stored model against sensed data. In our work the stored model is a map of the robot environment consisting of polygonal edges. The sensed data are lines (*i.e.* portions of map edges) extracted from range sensor data. At each level in the tree, the search algorithm attempts to find the model edge corresponding to one of the sensed edges. Heuristics are used while expanding each node to prune the search tree. A path from the root node to a leaf node represents an interpretation, *i.e.* the path forms a complete set of matched observed edge to map edge pairs. In the standard interpretation tree search, the branching factor at each node corresponds to the number of edges in the model database. The branching factor for the interpretation tree in our approach is the number of edges that should be visible given the current edge to segment matchings, i.e. given the current position in the search tree. At the top level, where no edge to segment matchings have been made, the branching factor is just the cardinality of the union of of candidate VIR label sets. This set is typically much smaller than that of an entire map. Hence, the verification process involves a greatly reduced search space.

The degree of speed-up gained from the scheme rests on the first step, *i.e.* being

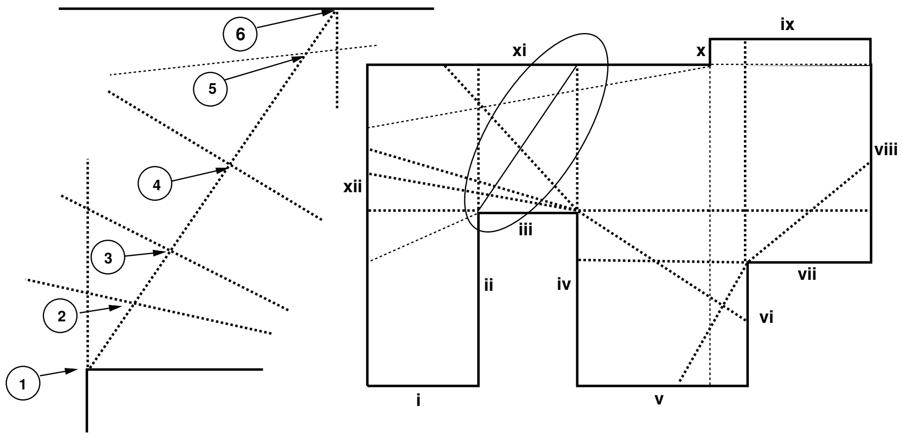

Figure 6: A view boundary traversal. The diagram on the left represents the area inside the oval on the right containing a view boundary (solid diagonal line) and its intersections with other view boundaries. Edges are numbered with roman numerals. The traversal is begun at the vertex labeled (1) with label set $\{iii, viii, ix, xi, xii, i, ii\}$. At point (2) the light side of the view boundary created by the vertex at the junction of edges *vii* and *viii* is entered. This adds the edge *vii* to the label set of the vertex at (2). Similarly at (3) and (4) *vi* and *v* are added to the current label set. At (5) edge *ix* is deleted from the label set after the view boundary is crossed. At (6) the traversed view boundary endpoint happens to coincide with another such endpoint resulting in the addition of *iv* to the label set of the vertex at (6).

able to select a small set of candidate VIRs. We propose to do this by using simple indexing functions that capture important features of the map as seen from within the VIRs. Functions for each VIR can be computed as part of VIR construction. During self-localization, functions computed from sensor readings can be used to select candidate VIRs. In our work to date we have used two indexing functions. The first is simply the number of visible edges. Thus if N_v edges are extracted from the range data, only VIRs whose label sets contain N_v edges are selected as candidates for verification. The second indexing function is based on the total length of the observed edges. VIRs whose label sets have significantly less total length are rejected as candidates. More robust and discriminating indexing functions are possible, e.g. separation between walls, shape features of sensed enclosures, and length of visible map perimeter. Methods of recognizing shapes from possibly erroneous or incomplete sensor data may be found in work by previous authors, e.g. [16],[6].

Once a set of VIRs is selected, an interpretation tree search is begun such that at each level the set of searched edges is the union of the current candidate VIR label sets. After a match has been made, VIRs whose labels sets do not contain that edge are removed and then the union is computed again. In this way the search is guided by the visibility information contained in the candidate VIRs. The search continues until a complete interpretation is found. During construction if we record along with each edge the union of VIR label sets that include that edge, then the union operations are effectively done in constant time during the localization search.

2.4 Complexity Issues

The cost of using VIRs for the task of self-localization can be divided into two components, one incurred during map construction and the other during self-localization. Both components depend on the number of VIRs that can be produced by a given polygonal map. This section gives a bound on the maximum number of regions and then discusses the cost of VIR construction and robot self-localization.

2.4.1 Upper and lower bounds on the number of regions

VIR regions are defined by their VIR vertices, and the number of vertices bounds the number of regions. In this section we derive a lower bound on the number of these vertices and hence a lower bound on any algorithm that performs VIR construction.

The total number of VIR vertices is the number of map vertices, n, plus the number of intersections that the view boundaries make with other view boundaries and map edges (*cf.* 5). Clearly the number of view boundaries is limited by the number of pairs of vertices, which is $\mathcal{O}(n^2)$ for an n-vertex polygon. If each view boundary crosses $\mathcal{O}(n^2)$ other boundaries and n map edges, then $\mathcal{O}(n^4)$ vertices could be produced. This can in fact occur in maps that are not simple polygons.

For simple polygons the maximum number of view boundary intersections is limited to $\mathcal{O}(n^3)$. Using the view boundary traversal as a model, it is easy to see that during a traversal each vertex can be exposed and occluded at most once. That is, the traversal can cross at most two view boundaries for each vertex in the map. Since the total number of view boundaries is $\mathcal{O}(n^2)$, and each can cross at most $2n$ other view boundaries, the total number of intersections is reduced to $\mathcal{O}(n^3)$. The VIR decomposition is unique, thus such a traversal on each view boundary will visit all possible VIR vertices that can be the result of intersection; the vertices of type 1 and 2.

Figure 7 gives two extreme VIR decompositions, one for simple polygons and one for polygons with holes. From this example it can be seen that the lower bounds on the number of VIR regions is $\Omega(n^3)$ for simple polygons and $\Omega(n^4)$ for polygons with holes. Since this lower bound meets the upper bound just given, the worst case number of regions are $\Theta(n^3)$ for simple polygons and $\Theta(n^4)$ for polygons with holes.

Because the output of VIR decomposition is the VIR regions and VIR vertices, these results provide lower bounds on the worst case time complexity for any algorithm that does VIR construction.

2.4.2 Complexity of Map Construction

The time complexity of map construction is the maximum complexity of the four construction steps. The PVS algorithm is $\mathcal{O}(n)$ for an n-vertex simple map polygon, or $\mathcal{O}(n \log n)$ for maps with holes [24]. Thus the first step runs in $\mathcal{O}(n^2)$ for a simple polygon and $\mathcal{O}(n^2 \log n)$ for a polygon with holes. The horizontal sweep intersection algorithm used for the second step runs in time $\mathcal{O}(n \log n + I)$, where I is the number of intersections [28]. By the argument in the previous section, I (and hence the complexity of the second and third steps) is in the worst case $\Theta(n^3)$ for simple polygons and $\Theta(n^4)$ for polygons with holes. The fourth step runs in time proportional to the number of VIRs in the map, which is bounded by the number of vertices found in step three. The time complexity for the entire map construction algorithm is therefore $\Theta(n^3)$ for a map that can be represented as a simple polygon and $\Theta(n^4)$ for a map that contains holes. Since the time complexity of the algorithms achieves the lower bound given in section 2.4.1, the algorithms are optimal within a constant factor. In addition the performance will improve with polygons not displaying worst case behavior. Since steps II, III, and IV run in time proportional to the number of intersections, the whole construction process will run in $\mathcal{O}(n^2 + I)$ for simple polygons and $\mathcal{O}(n^2 \log n + I)$, where I is the number of intersections and thus the size of the output.

2.4.3 Complexity of Self-localization

VIR-based self-localization proceeds in two stages. First, heuristic indexing functions based on local observations are used to identify a set of candidate VIRs. Second, a modified interpretation tree search that takes visibility into account is performed to identify the robot's position precisely. Thus the cost of self-localization using VIRs is the cost of candidate selection plus the cost of the search.

The cost of interpretation tree matching with n model edges and k observations is bounded by the size of the search tree, which is $\mathcal{O}(n^k)$. For non-pathological floor plans, however, pruning based on geometric constraints makes this bound of little practical significance. A more realistic estimate of the cost is $\mathcal{O}(n^2)$, since geometric constraints prune most branches of the search tree at the second level ([22]; see also section 3). The potential advantage of VIR-based self-localization comes from reducing the number of model edges considered at each level of the tree.

Let k be the number of observed line segments, v the number of VIRs in the candidate set, l_i the label set of the ith VIR, and C_s the cost of identifying the candidate set. The branching factor of the search tree is initially $|\cup_{i=1}^{v} l_i|$, so the cost of the search is bounded above by

$$\left|\bigcup_{i=1}^{v} l_i\right|^k + C_s.$$

The actual cost is typically lower for two reasons. First, as stated above, the use of geometric constraints tends to reduce the depth of the tree (and hence the exponent in the above expression) to around two. Second, at each node of the search tree our algorithm discards edges that are inconsistent with the hypotheses implied by that node. For example, if an observed edge is currently hypothesized to correspond to map edge m, VIRs whose label set does not contain m are provisionally excluded from the candidate set. In most cases this greatly reduces the branching factor at all levels above the root.

In the worst case the union of the candidate VIR label sets may have cardinality comparable to n. This occurs when either the individual VIR label sets or the number of candidates are large. The former problem arises when the workspace is structured so that there are many positions from which O(n) edges are visible, and leads to performance that is no better than that of the standard method. The latter situation occurs when the workspace contains many VIRs that are indistinguishable based on the heuristics used to identify candidates. For example, a workspace that is partitioned into many identical cubicles will produce this latter type of problem. In this case the consistency test described above may still produce a significant reduction of the branching factor.

The cost of VIR-based self-localization search depends heavily on the number of VIRs in the initial candidate set. There is an obvious tradeoff between the cost of candidate selection C_s and the cost incurred by searching the union of a larger number of candidates' visible edges. The optimal division of effort varies with the geometry of the map in question and the reliability and nature of the sensor information used for candidate selection. An important goal for further research is to explore this tradeoff and develop more and better heuristics for candidate selection.

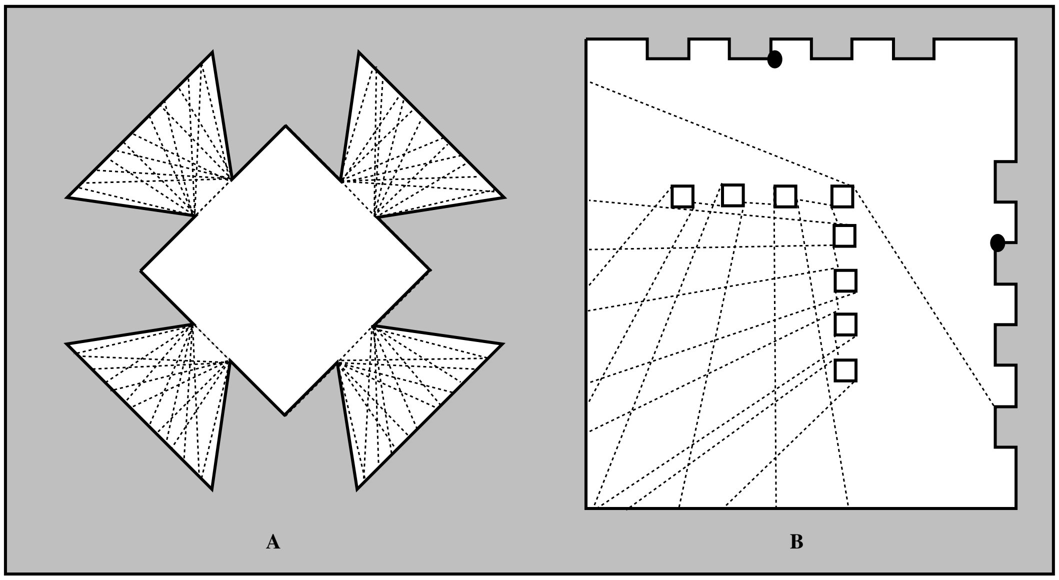

Figure 7: Examples that achieve worst case bounds for a simple polygon (A) and a polygon with holes (B). In both polygons some view boundaries are left out for clarity. In polygon A it can be seen that if there are $\Omega(n)$ niches, then each niche can contain $\Omega(n)$ view boundaries. Half of the view boundaries in each niche can cross the other half. This will create $\Omega(n^2)$ regions in each niche and therefore create $\Omega(n^3)$ regions in the polygon. Thus by the argument given in the text the worst case number of regions will be $\Theta(n^3)$ for simple polygons. In polygon B there are $\Omega(n)$ niches on each of the top and right sides and $\Omega(n)$ pillars in the middle of the room. Each niche vertex generates $\Omega(n)$ view boundaries (two for each of the $\Omega(n)$ pillars), and each such view boundary intersects $\Omega(n)$ other view boundaries. All of the boundaries associated with right-wall vertices cross all of the boundaries associated with top-wall vertices, producing a total of $\Omega(n^4)$ intersections. This gives the $\Theta(n^4)$ worst case bound.

3 Experiments

In order to explore the effect of VIR indexing on self-localization search, we collected performance data for interpretation tree matching with and without VIRs. The test data consisted of seven hand-drawn maps with varying numbers of edges. Each map was partitioned into VIRs automatically, using a variant of the algorithm described in the previous section. Two of the simpler maps are shown in figures 2 and 8.

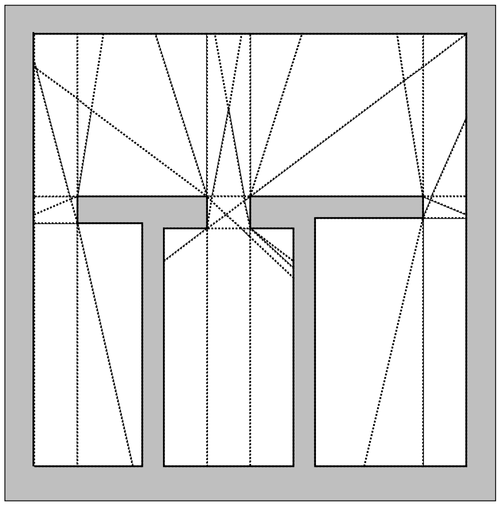

Figure 8: Map with 20 edges and 51 VIRs.

For each map, ten robot positions were chosen at random. For each position, a portion of each visible wall was extracted and added to the set of observations for that position. Each set of observations was then used as the basis for two self-localization searches. The first or "standard" search simply performed interpretation tree matching between the observations and the map. For the second, or "VIR" search, a set of candidate VIRs was selected using the edge-counting heuristic and then filtered using the visible perimeter heuristic. The union of the candidate VIRs label sets was calculated and these edges were used for the interpretation search. At each successive level in the tree, implausible VIRs were removed and the union was calculated again. Figure 9 shows the results for a typical trial in each of the seven maps used.

The interpretation tree algorithm used in all cases was an implementation of the Gaston and Lozano-Perez algorithm described previously. To insure a fair comparison, the implementation made use of the heuristics described in [7],[21] to determine the order in which map edges are considered at each level. The use of heuristics and geometric constraints makes computing the expected cost of the search extremely

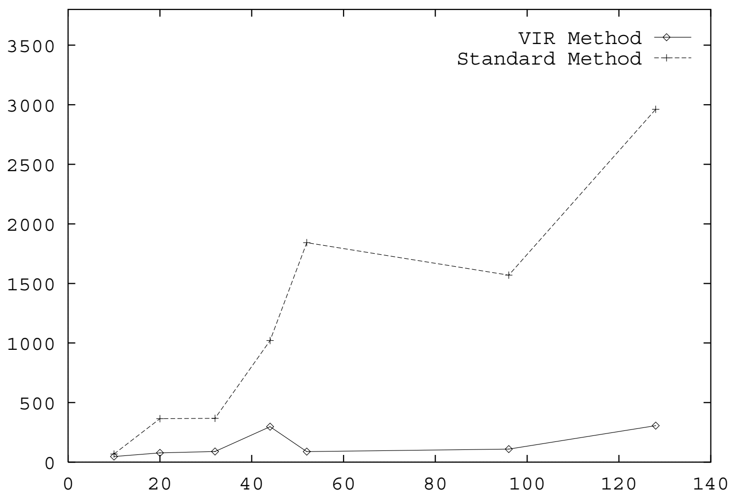

Figure 9: Plot of self-localization search performance for one trial on each of the seven different test maps. The X axis represents the number of edges in the map and the Y axis represents the number of nodes expanded during search. Each data point represents the number of nodes expanded in the actual search for either the VIR search method or the standard interpretation tree method.

difficult. Miller [22] reports that for an N-edge map the cost of a complete interpretation tree search is approximately proportional to N^2, and our experience tends to confirm this. In the experimental trials described above, 87% of the search paths were pruned at the second level of the tree, and another 11% were pruned at the third level. When the method is run until the first interpretation is reported, we found the average number of nodes expanded to be approximately $N^{1.7}$.

Figure 10 summarizes the results of all ten trials in each of the seven maps. The results lead to two conclusions. First, VIR search can result in substantial savings compared with standard interpretation tree search. The number of nodes expanded by the standard method is roughly quadratic in the number of map edges, while for VIR search it appears to grow much more slowly.

Second, variances for both searches tend to be large relative to the mean. This is because the running time of the algorithms are very sensitive to the quality of the heuristic information. For the VIR search a further speed-up could be expected with a better ranking of the candidate VIRs. The edge-counting and visible perimeter length heuristics used here provide no ranking information at all, so the algorithm can be expected to search half of the candidates during an average trial. In the worst case, this can cause VIR search to examine as many nodes as in the standard

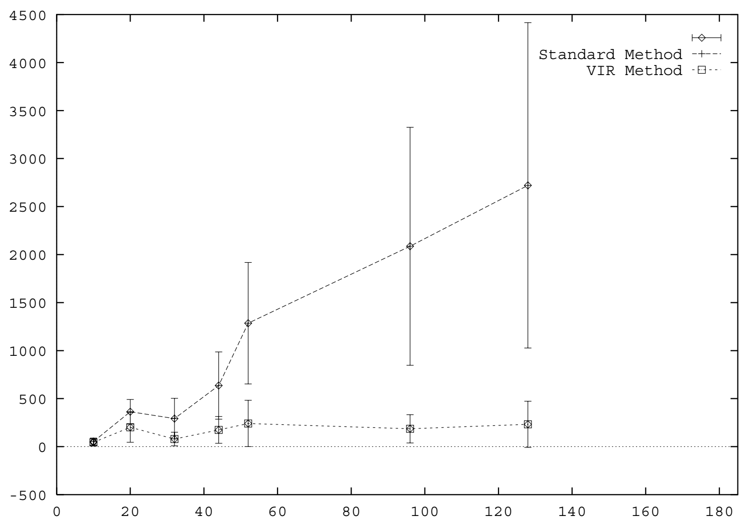

Figure 10: Plot of self-localization search performance for 10 trials on each of seven test maps with and without VIRs. The X axis represents the number of edges in the map and the Y axis represents the number of nodes expanded during search. Each data point represents the mean number of nodes expanded during ten trials, plus or minus one standard deviation.

search. The worst case occurs when there are several candidate VIRs whose label sets include a large proportion of the edges in the map, and the heuristics do a poor job of ranking the candidates. If under some metric the map continues to provide distinguishable VIRs as the map grows in size, then the VIR search can be expected to continue its relatively slow rate of growth. Hence the degree of speedup with VIR-based self-localization is directly linked to improvements in the quality of sensor data and heuristic selection.

4 Other Applications of View-Invariant Regions

The visibility information captured by the VIR decomposition is very powerful and general, and can be used for a number of other tasks in robot navigation and perceptual planning. It can also be applied to self-localization in other ways than that described in the previous section. In this section we discuss a number of these alternatives and applications.

4.1 Path Planning

A number of path planning algorithms make use of polygonal map decompositions to simplify the planning problem [25],[4],[22]. The general approach is to compute the region adjacency graph [1] of the decomposition, and then annotate it with geometric information (*e.g.* distances) as required by the particular algorithm in question. Path planning then reduces to finding a low-cost path through the graph. Clearly VIRs could serve as the basis for a path-finding algorithm of this type.

Suri [32] describes a method of finding the path of minimal link distance (that with the fewest turns) between two points using the chords of the weak visibility regions of a map. For any map edge E, the weak visibility region is the set of points within the map from which E is visible. This is just the union of all VIRs that have E in their label sets, so VIRs provide all of the information needed to implement Suri's algorithm.

4.2 Disambiguation

Any self-localization scheme that is based on local observations can produce multiple solutions when the environment is inherently ambiguous (*e.g.* contains many identical rooms). In this situation the robot must acquire more information before it can determine where it is. However, since this involves physically moving robot, it is potentially very time consuming and must be planned carefully.

VIRs and the VIR adjacency graph described above provide a basis for planning a series of moves that allow the robot to disambiguate its position efficiently. Assume that the robot has obtained a list of candidate solutions by interpretation tree search. Each solution consists of a particular position and orientation within some VIR. For any pair of candidate solutions, one of three conditions must be responsible for the ambiguity, and one of three solution methods will apply:

1. **problem** The solutions are in principle distinguishable from the hypothesized viewpoints; that is, the sets of edges visible in the two candidate VIRs differ geometrically. However, the robot's observations do not include the data needed to distinguish between the solutions.

solution The robot can disambiguate by obtaining better data. Exactly how this should be done depends on details of its sensor, but will usually involve moving closer to the feature(s) needed to make the distinction.

2. **problem** The solutions are indistinguishable based on observations from the two candidate viewpoints, but the VIRs containing the candidates are geometrically distinct. In this case the sets of edges that are visible from within the two candidate VIRs are identical in appearance.

   solution The VIR boundaries can be thought of as predictions about how the set of visible edges will change as the robot moves. If the two candidate VIRs have boundaries in different places relative to the robot, then the robot can distinguish between them by attempting to cross one of the boundaries and checking to see whether a vertex is in fact occluded or exposed.

3. **problem** Both the visible edges and the VIR boundaries for the two candidates are identical. In this case there is no way to distinguish between the candidates based on information gathered within the candidate VIRs.

   solution The robot must leave the current VIR. In order to decide where to go, perform a best-first search in parallel from both of the candidate positions along arcs of the VIR adjacency graph. Classify each VIR visited during the search according to the type of ambiguity to which it is subject, stopping when a pair of distinguishable VIRs is found. Let the cost function for the best-first search be the length of the path travelled, plus some measure of the cost of distinguishing between the two VIRs that terminate the search.

Note that these strategies are based on computations on the map and the VIR representation, and do not require the robot to move until after an optimal strategy has been chosen.

4.3 Searching for Objects

Many tasks to which mobile robots are well suited involve searching for some location, person or object; example tasks include delivering messages, fetching and transporting parts, and security applications. Because VIRs encode information about visibility, they are useful in planning move sequences that will accomplish these tasks.

Consider first the problem of locating some recognizable object whose position in the workspace is currently unknown. How can the robot determine what path it must follow to insure that it will see the object at some point? The VIRs are convex and span the workspace, so a path that visits every VIR must eventually bring the object into view. This requires more work than is strictly necessary, however. Simpler methods can be developed by modifying the VIR construction algorithm to construct view boundaries only for pairs of adjacent vertices; partitioning the map along these boundaries yields a simpler convex decomposition.

5 Conclusion

In this paper we have introduced a novel method of decomposing polygonal maps based on edge visibility, and used it to develop a fast algorithm for robot self-localization. Self-localization using our VIR decomposition improves on previous approaches by restructuring the search space to take visibility information into account. Thus eliminating search on implausible configurations. Experiments with synthetic data suggest that VIR-based search is substantially faster than previously published approaches. We have also presented an optimal-time algorithm for VIR construction, improving on the method described in [29].

VIR-based self-localization depends on a number of assumptions about the environment and the robot's sensors. The robot must be able to extract line segments corresponding to map edges from its range readings with reasonable accuracy. Drumheller [7] and others have shown that this is difficult with typical commercial sonar sensors, although signal analysis based on better models of sonar sensing has begun to address this problem [2],[18]. The speed-up of self-localization that our method provides depends critically on the accuracy of the initial selection of candidate VIRs. Better heuristic criteria for candidate selection are an important focus of our current research.

The self-localization algorithm presented here is faster than standard interpretation tree search. Its running time is very sensitive to features of the map and environment, leading to the the high variances observed in section three. The interpretation tree search method presented here displays a marked improvement in its worst case behavior over our previous method described in [30].

References

[1] Dana H. Ballard and Christopher M. Brown. *Computer Vision.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[2] Billur Barshan and Roman Kuc. Differentiating sonar reflections from corners and planes by employing an intelligent sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6), June 1990.

[3] Martin Beckerman and E.M Oblow. Treatment of systematic errors in the processing of wide-angle sonar sensor data for robotic navigation. *IEEE Journal of Robotics and Automation*, 6(2):137–145, 1990.

[4] Rodney A. Brooks. Solving the find-path problem by good representation of freespace. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 381–386, Pittsburgh, August 1982.

[5] Raja Chatila and Jean-Paul Laumond. Position referencing and consistant world modeling for mobile robots. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, St. Louis, 1985.

[6] Santanu Chaudhury, S. Subramanian, and Guturu Parthasarathy. Recognition of partial planar shapes in limited memory environments. *International Journal of Pattern Recognition and Artificial Intelligence*, 4(4):603–628, 1990.

[7] Michael Drumheller. Mobile robot localization using sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2):325–332, March 1987.

[8] Alberto Elfes. Sonar based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, June 1987.

[9] Peter C. Gaston and Thomas Lozano-Perez. Tactile recognition and localization using object models: The case of polyhedra on a plane. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):257–266, May 1984.

[10] Z. Gigus and J. Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, 1990.

[11] Z. Gigus and J. Malik. Computing the aspect graph of line drawings of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, Feb 1990.

[12] J. O'Rourke H. Edelsbrunner and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal of Computing*, 15:341–363, 1986.

[13] K. Ikeuchi. Generating an interpretation tree from a cad model for 3d-object recognition in bin-picking tasks. *International Journal of Computer Vision*, pages 145–165, 1987.

[14] k. Ikeuchi and T. Kanade. Modeling sensors and applying sensor model to automatic generation of object recognition program. In *Proc DARPA Image Understanding Workshop*, pages 697–710, April 1988.

[15] T.J. Olson K. T. Simsarian and N. Nandhakumar. View-invariant regions and mobile robot self-localization. *IEEE Transactions on Robotics and Automation*, 12(5):810–816, October 1996.

[16] Thomas F. Knoll and Ramesh C. Jain. Recognising partially visible objects using feature indexed hypotheses. *IEEE Transactions on Robotics and Automation*, 2(3), 1986.

[17] Eric Krotkov. Mobile robot localization using a single image. In *IEEE Proceedings of Robotics and Automation*, pages 978–983, 1989.

[18] Roman Kuc. A spatial sampling criterion for sonar obstacle detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7), July 1990.

[19] J. Leonard, H. Durrant-Whyte, and I. Cox. Dynamic map building for an autonomous mobile robot. In *Proceedings of the IEEE International Conference on Intelligent Robotic Systems*, 1990.

[20] Tod S. Levitt and Daryl T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44:305–360, 1990.

[21] David Miller. Two dimensional mobile robot positioning using onboard sonar. In *Proc. 9th William T. Pecora Memorial Remote Sensing Symposium*, pages 362–368, Sioux Falls, 1984. IEEE.

[22] David Miller. A spatial representation system for mobile robotics. In *Proceedings of the IEEE Int. Conf. Robotics and Automation*, pages 122–127, St. Louis, 1985.

[23] Hans P. Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE Int. Conf. Robotics and Automation*, pages 116–121, St. Louis, 1985.

[24] Joseph O'Rourke. *Art Gallery Thereoms and Algorithms.* Oxford University Press, 1981.

[25] Joseph O'Rourke. Convex hulls, voronoi diagrams, and terrain navigation. In *Proc. 9th William T. Pecora Memorial Remote Sensing Symposium*, pages 362–368, Sioux Falls, 1984. IEEE.

[26] F.P. Preparata and M.I. Shamos. *Computational Geomtery - An Introduction.* Springer-Verlag, New York, 1985.

[27] W. Brent Seales and Charles R. Dyer. Representing the dynamics of the occluding contour. In *Proc. SPIE: Sensor Fusion III: Perception and Recognition*, 1990.

[28] Robert Sedgewick. *Algorithms.* Addison-Wesley, Reading, Massachusetts, second edition, 1988.

[29] K. T. Simsarian, N. Nandhakumar, and T. J. Olson. Mobile robot self-localization from range data using view-invariant regions. In *IEEE Proceedings of the 5th International Symposium on Intelligent Control*, pages 1038–1043, 1990.

[30] Kristian T. Simsarian. View-invariant regions and mobile robot self-localization. Master's thesis, University of Virginia, May 1991.

[31] L. Stark, D. Eggert, and K. Bowyer. Aspect graphs and nonlinear optimization in 3-d object recognition. In *Proc. 2nd Int. Conf. on Computer Vision*, pages 501–507, 1988.

[32] Subhash Suri. On some link distance problems in a simple polygon. *IEEE Journal of Robotics and Automation*, 6(1):108–113, 1990.

[33] Raj Talluri and J.K. Aggarwal. Edge visibility regions - a new representation of the environment of a mobile robot. In *Proceedings of the IAPR Workshop on Machine Vision Applications*, pages 375–380, Tokyo, 1990.