Optimal Positioning of Sensors in 2D

Andrea Bottino and Aldo Laurentini

Dipartimento di Automatica e Informatica, Politecnico di Torino
Corso Duca degli Abruzzi, 24 – 10129 Torino, Italy
{andrea.bottino,aldo.laurentini}@polito.it

Abstract. Locating the minimum number of sensors able to see at the same time the entire surface of an object is an important practical problem. Most work presented in this area is restricted to 2D objects. In this paper we present a sensor location algorithms with the following properties. In principle, the algorithm could be extended to 3D objects. The solution given by the algorithm converges toward the optimal solution when increasing the resolution of the object. Limitations due to real sensors can be easily taken into account.

1 Introduction

Sensor planning is an important research area in computer vision. It consists of automatically computing sensor positions or trajectories given a task to perform, the sensor features and a model of the environment [1]. Sensor panning problems require considering a number of constraints, first of all the visibility constraint. Although in general the problem addressed is 3D, in some cases it can be restricted to 2D [2,7,11]. This is for instance the case of buildings, which can be modeled as objects obtained by extrusion. Placing sensors able to see entirely a 2D environment is similar, but not equal, to the popular Art Gallery Problem [4, 5], referring to the surveillance, or "covering" of polygonal areas with or without polygonal holes. The difference is that we are interested in observing only the *boundary* of the object. In addition, the task of our sensors could be to observe the exterior boundary of a set of general polygons (this problem is known as the fortress problem for a single polygon). Then, our 2D visibility problem can be called the internal or external edge covering problem. At present, no finite exact algorithm exists able to locate a minimum unrestricted set of guards (sensors) in a given polygon. In addition, no approximate algorithm with guaranteed approximation has been found. A detailed analysis of the edge covering problem compared with the classic Art Gallery problem is reported in [3]. Among other results, it is shown that in general a minimal set of guards for the Art Gallery problem is not minimal for the interior edge covering, and that also the edge covering problem is NP-hard. However, edge covering admits a restriction which makes practical sense and allows to construct a finite algorithm which supplies a minimum set of viewpoints able to cover internal or external polygonal boundaries. The restriction is that each edge must be observed entirely by at least one guard. It allows finding one or more sets of regions where a minimal set of viewpoints can be independently located. Observe that this idea is related with the practical requirement of observing a feature in its entirety [14]. In addition the solution provided by the algorithm asymp-

A. Sanfeliu et al. (Eds.): CIARP 2004, LNCS 3287, pp. 53–58, 2004.

totically converges to the optimal solution of the unrestricted problem if the edges are subdivided into shorter segments. Finally, the algorithm can easily take into account several other constraints.

2 The Positioning Algorithm

Here we briefly present the essentials of the 2D sensor-positioning algorithm. The steps of the algorithm are as follows.

1. Divide the space into a partition Π of maximal regions such that the same set of edges is completely visible from all points of a region.
2. Find the dominant zones (a zone Z of Π is dominant if no other zone Z* exists which covers the edges of Z plus some other).
3. Select the minimum number of dominant zones able to cover all the faces.

The idea of partition Π has also been proposed, in restricted cases, in [11] and [12]. Step 1), and 2) of the algorithm, as shown in the paper, can be performed in polynomial time. Step 3) is an instance of the well known set covering problem, which in the worst case is exponential. However, covering the edges using the dominant zone only usually substantially reduces the computation complexity. In addition, in many cases several dominant zones are also *essentials*, that is cover some edges not covered by any other dominant zone, and must be selected. Finally, polynomial selection algorithms exist with guaranteed performance (see [13]).

In the following paragraph we will detail the steps of the algorithm. The environment is assumed to consist of simple polygons. Partition Π is built by means of a particular set of lines, called the *active visibility lines*. Each resulting region will be associated with the list of the edges that are completely visible from that zone. This set can be built traversing the partition graph from an initial region whose set of visible edges is known. Observe that interior inspection is similar, with a polygon enclosing the workspace and defining the outer border.

2.1 Visibility Lines

A *visibility line* (VL) relative to an edge divides the space into areas where the edge is completely visible and areas where it is partially or totally hidden. The VLs relative to an edge start at one of its vertices and lye in the half plane opposite to the inner side of the object, as respect to the edge. The angle between this line and the edge is in the range [0,π]. Also, each of this lines has a positive side, which is the side closer to the originating edge. Examples can be seen in Fig. 1, where the arrows mark the positive side of the VLs.

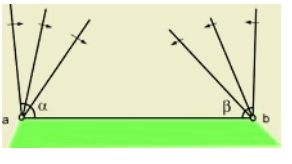

Fig. 1. VLs corresponding to the boundaries (a and b) of an edge

Each VL can have an *active* and an *inactive* part. Only the active VLs will be the effective boundary of the region of complete visibility of the edge they are related to. Active VLs can be found in two cases:

1. when the angle with its originating edge is π and the line does not enter the object in the proximity of the originating vertex (Fig. 2(a))
2. when the line is passing through another vertex of the object and, in the neighbourhood of this vertex, the inner side of the object lies on the negative side of the line (Fig. 2(b)). The inactive part of the VL is the segment connecting the originating and the intersecting vertices, while the active part is the remaining part of the VL

In both cases, any active VL stops if it intersects the object somewhere else (see for instance Fig. 2(a)).

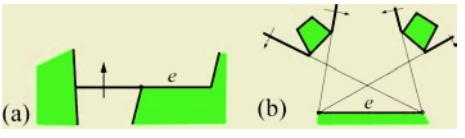

Fig. 2. (a) active VL making an angle of π with edge *e*. (b) VLs related to edge *e*; the active part of each line is bold. The arrows mark the positive side of the active VLs

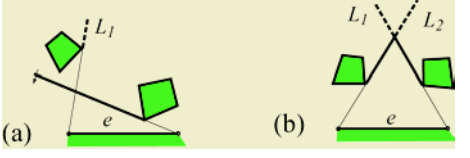

Fig. 3. Extra conditions to identifiy active VLs. (a) active VL intersecting an inactive VL. (b) two active VLs intersecting. The dashed lines represent inactive VLs

Two other conditions must be checked to identify an active VL:

1. if an active VL relative to an extreme of an edge intersect the inactive part of a VL relative to the other extreme of the same edge, then the second active VL is indeed inactive (e.g. L_1 in Fig. 3(a) is inactive)
2. if the active parts of two VL relative to the two extremes of the same edge intersect somewhere, then they both stop at the intersection point (e.g. L_1 and L_2 segments in Fig. 3(b) are not active)

We can associate to each active VL an operator ^, where `^j` means that the line is the boundary between a region where edge `j` is hidden and a region where the edge `j` is visible. The operator has also a direction, which points at the area where the edge is visible. In Fig. 4, we can see an example, where the operator ^ is applied to the VLs related to edges 5 and 8.

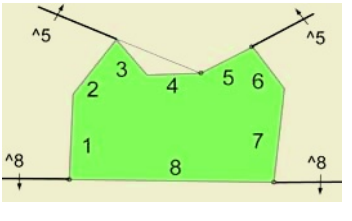

Fig. 4. Some active VLs (thick lines) and associated operators ^

2.2 The Partitioning Algorithm

Given the definition of active VL and operator ^, we can outline our partitioning algorithm:

1. found all the active VLs and the associate operator ^
2. intersect all the active VLs and subdivide the plane in regions
3. select one region and compute the set of visibile edges V(e1,…,en) for that zone
4. visit all the regions and compute their set of visible edges with the following rules:
   a. when crossing a boundary between R1 and R2 in the direction of the operator ^, the operand (j) is added to the set of visible edges of R2
   b. when crossing a boundary between R1 and R2 in the opposite direction of the operator ^, the operand (j) is removed from the set of visible edges of R2

An example of how the algorithm works is shown in Fig. 5, where the starting region has been marked with a different color. Note that several active VLs can be overlapping (or, in other words, we can have single VLs with multiple operators ^ associated). The dominant zones are highlighted by boxing their list of visible edges. The dominant zone in the upper left corner of the figure is also essential, and therefore it must be selected in the solution. It immediately comes that also the zone in the lower right corner must be selected to cover edges 6 and 7. The algorithm has been implemented, and some examples are shown in Fig. 6. For each example, the partition Π is shown, and the dominant regions that have been selected to cover all the edges are highlighted. In table 1 the number of objects, edges, regions of the partition and sensors placed for each case are summarized.

To find the active VLs, $O(n^2)$ pair of vertices must be considered. Checking if a line intersects the polygon at one the vertices can be done in constant time. For each edge, checking the extra conditions and finding the active segment requires intersecting each line with any other and sorting the intersections. Overall $O(n^2)$ VLs can be obtained in $O(n^3 \log n)$ time. A classic algorithm can create the partition Π in $O(n^4)$ time. The partition can also be constructed by a plane sweep algorithm in $O(p \log p)$ time, where p is the number of vertices of the partition (regions and edges also are $O(p)$) [8]. The total time for computing partition Π is $O(n^3 \log n + p \log p)$. Computing the visible edges of the starting region takes $O(n^2)$ time [9]. The time required for traversing the partition is $O(p)$ [10]. The overall time bound of step 1 is $O(n^3 \log n + p \log p + p)$. To find d dominant zones, we must compare the sets of visible edges of each region. This process can be shortened if we observe that a necessary condition for a region to be dominant is that all the positive crossing directions of the edges of the region lead to the interior of the region (except for the edges of the objects). Given c candidate found with this rule, d dominant regions can be found in $O(nc^2)$ time [3]. Steps 1 and 2 of the algorithm requires $O(n^3 \log n + p \log p + p + nc^2)$ time. Step 3 requires, in the worst case, exponential time. However, an interesting alternative could be using a greedy heuristic, which selects the region covering the largest number of uncovered edges each time, requiring polynomial time.

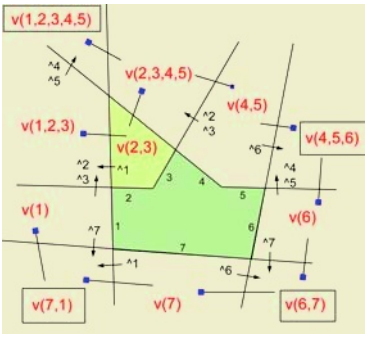

Fig. 5. Example of region labeling. The starting region is marked with a different color. The dominant zone are the one whose visible edges list is surrounded by a box

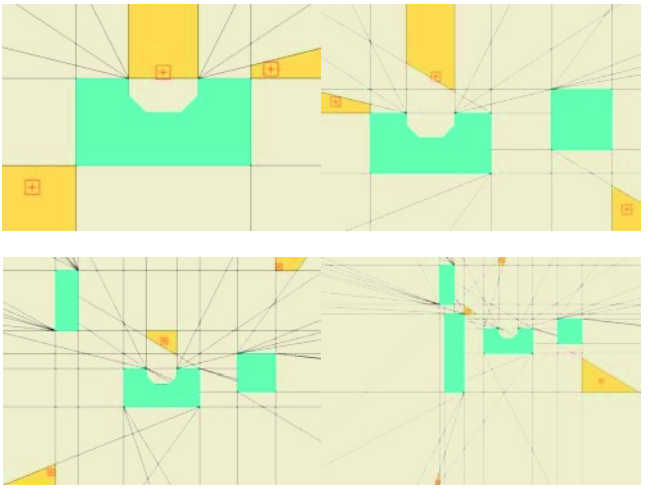

Fig. 6. Various examples showing several objects, the corresponding VLs and space partition, the dominant zones selected (drawn with a different background); for each zone, the cross represent a possible placement of a sensor

3 Conclusions

In this paper a method for positioning a minimum number of sensors into a polygonal environment has been presented for some sample cases. The approach has been implemented and results have been presented. Future work will be focused on extending the algorithm to 3D, maintaining the general idea of the 2D approach, that is: computing a partition Π of the viewing space of maximal regions Z_i, finding the dominant zones and selecting the minimum number of dominant zones able to cover all the faces. The idea is to find a suitable definition for active visibility surfaces, equivalent to the active VLs for the 2D case, in order to subdivide the space into regions where the same set of surfaces is completely visible.

Table 1. Total number of objects, edges, Active VLs, regions and sensors needed to cover all the edges of the objects of the examples of Fig. 6

Objects	Edges	Active VLs	Regions	Sensors
1	10	17	23	3
2	14	33	68	3
3	18	61	168	3
4	22	80	275	4

References

1. Scott W.R, Roth G. (2003) View Planning for Automated Three-Dimensional Object Reconstruction and Inspection. ACM Computing Surveys, Vol. 35(1), pp. 64–96
2. Kazakakis G. D., Argyros A.A. (2002) Fast positioning of limited visibility guards for inspection of 2D workspaces. Proc. Conf. On Intelligent Robots and Systems, pp.2843-2848
3. Laurentini A. (1999) Guarding the walls of an art gallery. The Visual Computer, vol.15, pp.265-278
4. O'Rourke J.(1987) Art gallery theorems and algorithms. Oxford University Press, New York
5. Shermer T.(1992) Recent results in art galleries. IEEE Proc. Vol. 80, pp.1384-1399
6. Nemhauser G., Wolsey L. (1988) Integer and Combinatorial Optimization. John Wiley& Sons
7. Danner T., Kavraki L.E. (2000) Randomized planning for short inspection paths. Proceedings. ICRA '00, vol. 2, pp. 971 – 976
8. Gigus Z, Canny J, Seidel R (1991) Efficiently computing and representing aspect graphs of polyhedral objects. IEEE Trans Patt Anal Machine Intell 13:542–551
9. Preparata F, Shamos M (1985) Computational geometry: an introduction. Springer, Berlin, Heidelberg, New York
10. Baase S (1988) Computer algorithms. Addison-Wesley, New York
11. Talluri R., Aggarwal J.K.(1996) Mobile robot self-location using model-image feature correspondence. IEEE Trans. On Robotics and Automation, 12(1), pp.63-77
12. Simsarian K.T., Olson T. J., Nandhakumar N. (1996) View-invariant regions and mobile robot self-localization, IEEE Trans. Robot. and Automat., vol. 12(5) , pp. 810-816
13. Nemhauser, Wolsey L. (1988), Integer and Combinatorial Optimization. John Wiley pag.466
14. K. Tarabanis, R.Y.Tsai, and Anil Kaul, (1996) Computing occlusion-free viewpoints. IEEE Trans. Pattern Analysis and Machine Intelligence, 18(3).pp.279-292