Randomized Planning for S hort Inspection Paths

Tim Danner   Lydia E. Kavraki

Department of Computer Science
Rice University, Houston, TX, USA
{tdanner,kavraki}@cs.rice.edu

Abstract

This paper addresses the following inspection problem: given a known workspace and a robot with vision capabilities compute a short path path for the robot such that the entire boundary of the workspace is visible from some point on the path. Autonomous inspection, such as by a flying camera, or a virtual reality architectural walkthrough, could be guided by a solution to the above inspection problem. Visibility constraints on both maximum viewing distance and maximum angle of incidence are considered to better model real sensors. An algorithm is presented for planar workspaces which operates in two steps: selecting art gallery-style guards and connecting them to form an inspection path. Experimental results for this algorithm are discussed. Next, the algorithm is extended to three dimensions and inspection paths are shown.

1 Introduction

The Problem and Motivation Consider the problem of exploring a workspace, sometimes called the watchman route problem [13]. The watchman route problem is defined as follows: given a workspace $\mathcal{W}$, compute a path p in $\mathcal{W}$ such that every point on $\partial\mathcal{W}$, the boundary of $\mathcal{W}$, is visible from some point on p. In the traditional art gallery problem, of which the watchman route problem is a direct descendant, "visible" is defined to mean that the line of sight from the guard to the point in question lies entirely in $\mathcal{W}$. Figure 1 shows an example of an inspection path.

There is an established need for autonomous inspection. A primary motivator of this work has been the idea of a flying camera that could inspect a space station for leaks, meteorite damage, parts that have fallen off, or other problems. Papers like [4] and [3] describe the development of such a camera, called AERCam, which stands for "Autonomous Extra-vehicular Robotic Camera".

Even on earth there are many places where inspection by humans is impractical, either because it is time-

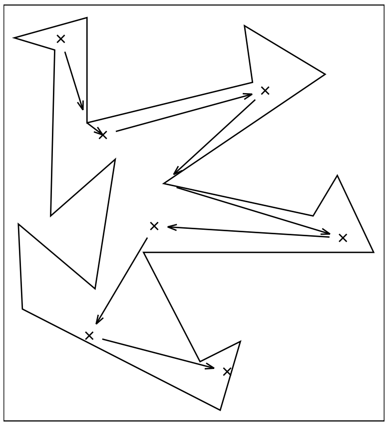

Figure 1: A path computed by our algorithm

consuming or because it is dangerous. For example, the paint in sewage stations and automated factories must be checked periodically. González-Baños et al describe a number of other applications for autonomous observers, and discuss a number of problems related to inspection, including model building, target finding, and target tracking [8]. The problem of inspection also finds use outside of robotics applications. For instance, architectural walkthroughs could be computed automatically, an excellent companion to a virtual reality exploration system such as the University of North Carolina's *Walkthrough* [1].

However, simple straight-line visibility is not a very realistic model of real sensors. For our purposes, we add two constraints on that definition, taken from [9]. First, the length of the line of sight must not exceed a maximum viewing distance. This corresponds to the limited range of real sensors, which often cannot register far away objects. Further, even if some object is near enough to be sensed, it might still be too far away for its details to be properly inspected.

Second, we constrain the line of sight's angle of incidence with the edge being observed. That is, the angle between the line of sight and the normal to the edge must not exceed a given maximum—60 degrees is a

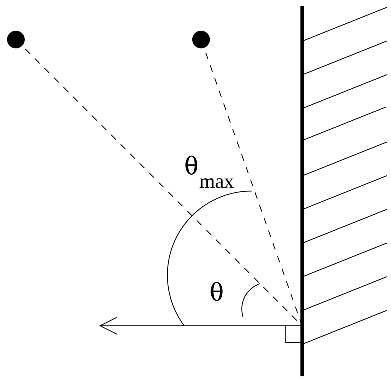

Figure 2: The incidence constraint. θ must be less than θ_{max}

typical value. Figure 2 illustrates the incidence constraint. Again, the constraint addresses two concerns: one, that the sensors will not pick up the surface at too grazing an angle, and two, that the sensing data will provide too little detail for proper inspection. By providing tunable visibility constraints, the algorithm can be easily adapted to different sensor characteristics and detail needs.

Related Work Finding the shortest watchman route, even without constraints on visibility, is not easy. Chin and Ntafos, who defined the problem, gave an $O(n^4)$ algorithm for solving it in simple polygons [2], but showed in the same paper that it becomes NP-hard in polygons with holes, even if those holes restricted to be convex or orthogonal. For a practical algorithm to inspect general polygonal regions, an approximating or randomized algorithm is needed.

Excellent work has been done in the area of exploring unknown polygons. By definition, a route which completely explores an unknown polygon is a watchman route. A competitive approximation algorithm for this problem was presented by Hoffmann et al in [10]. The competitive factor is 26.5. This is impressive because the path this algorithm computes *online* is within a constant factor of the length of the best possible route. However, for the inspection task, the workspace is known. Further, this algorithm only works in simple polygons and with unconstrained visibility.

González-Baños and Latombe described the model of constrained visibility we use [9]. Their work is in exploring an unknown workspace with discrete sensing events rather than continuous sensing over the entire path. They present an algorithm for determining the "next best view", which is where the robot should go next to best expand the known portion of the border. We expand on this work by optimizing the paths found for length rather than earliest learning, and by extending it to three dimensions.

Organization of this paper This paper is organized as follows. Section 2 describes our algorithm for inspecting a 2D workspace. Subsection 2.3 covers the experimental results obtained with that algorithm. The extension to three dimensions is introduced in Section 3, and preliminary results with that method are covered in Subsection 3.3. Finally, Section 4 ends the paper with a short discussion of possible future directions.

2 An Algorithm for Two-Dimensional Inspection

For many types of mobile robots, an inspection path computed with a two-dimensional model of the workspace is sufficient. Such a path would allow the robot to see all of the walls as shown on the floor plan, but not necessarily all of the shelves, ceiling features, etc. For other purposes, it will be necessary to consider the third dimension as well. In current real-world systems, sensing is a time-consuming operation. For inspection purposes, it may also be necessary to perform a detailed analysis on the data acquired by the sensors. Therefore, it makes sense to identify at which points on the inspection path it is actually necessary to perform the sensing operation and which are merely transportation. In light of this, our algorithm divides the path computation into two parts: solving an art gallery problem to choose a set of sensing locations (guards), and connecting those locations with a short path.

2.1 Selecting the Guards

Selecting a true minimal set of art gallery guards has been shown to be an NP-hard problem, even in simple polygons [13]. To model real environments, it will be necessary to handle holes, but this makes the problem even harder. To deal with these issues, we opt for a randomized approach. González-Baños and Latombe provide a suitable randomized, incremental algorithm in [9], on which this work is built.

The algorithm proceeds as a loop. At each iteration, a point p on the border $\partial\mathcal{W}$ of the workspace $\mathcal{W}$ which is not yet guarded is selected randomly. The border status is represented using a balanced tree for each segment in the border and is kept as compact as possible. The region which can see p is constructed (this is equivalent to the region p can see). That region is clipped to the range constraint, then clipped again to the incidence constraint. The resulting region is sampled k times. Each of these samples is evaluated as a potential new guard. The sample which can guard the most new length of border is retained as a guard and the border representation is updated to reflect the new guarded portion. The loop repeats until all of the border is guarded.

The unguarded portion of the workspace border decreases monotonically at each iteration of the loop, because the randomly selected point p is unguarded as a pre-condition of the loop, and is guarded as a post-condition. This ensures termination. However, under incidence constraints, environments with sharp interior angles may require a disproportionately large number of guards, which may be very hard to place. Small corners may be very hard to completely inspect because only a very small area can see the necessary parts. This algorithm performs poorly in such cases, but since the algorithm is incremental, it can be stopped early.

2.2 Connecting the Guards

Once the guards have been selected, it is necessary to find an order to visit them. Since a set of n points can be visited in $n!$ orders, the naïve algorithm will definitely not work. By connecting the guards in a graph, graph algorithms can be brought to bear on the problem. When viewed this way, the problem of finding an order for the guards can be recast as a Traveling Salesman Problem, which is known to be NP-complete in general graphs [7].

Approximation to TSP Fortunately, for graphs with edge weights that obey the triangle inequality,

$$d_{a \to b} + d_{b \to c} \geq d_{a \to c},$$

there is a simple 2-approximation to the traveling salesman problem. The triangle inequality holds for graphs arising from $\mathbb{R}^2$ or $\mathbb{R}^3$, which a graph of the guards would, and this approximation can be made to work.

In a graph whose edge weights obey the triangle inequality, a pre-order walk of a minimum spanning tree has total length less than or equal to twice the weight of a shortest Traveling Salesman tour [5]. However, the traveling salesman problem is defined for complete graphs, and the bound on this approximation only holds for complete graphs.

The shortest paths graph The graph we have chosen to use, which we call the shortest paths graph, consists of one node for each guard and one edge for each pair of guards. To each edge (i, j), a weight is assigned which is equal to the length of the shortest collision-free path from i to j. This may be a straight-line, or it may be more complex to avoid obstacles.

If the workspace is connected (that is, all parts are reachable from all other parts), the shortest paths graph will be complete. Since an inspection path cannot exist in a disconnected workspace, the assumption that the shortest paths graph will be complete is a reasonable one.

The triangle inequality property of the edge weights follows immediately from the fact that the shortest paths graph is composed of shortest paths: if $d_{a \to b} + d_{b \to c}$ is less than $d_{a \to c}$, then the edge from a to c must not actually be the shortest path. Therefore, the shortest paths graph is suitable for the minimum spanning tree-based approximation to TSP.

Computing a shortest path between two points in two dimensions is straightforward. For our purposes, it will be done with a simple search in another graph, the workspace-guard roadmap. The workspace-guard roadmap has one node for each vertex in the border of the workspace, $\partial\mathcal{W}$, and one node for each guard. Hence it includes the workspace visibility graph as defined in [12]. It has an edge between a pair of nodes i and j if and only if the straight-line path from i to j does not intersect any obstacles. The weight of an edge (i, j), if collision-free, is just the distance from i to j.

Optimizing graph building Because generating a complete graph of n nodes would yield n^2 edges, a shortcut for building the workspace-guard roadmap, and hence the shortest paths graph, is desirable. To keep the graph connection step sub-quadratic, when the sum of the number of guards and the number of vertices in $\partial\mathcal{W}$ is large, only nearby points are connected. This is accomplished by dividing the workspace into a regular rectangular grid such that the average number of points per cell is some small number, on the order of 10. Then connection is performed in a moving 3 by 3 window. For example, if the grid is 5 by 5, there will be 9 connection steps, one centered on each cell except the border cells. Since a minimum spanning tree only uses short edges, the effect of this approximation on the final path should be minimal, even though it reduces the number of calls to shortest-path from $O(n^2)$ to $O(n)$.

2.3 Experimental Results

We implemented the algorithm described above in C++ with the LEDA toolkit (see `http://www.mpi-sb.mpg.de/LEDA/`). The implementation supports computation of inspection paths in polygons with holes, which makes it easy to inspect either the interior or exterior of the workspace. By default, the interior is inspected. If an exterior inspection is desired, wrap the workspace in an enclosing rectangle and mark that rectangle as already guarded so that it will not influence guard selection.

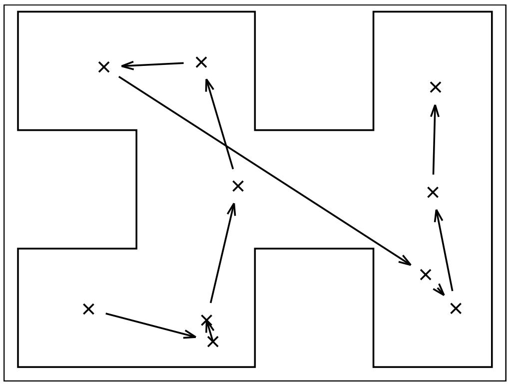

Figure 3: A simple example with both visibility constraints

Three examples here show paths generated by the code in polygons of various complexity. These three are actually instances of the Hilbert polygon as generated by LEDA. All times are on a Sun Ultra 10 with an UltraSPARC-IIi CPU at 300MHz and 128MB of system memory. The software is compiled with egcs 2.91, and linked against LEDA 3.8. In Figure 3, the incidence maximum is set to 60 degrees, and the range limit is equal to one unit of the underlying grid. The guards were selected in 0.55 seconds, and were connected to form the path in 0.07 seconds. In Figure 4, the incidence constraint is again 60 degrees, and the distance constraint was not set. The 27 guards were selected in 4.74 seconds and connected in 1.02 seconds. Finally, in Figure 5, neither constraint was set (this makes the problem somewhat simpler). The 161 guards were selected in 729.79 seconds, and connected in 328.13 seconds.

Profiling revealed that most of the computation time, especially in complex workspaces such as in Figure 5 is spent computing visibility polygons. Speeding that significantly would improve the execution time. By taking advantage of the maximum constraint and not considering workspace features which are far away, this should be possible in the future.

3 Three Dimensional Inspection

While an inspection path computed using a two-dimensional approximation of a real environment will do for a gross scan of the real workspace, a proper inspection path must be computed based on the true three-dimensional workspace. The two constraints on visibility, maximum distance and maximum incidence, remain the same in three dimensions. The general algorithm explained above will work largely unchanged in three dimensions. The primary problem is that, while it is relatively easy to compute a visibility polygon in two dimensions, it is very difficult in practice to compute a visibility polyhedron in three.

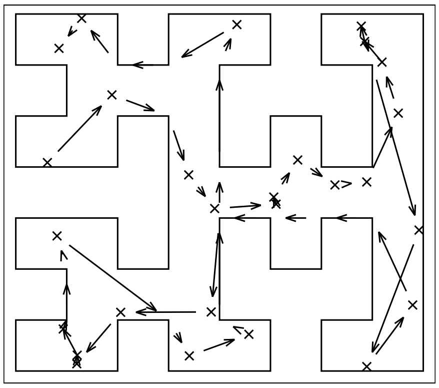

Figure 4: A larger example with a 60 degree maximum incidence

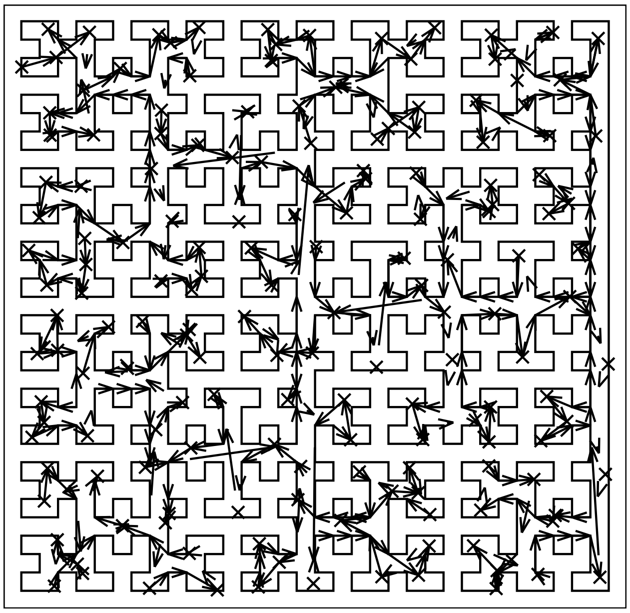

Figure 5: A large, complicated example (1026 edges)

3.1 Selecting the Guards

To use the guard selection algorithm as is in three dimensions, visibility volume computation would be needed for two purposes: to determine what portion of the surfaces of the environment a given point can see, and to determine the region to sample for potential guards for a point on the border. In the two dimensional case, these two purposes were served by the same visibility polygon code. In three dimensions, they can be treated separately, and doing so will allow us to avoid explicitly computing visibility volumes.

In the first case, what is really needed is a simple enumeration of the visible surfaces, similar to what is

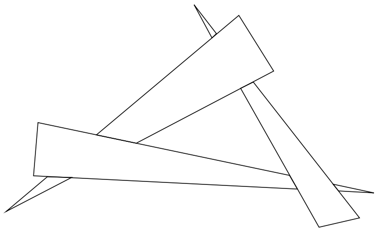

Figure 6: A circular dependence; the shown triangles have no front-to-back order

used in graphics. This process is detailed later in this section.

In the second case, it is only necessary to select random points in the intersection of the visibility polyhedron, the sphere with radius d_{max} that results from the distance constraint, and the infinite cone with angle $2 \times \theta_{max}$. The intersection of the sphere and the cone is easy to compute and sample. Each of these sample points needs to be tested to see if they lie in the visibility polyhedron by checking the line segment from the border sample point p to the sample for intersection with the various obstacles. Since the two functions of the visibility polyhedron in this algorithm can be fulfilled without an explicit representation, the problem of computing one can be avoided.

Visible Surface Determination To determine the effectiveness of a potential or actual art gallery guard, it is necessary to determine what obstacle surfaces are visible from the guard's position, subtract from those surfaces the portion that is visible from already-selected guards, and compute the total area remaining.

Clearly the face closest to the observer is completely visible, and the next closest face is also completely visible, except for any part which might be occluded by the first polygon. This suggests an algorithm which iterates over the faces in front to back order, with each face clipping those behind it. Such an algorithm faces three obstacles: determining the front to back order, resolving circular conflicts such as in Figure 6, and employing a data structure to represent what part of each face is still visible. In a polyhedral environment, the visible portion of any given face can be represented as a set of one or more disjoint polygons, possibly with holes. This data structure will need to support subtracting polygons from it and computing the remaining area. Alternatively, a conservative approximation could be made by breaking each face into small pieces and considering each piece either completely visible or not visible at all.

The problems relating to the front to back order are well-addressed by the binary space partitioning (BSP) tree [6]. In building a BSP tree, circular occlusion conflicts are resolved by splitting polygons as necessary. Once built, the BSP tree can be used to generate provide front to back orderings from any point in linear-time. For our implementation, we chose to use the BSP tree to solve these problems.

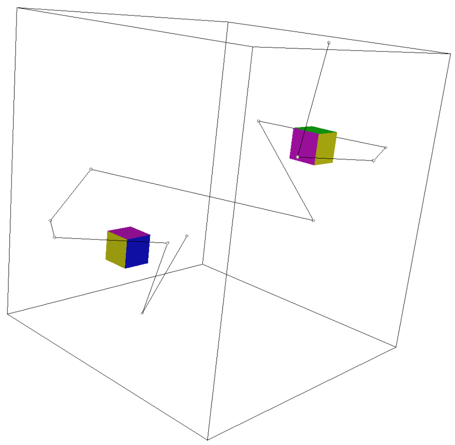

Figure 7: An inspection path for two cubes

3.2 Connecting the Guards

The procedure for connecting the guards in two dimensions will work largely unchanged. In the plane, obstacle vertices can be used to compute optimal shortest paths with a simple algorithm, but there is no easy analog to this procedure in three dimensions. Instead of augmenting the guard roadmap with workspace vertices to obtain the workspace-guard roadmap, random points in the free space are chosen, in a manner similar to probabilistic roadmaps [11].

3.3 Preliminary Results

There is an early implementation of this algorithm, using the conservative approximation to visibility described above. The figures shown in Figures 7 and 8 were generated with that implementation. The black lines represent the computed paths, and the white dots represent the locations where a sensing operation must take place.

Figure 7 is a path to inspect a pair of simple unit cubes in a workspace which extended ten units beyond the bounding box of the cubes in each direction. The incidence maximum was $\pi/4$, and the distance constraint was 3. The path was computed in 20 seconds. All times for the 3D implementation were taken on the same computer (Sun Ultra 10) as in Section 2.3.

Figure 8 is a path to inspect 4 cubes and 3 tetrahedra. The incidence and distance constraints were the same as in the single cube case, and the path was computed in 143 seconds.

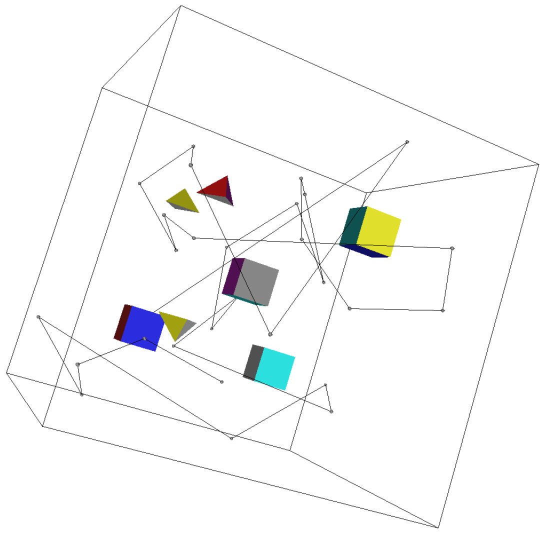

Figure 8: An inspection path for four cubes and three tetrahedra

4 Discussion

This paper considered the inspection problem under visibility constraints and described an algorithm for computing inspection paths in two dimensions. Experimental results with this algorithm were discussed. Next, the inspection path algorithm was extended to operate in three dimensions using a binary space partitioning tree to compute the necessary visibility information.

Much work remains to be done in this area. Other criteria of path goodness should be considered, such as dynamics. For a flying camera in space, the length of a path is much less important than the amount of propellant expended in following it. Instead of having specific sites that must be visited to inspect the entire workspace, a set of small regions could be selected, such that one sensing operation in each region would be sufficient to inspect the whole workspace. This kind of flexibility—to visit an area rather than a specific point—would probably be useful in optimizing a path for measures other than simple length.

The work so far has used a model of camera which is omnidirectional. While some range sensor implementations have this useful property, most visual cameras do not. It would be straightforward to modify this algorithm to deal with directional cameras, both in two and three dimensions. However, in optimizing the fuel usage of the path, it becomes important whether the camera is fixed relative to the direction of motion of the robot, or whether it is mounted on a turret and free to point in some other direction.

Acknowledgments Work on this paper by Lydia Kavraki has been supported in part by NSF CAREER Award IRI-970228 and NSF CISE SA1728-21122N. Work on this paper by Tim Danner has been supported by IRI-970228 and the corresponding REU portion of this award. The authors would like to thank Hector González-Baños, Alon Afrat, T. Murali, and the members of our research group, the Physical Computing Group at Rice, for their comments.

References

[1] F. Brooks et al. Six generations of building Walkthrough. Report TR92-026, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, NC, 1992.

[2] W.-P. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, 28:39–44, 1988.

[3] H. Choset, R. Knepper, J. Flasher, S. Walker, A. Alford, D. Jackson, D. Kortenkamp, R. Burridge, and J. Fernandez. Path planning and control for aercam, a free-flying inspection robot in space. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1396–1403, Detroit, MI, 1999.

[4] H. Choset and D. Kortenkamp. Path planning and control for aercam, a free-flying inspection robot in space. *ASCE Journal of Aerospace Engineering*, 1999.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[6] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.

[7] M. Garey and D. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, 1979.

[8] H. González-Baños, L. Guibas, J.-C. Latombe, S. LaValle, D. Lin, R. Motwani, and C. Tomasi. Motion planning with visibility constraints: Building autonomous observers. In Y. Shirai and S. Hirose, editors, *In Robotics Research - The Eighth International Symposium*, pages 95–101, 1998.

[9] H. González-Baños and J.-C. Latombe. Planning robot motions for range-image acquisition and automatic 3D model construction. In *Proc. AAAI Fall Symposium Series*, 1998.

[10] F. Hoffmann, C. Ickling, R. Klein, and K. Kriegel. The polygon exploration problem I: A competitive strategy. Technical report, Technical Report 245, Department of Computer Science, FernUniversität Hagen, Germany, 1998.

[11] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. and Autom.*, 12:566–580, 1996.

[12] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[13] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, NY, 1987.