The Eighth International
Symposium of Robotics Research
Hayama, Japan, October 3-7, 1997

Motion Planning with Visibility Constraints: Building Autonomous Observers

H.H. González-Baños L. Guibas J.C. Latombe S.M. LaValle
D. Lin R. Motwani C. Tomasi
Department of Computer Science
Stanford University, Stanford, CA 94305, USA
emails:{hhg,guibas,latombe,lavalle,dlin,motwani,tomasi}@cs.stanford.edu

Abstract

Autonomous Observers are mobile robots that cooperatively perform vision tasks. Their design raises new issues in motion planning, where visibility constraints and motion obstructions must be simultaneously taken into account. This paper presents the concept of an Autonomous Observer and its applications. It discusses three problems in motion planning with visibility constraints: model building, target finding, and target tracking.

1 Introduction

We are interested in mobile robots which autonomously perform vision tasks such as building 3-D models of unknown environments and finding/tracking unpredictable targets in cluttered environments. We call such robots *Autonomous Observers* (AOs for short). Multiple AOs may team up to accomplish tasks more quickly or to achieve goals that no AO could attain alone. E.g., it may not be possible for a single AO to reliably find a fast-moving target in a cluttered environment.

In the military domain, AOs may help assess the situation in a building by detecting potentially hostile targets and track their motions. In operating rooms, surgeons often operate by watching graphic displays of key tissues; AOs could automatically maintain visibility of the tissues in spite of obstructions caused by people and complex mechanical instruments. In robotics, researchers at one institution may want to conduct an experiment using hardware at another institution; AOs could be used to gather and transmit crucial real-time information over the Internet, allowing the remote researchers to effectively monitor their experiment. Other applications include remote monitoring of manufacturing operations in an assembly plant, search/rescue in a potentially hostile environment, and supervision of automated construction efforts in space.

AOs must execute motion strategies in which *visibility and motion obstructions are simultaneously taken into account.* In other words, they must not only avoid colliding with obstacles, an already well studied problem; they must also move so as to satisfy some visibility constraints. E.g., in the model building task, the AOs must eventually see the entire environment; in target finding, one AO must eventually see the target; and in target tracking, at least one AO must see the target at any one time. Concurrently, it is often desirable to minimize the number of AOs used. Note the relation with art-gallery problems (O'Rourke 1997), where the goal is to compute the locations of a minimal number of fixed guards that can collectively see all points in a given environment. In our case, AO mobility makes it possible to significantly reduce this number. Target tracking has an obvious connection to visual tracking of a moving object in an image sequence (Hutchinson et al. 1996). But, while the goal of the latter problem is to track the object *as long as it is visible in the images*, AOs must move to avoid potential visual obstruction by obstacles and keep the target in their field of view. Planning AO motions also relates to sensor placement (Briggs and Donald 1997) and active sensing (Maver and Bajcsy 1993).

In this paper we present our ongoing research on three specific planning problems related to the design of a team of AOs: i) *model building*, ii) *target finding*, and iii) *target tracking*. This sequence of problems corresponds to the following scenario: AOs are dropped into an unknown environment, of which they first have to build a model; then they have to find a target hiding among view-obstructing obstacles; finally, they must track this target's motions. However, the results obtained for each problem can be used independently. For lack of space, we only outline the main features of our approach for each problem and we show some experimental results. To give a better sense of the sort of algorithmic issues involved, we present one problem – target finding – in more detail.

2 Model Building

A basic task for an AO is to build a model of an environment using vision sensing. We wish this model to be usable for a variety of purposes, including future navigation (e.g., for target finding and tracking) and generation of realistic graphical renderings for virtual walkthrough operations. The model constructed by our AOs combines 2-D and 3-D geometry with texture maps.

A classical problem in automatic model building is known as the *next-best-view problem* (Banta et al. 1995; Maver and Bajcsy 1993; Pito 1995): Where to place the sensor next to maximize the amount of information that will be added to the partial model built so far? But existing techniques do not ideally suit AOs. One reason is inherent to the next-best-view problem itself: it is a *local* planning problem (Kakusho et al. 1995), so that a sequence of next-best views to build a complete model may yield too many sensing operations. In our case, each sensing operation is rather expensive: it requires acquiring 3-D and texture data, and merging this data with the current model. Hence, we wish to minimize the total number of operations. Moreover, a limitation of most next-best-view techniques is that they assume precise localization of the sensor. With our AOs, localization uncertainty must be taken into account. Finally, due to physical obstructions, a next-best-view technique may not suggest viewing positions that are accessible to the AOs.

These remarks led us to address model building in a different way. Assume for a moment that we are given a 2-D map describing the geometry of a horizontal cross-section of the environment at approximately the height of an AO's camera. An art-gallery algorithm computes the positions that the AOs must visit to eventually see the entire 2-D environment. We refine this approach as follows:
(1) The fact that the entire 2-D environment is visible from a set of positions does not in general entail that the entire 3-D environment is also visible. But this is almost true for many indoor environments. Some "holes" remain in the model built, but these are usually small and they can be filled by adding a few sensing operations at locations computed using a next-best-view technique.[1]

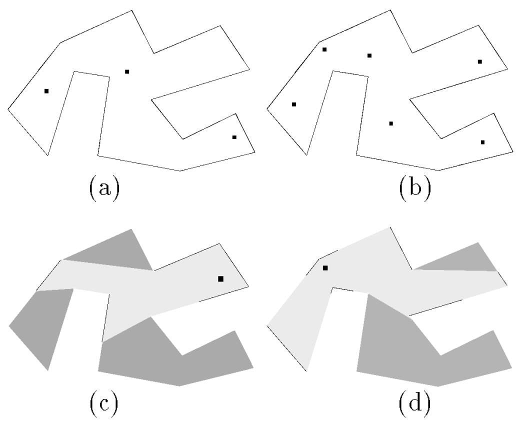


Figure 1: Computed AO positions in a 2-D map: (a) with no constraints; (b) with a minimum incidence of 60 degrees. The portions of walls seen from two AO positions are shown in solid lines in (c)-(d).

(2) Art-gallery algorithms use the simple "line-of-sight" visibility model: one point sees another if the line segment between them does not cross any obstacle. However, imperfections in vision sensors require that we use a more realistic definition taking distance and incidence into account. This yields new variants of the art-gallery problems.
(3) Art-gallery algorithms strive to minimize the number of positions to be visited by AOs. But 3-D sensing at these positions can yield partial models that have very small overlap between them. Uncertainty in AO localization requires that 3-D data from two partial models be aligned by partial shape matching before they are merged. Minimal overlap between the two models is needed.
(4) The 2-D model must be built in the first place. We do this by letting AOs navigate in the environment, each using a simple laser range sensor projecting a horizontal plane of light to obtain the environment's 2-D contour. Since this form of sensing is fast, the number of sensing operations is not critical, and a next-best-view technique is suitable to select on-line the successive viewing positions.

We are currently implementing this approach and experimenting with it. The planning component of our software computes the AO positions from a given polygonal map. Since the classical art-gallery problem is NP-complete, we have opted for a randomized algorithm that works as follows: First, it guesses many AO positions at random. Each position determines portions of the boundary of the 2-D map that an AO can see under the given distance and incidence constraints. The

[1] Since a wheeled AO is not a free-flying device, some holes cannot be eliminated. This can be dealt with by limiting virtual walkthroughs to viewing positions that could be achieved by an AO.

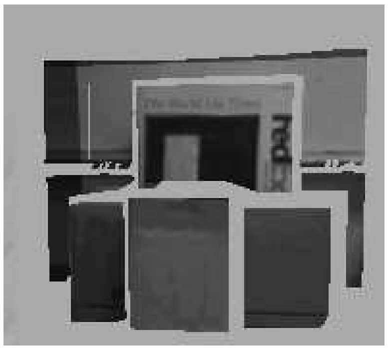


Figure 2: 3-D/texture model.

map's boundary is then partitioned into segments, each visible from the same subset of AO positions and labelled by this subset. If a segment has an empty label, more AO positions are picked. Finally, a greedy set-cover algorithm prunes the AO positions and selects a subset of them needed to see the entire boundary.[2] Fig. 1 shows an example of AO positions computed by our software under incidence constraints only.

In addition to this planning component, our software also includes the construction of a partial 3-D/texture model at a given AO's position, and the fusion of multiple partial models. Fig. 2 shows the graphic rendering of a model constructed from two different positions. The scene is a pile of boxes with a book on the foreground and a whiteboard as a background.

3 Target Finding

Suppose that a model for the environment F is available. The target-finding problem is to plan a motion strategy for the AOs to eventually see a target. This strategy must be such that, as the AOs move, their visibility region (i.e., the region that they collectively see) deforms and sweeps F so that the target has eventually no remaining place where to hide. Non-geometric pursuit-evasion problems have been studied in graphs (e.g., (Parsons 1976)). A problem similar to ours is analyzed in (Crass et al. 1995).

In the following, we assume that the target is unpredictable, has unknown initial position, and can move arbitrarily fast. (Hence, time is irrelevant.) F is an arbitrary 2-D polygon with n edges and h holes, where each of the AOs and target is modelled as a point. The visibility model is the line-of-sight model N denotes the minimal number of AOs needed for the existence of a guaranteed strategy.

[2] Set cover is NP-hard, but can be approximated within a log factor by a greedy algorithm.

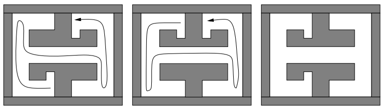


Figure 3: Geometry and number of AOs.

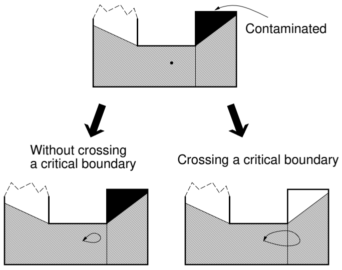


Figure 4: Critical event in information space.

Number of AOs. Clearly, $N > 1$ if $h > 0$, since the target can always hide behind a hole to avoid being seen by a single AO. However, even when $h = 0$, small geometric differences may affect N; e.g., in Fig. 3 the rightmost environment requires two AOs, while the other two require a single AO. We have established the following worst-case bounds on N (Guibas et al. 1997):

- For simply-connected environments ($h = 0$), $N = O(\lg n)$; there exist environments such that $N = \Omega(\lg n)$.
- For multiply-connected environments ($h > 0$), $N = O(\sqrt{h} + \lg n)$; there exist environments such that $N = \Omega(\sqrt{h} + \lg n)$.

Note that the art-gallery problem in an n-sided polygon with h holes requires $\lfloor (n+h)/3 \rfloor$ static guards (O'Rourke 1997).

In (Guibas et al. 1997) we also show that computing N for a given environment is NP-hard.

Single-AO Planner. The NP-hardness of computing N led us to investigate and develop a complete planner for the case of a single AO.

Let $V(q)$ be the AO's visibility region at position q in F. Each edge of $V(q)$ borders either an obstacle or free space. We call each edge bordering free space a *gap edge* and we associate a binary label with it: if the portion of free space that borders the gap edge is *contaminated* (i.e., may contain the target), then the label is 1; otherwise, it is 0. Let $B(q)$ denote the circular sequence of the labels of the gap edges in $V(q)$. We call $(q, B(q))$ the AO's *information state.*

Let the AO move along a closed-loop path starting and ending at q. The information state at q may change only if gap edges appear or disappear during the AO's motion. To illustrate, consider Fig. 4 where the AO is approaching the end of a corridor. If the closed-loop path on the left is

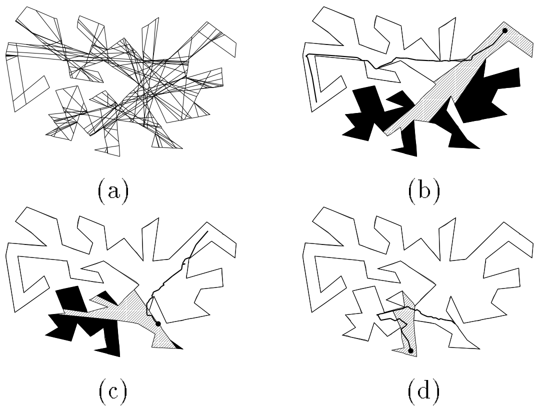


Figure 5: Target-finding strategy (example 1).

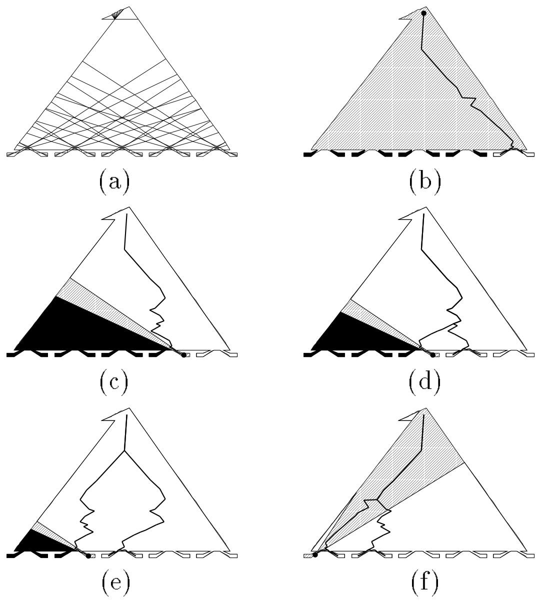


Figure 6: Target-finding strategy (example 2).

executed, the end of the corridor remains contaminated. If the path on the right is executed, a gap edge disappears and reappears, but with a different label (0 instead of 1).

We say that a subset S of F is *conservative* if no motion of the AO within S can cause a gap edge to appear or disappear. The critical places at which edge visibility changes form an arrangement of lines that decompose F into conservative cells. Such a decomposition has already been used in robot localization (Guibas et al. 1995; Talluri and Aggarwal 1996); it generates $O(n^3)$ cells for a simple polygon. To obtain conservative cells only, slightly fewer lines are needed than in a pure edge-visibility decomposition (Guibas et al. 1997).

A directed *information state graph* G is built and searched using this cell decomposition. For each cell κ, a set of vertices are included in G for each possible labeling of the gap edges. An arc connects two vertices v_i and v_j of G if the two corresponding cells κ_i and κ_j are adjacent and if the information state of v_j is obtained from the information state in v_i when the AO crosses the boundary between κ_i and κ_j. E.g., in the case shown in the lower right of Fig. 4, assume that the gap edge on the left is initially labelled by 0 and the gap edge on the right is labelled by 1. Let the first bit denote the leftmost gap edge label. The corresponding arcs of the information state graph are $(\kappa_i, 01)$ to $(\kappa_j, 0)$, and $(\kappa_j, 0)$ to $(\kappa_i, 00)$. In other situations, if two gap edges are merged into one, the new gap edge receives the label 1 if any of the original gap edges is labelled by 1. The search terminates when an information state is reached in which all labels are 0. Our planner uses the Dijkstra's search algorithm with an edge cost that is the shortest distance between the centers of the two cells.

Experimental results. The planner is written in C++ and runs on an SGI Indigo2 Workstation with a 200 Mhz MIPS R4400 processor. Fig. 5 shows a path computed by this planner. The edge-visibility decomposition displayed in (a) contains 888 cells yielding an information state graph with 103,049 vertices. The path is the concatenation of the three bold lines in frames (b)–(d). The gray region is the visibility polygon at the AO's position attained in each frame (thick point); black regions are contaminated areas, while white regions are cleared areas (other than the visibility polygon). The total computation time is 171.63s. There are 130 conservative cells and the complete information state graph contains 1727 vertices. Fig. 6 shows another example which took 10.73s to compute. The environment yields 246 conservative cells and an information state graph with 18,830 vertices. Note that the AO must clear the "beak" at the top-left multiple times.

The planner is efficient in practice, but we have not established its precise complexity. Examples can be constructed that yield an exponential number of information states; but, whether all these states may have to be considered to generate a path remains an open question. Fig. 6 shows, however, that there exist environments in which the same region is recontaminated $\Omega(n)$ times.

Multi-AO Planner. In theory, the techniques used in the single-AO planner also apply when $N > 1$, but they are likely to result in a very inefficient planner. Instead, we have developed a greedy multi-AO planner. This planner first plans for one AO, using the single-AO algorithm. If it

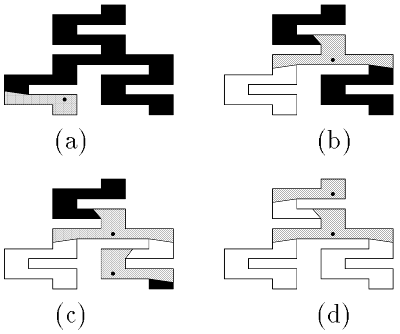


Figure 7: Target-finding strategy with two AOs.

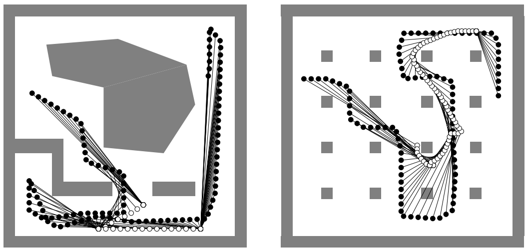
succeeds, the problem is solved. Otherwise, the planner clears as many cells as possible. The visibility polygon of the AO at its final position partitions F into components. The planner recursively treats each contaminated component as a new environment. Let N' be the number of AOs needed to clear the component that requires the most AOs; $N = N' + 1$, since a subset of the N' AOs can also be used for clearing each of the other components in sequence.

This planner always returns a motion strategy. If this strategy uses one or two AOs, it is optimal; otherwise, a plan with fewer AOs may perhaps exist. Fig. 7 shows an example generated by the planner. In (a)–(b), the first AO clears as many cells as possible, leaving two contaminated components that are cleared by a second AO in (c)–(d).

Current Work. We are studying two variations of the target-finding problem. In one we incorporate a more realistic visibility model in which the field of view of each AO is a cone with bounded depth. In the other we explore 3-D extensions that yield efficient algorithms. Other important variants include the cases where the target has bounded velocity and AO localization is imperfect; but such variants are more difficult to handle. For instance, when the target's velocity is bounded, time plays a critical role in AO strategies.

4 Target Tracking

Once a target has been found, the next step is to maintain visibility by appropriately moving the AOs, again taking visibility and motion obstructions into account. Unlike in target finding, time is now critical. The faster the planner and the more efficient the motion strategies, the better. This led us to develop two planning algorithms, depending on target predictability:
(1) For fully predictable targets, we have developed an off-line planner that computes an optimal solution for a given criterion (LaValle et al. 1997).



(a) (b)

Figure 8: Optimal target tracking strategies.

Fig. 8 shows two examples computed by this planner. The target is displayed as a black disc and the AO as a white disc. In (a), the tracking trajectory minimizes the total distance traveled, while in (b) it minimizes the time during which the AO does not see the target under the additional constraint that the AO's speed is only half that of the target.
(2) For partially predictable targets (e.g., we may only know their maximum speed), we have designed an on-line planner. In one variant, this planner maximizes the probability that the target will remain in view at the next time step. In another variant, it maximizes the minimum time in which the target could escape the visibility region.

Our general approach for a single AO is the following. We represent each of the AO and target by a point, and we model its motions using discrete-time transition equations. Let each time step be of length δ. We denote the position of the AO (resp. the target) at time $k\delta$ by q_k (resp. t_k). The transition equation for the AO is $q_{k+1} = f(q_k, \phi_k)$, where ϕ_k is an action chosen from some given action space Φ. Constraints such as bounded velocity can be encoded in f. Similarly. the equation for the target is $t_{k+1} = g(t_k, \theta_k)$, where θ_k is an action taken from some space Θ. When the target only partially pedictable, the AO knows Θ (and, possibly, a probabilistic distribution over Θ), but it does not know in advance the specific actions θ_k performed by the target.

At every time step the on-line planner computes a K-step motion strategy that optimizes a certain criterion over the next K time steps and the AO performs the first step of this strategy. In practice we take $K = 1$, as the cost of the computation increases dramatically with K. E.g., in the second variant of the on-line planner, the action ϕ_k is computed at each step in order to maximize the distance between t_k (as measured by the AO) and the boundary of the visibility region of the AO at its future position q_{k+1}. This choice, which corresponds to trying to minimize the time to escape, yields AO's motions that tolerate small errors in

Figure 9: Experimental setup for target tracking.

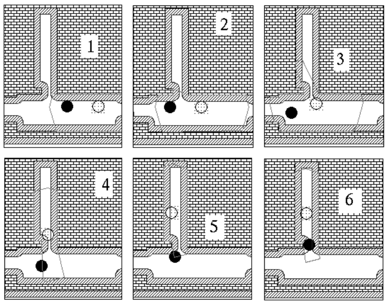


Figure 10: Target-tracking snapshots.

measuring t_k. It also easily extends to multiple AOs; we then maximize the distance between t_k and the boundary of the union of the visibility regions of the AOs at their future positions.

We have implemented these planners and performed numerous successful experiments both in simulation and with a real AO. Fig. 9 shows our experimental setup (the robot at the forefront is the AO; the other robot, with a distinctive "hat," is the target). Fig. 10 shows a run as it appears on the user's display (the gray disc is the target and the black disc the AO).

5 Conclusion

Motion planning with visibility constraints has received little attention so far, despite the fact that it has many potential applications. In this paper we presented three problems encountered in our AO project: model building, target finding, and target tracking. Our research combines theoretical investigation of planning problems with partly-idealized visibility models to produce guaranteed algorithms and pragmatic tradeoffs between algorithmic rigor and model realism to construct effective systems for realistic experimental setups.

Acknowledgments: This work is supported by ARO MURI grant DAAH04-96-1-007, ONR grant N00014-94-1-0721, and NSF grant IRI-9506064. We thank Prof. R. Bajcsy (U. of Pennsylvania) and Prof. J.L. Gordillo (ITESM, Monterrey, Mexico) and their students for having experimented with our AO over the Internet and for their useful suggestions.

References

Banta, J. E., Y. Zhien, X. Z. Wang, G. Zhang, M. T. Smith, and M. Abidi (1995). A best-next-view algorithm for three-dimensional scene reconstruction using range images. In *Proc. SPIE*, Vol. 2588, 418–429.

Briggs, A. J. and B. R. Donald (1997). Robust geometric algorithms for sensor planning. In *Algorithms for Robotic Motion and Manipulation (WAFR'96)*, 197–212. J.P Laumond and M. Overmars (eds.), A K Peters, Wellesley, MA.

Crass, D., I. Suzuki, and M. Yamashita (1995). Searching for a mobile intruder in a corridor – the open edge variant of the polygon search problem. *Int. J. of Comp. Geom. and Appl. 5*(4), 397–412.

Guibas, L. J., J. C. Latombe, S. M. LaValle, D. Lin, and R. Motwani (1997). Visibility-based pursuit-evasion in a polygonal environment. In *Proc. 5th Workshop on Algorithms and Data Structures (WADS'97)*, Springer Verlag, 17–30.

Guibas, L. J., R. Motwani, and P. Raghavan (1995). The robot localization problem. In *Algorithmics Foundations of Robotics (WAFR'94)*, 269–282. K. Goldberg et al. (eds.), A K Peters, Wellesley, MA.

Hutchinson, S., G. D. Hager, and P. Corke (1996). A tutorial on visual servo control. *IEEE Tr. on Robotics and Automation 12*(5), 313–326.

Kakusho, K., T. Kitahashi, K. Kondo, and J. Latombe (1995). Continuous purposive sensing and motion for 2D map building. In *Proc. IEEE Int. Conf. of Syst., Man and Cyb.*, 1472–1477.

LaValle, S. M., H. H. Gonzalez-Bãnos, C. Becker, and J. C. Latombe (1997). Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE Int. Conf. on Rob. and Automation.*

Maver, J. and R. Bajcsy (1993). Occlusions as a guide for planning the next view. *IEEE Tr. on Pattern Analysis and Machine Intelligence 15*(5), 417–433.

O'Rourke, J. (1997). Visibility. In *Handbook of Discrete and Computational Geometry*, 467–479. J.E. Goodman and J. O'Rourke (eds.), CRC Press, Boca Raton, FL.

Parsons, T. D. (1976). Pursuit-evasion in a graph. In *Theory and Application of Graphs*, 426–441. Y. Alavi and D. Lick (eds.), Lecture Notes in Mathematics 642, Springer Verlag, Berlin.

Pito, R. (1995). *A solution to the next best view problem for automated CAD model acquisition of free-form objects using range cameras.* Technical Report 95-23, GRASP Lab., U. of Pennsylvania.

Talluri, R. and J. K. Aggarwal (1996). Mobile robot self-localization using model-image feature correspondence. *IEEE Tr. on Robotics and Automation 12*(1), 63–77.