VSP

Advanced Robotics 23 (2009) 1533–1560

ADVANCED ROBOTICS
www.brill.nl/ar

Full paper

An Efficient Motion Strategy to Compute Expected-Time Locally Optimal Continuous Search Paths in Known Environments

Alejandro Sarmiento [a], **Rafael Murrieta-Cid** [b,*] **and Seth Hutchinson** [a]

[a] Beckman Institute, University of Illinois, Urbana, IL 61801, USA

[b] Centro de Investigación en Matemáticas, CIMAT, AP 402, Guanajuato, Gto 36000, México

Received 9 June 2008; accepted 19 October 2008

Abstract

In this paper we address the problem of finding time-optimal search paths in known environments. In particular, we address the problem of searching a known environment for an object whose unknown location is characterized by a known probability density function (PDF). With this formulation, the time required to find the object is a random variable induced by the choice of search path together with the PDF for the object's location. The optimization problem we consider is that of finding the path that minimizes the expected value of the time required to find the object. As the complexity of the problem precludes finding an exact optimal solution, we propose a two-level, heuristic approach to finding the optimal search path. At the top level, we use a decomposition of the workspace based on critical curves to impose a qualitative structure on the solution trajectory. At the lower level, individual segments of this trajectory are refined using local numerical optimization methods. We have implemented the algorithm and present simulation results for the particular case when the object's location is specified by the uniform PDF.

Keywords

Search, pursuit–evasion, path planning

1. Introduction

In this paper, we address the problem of minimizing the expected value of the time required to find an object in a known two-dimensional (2-D) environment modeled by polygons. This corresponds to finding a search trajectory that will minimize the average time to find an object if the search is performed many times. We assume that the search is performed by a mobile robot that is capable of recognizing the object

* To whom correspondence should be addressed. E-mail: murrieta@cimat.mx

 DOI:10.1163/016918609X12496339799170

and that all prior knowledge of the object's position is encoded in a probability density function (PDF) defined on the environment. We believe that the potential applications are many, from finding a specific piece of art in a museum to search and detection of injured people inside a building.

In this paper, we deal mainly with the theoretical development of the planning algorithms to deal with this problem. The implementation of our algorithms in a real robot would need significantly more development; in particular, computer vision algorithms able to robustly detect the search object.

The work we report here represents a combination and an extension of our previous work. In Ref. [1], we presented an approach to the problem of searching an object by sensing at discrete locations in a 2-D environment. In that case, we used a visibility-based decomposition of the polygon to convert the problem into a combinatoric one. In Ref. [2] we extended our work to 3-D environments. In Ref. [2], we again assumed that the robot only senses at discrete locations.

However, searching for an object with a robot that senses the environment continuously has advantages over sensing only at discrete locations. A robot that senses at discrete locations can only find the object when it reaches one of the locations, even if the object had entered its field of view long before. A robot that senses the environment continuously can find the object as soon as the object is within its field of view. In addition, in a continuous sensing scheme the robot may perform temporal integration of data, which allows robustness in the sensing data processing needed to detect the object. In our formulation, we assume that the robot is equipped with an omnidirectional sensor. Current technology provides these type of sensors, e.g., omnidirectional cameras [3, 4]. These are some of our motivations to propose a motion strategy where the robot senses the environment continuously as it moves.

In Ref. [5], we presented methods to find locally optimal continuous trajectories between adjacent regions in a polygonal decomposition of the workspace. In the present paper, we present a two-level planner that gives a full, heuristic solution to the problem of minimizing the expected time to find an object with a robot that senses continuously the environment. Our main contributions are:

(i) We claim that our work presents a new paradigm for search tasks. This corresponds to minimizing the expected value of the time to find an object. It can be specially useful in applications where the time assigned to the task is limited or not completely known.

(ii) We provide a two-level algorithm that determines an efficient ordering of visiting regions and then generates locally optimal subpaths to construct a complete efficient trajectory.

(iii) We provide an efficient procedure based on the calculus of variations to compute locally optimal paths.

1.1. Related Work

The present work is placed in the area of path planning for mobile robots. In this area, a primary task called navigation is to find a collision-free path for the robot to move from an initial to a final configuration. Several works have addressed this problem [6, 7]. Some of them use road maps [6] to guide the robot from an initial to a final configuration, others use potential functions to accomplish the task [7, 8]. Other works attempt to optimize a criterion such as distance and/or robot turns [9] or clearance from the robot path to the obstacles [10].

For the navigation problem, there are characteristics that can be added to make them more general, for example, kinematic constraints on movement [11], sensing and control uncertainty [12], limited sensors [10], moving obstacles [13], etc.

Similar to the work presented in Ref. [9], in this work we also propose semi-optimal paths (in our case locally optimal), and as in Ref. [14] we carry out our optimization procedure in a reduced search space.

Nevertheless, we need to find collision-free paths to the move the robot as in Refs [6, 7, 9, 10, 12]. Our main interest is to address the problem of finding a static object in a known environment. Our goal is to make the robot find an object as quickly as possible on average. This adds a new aspect to our planning problem.

Our search problem is related to art gallery problems, exploration, coverage and pursuit–evasion. The traditional art gallery problem is to find a minimal placement of guards such that their respective fields of view completely cover a polygon [15]. As we will see in Section 4, guard placements could be used in a partial solution to our search problem. A variation of the art gallery problem in which the task is to find the minimal length path that covers the polygon is known as the shortest watchman tour problem [16]. This is not exactly our problem since, as we will see in Section 2.1, a trajectory that minimizes the distance traveled may not minimize the expected value of the time to find an object along it.

In coverage problems (e.g., Refs [17, 18]), the goal is usually to sweep a known environment with the robot or with the viewing region of a sensor. In this problem, it is often desirable to minimize sensing overlap so as not to cover the same region more than once. Our problem is related to the coverage problem in the sense that any complete strategy to find an object must sense the whole environment.

Exploration problems usually involve the generation of a motion strategy to efficiently move a robot to sense and discover its environment and construct a representation (model) of the environment [19–22]. In exploration problems for the robot to move to an unexplored area, a local navigation problem must be solved. For instance, in Ref. [10] the authors propose algorithms for local path planning and map building based on the generalized Voronoi graph (GVG). The authors also deal with robot kinematic constraints by replacing the GVG's arcs with smooth local paths that a car-like robot is able to travel.

In exploration problems, unlike ours, the environment is not known *a priori* and the objective is to construct a complete representation rather than to find a specific target.

Finally, our research is somewhat related to the pursuit–evasion problem, where one robot or a team of robots — the pursuers — are interested in finding other mobile robots — the evaders [23–30]. In our problem the searched object is static. In some sense, this simplifies the problem (since once an area has been seen, there is no worry that the object could move into that area), which allows us to consider the more difficult optimization problem of minimizing the expected value of the time required to find the object.

In Ref. [31] a dynamic data structure (called the gap navigation tree) is proposed. This data structure corresponds to a minimal representation of the environment and allows the search of static targets. In the present paper, the problem of finding a static object is also addressed, but unlike the work presented in Ref. [31], instead of minimizing the distance traveled to find the object, here our goal is for the robot to find the object as quickly as possible on average. Our new formulation of optimality can be very useful in applications where the time assigned to the task is limited or not completely known.

The remainder of the paper is organized as follows. In Section 2, we give the mathematical definition of our problem. In Section 3, we give an overview of our two-level solution approach. We then describe the two levels of planning; in Section 4, we describe the top level, in which constraints on the qualitative structure on the solution path are derived, and in Section 5, we describe the lower level, in which locally optimal continuous trajectories are derived given these qualitative constraints. Finally, in Section 6, we present simulation results.

2. Problem Definition

In this section we give the mathematical definition of the problem of minimizing the expected value of the time required to find the object. We assume the robot is equipped with an omnidirectional sensor. We also assume that the environment is 2-D polygonal and known (i.e., the robot has a complete map), and that the robot will recognize the object when it is visible. In our formulation, we assume that the environment is know, but that we do not have any information about the search object location. Since there is no reason to believe that one object location is more likely than another, we assign equal values to every location. This is equivalent to defining a uniform PDF modeling the object location. In fact this assumption is already known in the literature and called the principle of insufficient reason [32]. We believe this reasoning is very general given that we do not need to assume a relation between a particular type of object and its possible location, which will reduce the scope of the applications.

While we believe that the formulation we develop in this section holds for arbitrary PDFs, in the sequel we will develop specific solutions for the case of a uniform PDF, since it is clear that additional work would be required to do so.

If the robot follows a path S starting from initial position (x_0, y_0) at time $t = 0$ (see Fig. 1), we define the random variable T to be the time at which the robot first

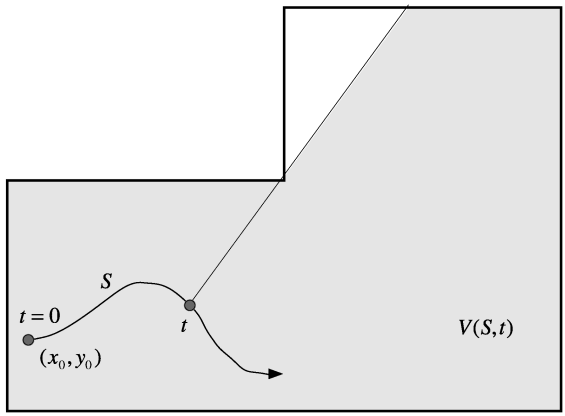


Figure 1. Simple example of a point robot following a route S.

sees the target object. The probability that the robot will see the object before time t is given by $P(T \leqslant t) = F_T(t)$, the cumulative distribution function (CDF) of the random variable T:

$$F_T(t) = \int_0^t f_T \, \mathrm{d}t = P(T \leqslant t).$$

Of course the probability of having seen the object prior to time t depends on the route S followed by the robot. We, therefore, define the CDF along any given path as:

$$F_T(t|S) = \int_{V(S,t)} f_{XY}(x, y) \, \mathrm{d}x \, \mathrm{d}y,$$

in which $f_{XY}(x, y)$ is the PDF for the object's location and $V(S, t)$ is the subset of the environment that has been seen by the robot as it moves along S until time t. We say that at time t the path S has covered the region $V(S, t)$ and, in general, we will use the term 'cover' to denote that the robot has sensed, not necessarily physically covered, a certain region.

In the particular case of a uniform f_{XY}, the probability of seeing the object before time t is proportional to the area of the environment that has already been seen:

$$\frac{V(S, t)}{\text{Total Area}} = P(T \leqslant t) = F_T(t).$$

From the CDF F_T we can obtain the PDF f_T and calculate the expected value of the time to find the object T following route S:

$$E[T|S] = \int_0^\infty t \cdot f_{T|S}(t|S) \, \mathrm{d}t.$$

We are interested in finding the trajectory S that minimizes $E[T|S]$, in other words, obtaining the path that, on average, makes the robot find the object as quickly as possible. This amounts to the optimization:

$$S^* = \arg\inf_S \{E[T|S]\} = \inf_S \left\{ \int_0^\infty t \cdot f_{T|S}(t|S)\, \mathrm{d}t \right\}. \quad (1)$$

This is an infinite dimensional optimization problem. We have shown that even a discrete version of this problem is NP-hard [1] and, thus, it is not practical to generate an optimal route. We will, therefore, present algorithms that make a trade-off between optimality and tractability.

2.1. *Expected Value* Versus *Worst Case*

It is useful to note the difference between minimizing the expected value of the time to find an object and minimizing the time it would take in the worst case. To minimize the worst case time, the robot must find the shortest path that completely covers the environment (the shortest watchman tour problem [16]). This usually means that no portions of the environment are given any priority over others and the rate at which new portions of the environment are seen is not important. On the other hand, to minimize the expected value of the time, the robot should gain probability mass of seeing the object as quickly as possible. For a uniform f_{XY}, this requires sensing large portions of the environment as soon as possible.

For a given environment, the route that minimizes the distance traveled typically does not minimize the expected value of the time to find an object along it. This is illustrated in the example shown in Fig. 2. In this example, there are two rooms, and these can be observed from viewpoints L_1 and L_2, respectively. The probability of finding an object in a room is proportional to the size of the room. Assume that the robot starts in the corridor at location L_0 and moves with unit speed.

There are only two routes the robot might take to solve this problem: Go to the smaller room first, $L_0 \to L_1 \to L_2$, or go to the larger room first, $L_0 \to L_2 \to L_1$. In the former case, the robot reaches L_1 at $t = 1$ and L_2 at $t = 7$, and the expected value of the time to find the object is:

$$E[T|(L_0, L_1, L_2)] = (0.1)(1) + (0.9)(7) = 6.4.$$

The robot always completes its search by $t = 7$. In the latter case, the robot reaches L_2 at $t = 5$ and L_1 at $t = 11$, and the expected time to find the object is:

$$E[T|(L_0, L_2, L_1)] = (0.9)(5) + (0.1)(11) = 5.6.$$

In the worst case, the robot completes its search at $t = 11$. Thus, as can be seen from this simple example, the trajectory that is optimal in the distance traveled does not necessarily minimize the expected value of the time to find the object.

3. Solution Overview

In general, it is not possible to solve (1) directly to obtain the globally optimal trajectory S^*. For this reason, we have adopted a two-level approach that constructs an approximation to S^*. At the top level, we use a decomposition of the workspace to impose a qualitative structure on our solution trajectory. At the lower level, individual segments of this trajectory are refined using local numerical optimization methods.

If the environment is convex then everything can be seen from a single point and the solution is trivial. If the environment contains a single corner, nonconvex vertex, as the one in Fig. 3a, any path that covers the polygon must reach one of the inflection rays (aspect graph lines) [33], either A_1 or A_2, associated to the nonconvex vertex. The nonconvex vertices are also called reflex vertices, those whose internal angle is bigger than π. Those vertices are key in our problem because they are the ones that break the environment convexity.

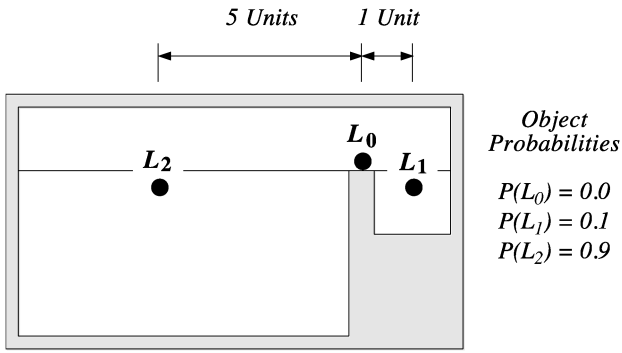


Figure 2. Example with a simple environment.

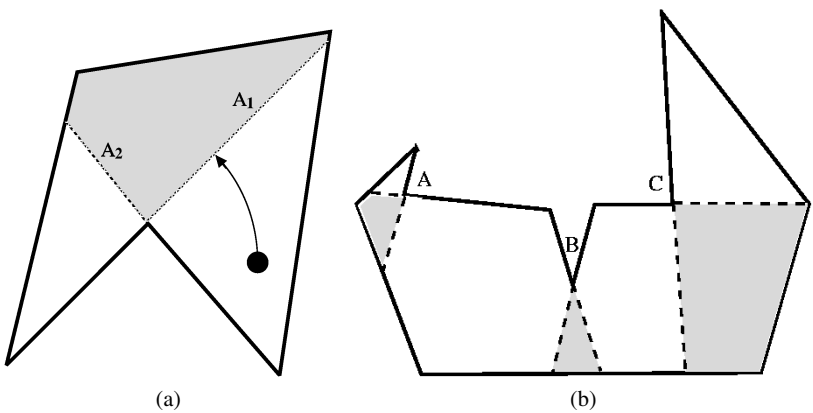


Figure 3. Optimal trajectories. (a) An environment with one nonconvex vertex. (b) Inflection rays (dotted segments) associated with nonconvex vertices A, B and C.

We call the area bounded by the inflection rays the corner guard regions [34]. In Fig. 3a the corner guard region is shown in gray. These regions have the characteristic that any point inside them can see 'both sides' of their associated nonconvex vertices. In Fig. 3b, the inflection rays are the dotted lines (incident to the vertices labeled A, B and C) that demarcate the corresponding corner guard regions shown in gray.

A sufficient and occasionally necessary condition for a searcher to view the entirety of a polygon is that the searcher visit each corner guard region. This observation is the motivation for our top-level strategy. Since a continuous path needs to visit the corner guard regions, it is important to decide in which order they are visited. The problem can be abstracted to finding a specific order of visiting nodes in a graph that minimizes the expected value of time to find an object.

We describe this strategy in Section 4. The basic idea is to select viewing locations L_i associated to each nonconvex vertex and to determine the optimal ordering of visiting the L_i locations.

Given the ordering on the locations L_i, the low-level strategy constructs locally optimal paths between regions. Each of these locally optimal path segments begins at the endpoint of the previous segment and terminates somewhere on the edge bounding the next region to be visited. The optimization is performed using the Runge–Kutta method to numerically solve the Euler–Lagrange equations found using the calculus of variations. This is described in Section 5. In Section 6, we present trajectories that are obtained when the two levels are combined.

4. Top-Level Trajectory

As mentioned before, our overall strategy is to partition the workspace with critical curves, calculate the locally optimal trajectory in each region and then concatenate the subpaths to construct the final trajectory.

Recall that to cover a polygon, it is sufficient that a trajectory visits at least one point inside each corner guard region (as defined in Section 3) associated to nonconvex vertices of the polygon.

To estimate the ordering to visit inflection rays (critical curves), we select discrete view locations L_i; each is associated to a nonconvex vertex. We place one L_i near each nonconvex vertex inside its corner guard region. We call these locations guards (from the art gallery problem [35]).

Thus, the high-level, combinatoric algorithm finds an ordering for the robot to visit the guards. Note that when the robot travels the continuous path to cover the environment, it does not visit the exact location of the guards, but the inflection rays associated to them. The guards' locations are only used to define an ordering to visit the corner guard regions associated to them.

Figure 4 shows a polygon with the two guards locations marked by an '$\times$', and labeled L_1 and L_3. The guards' visibility regions are shown with two gray colors and the visibility region intersection is shown in white.

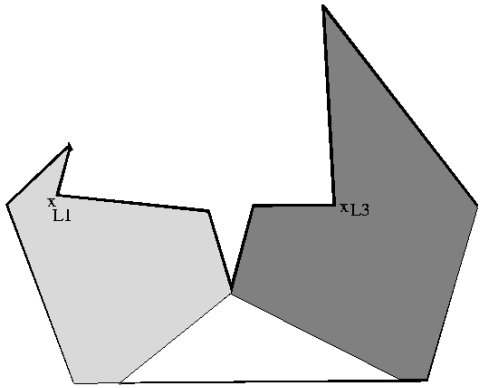


Figure 4. Guards (sensing locations) associated to nonconvex vertices and their visibility region $V(L_i)$.

4.1. Choosing an Ordering of Regions

Our algorithm to choose an ordering to visit regions builds on our previous research on expected-value search with mobile robots [1]. We use a utility function to drive a greedy algorithm in a reduced search space that is able to explore several steps ahead without incurring too high a computational cost. Below, we describe our efficient algorithm to define an order to visit sensing locations.

We define the visibility region for location L_j, denoted $V(L_j)$, as the set of points that have an unoccluded line of sight to L_j (the line segment connecting them does not intersect the exterior of P). Thus, if the object lies within $V(L_j)$, the robot will successfully recognize the object from L_j. If the set $\{L_i\}$ is chosen as described in Section 3, the associated visibility regions define a cover of P, i.e.:

$$\bigcup_j V(L_j) = P.$$

Thus, a sufficient condition to ensure that the object is found is that the robot visit each L_j. In this section, we give an algorithm for determining the optimal order in which the L_j should be visited.

For a given ordering of locations $L_{i_1}, L_{i_2}, \ldots$, we define T as the random variable that denotes the time required to find the object by sensing successively at these locations. We use the notation L_j to refer to a particular sensing location in the environment, while L_{i_k} refers to the kth location in the visitation sequence, i.e., the robot always starts at L_{i_1} and the kth location it visits is referred to as L_{i_k}. For a given sequence, the expected value of the time it takes to find the object is given by:

$$E[T|S] = \sum_j t_j P(T = t_j), \tag{2}$$

where:

$$P(T = t_j) = \frac{\text{Area}(V(L_{i_j}) \setminus \bigcup_{k<j} V(L_{i_k}))}{\text{Area}(P)}, \tag{3}$$

assuming that the object's location is specified by a uniform probability density on P. Here, t_j is the time it takes the robot to go from its initial position — through all locations along the route — until it reaches the jth visited location L_{i_j} and $P(T = t_j)$ is the probability of seeing the object for the first time from location L_{i_j}. Since the robot only senses at specific locations, we also denote this probability of seeing the object for the first time from location L_{i_j} as $P(L_{i_j})$.

In the remainder of this section we first describe a complete algorithm for determining the optimal ordering of viewing locations and show that the underlying problem is NP-hard. We then give a heuristic algorithm with near real-time performance, and present results that demonstrate its efficiency and effectiveness.

4.2. Computing the Optimal Sequence

Given the set $\{L_i\}$, determining the sequence that minimizes the expectation given in (2) can be solved using a combinatorial search on a graph with dynamic weights. The graph is constructed as follows:

(i) For each location L_j, create a node N_j in the graph.

(ii) For each pair of nodes N_j and N_k, add an edge with variable weight W_{jk}.

(iii) The weight W_{jk} is dynamic; it depends on the route followed by the robot before reaching N_j. These weights are calculated online.

The weight W_{jk} should correspond to the increase in expected time the robot incurs by going from L_j to L_k (i.e., the partial calculation of (2) along the current route). This is a function of the time at which it arrives at L_k, which in turn depends on the route followed by the robot up to that point. For example, consider the abstract environment in Fig. 5. In Fig. 5, the nodes represent locations and the arcs represent the time it takes to move from one location to another. In general, the robot may not be able to travel between two locations by following a straight line. In such cases, we use a reduced visibility graph [36] and Dijkstra's algorithm to follow the shortest path between them to compute these values. If the robot starts in location L_0 at time $t = 0$, it can take two paths to location L_2 and the increase in expected value of time W_{23} to go from L_2 to L_3 depends on which route the robot

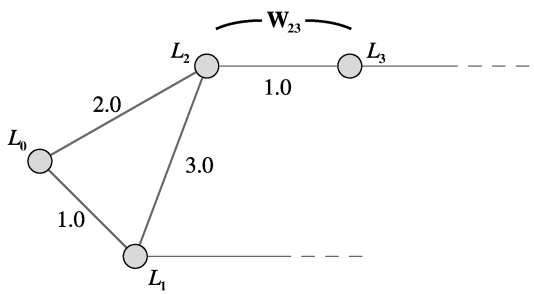


Figure 5. Example for the dynamic weight W_{23}.

follows. If the robot goes directly from L_0 to L_2, it reaches L_2 at time $t = 2.0$ and the increase is $W_{23} = 2.0 \cdot P(L_3)$. On the other hand, if the robot goes from L_0 to L_1 and then to L_2, it reaches L_2 at $t = 4.0$, so the the increase in expected value of time to reach L_3 from L_2 is $W_{23} = 4.0 \cdot P(L_3)$.

Given this graph, we must find the path of minimum cost that starts at the robot's initial location L_{i_1} and includes all other locations. This can be accomplished with a branch and bound graph search [37]. This search strategy maintains a list of nodes to be opened, ordered by their accumulated cost. The next node to be expanded (i.e., the one whose accumulated cost is currently minimal) is always at the head of the list. When a node is expanded, only those nodes that are adjacent and not already included in the current path are considered children. The added cost W_{jk} of expanding a child N_k from its parent N_j is given by:

$$W_{jk} = \text{Time}(N_k) \cdot P(L_k)$$

$$\text{Time}(N_k) = \text{Time}(N_j) + \frac{\text{Dist}(L_j, L_k)}{\text{Speed}}.$$

Then, the accumulated cost for the child is:

$$\text{Cost}(N_k) = \text{Cost}(N_j) + W_{jk},$$

with a cost of zero for the node corresponding to the initial position, $\text{Cost}(N_{i_1}) = 0$.

Initially, the branch and bound list contains only the starting location. Then, the head of the list is expanded and its children added to the ordered list until a solution is found — a path that contains all locations in $\{L_i\}$. When a solution is found, the currently best nodes still continue to be expanded until (i) a lower cost solution is found, in which case the better solution is saved and the process continues, or (ii) the lowest cost node is worse than the current solution, in which case we know that the current solution is optimal.

This algorithm finds the optimal solution — the one that minimizes the expected time to find the object. Unfortunately, its space and time complexities are not of polynomial order. Furthermore, the problem itself is intractable; more specifically, it is NP-hard, as we now show.

Proposition. Finding the sequence that minimizes the expectation given in (2) is an NP-hard problem.

Proof. We prove the proposition by a reduction from the minimum weight Hamiltonian path problem (MWHP), which is known to be NP-hard [38].

Consider a set $\{L_i\}$, for $i = 1, \ldots, n$, such that $V(L_i) \cap V(L_j) = \emptyset$, for all $i \neq j$, and for which $P(L_i) = 1/n$ (i.e., the probability of seeing the object is equally likely for all locations). This reduction is illustrated in Fig. 6. The environment on the left consists of four 'rooms' that are equally likely to contain the object. In the graph on the right, nodes correspond to these rooms and edges are weighted by the time required to travel between rooms.

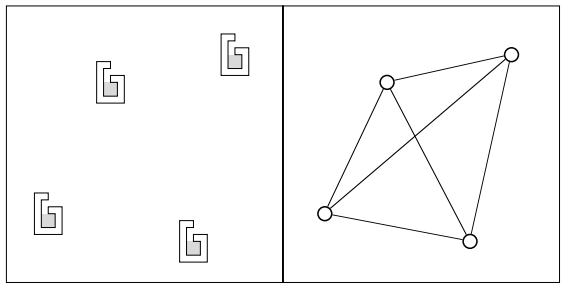


Figure 6. Polygon (one instance of our problem) corresponding to the MWHP problem.

In this special case, the ordering that minimizes the expected time to find the object will be exactly the same as the one that minimizes the distance traveled. This can be seen by letting $P(T = t_j) = 1/n$ in (2) and noting that the sum depends only on the accumulated time to traverse the sequence. Thus, this particular instance of our problem is identical to the MWHP problem (merely set edge weight for the edge from v_i to v_j to the time required to move from L_i to L_j). If a polynomial time algorithm were to exist for our problem, it would thus be possible to solve MWHP in polynomial time.

Given that our problem is intractable, we now turn our attention to a heuristic algorithm that finds an approximate solution.

4.3. Heuristic Algorithm

In this section we describe a heuristic algorithm that has proved to be both effective and efficient in practice. At the heart of the algorithm is a utility function that gives an estimate of the benefit gained by visiting a specific L_i. This utility function gives rise to the notion of dominating strategies, which can be used to limit the nodes that are explored by the search algorithm. Following the description of this algorithm, we present results that quantitatively demonstrate the algorithms performance against the exact algorithm described in Section 4.2.

At first, it might seem intuitive to assign to location L_i a utility value that is inversely proportional to the increase in the partial calculation of (2) along the current route. This approach performs poorly in practice, because the product in (2) causes locations with low probability to be given high utility value, thus causing the exploration strategy to begin by looking in the least likely locations. A more effective utility function balances the desire for a high probability of finding the object against the desire to minimize search time. Thus, we define the utility of moving from location L_j to location L_k as the ratio:

$$U(L_j, L_k) = \frac{P(L_k)}{\text{Time}(L_j, L_k)}. \tag{4}$$

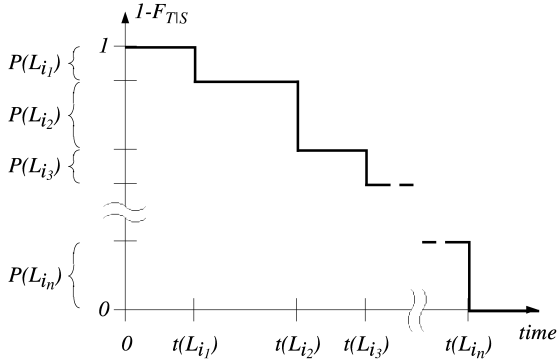


Figure 7. Function $1 - F_{T|S}$.

A robot using this function to determine its next destination will tend to prefer locations that are close and/or locations where the probability of seeing the object is high.

The utility function (4) is directly related to the expectation of (2). Consider the alternate definition of expectation for a non-negative random variable, such as time, from Ref. [39]:

$$E[T|S] = \int_0^\infty P(T > t|S)\,\mathrm{d}t = \int_0^\infty (1 - F_{T|S})\,\mathrm{d}t, \tag{5}$$

in which $F_{T|S}$ is a CDF that depends on the specific route followed. In our problem, every valid trajectory S defines a particular CDF of finding the object, $F_{T|S}$. Since we are dealing with a discrete problem, the distributions are piecewise constant, with the discontinuities being the times at which the robot reaches the distinct L_i along the route. By (5), the expected value of a random variable with distribution $F_{T|S}$ is the area under the curve $1 - F_{T|S}$, shown in Fig. 7 and it is this area that we wish to minimize.

The utility function in (4) can be used to define a one-step greedy algorithm. At each step, simply evaluate the utility function for all available locations and choose the one with the highest value. This algorithm has a running time of $O(n^2)$. However, in general, it is preferable to use a multistep look ahead. Unfortunately, this typically increases the complexity of the algorithm by a factor of $O(n)$ for each look ahead step. Therefore, we use the notion of dominating strategies to reduce the branching factor at each stage of the look ahead. In particular, if the current location is L_j, we say that location L_k strictly dominates location L_l if both of the following conditions are true:

$$P(L_k) > P(L_l)$$

$$\mathrm{Time}(L_j, L_k) < \mathrm{Time}(L_j, L_l),$$

i.e., if the time required to travel to L_k is smaller than that required to travel to L_l and the probability of seeing the object from location L_k is greater than the probability of seeing it from L_l.

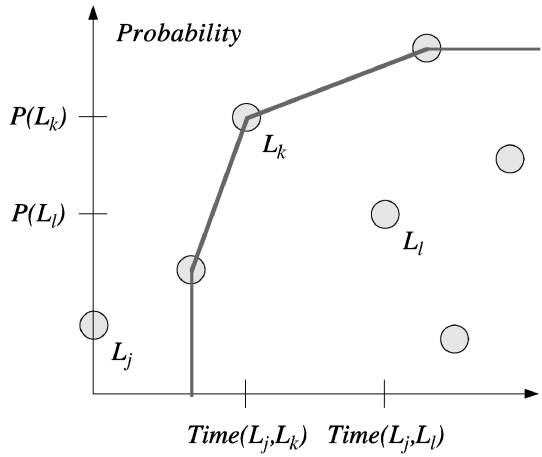


Figure 8. Location dominance.

If we plot unexplored locations on the 'time–probability' plane as shown in Fig. 8, it is easy to see that dominating locations will lie on the convex hull of the set of unexplored locations. The end points of this partial convex hull are not considered as candidates since they are not defined locations. By only considering this subset of the remaining locations at each step, we reduce the branching factor, making it possible to explore more steps ahead without incurring too high a computational cost. Of course, there is no guarantee that the optimal solution is indeed a member of this reduced search space or even that this will yield better results. However, we have found it to be a good heuristic in practice, as we will show below.

The full algorithm consists in iteratively exploring several steps ahead, choosing the most promising route up to that point and starting over from there. For n locations, if the branching factor (average number of children per node) is B, a tree of height $\log_B n$ can be explored in linear time. This creates a partial route of length $\log_B n$. Since a solution should be of length n, the process needs to be repeated $n/\log_B n$ times for the complete route. This is depicted in Fig. 9. In Fig. 9, the larger triangle represents the tree that would be generated if a complete exploration were made, whereas the small triangles represent the trees that are actually generated (explored) by the algorithm.

Thus, our final algorithm is as follows:

(i) For the last location along the current solution (initially just the robot starting location) explore the possible routes (create a tree breadth-first) until the number of nodes is of order $O(n)$.

(ii) For each node that needs to be expanded, compute the set of locations that are not strictly dominated by others and only choose those as children. This can be done with a convex hull algorithm with complexity $O(n \log n)$.

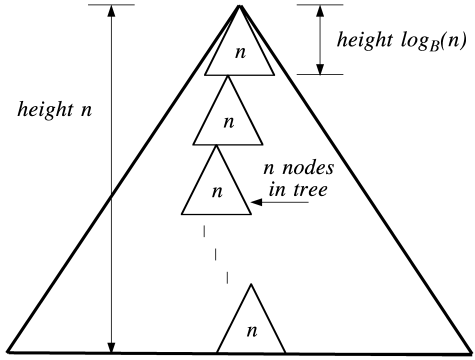


Figure 9. Exploration algorithm.

(iii) When the number of nodes in the exploration tree has reached order $O(n)$, choose the best leaf according to the heuristic in (4), discard the current tree and start over with the best node as root.

The complexity of the algorithm is proportional to exploring a tree of order $O(n)$, choosing the best children for each node in the tree with a convex hull algorithm in $O(n \log n)$ and repeating $n/\log n$ times to generate a complete route. This is:

$$O\left(n \cdot n \log n \cdot \frac{n}{\log n}\right) = O(n^3).$$

In the worst case, when the branching factor is not reduced at all, our algorithm only explores one step at a time and has a running time of:

$$O(n \cdot n \log n \cdot n) = O(n^3 \log n). \tag{6}$$

This analysis only considers the time complexity of the search algorithm itself. It does not include the time complexity of performing polygon clipping operations, which are required to compute the actual probabilities of seeing the object for the first time from location L_i. To date, we have implemented our algorithm only for the case of a uniform PDF of the object's location over the environment; consequently, the probability of seeing the object from any given location is proportional to the area of the visibility region from that location (point visibility polygon [40]). The probability of seeing the object for the first time from location L_{i_j} is proportional to the area visible from L_{i_j} minus the area already seen from locations L_{i_k} $\forall k < j$, as stated in (3). This requires polygon clipping operations to compute set unions and differences. Any clipping algorithm supporting two arbitrary polygons must have a complexity of at least $O(nm)$ where n and m are the number of vertices in each polygon [41]. The cost of performing these clipping operations must be added to the complexity in (6) to describe the total complexity of the algorithm when applied to general polygons. One of the polygons in every operation will be a point visibility polygon, with at most n vertices — the same as the number of vertices in the polygonal environment.

We have implemented this approach using standard routines for computing visibility polygons, the reduced visibility graph and shortest paths (Dijkstra's algorithm) [37]. To compute the union of visibility regions, we used the *gpc* library developed by Alan Murta based on an approach proposed in Ref. [42].

4.4. Numerical Results: Combinatoric Layer

We tested our algorithm using the polygonal world shown in Fig. 10. The black regions correspond to the obstacles and the small circles to the sensing locations L_i.

In this example, we chose a subset of nonconvex vertices whose visibility region union totally cover the environment. We have chosen a small subset, 10 locations (this order of magnitude is the maximum for problems for which one can hope to find the globally optimal solution, due to computational complexity) to be able to compute the globally optimal paths and compare them against the result obtained by our algorithm.

For comparison, we computed three routes: (a) the route that minimizes the expected value of the time to find the object (i.e., the optimal solution), (b) the route that minimizes the distance traveled and (c) the route generated by our heuristic algorithm. These are shown in Fig. 10a–c, respectively. The results are summarized in Table 1, where for each route we show the expected value for the time required to find the object, the total distance traveled (which is proportional to the worst-case time to find the object) and the required computation time. With respect to the optimal solution, the route generated by our algorithm is about 4% worse in expected value of the time to find the object and about 7% worse in distance traveled.

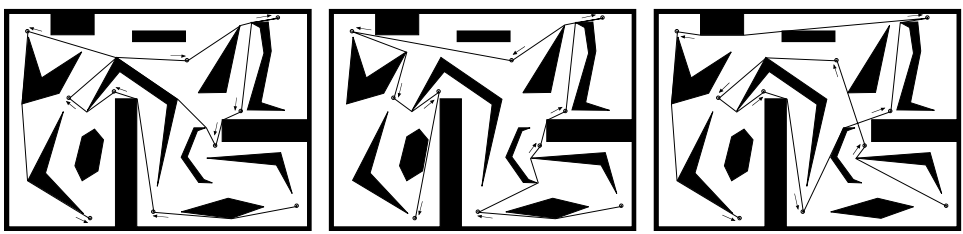


Figure 10. Routes to search for an object by different criteria: the optimal expected value of time (a), the optimal distance traveled (b) and the heuristic utility function (c).

Table 1.
Comparison between the three strategies

Strategy	Expected time	Distance traveled	Processing (s)
Optimal expected time	943.21	2783.20	892.82
Optimal distance	994.79	2273.09	488.87
Heuristic algorithm	982.21	2970.43	0.44

However, in execution time, our algorithm is more than 2000 times faster. With respect to the minimum distance route, the route generated by our algorithm is about 1% better in expected value of the time to find the object, even though it is about 30% longer in distance traveled. In execution time, our algorithm is more than 1000 times faster.

4.5. Decomposing a Polygon into Regions for Continuous Optimization

As we have described above, we divide the environment using inflection rays (such as the segment A_1 in Fig. 11a), which delimit corner guard regions. The corner guard regions are used as subgoals that the robot must visit to cover the whole environment.

In the continuous optimization of the local paths, we compute, by means of an integral, the increase of the area seen as the robot moves and the locally optimal robot trajectory (see Section 5). To do so, we triangulate the environment. Every triangle area is computed rotating a line passing through a nonconvex vertex. Hence, in our division of the environment, we also use lines connecting nonconvex vertices and convex vertices.

These lines delimit regions where the edge being seen 'through' a nonconvex vertex changes. A segment line connecting a nonconvex vertex and a convex vertex is labeled R_1 in Fig. 11a. In Fig. 11a, the edge seen through nonconvex vertex G_1 while the robot is inside the shaded region is E_1, but as the robot moves out of this region, the edge seen through G_1 changes to E_2. These lines are important because the visibility region generated by the nonconvex vertex will change non-smoothly when the robot crosses one of them. Hence, the integral used to compute the increase of the area seen (as the robot moves) changes its form. Note that these second kind on lines are only used to compute local optimal paths, they are not needed to establish an ordering to visit regions. If the robot is moving between two subgoals and it encounters one of such lines, then a new portion of the local

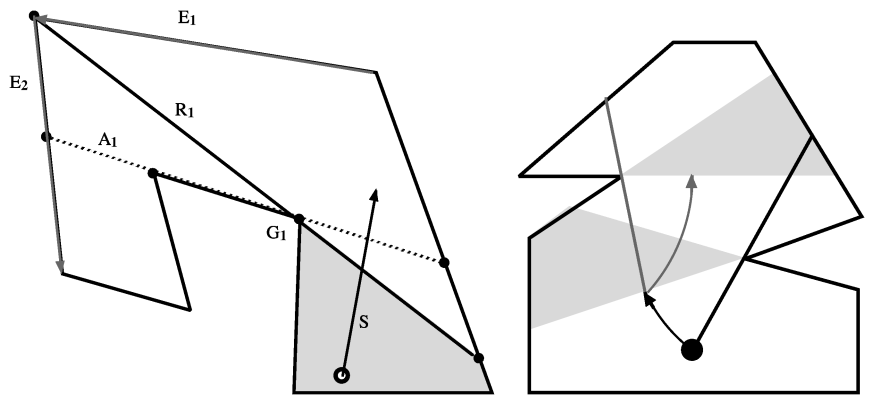


Figure 11. (a) Edges visible through a nonconvex vertex. (b) An environment with two nonconvex vertices.

optimal path is computed based on the polygon's segment associated to this line, but the robot continues its path toward the next corner guard region. Recall that the ordering to visit regions is defined by the top-level layer.

When the polygonal workspace contains more than one nonconvex vertex, as shown in Fig. 11b, then we generate paths that are optimal for only one corner at a time, shown in Fig. 11b. This strategy ignores the area visible behind other corners for the purpose of generating paths. The advantage of this policy is that it can be 'guided' to follow a certain order of nonconvex vertices. The top-level, combinatoric layer attempts to find global optimality by forcing a specific ordering for the low-level, continuous layer. Without this ordering, the end result would be a purely greedy algorithm that follows the visibility gradient and does not consider the amount of area visible in the future and the cost (distance) to travel the path. The visibility gradient is a vector that yields the maximal local reduction of the shadow caused by a visibility obstruction (nonconvex vertex). In symbols $\nabla f(V(S_r(q)|v, E)$, where $V(S_r(q)|v, E)$ is the visibility polygon which reduction depends on the robot path $S_r(q)$, given the nonconvex vertex position v and the environment E. The direction of ∇f is the orientation in which the directional derivative has associated the maximal local shadow area reduction.

We use this approach to make the robot follow an ordering of regions that globally reduces the expected value of time.

5. Optimizing the Local Paths

Once a visitation order for the L_i has been established by the top-level algorithm, it is necessary to generate a continuous path that successively visits the corresponding regions delimited by the inflection rays. In this section, we formalize the problem of generating locally optimal segments of this continuous path. Since the L_i correspond to nonconvex vertices in the environment, we consider here the case in which the robot will move around such a corner from one region to another. The final, global path will be a concatenation of these local paths. Below, we first derive the Euler–Lagrange equations that are satisfied by locally optimal paths and we then present numerical methods for their solution.

5.1. Conditions for Optimality

The problem we consider is shown in Fig. 12. In this case, the robot moves around a nonconvex vertex (corner) to explore the unseen area A'. For now, we assume that this is the only unseen portion of the environment.

Clearly, any locally optimal path for this problem will have the property that the unseen portion of the environment A' decreases monotonically with time (otherwise, the robot would be wasting time exploring previously seen areas). As can be seen in Fig. 12, as the robot moves around the corner, A' decreases monotonically if and only if the angle from the corner to the robot increases monotonically. For this reason, it is natural to express the path in polar coordinates (r, θ) with the origin at

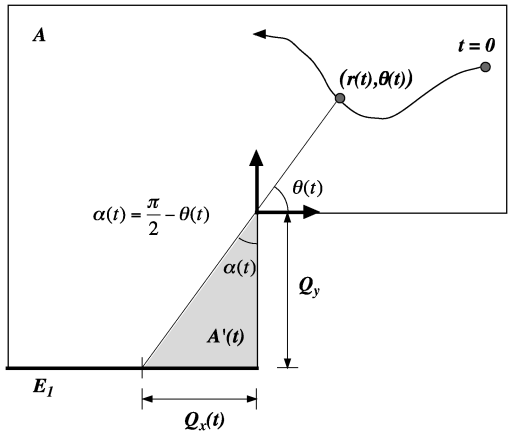


Figure 12. Base case for a continuous sensing robot.

the corner. For the moment, let us assume that the robot will have a starting position such that its line of sight will only sweep the horizontal edge E_1. While the analysis is only valid for an axis-parallel edge, it can be easily adapted to the general case.

Let $Q_x(t)$ and Q_y be horizontal and vertical distances from the origin to the point where the robot's line of sight through the origin intersects E_1. The area of the unexplored region $A'(t)$ (which corresponds to the probability mass of seeing the object and, therefore, the gain) is:

$$A'(t) = \frac{Q_y Q_x(t)}{2}. \tag{7}$$

As can be seen in Fig. 12:

$$\tan(\alpha(t)) = \frac{Q_x(t)}{Q_y},$$

and:

$$\alpha(t) = \frac{\pi}{2} - \theta(t).$$

Since $\tan(\pi/2 - \theta(t)) = 1/\tan(\theta(t))$, we have $\tan(\theta(t)) = Q_y/Q_x(t)$ and (7) can be written as:

$$A'(t) = \frac{Q_y Q_x(t)}{2} = \frac{{Q_y}^2}{2\tan(\theta(t))}.$$

For the case when the PDF of the object's location over the environment is constant, the probability of not having seen the object at time t is:

$$P(T > t|S) = \frac{A'(t)}{A} = \frac{{Q_y}^2}{2A\tan(\theta(t))}, \tag{8}$$

where A is the area of the whole environment.

Finally, from (5) and (8):

$$E[T|S] = \frac{{Q_y}^2}{2A} \int_0^{t_f} \frac{\mathrm{d}t}{\tan(\theta(t))}. \tag{9}$$

Equation (9) is useful for calculating the expected value of the time to find an object given a robot trajectory S expressed as a parametric function $\theta(t)$.

The calculus of variations [43] is a mathematical tool employed to find stationary values (usually a minimum or a maximum) of integrals of the form:

$$I = \int_a^b F(x, y, y')\,\mathrm{d}x, \tag{10}$$

where x and y are the independent and dependent variables, respectively. The integral in (10) has a stationary value if and only if the following Euler–Lagrange equation is satisfied:

$$\frac{\partial F}{\partial y} - \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial F}{\partial y'}\right) = 0. \tag{11}$$

In our case, we cannot apply the prototypical Euler–Lagrange equation directly to expression (9) for two reasons. First, r and θ are expressed as parametric equations, instead of one as a function of the other. This is not really a problem, because expressions very similar to (11) can be derived to accommodate the case of parametric functions [43]. The real problem is that (9) does not impose any constraints on the parametric equations describing the robot motion. The optimal trajectory without any constraints would be one where θ increases infinitely fast.

To address both of these problems, we introduce the constraint that the robot moves with constant (unitary) speed (note that by proceeding in this manner, we are considering the cost of moving the robot). To do this, we express its velocity vector as a generalized motion [44] on a basis where one component U_r is radial from the origin and the other U_θ is perpendicular, as shown in Fig. 13. Both U_r and U_θ are unit vectors and define an orthogonal basis. In this basis, the robot's velocity (in polar coordinates) is described as:

$$V = \dot{r}U_r + r\dot{\theta}U_\theta.$$

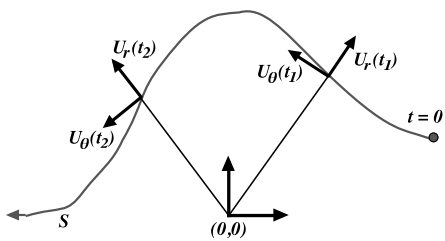


Figure 13. Generalized motion of a particle moving along path S.

The constraint that the robot speed is constant can be expressed as:

$$\|V\| = \dot{r}^2 + r^2\dot{\theta}^2 = 1. \tag{12}$$

Starting with (12), it is possible to express the differential of time as a function of a differential of θ. This will allow us to rewrite the parametric equation as a function in which θ and r are the independent and dependent variables, respectively:

$$1 = \frac{(\mathrm{d}r)^2}{(\mathrm{d}t)^2} + r^2\frac{(\mathrm{d}\theta)^2}{(\mathrm{d}t)^2}$$

$$(\mathrm{d}t)^2 = \big((\mathrm{d}r)^2 + r^2(\mathrm{d}\theta)^2\big)\frac{(\mathrm{d}\theta)^2}{(\mathrm{d}\theta)^2}$$

$$(\mathrm{d}t)^2 = \big(r'^2 + r^2\big)(\mathrm{d}\theta)^2$$

$$\mathrm{d}t = \big(r'^2 + r^2\big)^{1/2}\,\mathrm{d}\theta, \tag{13}$$

where $r' = \mathrm{d}r/\mathrm{d}\theta$. Substituting (13) into (9), we obtain an expression for the expected value of time to find an object where the robot's trajectory S is expressed as r being a function of θ:

$$E[T|S] = \frac{{Q_y}^2}{2A}\int_{\theta_\mathrm{i}}^{\theta_\mathrm{f}} \frac{1}{\tan(\theta)}(r'^2 + r^2)^{1/2}\,\mathrm{d}\theta. \tag{14}$$

To find stationary values of (14), we use (11) with $x = \theta$, $y = r$ and $F = 1/\tan\theta(r'^2 + r^2)^{1/2}$. After simplification, this yields the second-order nonlinear differential equation:

$$r'' = r + \frac{2r'^2}{r} + \frac{2}{\sin(2\theta)}\left(r' + \frac{r'^3}{r^2}\right). \tag{15}$$

Solutions to (15) define stationary values for the expected value of time in (14). If this is coupled with a sufficient condition for optimality (like the transversality condition [45]), then we will obtain the route to move around a nonconvex vertex (corner) to search the area on the other side optimally.

Since closed-form solutions do not exist for (15), we now turn our attention to numerical solutions.

5.2. Numerical Integration

We solved (15) numerically using an adaptive step-size Runge–Kutta method [46]. Since this equation is of second order, any numeric approach that integrates it as an initial value problem requires two initial conditions: $r(\theta_\mathrm{i})$ and $r'(\theta_\mathrm{i})$. We know the staring point $r(\theta_\mathrm{i})$ and the integration range $(\theta_\mathrm{i}, \theta_\mathrm{f})$, but we do not impose any other constraints on the trajectories other than unitary speed. Therefore, the possible solutions are a family of curves that depend on the value of the first derivative at the beginning of the integration range $r'(\theta_\mathrm{i})$. These are shown in Fig. 14.

Most of the possible solutions diverge long before they reach the end of the integration range. In fact, it is evident from (15) that the solution is not defined there (at $\theta_f = \pi/2$). However, it is possible to get arbitrarily close and, to do so, the first derivative at the end of the integration range must be such that the trajectory approaches the target manifold (the vertical line in Fig. 12) perpendicularly. This translates to stating that $r'(\theta_f) = 0$. In fact, the transversality condition for the Euler–Lagrange equation establishes that, in order to satisfy the equation and obtain a minimum, the solution function must be perpendicular to the target manifold at $t = t_f$ [45].

This observation allows us to integrate equation (15) as a two-point boundary value problem, where we specify the position at the beginning of the integration range $r(\theta_i)$ and the first derivative at the end $r'(\theta_f)$. For this, we coupled the Runge–Kutta algorithm with a globally convergent Newton–Raphson method [46].

Figure 15a shows the paths generated for six different starting positions in solid black lines. To save space, the figure only shows the upper right portion of an environment similar to that in Fig. 12 (the units on the axes are arbitrary).

To verify the accuracy of this solution, we have also found optimal trajectories using simulated annealing [47] and compared these to the trajectories found using numerical integration. In our implementation, we have defined the solution trajectory using a set of control point that collectively define the system state. The results are shown in Fig. 15b. As can be seen, the general shape of the trajectories generated for six different starting positions by our numerical integration (solid lines) and simulated annealing (control points) are very similar. We should point out, however, that each simulated annealing run took more than 1 h, whereas the numeric integration is done in a fraction of a second. As mentioned before, Fig. 15b only shows the upper right section of an environment, like that in Fig. 12 of arbitrary dimensions.

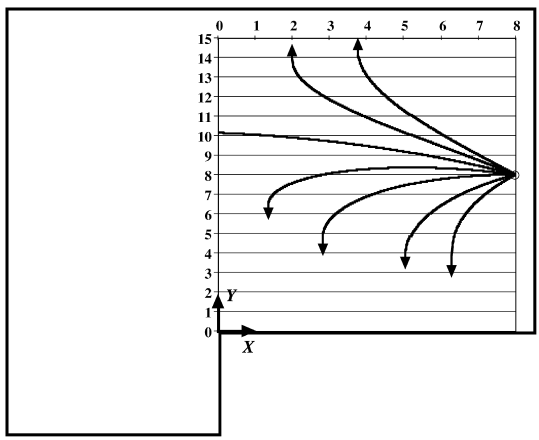


Figure 14. Family of curves depending on initial conditions.

6. Simulation Results

This section presents an example of how our proposed two-layered approach can be used to generate a continuous trajectory that covers a polygon with the goal of reducing the expected value of the time to find an object along that trajectory.

Figure 16a shows a polygon and a staring position P (near the bottom). We selected one L_i near each nonconvex vertex and used the algorithm described in Section 4 to find an efficient ordering for visiting the L_i. This algorithm returns a complete ordering, all the sensing locations (called guards) associated to the nonconvex vertices are included once. However, the set $\{L_i\}$ can be redundant; since sensing is done continuously the polygon may be completely covered before all L_i have been visited. As a consequence, some of the L_i late in the ordering may not need to be visited. This is the case of guards L_4 and L_5 in Fig. 16a. Note that the

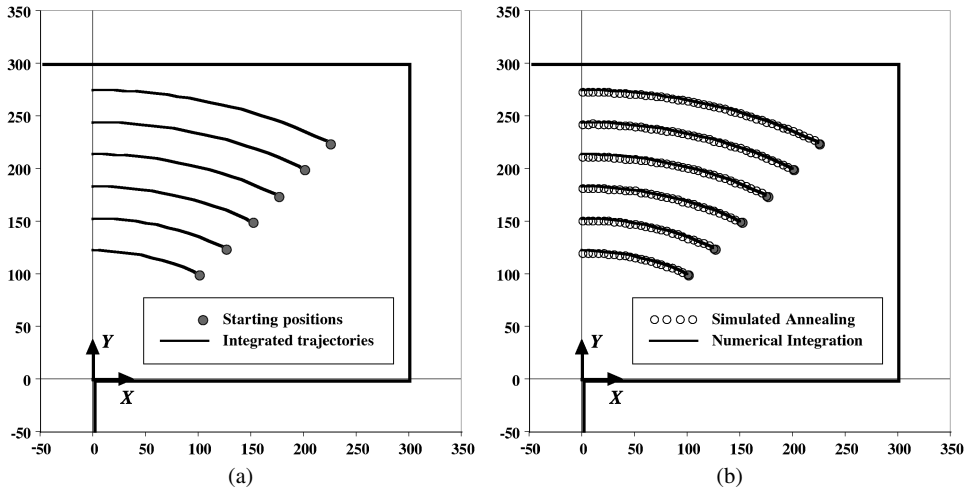


Figure 15. Optimal trajectories. (a) Numerical integration. (b) Simulated annealing.

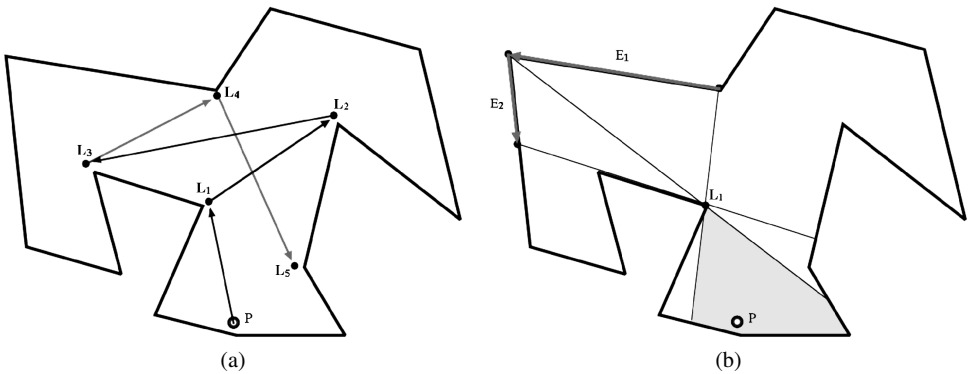


Figure 16. (a) A polygon and the guards. (b) Edges visible through guard L_1.

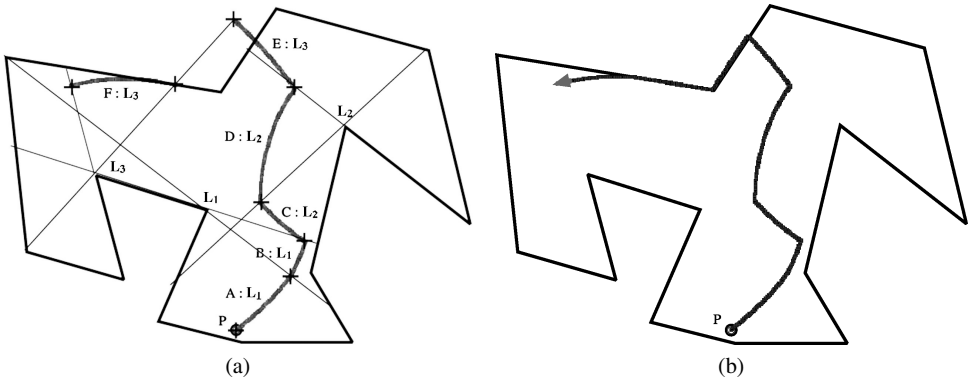


Figure 17. (a) Locally optimal trajectories for the guards that generated them. (b) The final trajectory for a polygon.

computed ordering is still valid, since the redundant sensing locations are the last in the ordering — the least important to minimize the expected value of time to find the object.

Once an ordering has been established, the trajectory is generated piecewise according to which guard is to be visited. The robot does not actually travel towards the guard, but rather it goes around its associated nonconvex vertex in a locally optimal trajectory, as described in Section 5. A locally optimal portion of the complete path is generated for every edge seen through the current nonconvex vertex. For example, in Fig. 16b, as the robot moves from the starting position P, in the shaded region, the section of the environment that will be visible through guard L_1 is bounded by edge E_1, i.e., as the robot moves, its line of sight through the corner will 'sweep' edge E_1 until it reaches edge E_2. At this point, the shape of the current subpath changes as it is now edge E_2 that will be swept. When the trajectory reaches one of the inflection rays associated with the nonconvex vertex of the current guard, the process starts over with the next region in the ordering.

Figure 17a shows all the path segments (A–F) generated for the polygon and the guards to which they correspond. There may be occasions, such as portion E, where the locally optimal trajectory would leave the polygon. In this case, the route is saturated and made to follow the polygon boundary. Note that the end-points of each path portion correspond to critical events, which occur at inflection rays or when there is a transition between the edges that are currently been seen through the corner.

Figure 17b shows the final trajectory. The zig–zag motion is not necessarily bad because a good path must find a compromise between advancing to the next region and sensing a larger portion of the environment as soon as possible.

For this particular example, the expected value of the time to see the object along the shown path is 115.3. This contrasts with the expected value along the straight line segments shown in Fig. 16a ($L_1 \rightarrow L_2 \rightarrow L_3$), which turns out to be 136.9.

7. Discussion and Conclusions

We addressed the problem of continuous sensing for expected value search in polygonal environments. This problem involves the generation of a motion strategy that minimizes the expected value of the time to find an object.

We presented a two-level algorithm that determines an efficient ordering of visiting regions and then generates locally optimal subpaths to construct a complete trajectory.

The final trajectory is not globally optimal for two reasons: (i) since the discrete version of the problem is NP-hard, we proposed a heuristic algorithm, and (ii) we chose to decouple the task of finding an ordering and moving between regions (bounded by inflection rays).

Obviously, the optimal paths will depend on the general shape of the polygon. For example, in polygons where most of the area is visible towards the end of the trajectory a motion strategy that moves the robot in the visibility graph will yield good results. This happens because it is reducing the distance to travel up to the point where it is more likely to find the object. In contrast, if the majority of the visible area lies near the starting point a completely greedy algorithm that follows the visibility gradient will perform better. In our case, the high-level, combinatoric layer attempts to find global optimality by forcing a specific ordering for the low-level, continuous layer. Without this ordering, the end result would be a purely greedy algorithm that does not consider the amount of area visible in the future and the cost (distance) to travel the path. For this reason, our algorithm presents a useful trade-off. Furthermore, our locally optimal paths are better than traveling in straight lines (the shortest paths in Euclidean sense).

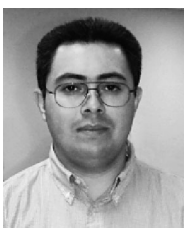
References

1. A. Sarmiento, R. Murrieta and S. A. Hutchinson, An efficient strategy for rapidly finding an object in a polygonal world, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, pp. 1153–1158 (2003).
2. A. Sarmiento, R. Murrieta and S. A. Hutchinson, A sample-based convex cover for rapidly finding an object in a 3-D environment, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Barcelona, pp. 3486–3491 (2005).
3. J. Gaspar, N. Winters and J. Santos Victor, Vision-based navigation and environmental representation with an omni-directional camera, *IEEE Trans. Robotics Automat.* **16**, 890–898 (2000).
4. H. Ishiguro, T. Sogo and M. Barth, Baseline detection and localization for invisible omnidirectional cameras, *Int. J. of Comp. Vis.* **58**, 209–226 (2004).
5. A. Sarmiento, R. Murrieta and S. A. Hutchinson, Planning expected-time optimal paths for searching known environments, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Sendai, pp. 872–878 (2004).
6. D. H. Yang and S. K. Hong, A roadmap construction algorithm for mobile robot path planning using skeleton maps, *Adv. Robotics* **21**, 51–63 (2007).
7. D. H. Kim and S. Shin, Local path planning using a new artificial potential function composition and its analytical design guidelines, *Adv. Robotics* **20**, 25–47 (2006).

8. D. H. Kim and S. Shin, New repulsive potential functions with angle distributions for local path planning, *Adv. Robotics* **20**, 25–47 (2006).
9. W. Yossawee, T. Tsubouchi, M. Kurisu and S. Sarata, A semi-optimal path generation scheme for a frame articulated steering-type vehicle, *Adv. Robotics* **20**, 867–896 (2006).
10. K. Nagatani, Y. Iwai and Y. Tanaka, Sensor-based navigation for a car-like mobile robot based on a generalized Voronoi graph, *Adv. Robotics* **13**, 779–792 (1998).
11. J.-B. Hayet, C. Esteves and R. Murrieta-Cid, A motion planner for maintaining landmark visibility with a differential drive robot, in: *Proc. Workshop in Algorithmic Foundations of Robotics*, Gunajuato, in press (2009).
12. S. LaValle and S. Hutchinson, An objective-based framework for motion planning under sensing and control uncertainties, *Int. J. Robotics Res.* **17**, 19–42 (1998).
13. T. Edmonds, A. Rowstron and A. Hopper, Using time encoded terrain maps for cooperation planning, *Adv. Robotics* **13**, 779–792 (1998).
14. C. G. Alvarez Jerez, Y. Hasimoto, T. Matsuda and T. Tsuchita, Environment representation using enclosed obstacles and minimum-turns path planning, *Adv. Robotics* **12**, 313–481 (1998).
15. J. O'Rourke, *Art Gallery Theorems and Algorithms*. Oxford University Press, Oxford (1987).
16. W. P. Chin and S. Ntafos, Optimum watchman routes, *Inform. Process. Lett.* **28**, 39–44 (1988).
17. S. Hert, S. Tiwari and V. Lumelsky, A terrain-covering algorithm for an AUV, *Autonomous Robots* **3**, 91–119 (1996).
18. E. U. Acar, H. Choset and P. N. Atkar, Complete sensor-based coverage with extended-range detectors: a hierarchical decomposition in terms of critical points and Voronoi diagrams, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Hawaii, Maui, HI, pp. 1305–1311 (2001).
19. H. H. González-Baños and J.-C. Latombe, Navigation strategies for exploring indoor environments, *Int. J. Robotics Res.* **21**, 829–848 (2002).
20. G. Borghi and V. Caglioti, Minimum uncertainty explorations in the self-localization of mobile robots, *IEEE Trans. Robotics Automat.* **14**, 902–911 (1998).
21. G. Oriolo, G. Ulivi and M. Vendittelli, Real-time map building and navigation for autonomous robots in unknown environments, *Trans. Syst. Man Cybernet.* **28**, 316–333 (1998).
22. B. Tovar, L. Muñoz, R. Murrieta-Cid, M. Alencastre, R. Monroy and S. Hutchinson, Planning exploration strategies for simultaneous localization and mapping, *Robotics Autonomous Syst.* **54**, 314–331 (2006).
23. I. Suzuki and M. Yamashita, Searching for a mobile intruder in a polygonal region, *SIAM J. Comput.* **21**, 863–888 (1992).
24. S. M. LaValle, D. Lin, L. J. Guibas, J.-C. Latombe and R. Motwani, Finding an unpredictable target in a workspace with obstacles, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, pp. 737–742 (1997).
25. L. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin and R. Motwani, Visibility-based pursuit–evasion problem, *Int. J. Comput. Geometry Applic.* **9**, 471–194 (1999).
26. S. M. LaValle and J. Hinrichsen, Visibility-based pursuit–evasion: an extension to curved environments, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Detroit, MI, pp. 1677–1682 (1999).
27. R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim and S. Sastry, Probabilistic pursuit–evasion games: theory, implementation, and experimental evaluation, *IEEE Trans. Robotics Automat.* **18**, 662–669 (2002).
28. B. Jung and G. Sukhatme, Tracking targets using multiple robots: the effect of environment occlusion, *J. Autonomous Robots* **12**, 191–205 (2002).

29. V. Isler, S. Kannan and S. Khanna, Locating and capturing an evader in a polygonal environment, in: *Proc. 7th Int. Workshop on the Algorithmic Foundations of Robotics*, Utrecht, pp. 351–367 (2004).
30. V. Isler, S. Kannan and S. Khanna, Randomized pursuit–evasion in a polygonal environment, *IEEE Trans. Robotics* **5**, 864–875 (2005).
31. B. Tovar, R. Murrieta-Cid and S. LaValle, Distance-optimal navigation in an unknown environment without sensing distances, *IEEE Trans. Robotics* **23**, 506–518 (2007).
32. P.-S. Laplace, *Théorie Analityque des Probabilités*. Courceir, Paris (1812).
33. K. W. Boyer and C. Dyer, Aspect graphs: an introduction and survey of recent results, *Int. J. Imaging Syst. Technol.* **2**, 315–328 (1990).
34. R. Murrieta, A. Sarmiento and S. Hutchinson, On the existence of a strategy to maintain a moving target within the sensing range of an observer reacting with delay, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, pp. 1184–1191 (2003).
35. T. C. Shermer, Recent results in art galleries, *Proc. IEEE* **80**, 1384–1399 (1992).
36. H. Rohnert, Shortest paths in the plane with convex polygonal obstacles, *Inform. Process. Lett.* **23**, 71–76 (1986).
37. T. H. Cormen, C. Stein, R. L. Rivest and C. E. Leiserson, *Introduction to Algorithms*, 2nd edn. McGraw-Hill, New York (2001).
38. M. R. Garey and D. S. Johnson, *Computers and Intractability*. Freeman, New York (1979).
39. S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*. Wiley, New York (1987).
40. J. E. Goodman and J. O'Rourke (Eds), *Handbook of Discrete and Computational Geometry*. CRC Press, Boca Raton, FL (1997).
41. G. Greiner and K. Hormann, Efficient clipping of arbitrary polygons, *ACM Trans. Graphics* **17**, 71–83 (1998).
42. B. R. Vatti, A generic solution to polygon clipping, *Commun. ACM* **35**, 56–63 (1992).
43. C. Fox, *An Introduction to the Calculus of Variations*. Dover, New York (1987).
44. R. Resnik and D. Halliday, *Physics*. Wiley, New York (1977).
45. A. P. Sage and C. C. White, *Optimum Systems Control*. Prentice Hall, Englewood Cliffs, NJ (1977).
46. W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge (1993).
47. S. Kirkpatrick, C. D. Gelatt, Jr and M. P. Vecchi, Optimization by simulated annealing, *Science* **220**, 671–680 (1983).

About the Authors



Alejandro Sarmiento obtained a BE in Electronic Systems Engineering (1995) and a MS in Automation–Intelligent System (1998) from Monterrey Institute of Technology and Higher Education. In 1998, he was awarded a fellowship from the National Council for Science and Technology to pursue graduate studies in Computer Science at the University of Illinois. He obtained his PhD in Computer Science from the University of Illinois, in 2004.

Rafael Murrieta-Cid received his PhD from the Institut National Polytechnique of Toulouse, France (1998). His PhD research was done in the Robotics and Artificial Intelligence group of the LAAS-CNRS. In 1998–1999, he was a Postdoctoral Researcher in the Computer Science Department at Stanford University. In 2002–2004, he was working as a Postdoctoral Research Associate in the Beckman Institute and Department of Electrical and Computer Engineering of the University of Illinois at Urbana-Champaign. From August 2004 to January 2006, he was Professor and Director of the Mechatronics Research Center in Tec de Monterrey, Campus Estado de México, México. Since March 2006, he has been working in the Mathematical Computing Group at Centro de Investigación en Matemáticas, Guanajuato México. He is mainly interested in robotics and robot motion planning, and has publised more than 40 papers in journals and international conferences on these topics.

Seth Hutchinson received his PhD from Purdue University, in 1988. In 1990, he joined the faculty at the University of Illinois in Urbana-Champaign, where he is currently a Professor in the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Beckman Institute for Advanced Science and Technology. He is Editor-in-Chief of the *IEEE Transactions on Robotics*, and previously served as the first Editor-in-Chief for the RAS Conference Editorial Board. He currently serves on the Editorial Boards of the *International Journal of Robotics Research* and the *Journal of Intelligent Service Robotics*. In 1996, he was a Guest Editor for a special section of the *Transactions* devoted to the topic of visual servo control, and in 1994 he was Co-chair of an IEEE Workshop on Visual Servoing. In 1996 and 1998, he co-authored papers that were finalists for the King-Sun Fu Memorial Best Transactions Paper Award. He was co-chair of IEEE Robotics and Automation Society Technical Committee on Computer and Robot Vision, from 1992 to 1996, and has served on the program committees for more than fifty conferences related to robotics and computer vision. He has published more than 150 papers on the topics of robotics and computer vision, and is co-author of the books *Principles of Robot Motion: Theory, Algorithms, and Implementations* (MIT Press) and *Robot Modeling and Control* (Wiley). He is a Fellow of the IEEE.