An Efficient Strategy for Rapidly Finding an Object in a Polygonal World

A. Sarmiento C.S. **R. Murrieta** E.C.E. **S.A. Hutchinson** E.C.E.
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
{ asarmien, murrieta, seth } @ uiuc.edu

***Abstract*— In this paper we propose an approach to solve the problem of finding an object in a polygon which may contain holes.**

We prove that a policy for complete exploration that is optimal in the distance traveled is not the best one for finding an object as quickly as possible.

The object search problem is shown to be NP-hard by reduction, therefore, we propose the heuristic of an utility function, defined as the ratio of a gain over a cost and an greedy algorithm in a reduced search space that is able to explore several steps ahead without incurring in too high a computational cost.

This approach was implemented and simulation results are shown.

I. Introduction

The problem of determining a good strategy to accomplish a visibility-based task such as environment modeling [2], pursuit-evasion [5] [6], or object finding [4] [12], is a very challenging an interesting research area. Specially when the sensors are not static but rather are carried by mobile robots.

We are interested in the problem of finding an object. Our goal is to find an efficient strategy to perform the object search.

In general, the robot will not be able to see the whole environment in a single sensing. Therefore, more than one perceptions will be needed to completely cover the environment. There are several schemes to generate and combine these perceptions.

One approach is to have the robot continuously sense the world as it moves along a given trajectory, thereby scanning the environment in a continuous fashion. In this case, it is not clear how to generate a globally optimal trajectory for a given criterion. A greedy strategy could use the gradient of the new visibility information to guide the search. This strategy could be based on critical events such as crossing lines in an aspect graph based on perspective projection [12].

Another approach is to make the robot sense the environment only at specific locations. This changes the nature of the problem from continuous to discrete, with information arriving in blocks. This introduces the problem of generating an "appropriate" set of sensing locations.

There are several criteria for determining the goodness of this set. For example, the minimal number of locations (art gallery problem [8]), locations along the shortest path that covers the whole environment (shortest watchman path [11]), and so on.

In this paper we will assume that the set of locations is given as input – they will not be generated automatically. In any case, once the sensing locations are known, it is still necessary to visit those locations in an specific order to minimize the expected time to find the object. This transforms the object search into a combinatorial problem.

In this paper our objective is to generate an exploration strategy based on the given sensing locations that finds the object as quickly as possible. That is, a strategy that minimizes the expected time it takes to find it (as explained in the next section). We will show that, under this definition of optimality, *the best exploration strategy is not necessarily the one that minimizes the distance traveled*.

II. Problem Definition

In general terms, we define the problem of searching for an object as follows: Given a mobile robot with some kind of sensing capabilities, a completely known environment and an object sitting somewhere in the world, develop a motion strategy for the robot to find the object in the least amount of time.

At this point we are not concerned with the geometry of the robot or the capabilities of the sensor (field of view, range, resolution and so on). For now, we consider only a point robot with an omnidirectional, infinite range sensor.

Furthermore, we assume that the known environment W is a polygon that may contain holes and that the probability of the object being in any specific point is evenly distributed throughout the polygon's interior. Therefore, the probability of the object being in any subset $R \subseteq W$ is proportional to the area of R.

We also assume that we are given a set of locations L (also known as *guards* from the art gallery problem [11]) from which every point in W can be seen. The visibility region of location L_j, denoted $V(L_j, W)$, is the set of points in W that have a clear line of sight to L_j (the line segment connecting them does not intersect the exterior of W). The set L is chosen so that the associated visibility regions define a cover of W. This means that their union adds up to the environment W.

$$\bigcup_j V(L_j, W) = W$$

We do not require nor assume the set L to be minimal.

Our exploration protocol is as follows: The robot always starts at a particular location in L (the starting point) and visits the other locations as time progresses (it follows the shortest paths between them). It only gathers information about the environment (sensing) when it reaches one of these locations – it does not sense while moving. We describe the route followed

by the robot as a series of locations L_{i_k} that starts with the robot's initial location and includes the other locations once. It is important to note that while L_j refers to locations in the environment, L_{i_k} refers to the *order* in which those locations are visited. That is, the robot always starts at L_{i_0}, and the k-th location it visits is referred to as L_{i_k}. Obviously, every L_{i_k} has a corresponding L_j in the environment, but their indices need not match.

For any route R, we define the time to find the object T as the time it takes to go through the locations – in order – until the object is first seen. We assume that the robot will be able to identify the object from any given viewpoint and that there are no other objects that could be mistaken as the searched object [3].

Our goal is to find the route that minimizes the expected value of the time it takes to find the object

$$E[T|R] = \sum_j t_j P(T = t_j) \tag{1}$$

where

$$P(T = t_j) = \frac{Area\left(V\left(L_{i_j}, W\right) \setminus \bigcup_{k<j} V\left(L_{i_k}, W\right)\right)}{Area(W)}$$

Where $P(T = t_j)$ is the probability of finding the object at the j-th *visited* location and t_j is the time the robot takes to go from its initial position, through all locations along the route until it reaches L_{i_j}. Since the robot only senses at specific locations, we denote this probability of finding the object at the L_{i_j} location as $P\left(L_{i_j}\right)$.

The probability of finding the object from a given location is proportional to the visibility polygon of that region minus the its intersection with the already explored space up to that point.

It is important to point out the difference between this problem and the problem of completely exploring the environment in the least amount of time. Both problems seem to be related, but the best trajectories for each of them are not the same.

For the problem of searching for an object, the robot stops when the object is found, therefore, not all locations are visited every time. Hence, it might payoff to visit the most promising locations first. In other words, if the robot first visits those locations where the probability of finding the object is higher, it is likely that it will be finished sooner. Of course, in the worst case the robot must visit all locations before finding the object, but we are interested in minimizing the *average* time it takes.

Consider the problem of complete exploration with a similar definition of viewing locations (guards) and associated visibility polygons. In this case, it is assumed that the robot will visit all locations at least once, so the shortest path between them must be found (minimum distance traveled). This problem completely disregards the relative "goodness" of each region, in fact, goodness is not even defined (one could think it refers to the size of the visibility regions or some other similar measurement).

Now we will prove that the route that minimizes the distance traveled is not the best one for searching an object.
Proposition: For a given set of sensing locations, the route that minimizes the distance traveled is not the route that, on average, finds the object as quickly as possible.
Proof: This is proved by counter example (see Fig. 1).

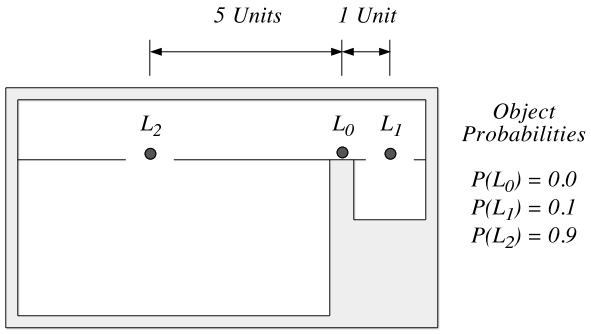

Fig. 1. Example with a simple environment

The robot starts on the corridor at location L_0. The object will always be in one of two rooms, and the probability of it being in either is related to the size of the room. The room to the right – seen from location L_1 – is smaller but lies closer to the initial location, while the room to the left – seen from L_2 – is larger but farther from initial position. There are only two routes the robot might take to solve this problem, go to the smaller room first ($L_0 \rightarrow L_1 \rightarrow L_2$) or go to the larger room first ($L_0 \rightarrow L_2 \rightarrow L_1$). For the following analysis, the robot is moving at a constant speed of 1 unit per second.

a) Route 1 –: If the robot goes to the smaller room first and then moves on to the larger room, it reaches L_1 at time 1 and L_2 at time 7. From (1), the expected value of the time it takes to find the object following this route is

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

The robot always completes its search after 7 seconds.

b) Route 2 –: If the robot moves to the larger room first and then goes to the smaller room, it reaches L_2 at time 5 and L_1 at time 11. The expected time in this case is

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

In the worst case, it will take the robot 11 seconds to find the object.

A robot following route 1 always finishes searching after 7 seconds, while a robot following route 2 takes 11. Route 1 is better for completely exploring the environment. However, the average time it takes for a robot following route 1 to find the object is $6.4\ secs$ whereas for route 2 is only $5.6\ secs$. Route 2 is better for finding the object as quickly as possible.

This example proves that a strategy that is optimal in the time it takes to completely explore the environment is not necessarily the best one to minimize the expected time to find an object in that environment. ∎

III. Proposed Solution

Since we assume that we are given a set of sensing locations that completely cover the environment, we are interested in finding an order of visiting those locations – the problem becomes a combinatorial search. In this section we present two algorithms for such a task. The first one is a traditional graph search that finds the optimal ordering but is intractable. The second is a greedy algorithm that can be computed in polynomial time and yields good results.

In general, the robot will not be able to travel between two locations by following a straight line. In this cases, we use a reduced visibility graph [9] and Dijkstra's Algorithm to follow the shortest path between them.

A. Algorithm for Optimal Ordering

Given a set of locations L that are guards to a polygonal region W, there exists an algorithm for computing the route that minimizes the expected time to find the object. It is described hereafter.

Construct a complete weighted graph as follows:

(1) For each location L_j, create a node N_j in the graph.
(2) For each pair of nodes N_j and N_k, add an edge with variable weight W_{jk}.
(3) The weight W_{jk} is dynamic, meaning it depends on the route followed by the robot before reaching N_j. These weights are calculated on-line.

The weight W_{jk} should correspond to the increase in expected time $\Delta E[T]$ the robot incurs by going from L_j to L_k. This is a function of the time in which it arrives at L_k, which in turn depends on the route followed by the robot up to that point.

In this graph, we need to find the path of minimum cost that starts at the robot's initial location L_{i_0} and includes all other locations. This can be accomplished with a *Branch and Bound* graph search. This search strategy maintains a list of nodes to be opened ordered by their accumulated cost. The next node to be expanded is always the head of the list, the one whose accumulated cost is currently minimal.

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

$$\begin{aligned} W_{jk} &= Time\,(N_k) \cdot P\,(L_k) \\ Time\,(N_k) &= Time\,(N_j) + Speed \cdot Dist\,(L_j, L_k) \end{aligned}$$

Then, the accumulated cost for the child is

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

Initially, the Branch and Bound list contains only the starting robot 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. When this happens, the currently best nodes continue to be expanded until

(a) A lower cost solution is found, in which case the better solution is saved and the process continues, or
(b) The lowest cost node is worse than the current solution. In this case we know that this 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, NP-hard.

B. Reduction from an NP-hard problem

The *Minimum Weight Hamiltonian Path Problem*, known to be NP-hard [1], can be reduced to the problem of finding the optimal visiting order of sensing locations which minimizes the expected time to find an object.

In order to make a formal reduction, we abstract the concept of environment and visibility regions. We only consider a set of locations which have an associated probability of finding the object which are independent of each other.

The reduction consists in defining the distance between the sensing locations as the edge weights of the Minimum Weight Hamiltonian Path Problem and setting the probabilities uniformly (same value for all).

Since the probabilities are set uniformly, the route that minimizes the expected time will be the exactly the same as the one that minimizes the distance traveled. This happens because the expected value of the time to find an object is determined only by the time it takes to reach locations along the route. Since time is proportional to distance, the route that minimizes time will also minimize the distance.

Given that the solutions to both problems are the same ordering of locations, finding a polynomial algorithm to solve these instances of the defined problem would also solve the Minimum Weight Hamiltonian Path Problem in polynomial time. Thereby proving that the proposed problem is NP-hard.

C. Utility Heuristic

Since trying to find an optimal solution is a futile effort, we decided to implement an iterative greedy strategy. One that tries to achieve a good result in one (or just a few) steps at a time.

In the obvious version of this algorithm the next location to visit is chosen as the one that causes the least increase in the partial calculation of (1) along the current route. That is, at each step of the route, calculate how much would the expected value of the time to find the object increase for going to the remaining locations and then choose the least increase. This has $O\left(n^2\right)$ complexity, because each step has to consider every available location.

This algorithm performs poorly. We believe this happens because the product in (1) makes locations with low probability be preferred and visited first, which seems contrary to what should be done.

For this reason, we propose an alternate greedy algorithm, called *utility greedy*, that tries to maximize an utility function. This function measures how convenient it is to visit a

determined location from another, and is defined as follows:

$$U\left(L_j, L_k\right) = \frac{P\left(L_k\right)}{Time\left(L_j, L_k\right)} \tag{2}$$

This means that if the robot is currently in L_j, the utility of going to location L_k is proportional to the probability of finding the object there and inversely proportional to the time it must invest in traveling.

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 finding the object is high. Intuitively, it is convenient to follow such an strategy, but its relationship with the expected value minimization will be more evident after the following analysis.

Consider a definition of expectation for a non-negative random variable, such as time, from [10]

$$E[T] = \int_0^\infty P(T > t)dt$$

This is equivalent to

$$E[T] = \int_0^\infty 1 - P(T \leq t)dt = \int_0^\infty 1 - F_T dt \tag{3}$$

Where F_T is a *cumulative distribution function*.

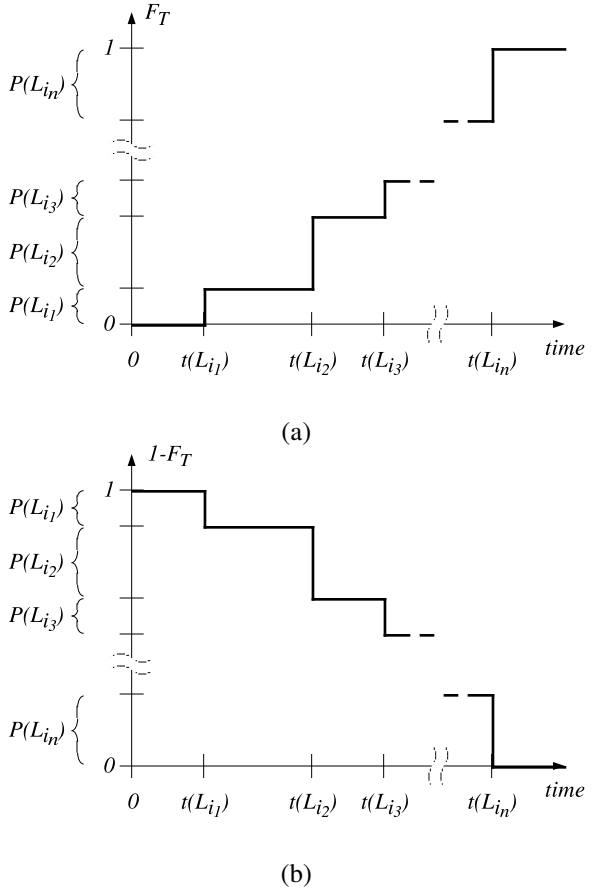

Fig. 2. Defined cumulative distribution functions. (a) F_T (b) $1 - F_T$

In our problem, every valid trajectory defines a particular cumulative distribution function of finding the object F_T. Since we are dealing with a discrete problem, the distributions are only piecewise continuous with the discontinuities being the times at which the robot reaches the distinct locations along the route, as shown in Fig. 2a.

By (3), we know that the expected value of a random variable with distribution F_T is the area under the curve $1-F_T$, shown in Fig. 2b. This area is the value we want to minimize.

One method for making this area small is to have the time intervals as small as possible and the probability changes (down step) as large as possible. This is the notion that our utility function in (2) captures; its value is larger when the probability of finding the object in a particular location is high (large down step) and/or when the location is near (small time interval).

D. Efficient Utility Greedy Algorithm

The utility function in (2) is sufficient to define a 1-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 $O\left(n^2\right)$.

However, it might be convenient to explore several steps ahead instead of just one to try to "escape local minima" and improve the quality of the solution found. The downside of this idea is that it usually increases the complexity of the algorithm by a factor of $O(n)$ for each step ahead.

To reduce this effect we propose a second heuristic that reduces the branching factor. The heuristic is that the children of each location can only be those other locations that are not strictly dominated according to the two variables in the utility function. As seen from the j-th location L_j, a location L_k strictly dominates another L_l if both of the following conditions are true

$$\begin{aligned} P\left(L_k\right) &> P\left(L_l\right) \\ Dist\left(L_j, L_k\right) &< Dist\left(L_j, L_l\right) \end{aligned}$$

Graphically, this is shown in Fig. 3. It is straightforward that dominating locations will lie on the convex hull of the remaining set of locations when plotted on the probability vs. distance plane.

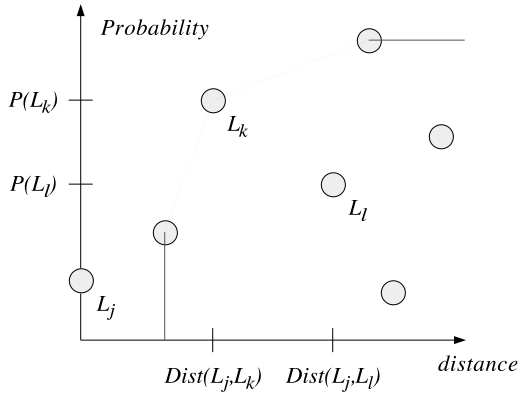

Fig. 3. Location dominance

By only considering a subset of the remaining locations at each step, we are reducing the branching factor, making it possible to explore more steps ahead without incurring in 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 described in the next section.

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 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 $\frac{n}{\log_B(n)}$ times for the complete route. This is depicted in Fig. 4.

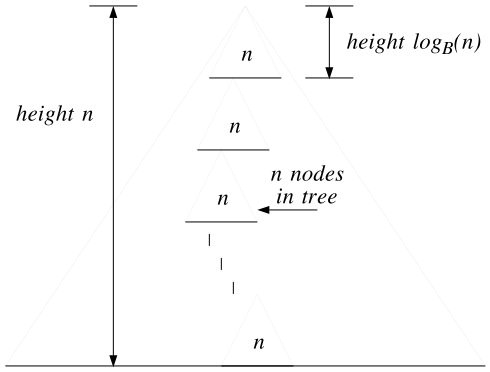

Fig. 4. Exploration algorithm

Thus, our final algorithm is as follows:

(1) 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 $O(n)$.
(2) 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 in $O\left(n\log(n)\right)$.
(3) When the number of nodes in the exploration tree has reached order $O(n)$, choose the best leaf according to the heuristic in (2), discard the current tree and start over with the best node as root.

This has to be repeated several times to generate a complete route, therefore the total complexity of the algorithm is

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

Of course, this result depends on the number of dominating locations being significantly smaller than n on average. Which may be difficult to determine for a specific problem. We know, for example, that the expected number of points on the convex hull of a set sampled uniformly from a convex polygon is of order $O\left(k\log(n)\right)$ for a k-sided polygon [7]. 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

$$O\left(n\cdot n\log(n)\cdot n\right)=O\left(n^3\log(n)\right)$$

This analysis only considers the time complexity of the search algorithm itself. It does not include the time complexity of performing polygon clipping operations. These are needed every step of the algorithm because they are used to calculate the probability of finding the object at any given location (which depends on the route followed up to that point).

IV. Simulation Results

For our simulations, we implemented routines for computing visibility polygons, the reduced visibility graph and shortest paths (Dijkstra's Algorithm). For calculating the union of visibility regions, we used the *gpc* library developed by Alan Murta [13].

This section presents the simulation results for the polygonal world shown in Fig. 5. The black regions correspond to the obstacles, the small circles to the sensing locations(guards) given as input and the grey region is the visibility polygon of the starting location.

For this instance, we generated the sensing locations manually. While we tried to find as few as possible, they do not correspond to any kind of optimal criteria.

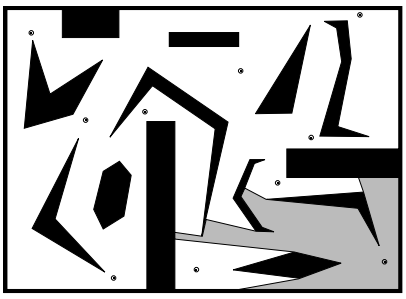

Fig. 5. Test polygonal world

For the following routes, we show the expected value of the time it takes to find the object following that particular route, and its total length. These values are given in arbitrary units, what really matters in the relative value differences between the routes. The execution times are in seconds for a regular PC workstation.

For this polygonal world, we computed three routes. The first one is the route that minimizes the expected value of the time to find the object (the optimal solution). For purposes of comparison, we also computed the route that minimizes the distance traveled, and finally, we show the route generated by our heuristic algorithm.

Fig. 6 shows the route that minimizes the expected value of the time to find the object – the optimal solution to our problem. For this route the expected value is 943.21 with a total distance traveled of 2783.20. This result took 892.82 seconds to compute.

Fig. 7 shows the route that minimizes the distance traveled from the starting location. In this case, the expected value of

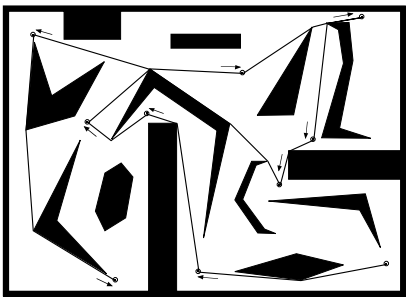

Fig. 6. Route that minimizes the expected time to find the object

the time to find the object is 994.79 with a total distance of 2273.09. This route was computed in 488.87 seconds. This result further shows that the best strategy to find an object as quickly as possible on average, is not the one that minimizes the distance traveled.

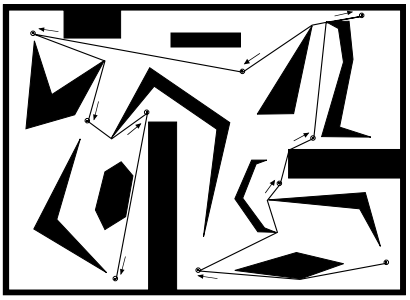

Fig. 7. Route that minimizes the distance traveled

Fig. 8 shows the route generated by our heuristic algorithm. The expected value along this route is 982.21 with a total distance traveled of 2970.43. This result was obtained in only 0.44 seconds.

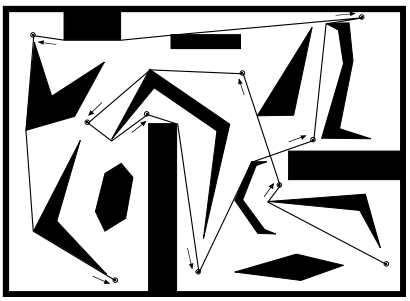

Fig. 8. Route generated by the utility heuristic algorithm

We were able to solve this instance of the problem optimally because the number of sensing locations is relatively small (10).

For this particular example, the expected value of the time to find the object along the route obtained by our heuristic algorithm is slightly smaller (by 1.2%) than along the route that minimizes the distance traveled. Of course, the length of the route is larger (by about 30%).

With respect to the optimal solution, the route generated by our algorithm is worse in both expected value of the time to find the object (by 4.1%) and distance travelled (by 6.7%). However, in execution time, our algorithm is more than 2000 times faster.

V. Conclusions

In this paper we proposed an efficient approach to solve the problem of searching an object in a polygonal environment. We defined an optimal solution as the route that minimizes the expected time it takes to find the object.

We proved that a policy for complete exploration that is optimal in the distance traveled is not necessarily the best one for finding an object as quickly as possible.

The problem itself was shown to be NP-hard by reduction, therefore, we proposed the heuristic of an utility function, defined as the ratio of a gain (increase in a cumulative distribution function) over a cost (travel time).

We also proposed an greedy algorithm in a reduced search space that is able to explore several steps ahead without incurring in too high a computational cost.

We showed experiments in simulation that suggest that the quality of the routes generated by our algorithm is close to the optimal solutions.

Future work will consist in the development of an approach to generate a set of sensing locations automatically. The set should be "helpful" to the problem of minimizing the expected time to find an object. It seems that desirable properties include low cardinality and low spread. A formal definition of these properties is a major part of the solution to the problem.

References

[1] Garey, M.R. and D.S. Johnson, *Computers and Intractability,* W. H. Freeman and Company, 1979.

[2] González-Baños, H.H. and J.C. Latombe, "Navigation Strategies for Exploring Indoor Environments," *to appear in Int. Journal of Robotics Research*.

[3] Lacroix, S., P. Grandjean and M. Ghallab, "Perception Planning for a Multi-Sensory Interpretation Machine," *in Proc. IEEE Int. Conf. on Robotics and Automation 1992*.

[4] LaValle, S.M. *et al*, "Finding an Unpredictable Target in a Workspace with Obstacles," *in Proc. IEEE Int. Conf. on Robotics and Automation 1997*.

[5] LaValle S.M. *et al*, "Motion Strategies for Maintaining Visibility of a Moving Target," *in Proc. IEEE Int. Conf. on Robotics and Automation 1997*.

[6] Murrieta-Cid, R., H.H. González-Baños and B. Tovar, "A Reactive Motion Planner to Maintain Visibility of Unpredictable Targets," *in Proc. IEEE Int. Conf. on Robotics and Automation 2002*.

[7] Preparata, F.P. and M.I. Shamos, *Computational Geometry: an Introduction,* Springer-Verlag New York, 1985.

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

[9] Rohnert, H., "Shortest Paths in the Plane with Convex Polygonal Obstacles," *Information Processing Letters,* 23:71-76, 1986.

[10] Ross, S.M., *Introduction to Probability and Statistics for Engineers and Scientists,* Wiley, 1987.

[11] Shermer, T.C., "Recent Results in Art Galleries," *Proc. of the IEEE,* Vol. 80, issue 9, September 1992.

[12] Tovar, B., S.M. LaValle and R. Murrieta-Cid, "Optimal Navigation and Object Finding without Geometric Maps or Localization," *accepted in IEEE Int. Conf. on Robotics and Automation 2003*.

[13] Vatti, B.R., "A Generic Solution to Polygon Clipping," *Communications of the ACM,* 35(7), pp.56-63, July 1992.