Proceedings of the 2005 IEEE
International Conference on Robotics and Automation
Barcelona, Spain, April 2005

A Sample-based Convex Cover for Rapidly Finding an Object in a 3-D Environment

Alejandro Sarmiento[†], Rafael Murrieta-Cid[‡] and Seth Hutchinson[†]
[†]*Beckman Institute for Advanced Science and Technology*
University of Illinois at Urbana-Champaign
Urbana Illinois, USA
{*asarmien, seth*}*@uiuc.edu*
[‡]*Mechatronics Research Center*
ITESM – Estado de México Campus
Atizapán de Zaragoza, Estado de México, México
rafael.murrieta@itesm.mx

***Abstract*— In this paper we address the problem of generating a motion strategy to find an object in a known 3-D environment as quickly as possible on average.**

We use a sampling scheme that generates an initial set of sensing locations for the robot and then we propose a convex cover algorithm based on this sampling. Our algorithm tries to reduce the cardinality of the resulting set and has the main advantage of scaling well with the dimensionality of the environment.

We then use the resulting convex covering to generate a graph that captures the connectivity of the workspace. Finally, we search this graph to generate trajectories that try to minimize the expected value of the time to find the object.

***Index Terms*— Motion Planning, Sensor Planning, Convex Cover, Hidden Guard Sets, Monte Carlo Approach**

I. Introduction

This work builds on our previous research on expected value search with mobile robots. We define the problem as follows: Given a mobile robot with sensing capabilities, a known environment and recognizable object in an unknown location, generate a motion strategy to find the object in the least amount of time on average.

We describe the object as a probability density function of its location over the environment; and we try to find a trajectory that minimizes the expected value of the time it takes to find it. We believe that minimizing the expected value is better than minimizing the total distance traveled (i.e. the worst-case time) because the robot stops once the object is found. Most times, it will not have to sense the whole environment. In [8] we proved that the two policies are not equivalent, that is, a trajectory that minimizes the distance traveled may not minimize the expected value of the time to find the object.

In [9] we proposed an approach to solve this problem in a polygonal environment. The basic idea was to define a set of sensing locations from which the whole environment is covered. We then defined a graph over these locations and used a greedy algorithm in a reduced search space able to explore several steps ahead without incurring too high a computational cost.

The present work builds on the same problem but in a three dimensional workspace. For this, we also define a set of sensing locations that cover the space, link them in a graph and perform a graph search to generate trajectories. The difference is that, in order to generate the sensing locations and calculate the size and shape of their visibility regions, we propose a probabilistic scheme instead of a deterministic one as before.

A. Approach Overview

To address the problem of rapidly searching for an object in a 3-D setting, we will use the sampling strategy proposed in [11] to cover the environment and provide probabilistic bounds on how good the coverage is. In our case, this strategy is also useful to determine an initial set of sensing locations that cover the environment.

Then, we will present an algorithm to transform this initial coverage into a convex decomposition of the space and define a new set of sensing locations that will be connected in a graph. Finally, we use our previously proposed greedy approach to search the resulting graph and generate trajectories that try to minimize the expected value of the time to find an object in the environment.

The convex cover and subsequent graph abstraction capture the connectivity of the workspace. There have been many related approaches that connect samples to capture connectivity of high dimensional spaces, for example, [4], [3] and [5] just to name a few. Our work is different in several aspects. First, we are interested in representing the workspace for searching an object, not in abstracting the configuration space for path planning purposes. Second, most techniques throw away "use-

less" samples, we keep all of them since we need to estimate the size of volumes. Finally, we can determine a clear-cut threshold of when it is no longer useful to continue sampling (as described below).

II. Sampling the Environment

We use probabilistic sampling to decompose the workspace into convex regions and to estimate the size of those regions. Our main motivation for sampling is the high computational complexity of visibility queries in three dimensions [2]. As mentioned before, we implemented the sampling strategy in [11] originally proposed to construct a probabilistic roadmap in the configuration space. The difference in our case is that we will sample the workspace (not the configuration space) and keep all samples, as opposed to throwing away those that do not join important configurations.

A. Sampling that Covers the Environment

In order to capture the size and shape of the workspace W, we generate a set of independent, uniformly distributed samples S in the interior of W. Among these samples, we choose a *hidden guard set* G. A set is called a hidden guard set if it covers the environment and individual member of the set are not visible to each other. Since we are approximating W with sample points, for the set G to cover the environment, every sample in S must be visible to at least one guard in G. That is,

$$\bigcup_{g_i \in G} Vis\,(g_i) = S,$$

where $Vis\,(g_i)$ is the set of points in S whose line segment to g_i does not intersect the exterior of the workspace $\left(\overline{W}\right)$. Also, since guards must not be mutually visible, $g_i \notin Vis\,(g_j) \quad \forall j \neq i$.

The algorithm to generate samples and the hidden guard set is as follows:

(1) Initialize the sample set S to empty.
(2) Initialize the hidden guard set G to empty.
(3) Set the counter of the number of consecutive visible samples n to zero.
(4) While n is less than a constant m,
  (4.1) Generate a uniformly distributed random sample in the interior of W.
  (4.2) If the sample is visible from at least one of the current guards in G, add it to the sample set S and increment n.
  (4.3) Else, add it as a new guard to G and reset n to zero.
(5) Return.

It is evident that this algorithm will generate a set of hidden guards that see all the other samples and – with a large enough m – the vast majority of the environment.

To determine how well the workspace has been covered, consider an environment of unit volume like the one shown in Fig. 1. Suppose that the portion of the environment that is visible from the guard set is A and has measure $\mu(A) = 1 - \epsilon$, while B is another portion with measure $\mu(B) = \epsilon$ that is not yet visible and does not contain a single sample point.

If samples are drawn independently, the probability of m consecutive points not falling into the uncovered region B is $P\,(A_m) = (1-\epsilon)^m$. After m consecutive samples, it is still possible that the actual size of B is greater than ϵ, but with a large m, we can bound this probability with a small value α. For this, we determine m as follows,

$$\begin{aligned} (1-\epsilon)^m &\leq \alpha \\ m\log(1-\epsilon) &\leq \log(\alpha) \\ m &\geq \frac{\log(\alpha)}{\log(1-\epsilon)}\ . \end{aligned}$$

Therefore, choosing a large enough m we can expect with certainty $(1-\alpha)$ that the size of the unseen region B is at most ϵ.

B. First Approach for Generating Trajectories

The algorithm above returns a set of guards that with high certainty cover the vast majority of the environment, and we can estimate the size of the visibility region of each guard (and their intersections) with a Monte Carlo approach [1]. This is sufficient to generate an efficient trajectory that tries to minimize the expected value of the time to find an object.

However, we would like to have more flexibility in the possible routes that the robot must follow. We would like to have the robot visit a region, as opposed to a single point (the guard). For this reason, we decided to decompose the space into overlapping convex regions, that is, a convex cover.

III. Sample-based Convex Cover

The algorithm we propose for the sample-based convex cover is based on the idea that there is a dual

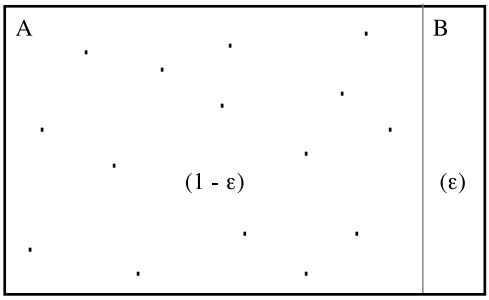


Fig. 1. An environment of unit volume and the sizes of covered (A) and uncovered (B) regions

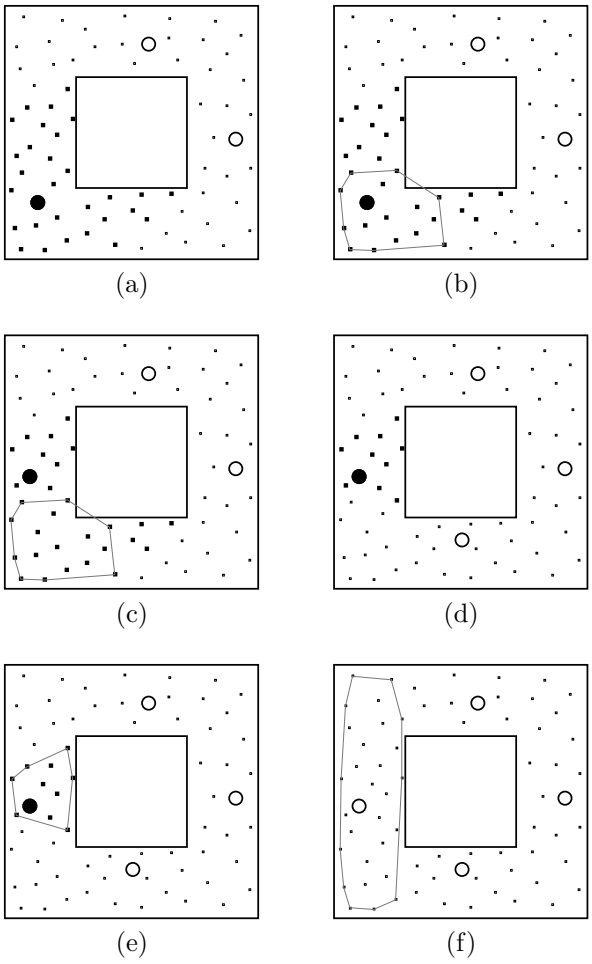


Fig. 2. Convex cover algorithm

between a *maximum hidden guard set* and a *minimum convex cover*. It has been proved that a minimum convex cover is an NP-hard problem [6], therefore our aim is just an efficient algorithm that tries to generate as few convex regions as possible, without any guarantee of optimality. Nevertheless, we have found that in practice, our algorithm does find a minimal cardinality set in many instances.

A. Convex Cover Algorithm

Given our strategy of stochastically choosing a hidden guard set, there will be a neighborhood around each guard that only that particular guard can see (since a small perturbation on a guard is not likely to make it visible to the others). Likewise, provided an adequate number of samples, there will be a set of sample points that only one particular guard can see. We call this set of points, the *kernel* of the guard, and denote it as

$$Ker\left(g_i\right) = Vis\left(g_i\right) \setminus \bigcup_{j \neq i} Vis\left(g_j\right).$$

In any minimum convex cover C, each convex region C_i has a set of points only contained in that particular region. Otherwise, region C_i could simply be removed and the cover would not have been minimal. Although we know there is not an exact equivalence, we use guard kernels as an approximation to these unique subsets. Thus, the main idea behind our convex cover algorithm, is that by "growing" convex regions around the guard kernels, we can generate a low cardinality convex cover.

The following description of our algorithm is depicted in Fig. 2. In the figure, guards are circles and samples small squares. The current guard, as well as its kernel are highlighted.

(1) Choose the guard g with the largest kernel (Fig. 2 (a)), $g = \text{argmax}_j \left\{\left|Ker\left(g_j\right)\right|\right\}$ and compute the convex hull $Q_O = Conv\left(Ker\left(g\right)\right)$ of the kernel. For this, we start with the guard and add kernel samples to the current convex hull using the iterative algorithm described in [7]. Each kernel point adds new edges to the current hull, and we check to make sure that they do not collide with facets of the environment.

(2) At some point, the convex hull may collide with the obstacle region (Fig. 2 (b)), that is $Q_0 \bigcap \overline{W} \neq$

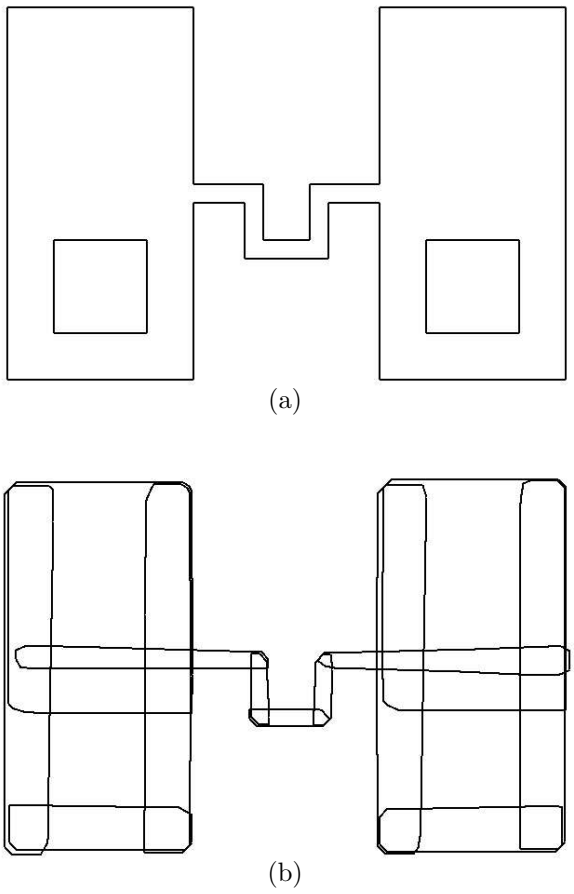
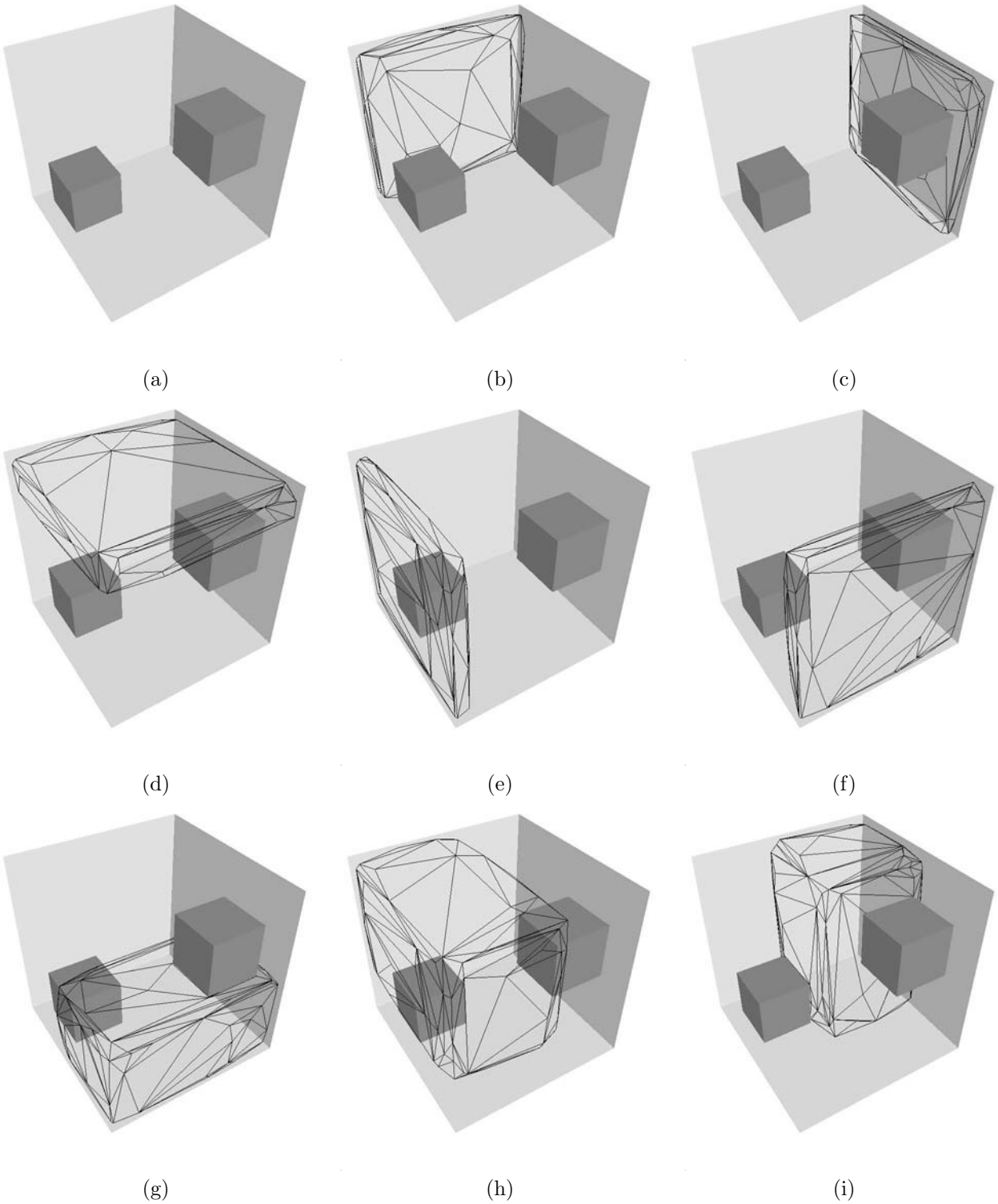


Fig. 3. A polygon (a) and the covering convex regions generated (b)



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Fig. 4. An environment (a) and the eight covering convex regions generated (b) – (i)

ϕ. In this case, we move the guard g to a random location inside the kernel but outside the current hull (Fig. 2 (c)). This does not violate the hidden property because the kernel was only visible to this guard. Then, we re-process the rest of the samples with the algorithm described in section II-A. After this, at least one new guard will be generated and added to G (Fig. 2 (d)) because the original guard g can no longer see all the previous kernel points (since it was moved outside a growing convex region). This process tries to generate a maximum cardinality hidden guard set in situations where a single guard covers both sides of a convex corner. In this case, the guard is moved to one side of the corner and a new one generated on the other side. We then go to (1) and start over.

(3) If the whole kernel can be contained in a single convex region that does not intersect the environment boundary (Fig. 2 (e)), $Q_0 \bigcap \overline{W} = \phi$, we continue to "grow" this region adding sample points in S as long as doing so does not generate a collision with the obstacle region (Fig. 2 (f)). That is, we generate the convex hull $Q_{i+1} = Conv\,(Q_i \bigcup \{s\})$ for some $s \in S$, but only if $Q_{i+1} \bigcap \overline{W} = \phi$. When the convex region has reached its maximum size, we remove the original guard g from G. If all the guards have been processed, terminate, otherwise continue at (1).

This algorithm scales well with the number of dimensions of the environment, as long as a suitable convex hull algorithm and a segment-facet intersection test are implemented. Therefore, it is possible to use the same algorithm to generate a convex cover in different dimensions. To illustrate this point, Fig. 3 shows the results of our algorithm when applied to a polygon (resulting regions were drawn with an offset for clarity) and Fig. 4 when applied to a 3-D environment (here, one of the boxes is on the ground while the other is "floating").

B. Second Approach for Generating Trajectories

As long as the dimensionality of the internal obstacles matches that of the environment, our proposed approach will decompose the workspace into overlapping convex regions. These regions can then be transformed into a graph that captures the connectivity of the workspace. The process is as follows: For every pair of regions, place a new sensing location (node) at the center of their intersection (if it exists). Then, join all the intersection centers within each region with straight line segments. This process is depicted in Fig. 5.

The locations on the resulting graph have the property of covering the whole environment (since there is at least one in every region). To each of these locations, we assign a reduced visibility region – just the two generating convex regions, and we estimate their size using the samples that fall inside of them. With this, it is possible to use the approach we proposed in [9] to search the graph and generate efficient trajectories that reduce the expected value of the time to find an object in the environment.

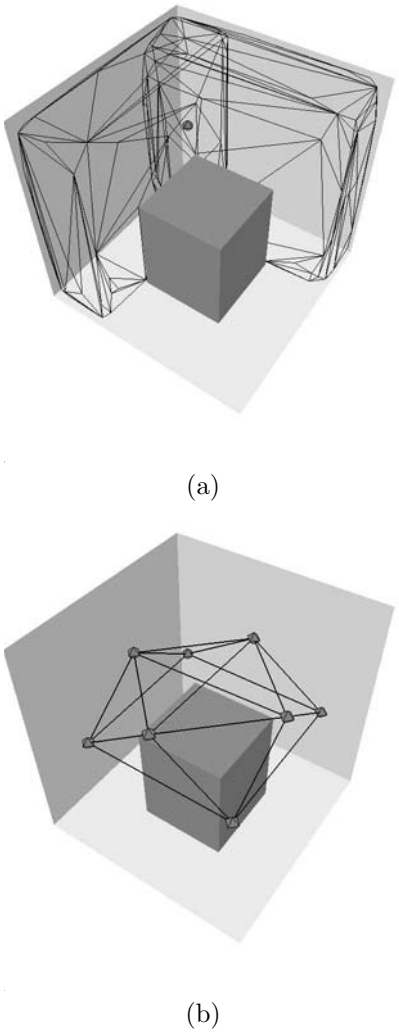
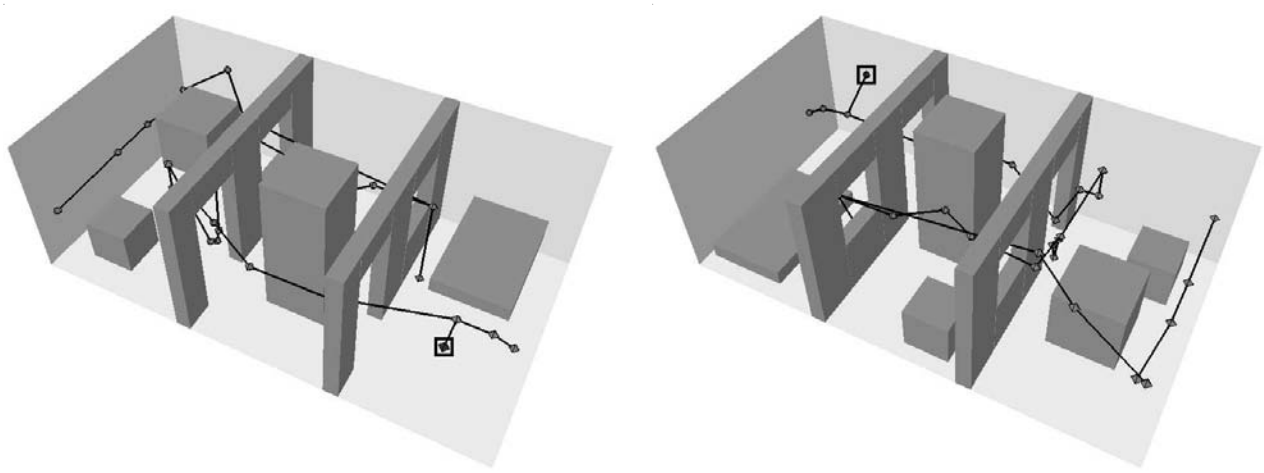


Fig. 5. The intersection center of two convex regions (a) and the graph connecting all the intersection centers (b)

IV. Simulation Results

We applied the convex cover algorithm described in section III and our previous approach to search a graph connecting sensing locations to the synthetic environment shown in Fig. 6. The environment has three rooms and several obstacles (some of which are floating). Fig. 6 also shows two views of the same trajectory with the starting position highlighted with a small square.

The trajectory first goes a bit to the left, as seen in



(a) (b)

Fig. 6. Two views of an environment with three rooms and the final covering path

Fig. 6 (a), then goes through the two doors to the last room, then back to the first room through the windows and finally returns to the third room to cover the last portion. Bear in mind that this route is *not* trying to minimize the distance traveled, and that the back and forth movements are actually good for minimizing the expected value of the time to find an object. In this case, what is required is to sense the largest portions of the environment as quickly as possible, even if it means investing more time later to finish sensing the environment [8].

V. Discussion and Future Work

We proposed a sample-based convex cover algorithm for rapidly finding an object in a 3-D environment. Our algorithm tries to reduce the cardinality of the resulting set and has the advantage of scaling well with the dimensionality of the environment.

Since our method explicitly computes and records the intersection of overlapping convex regions, it captures the connectivity of the workspace. We believe this can also be exploited in path planning. We intend to use this method to bias a path planner to those regions known to be connected in the workspace, since it is obvious that disconnected regions in the workspace will also be disconnected in the configuration space. We think this might be useful in locating narrow passages since they are not as narrow in the workspace (a robot with volume must be able to pass through).

Another extension to the proposed problem is to consider a volumetric robot. We have already followed that line of research and the results are in [10]. We addressed the object search problem – in simulation – using a mobile manipulator with a sensor on its end effector. We were specially interested in the problem of finding trajectories that compromise between moving the base and moving the robotic arm. These trajectories are useful in cases where, for example, the robot must sense behind an obstacle and must decided whether to go around it or extend the arm over it. For this, the convex decomposition was particularly helpful since it gave us flexibility on exactly where to place the end effector.

References

[1] Evans, M. and T. Swartz, *Approximating Integrals via Monte Carlo and Deterministic Methods,* Oxford University Press, 2000.

[2] In *Handbook of Discrete and Computational Geometry,* J. E. Goodman and J. O'Rourke, editors, CRC Press, 1997.

[3] Hsu, D., J. C. Latombe and R. Motwani, "Path Planning in Expansive Configuration Spaces," *in Proc. IEEE Int. Conf. on Robotics and Automation 1997.*

[4] Kavraki, L. E., P. Svestka, J. C. Latombe and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation,* Vol. 12, Num. 4, Jun 1996.

[5] Leven, P. and S. Hutchinson, "Real-Time Path Planning in Changing Environments," *Int. Journal of Robotics Research,* Vol. 21, Num. 12, Dec 2002.

[6] O'Rourke, J., "The Complexity of Computing Minimum Convex Covers for Polygons," *in Proc. 20th Annual Allerton Conf. on Communication, Control, and Computing 1982.*

[7] O'Rourke, J., *Computational Geometry in C,* Cambridge University Press, 1994.

[8] Sarmiento, A., R. Murrieta-Cid and S. Hutchinson, "A Strategy for Searching an Object with a Mobile Robot," *in Proc. Int. Conf. on Advanced Robotics 2003.*

[9] Sarmiento, A., R. Murrieta-Cid and S. Hutchinson, "An Efficient Strategy for Rapidly Finding an Object in a Polygonal World," *in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems 2003.*

[10] Sarmiento, A., "Generating Expected-Time Efficient Trajectories for Rapidly Finding an Object in Known Environments," Dept. of Computer Science Report No. 2491, U. of Illinois at Urbana-Champaign, Dec. 2004.

[11] Simeon, T., J. P. Laumond and C. Nissoux, "Visibility Based Probabilistic Roadmaps," *Advanced Robotics Journal,* Vol. 14, Num. 6, 2000.