ISSN 2007-9737

Saving Time for Object Finding with a Mobile Manipulator Robot in 3D Environment

Judith Espinoza and Rafael Murrieta-Cid

Centro de Investigación en Matemáticas, CIMAT, Guanajuato, Mexico

jespinoza@cimat.mx, murrieta@cimat.mx

Abstract. In this paper, we address the problem of reducing the time for finding an object. We consider both the time taken by our software to generate a search plan and the expected time to find the object when the plan is executed. The object is sought with a 7 degree of freedom mobile manipulator robot with an "eye-in-hand" sensor. The sensor is limited in both range and field of view. We propose two main strategies: 1) to coordinate the motion of robot's degrees of freedom optimizing only those most relevant for the task, and 2) to repair a previously computed plan whenever the environment changes locally. We have implemented all our algorithms and present simulation results in realistic environments.

Keywords. Search, path planning, 3D visibility, 3D coverage.

1 Introduction

In this paper, we address the problem of reducing the time for finding an object. The object is sought with a 7 degree of freedom mobile manipulator robot with an "eye-in-hand" sensor. The sensor is limited in both range and field of view. A search plan corresponds mainly to a set of sensing configurations to be visited and an order for visiting those configurations.

In [20], the authors investigated the problem of finding an object in a 3D environment *for the case of a point robot equipped with an omnidirectional sensor*. In [20], the authors have also introduced a probabilistic sampling method to decompose the workspace into convex regions. In [3, 5] the authors extended their work to a mobile manipulator robot equipped with a sensor *limited in both field of view and range*; a method to approximate the visibility region in 3D of the limited sensor is proposed, convex regions are used to facilitate this approximation.

We have found that the main disadvantage of the approach presented in [5] is that for large and complex 3D environments, the time needed to compute a plan might be excessively large, in the order of several hours. We have found that the task taking the longest computational running time corresponds to optimizing the motion of a large number of robot's degrees of freedom (DOF) for generating paths that minimize the expected value of the time to find an object. The computational running time of our algorithm refers to the time taken by our software to generate a plan to find the object. The expected value of the time refers to the average time in which the object will be found by executing that plan. So the first time refers to the generation of the plan, and the second one to the performance in average of this plan when the plan is executed.

In this paper, we propose two approaches for reducing the computational running time to generate a plan while preserving the expected value of the time to find the object.

First, we propose a strategy of selecting the most important DOF to be optimized. This strategy significantly reduces the computational running time to generate a plan. Furthermore, our motion planner coordinates the translation of the robot base and the rotations of the arm's links such that the expected value of the time remains almost the same compared with the case of optimizing all robot's DOF.

Second, we propose a method to repair a previously computed plan, for dealing with local changes in a 3D environment. The algorithm

presented in [5] receives as input a 3D map of the environment and returns as output a search plan. Once the plan is generated, whenever the environment changes, a new plan should be computed. In this paper, we present a technique that allows us to avoid the generation of a new plan from the beginning. For example, imagine the following scenario: a plan has been generated for finding some object inside a house, but after the generation of the plan, some furniture inside the house has changed its location, however, the house building has not changed. In this kind of situations, the method proposed in this paper is applicable and useful. We base our approach on a 3D convex region decomposition, in which the environment is divided. The plan is repaired by generating a new subset of sensing configurations and a new order for visiting those configurations, considering only the convex regions related to the change in the map of the 3D environment. An important advantage of repairing a plan instead of generating the whole plan again is that the time needed to repair the plan is in general significantly smaller than the time needed to generate the whole plan again. Moreover, in our experiments, we have found that this technique does not considerably or systematically increase the expected value of the time for finding the object.

The main distinguishing features of this work compared with the research presented in [4] are as follows. 1) We propose a method to coordinate the motion of robot's DOF optimizing only those most relevant for the task. 2) We present a comparison between optimizing all the robot's DOF as in [5] versus optimizing an appropriate subset of DOF, in terms of both the computational running time needed to generate a plan and the expected value of the time to find an object. 3) For the strategy of repairing plans, we present cases in which the percentage of the modified environment varies and we also compare the results.

Before proceeding to the description of the proposed approach, several points are important to keep in mind:

— The task that we are addressing requires to deal with problems of high computational complexity: e.g., generation of a minimum convex cover [15], determination of an order to visit sensing locations that minimizes the expected value of the time for finding the object [21], and computation of shortest paths for a robot with 7 DOF [11, 13].

— We deal with geometric aspects often neglected in order to find an object: e.g., 3D visibility computation, a robot with no trivial geometry, and a sensor limited in both range and field of view.

— We propose an approach that requires only a few minutes to generate a search plan, which diminishes the expected value of the time to find the object when the plan is executed. We also propose an approach to repair the plan for a local change of the environment.

— Any computer vision algorithm used to detect the object sought needs first to have that object inside the robot's field of view. It is this issue that our paper focuses on, that is, the task is to put the object within the robot's field of view.

The rest of this paper is organized as follows. In Section 2, we present related works. In Section 3, we briefly describe a general method for a plan generation. In Section 4, we present an approach for coordinating the motions of robot's DOF and for optimizing only those most relevant for the task. A comparison of the results in terms of both the computational running time needed to generate a plan and the expected value of the time to find an object is also presented. In Section 5, we present a method to repair a previously computed plan. In Section 6, results for repairing a plan are presented. Finally, in Section 7, we present conclusions and future work.

2 Related Work

Our search problem is related to robot motion planning, coverage, and art gallery problems.

In robot motion planning [11, 13], a typical goal is to find a collision-free path to move a robot (a mechanical system, which may have many degrees of freedom) from an initial to a final configuration.

Efficient algorithms have been proposed to solve this problem. These algorithms use sampling to capture connectivity of high dimensional configuration or state spaces, for example, [9, 8, 14, 23], just to name some classical works. In our work we also want to connect sensing configurations, but we have an additional goal. *We are interested in representing the free space inside the 3D workspace for searching for an object* and not only in representing the configuration space for avoiding robot collision. Nevertheless, we need to find collision-free paths to move the robot between sensing configurations. Our main interest is to address the problem of finding a static object. *This adds a new aspect to our planning problem*.

In coverage problems (e.g., [7, 1]), 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 strategy guaranteeing to find an object must sense the whole environment.

The traditional art gallery problem is to find a minimal placement of guards such that their respective visibility regions completely cover a polygon [16, 22, 6]. As we will see below in Section 3, a set of sensing configurations that collectively see the environment could be used as part of a solution to our search problem. Notice that in contrast to the works presented in [16, 22] and [6] we consider a 3D environment and not a polygon.

In [24], the authors propose an approach for adaptive motion planning of robots with many degrees of freedom such as mobile manipulators in dynamic environments with moving obstacles. In [10], an approach is proposed that enables a robotic system to react to unforeseen and unpredictable events. In particular, a method is proposed to switch from sensor-guided motions to trajectory-following motions. However, in [24] and [10] the authors do not propose a method for approximating 3D visibility computation and search for an object, so our paper addresses these problems.

There has been a considerable amount of research on search problems in robotics. Several authors have proposed methods for looking for one or several objects with robots, e.g., [2, 12, 17]. In [12], a method is proposed for looking for multiple objects with several robots; the method is based on dynamic programming, and the authors' objective is to minimize the expected time to find the objects. The proposed method addressed only the case of 2D environments and robots have simple geometry. In [2], the authors describe an interesting decentralized Bayesian approach to coordinating multiple autonomous sensor platforms searching for a single non-evading object. The approach is applied to a team of airborne search vehicles looking for a stationary target lost at sea. However, the proposed approach does not deal at all with obstacles in the environment that produce motion and visibility obstructions. In [17], an approach for visual search of a given target is proposed. This approach optimizes the probability of finding the target given a fixed cost limit in terms of a total number of robotic actions the robot requires to find a visual target. A robotic realization in a 3D environment is presented. However, a collision-free path for the robot to move is calculated considering only a 2D environment, and the authors propose as future work a more advanced path planning capability.

3 Original Plan Generation

In our problem formulation, we assume that the environment is known, but we do not have information about the location of the static object being searched. This is equivalent to defining a uniform probability density function (pdf) modeling the object location. We believe that this reasoning is general given that we do not need to assume a relation between a particular type (class) of object and its possible location (for instance, balloons are floating but shoes lie on the ground), which could reduce the scope of the applications.

The robot senses the environment at discrete configurations q_i (also known as *guards*, from the art gallery problem [16]). Let's call $V(q_i)$ the visibility region associated to the limited sensor. Our searching strategy is as follows. First, the whole environment is divided into a set of convex regions. To split the environment into convex regions we use the probabilistic convex cover proposed in [20].

That method divides the environment into a set called $\{C_r\}$, so that the union of all C_r covers the whole environment, that is, $\bigcup_r C_r = int(W)$. The interior of the workspace $int(W)$ is the free space inside a 3D environment, C_r denotes a convex region in the 3D environment, and r indexes the region label. Note that all points inside C_r can be connected by a clear line of sight from any point $p(x, y, z)$ inside C_r. Second, each convex region is covered with the sensor frustum denoted by $\mathcal{F}$.

We establish a route to cover the whole environment by decomposing the problem into two parts: first, an order to visit convex regions C_r is established, second, sensing configurations in a configuration space $\mathcal{C}$ of 7 dimensions are generated to collectively cover each convex region. These sensing configurations are linked in a graph and perform a graph search to establish the order to visit the configurations associated to a single convex region.

In [21] it has been shown that the problem of determining the global order for visiting sensing locations which minimizes the expected value of the time to find an object is NP-hard, even in a 2D polygonal workspace with a point robot. Hence, in [21], an efficient algorithm has been proposed, which aims just to diminish the expected value of the time. In this paper, we use that algorithm to establish the orders for visiting convex regions and for visiting sensing configurations inside a single convex region. Below, we briefly describe the main concepts applied in the algorithm proposed in [21].

The route followed by the robot corresponds to an order of visiting sensing configurations $q_{i,k}$ that starts with the robot's initial configuration and includes every other configuration. While q_i refers to a configuration, $q_{i,k}$ refers to the *order* in which configurations are visited. That is, the robot always starts at $q_{i,0}$, and the k^{th} configuration that the robot visits is referred to as $q_{i,k}$.

For any route R, we define the time to find the object T as the time it takes to go through the configurations – in a given order – until the object is first seen. The expected value of the time to find an object depends on two main factors: 1) the *cost* of moving the robot between two configurations, which is the elapsed time, and 2) the probability mass of seeing the object, which is equivalent to the *gain*.

The expected value of the time that a route takes to find the object is defined as follows:

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

where

$$P\left(T = t_j\right) = \frac{Volume\left(V(q_{i,j}) \setminus \bigcup_{k<j} V(q_{i,k})\right)}{Volume(int(W))}. \tag{2}$$

Here, t_j is the time it takes the robot to go from its initial configuration – through all sensing configurations along the route – until it reaches the j^{th} *visited* configuration $q_{i,j}$, i refers to the label (identifier) of the configuration. Since the robot only senses at specific configurations, $P\left(T = t_j\right)$ is the probability of seeing the object for the first time from configuration $q_{i,j}$. The probability of seeing the object for the first time from configuration $q_{i,j}$ is proportional to the volume visible from $q_{i,j}$ minus the volume already seen from configurations $q_{i,k}$, $\forall k < j$ as stated in Eq. 2.

We use the utility function defined below to measure how convenient it is to visit a determined configuration from another:

$$U\left(q_k, q_j\right) = \frac{P\left(q_j\right)}{Time\left(q_k, q_j\right)}. \tag{3}$$

This means that if a robot is currently in q_k, the utility of going to configuration q_j is directly 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 configurations that are close and/or configurations where the probability of seeing the object is high. $P\left(q_j\right)$ is equal to $P\left(T = t_j\right)$ defined in Eq. 2.

The utility function in Eq. 3 is sufficient to define a 1-step greedy algorithm. At each step, simply evaluate the utility function for all available configurations and choose the one with the highest value. This algorithm has a running time of $O\left(n^2\right)$, for n configurations.

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. So, we use this utility function to drive a partially greedy algorithm. This algorithm is able to explore several steps ahead without incurring a too high computational cost. In the worst case, this algorithm has a running time complexity of $O\left(n^3 \log n\right)$. A description of this algorithm can be found in [21], together with a comparison between the performance of the algorithm (in terms of the expected value of the time to find the object) vs. the optimal path. We emphasize that this algorithm often reduces in 3 orders of magnitude the computational running time compared with the algorithm needed to find the optimal solution, which is exponential since the optimization problem to be solved is NP-hard.

3.1 Paths to Move between Convex Regions

Since the expected value of the time depends on the cost (time) of moving the robot between sensing configurations, we need to find shortest paths to move the robot between convex regions. The actual paths depend on the metric used to measure the cost to move between convex regions. One way to define the cost between two configurations X and Y in a D-dimensional configuration space is

$$\|X - Y\|_\Lambda \equiv (X - Y)^T \Lambda (X - Y), \qquad (4)$$

where Λ is a diagonal matrix with positive weights $\lambda_1, \lambda_2, \ldots \lambda_D$ assigned to the different DOF. By weighting each DOF differently, we can assign different priorities to the two main components of our system: the mobile base and the robotic arm.

To find the shortest path between one given convex region and all the others, we use the wavefront expansion (called NF1) proposed in [11]. We apply the method proposed in [11] to compute the shortest path for the mobile robotic base, the degrees of freedom related to the robotic arm are planned using a sampling method, such that the robot does not collide with the obstacle. Optimizing only a subset of all robot's degrees of freedom greatly reduces the computational running time to generate a global path to explore the environment (see Section 4). Furthermore, we also coordinate the motion of the robot's base and the robot's arm, such that the expected value of the time to find an object does not increase even if only some DOF are optimized (see also Section 4).

3.2 Selecting and Connecting Sensing Configurations Inside a Single Convex Region

The method proposed in [3, 5] to cover each convex region with a limited sensor is based on sampling.

Sensing configurations $q_{(i,r)}$ are generated with a uniform probability distribution in a configuration space $\mathcal{C}$ of 7 dimensions: a sensing configuration $q_{(i,r)}$ is associated to a given region C_r. Each convex region has an associated set S_r of point samples $s_r \in S_r$. Each point sample s_r lies in the 3D space and is defined by a 3-dimensional vector $p(x, y, z)$. S_r is used to cover the convex region C_r with a limited sensor.

The algorithm for selecting sensing configurations has been inspired from the algorithm presented in [6], the latter was designed to cover the boundary ∂P of a polygon P. We have extended this method to cover the interior of the polyhedral representation of a 3D environment $int(W)$.

In the proposed method, the point samples lying inside the frustum associated to a sensing configuration $q_{(i,r)}$ are used to approximate the actual visibility region $V(q_{(i,r)})$. The robot's configurations used to cover a convex region have the property that all of them place the sensor inside the convex region being sensed. This property allows us to approximate the visibility region of the limited sensor without complex 3D visibility computations. The visibility region of the limited sensor at configuration $q_{(i,r)}$ is approximated by

$$V(q_{(i,r)}) = \bigcup_s s_r \in int(\mathcal{F} \cap C_r), \qquad (5)$$

where s indicates sample points.

While covering the region C_r, we also mark as sensed and logically remove all samples s_v belonging to the region $C_v, v \neq r$, if $s_v \in int(\mathcal{F} \cap C_r \cap C_v)$. It is guaranteed that these samples are not occluded from configuration $q_{(i,r)}$. In Figure 1, dark (magenta) dots are used to show the set S_v, and

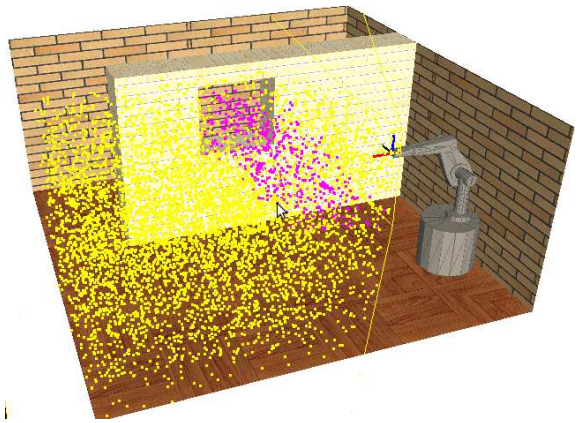


Fig. 1. Sets $\{s_r\}$ and $\{s_v)\}$

light gray (yellow) dots represent the set of point samples S_r belonging to the region in which the sensor resides and inside the frustum. A convex region C_r is totally covered if

$$\bigcup_s s_r \in int(C_r) = S_r. \tag{6}$$

Sensing configurations are selected based on the cardinality of its point samples. Iteratively, we select the configurations with the largest cardinality of point samples s_r until all the set S_r is sensed. Redundant sensing configurations with low point samples' cardinality are avoided, yielding a reduced set containing only sensing configurations with high cardinality of point samples and a small number of redundant point samples.

Additionally, in our sensing configuration sampling scheme, we reject candidate sensing configurations in whose view the frustum is in collision with the robot itself, thus avoiding occlusions generated by the robot body. We also reject sensing configurations that produce a collision of the robot with the obstacles and robot self-collisions.

Since we want to have options to move the robot between sensing configurations and thus further reduce the expected value of the time to find the object, we connect the sensing configurations of each set $\{q_{(i,r)}\}$ into a fully connected graph. For reducing the computational time to cover the environment with a limited sensor, we *estimate* the cost to move between sensing configurations as a straight line in the configuration space $\mathcal{C}$.

In the motion planning problem of generating collision-free paths to move between configurations, we use a lazy collision checking scheme [18, 19]. Since we proceed visiting convex regions one by one, it is likely to find collision free paths among configurations to cover the same convex region. Often a small region can be covered with small robot motions, and big regions offer a large open space to move the robot. We postpone the collision checking until an order of sensing configurations is established. Evidently, sometimes the fully connected graph splits into two connected components; if so, we use an RRT [14] to find a collision-free path between the two components. We point out that in our experiments we have found that only $\frac{1}{10}$ of the total number of paths to sense convex regions are computed with an RRT. All other times, a straight line in $\mathcal{C}$ was enough to find collision-free paths.

To cover a single convex region, the robot travels a tour, so the first sensing configuration and the last one are the same; this allows preserving the path and its cost of moving between convex regions, and consequently, the order to visit them which has been previously planned.

4 Coordinating the Motion of Degrees of Freedom

The coordination of the DOF motion is general in the sense that it is possible to coordinate the motion of the different degrees of freedom optimizing only a subset of them. However, the determination of the cost of moving each degree of freedom depends on the specifics of the application. In large environments and having as a goal to reduce the expected value of the time to find an object, it is typically more convenient to optimize the motion of the robot base. Consequently, we consider that the coordinates (x, y) defining the position of the robot's base are the most important DOF for our problem. Hence, we optimize only these two DOF. The shortest path between a given convex region and all the others is found using the wavefront expansion (called NF1) proposed in [11]. We plan the

motion of the other DOF using a sampling procedure. Thus, a robot path to move between convex regions is a sequence of robot's configurations, in which some DOF are planned optimally and the others do not produce collisions between the robot and the obstacles.

Furthermore, our motion planner coordinates the translation of the robot base and the rotations of base and the arm's links, such that both translation and rotations happen at once. If the maximal rotational speed for the DOF of the robot arm is large enough, the arm can move to its destination within the time that the base moves. In this specific case the cost of moving the arm is zero. In other words, when the robot finishes a translation motion, the degrees of freedom of the robotic arm are already in their desired final configuration. The translation time is divided in several intervals, each of them has a time stamp such that there is a limited number of iterations for the DOF of the robotic arm to reach their desired final value.

Finally, it is important to clarify that the trajectory of the robotic base might be different when the motion of all the DOF are computed using the wavefront expansion, compared with the trajectory in which the motions of the robotic arm DOF are computed based on a sampling procedure. Our sampling procedure might not find a collision-free configuration needed to obtain the true shortest path for the base (for a given resolution of the wavefront). Consequently, the expect value of the time to find the object will also be different. In our experiments (see next section), we have found that using our coordination approach, the expected value of the time to find the object does not increase considerably, while the computational running time to generate a plan is drastically reduced from several hours to a few minutes.

4.1 Simulation Results

All the results presented in this paper were obtained with a regular PC running Linux OS, the processing speed of the CPU is 2.2 GHz. The programming language used to obtain our simulation results was C++.

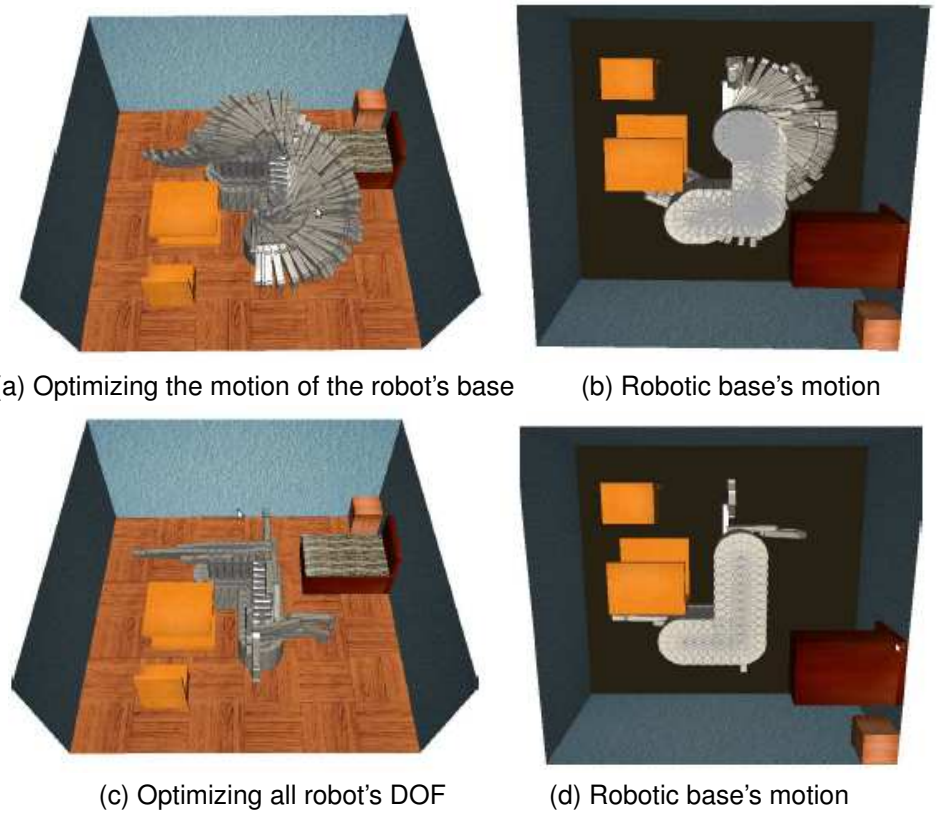
In Figures 2 a) and b) an example of a global path to visit convex regions is shown, in this example only the DOF related to the position of the robotic base are optimized. In Figures 2 c) and d) another example is shown. In this case all the DOF of the robot are computed using a wavefront expansion. Figures 2 b) and d) show a view taken from "under the ground" in order to better see the motion of the robot's base. Notice that the paths of the robot's base are a bit different.

In Figures 3 and 4 a more complex environment is shown. This environment was divided in 23 convex regions. Figures 3 a) and b) show some of these convex regions with a (red) mesh. Observe, for instance, the convex regions under the chair and the table. In this environment, we have run 20 different simulations, in 10 of them we have optimized only the motion of the robot's base and in the other 10 we have optimized all robot's DOF. In Table 1, we present the mean of both the computational running time needed to compute the paths to visit all the convex regions and the mean of the expected value of the time to find the object E[t]. These results clearly show that using our coordination approach, the expected value of the time to find the object does not increase considerably, while the computational running time to generate a plan is drastically reduced. In these experiments, the expected value of the time has increased only by 13%, while the computational running time to compute the global path was reduced almost 60 times.

Figure 4 a) shows an example of the global path to visit all convex regions, in which only the motion of the robot's base is optimized. Figure 4 b) shows an example of the global path to visit all convex regions, in which all robot's DOF are optimized.

5 Repairing a Plan

In [20], the authors have proposed an algorithm for a convex cover. That algorithm is based on sampling and divides the environment into overlapping convex regions. Roughly, the algorithm works as follows: first, to capture the size and shape of the workspace W one generates a set of independent, uniformly distributed samples S in the interior of W. Among these samples, one chooses a hidden



(a) Optimizing the motion of the robot's base (b) Robotic base's motion

(c) Optimizing all robot's DOF (d) Robotic base's motion

Fig. 2. Optimizing some degrees of freedom

Table 1. Statistics of experiment shown in Figure 4

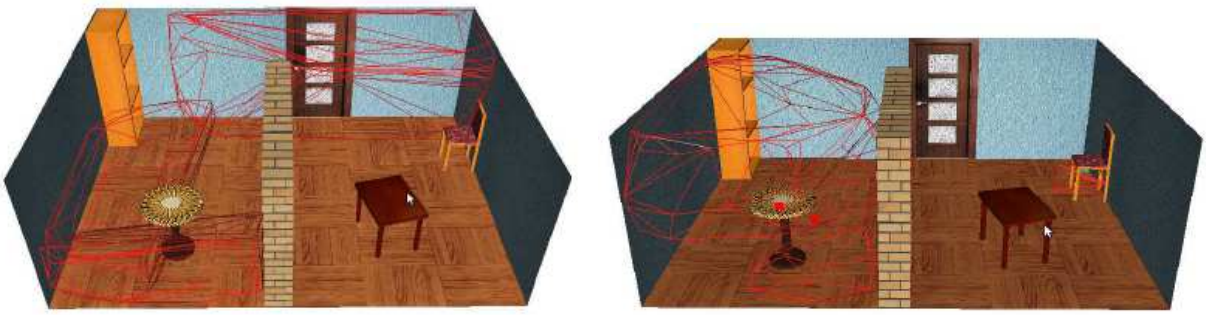
Number of DOF	Number of convex regions	Computational running time	E[t]
2 DOF	23	11 min 40 sec.	41.20 units
7 DOF	23	10 hrs 51 min.	36.43 units

guard set G. A set is called a hidden guard set if it covers the environment and individual members of the set are not visible to each other. There will be a set of sample points that only one particular guard can see. This set of points is called the kernel of the guard $g \in G$. Second, guard kernels are divided into convex regions by using convex hulls. The resulting convex regions are expanded by adding sample points as long as doing so does not generate a collision with the obstacles. The main idea behind this convex cover algorithm is that by "growing" convex regions around the guard kernels, one can generate a low cardinality convex cover (a detailed description of this algorithm can be found in [20]). Note that, it has been proved that a minimum convex cover even in a polygon is also an NP-hard problem [15]; therefore, the aim is just to obtain an efficient algorithm that tries to generate as few convex regions as possible. It has been found that in practice this algorithm does find a minimal cardinality set in some instances [5].

5.1 Modifying the Convex Cover to Deal with Changes in the Environment

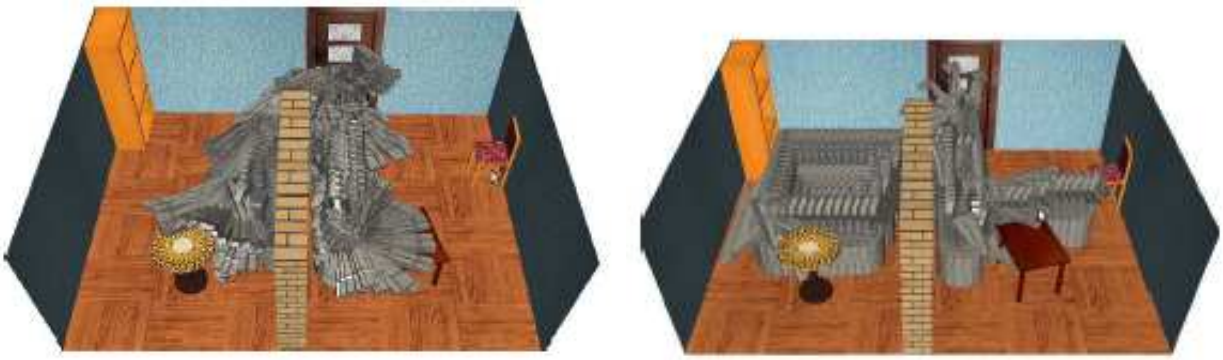
Changes in the environment are detected using the original convex cover. Indeed, a change of loca-

ISSN 2007-9737



(a) (b)

Fig. 3. Some of the convex regions



(a) Optimizing the motion of the robot's base (b) Optimizing all robot's DOF

Fig. 4. A more complex environment

tion of an obstacle in the environment will produce the following modifications over the convex regions originally generated. 1) The regions related to the original location of the obstacle must be modified, and 2) the regions related to the new position of the obstacle must also be modified. Let us call the first set of regions $\{C\}_t$ and the second set, $\{C\}_{t+1}$.

Note that we use the word "obstacle" to refer to an element of the environment which is moved to modify the original environment. This element, the so-called obstacle, is not the sought object. The sought object location is not known deterministically.

To define which regions are members of $\{C\}_t$, it is necessary to detect the regions which are adjacent to the original obstacle position. A way of detecting these regions is by measuring the distance between the convex regions and the obstacle. All regions which are closer than a given small ϵ to the obstacle are members of $\{C\}_t$. The set of regions $\{C\}_t$ has an associated set of point samples called S_t. The point samples in the interior of the union of all the regions in $\{C\}_t$ form the set S_t, that is, $S_t = \bigcup_s s \in int(\{C\}_t)$.

To define which regions are members of $\{C\}_{t+1}$ is simple. Merely all original convex regions are tested for collision with the obstacle at its new location, those regions in collision belong to $\{C\}_{t+1}$. The set of regions $\{C\}_{t+1}$ has also an associated set of point samples called S_{t+1} which is defined

by $S_{t+1} = \bigcup_s s \in int(\{C\}_{t+1})$ It is also needed to determine the point samples in collision with the original and new obstacle locations. Let's call the set of point samples in collision with the obstacle at its original location Obs_t, and the set of samples in collision with the obstacle at its new location, Obs_{t+1}.

The key idea to compute the new convex regions needed to take into account the change of the map is the following: only a subset of the samples used to compute the original convex cover are given as input to the algorithm that generates the local convex cover decomposition considering the change in the map. Let's call this subset S_Δ, S_Δ is the union of the point samples belonging to the sets S_t, S_{t+1}, and Obs_t minus the samples belonging to the set Obs_{t+1}, that is,

$$S_\Delta = \bigcup_s s \in \Big(S_t \bigcup S_{t+1} \bigcup Obs_t\Big) \setminus Obs_{t+1}. \quad (7)$$

The polyhedral representation of the environment considering the new obstacle location and the set of point samples S_Δ are given as inputs to the convex cover algorithm proposed in [20]. The algorithm returns as outputs a new set of convex regions needed to take into account the change in the environment. This set is denoted $\{C\}_\Delta$.

Let's call $\{F\}$ the set of all the original convex regions, and $\{N\}$ the set of all convex regions after having modified the environment. This new set of convex regions is composed by the original regions that have not been eliminated (i.e. $\{F\} \setminus (\{C\}_t \bigcup \{C\}_{t+1}$) plus the set $\{C\}_\Delta$, that is, $\{N\} = (\{F\} \setminus (\{C\}_t \bigcup \{C\}_{t+1})) \bigcup \{C\}_\Delta$; $\{N\}$ totally covers the modified environment.

5.2 Modifying the Orders to Visit Convex Regions and Sensing Configurations

As it was mentioned above, the problem of generating an order to visit sensing configurations was planned in two steps: first an order to visit convex regions is computed (we call it a global plan). Second, for every convex region an order to visit sensing configurations is established. We will show now that this heuristic of dividing a large problem into several smaller sub-problems facilitates to repair a previously computed plan.

To repair a previously computed search path and under the assumption that the change in the environment is local, it makes sense to modify the global plan only locally, preserving as much as possible the order to visit convex regions. This approach has the advantage that the computational running time to repair the plan is typically smaller than the one needed to recompute the whole global plan, adding reactivity to the re-planning process.

For repairing the plan, the spatial location of the new convex regions is taken into account. Furthermore, the new convex regions will appear at spatial locations related to the site occupied by the regions belonging to $\{C\}_t$ and $\{C\}_{t+1}$.

The goal of algorithm 1 is to compute a new order for visiting regions preserving as much as possible the original global plan. Let's call $\{O\}_F$ the ordered set of original convex regions, and $\{O\}_N$ the new ordered set of convex regions.

First, the set $\{F\}$ is ordered according to which region is visited earlier (by the robot) in the original global plan (line 1 of algorithm 1).

Regions $C_{r,j}$ index the elements of this set; r refers to the region's label (region's identifier), and j refers to the j^{th} visited region. Regions $C_v \in \{C\}_\Delta$ are the new generated regions, v refers to the region's label.

Regions $C_{r,k}$ index the elements of $\{O\}_N$ (k refers to the k^{th} visited region in the new set $\{O\}_N$).

Second, all regions $C_{r,j} \in (\{C\}_t \bigcup \{C\}_{t+1})$ are eliminated in the new plan.
Every region $C_{r,j} \in (\{C\}_t \bigcup \{C\}_{t+1})$ is checked for collision with every region $C_v \in \{C\}_\Delta$. Notice that more than one region C_v might intersect the same region $C_{r,j}$. All regions in $\{C\}_\Delta$, which intersect the same $C_{r,j}$ are stored in the set $\{C\}_{aux}$ (line 7 of algorithm 1). Let's call a_j the cardinality of the set $\{C\}_{aux}$, for each region $C_{r,j}$.

In algorithm 1 (line 10), the Local-Order method is used to establish the order of visiting regions in $\{C\}_{aux}$; $q_{robot} \in C_{r,k}$ denotes the first visited robot configuration in the convex region $C_{r,k}$. Local-Order does the following. Assuming that the robot is located at $q_{robot} \in C_{r,k}$, 1-step ahead evaluation of the utility function in Eq. 3 is used to establish

Algorithm 1 Computing the new order to visit convex regions

Input: Sets: $\{F\}$, $(\{C\}_t \bigcup \{C\}_{t+1})$, $\{C\}_\Delta$.
Output:$\{O\}_N$ new ordered set of convex regions
1. $\{O\}_F \leftarrow$ Order$\{F\}$;
2. $k = 1$, $e = a = 0$;
for $j = 1$ to $|\{O\}_F|$ **do**
  3. $C_{r,j} \leftarrow \{O\}_F$;
  if $C_{r,j} \in (\{C\}_t \bigcup \{C\}_{t+1})$ **then**
    4. $\{C\}_{aux} \leftarrow \emptyset$;
    5. $e = e + 1$;
    for $n = 1$ to $|\{C\}_\Delta|$ **do**
      6. $C_v \leftarrow \{C\}_\Delta$;
      if $(C_{r,j} \bigcap C_v \neq \emptyset)$ **then**
        7. $\{C\}_{aux} \leftarrow C_v$;
        8. $a = a + 1$;
      end if
    end for
    9.$\{C\}_\Delta \leftarrow \{C\}_\Delta \setminus \{C\}_{aux}$;
    10. Local-Order($\{C\}_{aux}$,$q_{robot} \in C_{r,k}$);
    11. $\{O\}_N \leftarrow \{C\}_{aux}$;
  else if $C_{r,j} \notin (\{C\}_t \bigcup \{C\}_{t+1})$ **then**
    if $e = 0$ **then**
      12. $\{O\}_N \leftarrow C_{r,j}$;
      13. $k = j$;
    else if $e \neq 0$ **then**
      14. $k = j + a - e$;
      15. $C_{r,k} = C_{r,j}$;
      16. $\{O\}_N \leftarrow C_{r,k}$;
    end if
  end if
end for

the region $C_{v,k+1}$. The region which maximizes Eq. 3 is chosen to be the $k+1$ region to be visited in the new order. Assuming now that the robot is located at $q_{robot} \in C_{v,k+1}$ and again using 1-step ahead utility function evaluation, the remaining $(a_j - 1)$ regions in $\{C\}_{aux}$ are evaluated to determine the visited region $C_{v,k+2} \in \{O\}_N$, and so forth until all a_j regions are ordered. For establishing an order for visiting regions in $\{C\}_{aux}$, the case in which the robot is already located in the region $C_{v,k} \in \{C\}_{aux}$ must be considered. This means that the cost to travel to this region is zero, and therefore there is no need to evaluate Eq. 3.

Third, the regions $C_{r,j} \notin (\{C\}_t \bigcup \{C\}_{t+1})$ are included in the new order with the following simple rules: if no region $C_{r,j}$ has been eliminated then $k = j$, and a region $C_{r,k}$ has the same place in the order as a region $C_{r,j}$ (line 12 of algorithm 1). If at least one region $C_{r,j}$ has been eliminated from the original plan then $k = j + a - e$, in which a is the number of new regions added to the new plan up to an ordered element j, and e is the number of regions which have been eliminated also up to j (lines 14, 15 and 16 of algorithm 1).

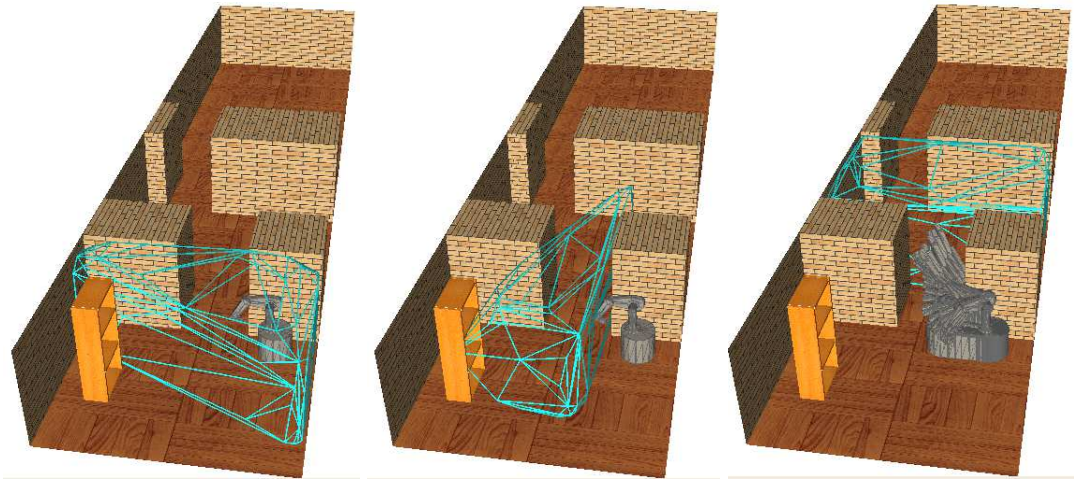
Note that each region $C_v \in \{C\}_\Delta$ is included only once in the new plan. The location of the first region $C_{r,j}$ which intersects the region C_v is taken into account to establish the order of the region C_v in the new plan. Once the region C_v is included in the new plan, it is eliminated from $\{C\}_\Delta$ (line 9 of algorithm 1). This procedure avoids redundancy in the new plan. Since regions $C_v \in \{C\}_\Delta$ replace regions $C_{r,j} \in (\{C\}_t \bigcup \{C\}_{t+1})$, there is no need of including a region C_v more than once. The new order is complete when all original $C_{r,j}$ regions have been considered.

The collision-free paths to move the robot between new convex regions in $\{C\}_\Delta$, and between a region in $\{F\}$ and a region in $\{C\}_\Delta$ are computed using the approach described in Section 4. Finally, new local plans are computed for visiting sensing configurations associated to every new region in $\{C\}_\Delta$. These local plans are computed with the approach described in Section 3.2.

Given that in our original global plan for covering the environment with the limited sensor, while covering region C_r, all samples s_p belonging to the region C_p ($p \neq r$) are marked and logically removed (whenever $s_p \in int(\mathcal{F} \cap C_r \cap C_p)$), then it is necessary to recompute new local plans for regions $C_{r,j} \notin (\{C\}_t \bigcup \{C\}_{t+1})$, which intersected the regions that have been eliminated and which were ordered ahead in the original order of visiting convex regions. Those new local plans are also computed with the approach described in Section 3.2.

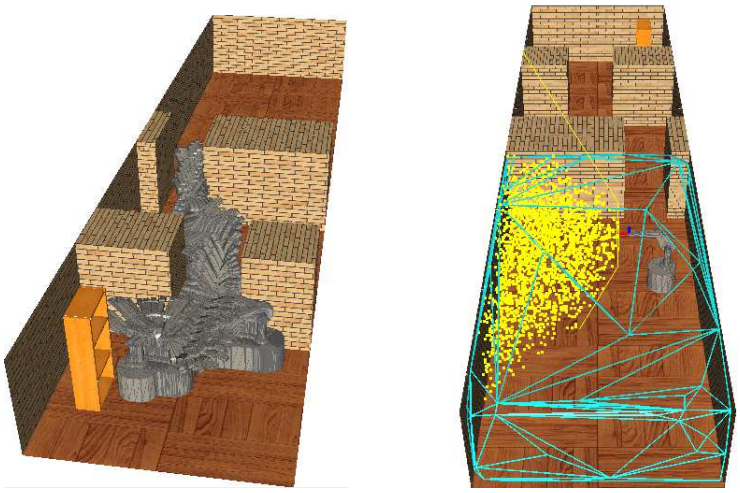
6 Simulation Results for Repairing a Plan

In this section we present two representative experiments, called 1 and 2, in which the portion of



(a) Region $C_{id9,1}$ (b Region $C_{id7,2}$ (c) Path $C_{id1,5} \rightarrow C_{id2,6}$

Fig. 5. Finding an object



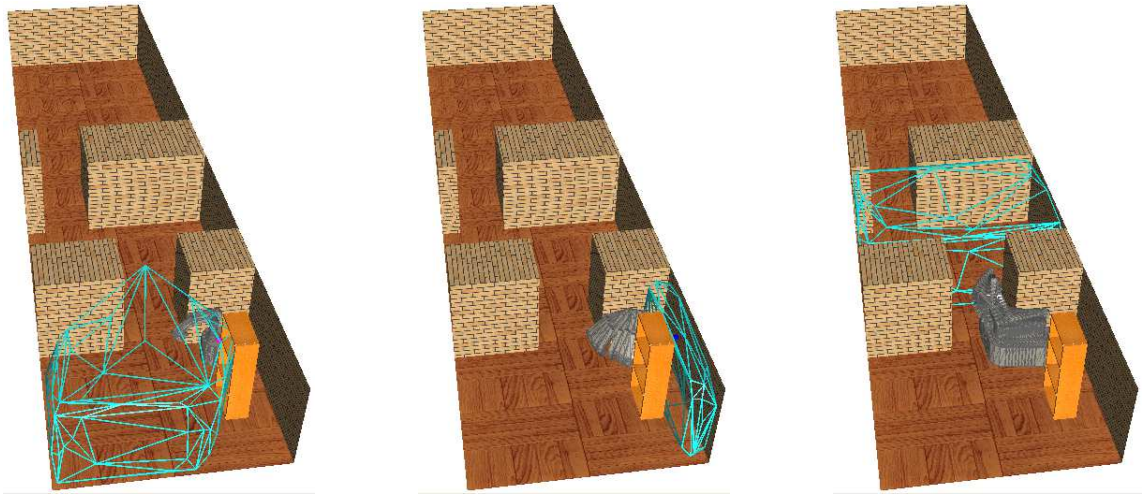
(a) Global path (b) Covering region $C_{id2,6}$ with the limited sensor

Fig. 6. (a) Global path and (b) covering region $C_{id2,6}$

the modified environment is different. In the figures a (cyan) mesh is used to show the convex regions.

In experiment 1, the environment was initially divided in 11 convex regions, 44 sensing configurations had to cover the environment with the limited sensor. The original order to visit convex regions was the following: $C_{id9,1} \rightarrow C_{id7,2} \rightarrow C_{id6,3} \rightarrow C_{id11,4} \rightarrow C_{id1,5} \rightarrow C_{id2,6} \rightarrow C_{id3,7} \rightarrow C_{id8,8} \rightarrow C_{id5,9} \rightarrow C_{id10,10} \rightarrow C_{id4,11}$.

Figure 5 (a) shows the initial robot configuration and the convex region $C_{id9,1}$. Figure 5 (b) shows the robot having the sensor inside the region $C_{id7,2}$. Figure 5 (c) shows the path to move between the regions $C_{id1,5}$ and $C_{id2,6}$.



(a) New obstacle location and region $C_{id18,2}$ (b) Path $C_{id18,2} \rightarrow C_{id16,3}$ (c) Path $C_{id12,8} \rightarrow C_{id1,9}$

Fig. 7. New order

Figure 6 (a) shows the global path to visit convex regions. Figure 6 (b) shows with light gray (yellow) the point samples used to approximate the visibility region of the limited sensor.

The original plan was modified as follows: $(\{C\}_t \bigcup \{C\}_{t+1})$=$\{C_{id4,11}, C_{id5,9}, C_{id6,3}, C_{id7,2}, C_{id8,8}, C_{id9,1}, C_{id10,10}, C_{id11,4}\}$. $C_\Delta = \{C_{id12}, C_{id13}, C_{id14}, C_{id15}, C_{id16}, C_{id17}, C_{id18}, C_{id19}\}$. Thus, 8 regions were removed from the original plan and 8 new regions were generated. The new order for visiting convex regions is $C_{id17,1} \rightarrow C_{id18,2} \rightarrow C_{id16,3} \rightarrow C_{id14,4} \rightarrow C_{id13,5} \rightarrow C_{id19,6} \rightarrow C_{id15,,7} \rightarrow C_{id12,8} \rightarrow C_{id1,9} \rightarrow C_{id2,10} \rightarrow C_{id3,11}$. 43 sensing configurations were needed to cover the modified environment.

Figure 7 (a) shows the new obstacle (a bookshelf) location and the robot having the sensor inside the region $C_{id18,2}$. Figure 7 (b) shows the path between the regions $C_{id18,2}$ and $C_{id16,3}$. Figure 7 (c) shows the path between the regions $C_{id12,8}$ and $C_{id1,9}$.

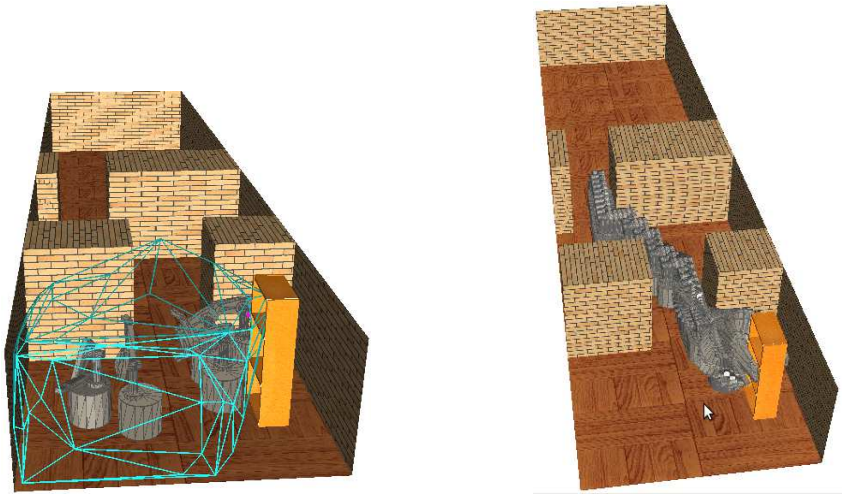
Figure 8 (a) shows the 4 sensing configuration that collectively covers the convex region $C_{id18,2}$ with the limited sensor. Figure 8 (b) shows the new global path to visit all convex regions.

The sub-order, sensing configurations, and paths to cover the room in which the obstacle has changed its location were modified. However, the sub-order, the paths, and the sensing configurations to cover the other two rooms were preserved.

Now, we present experiment 2, in which the obstacle (again a bookshelf) has been moved from the first to the last room, see figures 9 a) and b). Figure 10 a) shows the original global path for searching for the object, and Figure 10 b) shows the modified path.

In Tables 2 and 3, the two experiments are summarized and compared in terms of the number of convex regions, the number of sensing configurations to cover the whole environment, the computational running time to generate the original search plan, the computational running time to repair the plan, and the expected value of the time to find the object E[t]. Table 2 presents the results for the original plan generation, while Table 3 presents the results for the plan reparation. Notice that in experiment 1 the expected value of the time was actually improved with the new plan.

Comparing experiment 1 with experiment 2, it can be observed that the computational running time to repair the plan is larger in experiment 2 compared with experiment 1. In experiment 1 only one of the three rooms has suffered a change due



(a) Sensing configurations, region $C_{id18,2}$

(b) New global path

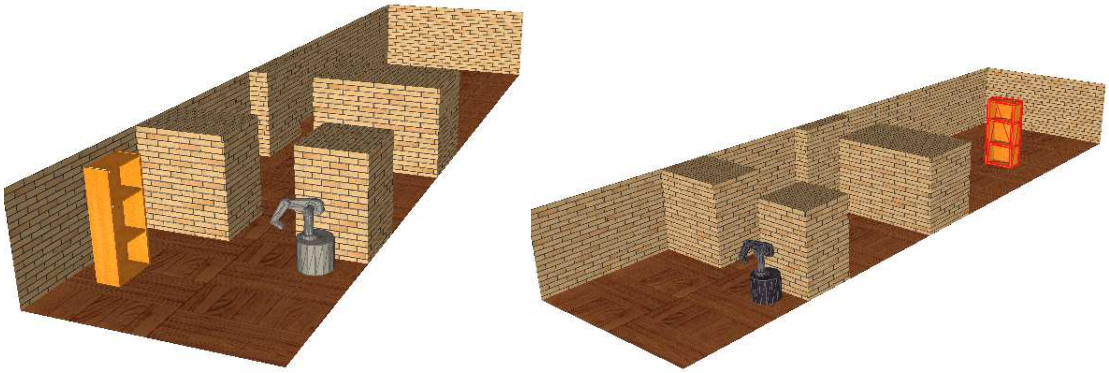
Fig. 8. (a) Sensing configurations, $C_{id18,2}$, and (b) new global path

Table 2. Parameters of original search plan

Experiment	# of convex regions	# of sensing configurations	computational running time to generate the plan	E[t]
1	11	44	1 min 58 sec.	796 units
2	10	49	2 min. 31 sec.	776.3 units

to the obstacle's change of location, while in experiment 2, two of the three rooms have changed due to the obstacle's change of location. In other words, the change of the environment in experiment 1 is more local while in experiment 2 the change of the environment is larger; this explains why the time to repair the plan is bigger. However, we highlight the fact that even in experiment 2, in which the change of the environment is more considerable, the time to repair the plan is still smaller than the time to generate the original search plan. These two experiments are the representative examples of other simulations we have run; in general, we have found that the expected value of the time for finding the object is almost the same after having updated the plan. In contrast, the computational running time needed to systematically repair the plan is smaller than the time needed to generate the whole plan again.

The time saved by repairing a plan vs. generating a new one depends on the change of the environment. A practical way to estimate this change can be computed as follows. First, compute the volume of the convex regions that have been modified minus the volume of their intersections; subtracting the volume of intersections avoids counting the same volume more than once. Second, divide this volume by the total volume of free space in the environment. This total volume corresponds to the sum of the volume of all the convex regions minus their intersection. The resulting value is normalized between zero and one. The smaller this value is, the more convenient is to repair a plan instead of generating a new one. In our experiments, we have also observed, that for almost empty environments, the expected time to find the object is often larger than for environments with many obstacles. Notice that in an almost empty environment there is more empty space to be covered with the limited sensor, and consequently more sensing configurations are



(a) Initial obstacle location (b) Final Obstacle location

Fig. 9. (a) Initial obstacle location and (b) final obstacle location

Table 3. Parameters of repaired plan

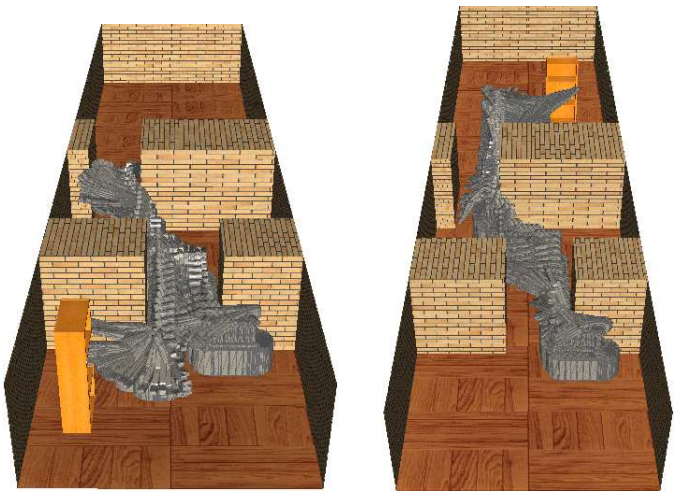
Experiment	# of convex regions	# of sensing configurations	computational running time to repair the plan	E[t]
1	11	43	21.5 sec.	771.68 units
2	10	42	1 min. 12 sec.	793.7 units

required to cover the whole environment. However, in a clutter environment, the computational running time to generate the robot's paths to visit sensing configurations is larger than in an almost empty environment.

7 Conclusions and Future Work

In this paper, we have addressed the problem of reducing the time for finding an object in a 3D environment. The object is sought with a 7 degree of freedom mobile manipulator robot with an "eye-in-hand" sensor. The sensor is limited in both range and field of view. First, we have proposed a strategy for coordinating the motion of robot's degrees of freedom optimizing only those most relevant for the task. Second, we have proposed an approach for repairing previously computed plans. We have shown that whenever the environment changes locally, our plan can also be repaired locally. We base our approach on a 3D convex region decomposition dividing the environment. The plan is repaired by generating a new subset of sensing configurations and a new order for visiting those configurations, considering only the convex regions related to the change in the map of the 3D environment.

The two proposed strategies are significant, because they considerably reduce the computational running time to generate a plan while the expected value of the time remains almost the same. The coordination of the robot's degrees of freedom, optimizing only a subset of them, allows one to generate search plans in a reasonable amount of time for big environments. In the statistics made over our experiments, when the first extension is applied, the expected value of the time has increased only by 13%, while the computational running time to compute the global search path was reduced almost 60 times. For the extension of repairing the original search plan, the reduction of the computational running time depends on how large the modification of the environment is. In our experiments, the processing time to repair a plan was always smaller than the time needed to generate the original plan. We have implemented all our algorithms and presented simulation results in realistic environments.



(a) Original search path (b) Modified search path

Fig. 10. (a) Original search path and (b) modified search path

We believe that the potential applications of our current approach are many, ranging from finding a specific piece of art in a museum to detecting injured people inside a building. In future we want to test our approach in a real robot equipped with a computer vision algorithm to detect the object.

Acknowledgements

This work was partially funded by CONACYT Project 106475 and by the NSF-CONACYT Project J110.534/2006.

References

1. **Acar, E., Choset, H., & Atkar, P. N.** (**2001**). Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and voronoi diagrams. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IEEE/RSJ-IROS 2001*, IEEE, pp. 1305–1311.
2. **Bourgault, F., Goktogan, A., Furukawa, T., & Durrant-White, H.** (**2004**). Coordinated search for a lost target in a bayesian world. *Advanced Robotics*, Vol. 18, No. 10, pp. 979–1000.
3. **Espinoza, J. & Murrieta-Cid, R.** (**2010**). A motion planner for finding an object in 3d environments with a mobile manipulator robot equipped with a limited sensor. In *Lecture Notes in Computer Science, IBERAMIA-2010*, volume 6433. pp. 532–541.
4. **Espinoza, J. & Murrieta-Cid, R.** (**2011**). Repairing plans for object finding in 3-d environments. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS 2011*, IEEE, pp. 4528–4535.
5. **Espinoza, J., Sarmiento, A., Murrieta-Cid, R., & Hutchinson, S.** (**2011**). Motion planning strategy for finding an object with a mobile manipulator in 3-d environments. *Advanced Robotics*, Vol. 25, No. 13-14, pp. 1627–1650.
6. **González, H. & Latombe, J.-C.** (**2001**). A randomized art-gallery algorithm for sensor placement. *Proc. 17th ACM Symp. on Computational Geometry (SoCG'01)*, ACM, pp. 232–240.
7. **Hert, S., Tiwari, S., & Lumelsky, V.** (**1996**). A terrain-covering algorithm for an auv. *Autonomous Robots*, Vol. 3, No. 2-3, pp. 91–119.
8. **Hsu, D., Latombe, J., & Motwani, R.** (**1997**). Path planning in expansive configuration spaces. *Proc. IEEE Int. Conf. on Robotics and Automation, IEEE-ICRA 1997*, IEEE, pp. 2719–2726.
9. **Kavraki, L. E., Svestka, P., Latombe, J., & Overmars, M. H.** (**1996**). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, Vol. 12, No. 4, pp. 566–580.

10. **Kroger, T. & Wahl, F. M.** (**2010**). Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, Vol. 26, No. 1, pp. 94–111.
11. **Latombe, J. C.** (**1991**). *Robot motion planning*. Kluwer.
12. **Lau, H., Huang, S., & Dissanayake, G.** (**2005**). Optimal search for multiple targets in a built environment. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IEEE/RSJ-IROS 2005*, IEEE, pp. 3740–3745.
13. **LaValle, S.** (**2006**). *Planning Algorithms*. Cambridge University Press.
14. **LaValle, S. M. & Kuffner, J. J.** (**2001**). Randomized kinodynamic planning. *International Journal of Robotics Research*, Vol. 20, No. 5, pp. 378–400.
15. **O'Rourke, J.** (**1982**). The complexity of computing minimum convex covers for polygons. *20th Annu. Allerton Conf. on Communication, Control, and Computing*, pp. 75–84.
16. **O'Rourke, J.** (**1987**). *Art Gallery Theorems and Algorithms*. Oxford University Press.
17. **Rodriguez-Sanchez, A. J., Simine, E., & Tsotsos, J. K.** (**2007**). Attention and visual search. *Int. J. Neural Syst*, Vol. 17, No. 4, pp. 277–288.
18. **Sanchez, G. & Latombe, J.** (**2002**). On the delaying collision cheching in prm planning. *International Journal of Robotics Research*, Vol. 21, No. 1, pp. 5–26.
19. **Sanchez, G. & Latombe, J.** (**2003**). A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In **Jarvis, R. & Zelinsky, A.**, editors, *ISRR'01*. STAR, Springer-Verlag, Berlin, pp. 403–417.
20. **Sarmiento, A., Murrieta-Cid, R., & Hutchinson, S.** (**2005**). A sample-based convex cover for rapidly finding an object in a 3-d environment. *Proc. IEEE Int. Conf. on Robotics and Automation, IEEE-ICRA 2005*, IEEE, pp. 3486–3491.
21. **Sarmiento, A., Murrieta-Cid, R., & Hutchinson, S. A.** (**2009**). An efficient motion strategy to compute expected-time locally optimal continuous search paths in known environments. *Advanced Robotics*, Vol. 23, No. 12-13, pp. 1533–1569.
22. **Shemer, T.** (**1992**). Recent results in art galleries. *Proc. IEEE*, Vol. 80, No. 9, pp. 1384–1399.
23. **Simeon, T., Laumond, J. P., & Nissoux, C.** (**2000**). Visibility based probabilistic roadmaps. *Advanced Robotics*, Vol. 14, No. 6, pp. 477–493.
24. **Vannoy, J. & Xiao, J.** (**2008**). Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes. *IEEE Transactions on Robotics*, Vol. 24, No. 5, pp. 1199–1212.

Judith Espinoza obtained her B.E. in Computer Science in 2004, and her M.Sc. in Computer Science in 2006, both from Beneménerita Universidad Autónoma de Puebla. In August 2012, she received the Ph.D. degree in Computer Science from the Centro de Investigación en Matemáticas (CIMAT), Guanajuato, Mexico. She is currently a professor at the Universidad Autónoma de Aguascalientes and is mainly interested in robot motion planning.

Rafael Murrieta-Cid received the B.Sc. degree in Physics Engineering from the Monterrey Institute of Technology and Higher Education, Monterrey, Mexico, in 1990, and the Ph.D. degree from the Institut National Polytechnique, Toulouse, France, in 1998. His Ph.D. research was done with the Robotics and Artificial Intelligence Group of the LAAS-CNRS. In 1998–1999, he was a Postdoctoral Researcher with the Department of Computer Science of the Stanford University, CA, USA. During 2002–2004, he was a Postdoctoral Research Associate with the Beckman Institute and the Department of Electrical and Computer Engineering of the University of Illinois at Urbana-Champaign, Urbana, IL, USA. From August 2004 to January 2006, he was a Professor and Director of the Mechatronics Research Center in Tec de Monterrey, State of Mexico Campus. Since March 2006, he has been with the Mathematical Computing Group, Centro de Investigación en Matemáticas, Guanajuato, Mexico. His research interests include robotics and robot motion planning. He has published more than 50 papers in journals and international conferences on these topics.

Article received on 12/11/2013, accepted on 01/09/2014.
Corresponding author is Rafael Murrieta-Cid.