Object Search by Manipulation

Mehmet R. Dogar[1], Michael C. Koval[1], Abhijeet Tallavajhula[2] and Siddhartha S. Srinivasa[1]

***Abstract*— We investigate the problem of a robot searching for an object. This requires reasoning about both perception and manipulation: certain objects are moved because the target may be hidden behind them and others are moved because they block the manipulator's access to other objects. We contribute a formulation of the object search by manipulation problem using visibility and accessibility relations between objects. We also propose a greedy algorithm and show that it is optimal under certain conditions. We propose a second algorithm which is optimal under all conditions. This algorithm takes advantage of the structure of the visibility and accessibility relations between objects to quickly generate optimal plans. Finally, we demonstrate an implementation of both algorithms on a real robot using a real object detection system.**

I. Introduction

Imagine looking for the salt shaker in a kitchen cabinet. Upon opening the cabinet, you are greeted with a cluttered view of jars, cans, and boxes—but no salt shaker. It must be hidden near the back of the cabinet, completely obscured by the clutter. You immediately start searching for it by pushing some objects out of the way and moving others to the counter until, eventually, you reveal your target.

Humans frequently manipulate their environment when searching for objects. If robotic manipulators are to be successful in human environments, they require a similar capability of searching for objects by removing the clutter that is in the way. In this context, clutter removal serves two purposes. First, removing clutter is necessary to gain visibility of the target. Second, it is necessary to gain access to objects that would be otherwise inaccessible.

Prior work has addressed the issues of interacting with objects to gain visibility and accessibility as separate problems. The work on *sensor placement* [1] and *search by navigation* [2]–[5] problems focuses on manipulating the sensor to gain visibility. One canonical example of the sensor placement problem is the Art Gallery problem [6], which would be equivalent to instrumenting the cabinet with enough sensors to guarantee that the salt shaker is visible. Similarly, the search by navigation problem would entail moving a mobile sensor through the cabinet to search for the target.

Conversely, the *reconfiguration planning* [7], [8] and *manipulation planning among movable obstacles* [9] problems focus on moving objects to grant the manipulator access to previously-inaccessible configurations. These approaches would be effective at gaining access to the salt shaker once

[1]M.R. Dogar, M.C. Koval, and S.S. Srinivasa are with The Robotics Institute, Carnegie Mellon University. {mdogar,mkoval,siddh}@cs.cmu.edu

[2]A. Tallavajhula is with the Indian Institute of Technology Kharagpur. 09me1028@iitkgp.ac.in

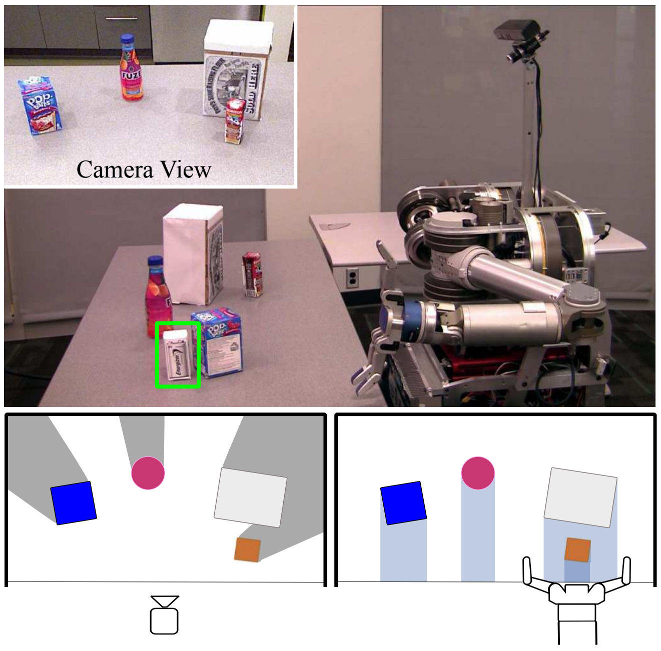
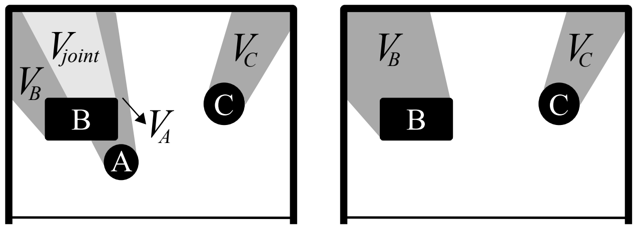


Fig. 1. An example of the object search problem on a real robot. The robot is searching for a target object (highlighted by the bounding box) on the table, but its view is occluded (drawn as gray regions) by other objects. The robot must remove these objects to search for the target. Objects may block the robot's access to other objects.

its pose is known, but are incapable of planning before the target is visually revealed. Recent work discusses the object search by manipulation problem [10], [11] but without any optimality guarantees.

One of our key insights is that the object search by manipulation problem requires simultaneously reasoning about both perception and manipulation. Some objects are moved because they are likely to hide the target, while others are moved only because they prevent the manipulator from accessing other objects in the scene.

Fig.1 shows a scene in which both situations occur. In this figure, HERB [12]—a robotic platform designed by the Personal Robotics Lab at Carnegie Mellon University—is searching for the white battery pack hidden on a cluttered table. HERB perceives the scene using its camera and uses the MOPED [13] system to detect and localize objects. As Fig.1-Top shows, HERB is initially unable to detect the battery pack because it is occluded by the blue Pop-Tart box. From HERB's perspective, the battery pack could be hiding in any of the occluded regions shown in Fig.1-Left. With no additional knowledge about the location of the target, HERB must sequentially remove objects from the scene subject to



(a) Initial Scene (b) `A` Removed

Fig. 2. An example of a scene containing a joint occlusion. Occlusions are drawn as dark gray and the joint occlusions as light gray. After `A` is removed, V_{joint} is only occluded by `B` and becomes part of $V_{\mathtt{B}}$

the physical limitations of its manipulator until the target is revealed. For example, Fig.1-Right shows that HERB is unable to grasp the large white box without first moving the brown juicebox out of the way.

In this paper, we formally describe the object search by manipulation problem by defining the *expected time to find the target* as a relevant optimization criterion and the concept of *accessibility* and *visibility* relations (Section II). Armed with these definitions, we are able to propose and analyze algorithms for object search by manipulation. We make the following theoretical contributions:

Greedy is sometimes optimal: We prove that under an appropriate definition of utility, the greedy approach to removing objects is optimal under a set of conditions, and provide insight into when it is suboptimal (Section III).

The connected components algorithm: We introduce an alternative algorithm, called the *connected components algorithm* and present a partial proof that it is optimal under all situations, and takes advantage of the structure of the scene to achieve polynomial time complexity on some scenes (Section IV).

Finally, we demonstrate both algorithms on our robot HERB (Section V) and provide extensive experiments that confirm the algorithms' theoretical properties (Section VI).

We are very excited about this research direction. The interplay between visibility and accessibility has revealed deep structure in the object search problem, structure that we were able to identify and exploit to derive the connected components algorithm. We discuss limitations and several extensions in Section VIII. We believe that our algorithms are a step towards enabling robots to perform complex manipulation tasks under high clutter and occlusions.

II. Object Search by Manipulation

We start with a scene $\mathcal{S}$ that is comprised of a known, static world populated with the set of movable objects $\mathcal{O}$, each of which has known geometry and pose.

A robot perceives the scene with its sensors and has partial knowledge of the objects that the scene contains. To the robot, the scene is comprised of the set of visible objects $\mathcal{O}_{seen} \subset \mathcal{O}$ and the volume of space V that is occluded to its sensors. In the object search problem, the occluded volume hides a target object $\mathtt{target} \in \mathcal{O}$ with known geometry, but unknown pose. For the remainder of this paper, we study a specific variant of the problem in which the target is the only hidden object, i.e. $\mathcal{O} = \mathcal{O}_{seen} \cup \{\mathtt{target}\}$. We discuss the presence of other hidden objects in Section VII.

The robot searches for the target by removing objects from $\mathcal{O}_{seen}$ until the target is revealed to its sensors. We define the order in which objects are removed from the scene as an *arrangement.*

Definition 1 (Arrangement): An arrangement of the set of objects o is a bijection $\mathcal{A}_o : \{1, \ldots, |o|\} \to o$ where $\mathcal{A}_o(i)$ is the i^{th} object removed.

Additionally, we define $\mathcal{A}_o(i,j)$ as the sequence of the i^{th} through the j^{th} objects removed by arrangement $\mathcal{A}_o$.

Given an arrangement $\mathcal{A}_o$ that reveals the target, the expected time to find the target is

$$E(\mathcal{A}_o) = \sum_{i=1}^{|o|} P_{\mathcal{A}_o(i)} \cdot T_{\mathcal{A}_o(1,i)} \quad (1)$$

where $P_{\mathcal{A}_o(i)}$ is the probability that the target will be revealed after removing object $\mathcal{A}_o(i)$ and $T_{\mathcal{A}_o(1,i)}$ is the time to move all objects up to and including $\mathcal{A}_o(i)$.

Our goal is to find the arrangement $\mathcal{A}^*_{\mathcal{O}_{seen}}$ that minimizes $E(\mathcal{A}^*_{\mathcal{O}_{seen}})$; i.e. reveals the target as quickly as possible.

A. Visibility

When the robot removes a set of objects from the scene it reveals the volume behind those objects.

Definition 2 (Revealed Volume): The volume of space V_o revealed by removing objects $o \subseteq \mathcal{O}_{seen}$ from scene $\mathcal{S}$.

In Fig.2a we show the revealed volumes of objects in an example scene[1]. V_{joint} is jointly occluded by object `A` and `B`, and is *not* included in either $V_{\mathtt{A}}$ or $V_{\mathtt{B}}$. This is because V_{joint} will not be revealed if only `A` or only `B` is removed from the scene.

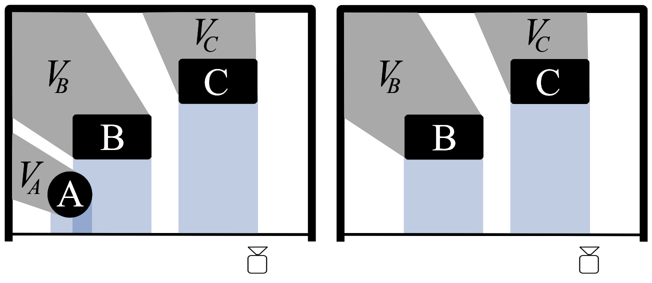
The volume that is revealed by removing an object may change as the scene changes. In Fig.2b we show $V_{\mathtt{B}}$ after `A` is removed from the scene in Fig.2a. Since `A` is no longer in the scene, $V_{\mathtt{B}}$ now includes V_{joint}. Similarly, $V_{\mathtt{A}}$ would expand to include V_{joint} if `B` was the first object removed from the scene. Regardless of the order in which `A` and `B` are removed, the revealed volume of $\{\mathtt{A},\mathtt{B}\}$ is $V_{\mathtt{A},\mathtt{B}} = V_{\mathtt{A}} + V_{\mathtt{B}} + V_{joint}$. In the most general case, an arbitrary number of objects can jointly occlude a volume. In that case, the volume would be revealed only after all of the occluding objects are removed from the scene.

We compute the probability that removing an object `A` will reveal the `target` using the revealed volume:

$$P_{\mathtt{A}} = \frac{V_{\mathtt{A}}}{V_{\mathcal{O}_{seen}}} \quad (2)$$

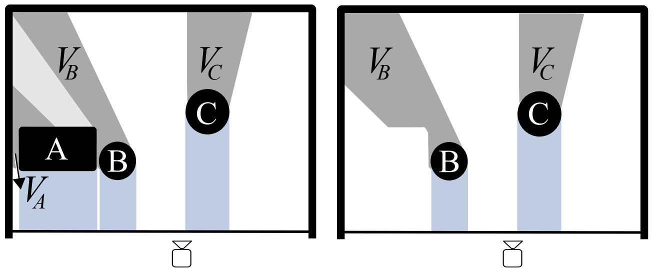
We compute the revealed volume in the configuration space (C-space) of the target object. In our implementation, we assume that the target rests stably on the workspace; i.e. the target's pose can be represented by $(x, y, \theta) \in \mathbb{SE}(2)$. We

[1]We use two-dimensional examples, e.g. Fig.2, throughout the paper for clarity of illustration. Our actual formulation and implementation uses complete three-dimensional models of the scene, objects, and the volumes.



(a) Initial Scene (b) A Removed

Fig. 3. A scene where the greedy algorithm performs suboptimally due to an accessibility constraint. Removing object `A` is necessary to access `B` and reveal the large volume $V_{\texttt{B}}$.



(a) Initial Scene (b) A Removed

Fig. 4. A scene where the greedy algorithm performs suboptimally due to a visibility constraint. Volumes $V_{\texttt{A}}$ and $V_{\texttt{B}}$ are individually small, but $V_{\texttt{A,B}}$ is large because of the joint occlusion between `A` and `B`.

discretize the target's C-space to generate a set of candidate poses and estimate V_o as the number of `target` poses that become visible if o is removed.

Partial views of objects are often hard to detect. Therefore, our visibility condition is conservative: we consider the target at a certain pose visible if the sensor can see it completely. We achieve this by sampling points on the target, shooting rays from the sensor to those points, and requiring that no ray is occluded by another object.

B. Accessibility

The manipulator uses a motion planner to grasp an object and remove it from the scene. To achieve this, the object must be *accessible* to the manipulator. Accessibility is blocked by other visible objects, and also by the occluded volume, which the manipulator is forbidden from entering.

Definition 3 (Accessibility Constraint): There is an *accessibility constraint* from an object `A` to object `B` if `A` must be removed for the manipulator to access `B`.

Any arrangement of objects in a scene must respect the objects' accessibility constraints. For example, in Fig.1-Right, the access to the big box is blocked by the smaller box in front of it.

We identify the accessibility constraints by using a motion planner, which returns a manipulator trajectory for each object in the scene. The manipulator trajectory for an object sweeps a certain volume in the space (illustrated as light blue regions in Fig.1). Objects that penetrate the swept volume result in accessibility constraints. Additionally, objects for which the occluded volume penetrates the swept volume also result in accessibility constraints.

We also use the manipulator trajectory for an object `A` to compute $T_{\texttt{A}}$ by estimating the time necessary to execute the trajectory on the robot. Since there is only a single action for each object, $T_{\texttt{A}}$ is constant for a given scene and does not depend on the sequence in which objects are removed.

III. Utility and Greedy Search

In this section, we discuss a greedy approach to solving the object search by manipulation problem.

While the overall goal is to minimize the amount of time it takes to find the target, a greedy approach requires a utility function to maximize at every step. The faster the robot reveals large volumes, the sooner it will find the target. We define the utility of an object much like the utility measures defined for sensor placement [1], [2].

Definition 4 (Utility): The *utility* of an object `A` is given by

$$U(\texttt{A}) = \frac{V_{\texttt{A}}}{T_{\texttt{A}}}$$

This measure naturally lends itself to greedy search. A greedy algorithm for our problem ranks the accessible objects in the scene based on their utility and the removes highest utility object. This results in a new scene, whereby the algorithm repeats until all objects are removed.

Unsurprisingly, it is easy to create situations where greedy search is suboptimal. Consider the scene in Fig.3. In this scene, $V_{\texttt{B}} \gg V_{\texttt{C}} > V_{\texttt{A}}$. For the sake of simplicity we assume that the time to move each object is similar, hence $U(\texttt{C}) > \texttt{U}(\texttt{A})$. As `B` is not accessible, the greedy algorithm compares $U(\texttt{A})$ and $U(\texttt{C})$ and chooses to move `C` first, producing the final arrangement $\texttt{C} \rightarrow \texttt{A} \rightarrow \texttt{B}$. However, moving the lower utility `A` first is the optimal choice because it reveals $V_{\texttt{B}}$ faster (Fig.3b), and gives the optimal arrangement $\texttt{A} \rightarrow \texttt{B} \rightarrow \texttt{C}$. It is easy to see that greedy can be made arbitrarily suboptimal by adding more and more objects with utility $U(\texttt{C})$ to the scene.

We present a second example of greedy's suboptimality in Fig.4. In this scene, all objects are accessible, $V_{\texttt{C}} > V_{\texttt{A}}$, and $V_{\texttt{C}} > V_{\texttt{B}}$. The greedy algorithm inspects the utilities and moves `C` first. However, there is a large volume jointly occluded by `A` and `B`, such that when either `A` or `B` is removed, the volume revealed by the second object significantly increases. We illustrate this in Fig.4b where `A` is removed. Hence, the optimal arrangement is $\texttt{A} \rightarrow \texttt{B} \rightarrow \texttt{C}$ because it quickly reveals the large volume jointly occluded by `A` and `B`.

The examples in Fig.3 and Fig.4 may suggest a k-step lookahead planner for optimality. However, the problem is fundamental: one can always create scenes where arbitrarily many objects jointly occlude large volumes, or where arbitrarily many objects block the accessibility to an object that hides a large volume behind it.

Surprisingly, however, it is possible to create nontrivial scenes where greedy search is *optimal*. We define the requirements of such scenes in the following theorem.

Theorem 3.1: In a scene where all objects are accessible and no volume is jointly occluded, a planner that is greedy over utility minimizes the expected time to find the target.

Proof: Suppose that $\mathcal{A}^*$ is a minimum expected time (i.e. optimal) arrangement. For any i, $1 \leq i < |\mathcal{O}_{seen}|$, we can create a new arrangement, $\mathcal{A}$, such that the i^{th} and $(i+1)^{\text{th}}$ objects are swapped; i.e. $\mathcal{A}(i) = \mathcal{A}^*(i+1)$ and $\mathcal{A}(i+1) = \mathcal{A}^*(i)$. $\mathcal{A}$ must be a valid arrangement because all objects are accessible.

No volume is jointly occluded, so the revealed volume of all objects will stay the same after the swap; i.e. $V_{\mathcal{A}^*(i)} = V_{\mathcal{A}(i+1)}$ and $V_{\mathcal{A}^*(i+1)} = V_{\mathcal{A}(i)}$. Since the rest of the two arrangements are also identical, using (1) and (2), we can compute the difference between $E(\mathcal{A})$ and $E(\mathcal{A}^*)$ to be:

$$E(\mathcal{A}) - E(\mathcal{A}^*) = V_{\mathcal{A}^*(i)} \cdot T_{\mathcal{A}^*(i+1)} - V_{\mathcal{A}^*(i+1)} \cdot T_{\mathcal{A}^*(i)}. \quad (3)$$

$E(\mathcal{A}^*)$ is optimal, therefore $E(\mathcal{A}) - E(\mathcal{A}^*) \geq 0$ and

$$\frac{V_{\mathcal{A}^*(i)}}{T_{\mathcal{A}^*(i)}} \geq \frac{V_{\mathcal{A}^*(i+1)}}{T_{\mathcal{A}^*(i+1)}},$$

which is simply $U(\mathcal{A}^*(i)) \geq U(\mathcal{A}^*(i+1))$. Hence, the optimal arrangement consists of objects sorted in weakly-descending order by their utilities.

There can be more than one weakly-descending ordering of the objects if multiple objects have the same utility. To see that all weakly-descending orderings are optimal, the same reasoning can be used to show that swapping two objects of the same utility does not change the expected time of an arrangement. ■

This result is rather startling. The greedy algorithm is incredibly efficient in terms of computational complexity. At each step, the algorithm finds the accessible object with maximum utility in linear time. In a scene of n objects, this results in a total computational complexity of $O(n^2)$. We show in Section IV that the worst-case complexity of the optimal search is $O(n^2 2^n)$. The theorem, however, shows that there are scenes in which greedy is optimal. We shall show in Section VI that these scenes do occur surprisingly regularly even with randomly generated objects.

In the next section we present a new algorithm which uses the collective utility of sequences of objects and generates an optimal plan.

IV. Connected Components Algorithm

The structure of the object search problem becomes more clear once we represent the visibility and accessibility constraints of a scene as a graph. Each node of this graph corresponds to an object in the scene. There is an edge between the nodes `A` and `B` if:

- `A` is blocking the access to `B`, or vice versa; or
- `A` and `B` are jointly occluding a non-zero volume.

An example scene and the corresponding graph is in Fig.5.

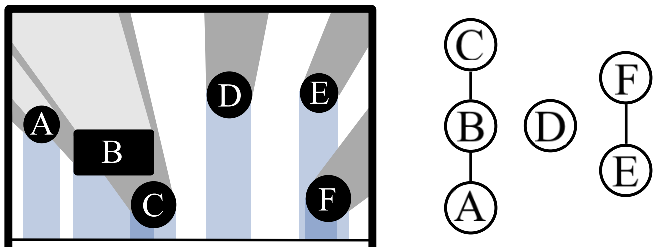


Fig. 5. Left: An example scene. Volumes occluded by a single object are shown in dark gray, joint occlusions are shown in light gray, and swept volumes are shown as light blue rectangles. Right: The corresponding graph with three connected components. Edges exist between nodes if the corresponding objects jointly occlude a volume or if one object blocks the access to the other.

Algorithm 1: ObjectSearchWithConnectedComponents

```
1  $\{c^1, c^2, ..., c^m\} \leftarrow$ FindConnectedComponents
2  foreach connected component $c^i$ do
3      $\mathcal{A}^*_{c^i} \leftarrow \mathbf{A^*}(c^i)$
4  $\mathcal{A}^*_{\mathcal{O}_{seen}} \leftarrow \emptyset$
5  repeat
6      bag $\leftarrow \emptyset$
7      foreach component arrangement $\mathcal{A}^*_{c^i}$ do
8          for $j \leftarrow 1$ to $|c^i|$ do
9              bag.Add( $\mathcal{A}_{c^i}(1, j)$ )
10     seq $\leftarrow \operatorname{argmax}_{\mathcal{A} \in \texttt{bag}} U(\mathcal{A})$
11     $\mathcal{A}^*_{\mathcal{O}_{seen}}$.Append(seq)
12     Remove seq from the $\mathcal{A}^*_{c^i}$ it belongs
13 until all objects are in the plan
14 return $\mathcal{A}^*_{\mathcal{O}_{seen}}$
```

We can divide the constraint graph into *connected components*. A connected component of the graph is a subgraph such that there exists a path between any two nodes in the subgraph [14]. For example, there are three connected components in Fig.5: {`A`, `B`, `C`}, {`D`}, and {`E`, `F`}.

A key insight is that the objects in a connected component do not affect the utility of the objects in another connected component. Hence, we can solve the arrangement problem for a connected component independently and then merge the solutions to produce a complete arrangement of the scene.

The examples in Fig.3 and Fig.4 show that the utility of a single object is not informative enough for general optimality when the scene contains joint occlusions or accessibility constraints. Instead, we need to consider the utility of removing multiple objects from the scene.

Definition 5 (Collective Utility): The *collective utility* of a set of objects o is given by

$$U(o) = \frac{V_o}{T_o}$$

We present a new algorithm in Algorithm 1 that uses the collective utility of sequences from connected components to generate an optimal arrangement of the complete scene. It first identifies the connected components in the scene. Then

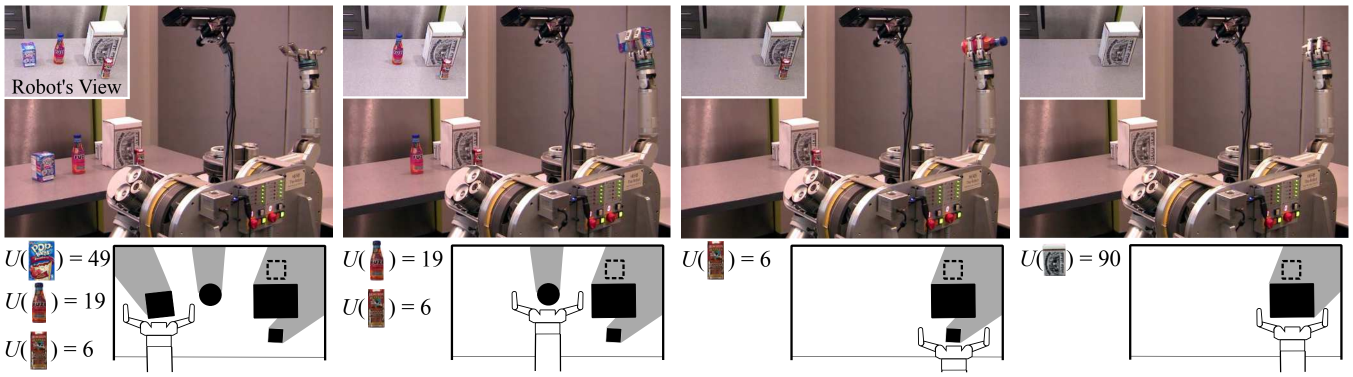


Fig. 6. Greedy planner. We present the utility of all accessible objects at each step. The pose of the target (unknown to the robot) is marked with dashed lines in the illustration.

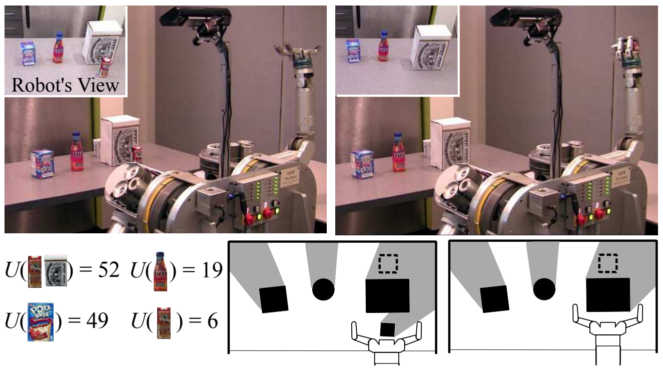


Fig. 7. Optimal connected-components planner. The utilities of partial sequences from different connected components are presented at each step.

it finds the optimal arrangement internal to a connected component using an A*-search implementation described below. It then merges these arrangements iteratively by finding the maximum utility[2] prefixes of the optimal arrangements of the connected components.

The way Algorithm 1 merges the sequences from different connected components guarantees that the final arrangement is optimal. In Appendix we present a partial proof.

A. A* Search Algorithm

We use A* search to find the optimal arrangement inside a connected component. A* works with a directed-acyclic-graph structure where the nodes are the set of remaining objects in the scene. Neighbors are scenes with one accessible object removed. Assume the partial arrangement $\mathcal{A}$ of k objects in the scene reaches a node in the search graph. The cost-to-come is

$$f = \sum_{i=1}^{k} \left(\frac{V_{\mathcal{A}(i)}}{V_{\mathcal{O}_{seen}}} \right) T_{\mathcal{A}(1,i)}$$

The cost-to-go can be approximated as

$$g = \left(\frac{V_{\mathcal{O}_{seen}} - V_{\mathcal{A}(1,k)}}{V_{\mathcal{O}_{seen}}} \right) \left(T_{\mathcal{A}(1,k)} + \min_{a \in \{\mathcal{O}_{seen} \setminus \mathcal{A}(1,k)\}} (T_a) \right)$$

[2]In the rare event that that multiple sequences share the maximum utility, the algorithm breaks the tie by choosing the sequence with the maximum utility prefix.

The cost-to-go heuristic optimistically reasons that in the $(k+1)^{\text{th}}$ action, all the remaining occluded volume will be revealed. Among the remaining objects, $\mathcal{O}_{seen} \setminus \mathcal{A}(1,k)$, we find the object that can be removed with the minimum time and use its time as the time of the $(k+1)^{\text{th}}$ action. The heuristic is admissible as it underestimates the time to find the target.

The A* search produces the optimal arrangement. However, running it on a large scene is intractable due to its high computational complexity. A* must search over a graph containing up to 2^n nodes and $O(n^2)$ edges, resulting in a worst-case complexity of $O(n^2 2^n)$.

B. Complexity of the Connected Components Algorithm

The connected components algorithm divides the set of objects into smaller sets, runs A* on each connected component, and then merges the plans for each component optimally. If the scene has no constraints, then there is one object per connected component and this algorithm reduces to the greedy algorithm. Conversely, if the constraint graph is connected, this algorithm is equivalent to running A* on the full scene. Therefore, the performance of this algorithm ranges from $O(n^2)$, the performance of the greedy algorithm, to $O(n^2 2^n)$, the performance of A*, depending upon the size of the connected components. Geometric limitations put an upper bound on the number of accessibility and joint occlusion constraints that are possible in a given scene, so it is unlikely that any scene will exercise the worst case performance. These performance gains will be most significant on large scenes in which objects are spatially partitioned, e.g. on different shelves in a fridge, but will be modest on small, densely packed scenes.

V. Implementation

We implemented the greedy, and connected components algorithms on our robot HERB. We used HERB's camera and the MOPED [13] system to detect and locate objects in the scene. We present an example scene where HERB successfully found the target object using the greedy algorithm in Fig.6. In this scene the target object, a battery pack, is hidden behind the large box, which also occludes the largest volume. Since the large box is inaccessible, the greedy

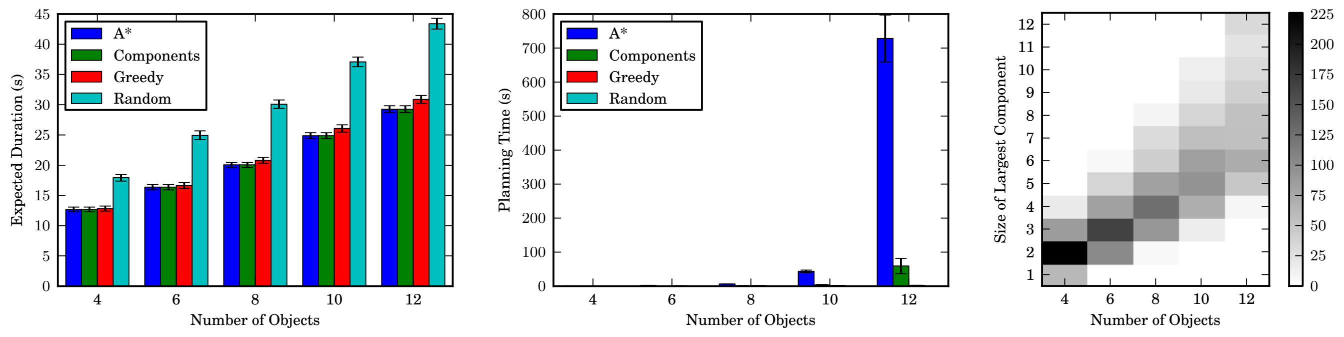
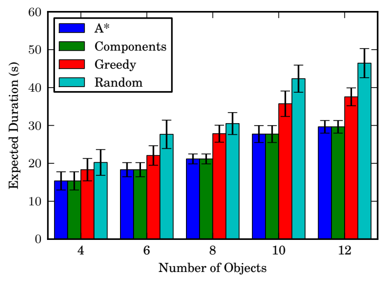
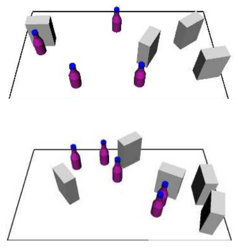


Fig. 8. Performance of the random, greedy, A*, and connected component planners as a function of number of objects. All results are averaged over approximately 400 random scenes and are plotted with their 95% confidence interval. The relationship between scene size and the size of the largest connected component is also plotted as a two-dimensional histogram.



(a) Expected Search Duration



(b) Example Scenes

Fig. 9. (a) 95th percentile of expected time to find the target (b) Two example scenes where greedy performed poorly. The black lines denote the workspace boundary.

planner compares the utilities of the other three objects, and removes the largest utility object at each step. Even though the large box is hiding a large volume, the greedy planner removes it last, resulting in a long task completion time.

In Fig.7 the scene is the same but HERB uses the optimal connected components algorithm. There are three connected components in this scene $\{\texttt{BlueBox}\}$, $\{\texttt{Bottle}\}$, and $\{\texttt{LargeBox}, \texttt{SmallBox}\}$. The connected components algorithm considers the collective utilities of multiple objects from each connected component. In the scene the algorithm considers both $U(\texttt{SmallBox})$ and $U(\texttt{SmallBox}, \texttt{LargeBox})$. The utility of `SmallBox` is very small compared with the other immediately accessible objects, but combined with `LargeBox`, their utility is large enough that the algorithm removes `SmallBox` as the first object. It then removes the large box and finds the target object. We present the actual footage of these experiments in the accompanying video.

VI. Experiments and Results

We also investigated the performance of the different algorithms we presented through extensive experiments in simulation. We implemented the greedy, A*, and connected components algorithms in OpenRAVE [15]. We also implemented a baseline algorithm which randomly picks an accessible object and removes it from the scene. We evaluated these algorithms on randomly generated scenes. Each scene contained n objects—half juice bottles and half large boxes—that were uniformly distributed over a wide 1.4×0.8 m workspace. None of the generated scenes contained hidden objects and the planner used a simulated motion planner based on the capabilities of a simple manipulator. The manipulator was only capable of moving straight, parallel to the table and at a constant speed of 0.1 m/s. Visibility was simulated using the pinhole camera model under the conservative assumption that an object is visible iff it is completely unoccluded.

We present results from scenes with 4, 6, 8, 10, and 12 objects in Fig.8 along with the 95% confidence intervals. We conducted approximately 400 simulations for each different number of objects, resulting a in total of 2000 different scenes. The data in Fig.8a shows that the greedy algorithm becomes increasingly suboptimal as the number of objects increases. All three algorithms significantly outperform the random algorithm, which serves as a rough upper bound for the expected search duration. Unfortunately, the optimality of A* comes with the cost of exponential complexity in the number of objects. This complexity causes the planning time of A* to dominate the other planning times shown in Fig.8b.

While still optimal, the connected components algorithm achieves much lower planning times than A*. By running A* on smaller subproblems, the connected components algorithm is exponential in the size of the largest connected component, k, instead of the size of the entire scene. Fig.8c shows that $k \approx n/2$ for $n \leq 8$ and increases when $n = 10$, causing the large increase in planning time between $n = 8$ and $n = 10$ in Fig.8b. With fixed computational resources, these results show that the connected components algorithm is capable of solving most scenes of size $2n$ in the amount of time it would take A* to solve a scene of size n. For sparse scenes, the connected components algorithm achieves optimality with planning times that are comparable those of the greedy algorithm.

One surprising results of our experiments is that, while greedy is not optimal in the general case, it does very well on average. We found that in 50% of the 2000 different scenes, the greedy algorithm produced the optimal sequence. Our explanation for greedy's performance is that the geometry

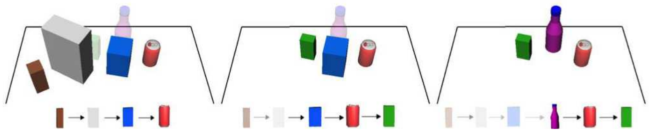


Fig. 10. Example of replanning on a scene with two hidden objects. Each replanning stage is shown as a separate frame along with the corresponding plan. Hidden objects are shown as semi-transparent and the workspace bounds are indicated by a black line.

of our workspace enforces a tradeoff between the volume occluded by an object and the number of objects that block its accessibility. For an object to occlude a large volume, it must be near the front of the workspace. This makes it unlikely that multiple objects can be placed in front of it.

To see how poorly greedy can perform, in Fig.9a we plotted expectations, this time using only 5% of the scenes where greedy performed poorly. Among all the different scenes, the worst performance was 2.04 times the expected time of the optimal sequence. We show two example scenes where greedy performs poorly in Fig.9b. In all these examples small bottles are making a large box inaccessible, with very little volume hidden behind the bottles.

VII. Replanning for Hidden Objects

All of the algorithms described above can be easily generalized to handle environments that contain hidden objects in addition to the target. Objects must be smaller than the target object or there is danger of the arm colliding with an hidden object while searching for the target. If this condition holds, then one can simply re-execute the planner on the remaining objects whenever an hidden object is revealed. This strategy is optimal given the available information if there is no a priori information about the type, number, or location of the hidden objects. If there are k hidden objects, then this replanning strategy multiplies the total planning time of an optimal algorithm by a factor of $O(k)$. In the case of the greedy or random algorithm, the replanning adds $O(k)$ overhead from reevaluating visibility after each object is revealed.

Fig.10 shows an example of replanning on a scene containing six objects. Two objects, shown as semi-transparent in the figure, are initially hidden and are revealed once the occluding objects are removed. The robot begins by executing the connected components planner on a scene containing the four visible objects. After executing the first two actions in that plan, the robot detects that a new object has been revealed and replans for the remaining objects. In this case, the optimal ordering is unchanged and the newly-revealed object is simply appended to the existing plan. After executing another action, the second hidden object is revealed and the robot must replan a second time. Order of the optimal sequence is changed by the addition of the hidden object and it is suboptimal to continue executing the previous plan.

VIII. Future Work

In future work, we are excited about exploring this problem deeper and relaxing some of the simplifying assumptions.

Perception Model. Our framework allows for any sensor model. We will explore relaxing the conservative requirement of the entire target being visible to other perceptual models that address partial visibility.

Placement Planner. Currently, we assume that objects removed can be place in an empty space nearby. A natural extension is planning the placement of objects. Doing so will require considering a number of constraints, e.g. will the replaced object make regions inaccessible, are we permitted to move the object more than once, and so on.

Complex Motion Planners. We use a straight-path reaching for objects, which is conceptually simple: there is a single action for an object. We are excited to study how a more complex motion planner, e.g. one returning a minimum-constraint violation trajectory [16], can be integrated into our system.

Sensor Planning. Aside from reaching to objects, the robot does not move its base. By combining the ability of search through manipulation with sensor planning, the robot could find targets quicker. Sensor planning would include working with multiple camera poses and planning for the base when searching for a target in a larger environment.

Appendix

Here we show that Algorithm 1 produces an optimal solution to the object search by manipulation problem.

We state a property of the collective utility as a lemma.

Lemma 8.1: Given an arrangement $\mathcal{A}_o$,

$$\begin{aligned} &U(\mathcal{A}_o(1,|o|)) \geq U(\mathcal{A}_o(1,k)) \\ \implies &U(\mathcal{A}_o(k+1,|o|)) \geq U(\mathcal{A}_o(1,|o|)) \end{aligned}$$

In other words, if the utility of the complete arrangement is larger than the utility of the first k objects, then the utility of the last $|o|-k$ objects must be larger than the utility of the complete arrangement.

Proof: We are given that

$$\frac{V_{\mathcal{A}_o(1,k)} + V_{\mathcal{A}_o(k+1,|o|)}}{T_{\mathcal{A}_o(1,k)} + T_{\mathcal{A}_o(k+1,|o|)}} \geq \frac{V_{\mathcal{A}_o(1,k)}}{T_{\mathcal{A}_o(1,k)}}$$

Rearranging yields

$$V_{\mathcal{A}_o(k+1,|o|)} \cdot T_{\mathcal{A}_o(1,k)} \geq V_{\mathcal{A}_o(1,k)} \cdot T_{\mathcal{A}_o(k+1,|o|)}$$

Adding $V_{\mathcal{A}_o(k+1,|o|)} \cdot T_{\mathcal{A}_o(k+1,|o|)}$ to both sides and rearranging, we get

$$\frac{V_{\mathcal{A}_o(k+1,|o|)}}{T_{\mathcal{A}_o(k+1,|o|)}} \geq \frac{V_{\mathcal{A}_o(1,k)} + V_{\mathcal{A}_o(k+1,|o|)}}{T_{\mathcal{A}_o(1,k)} + T_{\mathcal{A}_o(k+1,|o|)}}$$

■

Theorem 8.1: Given an optimal arrangement of a scene $\mathcal{A}^*$, for any two adjacent sequence of objects in the arrangement $\mathcal{A}^*(i,j)$ and $\mathcal{A}^*(j+1,k)$, where $i \leq j < k$, if there are neither accessibility constraints nor joint occlusions between the objects in the two sequences (i.e. if the sequences are from different connected components), then the utility of the former sequence is greater than or equal to the utility of the latter sequence: $U(\mathcal{A}^*(i,j)) \geq U(\mathcal{A}^*(j+1,k))$.

Proof: The proof proceeds similar to the proof of Theorem 3.1. We create a new arrangement $\mathcal{A}$ that is identical to $\mathcal{A}^*$ except that the two adjacent sequences are swapped: $\mathcal{A}(i, i+k-j) = \mathcal{A}^*(j+1, k)$ and $\mathcal{A}(i+k-j+1, k) = \mathcal{A}^*(i, j)$. $\mathcal{A}$ must be a valid arrangement since we are given that no object in $\mathcal{A}^*(i, j)$ is blocking access to $\mathcal{A}^*(j+1, k)$. Then we can compute the difference $E(\mathcal{A}) - E(\mathcal{A}^*)$ to be:

$$\sum_{l=i}^{j} \left(\frac{V_{\mathcal{A}^*(l)}}{V_{\mathcal{O}_{seen}}} \cdot T_{\mathcal{A}^*(j+1,k)} \right) - \sum_{l=j+1}^{k} \left(\frac{V_{\mathcal{A}^*(l)}}{V_{\mathcal{O}_{seen}}} \cdot T_{\mathcal{A}^*(i,j)} \right)$$

Since $\mathcal{A}^*$ is optimal, $E(\mathcal{A}) - E(\mathcal{A}^*) \geq 0$. After canceling out the common terms and rearranging, we are left with

$$\frac{\sum_{l=i}^{j} V_{\mathcal{A}^*(l)}}{T_{\mathcal{A}^*(i,j)}} \geq \frac{\sum_{l=j+1}^{k} V_{\mathcal{A}^*(l)}}{T_{\mathcal{A}^*(j+1,k)}}$$

Simply, $U(\mathcal{A}^*(i, j)) \geq U(\mathcal{A}^*(j+1, k))$. ■

We state a lemma without proof.

Lemma 8.2: The relative ordering of objects in the optimal arrangement of a connected component will be preserved in the optimal ordering for the complete scene. Formally, if $\mathcal{A}_c^*$ is the optimal arrangement for a connected component c, and $\mathcal{A}_o^*$ is the optimal arrangement of o, such that $c \subseteq o$, then

$$i < j \implies \mathcal{A}_o^{*-1}(\mathcal{A}_c^*(i)) < \mathcal{A}_o^{*-1}(\mathcal{A}_c^*(j))$$

where $1 \leq i, j \leq |c|$, and $\mathcal{A}_o^{*-1}$ returns the index of an object in the arrangement $\mathcal{A}_o^*$.

Finally we can prove that the connected components algorithm is optimal.

Theorem 8.2: Let's say we are given m connected components of a set of objects, o, and we are also given an optimal arrangement for each connected component $\mathcal{A}_{c^i}$ for $i = 1, \ldots, m$. Let's say we computed the utility of all sequences of objects in the form $\mathcal{A}_{c^i}(1, j)$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, |c^i|$, and found $\mathcal{A}_{c^*}(1, j^*)$ to have the maximum utility. Then an optimal arrangement for o starts with $\mathcal{A}_{c^*}(1, j^*)$.

Proof: Assume that the optimal arrangement $\mathcal{A}_o^*$ does not start with $\mathcal{A}_{c^*}(1, j^*)$. We will prove that this is not possible.

Given an arrangement of o, we can view it as a series of partitions, where each partition consists of a contiguous sequence of objects from the same connected component. Due to Lemma 8.2, each such partition in $\mathcal{A}_o^*$ can be represented as subsequences of the connected component arrangements $\mathcal{A}_{c^i}$. In particular, we are interested in two partitions of the optimal arrangement of o:

$$\mathcal{A}_o^* = [\mathcal{A}_{c'}(1, j') \ldots \mathcal{A}_{c^*}(k, l) \ldots]$$

where c' is one of the connected components, and $1 \leq j' \leq |c'|$. $\mathcal{A}_{c^*}(k, l)$ is the partition that includes the object $\mathcal{A}_{c^*}(j^*)$, hence $k \leq j^* \leq l$. We know that $\mathcal{A}_{c^*}(1, j^*)$ has the maximum utility of all the sequences in the form $\mathcal{A}_{c^i}(1, j)$ where c^i is any connected component and $j = 1, \ldots, |c^i|$. Then,

$$U(\mathcal{A}_{c^*}(1, j^*)) > U(\mathcal{A}_{c^*}(1, k-1)) \tag{4}$$

and also

$$U(\mathcal{A}_{c^*}(1, j^*)) > U(\mathcal{A}_{c'}(1, j')) \tag{5}$$

Using Lemma 8.1 and (4), we get

$$U(\mathcal{A}_{c^*}(k, j^*)) > U(\mathcal{A}_{c^*}(1, j^*))$$

Then from (5),

$$U(\mathcal{A}_{c^*}(k, j^*)) > U(\mathcal{A}_{c'}(1, j')) \tag{6}$$

Considering the utilities of all the partitions in $\mathcal{A}_o^*$ up to $\mathcal{A}_{c^*}(k, l)$, we know that they should be ordered in descreasing order of utility and be larger than $\mathcal{A}_{c^*}(k, j^*)$ (Theorem 8.1):

$$U(\mathcal{A}_{c'}(1, j')) > \ldots > U(\mathcal{A}_{c^*}(k, j^*))$$

which contradicts (6). ■

References

[1] J. Espinoza, A. Sarmiento, R. Murrieta-Cid, and S. Hutchinson, "Motion Planning Strategy for Finding an Object with a Mobile Manipulator in Three-Dimensional Environments." *Advanced Robotics*, 2011.

[2] Y. Ye and J. Tsotsos, "Where to look next in 3d object search," in *IEEE International Symposium on Computer Vision*, 1995.

[3] ——, "Sensor planning for 3d object search," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 145–168, February 1999.

[4] K. Shubina and J. Tsotsos, "Visual search for an object in a 3d environment using a mobile robot," *Computer Vision and Image Understanding*, pp. 535–547, 2010.

[5] K. Sjo, D. Lopez, C. Paul, P. Jensfelt, and D. Kragic, "Object search and localization for an indoor mobile robot," *Journal of Computing and Information Technology*, pp. 67–80, 2009.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[7] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 549–565, 1998.

[8] M. Dogar and S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, no. 3, pp. 217–236, June 2012.

[9] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IEEE International Conference on Robotics and Automation*, 2007.

[10] M. Gupta and G. Sukhatme, "Interactive perception in clutter," in *The RSS 2012 Workshop on Robots in Clutter: Manipulation, Perception and Navigation in Human Environments*, 7 2012.

[11] L. Kaelbling and T. Lozano-Perez, "Unifying perception, estimation and action for mobile manipulation via belief space planning," in *IEEE ICRA*, 2012.

[12] S. Srinivasa, D. Berenson, M. Cakmak, A. Collet Romea, M. Dogar, A. Dragan, R. A. Knepper, T. D. Niemueller, K. Strabala, J. M. Vandeweghe, and J. Ziegler, "Herb 2.0: Lessons learned from developing a mobile manipulator for the home," *Proceedings of the IEEE*, vol. 100, no. 8, pp. 1–19, July 2012.

[13] M. Martinez, A. Collet, and S. Srinivasa, "MOPED: A Scalable and Low Latency Object Recognition and Pose Estimation System," in *IEEE ICRA*, 2010.

[14] J. Hopcroft and R. Tarjan, "Algorithm 447: efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, no. 6, June 1973.

[15] R. Diankov and J. Kuffner, "OpenRAVE: A Planning Architecture for Autonomous Robotics," Robotics Institute, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[16] K. Hauser, "The minimum constraint removal problem with three robotics applications," in *In Workshop on the Algorithmic Foundations of Robotics (WAFR)*, June 2012.