Auton Robot (2014) 36:153–167
DOI 10.1007/s10514-013-9372-x

Object search by manipulation

Mehmet R. Dogar · Michael C. Koval · Abhijeet Tallavajhula · Siddhartha S. Srinivasa

Received: 8 March 2013 / Accepted: 15 October 2013 / Published online: 26 October 2013

Abstract We investigate the problem of a robot searching for an object. This requires reasoning about both perception and manipulation: some objects are moved because the target may be hidden behind them, while 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 takes advantage of the structure of the visibility and accessibility relations between objects to quickly generate plans. Our empirical evaluation strongly suggests that our algorithm is optimal under all conditions. We support this claim with a partial proof. Finally, we demonstrate an implementation of both algorithms on a real robot using a real object detection system.

Keywords Robotic manipulation · Manipulation planning · Object search

M. R. Dogar (✉) · M. C. Koval · S. S. Srinivasa
The Robotics Institute, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, PA, USA
e-mail: mdogar@cs.cmu.edu

M. C. Koval
e-mail: mkoval@cs.cmu.edu

S. S. Srinivasa
e-mail: siddh@cs.cmu.edu

A. Tallavajhula
Indian Institute of Technology Kharagpur, Kharagpur, India
e-mail: 09me1028@iitkgp.ac.in

1 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 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. Work on the *sensor placement* (Espinoza et al. 2011) and *search by navigation* (Ye and Tsotsos 1995, 1999; Shubina and Tsotsos 2010; Sjo et al. 2009; Ma et al. 2011; Anand et al. 2013) problems focuses on moving the sensor to gain visibility. One canonical example of the sensor placement problem is the Art Gallery problem (de Berg et al. 2008), 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 involve moving a mobile sensor through the cabinet to search for the target.

Conversely, the *reconfiguration planning* (Ben-Shahar and Rivlin 1998; Dogar and Srinivasa 2012; Ota 2009) and *manipulation planning among movable obstacles* (Stilman et al. 2007; van den Berg et al. 2008; Chen and Hwang 1991; Overmars et al. 2006) problems focus on moving objects

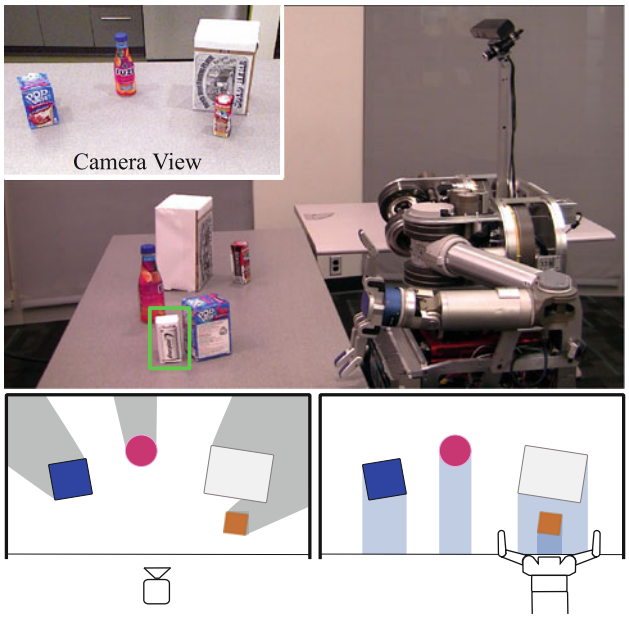

Fig. 1 An example of the object search problem on a real robot. The robot is searching for a target object (highlighted by the *bounding box*) on the table, but its view is occluded (drawn as *gray regions*) by other objects. The robot must remove these objects to search for the target. Objects may block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

to grant the manipulator access to previously-inaccessible configurations. These approaches would be effective at gaining access to the salt shaker once its pose is known, but are incapable of planning before the target is visually revealed. Recent work discusses the object search by manipulation problem (Gupta and Sukhatme 2012; Kaelbling and Lozano-Perez 2012) without reference to optimality. One of these works uses a generative model of object-object co-occurrence and spatial constraints (Wong et al. 2013) to guide the robot's search, which is similar to the prior distribution we introduce in §8. Other related work includes exploring the environment with the goal of building three-dimensional models of novel objects using maximally informative actions (van Hoof et al. 2012).

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.

Figure 1 shows a scene in which both situations occur. In this figure, HERB (Srinivasa et al. 2012)—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 uses its camera 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 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 (§2). Armed with these definitions, we are able to propose and analyze algorithms for object search by manipulation. We make the following theoretical contributions:

1.1 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 (§3).

1.2 The connected components algorithm

We introduce an alternative algorithm, called the *connected components algorithm*, which takes advantage of the structure of the scene to approach polynomial time complexity on some scenes (§5). Our extensive experiments show that this algorithm produces optimal plans under all situations, and we present a partial proof of optimality.

Finally, we demonstrate both algorithms on our robot HERB (§6.1, §6.2) and provide extensive experiments that confirm the algorithms' theoretical properties (§6).

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 several extensions in §7, §8, and §9. We discuss limitations and future work in §10. We believe that our algorithms are a step towards enabling robots to perform complex manipulation tasks under high clutter and occlusions.

2 Object search by manipulation

We start with a scene 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 $\texttt{target} \in \mathcal{O}$ 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. $\mathcal{O} = \mathcal{O}_{seen} \cup \{\texttt{target}\}$. We discuss the presence of other hidden objects in §9.

The robot searches for the target by removing objects from $\mathcal{O}_{seen}$ until the target is revealed to its sensors. As objects are removed, fewer objects remain in the scene, which we denote by $s \subseteq \mathcal{O}_{seen} \cdot \mathcal{O}_{seen}$ refers to the initial set of visible objects and does not change as objects are removed. We define the order in which objects are removed as an *arrangement*.

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

Additionally, we define $\mathcal{A}_o(i, j)$ as the sequence of the ith through the jth 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

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

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.

2.1 Visibility

When the robot removes a set of objects from the scene it reveals a set of candidate poses of the target object that were previously occluded. These *revealed configurations* are defined in target's configuration space C.

Definition 2 (*Revealed Configurations*) The set of candidate target poses $C_{o|s} \subseteq C$ revealed by removing objects $o \subseteq s$ from a scene containing objects $s \subseteq \mathcal{O}_{seen}$.

The probability of revealing the target after removing o from s is determined by the volume of $C_{o|s}$. We call this the *revealed volume* of those objects.

Definition 3 (*Revealed Volume*) The volume $V_{o|s}$ revealed by removing objects $o \subseteq s$ from a scene containing objects $s \subseteq \mathcal{O}_{seen}$ is

$$V_{o|s} = \int_{x \in C_{o|s}} P_0(x)\, dx \tag{2}$$

where $P_0(x)$ is a prior distribution over the pose of the target object.

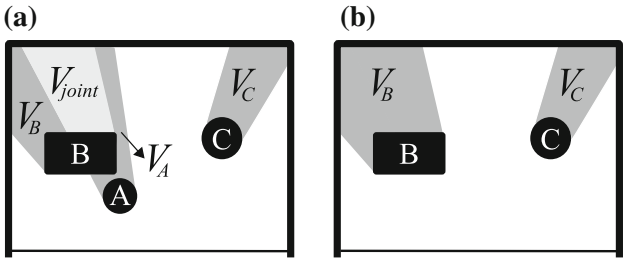

Fig. 2 **a** An example of a scene containing a joint occlusion. Occlusions are drawn as *dark gray* and the joint occlusions as *light gray*. **b** The scene after A is removed. **a** Initial scene, **b** A removed

We assume that $P_0(x)$ is uniform for the remainder of this discussion to simplify our examples. We discuss the general case of using a non-uniform $P_0(x)$ to encode semantic knowledge about the scene in §8.

Additionally, we will drop the scene s from this notation whenever it is obvious from the context. For example in Fig. 2a we write $V_{\texttt{A}}$ instead of the more verbose $V_{\{\texttt{A}\}|\{\texttt{A},\texttt{B},\texttt{C}\}}$. Similarly, instead of using $V_{\mathcal{A}_o(i,j)|\mathcal{A}_o(i,|o|)}$ to refer to the volume revealed between the ith and jth steps of an arrangement, we will simply use $V_{\mathcal{A}_o(i,j)}$.

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 $V_{\texttt{A}}$ or $V_{\texttt{B}}$. This is because V_{joint} will not be revealed if only A or only B is removed from the scene.

In Fig. 2b we show $V_{\texttt{B}}$ after A is removed from the scene in Fig. 2a. Since A is no longer in the scene, $V_{\texttt{B}}$ now includes V_{joint}. Similarly, $V_{\texttt{A}}$ 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 {A, B} is $V_{\{\texttt{A},\texttt{B}\}} = V_{\texttt{A}} + V_{\texttt{B}} + V_{joint}$. 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.

Given an arrangement $\mathcal{A}_{\mathcal{O}_{seen}}$ we compute the probability that the target will be revealed at the ith step using the revealed volume

$$P_{\mathcal{A}_{\mathcal{O}_{seen}}(i)} = \frac{V_{\mathcal{A}_{\mathcal{O}_{seen}}(i)}}{V_{\mathcal{O}_{seen}}} \tag{3}$$

2.2 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

[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 volumes.

other visible objects, and also by the occluded volume, which the manipulator is forbidden to enter.

Definition 4 (*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 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 $T_{\mathtt{A}}$ by estimating the time necessary to execute the trajectory on the robot. Since there is only a single action for each object, $T_{\mathtt{A}}$ is constant for a given scene and does not depend on the sequence in which objects are removed.

3 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. Using this intuition, we define the utility of an object similar to the utility measures defined for sensor placement (Ye and Tsotsos 1995; Espinoza et al. 2011).

Definition 5 (*Utility*) The *utility* of an object A is given by

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

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

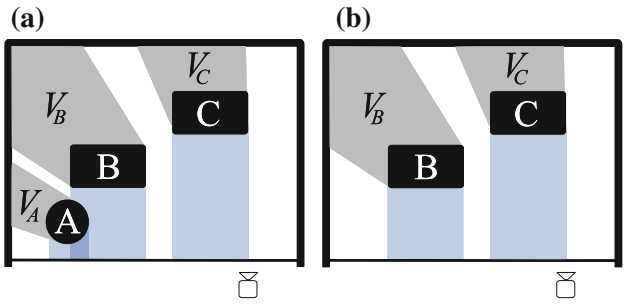

Fig. 3 A scene where the greedy algorithm performs suboptimally due to an accessibility constraint. **a** Initial scene, **b** *A* removed

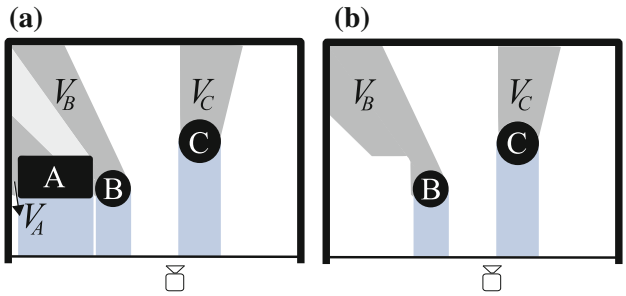

Fig. 4 A scene where the greedy algorithm performs suboptimally due to a visibility constraint. **a** Initial scene. **b** *A* removed

We present a second example of greedy's suboptimality in Fig. 4. In this scene, all objects are accessible, $V_{\mathtt{C}} > V_{\mathtt{A}}$, and $V_{\mathtt{C}} > V_{\mathtt{B}}$. 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 with A is removed. Hence, the optimal arrangement is A → B → C because it quickly reveals the large volume jointly occluded by A and B.

The examples in Figs. 3 and 4 may suggest a k-step lookahead planner for optimality. However, the problem is fundamental: one can 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 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 ith and $(i+1)$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 Eqs. 1 and 3, we can compute the difference between $E(\mathcal{A})$ and $E(\mathcal{A}^*)$ to be:

$$E(\mathcal{A}) - E\left(\mathcal{A}^*\right) = V_{\mathcal{A}^*(i)} \cdot T_{\mathcal{A}^*(i+1)} - V_{\mathcal{A}^*(i+1)} \cdot T_{\mathcal{A}^*(i)} \quad (4)$$

$E(\mathcal{A}^*)$ 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 §5 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 §6 that these scenes do occur surprisingly regularly even with randomly generated object poses. However, as we have shown above, the greedy algorithm can also produce arbitrarily suboptimal results.

In the next section we present an algorithm based on A-Star search, which is always optimal but has exponential computational complexity. Then, in §5 we present a new algorithm which approaches the polynomial complexity of the greedy algorithm, yet maintains optimality in the general case as shown by our empirical evaluations in §6.

4 A-star search algorithm

In this section we present an optimal algorithm for solving the object search by manipulation problem. We first formulate the problem as a deterministic single-source shortest path problem. We then find the optimal solution by executing an A-Star search with an admissible heuristic.

Formulating this problem as a deterministic single-source shortest path problem is possible only because of special structure in the problem. The optimal policy always removes objects from the scene in a deterministic order (i.e. an arrangement) until the target is found. See §10 for a derivation of this fact from a formulation of the problem as a Markov decision process.

4.1 Single-source shortest path problem

Define a directed acyclic graph $G = (N, E, c)$ with nodes $N \subseteq 2^{\mathcal{O}_{seen}}$, edges $E \subseteq \mathcal{O}_{seen} \times \mathcal{O}_{seen}$, and cost function $c : E \rightarrow \mathbb{R}^+$. A node $s \in N$ is the set visible objects remaining in the scene. The directed edge $(s, s \setminus \{a\}) \in E$ removes object a from scene s.

Consider the single-source shortest path problem in G with $\mathcal{O}_{seen}$ as the start node and $\emptyset$ as the goal node. Let edges exist between nodes s and s' if they differ by a single object and if that object is accessible in s. Every path from the start to the goal removes all objects from the scene in a different order and corresponds to a different arrangement.

Furthermore, each edge $(s, s \setminus \{a\}) \in E$ has cost

$$c(s, s') = \left(\frac{V_{a|s}}{V_{\mathcal{O}_{seen}}}\right) T_{\mathcal{O}_{seen} \setminus s'}$$

where $T_{\mathcal{O}_{seen} \setminus s'}$ is the total time required to reach $s' = s \setminus \{a\}$ from the initial scene $\mathcal{O}_{seen}$. For every path going through an edge, the cost indicates the probability that the target will be revealed at that edge multiplied with the time required to reach and execute the edge.

The sum of the edge costs along any path from the start to the goal is exactly Eq. (1). In other words, the cost of a path in this graph is equal to the expected time to find the target while following corresponding arrangement. Therefore, the minimum-cost in G corresponds the optimal arrangement $\mathcal{A}^*_{\mathcal{O}_{seen}}$ that minimizes $E(\mathcal{A}^*_{\mathcal{O}_{seen}})$.

4.2 Admissible heuristic

The single-source shortest path problem can be solved using several well-known algorithms. We use the A-Star search algorithm (Hart et al. 1968) with an admissible heuristic to efficiently find the optimal solution. A-Star is optimal if its heuristic is *admissible*, i.e. does not overestimate the cost from a state to the goal.

Suppose we start at the arbitrary node $s = \{a_1, a_2, \ldots, a_n\}$ and the minimum-cost path to the goal is given by the sequence of actions $[a_1, a_2, \ldots, a_n]$. Then, the optimal cost-to-go is

$$
\begin{aligned}
h^*(s) &= c(s, s \setminus a_1) + c(s \setminus a_1, s \setminus \{a_1, a_2\}) + \ldots b \\
&\quad + c(s \setminus \{a_1, a_2, \ldots, a_{n-1}\}, \emptyset) \\
&= \left(\frac{V_{a_1|s}}{V_{\mathcal{O}_{seen}}}\right)(T_{\mathcal{O}_{seen}\setminus s} + T_{a_1}) \\
&\quad + \left(\frac{V_{a_2|s\setminus a_1}}{V_{\mathcal{O}_{seen}}}\right)(T_{\mathcal{O}_{seen}\setminus s} + T_{a_1} + T_{a_2}) \\
&\quad + \ldots b \\
&\quad + \left(\frac{V_{a_n|s\setminus\{a_1,\ldots c,a_{n-1}\}}}{V_{\mathcal{O}_{seen}}}\right)(T_{\mathcal{O}_{seen}\setminus s} + T_{a_1} \\
&\quad + \ldots b + T_{a_n}).
\end{aligned}
$$

Now we present our heuristic,

$$
\begin{aligned}
h(s) &= \left(\frac{V_{a_1|s}}{V_{\mathcal{O}_{seen}}}\right)(T_{\mathcal{O}_{seen}\setminus s} + \min_{a\in s} T_a) \\
&\quad + \left(\frac{V_{a_2|s\setminus a_1}}{V_{\mathcal{O}_{seen}}}\right)(T_{\mathcal{O}_{seen}\setminus s} + \min_{a\in s} T_a) \\
&\quad + \ldots b \\
&\quad + \left(\frac{V_{a_n|s\setminus\{a_1,\ldots c,a_{n-1}\}}}{V_{\mathcal{O}_{seen}}}\right)(T_{\mathcal{O}_{seen}\setminus s} + \min_{a\in s} T_a) \\
&= \left(\frac{V_s}{V_{\mathcal{O}_{seen}}}\right)\left[T_{\mathcal{O}_{seen}\setminus s} + \min_{a\in s} T_a\right].
\end{aligned}
$$

Since $h(s) \leq h^*(s)$, $h(s)$ is admissible. Intuitively, $h(s)$ optimistically reasons that: (1) we execute the minimum-time action and (2) it reveals all of the remaining volume. Since $h(s)$ is admissible, A-Star is optimal and will return the minimum-cost path.

4.3 Computational complexity

Running an A-Star search on a graph with n nodes and m edges has a worst-case complexity of $O((m+n)\log n)$. Unfortunately, the graph constructed for a scene of size $n = |\mathcal{O}_{seen}|$ has up to 2^n nodes and no more than $n2^n$ edges, resulting in a worst-case complexity of $O(n^2 2^n)$. Our experimental results (§6) confirm that A-Star approaches this worst-case bound in practice and it is intractable to run this algorithm on large scenes.

5 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.

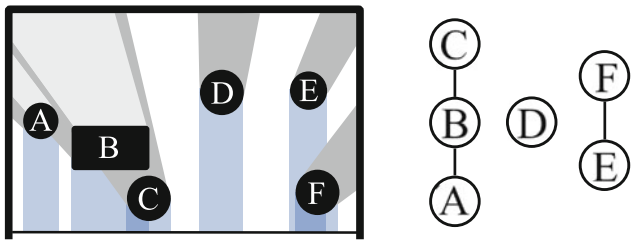

Fig. 5 *Left* an example scene. Volumes occluded by a single object are shown in *dark gray*, joint occlusions are shown in *light gray*, and swept volumes are shown in *light blue*. *Right* The corresponding graph with three connected components (Color figure online)

Algorithm 1: Object Search With Connected Components

```
1  {c^1, c^2, ..., c^m} ← FindConnectedComponents;
2  foreach connected component c^i do
3      A*_{c^i} ← AStar(c^i);
4  A*_{O_seen} ← [ ];
5  repeat
6      bag ← ∅;
7      foreach component arrangement A*_{c^i} do
8          for j ← 1 to |c^i| do
9              bag.Add( A_{c^i}(1, j) );
       seq ← arg max_{A∈bag} U(A);
10
11     Add seq to the end of A*_{O_seen};
12     Remove seq from the A*_{c^i} it belongs;
13 until all objects are in the plan;
14 return A*_{O_seen};
```

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

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 (Hopcroft and Tarjan 1973). 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 perform an optimal search, e.g. using A-Star, to solve the arrangement problem for a connected component independently and then merge the solutions to produce a complete arrangement of the scene.

It is non-trivial to merge arrangements of multiple connected components. The complete plan may switch from one connected component to the other and then switch back to a previous component. Our algorithm provides an efficient greedy way to perform this merge.

The examples in Figs. 3 and 4 show that the utility of a single object is not informative enough to achieve general optimality with a greedy algorithm. Instead, we consider the utility of removing multiple objects from the scene.

Definition 6 (*Collective Utility*) The *collective utility* of a set of objects o is given by

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

A general greedy approach which considers the collective utility of all possible sequences of all sizes in the scene would quickly become infeasible as the number of such sequences is $O(|o|!)$. In our case, we take advantage of the fact that we have optimal plans for each connected component in which the objects are already sorted. We then need to compute collective utilities of only the prefixes (i.e. the first k objects where k ranges from 1 to the size of the connected component) of these optimal sequences.

We present our algorithm in Algorithm 1 that uses the collective utility of sequences from connected components to generate an arrangement of the complete scene. It first identifies the connected components in the scene (Line 1). Then it finds the optimal arrangement internal to a connected component using A-Star search (Line 3). It then merges these arrangements iteratively by finding the maximum utility[2] prefixes of the optimal arrangements of the connected components.

In §6 we show that Algorithm 1 generates the optimal result in all scenes we tried it on and it uses a fraction of the time A-Star requires on the complete scene. We present a partial proof of our algorithm's optimality in the appendix.

5.1 Complexity of the connected components algorithm

The connected components algorithm divides the set of objects into smaller sets, runs A-Star on each connected component, and then merges the plans for each component. 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-Star 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-Star, 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.

[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 recursively.

6 Experiments and results

We investigated the performance of the different algorithms through extensive experiments in simulation and on a real robot. We implemented the greedy, A-Star, and connected components algorithms in OpenRAVE (Diankov and Kuffner 2008). 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 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.

In our implementation, we assume that the `target` rests stably on the workspace and $C = SE(2)$. We approximate C with a discrete set of configurations $\tilde{C} \subset C$. Next, we compute a discrete approximation of each set of revealed configurations $\tilde{C}_{o|s} = \{x \in \tilde{C} : \Omega(x|s \setminus o) \wedge \neg\Omega(x|s)\}$ using the visibility criterion $\Omega(x|s)$ that returns whether $x \in C$ is visible in the scene s. Finally, we compute $V_{o|s}$ by approximating the integral over $C_{o|s}$ in Eq. (2) with a summation over $\tilde{C}_{o|s}$. In principle, our framework supports any deterministic visibility criterion $\Omega : C \times 2^{\mathcal{O}_{seen}} \rightarrow \{0, 1\}$. However partial views of objects are difficult to detect in practice. Therefore, our implementation considers the target at a certain pose visible if and only if it is entirely visible. This is implemented by sampling points on the surface of the target, raytracing from the sensor to each point, and verifying that no rays are occluded.

We present results from scenes with 4, 6, 8, 10, and 12 objects in Fig. 6 along with the 95 % confidence intervals. We conducted 400 simulations for each different number of objects, resulting a in total of 2,000 different scenes. The data in Fig. 6a 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-Star comes with the cost of exponential complexity in the number of objects. This complexity causes the planning time of A-Star to dominate the other planning times shown in Fig. 6b (note the logarithmic scale).

While still optimal in all 2,000 scenes, the connected components algorithm achieves much lower planning times than A-Star. By running A-Star 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. 6c 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. 6b. With fixed com-

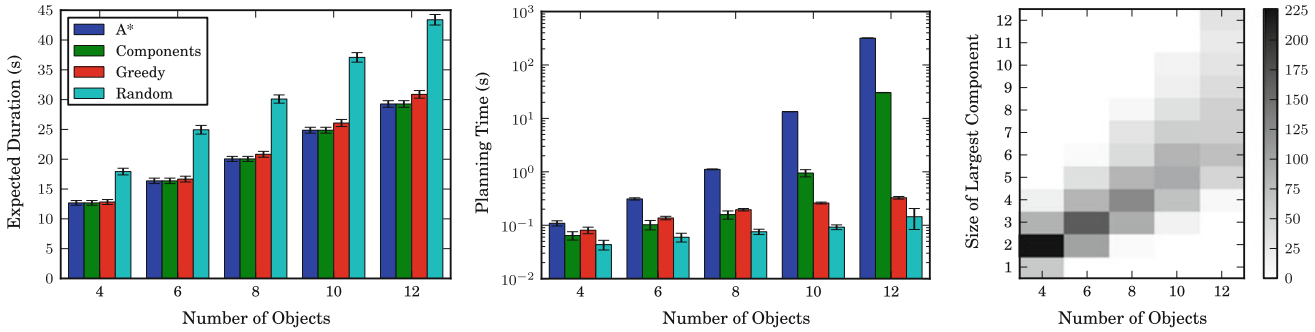

Fig. 6 Performance of the random, greedy, A-Star, and connected component planners as a function of number of objects. All results are averaged over 400 random scenes and are plotted with their 95 % confidence interval. The planning times are presented in log-scale, where the confidence intervals are also plotted as log-scale relative errors (Baird 1995). The relationship between scene size and the size of the largest connected component is also plotted as a two-dimensional histogram

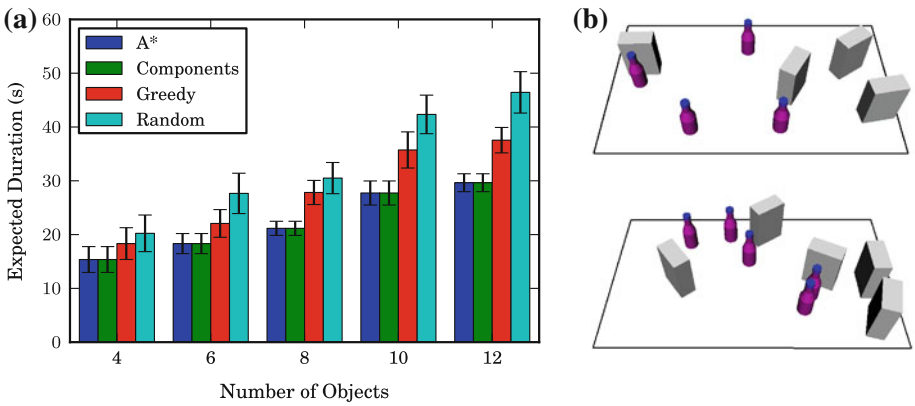

Fig. 7 **a** 95th percentile of expected time to find the target. **b** Two example scenes where greedy performed poorly. The *black lines* denote the workspace boundary. **a** Expected search duration. **b** Example scenes

putational 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-Star 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 remarkably well on average. We found that in 50 % of the 2,000 different scenes, the greedy algorithm produced the optimal sequence. Our explanation for greedy's performance is that the geometry 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, which makes it unlikely that multiple objects can be placed in front of it.

To see the greedy's worst-case behavior, we plotted the expected time to find the target for the 5 % of scenes where greedy performed worst in Fig. 7a. Across all the scenes, the worst performance was 2.04 times the expected duration of the optimal sequence. We show two example scenes where greedy performs poorly in Fig. 7b. Both scenes include small bottles blocking access to large boxes. There is very little volume hidden behind the bottles, so the boxes are—suboptimally—removed late in the plan.

If the goal is to minimize the total time to plan and also execute the search, then we must trade-off the gains in planning time achieved by the greedy algorithm with the extra actions the robot needs to execute due to the greedy algorithm's suboptimality. In a setting where action executions are fast and greedy is nearly optimal, one should use the greedy algorithm. If action executions are slow and greedy plans are increasingly suboptimal (e.g. in environments with a large number of objects), one should use the connected-components algorithm.

6.1 Real robot implementation

We implemented the greedy and connected components algorithms on our robot HERB. We used HERB's camera and the MOPED (Martinez et al. 2010) 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. 8. In this scene the target object, a battery

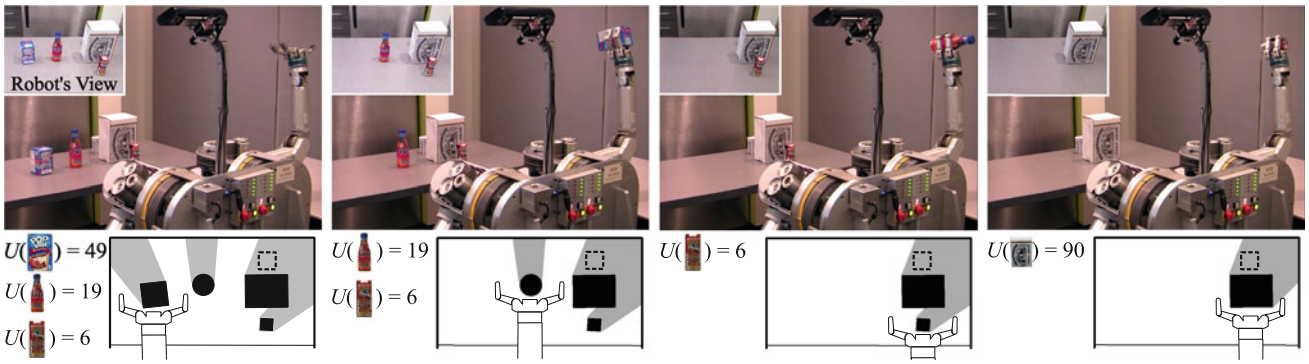

Fig. 8 Greedy planner. We present the utility of all accessible objects at each step. The pose of the target (unknown to the robot) is marked with dashed lines in the illustration

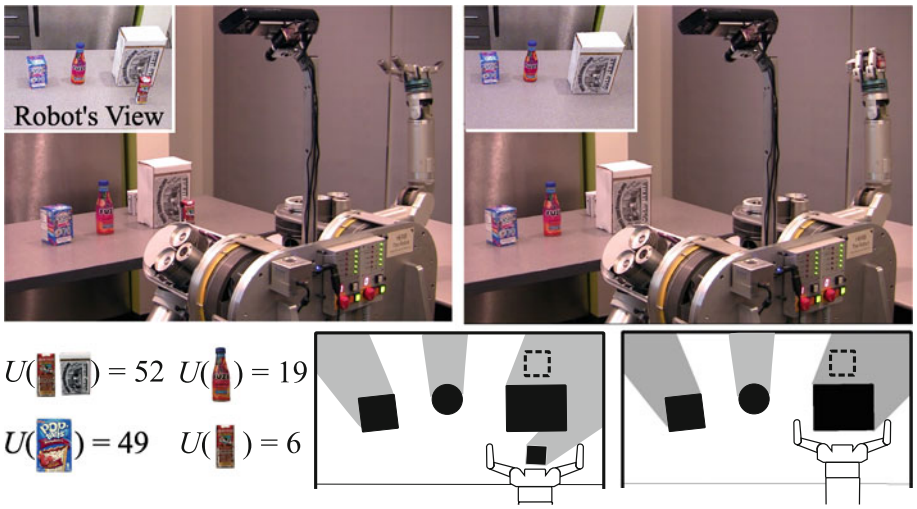

Fig. 9 Connected-components planner. The utilities of all prefixes from each connected component are presented at each step

pack, is hidden behind the large box, which also occludes the largest volume. Since the large box is inaccessible, the greedy 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. 9 the scene is the same but HERB uses the connected components algorithm. There are three connected components in this scene {`BlueBox`}, {`Bottle`}, and {`LargeBox`, `SmallBox`}. The connected components algorithm considers the collective utilities of multiple objects from each connected component, including both U(`SmallBox`) and U(`SmallBox`, `LargeBox`). The utility of `SmallBox` is very small compared with the other immediately accessible objects, but combined with the `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 at http://youtu.be/i06GBj1iDOo.

6.2 Performance in human environments

Objects are not distributed randomly in real human environments: they display a structure specific to human clutter. We conducted a simple evaluation of our planner by creating scenes which are similar in structure to human clutter.

For this evaluation we identified three different places where a robot might need to search for an object by manipulation: a bookshelf, a cabinet, and a fridge. We captured images of the natural clutter in these environments in our lab. We display these images as the leftmost column in Fig. 10.

Limitations of our robot's perception system and the difference between the sizes of a human arm/hand and our robot's manipulator prevented us from running our planner directly on these scenes. Therefore, we constructed new scenes that are scaled up to the dimensions of HERB's manipulator and consist of objects that our perception system can reliably detect. We attempted to faithfully mimic the relative size and configuration of objects in the original scenes as

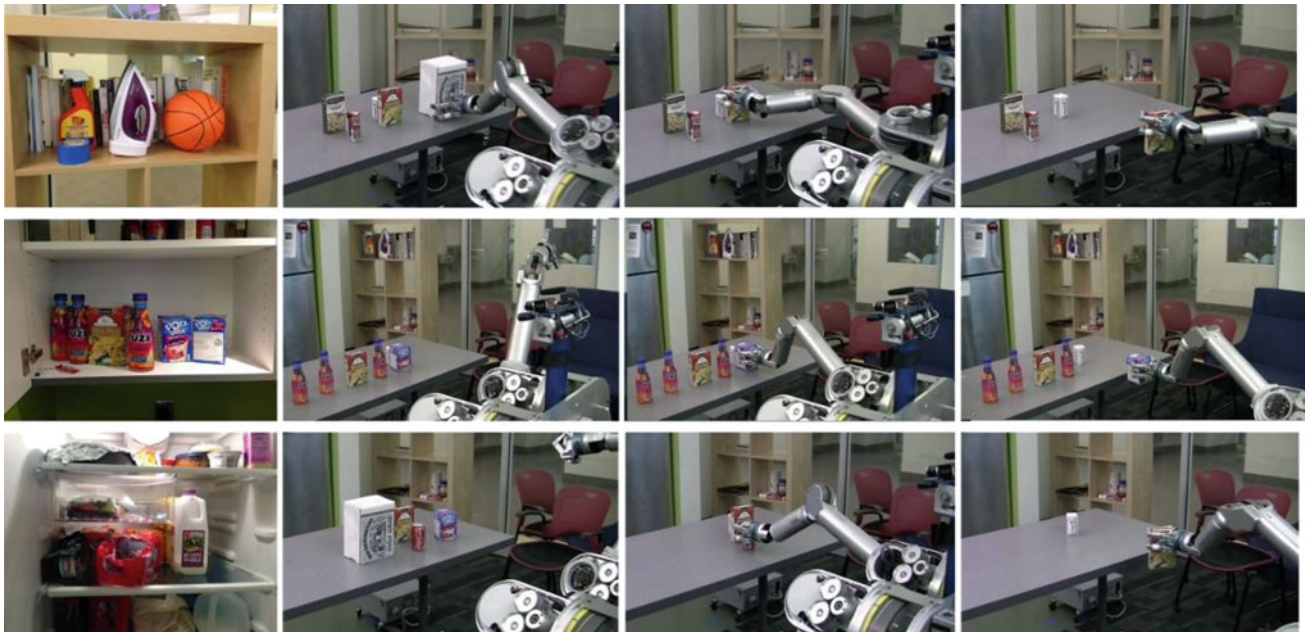

Fig. 10 Example executions on scenes inspired by real human environments. Scenes are inspired from a cluttered human shelf (*top*), a cabinet (*center*), and a fridge (*bottom*)

Table 1 Planning and execution times

	Total time (s)	Planning (s)	Execution (s)
Shelf	132.7	16.1	116.6
Cabinet	94.6	26.1	68.5
Fridge	242.0	16.7	225.3

much as possible. We hid the target object randomly in the occluded portion of the table.

We present snapshots from our robot's search for the target as the rows of Fig. 10. All plans were automatically generated by the connected components algorithm and HERB successfully found the target in all three scenes. Table 1 shows the planning time, execution time, and the total time it took the robot to find the object. Note that execution time is the dominating factor, emphasizing the importance of generating short plans when searching for objects with a real robot.

7 Planning to place objects

The robot must place an object down before picking up another one. If the robot is allowed to place the object at a new pose where it creates new visibility or accessibility relations, then the object search problem becomes a version of *reconfiguration planning* which is known to be NP-Hard (Wilfong 1988). We avoid this complexity by placing objects only at poses that do not create new accessibility or visibility relations.

Our formulation requires us to compute the time it takes to manipulate an object before we decide the arrangement. We use fixed placement poses on a nearby empty surface to satisfy this constraint. We found this to be a reasonable strategy in practice: Even when the robot is working in a densely crowded cabinet shelf, there is usually a nearby counter or another shelf to place objects on.

However, one can also *re-use* the explored space to place objects: after an object is picked up and the robot sees the volume behind that object, the planner can safely use this volume. In particular, for an arrangement $\mathcal{A}_{\mathcal{O}_{seen}}$, object $\mathcal{A}_{\mathcal{O}_{seen}}(i)$ can be placed where:

- it avoids penetrating $V_{\mathcal{A}_{\mathcal{O}_{seen}}(i+1,|\mathcal{O}_{seen}|)}$,
- it avoids occluding $V_{\mathcal{A}_{\mathcal{O}_{seen}}(i+1,|\mathcal{O}_{seen}|)}$,
- it avoids blocking access to the objects $\mathcal{A}_{\mathcal{O}_{seen}}(i+1,|\mathcal{O}_{seen}|)$,
- it avoids colliding the placement poses of the objects $\mathcal{A}_{\mathcal{O}_{seen}}(1,i-1)$.

In Fig. 11 we illustrate each of these constraints and the remaining feasible placement poses for an object.

Surprisingly, certain scene structures lead to very simple and fixed placement strategies. For example, in a scene where there are no accessibility relations and no joint occlusions (where the greedy algorithm is optimal), an object can be placed where it was picked up: the robot lifts an object, looks behind it, and places it back. This strategy respects the constraints listed above.

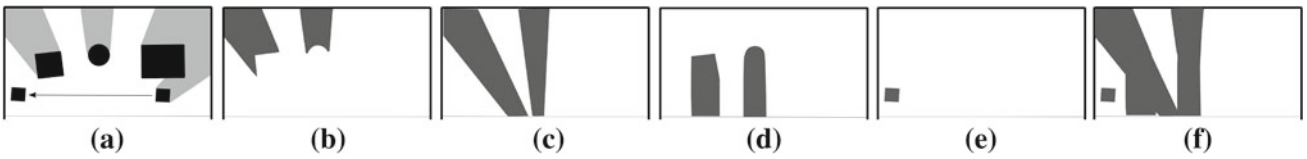

Fig. 11 An example illustrating placement contraints. **a** In this scene the *small box* is moved to the *left* and then the *large box* is picked up. Now the planner must place the *large box*. The *large box cannot* be placed where **b** it will penetrate the volume which is not explored yet; **c**it will occlude the volume which is not explored yet; **d** it will block access to the objects which are not moved yet; **e** it will collide with the new poses of the objects which are already moved. **f** The combined placement constraints for the *large box*

8 Encoding semantic knowledge

All of our examples of the object search by manipulation problem assume that the target is equally likely to be found anywhere in the workspace. However, human environments are not random: semantic knowledge about the environment provides useful information about where a hidden object may be found. For example, a can of soda is more likely to be in the refrigerator than in the dish washer.

There are several existing techniques for learning co-occurrence probabilities objects (Kollar and Roy 2009; Wong et al. 2013) in real environments. We can naturally incorporate this type of semantic knowledge into the prior distribution $P_0(x)$. The prior distribution changes the volume revealed by each target and can be directly exploited by same the greedy, A-Star, and connected components algorithms described above.

For example, suppose that the robot is searching for a bottle of mustard in the refrigerator. Fig. 12a shows an example of this scenario where the refrigerator contains a small bottle of ketchup K and two large food containers A and B. The mustard M is hidden behind the bottle of ketchup. Assuming that the robot has no prior over the location of the mustard, $V_{\mathtt{A}}, V_{\mathtt{B}} \gg V_{\mathtt{K}}$. Assuming $T_{\mathtt{A}} = T_{\mathtt{B}} = T_{\mathtt{K}}$, the optimal arrangement with no prior distribution is $\mathtt{A} \rightarrow \mathtt{B} \rightarrow \mathtt{K}$ because $U(\{\mathtt{A}, \mathtt{B}\}) > \mathtt{U(K)}$ and $U(\mathtt{B}) > U(\mathtt{K})$.

However, this plan ignores important semantic knowledge about the scene: mustard is more likely to be found near ketchup than the containers of food. This knowledge causes $P_0(x)$, shown in Fig. 14b, to be peaked around K. This prior influences the revealed volumes such that $V_{\mathtt{K}} \gg V_{\mathtt{A}}, V_{\mathtt{B}}$, the opposite relationship as above. This prior knowledge changes the optimal arrangement to $\mathtt{K} \rightarrow \mathtt{A} \rightarrow \mathtt{B}$ because $U(\mathtt{K}) > \mathtt{U}(\{\mathtt{A}, \mathtt{B}\})$ and reveals the mustard more quickly.

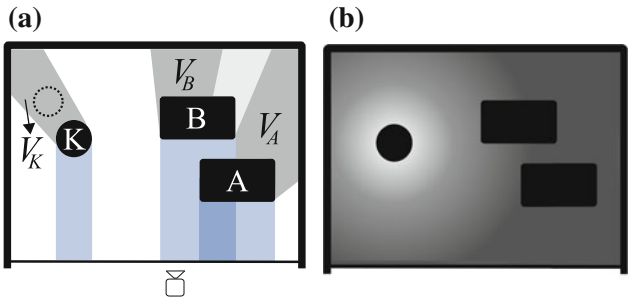

Fig. 12 Example scene where semantic knowledge influences the optimal order of removing objects. The robot is searching for the mustard (*dotted circle*) and has a strong prior distribution that it is near the ketchup (K). **a** Refrigerator scene. **b** Prior distribution $P_0(x)$

9 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. However, objects must be smaller than the target object to avoid the 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 and the plans are generated using an optimal algorithm in each updated scene. 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. 13 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

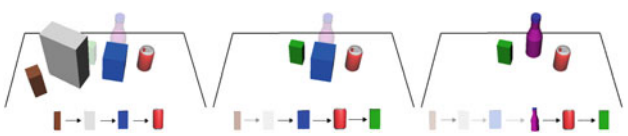

Fig. 13 Example of replanning on a scene with two hidden objects. Each replanning stage is shown as a separate frame along with the corresponding plan. Hidden objects are shown as semi-transparent and the workspace bounds are indicated by a *black line*

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. This time, order of the optimal sequence is changed by the addition of the hidden object and it would be suboptimal to continue executing the previous plan.

10 Future work

We are excited about exploring this problem deeper and relaxing some of the simplifying assumptions in future work.

Integrated motion planning We use a motion planner for the manipulator that is conceptually decoupled from finding the optimal arrangement. However, there are aspects of the object search problem that can be integrated into the motion planning process. For example, there may be multiple trajectories for grasping an object that require differing numbers of objects to be moved out of the way. In this respect, we are excited about studying how a more complex motion planner, e.g. one returning a minimum-constraint violation trajectory (Hauser 2012), can be integrated into our system. The object search formulation can also take into account the motion of moving from one object to the next one, trying to minimize the time spent in between. This can make the object search problem similar to the *traveling agent problem* (Moizumi and Cybenko 2001) where the latencies between nodes produced are produced by a motion planner.

Improved 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.

Integrated sensor planning Aside from reaching to objects, the robot does not move its base in our current implementation. Through combining the ability of search by manipulation with sensor planning, the robot can find targets faster. Sensor planning would include working with multiple camera poses and planning for the base when searching for a target in a larger environment.

Acknowledgments Special thanks to the members of the Personal Robotics Lab at Carnegie Mellon University for insightful comments and discussions. This material is based upon work supported by NSF-IIS-0916557 and NSF-EEC-0540865.

Appendix

Appendix 1: Formulation as a Markov decision process

In this section, we formulate the object search problem as a Markov decision process (MDP) and show that the optimal policy minimizes the expected time to find the target. Next, we show that the MDP is deterministic and that each policy corresponds to an arrangement. This provides insight into how the A-Star search algorithm can be used to find the optimal policy for a stochastic problem.

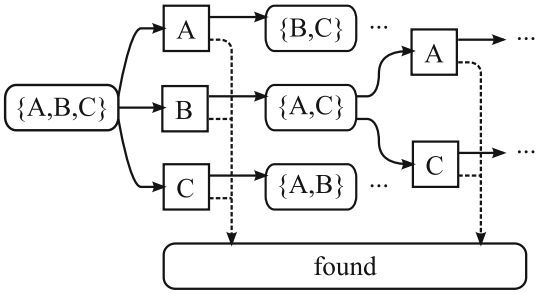

Fig. 14 Transition graph of the of the object search MDP. States (*rounded rectangles*) are sets of visible objects and actions (*squares*) correspond to removing an object from the scene. Each action has a probability of revealing `target` and transitioning into `found`

A MDP is the four-tuple (S, A, Γ, R) where S is the state space, A is the action space, $\Gamma(s'|s, a)$ is the transition model, and $R(s, a)$ is the reward function (Kaelbling et al. 1996).

For the object search problem, the state $s \in 2^{\mathcal{O}_{seen}} \cup \{\texttt{found}\}$ is the set of visible objects remaining in the scene and an absorbing goal state `found` that corresponds to the target being visible. The scene starts with all objects present $s_0 = \mathcal{O}_{seen}$ and the robot sequentially chooses an object $a \in s$ to remove. After removing a, we transition to the successor state s' according to

$$\Gamma(s'|s,a) = \begin{cases} 1 - \frac{V_{a|s}}{V_s} & : s' = s \setminus \{a\} \\ \frac{V_{a|s}}{V_s} & : s' = \texttt{found} \end{cases}$$

and receive reward $R(s, a) = -T_a$. This process continues until there is a transition into `found` and the MDP terminates with zero reward. Fig. 14 shows a graphical depiction of the state transition graph for a scene with three objects {`A`, `B`, `C`} and no accessibility relations.

A policy $\pi : S \rightarrow A$ specifies which action $\pi(s)$ to take when in state s. In the case of the object search problem, π dictates which object to remove from the scene at each step. We wish to find the optimal policy π^* that maximizes the sum of expected future reward $E[\sum_{t=1}^{|\mathcal{O}_{seen}|} R(s_t, a_t)]$.

Any policy π induces the value function $\Lambda^\pi : S \rightarrow \mathbb{R}$, where $\Lambda^\pi(s)$ is the sum of expected future reward from starting in state s and following π to termination.[3] The value function of the optimal policy π^* satisfies the Bellman equation

[3] We use Λ^π in place of V^π to denote the value function to avoid confusion with revealed volume V_s.

$$\Lambda^{\pi^*}(s) = \max_{a \in A} \sum_{s' \in S} \Gamma(s'|s,a) \left[R(s,a) + \Lambda^{\pi^*}(s') \right], \quad (5)$$

which recursively relates the value of state s with that of its successors (Kaelbling et al. 1996).

We can take advantage of the structure of the object search problem to reduce the Bellman equation to a simpler form. First, the object search problem has a sparse transition model: $\Gamma(s'|s,a) = 0$ for all $s' \notin \{s \setminus a, \texttt{found}\}$. Second, $\Lambda^{\pi}(\texttt{found}) = 0$ for the absorbing goal state. Using these observations, we can simplify Eq. (5) to

$$\begin{aligned}\Lambda^{\pi^*}(s) &= \max_{a \in s} \Big[R(s,a) + \Gamma(s \setminus \{a\}|s,a) \Lambda^{\pi^*}(s \setminus \{a\}) \\ &\quad + \Gamma(\texttt{found}|s,a) \Lambda^{\pi^*}(\texttt{found}) \Big] \\ &= \max_{a \in s} \left[-T_a + \left(1 - \frac{V_{a|s}}{V_s}\right) \Lambda^{\pi^*}(s \setminus \{a\}) \right],\end{aligned}$$

which, surprisingly, has the same form as the value function of a deterministic MDP. This agrees with our earlier intuition: while the outcome the object search problem is stochastic, the optimal order of removing objects is completely deterministic.

In fact, π is equivalent to an arrangement $\mathcal{A}^{\pi}_{\mathcal{O}_{seen}}$ that specifies an open-loop order in which to remove objects. During execution objects are removed according to $\mathcal{A}^{\pi}_{\mathcal{O}_{seen}}$ until the target is revealed and the robot halts (i.e. transitions to found). This equivalence is what enables us to formulate the object search problem as a deterministic search in §4.

Appendix 2: Optimality of connected components

We present a partial proof of optimality for Algorithm 1.

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

Lemma 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 2 *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 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 and leave its proof to future work.

Lemma 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*

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

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

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

Theorem 3 *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 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:

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

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,

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

and also

$$U(\mathcal{A}_{c^*}(1, j^*)) > U(\mathcal{A}_{c'}(1, j')) \tag{7}$$

Using Lemma 1 and Eq. (6), we get

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

Then from Eq. (7),

$$U(\mathcal{A}_{c^*}(k, j^*)) > U(\mathcal{A}_{c'}(1, j')) \tag{8}$$

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 2):

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

which contradicts Eq. (8). □

References

Anand, A., Koppula, H. S., Joachims, T., & Saxena, A. (2013). Contextually guided semantic labeling and search for three-dimensional point clouds. *International Journal of Robotics Research*, *32*(1), 19–34.

Baird, D. (1995). *Experimentation: An introduction to measurement theory and experiment design*. Englewood Cliffs: Prentice-Hall.

Ben-Shahar, O., & Rivlin, E. (1998). Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation*, *14*, 549–565.

Chen, P., & Hwang, Y. (1991). *Practical path planning among movable obstacles*. IEEE: In IEEE International Conference on Robotics and Automation.

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). *Computational geometry: Algorithms and applications*. Berlin: Springer.

Diankov, R., & Kuffner, J. (2008). OpenRAVE: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute.

Dogar, M., & Srinivasa, S. (2012). A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, *33*(3), 217–236.

Espinoza, J., Sarmiento, A., Murrieta-Cid, R., & Hutchinson, S. (2011). Motion planning strategy for finding an object with a mobile manipulator in three-dimensional environments. *Advanced Robotics*, *25*, 1627–1650.

Gupta, M., & Sukhatme, G. (2012). Interactive perception in clutter. In R. S. S. The (Ed.), *2012*. Workshop on Robots in Clutter: Manipulation, Perception and Navigation in Human Environments.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

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

Hopcroft, J., & Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, *16*(6), 372–378.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Researc*, *4*, 237–285.

Kaelbling, L., & Lozano-Perez, T. (2012). *Unifying perception, estimation and action for mobile manipulation via belief space planning*. IEEE: In IEEE International Conference on Robotics and Automation.

Kollar, T., & Roy, N. (2009). *Utilizing object-object and object-scene context when planning to find things*. IEEE: In IEEE International Conference on Robotics and Automation.

Ma, J., Chung, T. H., & Burdick, J. (2011). A probabilistic framework for object search with 6-dof pose estimation. *International Journal of Robotics Research*, *30*(10), 1209–1228.

Martinez, M., llet, A., & Srinivasa, S. (2010). MOPED: A scalable and low latency object recognition and pose estimation system. In *IEEE International Conference on Robotics and Automation*. IEEE.

Moizumi, K., & Cybenko, G. (2001). The traveling agent problem. *Mathematics of Control, Signals, and Systems*, *14*(3), 213–232.

Ota, J. (2009). Rearrangement planning of multiple movable objects by a mobile robot. *Advanced Robotics*, *23*, 1–18.

Overmars, M. H., Nieuwenhuisen, D., Nieuwenhuisen, D., Frank, A., & Overmars, H. (2006). An effective framework for path planning amidst movable obstacles. In *International Workshop on the Algorithmic Foundations of Robotics*.

Shubina, K., & Tsotsos, J. (2010). Visual search for an object in a 3D environment using a mobile robot. *Computer Vision and Image Understanding*, *114*, 535–547.

Sjo, K., Lopez, D., Paul, C., Jensfelt, P., & Kragic, D. (2009). Object search and localization for an indoor mobile robot. *Journal of Computing and Information Technology*, *17*, 67–80.

Srinivasa, S., Berenson, D., Cakmak, M., Dogar, M., Dragan, A., Knepper, R. A., et al. (2012). Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, *100*(8), 1–19.

Stilman, M., Schamburek, J. U., Kuffner, J., & Asfour, T. (2007). Manipulation planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*. IEEE.

van den Berg, J. P., Stilman, M., Kuffner, J., Lin, M. C., & Manocha, D. (2008). Path planning among movable obstacles: A probabilistically complete approach. In *International Workshop on the Algorithmic Foundations of Robotics* (pp. 599–614).

van Hoof, H., Kroemer, O., Amor, H. B., & Peters, J. (2012). Maximally informative interaction learning for scene exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5152–5158). IEEE.

Wilfong, G. (1988). Motion planning in the presence of movable obstacles. In *Proceedings of the Fourth Annual Symposium on Computational Geometry* (pp. 279–288).
Wong, L. L., Kaelbling, L. P., & Lozano-Pérez, T. (2013). Manipulation-based active search for occluded objects. In *IEEE International Conference on Robotics and Automation*. IEEE.
Ye, Y., & Tsotsos, J. (1995). Where to look next in 3D object search. In *IEEE International Symposium on Computer Vision*. IEEE.
Ye, Y., & Tsotsos, J. (1999). Sensor planning for 3D object search. *Computer Vision and Image Understanding*, *73*(2), 145–168.

Mehmet R. Dogar is a Ph.D. student at the Robotics Institute at Carnegie Mellon University where he is a member of the Personal Robotics lab. His interests include physics-based robotic manipulation, manipulation in cluttered environments, and grasping. Mehmet received his B.S. and M.S. degrees from Middle East Technical University, Turkey, in Computer Engineering.

Michael C. Koval is a Ph.D. student in the Robotics Institute at Carnegie Mellon University, where he is a member of the Personal Robotics Lab. His interests include planning under uncertainty, incorporating sensor feedback into manipulation primitives, and physics-based manipulation. Michael has B.S. degrees in Electrical and Computer Engineering and Computer Science from Rutgers University.

Abhijeet Tallavajhula is a first year Ph.D. student at the Robotics Institute at Carnegie Mellon University. He has a B.Tech degree in Mechanical Engineering from the Indian Institute of Technology, Kharagpur. His interests include motion planning and control.

Siddhartha S. Srinivasa is an Associate Professor at the Robotics Institute at Carnegie Mellon University where he founded and directs the Personal Robotics Lab. His research focuses on manipulation, with the goal of enabling robots to robustly and gracefully interact with the world to perform complex manipulation tasks in uncertain, unstructured, and cluttered environments. His current research focuses on physics-based nonprehensile manipulation for reconfiguring clutter, functional gradient methods for motion planning, and formalizing HRI principles using machine learning, motion planning and optimization algorithms. He received his B.Tech. from IIT Madras, and Ph.D. in Robotics from Carnegie Mellon University.