Searching for Physical Objects in Partially Known Environments

Xinkun Nie, Lawson L.S. Wong, Leslie Pack Kaelbling

***Abstract*— We address the problem of a mobile manipulation robot searching for an object in a cluttered domain that is populated with an unknown number of objects in an unknown arrangement. The robot must move around its environment, looking in containers, moving occluding objects to improve its view, and reasoning about collocation of objects of different types, all in service of finding a desired object. The key contribution in reasoning is a Markov-chain Monte Carlo (MCMC) method for drawing samples of the arrangements of objects in an occluded container, conditioned on previous observations of other objects as well as spatial constraints. The key contribution in planning is a receding-horizon forward search in the space of distributions over arrangements (including number and type) of objects in the domain; to maintain tractability the search is formulated in a model that abstracts both the observations and actions available to the robot. The strategy is shown empirically to improve upon a baseline systematic search strategy, and sometimes outperforms a method from previous work.**

I. Introduction

As the perception, locomotion, and manipulation abilities of robots begin to improve, we begin to contemplate constructing robots that can help with household chores or assist in disaster recovery. In open domains such as these, robots will have to be able to operate in cluttered domains and be able to locate objects of interest within them.

In this paper, we address the problem of a mobile manipulation robot searching for an object in a cluttered domain that is populated with an unknown number of objects in an unknown arrangement. The robot must move around its environment, looking in containers, moving occluding objects to improve its view, and reasoning about collocation of objects of different types, all in service of finding a desired object. We do not address issues of low-level perception or of manipulation; rather, we provide a general framework for reasoning about arrangements of unknown objects and for planning how to search effectively for a desired object.

The key contribution in reasoning is a Markov-chain Monte Carlo (MCMC) method for drawing samples of the arrangements of objects in an occluded container, conditioned on previously gathered information of a variety of types. Relevant information includes: spatial knowledge, such as the sizes and shapes of containers such as shelves, historical knowledge, of which objects have already been removed from the containers, type co-occurrence knowledge, which says which types of objects are most likely to occur near one another, and other global constraints in the domain.

We gratefully acknowledge support from NSF grants 1420927 and 1523767, from ONR grant N00014-14-1-0486, and from ARO grant W911NF1410433. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 {xnie,lsw,lpk}@csail.mit.edu

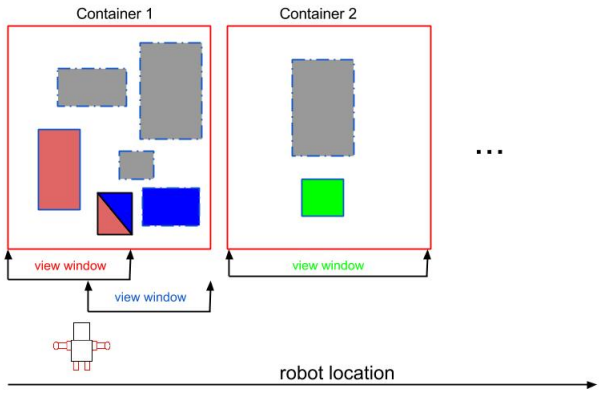

Fig. 1. A robot searching for objects in a 2-D domain. The colored rectangles are the objects that can be observed by the robot if the robot sits within the corresponding view window. The half-blue half-red object can be observed both from the red view window and the blue view window. The grey shaded objects are occluded, and are only visible after objects in the front have been removed. In our problem, the robot is asked to obtain an object of a specific type, and it must choose both where to look and possibly what to remove in order to find the object. When the target object type is occluded, contextual cues such as collocated object types and spatial constraints can be used in an inference process to guide the search.

The key contribution in planning is a receding-horizon forward search in the space of beliefs about (distributions over) arrangements (including number and type) of objects in the domain. In our domain, observations are drawn from a continuous space of object poses, so making this search tractable requires the construction of abstract observation models, which reduce the effective branching factor and number of state samples needed to represent beliefs.

II. Related Work

The object-search problem has recently attracted a significant amount of interest in the indoor service robotics community. The earliest approach framed the problem as active visual search [19], [17], which seeks the next best view at which to place a visual sensor in order to find the desired object. Much subsequent work in robotics has expanded on this formulation; Aydemir *et al.* [1] provides a good overview of the numerous works. Most of the later work attempt to capture additional structural information found in typical household environments, including object-location and object-object co-occurrences [8], [1], spatial relations [9], [11], object affordances [12] and scene ontologies [14]. While from these studies it is clear that additional knowledge is beneficial, most are focused on relatively uncluttered domains. In this case, since it is essentially guaranteed that

there exists some reachable viewing pose that will bring the target within view, only robot *motion* needs to be considered.

For general cluttered environments such as kitchens (imagine searching in the back of a fridge), we must be prepared to *manipulate* objects as well, such as by moving occluding ones out of the way to reveal hidden spaces. This requires a more detailed analysis of the geometry and physical constraints in the world. Our previous work showed that considering how much space is remaining can be useful, but uses greedy action selection [18]. Dogar *et al.* [3] provides a careful analysis of conditions under which greedy search (in terms of revealing the most unexplored space) is optimal, but also present reasonable examples where the conditions are violated. Lin *et al.* [10] also reasoned about object placements, but A* search is used for planning, which requires physical space to be discretized and is difficult to scale. Additionally, the examples shown in these works are significantly less cluttered than the ones we will consider.

To our knowledge, imperfect perception has not been considered within object search. We present a model that is capable of handling erroneous object type detections, and a sampling procedure for exploring the posterior distribution of both object types and their spatial arrangements. It is also capable of conditioning on prior knowledge about the density of objects in containers as well as global domain constraints on the number of instances of different types.

III. Problem Definition and Assumptions

The techniques presented here could be applied more broadly to a variety of single or multi-vehicle teams with arbitrary sensor suites, container sizes and shapes, and object types. For concreteness, we focus on a version of the problem motivated by a robot looking for something in a kitchen.

The kitchen can be thought of as being made up of a set of *containers*, which are spatial regions containing manipulable objects. Containers might be refrigerator shelves, cupboard shelves, kitchen drawers, etc. The number, size, and shape of the containers are known in advance. Objects are rigid and are drawn from a finite universe of known types. The number of objects in the universe is *not* known. For simplicity we assume that the objects on a particular shelf rest only on the shelf and not on one another. This allows us to model each container as a two-dimensional object (as if all the objects are projected down onto the shelf surface). The robot can view the container from the front "edge" and see any objects that in the front layer. This model is illustrated in Fig. 1.

More formally, we model the world as a set of N 2-dimensional containers $c_1, \cdots, c_N$. Each container has a finite set of views, which can be thought of as places from which a robot can observe the container; each view has a corresponding *view window* which may only give a partial view of the front of the container. The l-th view window of container c_n is denoted by v_{nl}. For the purposes of the experiments in this paper, we model objects as axis-aligned rectangles, but this assumption is not crucial; it mainly simplifies collision checking and other geometric routines.

The robot's goal is to find a target object of *query type* t_q by moving among the view windows, observing, and removing objects. The actions it can take are to:

- **Move** to a view point and make an observation;
- **Look** from the current view point;
- **Remove** an object that is currently in view.

An object can be observed and removed if the robot is at a view window from which it can see at least half of the object. The **Move** and **Remove** actions are entirely reliable, and always have their desired effects. This latter assumption of perfect manipulation may seem severe, but robustness in grasping is gradually becoming achievable [6], [2], although currently it is still inefficient and costly. There are no state changes in the domain except those made by the robot. When an object is **Remove**d from its containers, it simply disappears from the world (but the container it once belonged to is remembered for the purposes of performing inference); this is a reasonable model when there is a lot of "temporary storage" space in which to put objects that are removed from containers while looking for other objects. In domains with very little free space, the problem becomes a much more difficult, intricate mobile manipulation problem.

The **Look** action is stochastic. The *nominal observation* from a view point consists of a set of observations of the form (x, y, o), specifying a pose in the plane for the object and an observed type o. An object is observed if at least half of it is visible from the view window; its pose is always correctly observed but there is a probability that the type will be mistaken. That is, for visible objects, their true states are (x, y, τ), where τ and o may be different. We have a predetermined $T \times T$ confusion matrix M that characterizes the error modes of whatever object-recognition method the system is using; its entries are $M_{ij} = \mathbb{P}(o = t_j \mid \tau = t_i)$, the probability of observing type t_j when the true type is t_i.

When the robot executes the **Remove** action, it receives a perfect observation of the type of the object removed; this is a plausible model of the robot's ability to identify the type of an object it is actually holding in its hand (such as by bringing it to an ideal pose for its sensor it detect the type).

The goal of the system is to arrive in a belief state that assigns high probability to the event that an object of the target query type t_q is visible from the robot's current view point (including the case where the robot is holding onto such an object after a **Remove** action).

To achieve this task, the robot must plan a sequence of **Move**, **Look**, and **Remove** actions in the kitchen domain. Since there is uncertainty in the object arrangements, in particular, uncertainty about occluded objects and their types, we model the problem as a partially-observable Markov decision process (POMDP) [7]. We first describe how to represent the belief and perform updates given observations, which is a necessary step both during planning and execution. We then discuss a simple forward-search procedure, as well as abstractions to approximate the infinite observation space.

IV. Belief Representation and Estimation

We generally represent a belief state as a set of samples, because to our knowledge there is no good analytic representation for combining the disparate types of information we have about the state of the world. These samples are drawn from a prior belief or a posterior distribution that is conditioned on some history of previous actions and observations, using a Metropolis-Hastings (MH) sampler (see Gelman *et al.* [4] for background on sampling). The reasoning can be thought of as happening in two phases: first, incorporating prior information about object type co-occurrence and conditioning on observed object types; and second, incorporating prior knowledge about the spatial extent of containers, the amount of remaining unobserved space, and a prior distribution on the number of objects per type in the world and the density of objects in each container.

A. Modeling co-occurrence between object types

The basic prior distribution for this task is founded on the likelihood that objects of various types will co-occur in the same container. We adopt the model from our previous work [18]. Each object belongs to one of a finite universe of T object types, denoted by $\{t_i\}$. We assume the contents of the containers are independent and that the *composition* of a container can be modeled by θ, an element of the $(T-1)$-dimensional simplex (i.e., the space of T-element discrete distributions), representing a normalized vector of the counts of each type of object in the container. For example, a container with 2 objects of type t_1 and 3 of type t_3, when there are $T = 4$ possible types, would have $\theta = (.4, 0, .6, 0)$. We view the composition as a discrete distribution over the types of objects in the container.

Our prior on compositions is based on a T-dimensional vector η drawn from a multivariate normal distribution:

$$\eta \sim \mathcal{N}(\mu, \Sigma) \qquad \mu \in \mathbb{R}^T, \Sigma \in \mathbb{R}^{T\times T}, \Sigma \succ 0 \tag{1}$$

The covariance Σ encodes the object-object type correlation.[1] A logistic-normal transformation $\sigma(\cdot)$ is applied on η to get θ in order to preserve membership in the simplex:

$$\theta_i = \sigma(\eta) = \frac{\exp(\eta_i)}{\sum_{k=1}^{T}\exp(\eta_k)} \qquad 1 \le i \le T \tag{2}$$

$$\mathbb{P}(\tau = t_i \,;\, \theta) = \theta_i \qquad 1 \le i \le T \tag{3}$$

Given observations of objects with types $\{\tau\}$ in a container c, we would like to infer its *latent* composition θ (equivalently, η). In our previous work, we assumed that the types were observed perfectly. Unfortunately, even under this assumption, the posterior on η (and θ) is non-conjugate:

$$\mathbb{P}(\eta \mid \{\tau\}) \propto \mathbb{P}(\{\tau\} \mid \eta)\,\mathbb{P}(\eta) = \left[\prod_{\tau}\mathbb{P}(\tau \mid \eta)\right]\mathbb{P}(\eta) \tag{4}$$

$$= \left[\prod_{\tau}\frac{\exp(\eta_\tau)}{\sum_{k=1}^{T}\exp(\eta_k)}\right]\mathcal{N}(\eta\,;\,\mu,\Sigma) \tag{5}$$

[1] The hyperparameters μ and Σ can be estimated from empirical data of θ. See Hoff [5] and Wong *et al.* [18] for more details.

We draw samples from the posterior on η using a Markov-chain Monte Carlo (MCMC) sampler [5]. These samples are then transformed using the logistic function, giving posterior samples of the composition vector θ. When observing/removing a new object from the container, the probability that the object will be of type t_i is then approximated by the average value of θ_i in the samples. Returning to the task of searching for query type t_q, one strategy could be to visit the container with the highest inferred θ_q.

The above model was already explored in our previous work. To be slightly more realistic, one frequent source of noise was object type confusion. Although object poses tend to be accurate (especially with depth sensors), recognizing the object instance/type is often challenging. Hence we wish to model an extra layer of uncertainty, where the true object types may be also be unknown.

Recall that type error probabilities are given by:

$$M_{ij} = \mathbb{P}(o = t_j \mid \tau = t_i) \tag{6}$$

Then we wish to find the posterior:

$$\mathbb{P}(\eta \mid \{o\}) \propto \mathbb{P}(\{o\} \mid \eta)\,\mathbb{P}(\eta) = \left[\prod_{o}\mathbb{P}(o \mid \eta)\right]\mathbb{P}(\eta) \tag{7}$$

$$\propto \left[\prod_{o}\sum_{\tau}\mathbb{P}(o \mid \tau)\,\mathbb{P}(\tau \mid \eta)\right]\mathbb{P}(\eta) \tag{8}$$

$$= \left[\prod_{o}\sum_{\tau}\frac{M_{\tau o}\exp(\eta_\tau)}{\sum_{k=1}^{T}\exp(\eta_k)}\right]\mathcal{N}(\eta\,;\,\mu,\Sigma) \tag{9}$$

The extra summation over possible true types $\{\tau\}$ results in a few more computational steps in evaluating the likelihood of a configuration, a necessary step in MCMC sampling.

B. Incorporating constraints

In addition to information provided by the types of objects in the same container, there are strong constraints provided by geometry that govern the ability of sets of objects to be placed inside a container. Additionally, there are constraints from prior knowledge about the total number of objects in the world; for example, it is typically more likely to have ten forks than one fork in a kitchen.

To model geometric constraints on containers, we assume that a likelihood function $\gamma_n(\Lambda_n)$ on configurations Λ_n is specified for each container c_n. A configuration is the set of all object states $\{(x, y, \tau)\}$ within the container. We can use γ to specify prior information such as how sparsely populated we think some containers are, or knowledge about particular non-uniform arrangements (e.g., objects tend to be placed in the back). For our case, we specified an empirical distribution on the total counts of objects within each container, and encoded physical (non-collision) constraints by forcing any non-physical arrangement to have zero likelihood.

We also allow specification of global constraints, which generally have form $\phi(\Lambda_1, \ldots, \Lambda_N)$. For our application, we only consider likelihood functions of the form $\phi_i(k)$, which specifies the likelihood that there are a total of k objects of type t_i in the domain (i.e., total across all N containers).

Fig. 2. Example particle configurations for the prior (left) and posterior belief (right) after observing a set of objects (middle). In this bird's-eye projection, the 'front' of the container is at the bottom, where the robot's viewing window is situated. The samples on the right are consistent with the observation in the middle; the only variation between samples is the occluded region (top left of square), and the potentially-confused type of yellow objects.

Such global constraints cause the containers to no longer be independent of each other, since the presence of an object in one container has implications about the number of objects of the same type within other containers.

We have developed a Metropolis-Hastings (MH) algorithm to sample configurations incorporating the above constraints. Each sample configuration must be consistent with existing observations, so under our assumptions it must include a set of objects with types $\{\tau_i\}$ that could have potentially generated the observed types $\{o_j\}$, at the set of true observed locations (recall that we assumed these were detected perfectly). The configuration $(\Lambda_1, \ldots, \Lambda_N)$ in each particle is a *complete geometric realization* of objects in the containers.

Figure 2 shows an example of sampled configurations for a single container, in belief particles both before and after a belief update step. Before making any observations, the agent has some prior belief about the container's contents, which is shown in the five samples on the left. After observing the object types in the middle figure (from a view window that only sees part of the container, as seen if the robot was situated at the 'bottom' facing 'up'), a belief update is performed, resulting in new sample configurations, five of which are shown on the right. In this example, the red object type is not confused with any other types, hence all posterior samples reflect this perfect observation. On the other hand, blue objects are occasionally observed as yellow ones, as seen in the first configuration on the right.

The idea behind Markov-chain Monte Carlo (MCMC) methods is that it constructs a Markov chain (sequence) of samples, whose stationary distribution converges to the desired posterior distribution. Hence, the sequence of samples can be viewed as coming from the posterior, although samples adjacent in time will tend to be dependent. This is often the case because the sample space is complicated and exploration is based on *local* transitions.

The Metropolis-Hastings (MH) algorithm is a particular type of MCMC method that requires specifying two functions: a (unnormalized) posterior probability $\mathbb{P}(\Lambda \mid \{o\})$, and a proposal function/distribution $g(\Lambda \to \Lambda')$ (specifying the likelihood of transitioning from state Λ to candidate Λ'). To obtain a new sample from the current state Λ, the proposal function is queried for a candidate state Λ', and the following *acceptance ratio* is computed:

$$\alpha(\Lambda \to \Lambda') = \min\left[1, \frac{\mathbb{P}(\Lambda' \mid \{o\})\, g(\Lambda' \to \Lambda)}{\mathbb{P}(\Lambda \mid \{o\})\, g(\Lambda \to \Lambda')}\right] \tag{10}$$

A number is drawn from $\mathrm{Unif}(0, 1)$. If the number is below α, then the candidate is accepted and Λ' becomes the new state of the Markov chain. Otherwise, it is rejected, and the state remains at Λ (and is counted as another sample, hence illustrating that consecutive samples may be dependent).

In our example, the sample space is the space of configurations (of all N containers). At each sampling step, the MH algorithm may propose moves of the following types:

- **Add** an object of type t_i with probability θ_i to a randomly-chosen location in a randomly-chosen container c_n, as long as the move does not violate the spatial constraints of the container c_n, penetrate other existing objects in the container, occlude an object that has been observed, or placed in a location that could have been seen by a previous observation action.
- **Remove** an object of type t_i if the object has not been observed. This object is chosen uniformly from a randomly chosen container c_n, and from all existing objects in container c_n that have not been observed.
- **Relabel** the underlying types of the objects that have been (noisily) observed. Randomly generate a non-zero probability type-relabeling of the observed objects, as long as the new types do not violate spatial constraints or penetrate existing objects. This step is not technically necessary for MCMC convergence since it can be replicated by a sequence of add and remove steps. However, since there is often uncertainty in the object types, introducing an explicit move that can quickly move between different type labelings is helpful for faster mixing of the Markov chain.

Let Λ^k be the k-th sample in the MCMC chain. The state $\Lambda^k = (\Lambda_1^k, \ldots, \Lambda_N^k)$ specifies the configuration for each of the N containers. Recall that the configuration of container c_n, Λ_n^k, consists of a set of object states $\{(x, y, \tau)\}$.

The likelihood of a configuration is given by

$$\mathbb{P}(\Lambda^k \mid \{o\}) \propto \mathbb{P}(\{o\} \mid \Lambda_1^k, \ldots, \Lambda_N^k)\, \mathbb{P}(\Lambda_1^k, \ldots, \Lambda_N^k) \tag{11}$$

$$= \left[\prod_i \left(\prod_n L(t_i, c_n)\right) \phi_i\left(C(t_i)\right)\right] \left[\prod_n \gamma_n(\Lambda_n)\right] \tag{12}$$

As before, the index i iterates over object types, and n iterates over containers. In the above, $C(t_i)$ is the total number of objects of type t_i across all containers. $L(t_i, c_n)$ is the product of likelihood terms for of all objects of type t_i in container c_n. In particular, for each object in c_n with $\tau = t_i$, $L(t_i, c_n)$ contains the following product:

$$\theta_i \prod_o \mathbb{P}(o \mid \tau) = \theta_i \prod_o M_{\tau o} \tag{13}$$

Finally, recall that ϕ_i is the global constraint on numbers of objects of type t_i, and γ_n is likelihood of the geometric arrangement in container c_n (specified by Λ_n).

Finally, to complete the description of the MH algorithm, we have to specify the transition probabilities in the proposal distribution. Each move type is chosen with equal probability.

- **Add**: If we add an object of type t_i to container c_n at location l in the current sample configuration, then

$$\mathbb{P}(\Lambda \to \Lambda') = \mathbb{P}(c_n)\,\mathbb{P}(l \mid c_n)\,\theta_i^{(n)} \tag{14}$$

  We choose uniformly across the containers, so $\mathbb{P}(c_n) = \frac{1}{N}$, where N is the total number of containers. We obtain $\mathbb{P}(l \mid c_n)$ by finding the probability of placing an object of type t_i at the particular location $l = (x, y)$ given the available configuration space in container c_n by using rejection sampling to find locations that do not violate any constraints. We use the ratio of the number of samples we keep over the total number of samples as $\mathbb{P}(l \mid c_n)$, the probability of placing an object of type t_i at a particular location l.
- **Remove**: If we remove an object obj of type t_i in container c_n, then

$$\mathbb{P}(\Lambda \to \Lambda') = \mathbb{P}(c_n)\,\mathbb{P}(\text{select } obj) \tag{15}$$

  If there are m objects in the container configuration that have not been observed yet, $\mathbb{P}(\text{select } obj) = \frac{1}{N}$. We again have $\mathbb{P}(c_n) = \frac{1}{N}$.
- **Relabel**: Given a set of observed object types $\{o\}$, we consider all possible types in the configuration as long as there is non-zero probability. Since the space of relabelings is $O(T^{|\{o\}|})$, most of which have low probability, we do not choose a relabeling uniformly at random. Instead, we sample labeling candidates according to the likelihood of confusion given by M.

In order to determine how many samples to use, we verify convergence on the underlying distribution of configuration, we monitor convergence by running independent trials. We then compare the between- and within-sequence variances as outlined by Gelman *et al.* [4]. We then discard the first half of the samples as "burn-in", and use the rest of the samples to infer the probability an object resides in a container by counting the proportion of samples that contain it.

V. Action selection

The system interacts with the world by repeatedly: generating a set of samples from the current belief distribution, using these samples in a finite-horizon forward search to select an initial action, executing that action in the world, getting an observation, and using that action and observation to condition the next belief distribution.

We plan in belief space by growing a belief tree. Each belief node comprises a set of belief particles. Each belief particle contains the configuration of the contents of all the containers. We perform particle filtering on these belief particles each time we grow the tree to a deeper level based on the observation on each branch.

A. Clustered observation models

Because the set of possible observations is infinite, we must actually construct an abstraction of the complete belief tree. Rather than branching on every possible set of observed poses and types, the search tree branches on *classes* of observations. These classes, or clusters, partition the observation space a small number of possible "abstract" observations. The partitioning method is designed to aggregate observations that may have similar effects on the subsequent attainable value in the remainder of a plan. In particular, we would like to ignore any irrelevant aspects of observations, such as uninformative pose information, and project observations into a much smaller finite space. This is motivated purely for computational reasons in planning.

We explored two strategies for partitioning observations, the *type-based* observation model and the *target-based* observation model.

a) Type-based observation model: In the *type-based observation* model, we construct branches for observing each possible type of object as well as observing nothing. If there are T object types in the universe, the branching factor for the observation is $T + 1$. Given the container configuration in a particle, label the branch for type t_i with the probability of seeing at least one object of that type. For each particle,

$$\begin{aligned}\mathbb{P}_{t_i} &= \mathbb{P}_{c_n}(\text{observe at least one object of type } t_i) \\ &= 1 - \prod_{\text{observable } \tau} (1 - \mathbb{P}(o = t_i \mid \tau)) \qquad (16)\end{aligned}$$

$$\begin{aligned}\mathbb{P}_{\emptyset} &= \mathbb{P}_{c_n}(\text{observe nothing}) \qquad (17) \\ &= 1 - \sum_i \mathbb{P}_{c_n}(\text{observe at least one object of type } t_i)\end{aligned}$$

We then update the weights for all the particles, by multiplying the old weight with the above probability (depending on the observation branch). The weight update for a particle in the observation branch for observing t_i is

$$w_{t_i}^{\text{new}} = w_{t_i}^{\text{old}}\,\mathbb{P}_{t_i} \tag{18}$$

The observation probability for this particular branch is $\sum w_{t_i}^{\text{new}} / \sum_i \sum w_{t_i}^{\text{new}}$. We then renormalize the weight of the particles in each belief node, so the weights of all the particles in a belief node sums up to 1.

b) Target-based observation model: In the *target-based observation* model, we use an (unrealistically) idealized sensor model that tells us, for the container, either: an object of type t^* is present and visible ("see target"), an object of type t^* is present in the container but currently invisible ("hidden target"), or an object of the desired type is not present in the container ("no target"). Since we are ultimately interested in searching for an object of query type t_q, in our application we choose $t^* = t_q$.

Along the "see target" branch, the weight update is the same as for the corresponding case in the type-based observation model:

$$\begin{aligned}\mathbb{P}_{t^*}^{\text{front}} &= \mathbb{P}_{c_n}(\text{obs} \geq \text{one obj of type } \tau^* \text{ in front layer}) \\ &= 1 - \prod_{\text{observable } \tau} (1 - \mathbb{P}(o = t^* \mid \tau)) \qquad (19)\end{aligned}$$

Along the "hidden target" branch, we ignore all the objects in the front layer, and look at the likelihood of observing a

target object in the rest of the configuration, defined to be

$$\mathbb{P}_{\text{hidden}} = (1 - \mathbb{P}^{\text{front}}_{t^*})\, \mathbb{P}^{\text{back}}_{t^*} \tag{20}$$

$$\begin{aligned}\mathbb{P}^{\text{back}}_{t^*} &= \mathbb{P}_{c_n}(\text{observe} \geq 1 \text{ object of type } \tau^* \text{ in the back}) \\ &= 1 - \prod_{\text{unobservable } \tau} (1 - \mathbb{P}(o = t^* \mid \tau))\end{aligned} \tag{21}$$

For the "no target" branch,

$$\mathbb{P}_{\text{no target}} = 1 - \mathbb{P}_{t^*} - \mathbb{P}_{\text{hidden}} \tag{22}$$

By grouping observation branches into three categories, the target-based observation model significantly reduces the branching factor and makes it independent of the number of types in the domain, but it represents a significant perturbation of the actual observation model of the robot. We might expect it to generate action values that are over-optimistic, due to thinking it will get more information about a container than is actually available.

B. Actions

In order to control the size of the search space, we also construct an action space that is an abstraction of the true action space. We assume, for example, that rather than specifying the real-valued parameters of the robot's motion, they are parameterized by the observed position of the object it is going to pick up.

For all observation models, these two types of actions are available to the robot, both during planning and in execution.

- **Move**: Move to a different view window, which may or may not have been visited previously.
- **Observe**: Make an observation at the current view window.
- **RemoveAt**: Remove an object that has been previously observed in the real world (using the coordinates at which it was observed perfectly.)

Since we use abstract observation models that do not generate actual observations with type and pose information, we cannot use the **RemoveAt** action during planning. Instead, we introduce different abstract actions for each of the two clustered observation models. Each action can be applied an any (belief particle) configuration. The actions are designed to reflect the abstractions made in the observation model.

In the *target-based observation* model, these two additional actions are available:

- **RemoveOccluder**: If the most recent observation branch is labeled "hidden target" or "no target," remove the object in the front layer that occludes the most unviewed space in the container.
- **RemoveTarget**: If the most recent observation branch is "see target", remove the visible object that has the highest probability of being of the target type. That is, we consider $\mathbb{P}(o = t^* \mid \tau)$ for every object in the front layer in a particle's configuration.

In the *type-based* observation model, an additional family of actions is available:

- **RemoveType**(t): Remove the object that has the highest probability of being type t.

C. Search process

Starting at the root node, which contains a set of particles sampled from the current posterior belief distribution, the tree is grown by:

- Adding a branch for each possible action, and copying all of the particles into each node;
- Adding a branch for each possible observation (given the observation mode) and copying the parent set of particles, reweighting them according to the likelihood of that observation in each particle and renormalizing.
- If all of the particles contain a visible instance of the target type, that node is not expanded further.
- At a fixed depth K, the process is terminated. Instead of adding further layers of branching, a value is assigned to each node according to the static evaluation function.
- Values are propagated back up the tree, taking the expectation of the children values at observation nodes and the maximum over the children values at action nodes.
- The top-level action whose child node has the highest value is selected for execution.

These steps are described in more detail below.

a) Static evaluation function: The static evaluation function takes a belief node as input and returns an estimated cost of finding an object of the target type, under the assumption that there is at least one such object in the domain. We have explored two different choices for this function.

Let β be an individual belief particle in belief node b and let v be a view window. Every particle β has a specified view window, which is where the robot is positioned in that state; we denote it by $\beta.v$. The cost for the robot to move from v_1 to v_2 is MoveCost(v_1, v_2). The cost, in a particular world state β for the robot to remove the minimum set of objects occluding an object of type q from view window v is MinPickCost(β, q, v); if there is no object of type q then this value is infinite.

Define $a(h)$ to be the action sequence of length h. $Cost(\beta, a(h))$ is the cost of executing the action sequence $a(h)$ in the configuration layout of belief particle β.

We denote k^* as the number of steps we look ahead when we plan in the belief tree, so the value function of the leaf belief node would be $V(b_{k^*})$.

- Optimistic: $V_{optimistic}(b_{k^*}) = \mathbb{E}_\beta[\min_v[\text{MoveCost}(\beta.v, v) + \text{MinPickCost}(\beta, q, v)]]$
- Choosing the best action sequence with respect to the belief distribution: $V_0(b_{k^*}) = \min_{a(h)} \mathbb{E}_\beta\left[\text{Cost}(\beta, a(h))\right]$

b) Deciding on an action: We compute the value of a belief node recursively by taking the expected value across the child observation branches and choose the action a^* that minimizes the total cost in the tree below the action node itself. More concretely, at tree level k, we have

$$V(b_k) = \min_a \sum_o \mathbb{P}(o \mid b_k, a)\, V(b_{k+1}) + \text{cost}(a) \tag{23}$$

$$a^*_k = \arg\min_a \sum_o \mathbb{P}(o \mid b_k, a)\, V(b_{k+1}) + \text{cost}(a)$$

VI. Experiments

We are primarily interested in whether our more-detailed state estimator (Sec. IV) and non-myopic planner (Sec. V) help reduce the overall execution cost for finding a target object. We are also interested in seeing if there are any significant differences between the two clustered observation models used in planning (**target-based** and **type-based**). For planning, we evaluated both **1-step** and **2-step** lookahead search to examine the benefits of a longer planning horizon.

For comparison, we implemented two baseline methods. The first is a **systematic** action selection strategy that chooses, uniformly at random, a container to move to, removes all visible objects, observes at the same location again, and repeats this process until the target is found, or all objects in the container are removed. If a container is emptied, a new container is chosen and the process repeats. The second baseline is the **greedy** algorithm that we expanded based on the work by Wong *et al.* [18], which uses the object type co-occurrence prior that we build on, and greedily chooses the container with the highest probability of containing the target type. Once a container is chosen, the robot moves to the container, and takes an observation. Given an observed type o, the robot removes the object that has the highest $\mathbb{P}(\text{object is } t_q \,|\, \text{object observed to be } t_i)$. Now the robot has noiseless observation on the object that has just been picked up, as well as noisy observations on the rest of the scene prior to picking up the object. The robot then performs belief updates on θ using the MCMC update outlined by Wong *et al.* [18]. The robot then moves to the container that has the highest likelihood of containing the target type (possibly the same container as before), and the process repeats. If a container is emptied, the container with the next highest likelihood of containing the target type is chosen.

All experiments were performed in simulation. We generated simulated domains by specifying the positions of containers, then randomly sampled the true initial configuration from our prior. The prior hyperparameters were trained using a set of examples containing counts of object types in containers. Visualizations of four illustrative domains are shown in the top row of Fig. 3. All four domains are generated from the same prior. In these domains, we placed three containers, two of which are close together, and one far away. Four object types are present (color-coded). Yellow and red objects tend to be co-located; green and blue objects tend to be co-located. The cost of moving is given by the distance shown on the horizontal axis. The cost of removing an object is 15, and for observing is 8. The robot is initially at position 55. The total execution costs shown in Fig. 3 were averaged over 25 trials for each domain.

In the three leftmost domains, the goal is to find a blue object. Blue and yellow objects are confused with each other with probability 0.2; red and green objects both have noiseless observations. so even if a blue object is visible, multiple observations may be necessary. In the rightmost domain, the goal is to find a green object.

We observe three qualitative trends: Our proposed method occasionally achieves lower execution cost (and always finds the lowest among the methods), but the variance tends to be high; the **type-based** method performs similarly to the **target-based** method except in the first domain; and additional lookahead did not improve execution cost significantly.

In the first (left-most) domain, since our method tends to consider the likelihood of finding the target type as well as the action costs together, the robot moves to the middle container first in most trials despite the fact that the leftmost container has the highest likelihood of generating the target blue type. Since the observation between yellow and blue objects is noisy, observing two yellow objects in the middle container makes the planner thinks there is a high probability that at least one of the two yellow objects is in fact blue. On the other hand, the greedy algorithm lets the robot go to the leftmost container and removes an object. The strong prior on the leftmost container containing the target type makes the robot keep looking in the same container. Clearly the blue object can be revealed almost immediately after the robot removes the yellow object.

In the second domain, the blue objects are more occluded, so **systematic** becomes a poor strategy (especially if it chooses the container on the right). Based on the object type co-occurrence prior, only the left-most container is likely to contain blue objects, and since there is a blue object that is not occluded at all in the left-most container, all other strategies quickly find it and achieve similar performance.

In the third domain, our method could find solutions that are of the lowest cost. Instead of going straight to the leftmost container like **greedy**, our method considers moving cost, and chooses to visit the middle container first in most trials, and the innermost blue object in the middle container can be revealed instantly (the red objects only occluded a small portion of the blue object, and using our formulation, we consider the innermost blue object as instantly revealable.) We also observe that the 2-lookahead performs slightly better in terms of the variance of the cost across trials, because the 2-lookahead discourages the planner to choose to go to the rightmost container as its first action. The 2-lookahead considers a longer-term gain, so visiting the rightmost container is likely to incur a higher moving cost to move towards the other two containers in the case of not finding the target object in the rightmost container.

Finally, in the fourth domain, we see that our approach has the greatest advantage against **greedy**. In this domain, the goal is to look for the green object. **Greedy** tends to go to the leftmost container because of the strong prior that the leftmost container tends to have the green object. It then tends to remove everything in the leftmost container. Unlike our method, **greedy** is incapable of reasoning that given a few removed objects from a container, it greatly constrains the available space for the target object to exist. Instead, it only looks at θ, the generative probability of object types. Among the trials, we observe that in the cases that our method chooses to go to the leftmost container, it does not get stuck keeping removing objects from it. Instead, it could move away to other containers to find the green object.

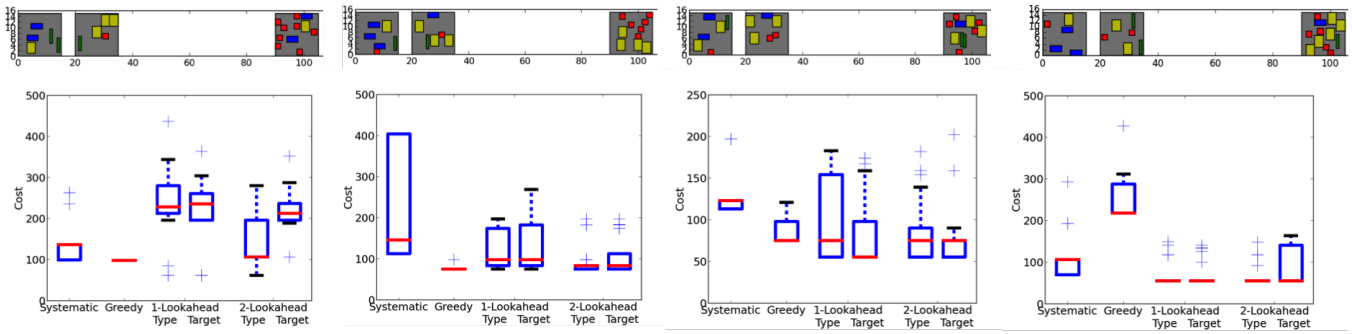

Fig. 3. Four illustrative simulated domains, and execution costs for different strategies on the domains. The top row shows the configurations of three containers; object types are color-coded, and the distances between containers are to-scale. The bottom row shows box plots of total execution costs (including motion, removing, and observation actions) over 25 trials for 6 different planning strategies. Each box and whisker visualizes the distribution of executions costs: the red line is the median, the box shows the first and third quartiles (i.e., is the inter-quartile range), the whiskers extend to the range of the costs, and the "+" markers are significantly beyond this range. See text for a comparison between the different strategies on each domain.

VII. Discussion

The lack of systematic performance differences between the two clustered observation models was somewhat surprising and warrants further investigation. In one case, a planning horizon of two was significantly more helpful than a look-ahead of one; it is possible that even deeper look-ahead would have value, but it would be computationally quite costly. For the observation models, although in our case they used a similar amount of time for planning and achieved similar execution costs, we note that their branching factors scale differently. In particular, while the **target-based** approach has a constant branching factor of 3, the **type-based** approach factor scales with the number of types T. In our case, T was 4, so performance was similar; in a more realistic scenario, we expect T to be much larger, and we expect the **target-based** abstract observation model to perform much better in terms of planning time.

We were also surprised to observe that **greedy** generally achieves good near-deterministic performance on our simulated domains, despite making drastic assumptions. Although lookahead planning is often on par and occasionally improves on the execution cost, it comes with a much greater computational cost and variance in execution performance. We believe that further complexity in the world will cause the lookahead strategy to gain further advantage. Also, more sophisticated online POMDP planning techniques [13] such as POMCP [15] and DESPOT [16] may be able to make planning more efficient while providing deeper-lookahead exploration. On the other hand, it may also turn out that a greedy strategy equipped with a good inference mechanism is sufficient for the object search problem.

In future work, we will seek ways to improve efficiency of the inference, to allow scaling up to realistic kitchen-sized domains. We will also study whether it is important to model error in the geometric aspects of sensing when planning at this level of abstraction, and consider strategies for handling objects that are stacked on top of one another, as well. These techniques may ultimately be the basis for mobile manipulation robots that use and understanding of space, visibility, and spatial organization to be of real assistance to humans in complex domains.

References

[1] A. Aydemir, A. Pronobis, M. Göbelbecker, and P. Jensfelt. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics*, 29(4):986–1002, 2013.

[2] N. Correll, K.E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J.M. Romano, and P.R. Wurman. Lessons from the Amazon Picking Challenge. *ArXiv e-prints*, 2016.

[3] M.R. Dogar, M.C. Koval, A. Tallavajhula, and S.S. Srinivasa. Object search by manipulation. *Autonomous Robots*, 36(1–2):153–167, 2014.

[4] A. Gelman, J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari, and D.B. Rubin. *Bayesian Data Analysis*. CRC Press, 3 edition, 2013.

[5] P. D. Hoff. Nonparametric modeling of hierarchically exchangeable data. Technical Report 421, Dept. of Statistics, U. of Washington, 2003.

[6] K. Hsiao, L.P. Kaelbling, and T. Lozano-Pérez. Robust grasping under object pose uncertainty. *Autonomous Robots*, 31(2):253–268, 2011.

[7] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[8] T. Kollar and N. Roy. Utilizing object-object and object-scene context when planning to find things. In *ICRA*, 2009.

[9] L. Kunze, K.K. Doreswamy, and N. Hawes. Using qualitative spatial relations for indirect object search. In *ICRA*, 2014.

[10] Yu-Chi Lin, Shao-Ting Wei, Shih-An Yang, and Li-Chen Fu. Planning on searching occluded target object with a mobile robot manipulator. In *ICRA*, 2015.

[11] M. Lorbach, S. Hofer, and O. Brock. Prior-assisted propagation of spatial information for object search. In *IROS*, 2014.

[12] B. Moldovan and L. De Raedt. Occluded object search by relational affordances. In *ICRA*, 2014.

[13] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[14] Z. Saigol, B. Ridder, M. Wang, R. Dearden, M. Fox, N. Hawes, D.M. Lane, and D. Long. Efficient search for known objects in unknown environments using autonomous indoor robots. In *IROS Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*, 2015.

[15] D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *Neural Information Processing Systems*, 2010.

[16] A. Somani, N. Ye, D. Hsu, and W.S. Lee. DESPOT: Online POMDP planning with regularization. In *Neural Information Processing Systems*, 2013.

[17] L.E. Wixson and D.H. Ballard. Using intermediate objects to improve the efficiency of visual search. *International Journal of Computer Vision*, 12(2–3):209–230, 1994.

[18] L.L.S. Wong, L.P. Kaelbling, and T. Lozano-Pérez. Manipulation-based active search for occluded objects. In *ICRA*, 2013.

[19] Y. Ye and J.K. Tsotsos. Sensor planning in 3D object search. *Computer Vision and Image Understanding*, 73(2):145–168, 1996.