Online Planning for Target Object Search in Clutter under Partial Observability

Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen and Christopher Amato

***Abstract*— The problem of finding and grasping a target object in a cluttered, uncertain environment, *target object search*, is a common and important problem in robotics. One key challenge is the uncertainty of locating and recognizing each object in a cluttered environment due to noisy perception and occlusions. Furthermore, the uncertainty in localization makes manipulation difficult and uncertain. To cope with these challenges, we formulate the target object search task as a partially observable Markov decision process (POMDP), enabling the robot to reason about perceptual and manipulation uncertainty while searching. To further address the manipulation difficulty, we propose Parameterized Action Partially Observable Monte-Carlo Planning (PA-POMCP), an algorithm that evaluates manipulation actions by taking into account the effect of the robot's current belief on the success of the action execution. In addition, a novel run-time initial belief generator and a state value estimator are introduced in this paper to facilitate the PA-POMCP algorithm. Our experiments show that our methods solve the target object search task in settings where simpler methods either take more object movements or fail.**

I. Introduction

Searching for a specific object in a cluttered scene is a common task in our daily lives. To find the target object, we need to reason about which other objects could occlude the target and rearrange objects to look for the desired one. While humans are inherently capable of solving this task, it is quite challenging for a robot to solve due to incomplete knowledge of the environment. The robot has to locate and recognize objects based on noisy sensor data, and the outcome of manipulation actions can be uncertain.

Consider a service robot in an office environment that has been tasked with retrieving a stapler from a tabletop (see Fig. 1(left)). From the robot's camera view, the stapler is fully occluded (see Fig. 1(right)). The robot has to first reason about which objects the stapler is likely to be hidden behind, and then either move around to gather more information from a different viewpoint or sequentially remove these objects in an efficient manner to be able to see and pick up the stapler. Occlusions caused by objects surrounding the stapler prevent the robot's perception system from accurately identifying the object's class and position. Not only does this make it harder to decide how to move the objects, but it also increases the likelihood of manipulation failures.

To overcome these challenges, we make three main contributions. First, we model the problem as a *partially observable Markov decision process* (POMDP) [1]. In contrast to previous work, our model can be used to solve more general problem instances that are not constrained by either the number of hidden objects or the configuration of the objects. Second, to find solutions to our model, we introduce PA-POMCP, an extension of POMCP [2], a state-of-the-art online POMDP solver, by parameterizing the actions with respect to the robot's current belief. Third, we propose a run-time initial belief generator that allows the robot to reason about the locations of hidden objects.

Fig. 1. A service robot is searching for a stapler (**Left**) which is fully occluded under its current view (**Right**).

To evaluate our method, we perform experiments both in simulation and on a robot. In simulation, our method can generate solutions in significantly fewer steps than baseline methods, and solve problem settings where these simpler methods fail to find a solution. We also demonstrate the practicality of our approach on a Fetch robot in a number of challenging real-world scenarios.

II. Related Work

Many early approaches address the target object search problem by reducing it to an active visual search problem [3]–[6]. Wong et al. [7] extend the previous work to solve the problem by allowing the robot to manipulate the objects to facilitate its detection. However, their method does not try to reduce the number of manipulation actions required to fetch the target object.

Recent work by Dogar et al. [8] generates plans based on a graph that models the visibility and accessibility relations between objects. Yet, they assume the target object to be the only hidden object. Srivastava et al. [9] propose a sample-based backtracking search algorithm that generates an offline policy for picking up a particular red cylinder blocked by many other blue cylinders. Chitnis et al. [10] improve upon the previous work by learning a policy for plan refinement and by using expert demonstrations that guide exploitation. However, both approaches simplify the object search problem without taking partial observability in the real world into account.

Li et al. [11] propose a POMDP model for an object search problem in a space confined environment (e.g., a shelf) and solve their model with an approximate online planner,

Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA {xiao.yuch, katt.s, tenpas.a, chen.shengji}@husky.neu.edu, c.amato@northeastern.edu

DESPOT [12]. Nevertheless, they do not account for fully occluded objects and require a segmentation method that reports the exact number of objects in the environment. This requirement significantly reduces perception uncertainty. Besides, their planning framework does not allow robot base movement and cannot handle potential manipulation failures.

Compared to related work, our approach is scalable and robust to any number of hidden objects and any degree of occlusions between objects that occur in the task. To tackle potential manipulation failures, we propose an extended version of POMCP, called Parameterized Action POMCP (PA-POMCP), that takes these failures into account.

III. Problem Formulation

We start with an overview of POMDPs, then describe our target object search problem and POMDP model.

A. POMDP Preliminaries

Partially observable Markov decision processes (POMDPs) [1] provide a framework for sequential decision making problems with uncertainty in observations as well as action outcomes. Formally, a POMDP can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{O}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a set of world states, $\mathcal{A}$ is a set of actions, and Ω is a set of observations. The state transition function, $\mathcal{T}(s, a, s') = p(s' \mid s, a)$, gives a probability distribution over next state $s' \in S$ after taking action $a \in A$ under current state $s \in S$. The observation model is a probability distribution over observations for each resulting state and action, $\mathcal{O}(s', a, o) = p(o \mid s', a)$. The state transition function captures the effects of the robot's actions and the observation model captures the effects of sensor noise. The immediate reward function $\mathcal{R}(s, a)$ gives a real-valued reward for taking action a in state s.

Because the world is partially observable, the robot does not have full knowledge of the current state s. Instead, it keeps track of a *belief state* b: a probability distribution over the state $s \in \mathcal{S}$. The robot starts with an initial belief b_0. Bayes' rule $b_t \propto p(o_t \mid s_t, a_{t-1}, b_{t-1}) p(s_t \mid a_{t-1}, b_{t-1})$ is used to calculate a new belief b_t given the received observation on time step t. A POMDP policy π is a mapping from belief states to actions. The objective of POMDP solution methods is then to find a policy that optimizes the expected sum of discounted rewards given an initial belief:

$$V^{\pi}(b_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)] \,\middle|\, a_t = \pi(b_t) \right], \qquad (1)$$

where the discount factor $\gamma \in [0, 1)$ determines the impact of future rewards on current decision making.

B. Target Object Search

In this paper, we address target object search in a cluttered environment. In general, the target object might be directly visible, partially occluded by surrounding objects, or even fully hidden somewhere. The objective in this task is to locate and fetch the target object, which often requires the robot to change perspective and manipulate other objects in an efficient way.

We assume that the robot knows the exact number of objects in the environment and has a geometric model for each of them. Yet, to identify these objects, the robot has to access an object detector that returns the 3D pose and the type (name string) of each detected object. The data captured by the robot's camera is usually very noisy due to the various degrees of occlusions of different objects. This makes the task harder because the robot needs to take the occlusions into account when it decides whether to go to other viewpoints or not, how to move the objects in succession and how to reach the target object in the end.

C. POMDP Model Formulation

State Space. For N objects, the state $s = [s_{rob}, s_{obj_i} | i = 1, \ldots, N]$ consists of the robot's state and each object's state. The robot's state, $s_{rob} = \langle x, y, z, \phi \rangle$, involves the robot's 3D position (x, y, z) in the world coordinate system and orientation as an angle ϕ about the z axis (vertical axis with respect to the ground). Each object's state $s_{obj_i} = \langle pos_i, t_i \rangle$, where $pos_i = (x, y, z, \theta)$ is the object's 3D position and orientation under the world coordinate system; $t_i \in \{0, 1\}$ indicates whether obj_i is the target or not. The state space is thus $(5N+4)$-dimensional and continuous. In this paper, we assume that the robot can access the true orientation of each object. Nevertheless, our model can be easily extended to cases where the orientation is partially observable by adding orientation to the observation space (see below).

Action Space. There are four action classes: MOVE_BASE(x, y, z, ϕ) navigates the robot to a specific location and orientation to change the point of view; MOVE_OBJ(i) picks up obj_i and places it into a predefined placing area; FETCH_OBJ(i) grasps obj_i and declares it is the target object, resulting in a terminal state; NO_TARGET affirms the target object does not exist on the table and terminates the task. MOVE_OBJ(i) and FETCH_OBJ(i) are detailed in Section IV-A.

Observation Space. Observation $o = [o_{rob}, o_{obj_i} | i = 1, \ldots, N]$ includes an observation of the robot's state (assumed to be fully observable), and an observation of each object (assumed to be partially observable), where $o_{obj_i} = \langle p_i, t_i, occl_i \rangle$ is comprised of three attributes: estimated position p_i, type t_i and occlusion ratio $occl_i$.

The observation space is discretized by partitioning the continuous position space into a grid based on sensor fidelity. Each object's estimated position p_i is calculated by mapping the object's estimated 3D position, returned by running segmentation on the point cloud data, to the grid. A string 'Moved' is assigned to p_i when obj_i has been moved away. The estimated position p_i of the hidden object has a value NONE if the object has not been detected in the current view. Type t_i has three values: 0 or 1 indicates whether obj_i is the target object or not, which can be determined by checking if the resulting label of obj_i, given by the object recognizer, matches with the target's; NONE type means the class of obj_i is not recognized due to the occlusion or the sensor's noise.

Perception uncertainties in cluttered environments are mostly caused by occlusions between objects. Accounting for

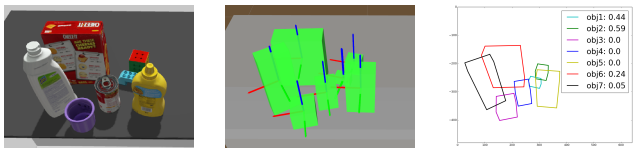


Fig. 2. Occlusion ratio calculation on the sensor's image plane. (**Left**) One example with 7 objects on a table under robot's view. (**Middle**) Segmentation result. (**Right**) Occlusion ratio of each object on the camera's image plane.

the occlusion ratio in POMDP observation models for object manipulation tasks has been proposed previously [11], [13]. However, unlike previous work, we calculate the occlusion ratio in a different way. We first project the bounding boxes returned by the segmentation to the camera's image plane, and then calculate the intersection ratio between the resulting irregular polygons for each object (one example shown in Fig. 2). The occlusion ratio space in our model is then further discretized into three levels: $occl_i =$ 'not occluded' when the occlusion ratio of obj_i is 0.0; $occl_i =$ 'partially occluded' when the occlusion ratio of obj_i is less than 0.3, otherwise $occl_i =$ 'fully occluded'.

Rewards. The rewards are inspired by the time required to execute each action. The reward of running action MOVE_BASE is -200. Successfully moving obj_i by executing MOVE_OBJ(i) gives a reward of -100. Otherwise, a penalty of -1000 is allocated if obj_i cannot be grasped. The reward of executing FETCH_OBJ(i) is 100 if obj_i is the target and currently reachable. Otherwise, the reward is -1000. Action NO_TARGET gives a reward of 100 if the target is not on the table. Otherwise, the reward is -1000.

IV. Approach

The POMDP model of target object search presented in Section III-C has a continuous state space and an observation space that grows exponentially in the number of objects. Fortunately, *Partially Observable Monte-Carlo Planning* (POMCP) [2], an online search method, has proven to be a scalable solver for large POMDPs [14]–[16].

At each time step, POMCP approximates the utility of the available actions at the current belief. It does so by incrementally building a look-ahead tree consisting of action-observation histories in a sample-based manner (as seen in Fig. 3). The samples represent interactions with a black-box simulator from the current belief until task completion. After every sample, a new leaf node is added to the tree—biasing tree expansion toward visited beliefs. The nodes contain statistics of how often they have been visited and their expected returns to allow exploration and exploitation to be considered within the tree using the Upper Confidence Bounds (UCB) method [17]. As more samples are generated, the utility estimates approach the true values, but the optimal action will often have the highest value with many fewer samples [2]. The action with the highest utility is chosen and the process repeats until the task has been completed.

We extend POMCP to Parameterized Action POMCP (PA-POMCP) that aims to deal with action failures due to partial observability, which is detailed below and summarized in Algorithms 1 and 2. PA-POMCP requires a prior distribution over the state space as the initial belief. To generate the distribution, we introduce a run-time initial belief generator that creates a set of particles as the initial belief (Section IV-B). Finally, we present a state value estimator for PA-POMCP that uses heuristics to guide the look-ahead tree search (Section IV-C).

Algorithm 1 PA-POMCP ($\mathcal{B}$, A, *num_sims*)

$A : \mathcal{B} \leftarrow$ Parameterize action space A given the current belief $\mathcal{B}$
$h_0 \leftarrow ()$
for $i \leftarrow 1 \cdots num\_sims$ **do**
  $s \sim \mathcal{B}$
  SIMULATE$(s, 0, h_0)$
$a : b \leftarrow \text{argmax}_{b \in \mathcal{B}} Q(h_0)$
return $a : b$

Algorithm 2 SIMULATE(s, d, h)

if ISTERMINAL(h) **or** $d =$*max_depth* **then**
  return 0
if $h \notin$ *Tree* **then**
  CONSTRUCTNODE(h)
  return SVE(s) // Call state value estimator
if $d = 0$ **then**
  $a : b \leftarrow$UCBSELECTION(h) // Parameterized with entire belief
  $param\_a \leftarrow a : b$
else
  $a : s \leftarrow$UCBSELECTION(h) // Parameterized with sampled state
  $param\_a \leftarrow a : s$
$(s', o, r) \sim \mathcal{G}(s, param\_a)$
$h' \leftarrow (h, param\_a, o)$
$R \leftarrow r + \gamma \cdot$SIMULATE$(s', d+1, h')$
$N(h, param\_a) \leftarrow N(h, param\_a) + 1$
$Q(h, param\_a) \leftarrow Q(h, param\_a) + \frac{R - Q(h, param\_a)}{N(h, param\_a)}$
return R

A. Parameterized Action POMCP (PA-POMCP)

POMCP typically assumes that the available actions are known a priori. However, in robotic manipulation under uncertainty, actions often require the exact position and orientation of the object. Inspired by Hsiao et al. [18], we propose a generalization of POMCP that parameterizes the action space with respect to the agent's belief, called Parameterized Action POMCP (PA-POMCP).

At the beginning of each planning step, PA-POMCP parameterizes action space A given the current belief $\mathcal{B}$, represented by $A : \mathcal{B}$. For example, in our experiments, actions MOVE_OBJ(i) and FETCH_OBJ(i) are parameterized by taking the mean position of the obj_i in the current belief. A simulation then starts by selecting a parameterized action

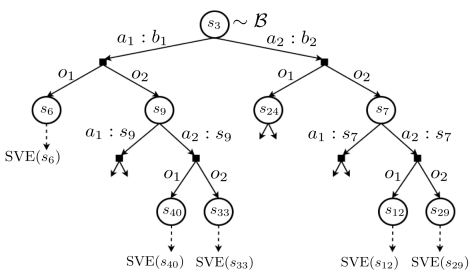


Fig. 3. A search tree constructed by PA-POMCP.

$a_i : b_i \in A : \mathcal{B}$ given a certain state sampled from the belief (see the actions under the root node in Fig. 3 denoted $a : b$), after which the black-box simulator returns an observation and a new state. At this point, the simulation continues as usual. For reasons of efficiency and accuracy, the belief is not maintained at later nodes in the POMCP tree. Therefore, in later simulation steps, we assume full knowledge of the state and the pose of obj_i and those actions in the tree are only parameterized with the particular state returned by the simulator (denoted by $a : s$ in Fig. 3). As a result, the values in PA-POMCP will only be approximations of the true values that consider manipulation uncertainty at later steps. Nevertheless, as an online planning method, it is most important to choose the correct action, which will often be maintained. Furthermore, the location uncertainty will be considered in a later planning step. After a fixed number of simulations, PA-POMCP selects the parameterized action with the highest estimated value. This process continues until all steps of the problem are completed.

B. Run-Time Initial Belief Generator

We represent a belief as a set of particles in a particle filter. Each particle is a sampled state of the objects in the environment, $s = [s_{obj_i} | i = 1, \ldots, N]$, that contains the position, orientation (assumed to be exactly known), and type of each object. The robot's state is not included in this particle filter because it is fully observable. Instead of manually specifying a prior over the state space, we designed a Run-Time Initial Belief Generator. This generator enables the robot to reason about the potential locations of hidden objects based on its perception and then automatically generate a belief.

The pipeline of generating one valid particle consists of four steps. First, we segment the point cloud to obtain each object's estimated position and draw a sample (x, y, z) from a Gaussian distribution centered at that position. Second, we locate the detected objects in the OpenRave simulator at the sampled positions. Third, for any undetected (hidden) objects, we randomly pick one of the detected objects and sample a position behind it for the hidden object. Fourth, we place the hidden object at the position sampled in Step 3 in the OpenRave simulator and calculate its occlusion ratio. If the occlusion ratio is larger than an upper bound (0.95 in our case), this particle is considered to be valid and will be added to the particle set. Otherwise, the particle is discarded.

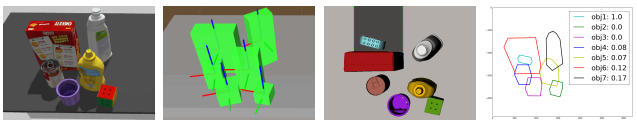


Fig. 4. An example of generating a valid particle. **Left to Right**: (a) A scenario with seven objects. (b) The segmentation result shows that six objects have been detected. (c) The detected objects are placed at the locations sampled from the segmentation result in the OpenRave simulator. The hidden object is assumed to be occluded by the red box, and is placed at a location sampled from the shaded region. (d) From the robot's view, the occlusion ratio of the hidden object is 1.0: the robot is not able to detect it. This particle thus is consistent with the situation in (a).

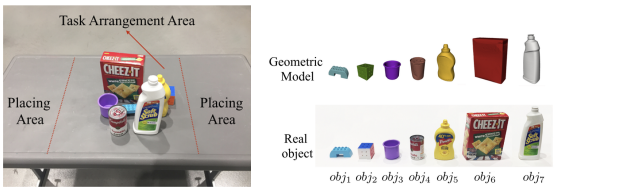


Fig. 5. Target object search domain. (**Left**) Domain settings. (**Right**) Seven objects from YCB with their geometric models.

For each object in each particle, we uniformly set the type to be 0 or 1. This indicates that the robot is highly uncertain about which object is the target object in the beginning. An example of generating a particle using our pipeline is shown in Fig. 4. To form an initial belief, we repeatedly run the above pipeline until a fixed number of valid particles has been generated.

C. PA-POMCP with State Value Estimator

POMCP traditionally evaluates a state s at a leaf node by utilizing a rollout policy $\pi_{rollout}$. However, running a rollout policy is a time-consuming operation, and the variance of the rollout estimator is also high due to the stochasticity of the environment, which we want to avoid. Also, defining a good rollout policy is often hard. As seen in Fig. 3, PA-POMCP estimates the value of a state s for a leaf node in a more efficient way by using a State Value Estimator (SVE).

Given a state s, our SVE(s) first calculates the number (N) of objects that exist in the environment and have not been moved. We assume that the moved object will not impact successive decision making or manipulation actions. If the target object is occluded in state s, SVE(s) returns the value $\sum_{n=0}^{N-2} \gamma^n(-100) + \gamma^{N-1}(100)$ which provides an upper bound value of a policy that moves every nontarget object away and fetches the target object in the end. If the target object is not occluded in state s, SVE(s) provides a value of 100 and assumes that the target can be directly grasped. Further, if the robot already moved the target object away without realizing it, the state s will receive a large penalty of value $\sum_{n=0}^{N-1} \gamma^n(-100) - \gamma^N(1000)$. This means that the robot has removed all the remaining objects and mistakenly declared that no target object exists in the environment in the end. In addition, under all above cases, as long as there exists one object out of the robot's workspace, a penalty of -200 is added for moving to other locations.

V. Experiments in Simulation

We performed experiments in simulation to demonstrate the effectiveness of our approach on a range of domains.

a) Domain Settings: The particular target object search domain we now address includes a table and seven different objects from the YCB Benchmarks—Object and Model Set [19] (see Fig. 5). The tabletop is divided into three areas: one task arrangement area and two object placing areas. In each placing area, there are eight predefined selectable locations to release the object. There are two predefined positions, one at the front and one at the back of the table, and the robot is allowed to move between them. Grasping poses for

each object are encoded in a Grasping Database that can be accessed during planning [20]. The observation space is discretized by partitioning the task arrangement area into a 6×6 grid. The type of each object is unknown to the robot, but a type observation is generated by a fine-tuned Fast R-CNN object recognition module [21]. To make the problem more challenging, we select the smallest object (Lego) as the target, because it is easily blocked by other objects. Note that the Lego can only be grasped from the top.

b) Experimental System: The architecture of the experimental system is shown in Figure 6. A Gazebo [22] environment with a fetch robot and seven YCB objects is used to simulate the real world. Based on the first observation o_{init}, the Initial Belief Generator provides a set of particles (400) as the initial belief $\mathcal{B}_{init}$, which is then passed to PA-POMCP to launch the planning. PA-POMCP evaluates each parameterized action $a_i : b_i$ by iterative interaction with an OpenRave simulator [23]. After a fixed number of simulations (600), PA-POMCP selects the parameterized action with the highest estimated value. The robot executes the parameterized action, receives a new observation o_{real}, and updates the belief for the next planning step.

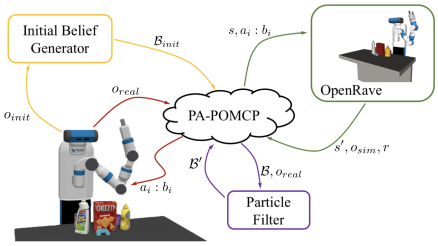


Fig. 6. Experimental system architecture.

A. Experimental Scenarios

Six selected scenarios with the initial beliefs generated by our initial belief generator are shown in Fig. 8 from (a) to (f). We briefly specify the purpose of each scenario below.

Scenario 1. Check if our planning framework is able to tell the robot to only perform manipulations on the objects (mustard bottle, cracker box and bleach bottle) which the target is likely to be hidden behind.

Scenario 2. Test if our planning framework is able to find the optimal solution to fetch the target without moving its base when all the objects in the scene can be detected by the robot and within its workspace in the beginning.

Scenario 3. Demonstrate that our planning framework is scalable to any number of hidden objects by starting with two hidden objects: Lego and tomato can.

Scenario 4. Show that our framework can deal with settings where objects close to the target need to be moved in a specific order. Here, if the target, Lego, is located between the mustard bottle and the cube (see Fig. 7). These objects block each other successively: the tomato can needs to be moved first, followed by the cube, and finally the Lego.

Scenario 5&6. Demonstrate that our framework is able to quickly learn that moving to the other side of the table is the optimal choice under two conditions: 1) all the potential locations of the target object are out of the robot's current workspace, 2) on the other side of the table, the target object can be fetched with less effort than at the current position.

Fig. 7. (**Left**) Scenario 4 settings (top view). (**Right**) The challenge of scenario 4 where most of the greedy policies fail by directly grasping the cube without removing tomato can first.

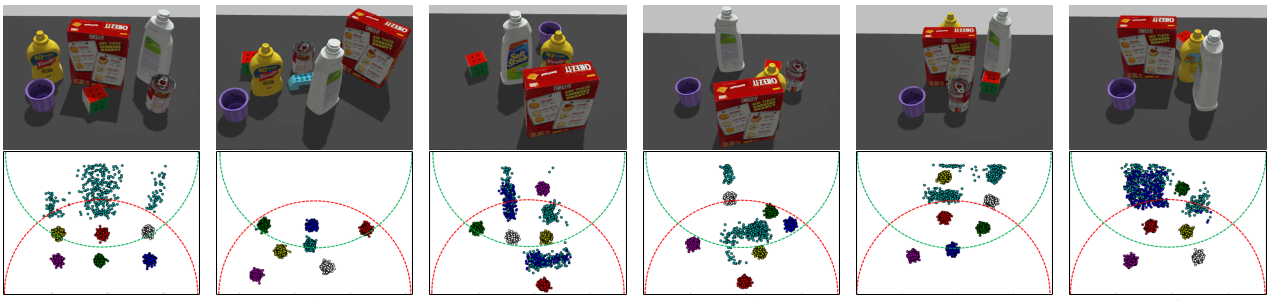
B. Results and Discussion

We compare our framework, PA-POMCP, with five greedy policies by performing 100 runs on the above scenarios. At the beginning of each run, the hidden objects are randomly placed at the locations where they are not visible to the robot. The average values over the 100 runs are shown in Table I.

The greedy strategies used in the experiments include: Greedy-M, which randomly moves all the non-target objects away with priority on the non-occluded ones and fetches the target object in the end; Greedy-T, which follows a similar approach as the previous one, but fetches the target object as soon as it is not occluded; Greedy-S, which first checks if any object is close to the target (within 10cm, which is the width of robot's gripper), and moves it away instead of immediately fetching the target one when it is not occluded; Greedy-O, which prefers to move the non-occluded object which reduces the current total occlusion ratio the most, and fetches the target instantly when it is exposed; and Greedy-OS, which combines Greedy-S and Greedy-O. When the robot intends to move an object which is out of the workspace, the greedy policies will navigate the robot to the opposite side first. All the greedy policies are purely based on the robot sensor's feedback.

The results shown in Table I demonstrate that PA-POMCP significantly outperforms the greedy policies in all six scenarios. Especially in scenario 4&6, when a similar challenging layout happens (see Fig. 7), PA-POMCP effectively solves this puzzle by realizing the successive blocking relationship between the objects and removing them in the appropriate order. In contrast, Greedy-OS, Greedy-O and Greedy-S became stuck in an infinite loop trying to remove the cube due to its proximity to the target, but without figuring out that the tomato can needs to be moved first. Notice that, while Greedy-M solved this difficult problem, it ended with a very low average value because the noisy sensor often incorrectly detected the occlusion relationship between objects, which caused the robot to randomly select to move the cube when the sensor's feedback reported both the cube and tomato can were occluded. These results strongly indicate that our approach can deal with uncertainty in both perception and manipulation.

To test the robustness of our planning framework, 100 random scenarios were created by uniformly sampling the position for each object in the task arrangement area and



(a) (b) (c) (d) (e) (f)

Fig. 8. Initial configuration of six simulation scenarios with robot view (**Top**) and initial belief particles (**Bottom**). **Bottom**, each point represents the possible position (bottom center) of the corresponding object, distinguished by colors: ● Lego, ● cube, ● cup, ● tomato can, ● mustard bottle, ● cracker box, ○ bleach bottle, - - the boundary of top grasping workspace at current position, - - the boundary of top grasping workspace at opposite side.

TABLE I

AVERAGE PERFORMANCE WITH (STANDARD ERRORS) OF PA-POMCP AND GREEDY POLICIES UNDER SCENARIO 1 TO SCENARIO 6.

	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
PA-POMCP	$\mathbf{-129.0 \pm 5.2}$	$\mathbf{-106.0 \pm 2.4}$	$\mathbf{-236.0 \pm 11.1}$	$\mathbf{-258.0 \pm 21.1}$	$\mathbf{-210.0 \pm 10.7}$	$\mathbf{-158.0 \pm 7.8}$
Greedy-OS	-524.0 ± 11.6	-187.0 ± 3.7	-410.0 ± 22.2	*-Inf*	-533.0 ± 6.7	*-Inf*
Greedy-O	-555.0 ± 16.9	-1193.0 ± 2.6	-930.0 ± 34.9	-989.0 ± 71.4	-587.0 ± 11.8	*-Inf*
Greedy-S	-451.0 ± 18.4	-237.0 ± 10.8	-401.0 ± 17.8	*-Inf*	-3480.0 ± 203.7	-1682.0 ± 177.6
Greedy-T	-470.0 ± 19.0	-1229.0 ± 11.4	-946.0 ± 40.1	-1030.0 ± 69.9	-3392.0 ± 200.6	-1954.0 ± 157.7
Greedy-M	-645.0 ± 13.4	-500.0 ± 0.0	-592.0 ± 10.0	-776.0 ± 9.5	-3310.0 ± 236.1	-1850.0 ± 186.1

retaining the non-trivial ones. We then ran each method on each scenario once. This time, we compare PA-POMCP with other greedy policies based on both the average value and the success rate (see Table II). The results show that our method solves all the random scenarios and always finds the solutions while moving fewer objects than the greedy policies. Most of the greedy methods often fall into an infinite loop, focusing on moving a particular object away without realizing it is blocked by others.

VI. HARDWARE EXPERIMENTS

We performed experiments on a Fetch robot [24]. We evaluated our methods using the six scenarios described in Section V-A. The robot starts with an initial belief including 200 particles and runs 400 simulations per planning step to figure out the best action under the current belief. The robot successfully performed the task in each scenario, and the solutions found were the same as in our simulation experiments. Fig. 9 depicts the sequential decisions made by the robot in Scenario 6. According to the initial belief (see Fig. 8f), the target object is likely out of the current workspace, however, it can always be reached from the opposite side. The robot then moves to the other side of the table first, which is also the most efficient way to locate both hidden objects (tomato can and Lego). From this side, our observation model correctly detects the interval between the cube and the tomato can, so that the robot directly moves the cube away and fetches the target object in the end.

TABLE II

AVERAGE PERFORMANCE WITH (STANDARD ERRORS) OF PA-POMCP AND GREEDY POLICIES OVER 100 RANDOM SCENARIOS.

	Average Value	Success Rate
PA-POMCP	$\mathbf{-138.0 \pm 14.6}$	**1.00**
Greedy-OS	*-Inf*	0.79
Greedy-O	*-Inf*	0.46
Greedy-S	*-Inf*	0.84
Greedy-T	-878.0 ± 74.3	0.44
Greedy-M	-1300.0 ± 140.8	**1.00**

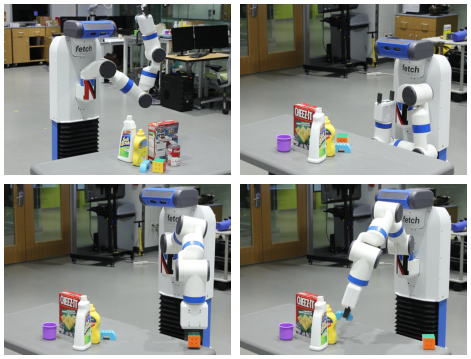


Fig. 9. Sequential actions performed by the robot with PA-POMCP in Scenario 6. (**Upper Left**) Initial settings. (**Upper Right**) Move base to the other side of the table. (**Lower Left**) Move cube away. (**Lower Right**) Fetch the target object (Lego).

VII. CONCLUSION

In this paper, we address the target object search problem in clutter with a POMDP model that can deal with intricate occlusion relationships between objects as well as manipulation uncertainty. We propose PA-POMCP that allows us to efficiently solve the POMDP model. PA-POMCP uses a novel run-time initial belief generator that reasons about the potential locations for hidden objects, parameterized actions to effectively reduce the action space, and a state value estimator to estimate the values at leaf nodes. Our experiments in both simulation and on hardware demonstrate the efficiency and robustness of our planning framework.

Acknowledgements. This research was funded by ONR grant N00014-17-1-2072.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[2] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems 23*, 2010, pp. 2164–2172.

[3] A. Aydemir, K. Sj, J. Folkesson, A. Pronobis, and P. Jensfelt, "Search in the real world: Active visual object search based on spatial relations," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2818–2824.

[4] Y. Ye and J. K. Tsotsos, "Sensor planning for 3d object search," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 145 – 168, 1999.

[5] K. Sjö, D. G. Lopez, C. Paul, P. Jensfelt, and D. Kragic, "Object search and localization for an indoor mobile robot." *Journal of Computing and Information Technology*, vol. 17, no. 1, pp. 67 – 80, 2009.

[6] A. Aydemir, M. Göbelbecker, A. Pronobis, K. Sjöö, and P. Jensfelt, "Plan-based object search and exploration using semantic spatial knowledge in the real world," in *Proceedings of the 5th European Conference on Mobile Robots (ECMR)*, 2011.

[7] L. L. S. Wong, L. P. Kaelbling, and T. Lozano-Prez, "Manipulation-based active search for occluded objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2814–2819.

[8] M. Dogar, M. Koval, A. Tallavajhula, and S. Srinivasa, "Object search by manipulation," *Autonomous Robots*, October 2013.

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[10] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 447–454.

[11] J. K. Li, D. Hsu, and W. S. Lee, "Act to see and see to act: POMDP planning for objects search in clutter," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 5701–5707.

[12] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: Online POMDP planning with regularization," in *Advances in Neural Information Processing Systems 26*, 2013, pp. 1772–1780.

[13] J. Pajarinen and V. Kyrki, "Robotic manipulation of multiple objects as a POMDP," *Artificial Intelligence*, vol. 247, no. C, pp. 213–228, 2017.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," in *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, 2012, pp. 1–43.

[15] S. Katt, F. A. Oliehoek, and C. Amato, "Learning in POMDPs with Monte Carlo tree search," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 1819–1827.

[16] Z. N. Sunberg and M. J. Kochenderfer, "Online algorithms for POMDPs with continuous state, action, and observation spaces," in *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS)*, 2018, pp. 259–263.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[18] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Robust grasping under object pose uncertainty," *Autonomous Robots*, vol. 31, no. 2–3, 2011.

[19] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[20] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, Dec. 2017.

[21] R. Girshick, "Fast R-CNN," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[22] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.

[23] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2008.

[24] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch & freight : Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots, International Joint Conference on Artificial Intelligence*, 2016.