MDP-based Planning for a Table-top Search and Find Task

Ram Kumar Hariharan, Kelsey Hawkins, Kaushik Subramanian

Abstract— **For mobile manipulators in human environments, a basic yet important task is object fetching. Even if the robot knows which table the goal object is on, it is often in a cluttered environment. The goal object could be partially occluded, underneath other objects, or completely covered. We investigate the task of optimally locating and grasping a goal object in a cluttered environment. Since, to a robot, the world is rife with uncertainty, we approach the task using two distinct Markov Decision Process (MDP) representations. Our first design is a grid-based representation which takes a very simplistic Partially Observable MDP (POMDP) approach in the case where vision is highly unreliable. The representation is solved using Point Value Iteration and the advantages are discussed. Our second approach attempts to exploit a more informed vision system. A tree of obstructing objects is gathered from the scene and planning is done over the possible tree configurations. Confidence values from recognition are weighed with expected graspabilities and whether to terminate early. We show that the search task can be parameterized to exhibit different behaviors but cannot be objectively optimized. Evaluation is performed by analyzing a few scenarios to show the types of behavior being demonstrated.**

I. INTRODUCTION

The problem of optimally searching for objects is fundamentally difficult. For example, people often lose common objects such as keys or remote controls in their house. Vision clues can signal whether the object might be behind something in one part of the room. Heuristics might give likely potential locations for the object. Even though the person might have ideas for the location of the object in one room, they could perform an exhaustive search of a single room only to find it was hidden in plain sight in another room. While searching a house is time consuming for a human, searching a table is time consuming for a robot. Since even the best grasping implementations take upwards of 25 seconds for a single grasp, the robot must search optimally for it to be of any usefulness in a domestic setting. This is the problem we attempt to address.

There are numerous challenges for the tabletop search environment:

- Uncertainty in scene structure
- Uncertainty in goal recognition
- Hidden objects
- Grasping failure
- Possibility the goal is not on the table
- Slow grasping
- Physics

To address some of these challenges, we recognized that MDPs can capture much of the uncertainty of the problem. MDPs are stochastic decision making algorithms which can handle uncertainty in manipulation. POMDPs extend this framework to include state uncertainty as well. Since the robot is uncertain about the location of the goal, this information can be encoded as a state of the world and POMDPs can give an optimal policy in the presence of this uncertainty. We implement two separate approaches to the problem based on different representations. The first representation uses a simplistic perception of the world based on dividing an image into discrete grid cells. The second representation exploits a precise description of the environment and effectively utilizes available sensor data. Either algorithm might be applicable depending on the robot application and the sensors available. These characteristics are explained in individual sections describing the representations and algorithms.

The following section describes the Related Work. Section III motivates the grid-based representation and evaluates it using the PBVI algorithm. This is followed by Section IV which describes the obstruction tree based representation and illustrates how it can be used in realistic scenarios. Finally in Section V, we provide our concluding remarks and a notes on future work.

II. RELATED WORK

Work on robotic search and find tasks has been primarily been on search and rescue task. Search and rescue has been mentioned as one of the scientific applications of POMDP in the survey paper by Casandra [3], but POMDP or Q-MDP is difficult to apply because such approaches would be highly computationally demanding. This is especially true for complex environments where the state space is constantly changing.

Work by Yiming Ye et. al [4] on sensor planning for object search is related. In their work, the search space is characterized by the probability distribution of the presence of the target and the goal is to reliably find the desired object with minimum effort. However, there are two significant differences. Their work is on sensor planning and vision for a particular scene rather than a general search algorithm. Also, they do not work on scenarios where the goal object might be hidden. Our work is targeted towards planning the search and find task for both currently visible and potentially hidden objects.

The work by Lambert E. Wixson [2] to exploit the world structure to efficiently search for objects is related to our use of occluded volume to search for goal objects. Again, it cannot be directly related or compared because their work involves sensor planning rather than object manipulation planning.

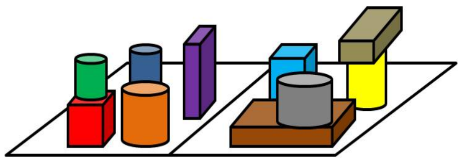


Fig. 1. Two cell grid world containing random objects.

III. OCCUPIED GRID CELL STATE REPRESENTATION

Consider a tabletop scene containing a random set of objects. We divide this tabletop into grid cells and make note of the cells which contain objects. We refer to these cells as "occupied cells". An example of this representation can be seen in Figure 1. The world consists of two grid cells and each cell consists of multiple objects of different size and color [1]. Each cell can be in one of 4 configurations. The cell could be empty, occupied by non-goal objects, contain the goal object partially occluded, contain the goal object fully visible. We make use of two camera angles of the scene, both a front view and a top view. Using the two camera views allows for objects to stacked along two dimensions (both length and height wise). On enumerating the possible combinations we arrive at a state complexity of $2n^4$ where n is the number of grid cells. But in reality several of these states can be pruned because we would never encounter them. For example a state which is empty from the top view but has the goal in the front view would never be possible in reality. After pruning we arrive at 36 possible states for a two-grid world. The advantage of this representation is that it does not depend explicitly on the number objects in the scene and the state space does not dynamically change during the course of action. It thus makes the problem tractable for computation-heavy algorithms.

In this abstracted space, our task is to find a particular object of desire [2]. We attempt to solve this problem using POMDP algorithms.

A. *Partially Observable Markov Decision Processes*

An POMDP is a tuple $\langle S, A, \mathcal{T}, R, \gamma \rangle$ with states S, actions A, observations O, transition functions $\mathcal{T} : S \times A \mapsto \Pr[S]$, observation function $\Omega : S \times A \mapsto \Pr[O]$, reward function $R : S \times A \mapsto [R_{min}, R_{max}]$, and discount factor γ. The observation function here allows us to mathematically model uncertainty with respect to the sensors. This information allows us to probabilistically estimate which state of the world we are most likely in. A simplistic example of this is the game of Blind mans bluff. The player is blindfolded and he/she must touch the other players. The blindfolded person is like a POMDP solver, they have no vision and therefore are never sure of where they are. They move randomly (actions) and use their sense of touch on their hands and body (observations) to narrow down on where they are most likely to be. This model therefore appears to fit well in our search and find scenario.

[1]The tabletop can be further discretized depending on the resolution required

[2]We take into account the possibility that the object may not be present on the table

In a POMDP we maintain a probability distribution over all possible states and continue to update these values based on the actions taken and the observations received. Such a probability distribution is known as a belief state. Given an initial belief state $b(s)$, then after taking action a and observing o, the new belief state is updated as -

$$b'(s') = \eta \Omega(o|s', a) \sum_{s \in S} T(s'|s, a) b(s) \qquad (1)$$

where $\eta = \frac{1}{P(o|b,a)}$ is a normalizing constant with

$$P(o|b, a) = \sum_{s' \in S} \Omega(o|s', a) \sum_{s \in S} T(s'|s, a) b(s) \qquad (2)$$

The POMDP framework has been well studied and it is notoriously difficult to obtain an exact solution for any problem. The complexity lies in state space representation of the POMDP. It is a continuous space with the number of dimensions equal to the number of states of the underlying Markov Decision Process (MDP). There is no longer a nice tabular format for the value function in this continuous space. There have been several approximate methods proposed both offline and online [8] and some have been used in real-world applications. We make use of an offline algorithm - Point-based Value Iteration algorithm (PBVI) [7] and test its utility on our grid-based representation. The algorithm samples belief states from the continuous space and runs random episodes from these states. It then runs value iteration on the sampled beliefs to generate a optimal set of alpha vectors (value function parameters). The algorithm has desirable properties such as efficiency in value function computation and optimality with respect to the policy [7]. We test each of these properties in our domain.

To setup the problem, we define the following specifics -

- State Space - We setup the state space according to the grid based representation described earlier. We use two grids cells.
- Actions and Transition Function - Our action space consists of five actions - remove top most object in cell 1 and the same for cell 2, remove the object closest to you in cell 1 or in cell 2 and do nothing. The actions remove the objects from the cluttered scene and place them in an open uncluttered area. We take into account stochastic transitions - we allow for each action to fail $\frac{1}{3}$rd of the time.
- Observation Function - Our observations consist of observing the color of the goal object. These observations can be made in either camera (front/top) and in either cell. Enumerating the combinations, we arrive at 17 possible observations. The Observation function defines the probabilities of receiving each of these observations in every state. The sum of the probabilities of getting these observations in a single state is 1.

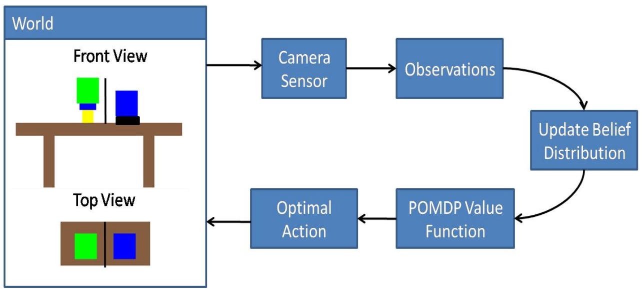


Fig. 2. A block diagram showing how the different components are connected.

- Reward Function - The aim of our task is to efficiently find our object or return saying the object is not present on the table. The reward function for this task must be setup carefully due to mutual dependence of the goals. The task of finding a specific object and the task of clearing a table do not necessarily follow the same plan. Therefore we setup the function in a manner that gives maximum reward to finding the object $R = 1$, a smaller reward for clearing the table $R = 0$ and $R = -1$ everywhere else.
- Assumptions - To make the problem tractable for the PBVI we assume that we know the color of the goal object we are looking for and that it has a unique color in the scene.

An initial prior belief state is built by taking into account the observations (camera data) at time $t = 0$. We then perform offline planning. Once algorithm has converged, we begin to take actions. Figure 2 provides a summary as to how the different components are connected.

B. Evaluation of POMDP Planner

We first test the efficiency of the PBVI POMDP planner to compute the optimal value function for the search and find problem. We incrementally increase the complexity of the problem (number of states) and note the time taken to compute for convergence. We use a discount factor of 0.9. A table of these results can be found in III-B.

Number of States	Time taken in seconds
8	6.50
12	26.66
36	76.93

While the number of states are relatively small, it shows that the algorithm is reasonably efficiently for our representation of the problem. This can be attributed to the preprocessing steps of the algorithm. The algorithm samples belief states from the belief space and performs a sequence of random actions until the episode ends. In this manner, it evaluates the action space and at the same time is able to prune the belief space to a set of reachable belief states.

We then tested the policy obtained from PBVI to find goal objects. For this purpose we setup a simulated version of the tabletop task in Python. We allowed for random transitions

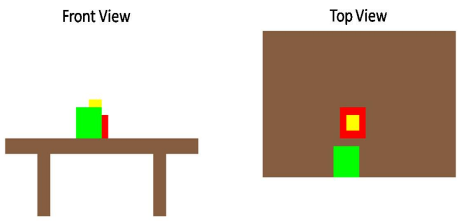


Fig. 3. Simple Simulated Scene. Goal is the partially visible yellow object.

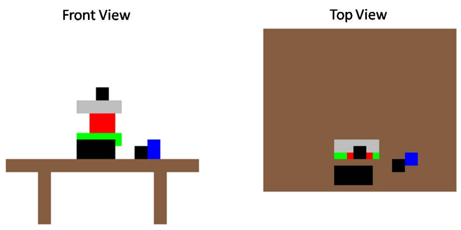


Fig. 4. Complex Simulated Scene. The goal object cannot be seen in this scene.

and observations within the simulator in order to replicate real-world conditions. The goal in all of our test cases is to find a yellow colored object. An image of a sample simulator task is shown in Figure 3. It shows two camera views of the tabletop scene in which the front view has the goal partially occluded. A more complex scene is shown in Figure 4 where the goal is not visible any of the camera views.

Given multiple such configurations, we executed the POMDP policy to obtain the goal. A summary of the results obtained is shown in Figure 5. We find that the PBVI algorithm is able to achieve the goal object in all the test cases. The numbers are slight greater than the Optimal due to stochastic transitions and observations encoded within the simulator. The stochasticity temporarily diverts the weight of the belief space to another location but as more actions are taken and observations are received, it eventually continues to find the goal object. This is due to inherent Markov property of the belief state. At any point in time, it encodes all the past history of actions taken and observations received.

The PBVI algorithm used on the grid representation was found to be efficient in computing the value function. The policy was optimal in terms of the sensor and actuator imprecision. The algorithm in itself is complete, however the execution of the policy is dependent on characteristics of the real-world as it true for any algorithm. The worst case performance of the algorithm is when it removes all the objects incrementally one by one until there is nothing left on the table.

Given the above results, we find that we are able to efficiently solve simple problems where there are only two occupied grid cells. However there is significant amount of information that we are not yet utilizing in our representation. In the real world we would like to use more information to further direct our search. These can be elaborated as -

- The grid representation flattens out the world and does

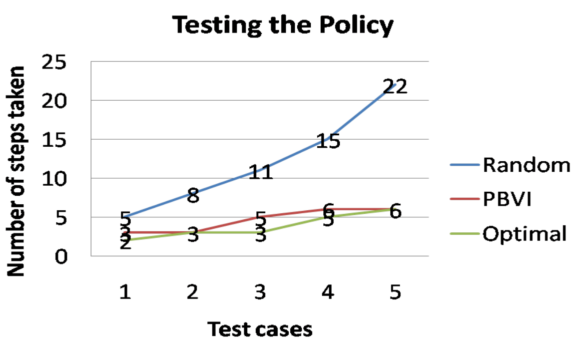


Fig. 5. Comparison on Test Cases.

not take into account the connectedness of the objects. By taking into account how one object is connected to another, we could make intelligent decisions when searching for an object.

- Our assumption that there can be no multiple goal colored objects is not realistic. We are likely to see tables with similarly colored objects. We could incorporate higher resolution spaces.
- Our action space is quite simplistic. We can advance only one step at time by removing only a single object. We limit them to make the representation tractable for the POMDP algorithm.
- We could take into account the fact that an action may fail stochastically and this may change the world (position of objects on the table). In this case we could replan (using a faster algorithm) given the current configuration of objects. This would not be possible using PBVI.
- We could bias the search by including more sensor data based on occluded volume, object recognition, graspability and so on. Combining this data with the PBVI makes it computationally expensive.

The PBVI algorithm with grid representations showed us that we could solve simple problems efficiently. We performed a test by increasing the number of observations, expanding the action space and increasing the state space. On this refined space we performed PBVI to compute the value function. We note the taken taken grows to a very large amount. Therefore as we incorporate the characteristics mentioned above, the complexity of the space and hence the algorithm grows to a large extent. It is clear that we are not utilizing significant data because of our assumptions and simplistic state representation. Using the results we obtained here we further expand on our representation to a tree based approach. To make the planning tractable in this representation, we use an MDP approximation of the POMDP known as QMDP.

IV. OBSTRUCTION TREE BASED REPRESENTATION

The grid-based state representation has a fundamentally simplistic concept of the possible states of the world which restricts the amount of inference that can be made into the results of the robot's actions. However, if computer vision is able to provide a more complex description of the tabletop object relationships, more informed decisions can be made about the optimal object to grasp. Our more precise state representation, Obstruction Tree-based, assumes that vision is able to segment the objects from each other and determine which objects are *obstructing* others. Though the specific algorithm is outside the scope of this paper, following is a short description of how this relationship can be computed. Suppose a color image and a segmented 3D point cloud is available to the robot which represents the different objects on the table. For each object we want to find whether any other object is occluding this object. First, take the point cloud clusters, project them onto the image, and find the minimum distance between points of each cluster. If this distance falls beneath a threshold, there is an occlusion boundary between these two objects. If the two closest points in projected space have a distance above a threshold in unprojected space, the object further away is the object being occluded. If the points are very close, such as in the case of one object resting on another, points around the occlusion boundary can be sampled to find out which object is on top. Once an occlusion relationship $<_{occ}$ can be determined between every pair of objects, the obstruction relation $<_{obs}$ is generated from its transitive closure, along with a $nothing$ element that obstructs everything. An object o_i is obstructed $obs(o_i)$ when $\exists o_j | o_i <_{obs} o_j, i \neq j, o_j \neq nothing$. An Obstruction Tree is the tree which represents the relationships between all of the objects in the scene. For each visible object, a potential object is added to the tree to represent the possibility that one object is behind each visible one. Potential objects cannot be grasped, but they must be considered as possible goals.

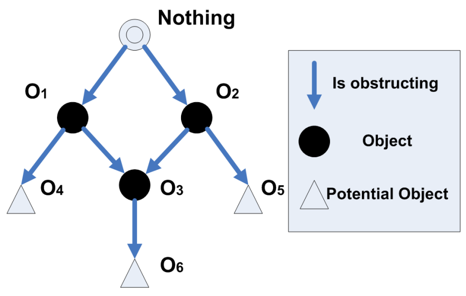


Fig. 6. A Sample Obstruction Tree

A. *Table States and Actions*

Once the initial obstruction tree is determined, future obstruction trees can be predicted in the case of a successful grasp. Trees are generated for all possible combinations of removing any number of visible objects in no order. The possible actions for each tree configuration include grasping any visible object, grasping one of the unobstructed objects and returning it as the goal, and quitting the search. Allowing the robot to quit the search is a feature which is essential in a multi-table task. If the robot is a mobile manipulator searching multiple tables for a goal object, it should stop searching once its confidence about the goal being off the table is high enough. Though an obstruction tree encodes the structure of the table, a full state also includes the identity of the goal object, or that none are the goal object. Thus, the set of all possible table states is the Cartesian product of the obstruction trees with the goal location labels. Actions only connect obstruction trees within each possible goal location "'world"'.

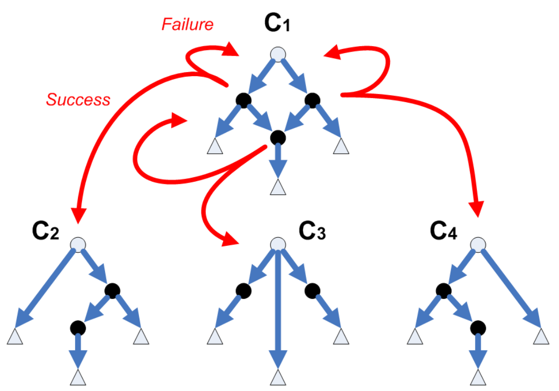


Fig. 7. The first future obstruction trees with connecting grasping actions. Each red arrow pair represents a grasp of that object with both its consequences.

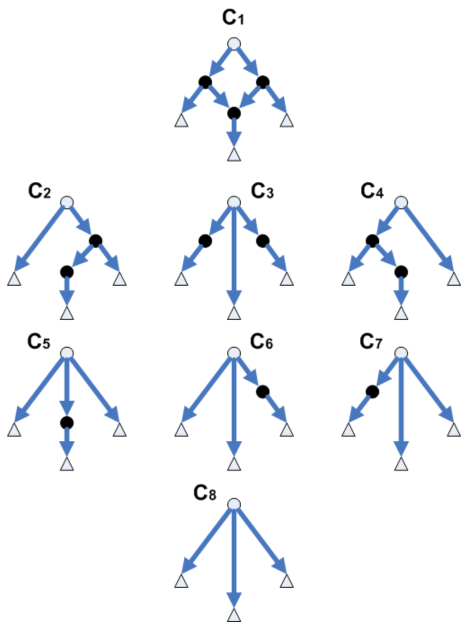
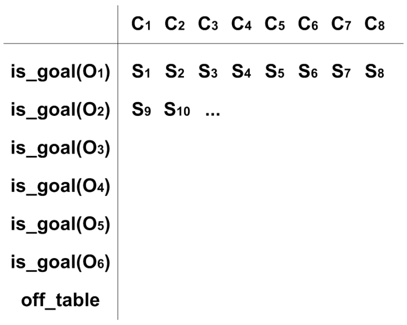


Fig. 8. The enumerated obstruction trees for the three object world.

It is important for the agent to understand whether or not an object is obstructed and in what conditions the object is unobstructed because graspability will change once the object is uncovered. Graspability is a probabilistic measure of how likely the robot will successfully grasp the object. For example, for some grasping implementations, larger objects



	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
is_goal(O_1)	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
is_goal(O_2)	S_9	S_{10}	...					
is_goal(O_3)								
is_goal(O_4)								
is_goal(O_5)								
is_goal(O_6)								
off_table								

Fig. 9. A grid representing the all of the possible table states for the three object world.

are more difficult. If a vision algorithm is available to predict graspability for the objects on the scene, the robot can plan to avoid grasps which are more difficult. Ideally, an algorithm should produce the current graspability of unobstructed objects and for obstructed, both when the object is obstructed and the predicted value when it is unobstructed. A non-zero graspability for obstructed objects indicates that the robot is welcome to attempt to grasp objects behind or underneath others if it deems uncovering the object first more costly. The graspability value is used as the transition function for the actions on each tree configuration from before. In case of a failure, it naïvely expects the state to remain unchanged. A more complex function is difficult to use because the actual result of a grasping action is highly unpredictable for robots.

B. *Belief Distribution*

This representation also allows for the incorporation of recognition probabilities and potential object discoveries to form a belief distribution over which goal location "'world"' the robot is in. Suppose a vision algorithm is available which provides, instead of a binary true or false recognition observation, a heuristic that estimates the probability that this object matches goal. Also assume that the heuristic can also be applied to partially occluded objects. Using these probabilities will both give the robot a good hypothesis about the location of the goal, and provide confidence confirmation once the suspected object has been uncovered. If the vision has low confidence even when the goal object is uncovered, it might want to consider whether further exploration is worth increasing its belief of the currently most likely goal.

Even when the goal object is completely occluded, inference can be made about its location. Since the robot is looking for something it can recognize, we can expect the volume of the object can be easily estimated. For a tabletop scene with a robot looking over, the only possible places in which the goal is fully occluded is in the volume formed by the projection of the visible surface of those objects onto the table. If the occluded volume is large with respect to the goal, the possibility the potential object behind the visible is the goal increases. An simple estimate for the probability that the goal is behind an object is given by

$$\Pr(isgoal(o_j)|behind(o_j, o_i)) = \alpha \max(0, 1 - \frac{vol(goal)}{occvol(o_i)}) \tag{3}$$

where $0 < \alpha \leq 1$ is a constant that represents the general likelihood of any object being fully occluded along with the probability that such an object has similar volume to the goal.

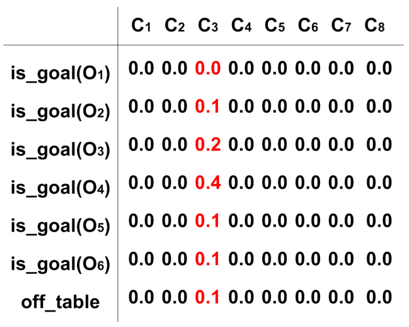
Using these goal probabilities we create a belief about the identity of the goal, a probability distribution over all visible objects, every potential object behind the visible ones, and the possibility it is not on the table. One useful assumption to make is that there is only one goal on the table. In this case, the belief can crafted to represent the exclusive probability that one object is the goal and every other object is not the goal. Since the recognition and full occlusion probabilites from before should be independent, the joint probability can be broken up.

$$\begin{aligned} b(isgoal_e(o_i)) &= \Pr(isgoal(o_i) \wedge \bigcap_j \neg isgoal(o_{j \neq i})) && (4) \\ &= \Pr(isgoal(o_i)) * \prod_j 1 - \Pr(isgoal(o_{j \neq i})) && (5) \end{aligned}$$

Finally, the possibility the goal is not on the table is computed as

$$b(offtable) = \prod_i 1 - \Pr(isgoal(o_i)) \tag{6}$$

Once the belief values are computed, they are normalized to sum up to 1. This implementation assumes full observability over obstruction trees but partial observability over goal locations. The total belief over all of the states in the set described in the previous section will be non-zero for all other tree configurations.



	C1	C2	C3	C4	C5	C6	C7	C8
is_goal(O1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
is_goal(O2)	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0
is_goal(O3)	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0
is_goal(O4)	0.0	0.0	0.4	0.0	0.0	0.0	0.0	0.0
is_goal(O5)	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0
is_goal(O6)	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0
off_table	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0

Fig. 10. The full belief over all the possible table states.

C. Obstruction Tree Planner: Q-MDP

Though this representation can be implemented as a POMDP, an offline policy cannot be generated because replanning is required at every step. When the robot fails a grasp, the resulting disturbances, collisions and physics make the following tree configuration difficult to predict. Furthermore, even if the grasp is successful, a complex sub-configuration could be revealed behind the object. General POMDP's, as shown by our previous work, can take minutes or hours to compute a policy. Since a timeframe longer than a few seconds is unacceptable, for this task, a Q-MDP planner is used to quickly estimate a POMDP [1]. The planner works by simply assuming only one step of uncertainty and not trying to explicitly predict what the belief distribution will look like at each step. Specifically, for each goal location, MDP value iteration is performed assuming that this goal is the actual goal. For each of the current actions, the Q-value is computed using the V-values from value iteration.

$$\begin{aligned} Q_{MDP}(a, s) &= R_a(s, s') + \gamma \sum_{s'} \Pr_a(s, s') V_{MDP}(s') && (7) \\ a^* &= \operatorname*{argmax}_a \sum_s b(s) Q_{MDP}(s, a) && (8) \end{aligned}$$

There are four parameters to the planner which must be set based on the type of behavior the user would like the robot to exhibit. These parameters form the reward functions of the underlying MDP and include a reward for grasping and returning the goal object, a cost for executing a grasp, a penalty for returning the wrong object, and a penalty for quitting to say the object is not on the table. First, the goal reward should be set to some positive value. The precise value is unimportant because it is relative rewards which change the actions. Next, the cost of every grasp action should be set to reflect the amount of time an average grasp takes. If the grasp cost to goal reward ration is very high, the planner will try to exit as soon as possible and report that there is no object on the table. If set very low, the robot will temporarily pass over the goal, looking for objects in every location until the certainty of the belief state is at a maximum. The most useful setting is somewhere in the middle of these extremums. There is also a cost assigned to returning the incorrect object. A high incorrect object cost to correct object reward ratio will not terminate if there are other significant goal beliefs. A low ratio will go for the easiest significant belief and return it, regardless of whether it is even the maximum belief state. Again, this must be tuned for desired effect. Cost is also assigned to quitting the table, a suggestion as to how many more grasps is worth looking for the object if it probably is no on the table. A fifth optional parameter could be to assign differential costs to grasping each of the objects based on task constraints such as expected fragility. Though a larger penalty with respect to the other objects could still result in the object being moved, it will try to avoid the object if reasonable.

D. Evaluation of Q-MDP Planner

When assessing the optimality of this approach, it should be recognized that, as discussed in the previous section, the behavior of the robot is highly dependent on what is important to the user, the quality of the vision algorithms. For each set of parameters, Q-MDP finds approximately optimal policies given that future belief states cannot be faithfully predicted. The existence of these tradeoffs provides insight into the general task of searching for an object on a table. If the vision system provided is imperfect, there is no objectively optimal solution to this problem. In addition, if time is an important constraint for the user, the task of finding the goal will not be complete and might quit before actually finding the goal. Also, even if the vision is imperfect and the objective is to find the most likely goal object, under certain parameters the planner will be unsound and can return the less likely of two goals. While these tradeoffs seem costly, the gain is far more important in a robot mainpulation environment, reducing the amount of time spent at the table.

Since both the parameters of the algorithm and the probabilities recieved from the world are highly nondeterministic, it is unrealistic to perform empirical trial testing with respect to the grid implementation or some other implementation. Instead, scenarios which demonstrate desirable behavior will be described. For these scenarios, yellow designates the goal object and the robot's vision is from the directly above perspective.

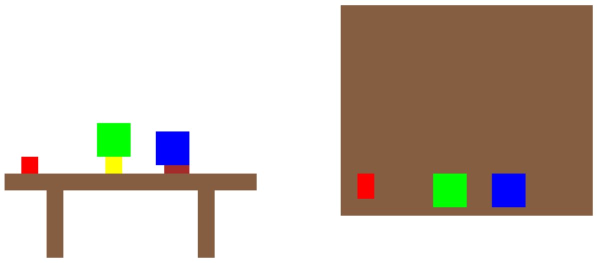


Fig. 11. A table setup with a goal (yellow) that has a low but significant recognition value. Both blue and green are easy to grasp, red is harder.

One capability the planner gives is the ability to accommodate weak recognition values. This means the robot should return the object only after it is confident that other possible objects are not the goal. For the table in figure 11, the green block is first removed because it is easier than the blue or red, and the robot sees the unobstructed goal. While the recognition value is significant, it is not high $(\Pr(isgoal(o_i)) = 0.5)$. Since the blue object is occluding a lot of volume, the robot decides it should look under the blue block next. There, it sees the brown object but recognizes it with a low value. Having decreased its uncertainty, the robot now returns with the goal object. This plan is considered optimal because the brown object could have just as easily been the goal with a higher recognition probability. Under the same scenario, a higher recognition value will have the robot return the object immediately after uncovering it. A lower recognition value will prompt the robot to look under the red object as well, despite it being hard to pick up.

This planner can also handle multiple goal possibilities. In figure 12, the left yellow object is the actual goal and the right object is a decoy that looks like the goal when partially occluded. While the left object is marginally more likely the goal, the red object is more significantly more difficult to pickup than the black object. Thus, the robot decides to

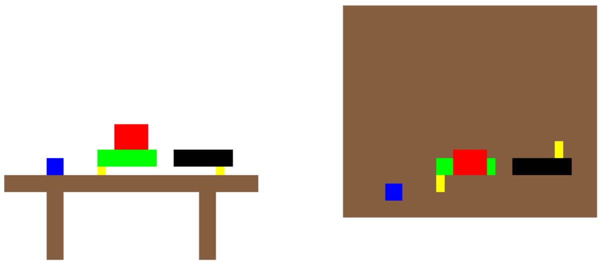


Fig. 12. A table setup with both a goal (left yellow) and a decoy (right yellow). Red is difficult to grasp and black is easy.

pickup the black object first and inspect the decoy. Once it realizes it is not the goal, it continues by lifting the red object followed by the blue and returning the goal. Again, this is optimal because the algorithm should favor good leads weighted by their ease.

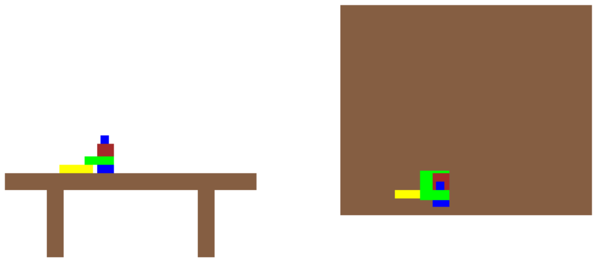


Fig. 13. A table setup where the goal (yellow) has a small but significant current graspability. The other objects have somewhat high graspabilities.

The planner can also pull out objects from underneath stacks if the graspability is high enough. For figure 13, the algorithm immediately removes the goal object, then recognizes it and returns it.

The significance of the Obstruction Tree-based representation is its ability to incorporate all of the above capabilities into a single framework. Though any one of the behaviors could be manually added into a naïve approach, this planner can handle all these problems continuously and concurrently.

Since speed is a priority for this algorithm and re-planning is done at every step, planning must be fast. Specifically, it must run in less than 10 seconds and ideally less than 3 seconds for it to be beneficial for real-world applications. The number of states in the underlying MDP is $\mathcal{O}(N2^N)$ where N is the number of visible objects. In basic implementation, value iteration will run N times in a state space of that size. Empirical testing shows that the planning becomes costly for $N \geq 7$ (3.5 seconds for $N = 7$, 10 seconds for $N = 8$). There are a few modifications that can be made to reduce running time without compromising optimality greatly. When belief about the location of the object is low, the Q-values and consequently V-values for those beliefs do not need to be calculated because they will drop out. Our first modification pruned the number of value iterations to the top 7 beliefs. This proved to have only modest improvements, shifting the running time numbers up one object (3 seconds for $N = 8$, 10 seconds for $N = 9$). Decreasing the discount value also trimmed about a second off the costs, but clearly sub-optimal policies were apparent for $\gamma < 0.99$. It is clear that the barrier to faster computation is the large number of states for the MDP. Though there are methods for prioritized state pruning, they have not been implemented.

V. CONCLUSIONS AND FUTURE WORK

A. Conclusions

On reviewing the results, we find the quality of the policy obtained to be closely tied with the representation and the algorithm used on top of that. The grid-based representation was primarily a flattened world with limited capabilities. It was tractable for computationally expensive POMDP algorithms and was able to solve simple problems. However when incorporating additional sensory information into the state space, the algorithm ceases to be efficient. To overcome this, we defined a new representation, obstruction tree-based and combine it with a POMDP approximation, the Q-MDP. We find that it is able to leverage multiple capabilities of the robot, uncertainties in the environment, and at the same time, perform dynamically based in a changing environment. We also note that since the representations and expected observations are different, they are not directly comparable.

B. Future Work

There are a few improvements to the Obstruction Tree representation to make the robot more capable. As mentioned earlier, prioritized value iteration can be implemented to decrease running times and increase the number of objects the robot can feasibly reason on. Currently, the Obstruction Tree-based representation only allows for grasp-and-place actions to change the structure of the world. While grasping is often difficult for a robot, pushing is much easier. Allowing the robot to push objects out of the way or topple stacks can give the robot a more reliable way of uncovering goals. Also, the representation assumes that there is always open space available to place removed objects. Realistically, in a cluttered environment, open space is limited. Adding constraints into the planner will make the algorithm more robust.

Real-world implementation is also a priority for future development. Obstruction Tree planning assumes certain vision algorithms are available to the robot for generating input. Implementing some of these algorithms will allow us to discern whether and which vision assumptions are realistic and discuss solutions and the effects on the performance of the planner. With working vision algorithms, we hope to implement our work on Willow Garage's PR2 Personal Robot for realistic evaluation. The results of testing with an actual robot will be both more relevant and will capture the actual effects of failures in vision and grasping.

REFERENCES

[1] M. Littman, A. Cassandra, and L. Kaelbling, "Learning policies for partially observable environments: Scaling up (1995)", *Machine Learning: Proceedings of the Twelfth International Conference*, Tahoe City, CA, 1995, pp. 362-70.

[2] Lambert E. Wixson, "Exploiting World Structure to Efficiently Search for Objects", *Technical Report 434*, Dept. Comp. Sci, Univ. of Rochester, 1992.

[3] Anthony R. Cassandra, "A Survey of POMDP Applications"

[4] Yiming Ye and John K. Tsotsos, "Sensor planning for 3D object search", *Comput. Vis. Image Underst.*, February 1999.

[5] H. Kwakernaak and R. Sivan, *Modern Signals and Systems*, Prentice Hall, Englewood Cliffs, NJ; 1991.

[6] D. Boley and R. Maier, "A Parallel QR Algorithm for the Non-Symmetric Eigenvalue Algorithm", *in Third SIAM Conference on Applied Linear Algebra*, Madison, WI, 1988, pp. A20.

[7] Joelle Pineau and Geoffrey J. Gordon and Sebastian Thrun, "Point-based value iteration: An anytime algorithm for POMDPs", *in IJCAI 2003*.

[8] Stephane Ross and Joelle Pineau and Sebastien Paquet and Brahim Chaib-draa, "Online Planning Algorithms for POMDPs", *Journal of Artif. Intell. Res. (JAIR)*, vol 32, year 2008.