2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Interactive Environment Exploration in Clutter

Megha Gupta*, Thomas Rühr†, Michael Beetz‡ and Gaurav S. Sukhatme*

Abstract— **Robotic environment exploration in cluttered environments is a challenging problem. The number and variety of objects present not only make perception very difficult but also introduce many constraints for robot navigation and manipulation. In this paper, we investigate the idea of exploring a small, bounded environment (e.g., the shelf of a home refrigerator) by prehensile and non-prehensile manipulation of the objects it contains. The presence of multiple objects results in partial and occluded views of the scene. This inherent uncertainty in the scene's state forces the robot to adopt an observe-plan-act strategy and interleave planning with execution. Objects occupying the space and potentially occluding other hidden objects are rearranged to reveal more of the unseen area. The environment is considered explored when the state (free or occupied) of every voxel in the volume is known. The presented algorithm can be easily adapted to real world problems like object search, taking inventory, and mapping. We evaluate our planner in simulation using various metrics like planning time, number of actions required, and length of planning horizon. We then present an implementation on the PR2 robot and use it for object search in clutter.**

I. Introduction

Exploration of our environment is a commonplace occurrence in our everyday lives. Think about looking for a matching pair of socks in your drawer, finding a spot to place a milk carton in your already overstocked refrigerator, searching for and counting coins to do your laundry - the examples are endless. The exploration is made tougher by the fact that our homes are cluttered - unstructured, and containing a lot of objects. The location of the same object may also vary from home to home. The good news is that we usually succeed at the task and find what we were looking for in reasonable amounts of time. An important factor that facilitates or even enables this exploration is our ability to manipulate the world around us.

It is this idea of interactive exploration in clutter that we address in this paper. The environment to be explored is bounded and small, however it is only partially observable. The robot can manipulate the world to observe more of it but the kind of actions allowed include only rearrangement of the objects - they cannot be permanently removed. This setup suggests an observe-plan-act strategy where perception, planning, and execution have to be interleaved since execution of an action may result in revealing information about the world that was unknown hitherto, and hence a new plan needs to be generated.

The authors are with the Robotic Embedded Systems Laboratory and the Computer Science Department, University of Southern California, Los Angeles, CA, USA*, Technische Universität München, Munich, Germany†, and University Bremen, Germany‡ meghagup@usc.edu, ruehr@in.tum.de, beetz@cs.uni-bremen.de, gaurav@usc.edu

We present a multi-step lookahead algorithm which plans for a sequence of actions that will give the maximum reward under the current information about the world state. We evaluate our planner for planning time and number of moves required using simulations over a large set of scenarios. Simulation results show that as compared to rearranging objects greedily or randomly, our proposed algorithm guarantees complete exploration of the environment (unless no valid moves are left) and requires a fewer number of moves to do so. These savings in number of actions are even more significant as the number of objects in the scene increases. This is important since manipulation by a real robot is usually very slow and thus, we want to minimize the number of moves.

We then present an implementation of the algorithm on the PR2 robot and apply it to object search. The robot has a fixed and partial view of the scene but has access to two manipulation primitives - pick and place, and push. We present preliminary results with a pipeline that uses these simple motion primitives to rearrange a cluttered scene in ways that locate the target object finally.

Our algorithm interleaves adaptive lookahead planning with object manipulation and has direct applications in real world problems like object search, object counting, and scene mapping. Our work simultaneously falls in the categories of search and exploration, sequential decision making, and interactive perception. We discuss the relevant related work in each category in the next section.

II. Related Work

The problem of exploration and object search in different kinds of environments has been studied extensively. *Active visual search* ([1], [2], [3]) searches for a target object lying out in the open but the challenge is to decide where to move the camera to locate the target. In *pursuit-evasion* ([4], [5], [6]), the problem is to come up with an exploration strategy for the pursuer such that all evaders are detected quickly. This has been studied under various constraints on the number of pursuers and evaders, visibility range, discrete and continuous world, etc. Work on *search and rescue operations* ([7], [8]) focuses on efficient algorithms for robot navigation and coordination of a team of robots to search a disaster site thoroughly.

Partially Observable Markov Decision Processes, or POMDPs, are a popular planning framework for *sequential decision making* under uncertainty in states and state transitions. However, a POMDP becomes intractable very quickly as the size of the state space grows. Heuristic-based algorithms like A^*, D^*, and D^*-lite ([9], [10]) work well

when the goal state is known and a good heuristic is used. Our problem of environment exploration requires planning under uncertainty but the goal state is unknown, and there is no prior on object locations.

Traditionally, research on mobile manipulation considers collisions with obstacles unacceptable. More recently, there has been some work on *interactive perception* where the robot actively manipulates the world to complete the overall task at hand. Fitzpatrick [11], Chang [12], and Schiebener et al. [13] use interaction with objects for segmentation and recognition of unfamiliar objects; Katz and Brock [14] employ it to obtain a kinematic model of an unknown object and use it for purposeful manipulation; Gupta and Sukhatme [15] use deliberate interactions with objects to sort them from clutter; and Dogar and Srinivasa [16] use it for more effective grasping in clutter.

There has been some very recent work on *object search using manipulation*. Wong, Kaelbling, and Lozano-Pérez [17] use spatial and object co-occurrence constraints to guide the search. Objects are permanently removed from the scene in the course of the search. Dogar et al. [18] prove theoretical guarantees about their algorithm being optimal within certain conditions, such as a perfect segmentation of the scene, recognition of the objects (that have known shape) and the possibility to remove all objects from the scene. They assume that the target is the only hidden object in the scene - if not, then all other hidden objects are smaller than the target in size. Our planning algorithm uses rearrangement of objects as opposed to permanent removal and employs two different types of maneuvers - pick and place, and push - depending on the object to be manipulated. We do not impose any constraints on the number or size of hidden objects. We do not require object shapes to be known either.

III. Interactive Environment Exploration

The goal of interactive environment exploration in clutter is to explore a small, bounded, cluttered environment (e.g., a shelf of a refrigerator or a book shelf), by rearranging the objects till the whole environment has been explored. We assume objects cannot be removed permanently from the scene, such as by placing them on another table. Here, we are not interested in identifying the individual objects in the search area, but only in knowing the state of all areas of the volume as occupied or free. It is important to note that the total number of objects in the environment is not known a priori. The exploration algorithm could also be applied to object search or object counting. Since the scene is only partially visible due to object occlusions, an observe-plan-act strategy is called for, as new objects could be discovered that the current plan does not yet account for.

To begin understanding the nature of this problem, we study a simplified setup with the following assumptions:

1) The size of the world is (approximately) known.
2) The environment is a grid-world with at most one object in each cell. Additionally, each object occupies only one cell.
3) An object is visible to the camera only if there is no object in front of it in the grid. Therefore, no object is partly occluded by another object.
4) Objects are not in contact with each other.
5) The robot has a fixed point of view.

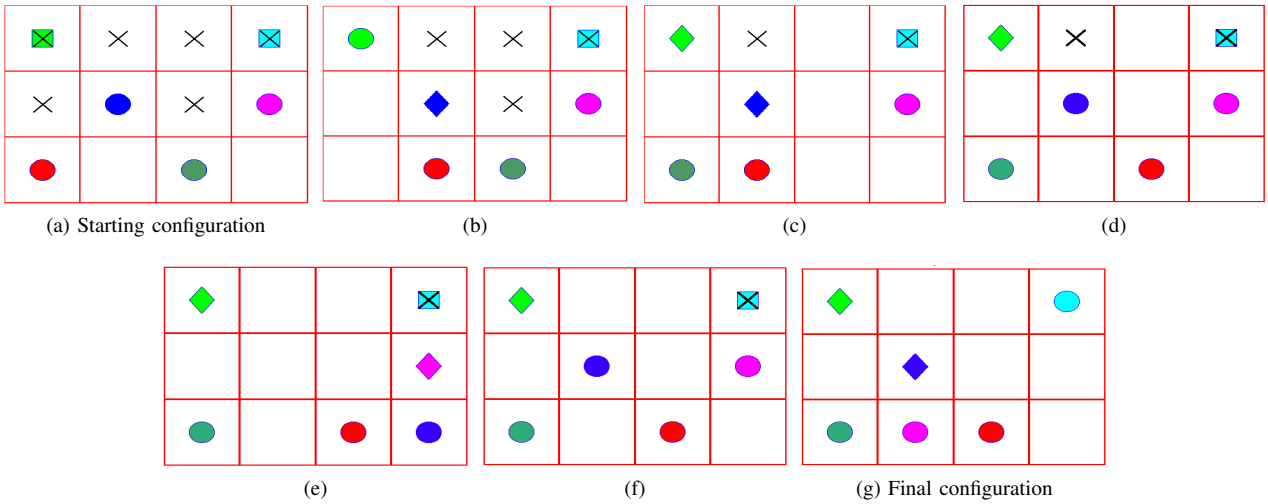
We will remove some of these assumptions in Section V where an adaptation to problems in a real, continuous world is discussed.

We propose an adaptive lookahead exploration algorithm that guarantees complete exploration of the environment unless there are no more actions possible. We presented a variant of this algorithm in a recent workshop [19]. Here, we modify it to make it more applicable to a real world scenario and then extend it to continuous world (Section V). We also give it a fuller treatment in terms of evaluation and implementation.

To start with, our algorithm assigns a state of free, occupied, or unknown to every grid cell. The goal is to know the state of every cell and only the objects that are visible at a particular instant can be moved by the robot. Allowed actions are to move a visible object from its current cell to any cell known to be free as long as the free cell is not occluded by an object in front. It takes a starting horizon (or lookahead) value, H_0, as input and in every iteration, chooses the sequence of actions of length H_0 that would reveal most of the unknown cells amongst all action sequences. We refer to the number of unknown cells whose state is revealed by an action sequence as the *information gain* of that action sequence. If all H_0 length plans result in no information gain, the horizon is incremented by 1 until a non-zero information gain is obtained. The algorithm then plans with the longer horizon, executes the plan, and falls back to H_0 for the next iteration. This procedure is repeated until all cell states are known (free or occupied).

The details of the algorithm are given in Algorithm 1. The inputs to the algorithm are the number of rows (N_r) and the number of columns (N_c) in the discretized world grid, and the starting horizon value (H_0). The algorithm has the property that it chooses a longer planning horizon, which requires more planning time, only when it is absolutely necessary and hence, it is adaptive. It guarantees complete exploration of the environment unless no valid moves are left.

The Adaptive Horizon Exploration algorithm tries to maximize the information gain of the search with the given planning horizon. The motivation behind this is to minimize the number of actions required to completely explore the environment by exploring as much environment as possible in each iteration. However, it should be pointed out that the environment is partially observable and the total number of objects unknown. Planning is done with partial information and thus, may end up being non-optimal if hidden objects are revealed during the exploration. Choosing a longer horizon does not necessarily reduce the number of actions required for complete exploration for the same reason. Another important point is that if we choose a longer planning horizon, those actions will be chosen that result in object placements



(a) Starting configuration (b) (c) (d)

(e) (f) (g) Final configuration

Fig. 1: Simulation of the adaptive horizon exploration on a 3×4 grid with 6 objects and a horizon of 2. The camera is placed along the lower boundary of the grid. Circles represent objects that are visible, squares represent objects that have never been seen so far and are thus, unknown, diamonds represent objects that have been seen at least once so far and so, their locations are exactly known, and crosses represent grid cells whose state is unknown. The unmarked cells with no objects are known to be free. In the final configuration, all cell states are known, as shown by the absence of cross marks. This exploration needed a total of 6 actions and 1.4 seconds.

Algorithm 1 Adaptive Horizon Exploration(N_r, N_c, H_0)

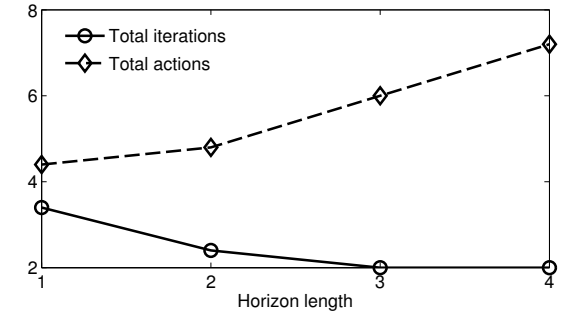
while true **do**
  V = Find-Visible-Objects
  Occ = Update-Occupied-cells
  $Free$ = Update-free-cells
  $N(unknown) = N_r * N_c - (Occ + Free)$
  if $N(unknown) = 0$ **then**
    Environment explored. Exit.
  end if
  $H_{curr} = H_0$ $\{H_{curr}$ - current horizon$\}$
  $[IG, A] = plan(V, Occ, Free, H_{curr})$
  $\{IG$ - information gain, A - action sequence$\}$
  if $IG = 0$ **then**
    $H_{curr} = H_{curr} + 1$
    Re-plan. Go to step 10.
  end if
  Execute all actions in A.
end while

that do not block other objects that are visible (and hence, movable) at the time of planning.

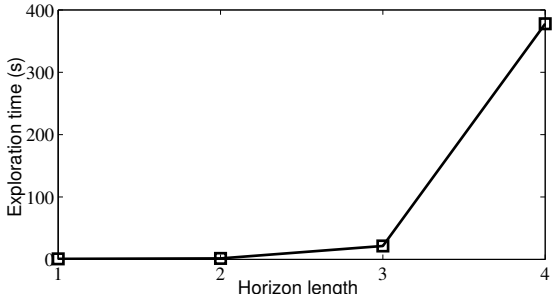
IV. Simulation Results

We carried out simulations of Algorithm 1 in MATLAB. Fig. 1 depicts the steps of the algorithm for a 3×4 grid with 6 objects and an initial horizon of 2. This environment was fully explored in 1.4 seconds using 6 actions. This time, of course, does not include any time required for the actual manipulation of objects.

Fig. 2 shows the effect of starting with different horizon values (H_0) on the number of actions required and time taken (averaged over 5 different placements of objects) to explore the whole grid. The test environment was a 3×4 grid with 5 objects. We observe that though the number of planning iterations goes down as horizon increases, each iteration takes longer and in fact, the number of actions required increases monotonically. Note that the horizon is adaptive



(a) Number of iterations and actions



(b) Time taken

Fig. 2: Comparison of the performance of the adaptive horizon exploration with varying horizons on a 3×4 grid with 5 objects.

and so, it may not stay at H_0 all the time during exploration. It is worthwhile to reiterate here that since planning is done with partial information about the environment, the plans may end up being non-optimal. Therefore, a longer horizon may not necessarily result in fewer actions.

Many other simulations on environments of varying sizes and degree of clutter suggest that a starting horizon of 1 or 2 is often the best choice. Keeping the planning speed in mind, it may be a good idea to start with $H_0 = 1$ and the algorithm will switch to a horizon of 2 as and when needed.

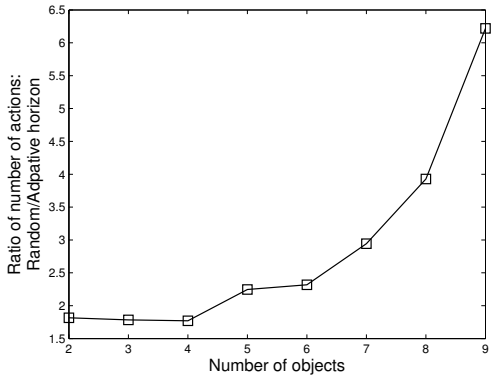
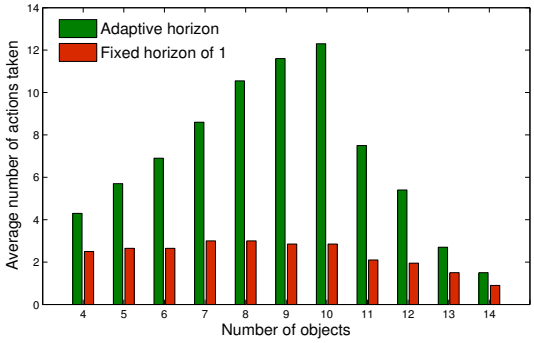


Fig. 3: Comparison of the performance of the adaptive horizon exploration with random planning. X-axis shows the number of objects in a 3×4 grid-world and the Y-axis shows the ratio of the number of actions required for complete exploration for the random planning algorithm to the adaptive look-ahead exploration.

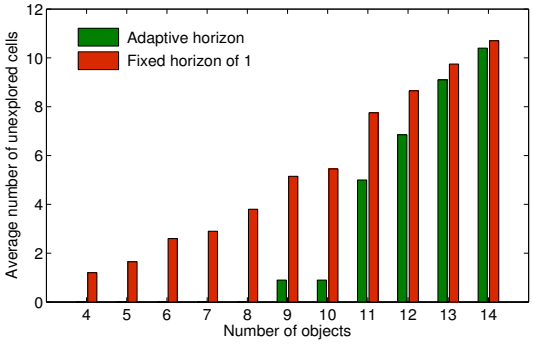
We also compare our approach to a random planning approach where a sequence of actions is chosen randomly from the available set of actions at any time. Results comparing the performance of the two approaches on a 3×4 grid-world and a horizon of 2 with increasing clutter are shown in Fig. 3. We plot the ratio of the average (over 50 different placements of objects) number of actions taken for random planning to those for adaptive horizon planning. We see that random planning needs nearly 2 to 6 times more actions than our approach before the whole grid is explored. Since manipulation dominates the execution time on an actual robot, exploration using random planning would clearly be much slower, particularly as the degree of clutter increases.

Fig. 4 compares our algorithm to an algorithm with a fixed horizon of 1 on a 4×4 grid with varying number of objects. Fig. 4a plots the number of actions required (averaged over 20 different placements of objects) by the two algorithms to explore *as much of the environment as possible* (i.e, before no more information gain is possible or no moves are left) as the number of objects in the environment increases while Fig. 4b plots the average number of unexplored cells with increasing degree of clutter. Fig. 4a seems to indicate that the fixed horizon algorithm uses fewer actions than our approach to explore the environment but we see from Fig. 4b that it consistently fails to *completely* explore the environment. This happens because fixed horizon plans soon result in no more information gain, thus ending the exploration prematurely. As clutter increases, more and more of the environment remains unexplored by the fixed horizon algorithm. For very cluttered scenes with most of the cells in the grid occupied, the adaptive horizon algorithm also fails to complete exploration because no valid moves are possible.

Fig. 5 shows the number of times (averaged over 20 configurations) a horizon value is used by the adaptive horizon algorithm as the degree of clutter in a 4×4 environment increases. We see that higher planning horizons are needed more and more as clutter increases while the usage of lower horizons goes up as well. This shows that adpating



(a) Average number of actions



(b) Average number of unexplored cells

Fig. 4: Comparison of the performance of the adaptive horizon exploration with fixed horizon exploration (horizon = 1) on a 4×4 grid with varying number of objects. All numbers are averaged over 20 different object configurations.

the horizon is essential to achieve complete exploration, particularly in highly cluttered scenes.

The simulation results on a simple grid world presented in this section helped us evaluate the nature of the environment exploration problem and analyze various aspects of our adaptive horizon exploration algorithm. These results indicate that our algorithm performs better than random and greedy approaches on average in terms of number of actions. They also suggest that starting with an initial horizon of 1 and adapting it as and when needed may be better than choosing a longer horizon to begin with. We present the application of our proposed algorithm to object search in a real world cluttered environment in the next section.

V. Interactive Object Search

In robotics, object search has typically been confined to active visual search which refers to moving the camera so as to locate the target which is, otherwise, lying out in the open, not occluded by other objects. Here we show how robotic manipulation can be used to interact with the world and locate the target.

In practice, the assumptions of a grid world and absence of partial occlusions made in Section III are unrealistic. Moreover, it may be possible to manipulate an object in different ways depending on its size and surrounding environment. While an object that is not surrounded closely by other objects may be picked up and placed at a new location, it

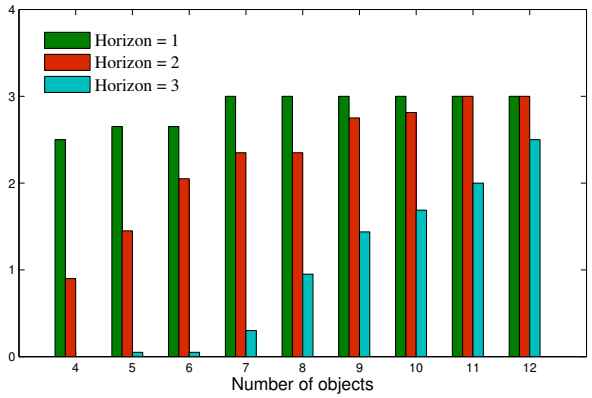


Fig. 5: Figure showing the average number of times a particular value of horizon results in information gain (and hence, a plan) with respect to degree of clutter for Algorithm 1.

may only be possible to push and slide objects around in heavy clutter. Some objects may simply be too large for the robot to grasp. We, therefore, relax the assumptions of a grid world and no partial occlusions, and introduce two kinds of manipulation (pick and place, and push) for implementation on a real robot.

Fig. 6: Experimental setup. The PR2 with a head-mounted Kinect views the objects on a shelf from the front. The task is to search for a target object by rearranging other objects on the shelf.

We implemented our planner on the PR2 robot [20] to test its feasibility in the real world. PR2 is a semi-humanoid robotic platform developed by Willow Garage. Fig. 6 shows our experimental setup. We mounted a Microsoft Kinect sensor on the robot head. All perception data in this paper are from this sensor. Several objects of different shapes and sizes are placed on a high shelf in front of the PR2 such that the robot can see only some of the objects. The shelf is high enough to deny the robot a top view of the objects but not too high to hinder manipulation. The next section gives the details of the algorithm as applied to a real world environment.

A. Implementation

The planner was tested on the PR2 by building a ROS package that implements the pipeline depicted in Fig. 7. All steps of the implementation are detailed below.

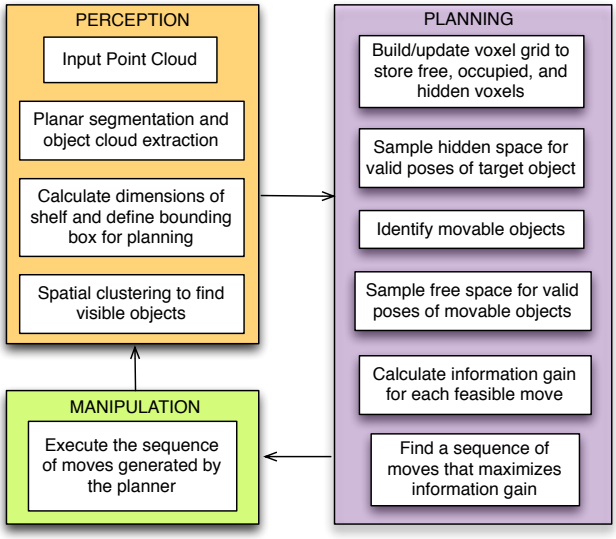


Fig. 7: An overview of the implementation pipeline on the PR2 robot searching for an object in a real world scenario.

Perception:

a) Read in a point cloud from the Kinect.
b) Use thresholding to retain only that region from the point cloud that contains the shelf (Fig. 8a, 8b) and then separate the planar surface from the objects (Fig. 8d) using planar segmentation. The approximate size and location of the shelf need to be known for this step. Reduce the size of object cloud using downsampling. These algorithms are available in the Point Cloud Library (PCL) [21].
c) Calculate the dimensions of the planar surface. This will constitute our *planning bounding box* as it defines the area in which the objects may be located (Fig. 8d).
d) Extract individual object clouds using spatial clustering of the objects point cloud. Note that not all objects may be fully visible and for most objects, only the front surface is visible to the camera. Therefore, these point clouds are incomplete and may not represent the objects correctly. To take this into account, bounding boxes are defined for each cluster. Since object depth is often not perceived correctly in this setup, the bounding box is given some additional depth if the depth of an object cluster is below a threshold (Fig. 8e). Also, if the point cloud does not touch the table, it corresponds to an occluded object and its bounding box is extended to the table. A number of points are then generated to fill this *object bounding box* and this new point cloud is then used for planning instead of the actual object clusters.

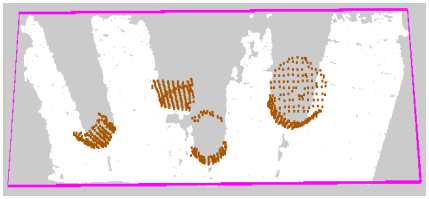
Planning:

a) Build a voxel grid using an octree [22] from the object cloud to represent free, occupied, and unknown voxels in the planning scene bounding box. In Fig. 8f, the voxels marked as red encode the occupied and hidden

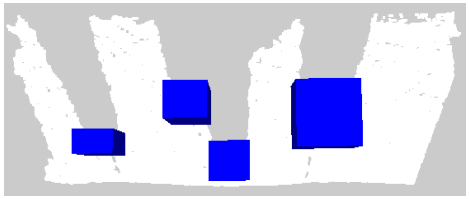
(a) Point cloud of the shelf obtained from the head-mounted Kinect on the PR2

(b) Top view of the point cloud seen by the robot shows the occluded areas of the shelf.

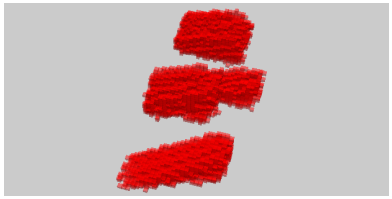
(c) The actual scene has the target object, a small salt shaker, hidden behind the yellow can.



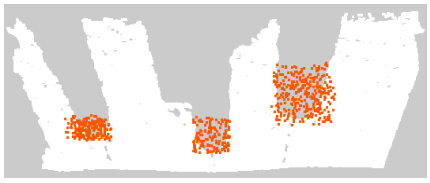
(d) The largest plane detected i.e. the shelf is shown in white while the brown points belong to objects on the shelf. The pink lines mark the area in which the objects could be present and thus, define the bounding box for planning.



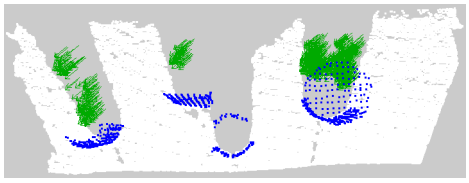
(e) Bounding boxes of visible objects. If the depth of an object point cloud is below a threshold, a fixed depth is added to its bounding box for realistic planning of moves and collision checks.



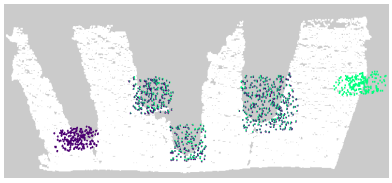
(f) Side view of the octree - representation of the space that is either occupied or hidden and thus, could be hiding the target object.



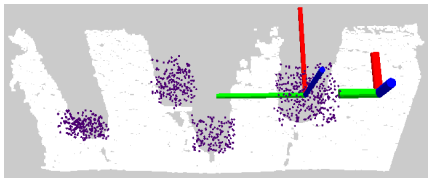
(g) Movable Objects - these objects are not occluded by others.



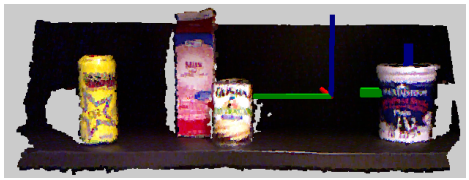
(h) The green arrows depict the valid target object poses sampled in the hidden space.



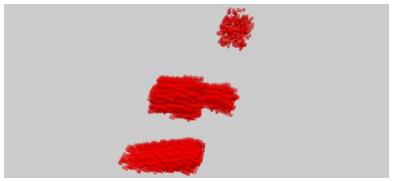
(i) An example valid move. Purple: old positions, green: new positions



(j) Move output by the planner - the axes show the source and the destination poses of the object to be moved.



(k) Point cloud obtained from Kinect after executing the plan



(l) Side view of the occupied and hidden voxels in the octree after executing the plan. Note that the information about the voxels behind the moved object being free is retained.

Fig. 8: RViz snapshots of the algorithm running on a scene of 5 objects - 4 visible and 1 hidden. The planning horizon is 1 to begin with in this example and the dimensions of the target object (a salt shaker) were 0.04m×0.04m×0.1 m.

space. The rest of the voxels in the planning bounding box are encoded as free (not shown in the figure).

b) Among the detected clusters, clusters corresponding to *movable* objects are identified. These are objects that are not partially occluded by other objects and offer a clear path for the gripper for a front grasp (Fig. 8g). For this paper, we use only front grasps for all objects.

c) Sample the hidden space for valid poses of the target object (Fig. 8h). The size of the target object is assumed to be known and it is modeled as a point cloud. A valid pose is one for which none of the points in the target point cloud are in the free space (and hence, not visible to the camera), the object touches the support surface, and is upright. Rotation about an axis perpendicular to the support surface is allowed.

d) For each *movable* object, sample the free space for locations where the object may be relocated. If the object can be grasped, its new location could be anywhere in the free space. If the object is too big to be grasped, its new location could only be in the neighboring free space where it can be pushed to. These locations constitute the set of *possible moves* (Fig. 8i). Consecutive moves of the same object in the same planning iteration are not allowed.

e) Simulate each possible move and calculate the percentage of sampled target poses that are revealed by it. We will refer to this percentage as the *information gain* of the move.

f) Use Algorithm 1 to find a sequence of actions that maximizes information gain (Fig. 8j) and send it to

the manipulation node for plan execution.

Manipulation:

a) Each move contains information regarding the point cloud to be manipulated, its origin and destination pose, and the kind of action to be taken (pick and place, or push). Depending on the kind of action and the direction of displacement, waypoints are generated for the robot hand to move to, thus displacing the object in the process. Whether the left arm is used or the right is decided based on the destination of the object. If the destination lies in the left half of the planning bounding box, the left arm is used; otherwise the right arm is used.

A new point cloud is obtained from the Kinect after plan execution (Fig. 8k) and the process is repeated. The target object is currently identified by its size or color (assuming it is the only object in the scene of that size or color) to simplify object recognition. The search is considered complete when the target object becomes visible to the camera. If the target object cannot be found, the search stops when there are no hidden areas left where the target could be located. Since execution of an action may occlude space earlier observed to have been free or occupied, the octree is updated after successful execution of moves to retain information of occupied and free voxels across planning iterations (Fig. 8l).

The accompanying video shows the PR2 searching for a small salt shaker in a kitchen shelf containing 3 other objects of varying size. The starting horizon was 1 in this example. Note how the first move is a push because the object is too big to grasp. The second object is however, picked up and placed at a new location, resulting in the salt shaker being revealed.

VI. Complexity Analysis

We now analyze the running time of our algorithm. Let us introduce some notation first: let N be the total number of objects; p the maximum number of points in the downsampled point cloud of an object; k_v the sample size of movable objects; k_t the sample size of the target object; and H the horizon length. The worst case complexity of various steps in one iteration of the planner is as follows:

a) Sample target pose: $a = \mathcal{O}(pk_t)$
b) Find movable objects: $b = \mathcal{O}(pN) + \mathcal{O}(N^2)$
c) Find possible moves: $c = \mathcal{O}(pk_vN)$
d) For each possible move, simulating the move and checking if it results in a collision is $d = \mathcal{O}(p^2N)$. Then, the planner is recursively called with a smaller horizon.

Therefore, if m is the maximum number of possible moves in a recursion, the complexity $\mathcal{W}$ with horizon H is:

$$\begin{aligned}\mathcal{W} &= (a+b+c+md)\sum_{i=0}^{H-1} m^i + m^H d \\ &= (a+b+c+md)\frac{m^H-1}{m-1} + m^H d \\ &= \mathcal{O}(p^2N^{H+2}k_v^{H+1} + pN^{H-1}k_v^{H-1}k_t)\end{aligned}$$

Thus, we see that the complexity scales rapidly with increasing horizon, as expected. However, downsampling the object point clouds will speed up the planner. In our experiments, we found that as few as 100 points were enough to correctly represent the boundary and volume of a typical kitchen object. Also, dense sampling of the free space for new object locations does not help much because many samples are then close together, resulting in similar information gains. So, we can afford to choose a small k_v.

We carried out experiments with a sample size of 20,000 for the target object pose (k_t) and 50 for poses of movable objects (k_v). With p, k_v, k_t as constants, the complexity of each iteration becomes $\mathcal{O}(N^{H+2})$. Thus, if complete exploration requires i iterations, the overall complexity is $\mathcal{O}(iN^{H+2})$. Since each planning iteration results in non-zero information gain (i.e, at least one target sample is revealed), there can be a maximum of k_t iterations. In practice, each action will reveal several target samples and thus, i would be much smaller than k_t. For a scene containing 4-6 objects, the average planning time for each iteration on the PR2 robot is 16 seconds with a planning horizon of 1 and 60 seconds with a horizon of 2.

VII. Discussion

Our solution is effective because meaningful simplifications were made that offer a good compromise between computational effort and solving power for the presented domain. However, we consider only a small subset of possible environment interactions (picking from the front, pushing to the side), and other kinds of maneuvers might be necessary to solve a given problem. Incorrect initial segmentation is partially accounted for in the current algorithm, as the segmentation gets updated when changes occur due to manipulation. There are still paradox situations though. For example, if there are two objects close to each other and the only way to move one of them is to push it into the other, the current algorithm will not allow this to happen as we explicitly forbid object-object collisions. If they were incorrectly segmented as one cluster, the robot instead happily pushes both together. We are looking into modeling the physical interactions between objects and allowing pushing one object into another, which is a powerful manipulation strategy in many real world situations. Uncertainty in a pushing action's outcome will then come into play and will have to be accounted for.

VIII. Conclusion and Future Work

We presented an algorithm for environment exploration in small cluttered environments, e.g., a kitchen shelf, using manipulation of objects. Simulation results show that as clutter increases, significantly fewer number of moves are required to completely explore the environment using our algorithm as opposed to rearranging objects randomly. This is important since manipulation by a real robot is usually very slow and thus, we want to minimize the number of moves. Our algorithm also guarantees *complete* exploration as opposed to a greedy exploration as long as there are

valid moves left. We then presented an implementation of the algorithm on the PR2 robot and applied it to object search. The robot has a fixed and partial view of the scene but has access to two different manipulation primitives. We presented preliminary experimental results with a pipeline that manipulates a cluttered scene in ways that help the robot to locate the target object finally. Our algorithm interleaves adaptive look-ahead planning with object manipulation and has direct applications in real world problems like object search, object counting, and scene mapping.

In discrete domains, planning can often be optimized by pruning as identical states are expanded multiple times. Additionally, the state space is constrained due to the discrete amount of possible manipulation primitives and object positions, while the problem of high dimensionality prevails. The fact that in continuous space, there are infinitely many manipulation primitives as well as infinitely many states constitutes a distinct challenge that we can only meet by dropping the claim of completeness.

In order to provide an algorithm that would be probabilistically complete with less simplifications to the domain, we are looking into sampling from the large unconstrained set of possible push and pick & place primitives directly. This could be done in an anytime manner such as RRT*. The tree could initially be filled using a small set of proven manipulation primitives before allowing the algorithm to explore the vast space of possible solutions when more time is available and a solution is not yet found. This approach will require to consider grasp and push planning as well as the modeling of arbitrary physical interactions. In turn, it would allow the most complex problems to be solved.

Going forward, we would like to add the following functionalities to our pipeline:

- Incorporate feasibility check for a move within planning. It is not enough to find a free location for an object to be moved to, it is also essential to check whether that move can be executed by the robot arm given its reachability constraints.
- Relax the use of only front grasps and use grasp planning to include all feasible grasps.
- Pushing may be done by an active control regime including vision-based tracking of the pushed object.
- Introduce more manipulation primitives. For example, one useful primitive may be where the robot picks up an object with one arm temporarily, takes an observation, plans, executes the plan with the other arm, and replaces the object in hand in the shelf.
- It is unnatural for the robot to have a fixed point of view during the search. It would be interesting to explore how combining manipulation with multiple camera views affects the results.

IX. Acknowledgment

This work was supported in part by the US National Science Foundation Robust Intelligence Program (grant IIS-1017134), and by the Defense Advanced Projects Research Agency under the Autonomous Robot Manipulation program (contract W91CRBG-10-C-0136).

References

[1] 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 *Proc. of the European Conference on Mobile Robotics*, 2011.

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

[3] J. Ma, T. Chung, and J. Burdick, "A probabilistic framework for object search with 6-DOF pose estimation," *The Intl. Journal of Robotics Research*, vol. 30, no. 10, pp. 1209–1228, 2011.

[4] T. Parsons, "Pursuit-evasion in a graph," *Theory and applications of graphs*, pp. 426–441, 1978.

[5] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM Journal on computing*, vol. 21, no. 5, pp. 863–888, 1992.

[6] B. Gerkey, S. Thrun, and G. Gordon, "Visibility-based pursuit-evasion with limited field of view," *The Intl. Journal of Robotics Research*, vol. 25, no. 4, pp. 299–315, 2006.

[7] V. Kumar, D. Rus, and S. Singh, "Robot and sensor networks for first responders," *Pervasive Computing*, vol. 3, no. 4, pp. 24–33, 2004.

[8] D. Calisi, A. Farinelli, L. Iocchi, and D. Nardi, "Multi-objective exploration and search for autonomous rescue robots," *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 763–777, 2007.

[9] S. Koenig and M. Likhachev, "D^* lite," in *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002, pp. 476–483.

[10] C. Tovey, S. Greenberg, and S. Koenig, "Improved analysis of d*," in *IEEE International Conference on Robotics and Automation, 2003*, vol. 3. IEEE, 2003, pp. 3371–3378.

[11] P. Fitzpatrick, "First contact: an active vision approach to segmentation," in *Proc. of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, vol. 3, 2003, pp. 2161–2166.

[12] L. Chang, J. Smith, and D. Fox, "Interactive singulation of objects from a pile," in *Proc. of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, 2011.

[13] D. Schiebener, A. Ude, J. Morimoto, T. Asfour, and R. Dillmann, "Segmentation and learning of unknown objects through physical interaction," in *11th IEEE-RAS Intl. Conference on Humanoid Robots (Humanoids)*, 2011, pp. 500–506.

[14] D. Katz and O. Brock, "Manipulating articulated objects with interactive perception," in *Proc. of the IEEE Intl. Conference on Robotics and Automation*, 2008, pp. 272–277.

[15] M. Gupta and G. S. Sukhatme, "Using Manipulation Primitives for Brick Sorting in Clutter," in *Proc. of the IEEE Intl. Conference on Robotics and Automation*, 2012.

[16] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," in *Robotics: Science and Systems*, 2011.

[17] L.L.S. Wong, L.P. Kaelbling, and T. Lozano-Pérez, "Manipulation-based Active Search for Occluded Objects," in *Proc. of the IEEE Intl. Conference on Robotics and Automation*, 2013.

[18] M.R. Dogar, M.C. Koval, A. Tallavajhula, and S.S. Srinivasa, "Object Search by Manipulation," in *Proc. of the IEEE Intl. Conference on Robotics and Automation*, 2013.

[19] M. Gupta and G.S. Sukhatme, "Interactive Perception in Clutter," in *Workshop on Interactive Perception, Robotics Science and Systems*, 2012.

[20] W. Garage, "Overview of the PR2 robot," 2009.

[21] R. Rusu, S. Cousins, and W. Garage, "3D is here: Point Cloud Library (PCL)," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China*, 2011.

[22] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com. [Online]. Available: http://octomap.github.com