Robot Task Planning Under Local Observability

Max Merlin, Shane Parr, Neev Parikh, Sergio Orozco, Vedant Gupta, Eric Rosen, and George Konidaris

Abstract— **Real-world robot task planning is intractable in part due to partial observability. A common approach to reducing complexity is introducing additional structure into the decision process, such as mixed-observability, factored states, or temporally-extended actions. We propose the *locally observable Markov decision process*, a novel formulation that models task-level planning where uncertainty pertains to object-level attributes and where a robot has subroutines for seeking and accurately observing objects. This models sensors that are range-limited and line-of-sight—objects occluded or outside sensor range are unobserved, but the attributes of objects that fall within sensor view can be resolved via repeated observation. Our model results in a three-stage planning process: first, the robot plans using only observed objects; if that fails, it generates a target object that, if observed, could result in a feasible plan; finally, it attempts to locate and observe the target, replanning after each newly observed object. By combining LOMDPs with off-the-shelf Markov planners, we outperform state-of-the-art-solvers for both object-oriented POMDP and MDP analogues with the same task specification. We then apply the formulation to successfully solve a task on a mobile robot.**

I. INTRODUCTION

Robot task planning is characterized by pervasive partial observability, sparse rewards, and long horizons, but planning in the presence of generic environmental uncertainty is NP-Hard [1], leading to scalability and performance problems in practice [2]. More effective planning is possible using structured models of partial observability [3, 4, 5], at the cost of additional assumptions about its form. Fortunately, those assumptions need not necessarily hold in the task directly; instead, the robot control system can be structured to include techniques (e.g., vision, SLAM [2], or inverse kinematics) that result in a *task-level* planning problem where they are reasonable. For example, the object-oriented POMDP [5] models object-centric attributes; objects are not directly present in pixel observations, but computer vision can process the scene into a labeled set of objects, giving a task-level model where the object-centric assumption holds.

We propose to abstract out the subproblem of sensing uncertainty from a POMDP by introducing the *locally observable Markov decision process* (LOMDP), a decision process that models object-level partial observability stemming from line-of-sight, range-limited sensing. We make three core modeling assumptions. First, task-level uncertainty pertains to object-level attributes (including existence and location); just as uncertainty for navigation is primarily to do with *space* (and can therefore be resolved by SLAM [2]), so uncertainty for manipulation is primarily to do with *objects*. Second, that the robot has access to two sensing subroutines: *observe* and *find*. *Observe* eliminates the uncertainty of an object within sensor view, in practice using a closed-loop subroutine that repeatedly generates views to reduce attribute uncertainty below a threshold (after which we consider it fully observed). *Find* searches for a given target object and executes *observe* when a novel object is brought within sensor range; the range- and line-of-sight assumptions guarantee that it identifies at least one previously unobserved object each time it is executed. Finally, we assume that interacting with an object requires it to be within sensor range before and after interaction; therefore, in a (static, single-robot) LOMDP, an observed object is never subject to state uncertainty again. Solving a task under these assumptions requires the robot to observe every object used in its solution plan, leading to a simple planning process: the robot uses a Markov planner to solve the task using known objects; if that fails, it generates a target object that, if observed, could lead to a successful plan and attempts to locate and observe that object; the process repeats when it observes a novel object.

We propose that these assumptions often hold at the task-level, and that separating observation from planning, and exploiting efficient Markov planners, can scale to realistic tasks. To demonstrate this, we compare the LOMDP framework to MDP and POMDP formulations in a challenging, combinatorially complex sandwich-making domain. As the number of extraneous objects in the scene increases, LOMDPs support efficient solutions while the alternatives are slow or fail entirely. Finally, we apply LOMDPs to realize efficient planning in a robot coffee-making task.

II. BACKGROUND AND RELATED WORK

A *partially observable Markov decision process* (or POMDP) is a sequential decision-making task where an agent must choose an action to execute at every timestep, but where it cannot always observe the information necessary to make that decision. Formally, a POMDP is defined as a tuple $(S, A, \Omega, O, T, R, \gamma)$, where S is a set of states, which the agent cannot observe directly; A is a set of actions, one of which it must choose to execute at each timestep; Ω is an observation space; $O(\omega|s)$ is an observation function giving the probability of observing $\omega \in \Omega$ when in state s; $T(s'|s,a)$ is the transition function, describing the probability

Department of Computer Science, Brown University, Providence, RI 02912, USA {max_merlin, shane_parr, sergio_orozco, vedant_gupta, eric_rosen}@brown.edu, neev.v.parikh@gmail.com, and gdk@brown.edu. This work was supported by NSF CAREER #1844960 to Konidaris, NSF grant number #1955361, ONR grant number N00014-21-1-2200, ONR PERISCOPE MURI N00014-17-1-2699, and NSF Fellowship #2022346540 to Parr. Partial funding for this work was provided by The Boston Dynamics AI Institute ("The AI Institute") and Echo Labs. Disclosure: George Konidaris is the Chief Scientific Advisor of Realtime Robotics, which develops motion planning software.

that action a in state s causes a transition to state s'; $R(s,a,s')$ is the reward received for transitioning from state s to s' by executing a; and γ is a discount factor. The agent must choose actions that maximize its expected discounted sum of rewards, without observing the state s directly.

There are several approaches to adding structure to Markov decision processes. In factored MDPs [3], the state is composed of a vector of state variables (or factors), which transition infrequently and independently of most other factors. Factored MDPs support efficient planning algorithms that exploit this structure [6]. Object-oriented MDPs [7] introduce even more structure by factoring state into a collection of objects, each with their own state variables and type; there is now a transition function for each type. Object-oriented POMDPs [5] generalize this to the partially-observable setting; LOMDPs extend OO-POMDPs.

We also draw inspiration from the mixed observability MDP (MOMDP) [4], where the state is partitioned into factors (state variables) that are fully observable and factors that are partially observable. While MOMDPs capture some of the modeling aspects we propose, each factor is always either fully or partially observable and cannot change from one to the other, whereas in LOMDPs objects transition from unobserved to fully observed as they are sensed. By contrast, [8] provides a structure to allow state variables to transition from unknown to known. However, LOMDPs provides a more extensible and accessible model rooted in POMDPs.

III. Locally Observable MDPs

The POMDP formulation can model general hidden state variables and complex observation functions. That generality comes with a severe complexity penalty; a promising approach to efficient planning is to incorporate structural assumptions specific to real-world robots. Consider a home service robot tasked with making a cup of tea, which requires fetching a cup, using a kettle to boil water, and obtaining and combining specific ingredients (a teabag, sugar, and milk). Here there are several uncertain state variables: the exact location of the cup and the ingredients, whether the kettle has sufficient water, whether that water has been boiled, and whether the kettle is plugged in. Assuming that the robot has mapped the kitchen using SLAM [2]—thereby resolving uncertainty to do with the map—the remaining uncertainty is all over object attributes. We propose that this object-level uncertainty is primarily due to the robot's sensor limitations, and propose to model this using the *Locally Observable Markov Decision Process* (or LOMDP), a subclass of OO-POMDPs in which four assumptions hold:

1) Sensors are range limited: they return useful readings when an object is within range, but are essentially uninformative beyond that range. This condition is necessary but insufficient for observability.
2) Sensors are line-of-sight: object attributes can only be observed if unoccluded; occlusions may occur due to the scene (e.g., a wall), another object, or even the object itself (e.g., a microwave door being closed occludes the variable describing its contents). This condition is also necessary; satisfying both it and the range condition is sufficient for useful sensor readings.
3) The robot is equipped with two subroutines: `observe`, which adjusts the robots sensor(s) around the region local to the robot, terminating once the local objects are fully observed, and `find`, which searches the entire domain for an object, running `observe` in new locations until the desired object is found.
4) Manipulation requires observation, which holds throughout the manipulation; also, objects are static and no other agents modify the world. Hence, once an object attribute has been observed, it remains so.

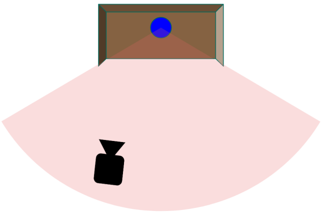

Fig. 1. The locality function $L(i,v)$. If the blue object (i) is at a particular position inside of a cupboard (v) the `observe` subroutine succeeds when the robot's sensor is in the red shaded region.

Taken together, these four modeling assumptions capture the structure present in task-level robot planning. We formalize these assumptions into a structured OO-POMDP that supports efficient planning. A LOMDP is a tuple:

$$(S, A, O, \Omega, L, R, T, \gamma),$$

where S is the set of states, A is a set of actions, O is an observation function, Ω is an observation space, L is a *locality function*, R is a reward function, T is a transition function, and γ is the discount factor. Most elements are the same as the standard OO-POMDP, but one element—the locality function L—is novel to LOMDPs.

As in an OO-POMDP, the state space S is factored into the robot state S_r and the state of n objects: $S = S_r \times S_{o_1} \times ... \times S_{o_n}$. Each object has its own state vector $s_{o_i} \in S_{o_i}$, including its pose; because pose plays a special role in LOMDPs, we write the ith object's pose as $s_{o_i}^{pose}$, and the robot's as s_r^{pose}.

Modeling the output of the *observe* subroutine, at each timestep, the agent receives an observation where each object is either accurately observed or not observed at all. The observation space is therefore $\Omega = S_r \times \{S_{o_1} \cup \phi\} \times ... \times \{S_{o_n} \cup \phi\}$, where robot state is always observed, and each object's state is either accurately observed ($\omega[i] = s_{o_i}$, i.e., the correct state of object i) or has a null observation (ϕ).

We next turn to the observation function, which in a POMDP is of the form $O(\omega \mid s', a)$, returning a probability distribution over observations given a state and action. In LOMDPs observations are deterministic, so it is unnecessary to return a probability distribution. Instead, the observation function deterministically maps a given state and action to an observation: $O : (s', a) \rightarrow \omega \in \Omega$.

In addition to these standard POMDP elements, LOMDPs have a new element, the *locality function*, which takes in a object state and outputs the set of states from which that object can be perfectly observed. The locality function can be derived from the observation function: if the robot is in the locality of a given object i, the observation it receives for that object is (considered) its true state v. If the robot is outside the locality of i, it observes ϕ (no observation). We therefore define the locality $L(i,v)$ of object i with state v:

$$L(i,v) = \{s \in S \mid O(i) = s_{o_i} \wedge (s_{o_i} = v)\}.$$

$L(i,v)$ is the set of states from which object i can be observed if it has state v. The *global locality* $L(i)$ for object i, the set of all states from which i is observable, is:

$$L(i) = \{s \in S \mid O(i) = s_{o_i}\} = \bigcup_{v \in S_{o_i}} L(i,v).$$

The global locality function allows us to redefine the observation function in a more structured way:

$$O(i) = \begin{cases} s_{o_i} & \text{if } s \in L(i) \\ \phi & \text{otherwise.} \end{cases}$$

Because the observation function can be written as a function of the locality function and vice versa, only one need be defined for a given domain. Taken together, the above definitions formalize assumption 3.

Additionally, the locality function must model the fact that the robot's sensors are range-limited—an object can only be sensed when it is within sensor range. This results in the following constraint on the locality function:

$$s \in L(i) \implies \text{inrange}(s_r^{pose}, s_{oi}^{pose}),$$

where `inrange` encodes the robot's sensor range constraints. For example, a mobile robot with a fixed-height 360° vision sensor with maximum range r has range constraint: $\text{inrange}(p_1, p_2) = \|p_1 - p_2\|_2 \leq r$, while a robot with a fan-shaped sensor requires an additional orientation constraint. Similar (though more complex) models apply for 3D sensors. This sensor-specific condition encodes assumption 1.

The range condition is necessary for observability but not sufficient. The observed object must also be line-of-sight, and can be unobservable if the static scene or another object occludes it; in the latter case the occluding object is observed instead.[1] This is modeled by the following constraint:

$$(s \notin L(i)) \wedge \text{inrange}(s_r^{pose}, s_{o_i}^{pose}) \implies$$
$$(\exists b \ s.t. \ \text{occludes}(s_r, o_b, o_i) \wedge s \in L(b)) \vee \text{occludes}(r_s, \text{scene}, o_i).$$

The `occludes` function depends on the sensor characteristics and the geometry of the two objects, but a reasonable approximation in practice is that $\text{occludes}(r,b,t)$ holds if both objects are in range (i.e., $\text{inrange}(r,b)$ and $\text{inrange}(r,t)$), and the lines between robot sensor r and the relevant points of interest on target object t intercept the body of occluding object b. We include the special-case `scene` object to cover the case where an object is occluded by a static obstacle with no state of its own; such obstacles are typically walls and similar features present in the robot's environmental map. This encodes assumption 2.

For most physical sensors, the set of poses in which a target object would be visible from a given robot pose s_r^{pose} is compact. For example, a robot with a fan-shaped sensor can observe an object located at any point inside the solid volume of space covered by the fan. Portions of these regions can be removed by an occluding object (or the scene), but since the removed regions are also compact, they merely break the visibility poses from any point into a set of compact regions. Even if a target object *is* occluded on all sides, the object can be observed as long as occluding objects can be moved to some pose where it no longer occludes the target. As a result, it is always possible for the robot to generate a finite number of states to transition through to observe any object with certainty. We instantiate this search process as a `find` subroutine; there are already many existing works on object search [9, 10, 11, 12]. Similarly, existing work can instantiate the `observe` routine [13] by repeatedly gathering observations to reach near-certainty in object recognition and pose estimation.

The available action set A is determined by the actions available to the robot; here LOMDPs impose two conditions. Actions that modify object i can only be available in $L(i)$, the global locality of o_i. This models assumption 4. (Of course, they may have other preconditions restricting where they can be executed.) Additionally, the robot has an observation action $find\_observe(i)$ that runs `find` for object i until any new object is encountered, and then `observes` it, which can be applied when s_{o_i} is unknown. This models assumption 3.

A core aim of LOMDP planning is to manipulate the state of the world, expressed as the state of its constituent objects, to reach some goal state. We model a task reward function as sum of two components: $R(s,a,s') = R(s') - C(a)$, where $R(s')$ rewards entering a terminal set of *goal states* G, and C expresses the (state independent) cost of executing a particular action. The goal, as expressed in G, will typically specify desired configurations of only some objects present in the environment, in which case the robot may reach the goal without having interacted with, or even necessarily observed, many of the objects in the environment.

Finally, we turn to the transition function T, which is defined as usual in a POMDP, except that it expresses the assumption that manipulation implies observation: any time a state variable changes, the robot must have been able to observe its value before and after the change:$s_{o_i} \neq s'_{o_i} \implies (O(i) = s_{o_i}) \wedge (O(i') = s'_{o_i})$, for all objects i and actions a where $T(s'|s,a) > 0$. This constraint encodes assumption 4.

IV. LOMDP Planning

The assumptions underlying the LOMDP model give rise to two important properties. The first is that a robot can always observe some new object using the `find_observe` subroutine, though it is possible the found object is not the target of the search. The second is that *an observed object stays observed*, and can therefore be treated as fully

[1] Note that several objects may occlude object i; our model simply states that at least one such occluding object is observed.

observable for planning purposes. These properties suggest that LOMDP solutions could be constructed from a sequence of navigation actions that observe relevant objects, followed by a (Markovian) planning process over those now completely observed objects. In fact, such a structure is *necessary* for solving a LOMDP—any solution that uses an object necessarily involves first observing it, which suggests the planner shown in Algorithm 1, which has three subroutines:

Algorithm 1: Outline of a LOMDP Planner.

```
observed ← $\{s_r\}$
unobserved ← $\{s_{o_1}, ..., s_{o_n}\}$
**repeat**
    (solved, plan, info) ← MDP_plan(observed, $R$)
    **if** *not solved* **then**
        target ← next_target($R$, info, unobserved)
        obs ← find_observe(target)
        observed ← observed ∪ obs
        unobserved ← unobserved \ obs
**until** *solved*
```

1) MDP_plan(o, *R*) attempts to solve for goal *R* using only the collection of fully observed objects *o* as input to a Markov planner. It returns a tuple of three elements: a boolean indicating whether or not a solution was found, the solution plan if found, and, if not, optional diagnostic information indicating why.
2) next_target(*R*, i, u) proposes an object to attempt to observe next, selected from the list of unobserved objects *u* using goal *R* and perhaps the optional diagnostic information *i* from MDP_plan.
3) find_observe(t) attempts to observe object *t* by navigating through its locales using find, and calling observe when it encounters any previously unobserved object (not necessarily *t*), then halting and returning the newly observed object.

The key subroutine here is find_observe(t), which expands the set of observed objects available to the MDP_plan. Any off the shelf object search/active perception algorithm [13] can be applied here. If the robot attempts to observe *t* but is occluded by a previously unobserved object, the occluding object is observed and the subroutine returns it and halts. This early termination allows the robot to potentially find a plan using an object it had not intended to observe, and means that find_observe need only ever compute locales using known objects; since all occluders are eventually observed, repeatedly calling the subroutine with the same target will eventually result in its observation.

For clarity, the formulation above assumes that the robot knows which objects are present in the environment, even though their state and locations are unobserved. That is unrealistic in many tasks, where the robot does not know which, or how many, objects are present. Fortunately we can simply replace the unobserved object list with a list of object classes; each stands in for the hypothetical object instance that the robot would find first, were it to search for an object of that class. The robot can generate the locales to search for each class without knowing that instance objects are present, or how many there are. Observed instances are simply added to the observed list as before, and a class is only deleted from the unobserved list once it is certain that no instances remain.

V. Experiments

We evaluate the potential for LOMDPs to improve planning in two deterministic domains: first, a simulated domain where a LOMDP planner could be thoroughly empirically compared to a state-of-the-art POMDP planner, and second, a robot domain which demonstrates that LOMDP planning can generate goal-directed behavior on a real robot. In both experiments, we use an off-the-shelf PDDL planner as the Markov planner because both domains are deterministic, but a more general planner could be used for stochastic domains.

A. Simulation Experiments

We empirically evaluate the properties of LOMDP planning in the peanut butter and jelly (PBJ) domain, where a robot in a kitchen must make a sandwich. Making the sandwich requires spreading peanut butter on bread, spreading jelly on bread, and then putting the two halves together. In order to perform each spread action, the jelly or peanut butter must be on the table along with the bread and the knife. However, all of these ingredients are distributed among a set of closed cupboards and are unobserved at the start of the task. To observe each object, the robot must be in front of the cupboard that the object is in and that cupboard door must be open. Additionally, obstacle objects are randomly distributed within the cupboards; the cupboards are treated as stacks, so the robot may need to take obstacles out of the cupboard in a specific order to reach an object that it needs.

To solve this task as a LOMDP using Algorithm 1, only the initially observed objects of the domain were encoded in PDDL [14]. If the task was not solvable an unseen object is selected as the next_target. The robot then navigated through unseen locales to attempt to find next_target; for each new target locale a new PDDL problem file was written and solved with that locale given as its goal. Similarly, each time new objects were observed the PDDL file was updated to reflect that the observed state had expanded. This was initially implemented by selecting the next_target and the next locale to search at random, which is clearly naive yet still showed improvements over our baselines. We later show how heuristics of which objects are relevant or priors over where objects might be located can be utilized to further improve planning. Once all the necessary objects were observed the final plan to complete the task could be found and executed. Each of these PDDL problem files were solved using fast downward [15], a well-established off-the-shelf planner. The use of PDDL is not necessary to solve a LOMDP, but the ability to utilize off-the-shelf PDDL solvers like fast downward in domains exhibiting partial observability is one of its advantages.

To evaluate the benefits of our planner, we compared it to the state-of-the-art object-oriented POMDP planner OO-POMCP [5]. Like our planner, OO-POMCP exploits the

	10 objects, 5 cupboards			15 objects, 10 cupboards		
	Avg Time (s)	Nodes Expanded	Succ %	Avg Time (s)	Nodes Expanded	Succ %
POMDP	265.673 ± 2.409	N/A	14.0	N/A	N/A	0.0
MDP	8.312 ± 0.889	873880	**100.0**	163.691 ± 8.396	11649654	53.0
LOMDP	3.905 ± 0.352	375899	**100.0**	78.860 ± 7.917	5347269	93.0
LOMDP-H	**2.680 ± 0.258**	340473	**100.0**	51.220 ± 6.166	4934536	97.1
LOMDP-PR	2.979 ± 0.227	378616	**100.0**	60.215 ± 6.920	5884534	**98.6**
LOMDP-HPR	3.048 ± 0.322	**290851**	**100.0**	**32.202 ± 4.224**	**2351858**	95.0

TABLE I

SOLVING LARGER PBJ DOMAINS AS POMDPS, MDPS, AND LOMDPS (WITH SIMPLE OPTIMIZATIONS APPLIED), AVERAGED OVER 100 TRIALS.

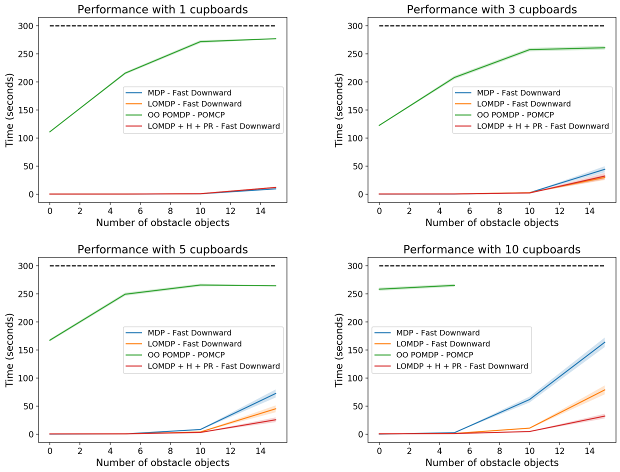

Fig. 2. Solution time using POMDP, MDP, LOMDP, and LOMDP with basic optimizations, with shaded standard error. Domain size is varied by increasing the number of objects and the number of regions that would facilitate information gain. Each solver was run for 100 iterations on each parameter setting, capped at 5 minutes (dotted line).

object-centric nature of the domain; unlike our planner, it does not exploit the structural nature of LOMDPs by using an observe-plan loop. Finally, we included a baseline that solved fully observable versions of the task by applying fast downward to the complete PDDL domain descriptor.

We varied the domain by changing the number of cupboards and the number of additional obstacle objects that could clutter the cabinets. These extra objects highlight an interesting failure mode: POMDP solvers tend to try to try to model the complete distribution of possible states before uncertainty has been reduced; we found in practice that POMCP scaled poorly in the number of obstacles and was unable to find any reward, devolving to taking actions seemingly at random. We evaluate each baseline for planning time, while varying both the number of cupboards and the number of obstacle objects. The results are shown in Figure 2, where each graph shows performance as the number of obstacles is increased, for a fixed number of cupboards.

Our results show that OO-POMCP always performs the worst by a significant margin and on our larger test cases it failed to solve the task at all, reaching our experimental timeout of five minutes. It generally converged to repeating a sequence of actions that made no progress towards the goal as the branching factor and task horizon exceeded its ability to reach any reward. By contrast, the LOMDP planner scales gracefully—despite its simple instantiation—solving tasks with 10 cupboards and 15 objects in less than two minutes. This is because it quickly decides it lacks sufficient information to solve the task and looks in a cupboard to find a new object, instead of expending computation modeling every possible assignment of object to cupboard without making progress on the task, which is OO-POMCP does.

Interestingly, as the number of cupboards scaled up we found that the LOMDP model actually outperformed the MDP with full information. We hypothesize that this is because the LOMDP is providing a form of scoping [16]. In a worst case run the LOMDP planner would observe every object, after which it plans over exactly the same domain description as the MDP. But as the amount of irrelevant information in the domain grows, the agent can solve the task without observing many useless objects, leaving the planning process operating over a smaller domain and therefore more efficiently. In Table I, we see that in the largest domain, all LOMDP based methods succeed[2] at rates greater than 90%, where the POMDP and MDP based methods struggle at 0% and around 50% respectively.

In Table I, we see that as the number of obstacles increases, the heuristic for which objects are task relevant (indicated by -H) improves both computation time and decreases the total number of nodes expanded over selecting an observation target at random. To model situations in which objects were biased to be in some cupboards more often than others, we also randomly generated categorical prior distributions by sampling from a symmetric Dirichlet distribution. When those priors are made available (indicated by -PR), the amount of time required to solve the task goes down, even though the total number of nodes expanded remains comparable. When combining knowledge of prior distributions with the heuristic for prioritizing important objects (indicated by -HPR), we see a strong cumulative improvement over the naive planner. This suggests that, despite the large magnitude of improvement over both the POMDP and MDP baselines, even further gains are possible via informed implementations of `next_target` and `find`.

[2] Success is measured as solving the task within five minutes.

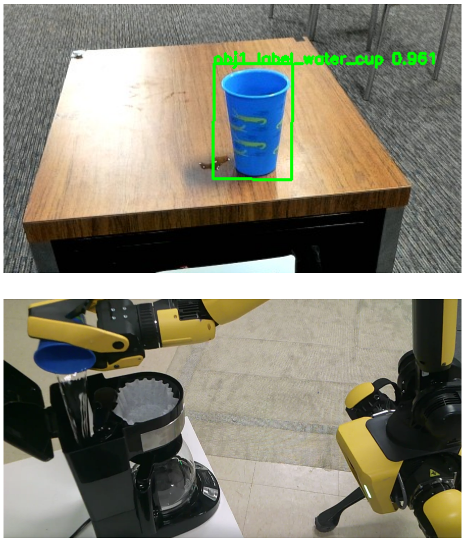

Fig. 3. Upon reaching a new locale, the robot observes the water cup and adds it to the MDP (left). With the water cup observed, Spot can plan to pour the water into the coffee machine (right).

B. Robot Experiments

To show that a LOMDP planner can generate goal-directed behavior on a real robot, we implemented LOMDP planning for a coffee-making task on a Boston Dynamics Spot[3]. The robot must find two cups with initially unknown positions, one containing coffee grinds and one containing water, and bring each one back to the coffee machine to pour its contents into the appropriate receptacle of the machine. The robot must then close the lid of the machine and press the button to turn it on. An additional object was present and initialized to an unobserved location; it could be interacted with similarly to the other cups, but did not progress towards the goal.

Navigation between locales was aided by AR tags at each location. Once the robot reached a locality it used ResNet [17] to `observe`, using multiple matching outputs to confirm object classification. This identified and localized objects in the scene. The remaining skills for pouring and interacting with the coffee machine were designed by hand.

A typical task execution proceeded as follows: At task start the location of all three cups is unknown. The agent tries to plan, but no ingredient cups have been observed, so the task is impossible to solve. The robot now chooses to observe the water cup, since observing it may help solve the task, and plans to navigate to locale 1, which is one of the tables. At locale 1 it observes the coffee cup, which it adds to its list of known objects. However, the water cup has not yet been observed so it now plans to navigate to locale 2, where it observes the water cup. The robot now has sufficient information to solve the task. No additional locales are visited, so the extra cup remains unobserved. The robot first brings the water over to the coffee machine and pours it in, then returns to locale 1 to do the same for the coffee grinds. Finally, it closes the lid of the machine and presses the button to turn the machine on.

Our experiments demonstrate that Markov planners can be made effective at solving challenging partially observable tasks when used as a component of planning within the LOMDP framework. We show that this technique scales in a combinatorially complex symbolic domain, and demonstrate that our approach can solve the challenging task of making coffee from ingredients initially hidden from the robot.

VI. CONCLUSION

Efficiently dealing with state uncertainty is a core challenges in robotics [2]; however, the most common general model of that uncertainty—the POMDP—does not admit efficient solutions, despite two decades of effort since its introduction [1]. LOMDPs are motivated by the hypothesis that the key to efficient planning is not more sophisticated planners, but a more structured model—that by carefully modeling the characteristics of real robots we can formulate a model sufficiently general to be widely useful but specific enough to be efficiently solvable. LOMDPs model the limited sensor range and line-of-sight nature of real robot sensors, while making the additional assumption that observable objects can be sensed accurately. This structure enables a robot to plan to observe objects of interest, after which those objects are treated as observed and input to a Markov planner.

A useful comparison here is to SLAM [2], which exploits a structured model of sensors and space to efficiently localize (turning a POMDP into an MDP) and map unknown space (learning a model of that MDP, thereby enabling planning). SLAM is now so effective that off-the-shelf implementations are included with most mobile robots [18, 19, 20]. We propose that, just as SLAM deals with uncertainty about space (the central type of uncertainty for navigation), a similarly specialized model is required for uncertainty about *object state* (which we hypothesize is the central type of uncertainty for manipulation). Similarly, just as SLAM builds an envelope of mapped space around the robot that is suitable for use by a navigation planner, so should the robot build a collection of observed objects that are suitable for input to a manipulation planner; and just as a core question for task-based SLAM is which region of space to explore next [21], a core question for manipulation planning under uncertainty should be which object to observe next. The LOMDP formalism was designed to meet these criteria.

Our results show that a simple planner that exploits the properties of a LOMDP—most importantly, decomposing planning into searching for unobserved objects and task planning using only observed objects—can solve problems much larger than would be feasible using a generic POMDP planner. This promising result is strongly suggestive of the potential of structured models of uncertainty; in future work, we plan to design more efficient planners with the goal of realizing a planner as efficient under object state uncertainty as SLAM approaches are under spatial uncertainty.

[3] Video and further description can be found at https://youtu.be/K90l3tYlg2U

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] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2005.

[3] C. Boutilier, R. Dearden, and M. Goldszmidt, “Exploiting structure in policy construction,” in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995, pp. 1104–1113.

[4] S. C. Ong, S. W. Png, D. Hsu, and W. S. Lee, “Planning under uncertainty for robotic tasks with mixed observability,” *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1053–1068, 2010.

[5] A. Wandzel, Y. Oh, M. Fishman, N. Kumar, W. L. LS, and S. Tellex, “Multi-object search using object-oriented POMDPs,” in *Proceedings of the 2019 IEEE International Conference on Robotics and Automation*, 2019, pp. 7194–7200.

[6] D. Koller and R. Parr, “Policy iteration for factored MDPs,” in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000, pp. 326–334.

[7] C. Diuk, A. Cohen, and M. Littman, “An object-oriented representation for efficient reinforcement learning,” in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 240–247.

[8] R. P. Petrick and F. Bacchus, “Extending the knowledge-based approach to planning with incomplete information and sensing,” in *International Conference on Automated Planning and Scheduling*, 2004.

[9] 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*.

[10] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *Proceedings of the 2017 IEEE International Conference on Robotics and Automation*, 2017, pp. 3357–3364.

[11] B. Mayo, T. Hazan, and A. Tal, “Visual navigation with spatial attention,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 898–16 907.

[12] K. Zheng, R. Chitnis, Y. Sung, G. Konidaris, and S. Tellex, “Towards optimal correlational object search,” in *Proceedings of the 2022 IEEE International Conference on Robotics and Automation*, 2022, pp. 7313–7319.

[13] R. Zeng, Y. Wen, W. Zhao, and Y.-J. Liu, “View planning in robot active vision: A survey of systems, algorithms, and applications,” *Computational Visual Media*, vol. 6, pp. 225–245, 2020.

[14] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “PDDL—the planning domain definition language,” Yale Center for Computational Vision and Control, Tech. Rep. CVC TR98003/DCS TR1165, 1998.

[15] M. Helmert, “The fast downward planning system,” *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[16] N. Kumar, M. Fishman, N. Danas, S. Tellex, M. Littman, and G. Konidaris, “Task scoping for efficient planning in open worlds (student abstract),” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 10, 2020, pp. 13 845–13 846.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[18] S. Macenski and I. Jambrecic, “SLAM toolbox: SLAM for the dynamic world,” *Journal of Open Source Software*, vol. 6, no. 6, p. 2783, 2021.

[19] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D LIDAR SLAM,” in *Proceedings of the 2016 IEEE International Conference on Robotics and Automation*, 2016, p. 1271–1278.

[20] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with Rao-Blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, pp. 34–46, 2007.

[21] G. Stein, C. Bradley, and N. Roy, “Learning over subgoals for efficient navigation of structured, unknown environments,” in *Proceedings of the Conference on Robot Learning*, 2018.