Partially Observable Task and Motion Planning with Uncertainty and Risk Awareness

Aidan Curtis, George Matheos, Nishad Gothoskar, Vikash Mansinghka,
Joshua Tenenbaum, Tomás Lozano-Pérez, Leslie Pack Kaelbling

MIT Computer Science and Artificial Intelligence Laboratory

{curtisa, gmatheos, nishadg, vkm, tlp, lpk}@mit.edu.

Abstract—Integrated task and motion planning (TAMP) has proven to be a valuable approach to generalizable long-horizon robotic manipulation and navigation problems. However, the typical TAMP problem formulation assumes full observability and deterministic action effects. These assumptions limit the ability of the planner to gather information and make decisions that are risk-aware. We propose a strategy for TAMP with Uncertainty and Risk Awareness (TAMPURA) that is capable of efficiently solving long-horizon planning problems with initial-state and action outcome uncertainty, including problems that require information gathering and avoiding undesirable and irreversible outcomes. Our planner reasons under uncertainty at both the abstract task level and continuous controller level. Given a set of closed-loop goal-conditioned controllers operating in the primitive action space and a description of their preconditions and potential capabilities, we learn a high-level abstraction that can be solved efficiently and then refined to continuous actions for execution. We demonstrate our approach on several robotics problems where uncertainty is a crucial factor and show that reasoning under uncertainty in these problems outperforms previously proposed determinized planning, direct search, and reinforcement learning strategies. Lastly, we demonstrate our planner on two real-world robotics problems using recent advancements in probabilistic perception.

I. INTRODUCTION

In an open-world setting, a robot's knowledge of the environment and its dynamics is inherently limited. If the robot believes it has full knowledge of the state and dynamics of the world, it may confidently take actions that have potentially catastrophic effects, and it will never have a reason to seek out information. For these reasons, it is crucial for the robot to know what it does not know and to make decisions with an awareness of risk and uncertainty.

Advances in techniques like behavior cloning (BC) [1, 2], reinforcement learning (RL) [3, 4], and model-based control [5, 6] have made it possible to develop robotic controllers for short time-horizon manipulation tasks in partially observable or stochastic domains. In situations matching a narrow training distribution (BC), with dense reward and short horizons (RL), or conforming to modeling assumptions, these controllers can be quite robust. However, these methods typically do not generalize to solving arbitrary complex goals over long time horizons.

Simultaneously, recent advances in Task and Motion Planning (TAMP) have illustrated the viability of using planners to sequence such controllers to robustly achieve tasks over longer time horizons in large open-world settings [7, 8, 9]. In
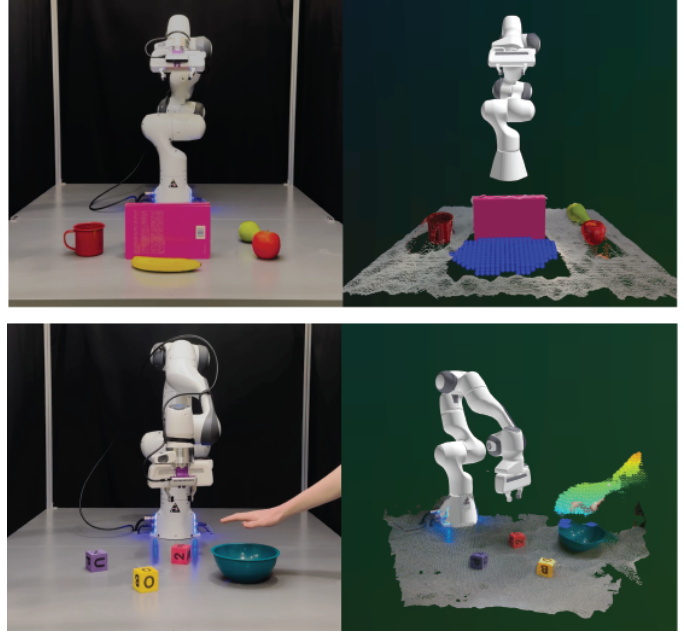

Fig. 1: Top: Robot with wrist mounted camera looking for a banana. The robot plans to take information gathering actions based on a posterior estimate of the banana's pose shown in blue. Bottom: Robot with one wrist mounted camera and one external camera plans to complete a long-horizon manipulation task while avoiding a human in the workspace.

TAMP, the key to tractable planning over long time horizons is to sequence short-horizon controllers, exploiting a description of the conditions in which each controller can be expected to work, and of each controller's effects. However, most TAMP formulations assume that these symbolic descriptions perfectly and deterministically characterize the effects of running the controllers. In real robotics settings, stochasticity and partial observability make it impossible to exactly predict the effects of controllers. Furthermore, it is typically impossible to obtain exact symbolic descriptions that fully capture the effects and preconditions of each controller.

This paper shows how to extend TAMP to settings with partial observability, uncertainty, and imperfect symbolic descriptions of controllers. Our approach, TAMPURA, is to exploit a coarse model of each controller's preconditions and effects to rapidly solve deterministic, symbolic planning problems that guide the construction of a non-deterministic Markov Decision
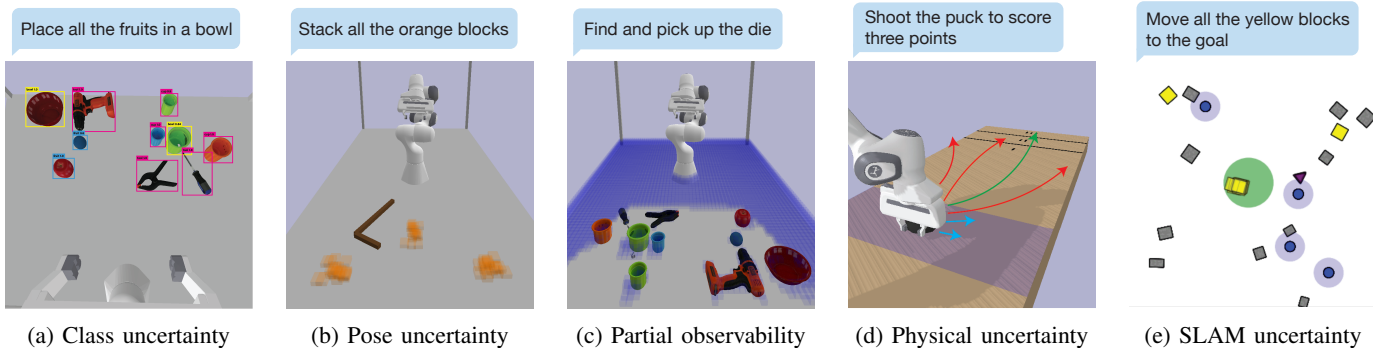
Fig. 2: This figure illustrates five long-horizon planning tasks that TAMPURA is capable of solving. Each of them contains a unique type of uncertainty including uncertainty in (a) classsification, (b) pose due to noisy sensors or (c) partial observability, (d) physical properties such as friction or mass, and (e) localization/mapping due to odometry errors

Process (MDP) with only a small number of actions applicable from each state (Figure 3). This smaller model captures the key tradeoffs between utility, risk, and information-gathering in the original planning problem. The resulting MDP is sparse enough that high-quality uncertainty-aware solvers like LAO* [10] can be applied. The guidance used to distill this small MDP comes in the form of symbolic descriptions of the preconditions, and the uncertain but possible effects, of each controller. The MDP is constructed by learning the probability distribution over these possible effects for each controller, refining coarse and imperfect descriptions into transition distributions which can be used for uncertainty and risk-aware planning. In this paper, we use controller descriptions provided by engineers; in Section VIII, we comment on how future work could enable such descriptions to be learned or generated with large language models, following [11] or [12].

We demonstrate the applicability of TAMPURA in a wide range of simulated problems (Figure 2), and show how it can be applied to two real-world robotics tasks: searching a cluttered environment to find objects, and operating safely with an unpredictable human in the workspace (Figure 1). We show that in tasks requiring risk sensitivity, information gathering, and robustness to uncertainty, TAMPURA significantly outperforms reinforcement learning, Monte Carlo tree search, and determinized belief-space task and motion planners, even when these algorithms are all given access to the same controllers.

II. RELATED WORK

Planning under environment uncertainty and partial observability is a longstanding problem with a diversity of approaches. While exact methods are typically only suited for small discrete problems, approximate methods [13, 14] have shown that online planning with frequent replanning can work well for many non-deterministic, partially observable domains with large state and observation spaces. These methods directly search within a primitive action space and are guided by reward feedback from the environment. However, in the absence of dense reward feedback, the computational complexity of these methods scales exponentially with the planning horizon, action space, and observation space.

Task and motion planning refers to a family of methods that solve long-horizon problems with sparse reward through temporal abstraction, factored action models, and primitive controller design [15, 16, 7]. While most TAMP solutions assume deterministic transition dynamics and full observability, several approaches have extended the framework to handle stochastic environments or partial observability. Some TAMP solvers remove the assumption of deterministic environments [17, 18, 11]. While these approaches can find contingent task and motion plans [17] or use probabilities to find likely successful open-loop plans [11], they operate in state space and assume prior information about transition probabilities in the form of hardcoded probability values [17], or demonstration data in the form of example plans [11]. In contrast, our proposed approach learns belief-space transition probabilities through exploration during planning.

Another family of TAMP solvers plan in belief space [19, 20, 21, 18], allowing them to plan to gather necessary information, even in long-horizon contexts. Some approaches, such as IBSP and BHPN do forward or backward search in the continuous belief space, respectively [21, 20]. SSReplan [19] embeds the belief into the high-level symbolic model. While these approaches to TAMP in belief space have their tradeoffs, all of them perform some form of determinization when planning. Determinization allows the planner to only consider one possible effect of an action, which inherently limits its ability to be risk-aware, incorporate action cost metrics, and perform well under large observational branching factors.

To our knowledge, there exists one other belief space TAMP planner that does not determinize the transition dynamics during planning [18]. However, this approach focuses on contingent planning, where there are no probabilities associated with nondeterministic outcomes. In our experiments, we compare to many of these approaches to show the power of stochastic belief-space planning with learned action effect probabilities.
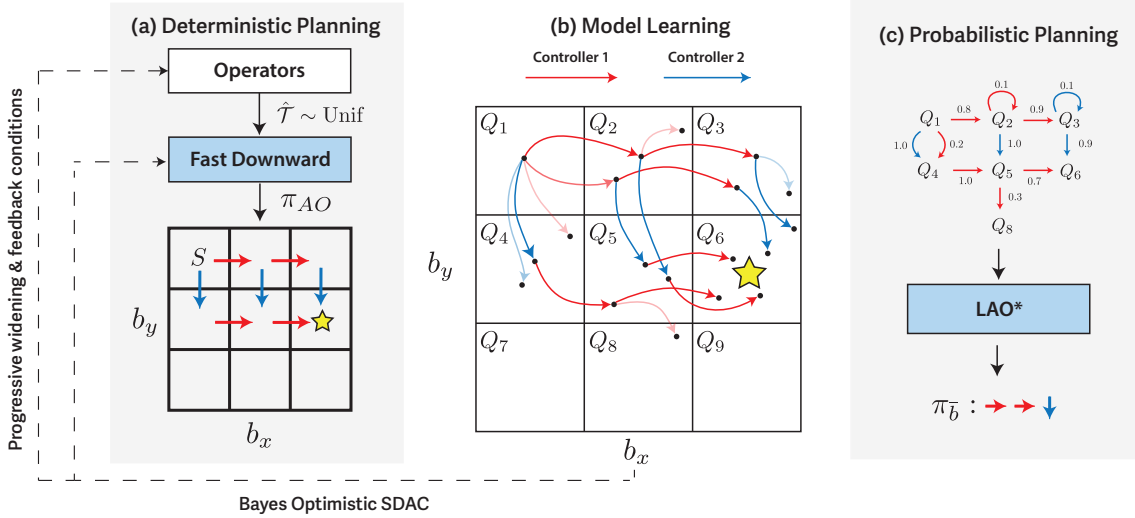
Fig. 3: Uncertainty and Risk Aware Task and Motion Planning. (a) The robot's continuous space of probabilistic beliefs about world state is partitioned into a discrete abstract belief space, here with 9 states. TAMPURA considers a set of operators, each containing a low-level robot controller, and a description of the possible effects of executing the controller. Determinized planning computes possible sequences of controllers to reach the goal. These plans do not factor in uncertainty or risk and would be unsafe or inefficient to execute in the real world. (b) The determinized plans are executed in a mental simulation. The distribution of effects is recorded, to learn an MDP on the space of abstract belief states visited in these executions. By iterating between determinized planning and plan simulation (Sec. V-C), TAMPURA learns a sparse MDP related to the original decision problem. (c) The robot calculates an uncertainty and risk aware plan in the sparse MDP, and executes it.

III. BACKGROUND

One way to formulate many sequential decision problems involving uncertainty is as Partially Observable Markov Decision Processes (POMDPs). A POMDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, r, b_0, \gamma \rangle$.[1] $\mathcal{S}, \mathcal{O}$, and $\mathcal{A}$ are the state, observation, and action spaces. The state transition and observation probabilities are $\mathcal{T}(s_{t+1} \mid s_t, a_t)$ and $\mathcal{Z}(o_t \mid s_t)$, and b_0 is a probability distribution on $\mathcal{S}$ giving the distribution of possible initial states. The reward function is $r(s_t, a_t, s_{t+1})$, and γ is a discount factor quantifying the trade-off between immediate and future rewards.

A. Belief-State MDP

From any POMDP, one can derive the continuous belief-space MDP $\mathcal{M}_b = \langle \mathcal{B}, \mathcal{A}, \mathcal{T}_b, r_b, b_0, \gamma \rangle$ [22]. The state space of this MDP is $\mathcal{B}$, the space of probability distributions over $\mathcal{S}$, or belief states. The initial belief state is $b_0 \in \mathcal{B}$, describing the robot's belief before any actions have been taken or any observations have been received. The reward r_b is derived from r; γ is unchanged. The transition distribution $\mathcal{T}_b(b_{t+1} \mid b_t, a_t)$ is the probability that after being in belief state b_t and taking action a_t, the robot will receive an observation causing it to update its belief to b_{t+1}.

B. Belief updates

As TAMPURA plans in a belief-space MDP, it must keep track of the robot's current belief state. However, computing

the belief updates exactly is intractable in many problems. Fortunately, in cases where exact belief updates cannot be computed, it can suffice to compute approximate belief states using approximate Bayesian inference methods like particle filtering, or more generally, sequential Monte Carlo [23, 24]. Further, recent advances in probabilistic programming [25, 26, 27, 28] and its application to 3D perception [29, 30, 31] have made it practical to generate belief distributions over latent states describing the poses of 3D objects, their contact relationships, and to update these beliefs in light of new RGB-Depth images of a 3D scene. In our real-world robotic experiments, probabilistic perception is performed using Bayes3D [29].

C. Belief-State Controller MDP

When the action space $\mathcal{A}$ represents primitive controls to the robot such as joint torques or end-effector velocity commands, the time horizons to perform meaningful tasks can be enormous, rendering planning intractable. To mitigate this, we introduce the concept of a belief-space controller, which takes the current belief as input and executes in closed-loop fashion over extended time horizons. For example, in our 2D SLAM task modeling a mobile robot moving with pose uncertainty, our "move to point X" controller has access to a distribution over possible robot poses. The controller selects actions that would not result in collisions under any possible poses in the support of this distribution, sometimes ruling out unsafe low-level actions that would seem safe and more efficient given only the most likely pose of the robot.

Given a set of learned or designed controllers, the primitive belief-space MDP can be lifted to a temporally abstracted

---

[1] A reference for all notation introduced henceforth is provided in Table IV in the appendix.

belief-space controller MDP $\mathcal{M}_c = \langle \mathcal{B}, \mathcal{C}, \mathcal{T}_c, r_c, \gamma \rangle$. The action space of this MDP is the space of controllers, and the transition model $\mathcal{T}_c(b_{t+1} \mid b_t, c)$ gives the probability that if controller $c \in \mathcal{C}$ is executed beginning in state b_t, after it finishes executing, the belief state will be b_{t+1}. This paper considers the problem of planning in this belief-state controller MDP, to enable a robot to determine which low-level controllers to execute at each moment.

IV. PLANNING WITH AN ABSTRACT BELIEF-STATE MDP

Direct search in $\mathcal{M}_c$ is intractable in many problems due to the large branching factors in the action space and continuous belief outcomes. To make the problem tractable, TAMPURA applies a series of reductions to $\mathcal{M}_c$, ultimately producing a sparse abstract MDP $\mathcal{M}_s$ that can be solved efficiently with a probabilistic planner (Figure 3).

The reduction from $\mathcal{M}_c$ to $\mathcal{M}_s$ is performed in several steps. First, we perform belief-state abstraction, lifting from an MDP on $\mathcal{B}$ to an abstract MDP on $\overline{\mathcal{B}}$, which is a partition of $\mathcal{B}$ that groups operationally similar beliefs (Section IV-B). Second, we leverage symbolic information describing the preconditions and possible but uncertain effects of each controller $c \in \mathcal{C}$ (Section IV-C) to construct a determinized shortest path problem on the abstract belief space. Determinized planning in abstract belief space with controller descriptions is now tractable, but the resulting plans are not risk-aware due to determinization. In addition, the resulting plans may be overly optimistic because they are untethered from geometric and physical constraints. Third, we approximately learn the transition model $\overline{\mathcal{T}}$ on $\overline{\mathcal{B}}$, using the efficient determinized planner to focus exploration on the task-relevant parts of the abstract belief space (Section V). The subset $\mathcal{B}_{\text{sparse}} \subseteq \overline{\mathcal{B}}$ of abstract belief states focused on in model learning forms the state space of the sparse abstract MDP; its transition model is the learned transition probabilities, $\hat{\mathcal{T}}$, approximating the true $\overline{\mathcal{T}}$. This sparse MDP distills key tradeoffs about risk, information-gathering, and outcome uncertainty from the original problem. It can be solved with a probabilistic planner such as LAO*, resulting in a risk and uncertainty aware policy in the abstract belief-state MDP. The first action recommended by this policy is the next controller to execute on the robot. The full TAMPURA robot control loop is given in Algorithm 1.

A. Belief state propositions

To apply TAMPURA in a controller-level MDP $\mathcal{M}_c$, an abstract belief space must be defined through the specification of a set $\Psi_{\mathcal{B}}$ of belief state propositions. Each $\psi \in \Psi_{\mathcal{B}}$ is a boolean function $\psi : \mathcal{B} \to \{0, 1\}$ of the robot's belief. As described in [20], belief propositions can be used to describe comparative relationships such as (MLCat ?o ?c), meaning the most likely category of object ?o is category ?c, statements about the probable values of object properties such as (BPose ?o ?x), meaning the probability object ?o's position is within δ of ?x is greater than $1 - \epsilon$, and other statements about the distribution over a value such as (BVPose ?o), meaning there exists some position ?x such that object ?o is within δ

Algorithm 1 TAMPURA Control Loop

Require: Planning problem: $(\mathcal{T}_c, \mathbb{O}, r_c, b_0)$
1: $s \leftarrow \emptyset$ ▷ Initialize state used by model learning.
2: $\mathcal{B}_{\text{sparse}}, \hat{\mathcal{T}} \leftarrow \emptyset, \emptyset$ ▷ Initialize result of model learning.
3: **while** abs(b) $\notin G$ **do**
4:     **if** abs(b) $\notin \mathcal{B}_{\text{sparse}}$ **then**
5:         args $\leftarrow (b_0, G, \mathbb{O}, s)$
6:         $s, \hat{\mathcal{T}}, \mathcal{B}_{\text{sparse}} \leftarrow$ Model-Learning(args)
7:     ▷ Solve the MDP with $\hat{\mathcal{T}}$ and r_c over $\mathcal{B}_{\text{sparse}}$
8:     $\pi \leftarrow$ LAO-Star($\mathcal{B}_{\text{sparse}}, \hat{\mathcal{T}}, r_c$)
9:     ▷ Get controller recommended by policy π in b_0.
10:     $c \leftarrow \pi(\text{abs}(b_0))$
11:     $(\vec{o}, \vec{a}) \leftarrow$ Execute(c)
12:     $b_0 \leftarrow$ BeliefUpdate($b_0, \vec{o}, \vec{a}$)

---

of x with probability greater than $1 - \epsilon$. In practice, these are implemented as lifted symbol grounding functions that take in a set of entities along with the current belief and return a boolean value.

B. The abstract belief-state MDP

Given a particular belief $b \in \mathcal{B}$, evaluation under the proposition set results in an abstract belief abs(b) := $\{\psi \mapsto \psi(b) : \psi \in \Psi_{\mathcal{B}}\}$ which is a dictionary mapping from each proposition ψ to its value $\psi(b)$ in belief b. The abstract belief space is then defined as $\overline{\mathcal{B}} := \{\text{abs}(b) : b \in \mathcal{B}\}$. Under a particular condition on this set of propositions (Appendix J) which we assume to hold in this paper,[2] we can derive from $\mathcal{M}_c$ an abstract belief-state controller MDP $\overline{\mathcal{M}_c} := \langle \overline{\mathcal{B}}, \mathcal{C}, \overline{\mathcal{T}}, r_c, \bar{b}_0, \gamma \rangle$. The state space of this MDP is the discrete abstract belief space $\overline{\mathcal{B}}$, rather than the uncountably infinite belief space $\mathcal{B}$; transition probabilities on $\overline{\mathcal{B}}$ are given by $\overline{\mathcal{T}}$. The initial state is $\bar{b}_0 := \text{abs}(b_0)$.

In this paper, we focus on planning problems with objectives modeled as goals in belief space (e.g., the goal may be to believe that with high probability the world is in a desired set of states.) Specifically, we consider the case where the reward is a subset of the abstract belief space: $G \subset \overline{\mathcal{B}}$ such that the goal can be defined in terms of the belief-space propositions. We also model episodes as terminating at the first moment a goal belief state is achieved. This restricts the controller-level MDP under consideration to a belief-space stochastic shortest paths problem (BSSP).

C. Operators with uncertain effects

These belief-space propositions give us a language to describe the preconditions and possible effects of executing a controller. We call such descriptions operators $\text{op} \in \mathbb{O}$. Each operator is a tuple $\langle \text{Pre}, \text{Eff}, \text{UEff}, \text{UCond}, c \rangle$. Here, $\text{Pre} \subseteq \Psi_{\mathcal{B}}$ is the set of belief propositions that must hold for a controller $c \in \mathcal{C}$ to be executed, $\text{Eff} \subseteq \Psi_{\mathcal{B}}$ is the set of effects that are guaranteed to hold after c has been executed, $\text{UEff} \subseteq \Psi_{\mathcal{B}}$ is the set of belief propositions that have an

---

[2]TAMPURA can be run when this condition does not hold and we expect its performance to degrade gracefully in this case. See Appendix J for details.

unknown value after the completion of c, and UCond $\subseteq \Psi_{\mathcal{B}}$ represents the set of propositions upon which the probability distribution over the UEff may depend. (That is, given an assignment to the propositions in UCond, there should be a fixed distribution on UEff, though this distribution need not be known a priori.)

As a result of this additional structure, from any given abstract belief space $\bar{b} \in \overline{\mathcal{B}}$, only a small number of operators can be applied, as most operators will not have their preconditions satisfied. Additionally, planners can exploit the knowledge that after applying an operator from state $\bar{b}$, the only reachable new states are those which modify $\bar{b}$ by turning on the propositions in Eff, and possibly turning on some propositions in UEff.

D. Operator schemata

In our implementation, the set of operators and the set of controllers are generated from a set of operator schemata. Each operator schema describes an operation which can be applied for any collection of entities with a given type signature, for any assignment to a collection of continuous parameters the controller needs as input. $\mathbb{O}$ is the set of grounded, concrete operators generated from an assignment of objects and continuous parameters to an operator schema.

We introduce an extension to PDDL for specifying schemata for controllers with uncertain effects. An example operator schema written in this PDDL extension is shown below.

```
(:action pick
 :parameters (?o - object ?g - grasp)
 :precondition (and (BVPose ?o) (BHandFree))
 :effects (and ¬ (BVPose ?o))
 :uconds (and (BClass ?o @glass))
 :ueffects (maybe (Broken ?o) (BGrasp ?o ?g)))
```

For any entity o with type(o) = object, and any continuous parameter g with type type(g) = grasp, this operator schema yields a concrete operator $\text{pick}_{o,g} \in \mathbb{O}$. As specified in the :precondition, this operator can only be applied from beliefs where the pose of o is known (BVPose ?o) and the robot's hand is believed to be free (BHandFree). As specified in :effects, after running this, it is guaranteed that there will not exist any pose p on the table such that the robot believes o is at p with high probability. The :ueffects field specifies two possible but not guaranteed effects. Following a controller execution, these effects evaluated on the updated belief belief using the symbol grounding functions in $\Psi_{\mathcal{B}}$. The overall effect of this operator can be described as a probability distribution on the four possible joint outcomes. The :uconds field specifies that this probability distribution will be different when o is believed to be glass than when it is not. Such a difference in outcome distributions may lead the planner to inspect the class of an object before attempting to grasp it. Our semantics are similar to those in PPDDL [32] and FOND [33], but are agnostic to the exact outcome probabilities and ways in which the conditions affect those probabilities.

V. Learning the Sparse Abstract MDP

While the operator schemata are helpful for guiding planning, they lack outcomes probabilities that are crucial for finding a kinematically and geometrically valid plan that is safe and efficient. [3] For any controller c and any abstract belief state $\bar{b}$, it is possible to learn the outcome distribution $\overline{\mathcal{T}}(\cdot \mid \bar{b}, c)$ by simulating executions of c from belief states consistent with $\bar{b}$. However, obtaining estimates of these probabilities is computationally expensive as it can involve geometric calculations, perceptual queries, and simulations. Naive strategies like learning transition probabilities for $(\bar{b}, c)$ pairs sampled at random is highly inefficient (Figure 4, panel 2). Our solution is to leverage the symbolic structure and specified goal to determine which outcome distributions to learn for more efficient online model learning.

A. Solution-guided model learning

A seemingly natural strategy for goal-directed model learning is to first initialize $\overline{\mathcal{T}}$ so that $\overline{\mathcal{T}}(\cdot \mid \bar{b}_t, c)$ is uniform on the set of symbolically possible abstract belief states $\bar{b}_{t+1}$ specified in the UEffs of the operator corresponding to controller c. After solving the abstract belief state MDP derived from the partially learned $\overline{\mathcal{T}}$ to obtain a policy π, we could simulate π, producing a sequence of $(\bar{b}_t, c_t, \bar{b}_{t+1})$ transitions. The transitions could be used to update the transition probabilities and construct a more accurate MDP in a process of iterative improvement. The problem with this approach is that using the maximum likelihood estimate of the transition probabilities to guide exploration can converge to local optima, due to under-exploring actions for which the initial experience pool is poor. Workarounds like ϵ-greedy exploration can alleviate this, but are inefficient in problems with large action spaces as they explore the locally feasible action space rather than focusing on task-relevant actions (Figure 4, panel 5). For example, in a setting where the robot must pick up a particular object, local exploration would experiment with picking unrelated objects.

B. Bayes optimistic model learning

Ideally, we would like our exploration strategy to be optimistic in the face of model uncertainty. One standard way of implementing optimism in a planning framework is with all-outcomes determinization [34], wherein the planner is allowed to select the desired outcome of an action. This is done by augmenting the MDP action space with the possible action outcomes, resulting in a deterministic transition function $\mathcal{T}_{AO} : \mathcal{B} \times (\mathcal{A} \times \mathcal{B}) \to \mathcal{B}$.

This optimism leads to bad policies when useful outcomes occur with low probability. To avoid this, cost weights J can be added to the actions such that selecting outcomes with low probability is penalized. An optimal policy under the all-outcomes determinized model is an optimal open loop plan

---

[3]Transition probabilities can capture geometric and kinematic constraints that hard symbolic constraints do not rule out. For instance, a controller $\text{pick}_{o,g}$ may have (BGrasp ?o ?g) in its UEffs, even for a grasp g which is kinematically infeasible. This infeasibility will be captured once the transition probabilities are learned: the outcome has probability 0.

Algorithm 2 TAMPURA Online Model Learning

Require: Parameters for Bayesian model learning prior: α, β
Require: Parameters controlling runtime: I, K, S
Require: Planning problem: $(b_0, G, \mathbb{O})$
Require: State from past iterations of model learning: s

1: **if** $s = \emptyset$ **then**
2:    ▷ Initialize count dictionaries w/ default value 0.
3:      $N \leftarrow \texttt{DefaultDict}(\{\}, \text{default} = 0)$
4:      $D \leftarrow \texttt{DefaultDict}(\{\}, \text{default} = 0)$
5:    ▷ Initialize dict from abstract beliefs to corresponding concrete beliefs.
6:      $P_\mathcal{B} \leftarrow \texttt{DefaultDict}(\{\text{abs}(b_0) : [b_0]\}, \text{default} = [])$
7: **else**
8:      $(N, D, P_\mathcal{B}) \leftarrow s$
9:    ▷ Main model learning loop.
10: **for** $i = 1, \ldots, I$ **do**
11:    ▷ Plan + concatenate + filter K trajectories.
12:      $(\tau_k)_{k=1}^K \leftarrow \texttt{Determinized-Planner}(\bar{b}_0, K, \mathbb{O}, N, D, G)$
13:      $\tau^* \leftarrow [(\bar{b}, \text{op}, \bar{b}') \in \text{concat}(\tau_1, \ldots, \tau_k) : P_\mathcal{B}[\bar{b}] \neq []]$
14:    ▷ Compute preconditions + effects for each transition.
15:      $\vec{\Psi}_{\text{pre}} \leftarrow [[\bar{b}[\psi] : \psi \in \text{op.UCond}] : (\bar{b}, \text{op}, \bar{b}') \in \tau^*]$
16:      $\vec{\Psi}_{\text{eff}} \leftarrow [[\bar{b}'[\psi] : \psi \in \text{op.UEffs}] : (\bar{b}, \text{op}, \bar{b}') \in \tau^*]$
17:    ▷ List of controllers.
18:      $\vec{c} \leftarrow [\text{op.}c : (\bar{b}, \text{op}, \bar{b}') \in \tau^*]$
19:    ▷ Look up s: num times c with preconditions Ψ_{pre} led to effects Ψ_{eff}.
20:      $\vec{s} \leftarrow [D[x] : x \in \text{zip}(\vec{\Psi}_{\text{pre}}, \vec{c}, \vec{\Psi}_{\text{eff}})]$
21:    ▷ Compute f, num "failures" where c in Ψ_{pre} did not cause Ψ_{eff}.
22:      $\vec{f} \leftarrow [N[\Psi_{\text{pre}}, c] - s : (\Psi_{\text{pre}}, c, s) \in \text{zip}(\vec{\Psi}_{\text{pre}}, \vec{c}, \vec{s})]$
23:    ▷ Compute entropy H to focus simulations on uncertain cases.
24:      $\vec{H} \leftarrow [H(\alpha + s, \beta + f) : (s, f) \in \text{zip}(\vec{s}, \vec{f})]$
25:    ▷ Controller simulation loop.
26:     **for** $j = 1, \ldots, S$ **do**
27:        $(\bar{b}_1, \text{op}, \bar{b}_2) \leftarrow \text{pop}(\tau^*, \text{argmax}(\vec{H}))$
28:        $b_1 \sim \text{Unif}(P_\mathcal{B}[\bar{b}_1])$
29:        $b_2 \leftarrow \text{Simulate}(b_1, \text{op.}c)$
30:        $P_\mathcal{B}[\text{abs}(b_2)] \leftarrow \text{Append}(P_\mathcal{B}[\text{abs}(b_2)], b_2)$
31:        $\Psi_{\text{pre}} \leftarrow [\bar{b}_1[\psi] : \psi \in \text{op.UCond}]$
32:        $\Psi_{\text{eff}} \leftarrow [\psi(b_2) : \psi \in \text{op.UEff}]$
33:        $N[\Psi_{\text{pre}}, \text{op.}c] \leftarrow N[\Psi_{\text{pre}}, \text{op.}c] + 1$
34:        $D[\Psi_{\text{pre}}, \text{op.}c, \Psi_{\text{eff}}] \leftarrow D[\Psi_{\text{pre}}, \text{op.}c, \Psi_{\text{eff}}] + 1$
35:    ▷ Compile transition counts to sparse abstract MDP (Appendix B).
36:      $\hat{\mathcal{T}}, \mathcal{B}_{\text{sparse}} \leftarrow \text{Compile}(D, N, \mathbb{O})$
37: **return** $(N, D, P_\mathcal{B}), \hat{\mathcal{T}}, \mathcal{B}_{\text{sparse}}$

when cost weights are set to be $-\log(p)$ where p is the true outcome probability [20]. We make use of both of these strategies by initially collecting deterministic plans from a fully optimistic transition model, simulating the optimistic plans to gather transition data, and increasing the costs of outcomes as we gain certainty about the true transition probabilities.

We model partial knowledge about each outcome probability $\overline{\mathcal{T}}(\bar{b}_{t+1} \mid \bar{b}_t, c_t)$ using a Beta(α, β) distribution. (For our prior we use $\alpha = 1, \beta = 1$.) Given simulations of c_t from $\bar{b}_t$, the updated posterior is Beta$(\alpha + s, \beta + f)$, where s is the number of "successful" simulations which led to $\bar{b}_{t+1}$ and f is the number of other "failed" simulations.

To model optimism in the face of uncertainty, we set outcome costs in all-outcomes determinized search according to an upper confidence bound of the estimated probabilities. Since we model outcome probabilities using beta distributions, we use a Bayesian-UCB [35] criterion where the upper bound is defined by the ν quantile of the posterior beta distribution. This quantile decreases with respect to the total number of samples across all sampled outcomes. As in any UCB, the rate of this decrease is a hyperparameter, but a common choice is $\nu = 1/i$ because it leads to sublinear growth in risk and asymptotic optimality under Bernoulli distributed rewards. At the ith iteration of model learning, the Bayesian-UCB criterion corresponds to using all-outcomes costs

$$J(\bar{b}_t, c_t, \bar{b}_{t+1}) = \log\left[F^{-1}_{\text{Beta}(\alpha+s, \beta+f)}\left(1 - \frac{1}{i}\right)\right] \quad (1)$$

Here, $J(\bar{b}_t, c_t, \bar{b}_{t+1})$ is the cost applied to transition $(\bar{b}_t, c_t) \rightarrow \bar{b}_{t+1}$ in all-outcomes planning, F^{-1} is the inverse CDF of the Beta posterior, and s and f are as above.

This approach to model learning explores in the space of symbolically feasible goal directed policies, which is significantly more efficient than random action selection in problems with large action spaces and long horizons. In Figure 4 we compare our Bayes optimistic model learning strategy to the solution guided strategy described in Section V-A. Our experiments show that the bayes-optimistic approach to model learning outperforms ϵ-greedy for all values of ϵ even in a domain with a relatively small action space. Note that although we use determinization to attain optimism in model learning, we perform full probabilistic planning on the learned model, making the resulting policy risk-aware.

C. The TAMPURA model-learning algorithm

In this section, we describe the details of the TAMPURA model learning algorithm outlined in Algorithm 2. For each task-relevant operator $\text{op} \in \mathbb{O}$, model learning must learn a probability table which induces a distribution over joint assignments to the propositions in op.UEffs, given any joint assignment to the propositions in op.UConds. Algorithm 2 writes Ψ_{pre} to denote assignments to an operator's UConds and Ψ_{eff} for assignments to UEffs. This probability table is a compressed representation of $\overline{\mathcal{T}}$: for any symbolically feasible transition $(\bar{b}_t, c_t, \bar{b}_{t+1})$, the value of $\overline{\mathcal{T}}(\bar{b}_{t+1} \mid \bar{b}_t, c_t)$ only depends on $\bar{b}_t$ and $\bar{b}_{t+1}$ through their assignments to the propositions in UConds and UEffs respectively.

In the first model learning iteration, Algorithm 2 initializes a count map N where $N[\Psi_{\text{pre}}, c]$ is the number of times model learning has simulated controller c from a belief state b consistent with UCond assignment Ψ_{pre}. It also initializes map D where $D[\Psi_{\text{pre}}, c, \Psi_{\text{eff}}]$ is the number of simulations in which the belief state which arose after simulating c induced assignment Ψ_{eff} to the UEff propositions for the operator corresponding to c. The algorithm also initializes an abstract to concrete belief map $P_\mathcal{B}$. (Lines 1-8.) Inside of a model learning loop, the algorithm performs Bayes optimistic determinized planning using the Fast Downward planner with state-dependent action costs (SDAC [36]) derived from the partially
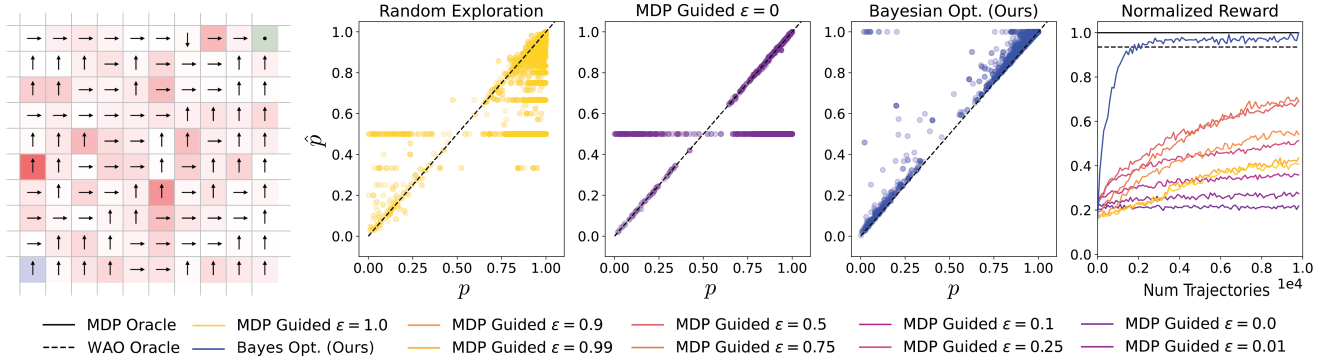
Fig. 4: Comparisons of model-learning strategies on a simplified grid-world environment in which an agent must navigate from the blue cell to the green cell. Red intensity corresponds to p, the probability of transitioning to an irrecoverable state. p for each cell is initially unknown, and must be estimated through interaction with the environment. The optimal policy given known p for this sample environment is indicated with arrows. The scatter plots compare the estimated $\hat{p}$ to true p at the end of model learning for several strategies across 50 different environments. The rightmost plot shows average normalized reward as a function of the number of training trajectories for our method as well as the MDP-guided method with a variety of values of epsilon. Our method quickly reaches near optimal performance, surpassing the weighted all-outcomes determinized solution under ground truth outcome probabilities.

learned model according Equation 1. The resulting plans take the form of a sequence of triples $(\bar{b}_t, \text{op}, \bar{b}_{t+1})$ specifying that controller $\text{op}.c$ executed in abstract belief state $\bar{b}_t$ transitions to $\bar{b}_{t+1}$ (Line 12). These triples are filtered to those where the abstract belief has known corresponding concrete beliefs in $P_{\mathcal{B}}$ (Line 13). A subset of the remaining triples are then chosen for simulated outcome sampling; each chosen triple $(\bar{b}_1, \text{op}, \bar{b}_2)$ is selected if the Beta distribution describing partial knowledge about $\overline{\mathcal{T}}(\bar{b}_2 \mid \bar{b}_1, \text{op}.c)$ has maximal entropy H among the available options (Line 24). After simulating $\text{op}.c$ from some belief state b_1 consistent with $\bar{b}_1$, and producing concrete belief state b_2, the algorithm computes the Ψ_{pre} corresponding to $\bar{b}_1$ and the Ψ_{eff} corresponding to b_2, and updates the counts in N and D (Lines 29-34). At the end of model learning, N and D are compiled into a transition model $\hat{\mathcal{T}}$ on the subset $\mathcal{B}_{\text{sparse}} \subseteq \mathcal{B}$ of abstract belief states reachable from b_0 by applying sequences of operators explored in model learning (Line 36). See Appendix B for details.

D. Progressive widening

Per Section IV-D, operators and controllers are derived by binding operator schemata to assignments of objects and continuous parameters. In Algorithm 2, we assume the set of continuous inputs are fixed and prespecified. The algorithm can be extended by using progressive widening to gradually expand the operator set $\mathbb{O}$, adding in new operator instances corresponding to applications of controller schemata bound to new continuous parameters drawn from a stream of sampled values (see Appendix G).

Because this effectively increases the state and action space of the abstract MDP, care must be taken to expand this set gradually so that the expansion of the MDP does not outpace the optimistic exploration. To achieve this, we use a progressive widening criteria typically used in hybrid discrete-continuous search problems [37]. Our full TAMPURA imple-

mentation incorporates progressive widening by adding a line before Line 12 in Algorithm 2 to add elements to $\mathbb{O}$.

Such widening increases continuous action input samples based on the number of times a ground operator has been visited, maintaining the following relationship during model learning for each controller simulation from belief b:

$$k \cdot \sum_{\text{op}}^{O} N[\Psi_{\text{pre}}, \text{op}]^{\alpha} \geq |\{\text{op}' \in \mathbb{O} : \bar{b}[\text{op}'.\text{Pre}]\}|. \quad (2)$$

where $\alpha < 1$ and coefficient k are hyperparameters. In words, the branching factor of a lifted operator expands as a function of the number of times a lifted operator has been sampled.

E. Learning UConds from controller feedback.

There are many cases where it is not obvious ahead of time what belief state propositions $\psi \in \Psi_{\mathcal{B}}$ affect the outcome distribution of a controller, making it difficult to construct an appropriate UCond set until simulations are run and it becomes evident what aspects of the environment are relevant to the controller outcome. For instance, simulators often know when a controller failed due to the robot colliding with a particular object, and can indicate that a proposition describing the position of this object ought to be added to the UCond set. Our full TAMPURA implementation allows controller simulation (Alg. 2, Line 29) to additionally return a set of propositions UCond+ which TAMPURA immediately adds to the UCond set of the operator being simulated. This modification does not increase the ability of TAMPURA to find correct plans in the limit (as one could conservatively start with overly large UCond sets), but can greatly increase the algorithm's efficiency.

VI. SIMULATED EXPERIMENTS & ANALYSIS

We applied TAMPURA to five simulated and two real-world robotics problems, illustrated in Figure 2 and Figure 1,

Model Learning	Decision Making	A	B	C	D	E-MF	E-M
Bayes Optimistic	LAO*	**0.87 ± 0.01**	**0.66 ± 0.07**	**0.63 ± 0.07**	**0.52 ± 0.11**	**0.95 ± 0.00**	**0.81 ± 0.02**
Bayes Optimistic	MLO	0.66 ± 0.09	**0.65 ± 0.07**	0.27 ± 0.10	0.29 ± 0.10	**0.95 ± 0.00**	0.41 ± 0.09
Bayes Optimistic	WAO	**0.78 ± 0.06**	**0.70 ± 0.07**	0.32 ± 0.10	0.24 ± 0.09	**0.95 ± 0.00**	0.56 ± 0.08
ϵ-greedy	LAO*	0.69 ± 0.08	0.58 ± 0.07	0.45 ± 0.10	**0.42 ± 0.10**	0.93 ± 0.00	**0.74 ± 0.06**
None	LAO*	0.00 ± 0.00	0.13 ± 0.04	0.20 ± 0.09	0.34 ± 0.09	**0.95 ± 0.00**	0.00 ± 0.00
Q-Learning	Q-Learning	0.42 ± 0.08	0.00 ± 0.00	0.20 ± 0.08	0.34 ± 0.10	**0.93 ± 0.04**	0.00 ± 0.00
MCTS	MCTS	0.00 ± 0.00	0.00 ± 0.00	0.04 ± 0.05	0.24 ± 0.09	**0.92 ± 0.01**	0.03 ± 0.03
DQN	DQN	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.47 ± 0.09**	0.55 ± 0.10	0.00 ± 0.00

TABLE I: Average and standard error of discounted return ($\gamma = 0.98$) for various model learning and decision-making strategies on the tasks in Figure 2. Our TAMPURA algorithm is in the top row. We bold all scores within a 75% confidence interval (N=20) of the top performing approach for each task. Solution times for each method and environment are reported in the Appendix; all solvers required comparable CPU time of 20-200 seconds, depending on the task (See Table II).

respectively. In our simulated experiments, we compared the performance across this range of tasks to Monte Carlo tree search and reinforcement learning baselines, as well as to belief-space task and motion planning algorithms without efficient model-learning and without uncertainty-aware planning (Table I). This section provides a brief overview of the simulated environments and baselines, with more details in Appendix C and D. All simulated robot experiments are performed in the pybullet physics simulation [38]. Grasps are sampled using the mesh-based EMA grasp sampler proposed in [7], inverse kinematics and motion planning are performed with tools from the pybullet planning library [39].

A. Simulated Domains

The CLASS UNCERTAINTY (A) task requires a robot to place all objects of a certain class in a bowl, despite noisy classifications (as occur when using detectors like MaskR-CNN [40]). The POSE UNCERTAINTY (B) task requires the robot to stack objects with local pose uncertainty, as arises when using standard pose estimation techniques from RGB-Depth video. The PARTIAL OBSERVABILITY (C) task requires the robot to find and pick up a hidden object in the scene. The PHYSICAL UNCERTAINTY (D) requires the robot to hit a puck with unknown friction parameters to a goal region. The SLAM UNCERTAINTY (E-M) task is a 2D version of a mobile manipulation task, in which a robot must bring yellow blocks in the environment to a goal region, with ego-pose uncertainty increasing over time, except when the robot visits a blue localization beacon (similarly to mobile robots using AR tags to localize). Manipulation-free SLAM variant (E-MF) just requires the agent to move to a goal region without interacting with blocks; this only requires 1-2 controller executions and was used to verify correctness of the baselines' implementations. This suite of tasks tests the planner's competence in a range of scenarios, including planning with risk awareness, planning to gather information.

B. Baselines

In our simulated experiments (Table I), we compare to many baselines from across the literature on sample-based POMDPs, reinforcement learning, and belief-space TAMP. Because our approach relies on closed-loop belief-space controllers, we are unable to fairly compare to point-based POMDP solvers that

plan in the state space [13, 14]. Instead we elect to compare to a variant of these planners that performs the MCTS in belief space [37, 41]. We also compare TAMPURA to ablations resembling the limitations of previous belief-space TAMP methods. TAMPURA uses the proposed Bayes optimistic model learning strategy, and the LAO* [10] probabilistic planner to solve the resulting BSSP problem. We compare to other non-probabilistic decision-making strategies commonly used in belief-space TAMP such as weighted all outcomes (WAO) [19] and maximum likelihood observation (MLO) determinized planning [20, 21]. Additionally, we compare to different model learning strategies including epsilon-greedy exploration described in Section V-A and contingent belief-space TAMP [18], which corresponds to performing no model learning and positing an effective uniform distribution over possible observations.

Our experiments show that Bayes optimistic model learning with full probabilistic decision making is best of these methods across this set of tasks. While determinized planning in belief space is sufficient for some domains like POSE UNCERTAINTY where most failures can be recovered from and the observational branching factor is binary, it performs poorly in domains with irreversible outcomes and higher observational branching factors. Our experiments verify that relative to Bayesian Optimistic model learning, ϵ-greedy frequently falls into local minima that it struggles to escape through random exploration in the action space. We observed that contingent planning frequently proposes actions that are geometrically or kinematically implausible, or inefficient because it does not consider the probabilities associated with different belief-space outcomes. For example, in the PARTIAL OBSERVABILITY domain, we saw the robot look behind and pick up objects with equal probability rather than prioritizing large occluders. Finally, MCTS and DQN performed poorly in most domains because they do not use high-level symbolic planning to guide their search. Without dense reward feedback, direct search in the action space is not sample efficient. The exception to this (other than SLAM-MF, used to verify implementation correctness) is PHYSICAL UNCERTAINTY, where the time horizon is short, with optimal plans only requiring 1-4 controller executions.
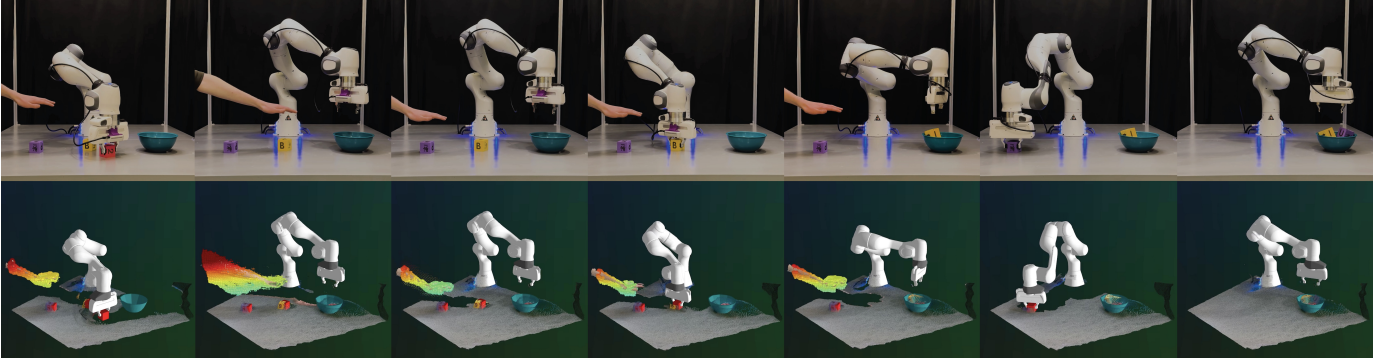
Fig. 5: TAMPURA moving cubes into a bowl without hitting a human in the workspace. Top row: images of robot execution. Bottom row: the robot's belief about object poses and the probabilistic occupancy grid describing the human in the workspace.

VII. REAL-WORLD IMPLEMENTATION

We implemented TAMPURA on a Franka robot arm to solve two tasks involving partial observability and safety with human interaction. Our robot experiments use Realsense D415 RGB-D cameras with known intrinsics and extrinsics. We use Bayes3D perception framework for probabilistic pose inference [29]. In our experiments we used objects with known mesh object models, but Bayes3D also supports few-shot online learning of object models. See the supplementary material for videos of successful completions under various initializations of these tasks.

A. Searching for Objects in Clutter

This task is the real-world counterpart to the PARTIAL OBSERVABILITY simulated experiment. In this task, the robot is equipped with a single RGBD camera mounted to the gripper, and must find and pick up a small cube hidden in the environment. This requires looking around the environment, and potentially moving other objects out of the way to make room to see and grasp the cube. Using Bayes3D's capacity to not only estimate poses of visible objects, but represent full posterior distributions over the latent scene given RGBD images, TAMPURA can characterize the probability that an unseen object is hidden behind each visible object. We experimented with various object sets and arrangements, and observed qualitatively sensible plans. For instance, TAMPURA moved larger objects with a larger probability of hiding the cube before moving smaller objects aside. The primary failure modes were (1) failure in perception (due, we believe, to improperly calibrated hard-coded camera poses), and (2) issues with tension in the unmodelled cord connected to the camera. Planning sometimes failed due to insufficient grasp and camera perspective sampling, which could be resolved by increasing maximum number of samples.

B. Safety in Human-Robot Interaction

In this task, several cubes and a bowl are placed on a table. The robot's task is to move these cubes into the bowl without colliding with a human's hand moving around in the workspace. The robot's belief states consist of a posterior over static object poses returned by Bayes3D, and a probabilistic 3D occupancy grid representing knowledge about dynamic elements in the scene (namely, the human). We update the occupancy belief probabilities over time as follows. Let $P(t, x, y, z)$ be the probability that the voxel at coordinates (x, y, z) was occupied at time t. The updated probability $P(t + \Delta t, x, y, z)$ at time $t + \Delta t$ is given by

$$P(t + \Delta t, x, y, z) = P(t, x, y, z) \times \gamma^{(C\Delta t)} \qquad (3)$$

where γ is the decay rate constant, C is the decay coefficient, and Δt is the time step. At each time t, the current frame of RGBD video was processed to obtain a point cloud, and each voxel occupied by a point had its occupancy probability reset to 1. A visualization of this grid can be seen in Figure 1. Given the current probabilistic occupancy grid, generated from point cloud data from RGB-Depth cameras, we approximate a motion planning path with gripper interpolation and calculate collision probabilities by integrating grid cell occupancy probabilities along the trajectory. For details, see Appendix D2. The resulting planner is able to make high-level decisions about safety and human avoidance. The supplemental video contains examples of the planner picking objects in an order that has the lowest probability of collision and waiting for the human to clear from the workspace before reaching for objects.

VIII. DISCUSSION

The TAMP framework enables zero-shot generalization to novel objects, scenes, configurations, and tasks, but typically relies on assumptions of full-observability and deterministic outcomes. In this work, we presented a TAMP method capable of reasoning about uncertainty and risk at both the task and motion planning levels to act efficiently in arbitrary partially observable environments. We demonstrated that resulting planner produces efficient risk and uncertainty aware plans across a range of real and simulated robotic tasks. Our approach enables long-horizon robot planning without precise descriptions of the effects of low-level controllers, lending itself to arbitrary learned or designed controllers.

Despite these novelties, TAMPURA, and TAMP in general, have several limitations. First, planning time can be burdensome when used in the context of real-time systems. Recent
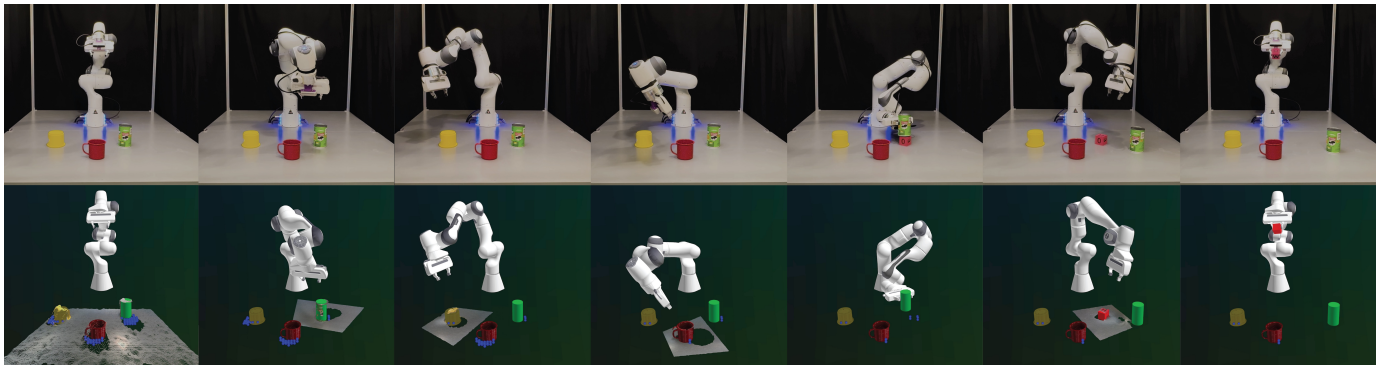
Fig. 6: TAMPURA searching a workspace to find and pick up a cube, looking around and moving objects to find it. Top: images of robot execution. Bottom: the robot's belief about the location of the target object over time. Each blue point in the robot's belief visualization is the centroid of a possible object location in the posterior returned by Bayes3D. Since the object models are known, the robot knows that the target object could be under the green cup or yellow cups with low probability. Because the yellow cup is too large to be grasped, the robot looks under the green cup after ruling out other possible locations.

developments in faster motion planning [42] or GPU-based parallel simulation [43] could ease this burden. Second, this planner requires user-provided belief representations, abstractions, and belief updating functionality. Designing each of these components can take considerable engineering effort, and may require expert knowledge in perception, inference, planning, and control. We believe extending this framework to handle learned abstractions and probabilistic models is an important direction for future research.

IX. ACKNOWLEDGEMENTS

REFERENCES

[1] Angelos Filos, Panagiotis Tigas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? *CoRR*, abs/2006.14911, 2020. URL https://arxiv.org/abs/2006.14911.

[2] Daniel S. Brown, Scott Niekum, and Marek Petrik. Bayesian robust optimization for imitation learning. *CoRR*, abs/2007.12315, 2020. URL https://arxiv.org/abs/2007.12315.

[3] Annie Xie, Shagun Sodhani, Chelsea Finn, Joelle Pineau, and Amy Zhang. Robust policy learning over multiple uncertainty sets, 2022.

[4] Jingda Wu, Zhiyu Huang, and Chen Lv. Uncertainty-aware model-based reinforcement learning with application to autonomous driving. *CoRR*, abs/2106.12194, 2021. URL https://arxiv.org/abs/2106.12194.

[5] Tao Pang, H. J. Terry Suh, Lujie Yang, and Russ Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models, 2023.

[6] Florian Wirnshofer, Philipp S. Schmitt, Georg von Wichert, and Wolfram Burgard. Controlling contact-rich manipulation under partial observability. In *Robotics: Science and Systems*, 2020. URL https://api.semanticscholar.org/CorpusID:220069704.

[7] Aidan Curtis, Xiaolin Fang, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Caelan Reed Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. *CoRR*, abs/2108.04145, 2021. URL https://arxiv.org/abs/2108.04145.

[8] Nishanth Kumar, Willie McClinton, Rohan Chitnis, Tom Silver, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning efficient abstract planning models that choose what to predict, 2023.

[9] Zhutian Yang, Caelan Reed Garrett, Tomás Lozano-Pérez, Leslie Kaelbling, and Dieter Fox. Sequence-based plan feasibility prediction for efficient task and motion planning, 2023.

[10] Eric A. Hansen and Shlomo Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1):35–62, 2001. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(01)00106-0. URL https://www.sciencedirect.com/science/article/pii/S0004370201001060.

[11] Tom Silver, Rohan Chitnis, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. *CoRR*, abs/2103.00589, 2021. URL https://arxiv.org/abs/2103.00589.

[12] Lionel Wong, Gabriel Grand, Alexander K. Lew, Noah D. Goodman, Vikash K. Mansinghka, Jacob Andreas, and Joshua B. Tenenbaum. From word models to world models: Translating from natural language to the probabilistic language of thought, 2023.

[13] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[14] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: online POMDP planning with regularization. *CoRR*, abs/1609.03250, 2016. URL http://arxiv.org/abs/1609.03250.

[15] Caelan Reed Garrett, Rohan Chitnis, Rachel M. Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *CoRR*, abs/2010.01083, 2020. URL https://arxiv.org/abs/2010.01083.

[16] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Stripstream: Integrating symbolic planners and blackbox samplers. *CoRR*, abs/1802.08705, 2018. URL http://arxiv.org/abs/1802.08705.

[17] Naman Shah and Siddharth Srivastava. Anytime integrated task and motion policies for stochastic environments. *CoRR*, abs/1904.13006, 2019. URL http://arxiv.org/abs/1904.13006.

[18] Aliakbar Akbari, Mohammed Diab, and Jan Rosell. Contingent task and motion planning under uncertainty for human–robot interactions. *Applied Sciences*, 10(5), 2020. ISSN 2076-3417. doi: 10.3390/app10051665. URL https://www.mdpi.com/2076-3417/10/5/1665.

[19] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. *CoRR*, abs/1911.04577, 2019. URL http://arxiv.org/abs/1911.04577.

[20] Leslie Kaelbling and Tomas Lozano-Perez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32:1194–1227, 08 2013. doi: 10.1177/0278364913484072.

[21] Dylan Hadfield-Menell, Edward Groshev, Rohan Chitnis, and Pieter Abbeel. Modular task and motion planning in belief space. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4991–4998, 2015. doi: 10.1109/IROS.2015.7354079.

[22] Edward Sondik. The optimal control of partially observable markov process over the infinite horizon: Discounted costs. *Operations Research*, 26:282–304, 04 1978. doi: 10.1287/opre.26.2.282.

[23] Nicolas Chopin, Omiros Papaspiliopoulos, et al. *An introduction to sequential Monte Carlo*, volume 4. Springer, 2020.

[24] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[25] Marco F Cusumano-Towner, Feras A Saad, Alexander K Lew, and Vikash K Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th acm sigplan conference on programming language design and implementation*, pages 221–236, 2019.

[26] Alexander K Lew, Matin Ghavamizadeh, Martin C Rinard, and Vikash K Mansinghka. Probabilistic programming with stochastic probabilities. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1708–1732, 2023.

[27] Alexander K Lew, George Matheos, Tan Zhi-Xuan, Matin Ghavamizadeh, Nishad Gothoskar, Stuart Russell, and Vikash K Mansinghka. Smcp3: Sequential monte carlo with probabilistic program proposals. In *International Conference on Artificial Intelligence and Statistics*, pages 7061–7088. PMLR, 2023.

[28] Marco Cusumano-Towner, Alexander K Lew, and Vikash K Mansinghka. Automating involutive mcmc using probabilistic and differentiable programming. *arXiv preprint arXiv:2007.09871*, 2020.

[29] Nishad Gothoskar, Matin Ghavami, Eric Li, Aidan Curtis, Michael Noseworthy, Karen Chung, Brian Patton, William T Freeman, Joshua B Tenenbaum, Mirko Klukas, et al. Bayes3d: fast learning and inference in structured generative models of 3d objects and scenes. *arXiv preprint arXiv:2312.08715*, 2023.

[30] Nishad Gothoskar, Marco Cusumano-Towner, Ben Zinberg, Matin Ghavamizadeh, Falk Pollok, Austin Garrett, Josh Tenenbaum, Dan Gutfreund, and Vikash Mansinghka. 3dp3: 3d scene perception via probabilistic programming. *Advances in Neural Information Processing Systems*, 34:9600–9612, 2021.

[31] Guangyao Zhou, Nishad Gothoskar, Lirui Wang, Joshua B Tenenbaum, Dan Gutfreund, Miguel Lázaro-Gredilla, Dileep George, and Vikash K Mansinghka. 3d neural embedding likelihood: Probabilistic inverse graphics for robust 6d pose estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 21625–21636, 2023.

[32] Håkan L. S. Younes and Michael L. Littman. Ppddl 1 . 0 : An extension to pddl for expressing planning domains with probabilistic effects. 2004. URL https://api.semanticscholar.org/CorpusID:2767666.

[33] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(02)00374-0. URL https://www.sciencedirect.com/science/article/pii/S0004370202003740. Planning with Uncertainty and Incomplete Information.

[34] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *International Conference on Automated Planning and*

Scheduling, 2007. URL https://api.semanticscholar.org/CorpusID:15013602.

[35] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. On bayesian upper confidence bounds for bandit problems. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 592–600, La Palma, Canary Islands, 21–23 Apr 2012. PMLR. URL https://proceedings.mlr.press/v22/kaufmann12.html.

[36] David Speck. *Symbolic Search for Optimal Planning with Expressive Extensions*. PhD thesis, University of Freiburg, 2022.

[37] Zachary Sunberg and Mykel J. Kochenderfer. POM-CPOW: an online algorithm for pomdps with continuous state, action, and observation spaces. *CoRR*, abs/1709.06196, 2017. URL http://arxiv.org/abs/1709.06196.

[38] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[39] Caelan Reed Garrett. PyBullet Planning. https://pypi.org/project/pybullet-planning/, 2018.

[40] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870.

[41] Aidan Curtis, Leslie Kaelbling, and Siddarth Jain. Task-directed exploration in continuous pomdps for robotic manipulation of articulated objects, 2022.

[42] Wil Thomason, Zachary Kingston, and Lydia E. Kavraki. Motions in microseconds via vectorized sampling-based planning. In *IEEE International Conference on Robotics and Automation*, 2024.

[43] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[44] Albert Wu, Thomas Lew, Kiril Solovey, Edward Schmerling, and Marco Pavone. Robust-rrt: Probabilistically-complete motion planning for uncertain nonlinear systems, 2022.

[45] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

[46] Malte Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246, jul 2006. ISSN 1076-9757.

[47] David Speck, Robert Mattmüller, and Bernhard Nebel. Symbolic top-k planning. In Vincent Conitzer and Fei Sha, editors, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 9967–9974. AAAI Press, 2020.

[48] Aidan Curtis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Discovering state and action abstractions for generalized task and motion planning. *CoRR*, abs/2109.11082, 2021. URL https://arxiv.org/abs/2109.11082.

[49] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10):12120–12129, Jun. 2023. doi: 10.1609/aaai.v37i10.26429. URL https://ojs.aaai.org/index.php/AAAI/article/view/26429.

A. Code release

Our TAMPURA implementation, as well as full implementations of our simulated experiments (including the environments, controllers, and stream specifications), will be released at a public repository.

B. Compiling simulation outcome counts into the sparse abstract MDP

Line 36 of Algorithm 2 compiles the dictionaries N and D storing simulation counts into a learned transition distribution $\hat{\mathcal{T}}$ on a subset $\mathcal{B}_{\text{sparse}} \subseteq \overline{\mathcal{B}}$ of the set of all abstract beliefs. These components, $\mathcal{B}_{\text{sparse}}$ and $\hat{\mathcal{T}}$, define an MDP (which we refer to throughout as the "sparse MDP") that is passed into the LAO* probabilistic planner in Algorithm 1, for uncertainty and risk aware planning. We now elaborate on how $\mathcal{B}_{\text{sparse}}$ and $\hat{\mathcal{T}}$ are constructed.

The set $\mathcal{B}_{\text{sparse}}$ consists of all abstract belief states reachable from the current belief state b_0, by applying the guaranteed effects (Effs) of operators visited during model learning, and applying assignments of uncertain effects (UEffs) present in at least one simulation from model learning. The key set of D consists of values of the form $(\Psi_{\text{pre}}, c, \Psi_{\text{eff}})$, where Ψ_{pre} is an assignment to the UCond set of the operator corresponding to controller c, and Ψ_{eff} is an assignment to its UEffs. D thereby stores the set of all operators which were simulated during model learning (as each controller c corresponds to a particular operator), and all UEff assignments produced in model learning simulations.

The transition probabilities $\hat{\mathcal{T}}$ are as follows. Consider any operator $\text{op} \in \mathbb{O}$, any abstract belief state $\bar{b}$, and any assignment Ψ_{eff} to op.UEffs. Let $\bar{b}'$ is the abstract belief state obtained by beginning in $\bar{b}$ and applying each effect in op.Effs, as well as each effect in op.UEffs marked as true in Ψ_{eff}. Let Ψ_{pre} denote the assignment to op.UConds in $\bar{b}$. Then the transition probability is the fraction of simulations of $\text{op.}c$ run in model learning from Ψ_{pre} that resulted in assignment Ψ_{eff}:

$$\hat{\mathcal{T}}(\bar{b}' \mid \bar{b}, \text{op.}c) := \frac{D[\Psi_{\text{pre}}, \text{op.}c, \Psi_{\text{eff}}]}{N[\Psi_{\text{pre}}, \text{op.}c]}$$

For $(\bar{b}, \text{op}, \bar{b}')$ where the resulting pair $(\Psi_{\text{pre}}, \text{op})$ was explore during model learning, but never produced UEffs matching $\bar{b}'$, $\hat{\mathcal{T}}(\bar{b}' \mid \bar{b}, \text{op.}c) := 0$. In the case that the pair $(\Psi_{\text{pre}}, \text{op})$ was never explored during model learning, we do not even add an entry for $(\bar{b}, \text{op}, \bar{b}')$ to the data structure representing $\hat{\mathcal{T}}$. The fact that $\hat{\mathcal{T}}$ does not contain any entries beginning with $(\bar{b}, \text{op})$ represents to the probabilistic planner that in belief state $\bar{b}$, operator op cannot be appliex and should not even be considered (as it was never visited during model learning in an abstract belief state with UConds matching $\bar{b}$). This is important to performant planning by LAO*, as it ensures that there are only a relatively small number of operators applicable from each abstract belief space $\bar{b}$. It is in this sense that the MDP given to LAO* is sparse; we believe this sparsity plays a key role in the tractability of solving the MDP given to LAO*.

In many cases, the set $\mathcal{B}_{\text{sparse}}$ can be constructed explicitly by initializing $\mathcal{B}_{\text{sparse}}$ the set $\{\bar{b}_0\}$ just containing the initial abstract belief state, and then iteratively adding all abstract belief states that would result from applications of explored operators to the states currently in $\mathcal{B}_{\text{sparse}}$. (In fact, we implemented this and used it to produce the results in Figure 4. These results value iteration rather than LAO* to solve the sparse MDP, to ensure the comparison targeted the quality of the learned transition model without effects related to the interaction with an approximate MDP solver like LAO*. Value iteration requires explicit representation of the MDP state space $\mathcal{B}_{\text{sparse}}$.) However, the full TAMPURA implementation never explicitly constructs $\mathcal{B}_{\text{sparse}}$. Instead, it gives LAO* the sparse MDP in the form of data structures which, for any $\bar{b} \in \mathcal{B}_{\text{sparse}}$, can list the operators which can be applied in $\bar{b}$, and the distribution over possible outcome abstract belief states $\bar{b}'$ induced by applying each operator.

It is the sparsity of the action branching factor that makes the sparse MDP tractable solve, not the small size of the state space. (Indeed, $\mathcal{B}_{\text{sparse}}$ can be large enough we found it desirable not to have to construct it explicitly.) The ability to learn a transition model on a relatively large set of abstract belief states, but also produce efficient probabilistic plans in the resluting MDP due to its action sparsity, is a key feature of TAMPURA's approach. (One benefit of having $\mathcal{B}_{\text{sparse}}$ cover more states is that it decreases the frequency of replanning.) The ability to learn a transition model on all of $\mathcal{B}_{\text{sparse}}$ derives from learning probability tables from (UConds, op) pairs to distributions over UEffs, rather than directly learning transition probabilities from each pair $(\bar{b}, \text{op})$ to a distribution on resulting abstract belief states. (Each UCond and UEff assignment is consistent with many abstract belief states, so this learning representation is much more efficient.)

C. Extended Task Descriptions

1) CLASS UNCERTAINTY: This task considers a robot arm mounted to a table with a set of 2 to 10 objects placed in front of it, with at least one bowl in the scene. The robot must place all objects of a certain class within the bowl without dropping any objects. We add classification noise to ground truth labels to mimic the confidence scores typically returned by object detection networks like MaskRCNN [40]. Object grasps have an unknown probability of success, which can be determined through simulations during planning. The agent can become more certain about an object category by inspecting the object more closely with a wrist mounted camera. A reasonable strategy is to closely inspect objects that are likely members of the target class and stably grasp and place them in the bowl. The planner has access to the following controllers: `Pick(?o ?g ?r)`, `Drop(?o ?g ?r)`, `Inspect(?o)`, for objects o, grasps g, and regions on the table r.

2) POSE UNCERTAINTY: This task consists of 3 cubes placed on the surface of a table and a hook object with known pose. The cubes have small Gaussian pose uncertainty

in the initial belief, similar to what may arise when using standard pose estimation techniques on noisy RGBD images. The goal is to stack the cubes with no wrist-mounted camera. A reasonable strategy is to use the hook to bring the objects into reach or reduce the pose uncertainty by aligning the object into the corner of the hook such that grasping and stacking success probability is higher. The planner has access to controllers `Pick(?o, ?g)`, `Place(?o, ?g, ?p, ?r)`, `Stack(?o1, ?g, ?o2)`, and `Pull(?o1, ?g, ?o2)`, for an physical objects o, o_1, o_2, grasps g, regions on the table r, and 3D pose p. (`Pull` pulls one object using another object.)

3) PARTIAL OBSERVABILITY: This task, the agent has 2 to 10 objects placed in front of it with exactly one die hidden somewhere in the scene such that it is not directly visible. The goal is to be holding the die without dropping any objects. The robot must look around the scene for the object, and may need to manipulate non-target objects under certain kinematic, geometric, or visibility constraints. The planner has access to `Pick(?o, ?g)`, `Place(?o, ?g)`, `Look(?o, ?q)`, and `Move(?q)` controllers for this task.

4) PHYSICAL UNCERTAINTY: This task consists of a single puck placed on a shuffleboard in front of the robot. The puck has a friction value drawn from a uniform distribution. The goal is to push the puck to a target region on the shuffleboard. The robot can attempt pushing the puck directly to the goal, but uncertainty in the puck friction leads to a low success rate. A more successful strategy is to push the puck around locally while maintaining reach to gather information about its friction before attempting to push to the target. The planner has access to a `PushTo(?o, ?r)` controller that pushes object o to a target region r and `PushDir(?o, ?d)` controller that pushes object o with fixed velocity in a target direction d, both of which are implemented as velocity control in Cartesian end-effector space.

5) SLAM UNCERTAINTY: The task is a 2D version of a mobile manipulation task, where the robot must gather yellow blocks and bring them to a green region. The number, location, and shape of the objects and obstacles is randomly initialized along with the starting location of the robot. The initial state is fully known, but the robot becomes more uncertain in its position over time due to action noise. The robot can localize itself at beacons similar to the way many real-world base robots use AR tags for localization. To verify that all baselines were implemented correctly, we also consider a manipulation-free variant of this task (SLAM-MF) requiring only 1 or 2 controller executions for success. The goal is to enter the target region without colliding with obstacles; blocks need not be moved. The planner has access to `MoveRegion(?r)`, `MoveLook(?r)` (which moves to region r and then localizes itself by looking at a beacon0, `MovePick(?r, ?o)`, `MovePlace(?r, ?o)`, and `MoveCorner(?r, ?c)` controllers. All moving controllers use a belief-space motion planner [44] except for `MoveCorner(?r, ?c)` , which simply navigates to a particular the corner of a workspace.

D. Experimental Details

All experiments were run on a single Intel Xeon Gold 6248 processor with 9 GB of memory. We report planning times for each algorithm and environment combination in table II. It is important to note that we did not optimize for planning time, and all of these algorithms run in an anytime fashion, meaning that planning can be terminated earlier with lower success rates.

We now provide several more details about the two tasks we performed using TAMPURA on the real robot.

1) Object finding: The robot has access to a number of controllers that it could use to find and hold a small cube. A `Pick(?o, ?g)` controller will grasp an object o with grasp g, if the variance of the object pose is below some threshold. A `Place(?o, ?g)` controller places an object o held at grasp g assuming the probability of collision of that placement is below some threshold. Lastly, a `Look(?q)` controller moves the robot arm to a particular joint configuration q and captures an RGBD image.

2) HRI: The collision probability for each trajectory segment used in the motion model is determined as follows:

$$P_{\text{collision}} = 1 - \prod_{t=1}^{T} \prod_{i=1}^{n_t} (1 - P(t, x_i, y_i, z_i)) \qquad (4)$$

Here, T represents the total number of time steps in the trajectory, n_t is the number of cells encountered at time step t, and (x_i, y_i, z_i) are the coordinates of the i-th cell at time t along the trajectory.

The robot has access to `Pick(?o, ?g)`, `Place(?o, ?g)` and `Wait()` controllers, for objects o and grasps g.

E. Additional Baseline Details

1) DQN: In our DQN baseline, we use the CleanRL [45] implementation. The state space of DQN is a vectorized version of the belief. To vectorize beliefs, we flatten all continuous properties of the belief, and one-hot encode all discrete properties. The action space is a pre-discretized version of the original continuous action space. We perform this discretization by sampling three possible continuous samples from each sampler described in Appendix G. To make for a fair comparison with other methods, we limit the DQN training to 1000 simulator samples per action. At each simulator step after sufficient data exists for a single batch, we update the network. We find that after approximately 1000 such network updates, the loss converges. Data is retained in the buffer across execution steps.

We chose to implement DQN in an online fashion (i.e. simulations performed from the initial belief state) instead of in an offline fashion, which is the typical application of RL. In an offline setting, the RL agent would be trained on a distribution of possible environments. We did not attempt to compare to RL in this way because we are interested in testing zero-shot performance on novel problems that are out of distribution for an RL agent trained on a particular

Model Learning	Decision Making	Task A	Task B	Task C	Task D	Task E-MF	Task E-M
Bayes Optimistic	LAO*	28 ± 26	21 ± 13	57 ± 38	23 ± 7	31 ± 11	129 ± 55
Bayes Optimistic	MLO	54 ± 3	38 ± 20	49 ± 43	30 ± 41	35 ± 24	90 ± 56
Bayes Optimistic	WAO	33 ± 24	29 ± 20	87 ± 32	46 ± 41	34 ± 15	100 ± 52
ϵ-greedy	LAO*	3 ± 0	40 ± 34	72 ± 38	27 ± 15	35 ± 17	110 ± 40
None	LAO*	1 ± 0	15 ± 6	3 ± 2	1 ± 0	1 ± 0	1 ± 0
Q-Learning	Q-Learning	10 ± 3	181 ± 24	88 ± 54	11 ± 11	72 ± 31	186 ± 89
MCTS	MCTS	29 ± 29	60 ± 10	54 ± 51	16 ± 7	169 ± 13	207 ± 28
DQN	DQN	17 ± 4	12 ± 3	83 ± 34	72 ± 29	28 ± 4	84 ± 4

TABLE II: Average and standard deviation of per-step planning times (seconds) averaged over trials and steps within each trial. These include execution time of the selected controller in simulation.

distribution of tasks. Testing out-of-distribution generalization of an RL agent trained on offline data would require a separate experimental setup, and is outside of the scope of this paper.

2) MCTS: Our MCTS implementation uses a pre-discretized action space and plans in the abstract belief space. We make selections according to the standard UCB selection criterion $UCB = \bar{X}_j + C\sqrt{\frac{2 \ln n}{n_j}}$ where $\bar{X}_j$ is the average reward obtained from node j, n is the total number of simulations that have been run from the parent node, n_j is the number of simulations that have been run from the child node j, and C is the constant determining the trade-off between exploration and exploitation. We use $C = 1$ in our experiments.

3) Q-Learning: We perform Q learning on a pre-discretized action space and the abstract belief space. A sparse Q table is maintained due to the intractably large abstract belief space. During each simulation, the sparse Q table is updated according to the following rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

In our experiments we use $\alpha = 0.2$, and we take a random action with an $\epsilon = 0.2$. The ultimate action is selected via $\text{argmax}_a[Q(\bar{b}_0, \cdot)]$.

F. Additional Experimental Statistics

In Table III we provide some additional statistics on our simulated experiments for the TAMPURA algorithm. These statistics include the average number of abstract belief states visited during model learning on the first execution step, and the average number of controllers TAMPURA executed on the robot in order to achieve the goal.

Task Name	# Visited $\bar{b}$	# Executed steps
Task A	118.06 ± 14.12	6.44 ± 1.21
Task B	27.81 ± 10.53	9.69 ± 0.68
Task C	17.40 ± 5.43	5.40 ± 2.33
Task D	5.00 ± 0.00	3.75 ± 1.79
Task E-MF	50.70 ± 9.26	1.20 ± 0.51
Task E-M	68.67 ± 26.56	9.22 ± 3.84

TABLE III: Additional Simulated Experimental Statistics

G. Continuous Action Parameter Samplers

Continuous action parameters such as force vectors, grasps, and joint configurations are sampled from during the model learning process. Such samples are often conditioned on other elements of the action input. For example, a grasp is specific to a particular object. Likewise, an inverse kinematics solution is specific to a particular grasp, object, and object pose. These relationships are expressed as streams, as in PDDL-Stream [16]. Each stream is associated with generator capable of outputting an infinite stream of samples. An example of such streams for sampling object grasps, placement poses, and inverse kinematics solutions are shown below.

```
(:stream sample-grasp
 :parameters (?o - obj)
 :domain (and (Graspable ?o))
 :output (?g - grasp)
 :certified (and (Grasp ?o ?g))
```

```
(:stream sample-placement
 :parameters (?o - obj ?s - surface)
 :output (?p - pose)
 :certified (and (Pose ?o ?p) (Support ?p ?r))
```

```
(:stream sample-ik
 :parameters (?o - obj ?g - grasp ?p - pose)
 :domain (and (Grasp ?o) (Pose ?o ?p))
 :output (?q - conf)
 :certified (and (IKSol ?o ?g ?p ?q))
```

The atoms in the domain and certified of each stream are nonfluents, which means they cannot change during planning and thus do not exist in the effects of actions. These nonfluents are referenced in the preconditions of actions to enforce relationships between input objects.

H. Planner Hyperparameter Details

In all of our simulated experiments we set a maximum of $K = 1000$ simulated controller executions per real execution step, kept constant across baselines. We use an MDP γ value of 0.98 during planning and for our evaluation metric. During model learning, we query the Top-K symbolic planner with a batch size of 20 symbolic plans. For baselines using progressive widening, we use $k = 3$ and $\alpha = 0.2$. Lastly, we allow all methods to run for a maximum of 20 environment steps before the planner is terminated with a failure result.

I. Fast Downward Planning Details

Our algorithm uses Fast Downward [46] planner to solve deterministic planning problems during model learning. The input to Fast Downward is a problem file describing the initially true atoms in PDDL and a domain model describing the deterministic transition model in terms of a set of action schema. All-outcome determinization of the stochastic transition dynamics described in Section IV-C to standard deterministic transition dynamics is done by creating a separate action for each possible outcome. Costs are then added for each outcome's estimated log probability of occurrence using state-dependent action costs [36].

We attain batches of determinized plans using a SymK [47], which is a Top-K planner built on Fast Downward. The batch size to query the planner with is a user-selected hyperparameter. We run FastDownward with A-star search using the Landmark Cut heuristic.

J. Stationarity of the abstract belief state MDP

The condition needed for $\overline{\mathcal{T}}$ to be well defined is that the that there exists a stationary probability kernel $P(b; \bar{b})$ describing the probability that the agent is in belief state b, given that the abstracted version of its belief state is $\bar{b}$. In this paper, we assume the abstractions resulting from the user-provided operators are stationary in this sense. Verifying this soundness property and learning sound abstractions, as is sometimes done in fully observable TAMP [48, 11, 49], is a valuable direction for future work.

We now formally define the stationarity condition needed for $P(b; \bar{b})$, and hence $\overline{\mathcal{T}}$, to be well defined. Let

$$\mathcal{B}_{\bar{b}} := \{b \in \mathcal{B} : \mathrm{abs}(b) = \bar{b}\}$$

For each operator c, let $\mathcal{T}(b' \mid b, c)$ denote the probability distribution on the belief state resulting from running the controller in c beginning from belief b.

For any $t \in \mathbb{N}$ and any sequence of operators $c_1, \ldots, c_t$, consider the probability distribution $P(b \mid c_1, \ldots, c_t)$ over the robot's belief state after applying the sequence of controllers $c_1, \ldots, c_t$:

$$P(b \mid c_1, \ldots, c_t) = \sum_{b_{t-1} \in \mathcal{B}} \sum_{b_{t-2} \in \mathcal{B}} \cdots \sum_{b_1 \in \mathcal{B}}$$
$$\mathcal{T}(b \mid b_{t-1}, c_t) \mathcal{T}(b_{t-1} \mid b_{t-2}, c_{t-1}) \ldots \mathcal{T}(b_1 \mid b_0, c_1) \quad (5)$$

We will define $P(b \mid \bar{b}, c_1, \ldots, c_t)$ to denote the conditional probability of being in belief state b after applying operator sequence $c_1, \ldots, c_t$, given that the abstract belief state corresponding to b is $\bar{b}$. For each $b \notin \mathcal{B}_{\bar{b}}$, $P(b \mid \bar{b}, c_1, \ldots, c_t) := 0$, and for $b \in \mathcal{B}_{\bar{b}}$,

$$P(b \mid \bar{b}, c_1, \ldots, c_t) := \frac{P(b \mid c_1, \ldots, c_t)}{\sum_{b' \in \mathcal{B}_{\bar{b}}} P(b' \mid c_1, \ldots, c_t)} \quad (6)$$

The abstract belief-state MDP is well defined so long as there exists some probability kernel $P(b; \bar{b})$ from $\overline{\mathcal{B}}$ to $\mathcal{B}$ such that for all $t \in \mathbb{N}$ and all operator sequences $c_1, \ldots, c_t$,

$$P(b \mid \bar{b}, c_1, \ldots, c_t) = P(b; \bar{b}) \quad (7)$$

That is, given an abstract belief state $\bar{b}$, the distribution over the concrete belief state corresponding to this is independent of the time elapsed in the environment and the controllers which have been executed so far.

We anticipate that in most applications of TAMPURA, the user-provided abstractions will be imperfect, and this stationarity property will not hold exactly. That is, the distribution $P(b \mid \bar{b}, c_1, \ldots, c_t)$ will vary in t and $c_1, \ldots, c_t$. The TAMPURA algorithm can still be run in such cases, as TAMPURA never computes $P(b; \bar{b})$ exactly, but instead approximates this by constructing a table of all $(b, \bar{b})$ pairs encountered in simulations it has run. In the case where $P(b; \bar{b})$ is not well defined, but there exists a bound on the divergence between any pair of distributions $P(b \mid \bar{b}, c_1, \ldots, c_t)$ and $P(b \mid \bar{b}, c'_1, \ldots, c'_{t'})$, we expect that TAMPURA can be understood as approximately solving an abstract belief state MDP whose transition function is built from any kernel $\tilde{P}(b; \bar{b})$ with bounded divergence from all the $P(b \mid \bar{b}, c_1, \ldots, c_t)$. We leave formal analysis of TAMPURA under boundedly nonstationary abstractions to future work.

MDP Components

$\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{Z}, \gamma$	(III)	State space, observation space, action space, observation function, and discount factor in the original POMDP.
$\mathcal{M}, \mathcal{M}_b, \mathcal{M}_c$	(III, III-A, III-C)	The original MDP, belief-state MDP, and belief-state controller MDP, respectively.
$\mathcal{T}, \mathcal{T}_b, \mathcal{T}_c$	(III, III-A, III-C)	State transition probabilities for original POMDP, belief-state MDP, and belief-state controller MDP, respectively.
r, r_b, r_c	(III, III-A, III-C)	Reward functions in the original POMDP, belief-state MDP, and belief-state controller MDP, respectively.
$\mathcal{B}$	(III-A)	The belief space.
b_0	(III-A)	Initial belief state in the planning problem.
$\mathcal{C}$	(III-C)	The space of controllers. Each controller $c \in \mathcal{C}$ is an eventually-terminating control policy that can be executed within the belief-state MDP.

Abstraction

$\overline{\mathcal{B}}$	(IV)	Abstract belief space, partitioning the continuous belief space into operationally similar groups based on belief state propositions.
$\overline{\mathcal{T}}$	(IV)	Abstract transition model giving probabilities $\overline{\mathcal{T}}(\bar{b}' \mid \bar{b}, c)$ of arriving in abstract belief $\bar{b}'$ after running controller c in abstract belief state $\bar{b}$. This is the transition model for the abstract belief-state MDP $\overline{\mathcal{M}}_c$.
$\overline{\mathcal{M}}_c$	(IV)	The abstract belief-state controller MDP.
$\mathcal{M}_s$	(IV)	The sparse abstract MDP we aim to learn and solve with a probabilistic planner
$\mathcal{B}_{\text{sparse}}$	(III)	The sparse belief space of the reduced mdp $\mathcal{M}_s$ that comes out of model learning
$\Psi_{\mathcal{B}}$	(IV-A)	Set of belief state propositions used to define the abstract belief states within the belief-state MDP.
abs	(IV-B)	A map from beliefs in $\mathcal{B}$ to abstract beliefs in $\overline{\mathcal{B}}$
$\bar{b}_0$	(IV-B)	Initial abstract belief state, derived from the initial concrete belief state through the application of belief state propositions.
G	(IV-B)	The set of abstract belief states that represents the goal of our planning problem.
$\mathbb{O}$	(IV-C)	Set of operations or actions available in the planning environment.
op	(IV-C)	An operator (an element of $\mathbb{O}$). This is a tuple of values $\langle \texttt{Pre}, \texttt{Eff}, \texttt{UEff}, \texttt{UCond}, c \rangle$. These values are sometimes denoted op.Pre, op.Eff, op.UEff, op.UCond, and op.c.

Model Learning

α, β	(V-B)	Parameters for Bayesian model learning prior.
$\mathcal{T}_{\text{AO}}$	(V-B)	The all-outcome determinized version of the abstract transition model, constructed by making all possible action/outcome combinations separate actions with deterministic outcomes.
J	(V-B)	A model-learning hyperparameter defining the cost of an action for a determinized version of a stochastic planning problem.

Algorithm Structures

I, K, S	(V-C)	Parameters controlling runtime: I is the total iterations, K is the number of trajectories, S is the number of simulations.
s	(V-C)	State from past iterations of model learning. This is initialized to $\emptyset$; after the first iteration of model learning it is a 3-tuple $(N, D, P_{\mathcal{B}})$.
N, D	(V-C)	Default dictionaries used for counting simulations performed during model learning. $N[\Psi_{\text{pre}}, c]$ is the number of times controller c was simulated from a belief state consistent with precondition set Ψ_{pre}. $D[\Psi_{\text{pre}}, c, \Psi_{\text{eff}}]$ is the number of times that controller c was simulated from a belief state consistent with precondition set Ψ_{pre}, and the resulting belief state was consistent with effect set ψ_{eff}.
$P_{\mathcal{B}}$	(V-C)	Mapping from abstract beliefs to corresponding concrete beliefs. This is a DefaultDict object. The keys are abstract beliefs, and the values are lists of concrete beliefs. The default value is the empty list.
τ_k	(V-C)	simulated controller execution trajectories during model learning consisting of a list of low-level action, observation pairs.
$\vec{\Psi}_{\text{pre}}, \vec{\Psi}_{\text{eff}}$	(V-C)	Vectors of preconditions and effects for each transition. Each Ψ_{pre} in $\vec{\Psi}_{\text{pre}}$ is a boolean assignment to each predicate in op.UCond, for some operator op. Each Ψ_{eff} is a boolean assignment to each predicate in op.UEff.
$\vec{c}$	(V-C)	List of controllers involved in transitions.
$\vec{s}, \vec{f}$	(V-C)	Vectors tracking successful and failed transitions, respectively.
$\vec{H}$	(V-C)	Vector of entropy values calculated to focus simulations on uncertain cases.
k, α	(V-D)	Hyperparameters in progressive widening, influencing the expansion rate of action space based on the sampling frequency of actions.

TABLE IV: Notation reference.