2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 1-5, 2023. Detroit, USA

On-Robot Bayesian Reinforcement Learning for POMDPs

Hai Nguyen[1*], Sammie Katt[1], Yuchen Xiao[1], Christopher Amato[1]

Abstract— **Robot learning is often difficult due to the expense of gathering data. The need for large amounts of data can, and should, be tackled with effective algorithms and leveraging expert information on robot dynamics. Bayesian reinforcement learning (BRL), thanks to its sample efficiency and ability to exploit prior knowledge, is uniquely positioned as such a solution method. Unfortunately, the application of BRL has been limited due to the difficulties of representing expert knowledge as well as solving the subsequent inference problem. This paper advances BRL for robotics by proposing a specialized framework for physical systems. In particular, we capture this knowledge in a factored representation, then demonstrate the posterior factorizes in a similar shape, and ultimately formalize the model in a Bayesian framework. We then introduce a sample-based online solution method, based on Monte-Carlo tree search and particle filtering, specialized to solve the resulting model. This approach can, for example, utilize typical low-level robot simulators and handle uncertainty over unknown dynamics of the environment. We empirically demonstrate its efficiency by performing on-robot learning in two human-robot interaction tasks with uncertainty about human behavior, achieving near-optimal performance after only a handful of real-world episodes. A video of learned policies is at `https://youtu.be/H9xp60ngOes`.**

I. INTRODUCTION

Mainstream reinforcement learning (RL) techniques [1]–[3] are not sample efficient enough for online applications in physical systems: long training hours or unguided exploration can wear out or break fragile, expensive robot systems. Instead, the common approach for policy learning in robotics is to use simulators, and transfer learned policies to the real hardware (*sim2real*). However, simulators cannot capture the real world exactly; therefore, additional techniques are required for successful transfers, such as online system identification [4] or domain randomization [5]. These approaches have shown success, but the research question is far from solved, and transfers fail when the sim2real gap is large. Moreover, none of these approaches can, in principle, exploit prior (expert) knowledge, which is useful for faster learning and often exists in most robotics problems. In our view, the ideal approach should learn directly on physical hardware (*on-robot*) while leveraging prior knowledge.

Bayesian Reinforcement Learning (BRL) has great potential for on-robot learning. BRL is sample-efficient and provides a principled solution to the exploration-exploitation trade-off by explicitly incorporating uncertainty into the decision-making process. Furthermore, BRL allows expert knowledge to be easily integrated into the learning process as "priors", allowing the agent to exploit knowledge it would otherwise have needed many precious samples to learn. Despite its promise, previous attempts at BRL for on-robot learning have been limited, even for the fully observable settings. This can be explained by the scaling issues of BRL methods and the difficulty of representing the complex expert knowledge available in physical systems. In particular, these methods typically assume straightforward priors, such as a single large Dirichlet table or neural networks [6], [7], but rarely provide tools to tackle more sophisticated applications.

This paper presents a Bayesian approach incorporating expert knowledge for efficient on-robot learning under partial observability. We first identify typical and reasonable assumptions in physical systems, including some inspired by mixed observability Markov decision processes [8], and translate them to a Bayesian setting. The next step derives the corresponding Bayesian inference problem, showing how the nature of the prior that we discovered shapes the posterior over the unknown quantities of the problem. This leads to the formalization of a Bayes-adaptive (BA) model specialized for robotics systems. Finally, we propose a method for solving this model by specializing Monte-Carlo tree search and particle filtering, which provides an efficient and principled solution to the original learning problem. We empirically demonstrate its efficiency with on-robot learning in two human-robot interaction tasks with uncertainty about human behavior, achieving near-optimal performance after only a handful of real-world episodes.

II. RELATED WORK

A. Bayesian RL under Partial Observability

Bayesian RL involves maintaining a probability distribution over an augmented state, including both the environment dynamics and the environment state, as proposed by the BA-POMDP framework [7]. State-of-the-art BRL methods [6], [9], [10] for partially observable domains have improved on the BA-POMDP by incorporating the Partially Observable Monte Carlo Planning (POMCP) [11], an efficient online planner for large POMDPs. While earlier methods [9], [10] are only applicable to small discrete POMDPs or ones with factorizable dynamics, Bayes-adaptive deep dropout RL (BADDr) [6] was introduced to tackle more complex domains. BADDr leverages the representation power of dropout networks [12] to capture priors, making BRL scalable to larger domains while being considerably more sample-efficient than pure RL methods. In this paper, we also use dropout networks to construct (parts of) a specialized dynamics model, leveraging assumptions about the factored dynamics and full observability of the robot state.

[1]Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA. *Corresponding author `nguyen.hai1@northeastern.edu`.

2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-9190-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IROS55552.2023.10342114

B. On-Robot Reinforcement Learning

Recently, many methods have performed learning directly on physical hardware. In [13], multiple robot workers are used to collect data for online training of a robot arm to open a door autonomously. [14] used human feedback to learn manipulation tasks in 1-4 hours. Meanwhile, [15] combined RL and contrastive learning [16] to learn manipulation tasks online from pixels. In addition, [17], [18] used symmetry-aware neural networks to encode domain symmetries to perform online learning manipulation tasks within several hours, while [19], [20] conducted online learning for locomotion tasks on legged robots. Nevertheless, unlike ours, none of these works addresses partially observable domains or employs a Bayesian approach directly to a physical system.

III. BACKGROUND

A. Partially Observable MDP (POMDP)

A POMDP [21] is formally defined by the tuple $(\mathcal{S}, \mathcal{A}, \Omega, T, O, R, p_{s_0})$, where $\mathcal{S}, \mathcal{A}, \Omega$ are respectively the state, action, and observation spaces; $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S}), O : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are respectively the transition, observation, and reward functions; $p_{s_0} = \Delta(\mathcal{S})$ is the prior about the initial state s_0. The goal is to find a sequence of actions to maximize the discounted return defined as $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$ [22] with some discount factor $\gamma \in [0, 1]$. An action a taken in state s will result in a state s', sampled from the transition function $T(s, a, s') = p(s' \mid s, a)$. In a POMDP, the agent can only observe $o \in \Omega$, indirectly related to s' via the observation function $O(s', a, o) = p(o \mid s', a)$. This implies that the agent might need to use the entire action-observation history, which grows with the episode length, to act optimally. Alternatively, acting optimally can rely on a belief (posterior probability distribution) $b \in \Delta(\mathcal{S})$ over possible states, where $b(s)$ denotes the probability that the environment's true state is s. Given a new action a and observation o, a new belief b' can be calculated as:

$$b'(s') \propto O(s', a, o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s). \tag{1}$$

Equation (1) indicates that explicitly belief tracking (and therefore planning) must rely on access to the *known* dynamics $T(s, a, s')$ of the system. When these are unknown, we must turn to learning-based approaches instead. This work adopts the *Bayesian* perspective.

B. General Bayes-Adaptive POMDP (GBA-POMDP)

BRL assumes that, instead of access to the system's dynamics $\mathcal{D}$, we have a parameterized *prior* probability distribution $p(\mathcal{D}; \theta)$. On a high level, our agent will maintain a belief $\bar{b} \in \Delta(S, \Theta)$ over the dynamics and the state and pick actions that optimize future return *with respect to this belief*. Technically, this Bayesian framework constructs a "belief" POMDP: an augmented POMDP of which the state space (and dynamics) include parameters that describe the dynamics of the underlying system. Even if the dynamics of the original system are unknown, it is now again possible to plan and track beliefs in the larger POMDP instead.

Definition 1 (GBA-POMDP): Given a dynamics prior $\theta \in \Theta$, and a parameter update function $\mathcal{U}$, then a general BA-POMDP is a POMDP defined by the tuple $(\bar{\mathcal{S}}, \mathcal{A}, \Omega, \bar{\mathcal{D}}, O, R, p_{\bar{s}_0})$ with augmented state space $\bar{\mathcal{S}} = S \times \Theta$ and prior $p_{\bar{s}_0} = (p_{s_0}, \theta_0)$. Denote δ_x as the Kronecker-delta function, which is zero everywhere except at x, then the update function $\mathcal{U}$ determines the augmented dynamics model $\bar{\mathcal{D}}(s', \theta', o \mid s, \theta, a)$, specified as:

$$\bar{\mathcal{D}} = p(s', o \mid s, a; \theta) \delta_{\theta'}(\mathcal{U}(\theta, s, a, s', o)), \tag{2}$$

where $p(s', o \mid s, a; \theta)$ is the dynamics according to a model parameterized by θ.

Since the GBA-POMDP *is a* POMDP, it can be solved through belief tracking and planning. Unfortunately, the state space is very large, and we need to reach for approximation techniques instead, which will be covered next.

Algorithm 1 Online Belief Tracking (b, a, o, P)

Require: belief $b = \{s, \theta\}^P$, action a, and observation o
1: $b' \leftarrow \emptyset$ ▷ Empty next belief
2: **while** sizeof$(b') < P$ **do**
3: $(s, \theta) \sim b$ ▷ Sample augmented state
4: $(s', \theta', \tilde{o}) \sim \bar{\mathcal{D}}(s, \theta, a)$ ▷ Use GBA-POMDP dynamics
5: **if** $\tilde{o} = o$ **then** ▷ Compare with real observation
6: Add (s', θ') to b'
7: **end if**
8: **end while**
9: **return** b'

1) Belief Tracking: Belief tracking is approximated with particle filtering, in this case, rejection sampling (algorithm 1). Given a current belief $b = p(s, \theta)$ and new action-observation pair (a, o), rejection sampling repeatedly samples, updates, and accepts/rejects particles. In particular, each iteration samples an augmented state from the belief $(s, \theta) \sim b$ and proposes a next augmented state using the dynamics $(s', \theta', \tilde{o}) \sim \bar{\mathcal{D}}(s, \theta, a)$. The proposal is accepted and added to the new belief when the simulated observation matches the *real* observation $\tilde{o} = o$, and otherwise rejected. The process repeats until we have P particles to represent b'. Note that the updated parameters θ are *preserved* throughout (not reset in-between episodes) so that the dynamics are continuously learned through episodes.

2) Online Planning: Algorithm 2 shows how Partially Observable Monte Carlo Planning (POMCP) [11], a Monte-Carlo tree search method, is used for action selection by building a look-ahead tree to evaluate the expected return of each action. The tree is incrementally built through simulations by calling a *Simulate* function N_s times, each starting with a sampled augmented state from the current belief (represented by P particles). When the tree is completed, the action with the largest value is selected. At the next timestep of action selection, the tree will be discarded and built anew. For more details on modifying POMCP for GBA-POMDP, please refer to [9].

Algorithm 2 POMCP (b, N_s, P)

Require: Current belief $b = \{s, \theta\}^P$
Require: Number of simulations N_s
1: $h_0 \leftarrow \emptyset$ ▷ Empty history
2: **for** $i = 1 \rightarrow N_s$ **do**
3: $(s, \theta) \sim b$ ▷ Sample augmented state
4: Simulate(s, θ, h_0) ▷ Build a tree
5: **end for**
6: Return $\text{argmax}_b\, Q(h_0 b)$ ▷ Greedy action selection

C. Bayes-Adaptive Deep Dropout RL (BADDr)

The previous section described the GBA-POMDP as a recipe that constructs a belief POMDP from a prior θ_0 and parameter update $\mathcal{U}$. BADDr [6] is a realization of GBA-POMDP that uses dropout neural networks [12] to represent the dynamics. In particular, it assumes that the prior over the dynamics is captured by a parameter set $w \in \mathcal{W}$ (*e.g.*, w can be neural network weights) and chooses the update function to be stochastic gradient descent with dropout (SGD) [23], which can be interpreted as an approximation of Bayesian inference over neural network parameters [24]. Consequently, BADDr is a POMDP with augmented state space $\bar{\mathcal{S}} = \mathcal{S} \times \mathcal{W}$ and dynamics:

$$\bar{\mathcal{D}}(\bar{s}', o \mid \bar{s}, a) = p(s', o \mid s, a; w)\delta_{w'}(\mathcal{U}(w, s, a, s', o)), \quad (3)$$

$$\mathcal{U} = w - \nabla\mathcal{L}((s, a), (s', o); w), \quad (4)$$

where $\mathcal{L}((s, a), (s', o); w) = -\log p(s', o \mid s, a; w)$ is the cross-entropy loss between the predicted and true next state and observation.

1) Prior Construction: The initial state distribution of BADDr $p_{\bar{s}} \in \Delta(S, \mathcal{D})$ is the product of the (given) initial POMDP state distribution p_s and a prior over the dynamics. BADDr parameterizes this prior $p_{\mathcal{D}}$ with parameters w_T and w_O, representing the transition and observation models, respectively. In general, it is unclear how to set network parameters to reflect prior knowledge, so a data-centric approach is taken instead. In particular, we sample POMDPs from the (assumed given) prior over the dynamics $p_{\mathcal{D}}$, which in turn are used to generate transitions (s, a, s', o) with, for example, a random policy. The prior networks w_T and w_O are trained using eq. (4) on transitions generated from the simulators. After training, $p_{\mathcal{D}}$ is represented with an ensemble of N parameter sets $\{(w_T, w_O)\}^N$.

2) Online Adaptation: A solution to BADDr is, as in any (GBA-)POMDP, a combination of belief tracking and online planning. After picking an action with POMCP and receiving observation, the agent updates its belief over both the current POMDP state and its dynamics according to BADDr's dynamics. As a result, the belief over (the dynamics) parameters w_T and w_O are updated like in eq. (4). This time, however, uses a *single* sample with the *real* observation from the environment (line 4-6 in algorithm 1).

IV. Baysian Inference in Robotic Systems

The Bayes-adaptive models described in the background are powerful in their ability to capture a broad class of problems. In practice, however, we have *more intricate prior knowledge than previous literature can capture*. For example, we may be confident enough in our understanding of the sensors that there is no need to learn a model of them or have a reliable low-level (physics) simulator. On the other hand, certain aspects of the real world will be unknown, such as preferences of collaborating humans, and encoding prior knowledge (that can be updated during execution) over them will be important. This knowledge can not be expressed with, for example, BADDr, which assumes a single set of parameters to describe the prior as a distribution. Similarly, most robots come with a reliable low-level (physics) simulator, which would be unrealistic to try to represent with Dirichlet distributions or even neural networks. In short, there is a need to incorporate mixed and partial prior knowledge from real-world applications to Bayesian methods.

A. Prior Knowledge in Robotic Systems

Typical assumptions include:

- There is a high-fidelity simulator of the state of the robot
- The (internal) state of the robot is fully observable
- Some high-level system behavior is unknown
- The model of the sensors is somewhat known

Expert knowledge in robotic tasks typically includes a physical simulator of the robot in which we can simulate actions. Similarly, the internal of the robot state is observable in most systems. What is likely unknown, and over which we may only have a distribution, is some high-level element of the dynamics of the environment in which the robot will be deployed. In this work, this is the human behavior in a human-robot setting. Other assumptions may apply for different applications, but we hope that most classes of assumptions have been covered here and that it will be relatively straightforward to adapt to future requirements.

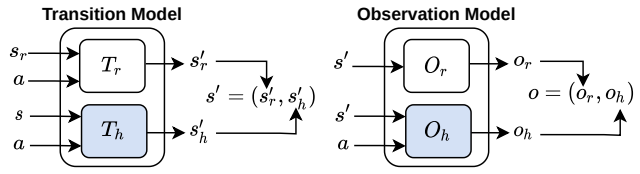


Fig. 1: By assuming that part of the dynamics $T_r(s_r, a, s'_r)$ is known and part of the state (o_r) computed from the state through O_r is fully observable, we only need to learn smaller transition (T_h) and observation (O_h) models (blue boxes).

We observe that, without loss of generalization, the state can be divided into internal (robot s_r) and external (here "human's" s_h) parts: $s = (s_r, s_h)$ (see fig. 1). Then we denote $T_r(s_r, a, s'_r)$ as the accurate but low-level simulator in which we can accurately predict the *agent's internal state* by unrolling a controller. On the other hand, the dynamics of the environment $T_h(s, a, s'_h)$ are unknown before deployment.

We make a similar distinction in the observation and divide it into an internal (robot) o_r and external (human) o_h part $o = (o_r, o_h)$. We assume that the internal state o_r (can be computed from the state through a given function O_r: $s' \to o_r$) is fully observable (note that a similar assumption is formalized in the *mixed observability* MDP [8]). The observation function over the variable o_h, $O_h(s', a, o_h)$, may or may not be given depending on our understanding of the robot's sensors.

B. Factored Bayesian Inference

The GBA-POMDP does not provide the tools to capture the expert knowledge we discussed above or how to do further inference over. Instead, we derive a more sophisticated framework starting from the assumptions stated above.

a) Priors: For simplicity, let us consider only the state transition model and note that the state space consists of features $\mathcal{S} = (\mathcal{S}_r, \mathcal{S}_h)$. According to our assumptions, the transition model T can be factorized into each feature: $T(s, a, s') = T_r(s_r, a, s'_r)T_h(s, a, s'_h)$. Indeed, our prior knowledge is factored this way: $p(T) = p(T_r)p(T_h)$. Specifically, the dynamics over the internal robot state are assumed known (*i.e.*, a physics simulator f), represented with the Kronecker-delta distribution $p(T_r) = \delta_f(T_r)$.

b) Posteriors: In the interest of updating our belief over the dynamics, we consider the shape of the posterior $p(T|\{s, a, s', o\})$. Given the transition $sas'o = \{s, a, s', o\}$, we first apply Bayes' rule and notice an interesting but crucial property:

$$\begin{aligned} p(T_r, T_h \mid sas'o) &= p(T_r \mid sas'o)p(T_h \mid \cancel{T_r}, sas'o) \\ &= p(T_r \mid sas'o)p(T_h \mid sas'o). \end{aligned} \tag{5}$$

Equation (5) indicates that the transition functions are independent of each other given the transition, $T_h \perp\!\!\!\perp T_r | sas'o$. This, first, implies that the posterior is also factored. Second, as a result, Bayesian inference can be made in separate computations, one for each factor of the prior! In this instance, due to knowing T_r, the posterior simplifies to $\delta_f(T_r)p(T_h \mid sas'o)$, where the last term depends on the prior representation $p(T_h)$.

C. Formal Definition as a Specialized Bayesian Model

Here we leverage the shape of the posterior derived above to define a Bayesian framework for robotics. In particular, we use the factorization of the posterior computation to construct a model with a factorization of the parameter update rule $\mathcal{U}$ — one for each factor.

The transition posterior computation, for example, is defined by an update rule for the distribution over the physics simulator and one for the distribution over the human dynamics. The update rule that corresponds to the Kronecker-delta function is simply its identity $\mathcal{U}(T_r, s, a, s', o) = T_r$. The posterior parameters of the human dynamics depend on the choice of the prior model $p(T_h)$, which in this case will be neural networks w_{hT} with the corresponding dropout SGD update rule $\mathcal{U}(w_{hT}, s, a, s', o)$ as in eq. (4). For example, eq. (5) can be (re-)written with a factored parameter update rule:

$$\begin{aligned} eq.\ (5) &= \mathcal{U}(\delta_f(T_r), s, a, s', o)\mathcal{U}(w_{hT}, s, a, s', o) \\ &= \delta_f(T_r)\text{SGD}(w_{hT}, s, a, s', o)\,, \end{aligned} \tag{6}$$

and we apply the same approach to the observation function.

The prior of known components (*e.g.*, T_r and O_r) are, as discussed before, encoded by Kronecker-delta distributions. Priors over other parts of the dynamics, such as the human's dynamics T_h and observations O_h, are a design choice, and here are assumed to be represented by some network parameters (w_{hT}, w_{hO}).

Definition 2: Formally, the resulting Bayes model is a POMDP with state space $\bar{\mathcal{S}} = (\mathcal{S}_r, \mathcal{S}_h, \mathcal{W}_{hT}, \mathcal{W}_{hO})$ and dynamics:

$$\begin{aligned} &\bar{\mathcal{D}}(s'_r, s'_h, w'_{hT}, w'_{hO}, o_r, o_h \mid s_r, s_h, w_{hT}, w_{hO}, a) = \\ &\text{(state)} \quad p(s'_r \mid s_r, a; T_r)p(s'_h \mid s, a; w_{hT})\times \\ &\text{(obs)} \quad p(o_r \mid s'; O_r)p(o_h \mid s', a; w_{hO})\times \\ &\text{(params)} \quad \delta_{w'_{hT}}(\mathcal{U}_{hT}(w_{hT}, s, a, s'_h))\times \\ &\qquad \delta_{w'_{hO}}(\mathcal{U}_{hO}(w_{hO}, s', a, o_h))\,, \end{aligned} \tag{7}$$

where the updates of the known components are omitted for brevity, and those of the neural network parameter are:

$$\mathcal{U}_{hT} = w_{hT} - \nabla\mathcal{L}((s, a), s'_h; w_{hT}) \tag{8}$$

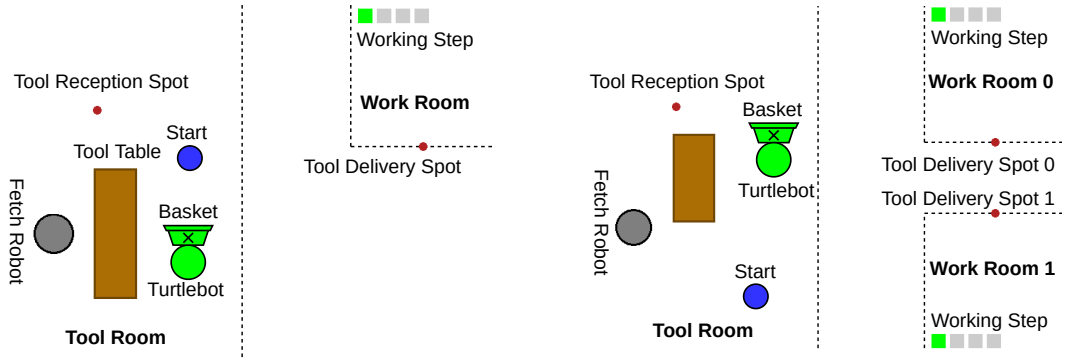
$$\mathcal{U}_{hO} = w_{hO} - \nabla\mathcal{L}((s', a), o_h); w_{hO}). \tag{9}$$

D. Solving the Bayesian Model

Starting with the problem definition of a general robotic system, we have developed a belief POMDP that captures typical expert prior knowledge and allows for Bayesian inference over unknown parts of the dynamics. To solve the resulting problem, however, we require efficient action selection and belief approximation. In particular, the belief space is far too large to either apply naive planning or exact inference. Instead, we propose to combine: a specialized POMCP algorithm, a particle filtering approximation technique, and a targeted prior construction.

First, we train networks on the priors we have over the unknown dynamic terms (*e.g.*, $\{w_{hT}\} \approx p(T_h)$). Then we initialize our belief b_0 by sampling particles (s, w_{hT}, w_{hO}) from our joint prior ($s \sim p_{s_0}$, and $(w_{hT}, w_{hO}) \sim \{w_{hT}, w_{hO}\}$). At each time step, our planner builds a look-ahead tree, as POMCP does, but where each simulation starts with sampling an augmented state $\sim b$ and using dynamics in eq. (7) to generate trajectories. For belief tracking, we use a new type of rejection sampling. This algorithm samples particles (s, w_{hT}, w_{hO}), proposes the next augmented states (again with dynamics in eq. (7)) and accepts those that generated the perceived observation.

This approach, first, can do approximate belief tracking through particle filtering in our specialized Bayes model and, second, pick actions efficiently with respect to our most up-to-date understanding of the environment. The result is a quick and feasible online algorithm that manages to both be sample efficient and capable of exploiting expert knowledge directly into the initial belief in a principled manner.



a) Ordered-Tools-Delivery (OTD) b) Speedy-Tools-Delivery (STD)

Fig. 2: Two human-robot interaction domains are used in our experiments.

V. EXPERIMENTS

We experiment on two human-robot interaction domains in which the agent needs to learn the tool order and working speeds of human collaborators that cannot be directly observed. Fortunately (and per usual), the robots come with extensive simulators and need not learn this part of the dynamics. However, before deployment, the robots have no idea in what order or how quickly these tools must be delivered. Instead, they have a uniform prior over these unknowns and must learn these factors online.

A. *Ordered-Tools-Delivery (*`OTD`*)*

A Turtlebot must deliver T tools to a human in an unknown order to complete an assembly task (fig. 2a). Turtlebot can carry multiple tools in its basket at once, and the tools are stored in a tool room while the human works in a work room. Each time the human worker receives a correct tool, he needs a fixed number of timesteps to use it before needing another. When a correct tool is delivered, the human's working step (located in the top right corner) increases by one.

State. A state includes the 2D coordinate x_{coord} of Turtlebot, the tools currently carried t_{carry}, and the currently needed tool t_{need}. The coordinates are internal and known to the robot, while the next needed tool is not. *Moreover, even t_{carry} is known to the robot, we consider it as a component of s_h because t_{carry} cannot be determined just using the previous (x_{coord}, t_{carry}) and the action*. For instance, with *Deliver* actions, determining t_{carry} requires extra information, such as whether the human workers are waiting for a tool. Therefore, $s = (s_r, s_h)$, where

$$s_r = x_{coord} \quad s_h = (t_{carry}, t_{need}). \tag{10}$$

Observation. The agent can observe the current room location x_{room} (work or tool room), the tool it is carrying t_{carry}, and the current working step of the human w_{step}, which is only observable in the work room. The location and tools it is carrying are a function of the robot's known internals (*e.g.*, coordinates), whereas the human working step is external, *i.e.*, $o = (o_r, o_h)$, where

$$o_r = (x_{room}, t_{carry}) \quad o_h = w_{step}. \tag{11}$$

In this experiment, we consider the observation function known a-priori:

$$O_r(s) \coloneqq (x_{room}, t_{carry}) \tag{12}$$

$$O_h(s) \coloneqq \begin{cases} w_{step}, & \text{if } x_{room} = \text{work room} \\ \varnothing & \text{if } x_{room} = \text{tool room} \end{cases} \tag{13}$$

Actions/Controllers. The action space consists of two types of actions. *Get-Tool(i)* with $i \in \{0, 1, \ldots, T-1\}$ moves Turtlebot to a *tool reception spot* in the tool room, where it can receive tool i from a Fetch robot that would get the tool from a table. *Deliver* moves Turtlebot to *a tool delivery spot* in the work room, where the human can take the tool that he needs *if* it is currently carried by the Turtlebot.

The navigation and tool pickup transitions are known a-priori $(s_r{=}x_{coord} \sim T_r(s_r, a)$, while the human behavior $(s_h{=}(t_{carry}, t_{need}) \sim T_h(s, a)$ is not.

Reward. The agent is rewarded $+100$ for delivering a tool. To encourage the agent to achieve the task as quickly as possible, each timestep is given a step reward of -1. The cost of *Get-Tool(i)* tools depends on the time it takes for Fetch to pick and place items and for the Turtlebot to navigate.

Episode Initialization. An episode starts with Turtlebot (not carrying any tool) at the blue spot; all tools are placed on the tool table; and the human begins with $w_{step} = 0$.

B. *Speedy-Tools-Delivery (*`STD`*)*

In this domain, the Turtlebot must deliver tools to two human workers (Human-0 and Human-1) in separate rooms (fig. 2b). Importantly, given the same tool, one of the workers works faster than the other. The table now holds two identical sets of tools, each containing three tools, and each human requires one set to complete the task. The tool order is the same for both workers and assumed to be known, *i.e.*, Tool $0 \to 1 \to 2$.

State. In this domain, the state additionally includes a speed bit b_{speed}, which is 1 if Human-0 works faster than Human-1 and 0 otherwise. Moreover, t_{need} is now the tools needed by *two* humans, and for the same reason described in `OTD`, t_{carry} is unknown, making $s = (s_r, s_h)$, where

$$s_r = x_{coord} \quad s_h = (t_{carry}, t_{need}, b_{speed}). \tag{14}$$

Observation. Unlike OTD, the agent additionally can observe the unused tools on the table t_{table} when it is in the tool room, and w_{step} is the working steps of *two* humans (only observable in the corresponding work rooms). Adding t_{table} to the observation is necessary because there are two identical sets of tools. Consequently, the complete observation is $o = (o_r, o_h)$, where

$$o_r = (x_{room}, t_{carry}) \quad o_h = (w_{step}, t_{table}). \tag{15}$$

Like before, we also assume a known observation function:

$$O_r(s) \coloneqq (x_{room}, t_{carry}) \tag{16}$$

$$O_h(s) \coloneqq \begin{cases} (w_{step}, \varnothing), & \text{if } x_{room} = \text{work room} \\ (\varnothing, t_{table}), & \text{if } x_{room} = \text{tool room} \end{cases} \tag{17}$$

Action/Controllers. Similar to OTD, *Get-Tool(i)* with $i \in \{0, 1, 2\}$ will get tool i from the table. However, *Deliver(j)* with $j \in \{0, 1\}$ will deliver tools for Human-j by moving to the delivery spot of the corresponding work room. Like before, the navigation and tool pick-up transition is known ($s_r{=}x_{coord} \sim T_r(s_r, a)$, whereas the human factors ($s_h{=}(t_{carry}, t_{need}, b_{speed}) \sim T_h(s, a)$ must be learned.

Reward. We apply the same reward for a correct tool delivery and use the same cost for *Get-Tool(i)*. Apart from a step reward of -1, a step penalty of -30 and -10 are incurred if the faster and the slower human must wait, respectively. These additional rewards are necessary for the agent to learn to prioritize the faster worker.

Episode Initialization. Like OTD, Turtlebot starts at the blue spot with no tool. The two tool sets are placed on the table, and the two humans begin with $w_{step} = \{0, 0\}$.

C. Priors

In OTD, the transition prior assumes a uniform distribution over $T!$ possible delivery orders for T tools. Assuming a *correct* observation model, we represent these priors using $T!$ parameter sets (w_{hT}, w_{hO}). To generate training data for training a parameter set, we pair a random tool order with the correct observation model to sample the dynamics. We then use a random policy to interact with the resulting dynamics to generate the data.

In STD, we consider a set S consisting of $|S|$ possible working speeds. Because the speeds are different (one human worker works faster than the other worker), $|S|^2 - |S|$ possible different speed combinations are possible. Over these combinations, we assume a uniform transition prior, which is defined using $|S|^2 - |S|$ parameter sets (w_{hT}, w_{hO}). Similar to OTD, to generate training data for training a parameter set, we pair a random speed combination with a *correct* observation model and the *correct* tool order to create the dynamics and then use a random policy to interact.

D. Real-World Experiments

Set-up. The experiments take place in a rectangular workspace that measures 5.0×7.0 meters and contains two tables that represent work rooms (see fig. 3). Two identical sets of tools are available, each including a clamp, a sandpaper package, and a tape measure. Only one set of tools is used for the OTD task.

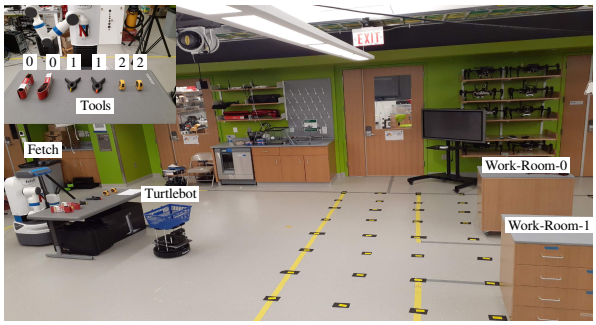


Fig. 3: The lab workspace to perform experiments.

Experiment Scenarios. We perform experiments in two scenarios for each domain:

- OTD. The number of tools $T = 3$ and the correct tool orders are Tool $0 \to 1 \to 2$ and Tool $0 \to 2 \to 1$.
- STD. The set of possible speeds $S = \{10, 20, 30\}$, and the true speeds are $(10, 20)$ and $(20, 10)$.

Evaluation Metrics. We report the mean discounted return with $\gamma = 0.95$, averaged over five runs. Each run lasts 10 and 20 episodes for OTD and STD, respectively. We also visualize the one standard error area around the mean.

(a) OTD Real (b) STD Real

Fig. 4: Real-world results in OTD with three tools and STD with two speed combinations. The dotted lines indicate the upper bound of POMCP [11] using the *true* POMDP. Results averaged over five seeds with shaded one standard error.

Results. Figure 4a shows that our method can reach near-optimal performance within ten episodes in both domains. Such performance will unlikely be achievable by pure RL methods (see a comparison in section V-E). We will later show in section V-F that our approach also outperforms BADDr, given the same amount of initial training for the prior networks.

In OTD, our agent nearly reaches the performance of POMCP running on the *true* POMDP after three episodes. During the first episode of OTD, the agent relies solely on the initial prior, as the dynamics parameters have not yet been updated. The observed behavior of the agent is to gather all three tools and then deliver them. While this strategy yields a reasonable outcome, it is sub-optimal as the worker only requires a tool once they have finished using the current one. In the second episode, the agent performs better by only taking two tools in two consecutive actions, delivering them,

and then returning later for the third tool. In the third episode, it learns to retrieve one tool at a time and deliver it until all tools have been supplied, which is optimal. In order to see the robot in action, please see our video.

STD seems more challenging as the agent needs more episodes to perform well. Specifically, fig. 4b indicates that our agent can reach the performance of POMCP within ten episodes by delivering all tools in the correct order for both tested working speed combinations. The final policy involves taking two tools of the same type (*e.g.*, tool 0) and delivering them one by one, starting with the faster worker and then moving on to the slower worker. This strategy allows the agent to leverage the close distance between the two work rooms. Please see our video for the learned policy.

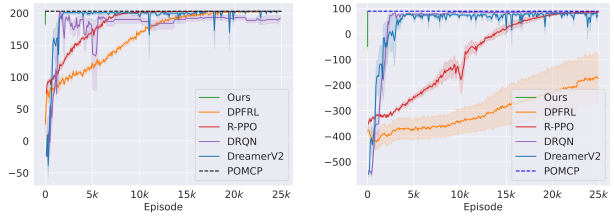
Simulating $T_r(s_r, a, s'_r)$. We first build a map of the experiment area and import it to a commonly used simulator of Turtlebot in Gazebo. This simulator then acts as T_r, where we can set the 2D coordinate x_{coord} of Turtlebot, send moving actions, and retrieve the next coordinates x'_{coord}.

Obtaining Observations. Tools on table t_{table} are determined using the RGB image of the tool table taken by Fetch's head camera. The human workers' working steps w_{step} are the number of times the minimum depth in the image from Turtlebot's depth camera falls below a certain threshold, indicating that a human is approaching to pick up a tool. The room location x_{room} is determined by comparing x_{coord} with the rooms' dimensions. During an episode, t_{carry} is calculated by keeping track of t_{table} and w_{step}.

Other Implementation Details. *Get-Tool(i)* actions of Fetch are pre-recorded using a multi-step process. First, point cloud data from the head camera is projected into an OpenRAVE [25] environment. Then, the OMPL [26] library is used for motion planning. Fetch and Turtlebot are controlled through a ROS [27] node, which sends observations to a planning node via a ROS service. The planning node, allowed $N_s = 1024$ simulations, responds with the computed action after about 2.5 seconds of planning. The prior networks implemented in PyTorch are three-layered with 128 neurons per layer and `Tanh` activation function with a dropout rate of 0.1. They are trained for 2,000 epochs using SGD with a learning rate 0.1, then switched to 0.001 for online learning.

E. Non-Bayesian RL Comparison in Simulation

Baselines. To further investigate the sample efficiency of our method, we compare it against several non-Bayesian RL baselines in *simulation*: DRQN [28] is a recurrent version of Deep Q-Networks (DQN) [1] and is a classical baseline for POMDPs. R-PPO is the recurrent version of Proximal Policy Optimization (PPO) [3]. Discriminative Particle Filter RL (DPFRL) [29] is a model-based RL POMDP method that performs reinforcement learning on the features from a differentiable particle filter. DreamerV2 [30] is a strong model-based agent which alternates between learning a *recurrent* world model (therefore can tackle POMDPs) and performing RL using imagined data. It outperformed strong model-free baselines such as [31] in the Atari benchmark. Finally, like before, we also include POMCP running on the *true* POMDP as an upper bound performance.



(a) OTD Sim (b) STD Sim

Fig. 5: Simulation results against RL baselines. Averaged five seeds with shaded one standard error.

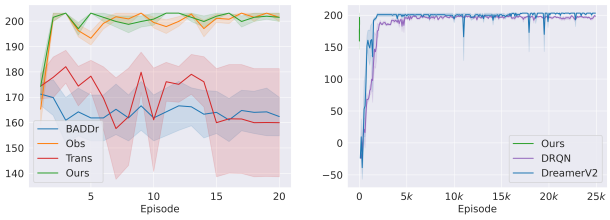
Results. In fig. 5, our method (barely seen in the top left corner) exhibits significantly greater sample efficiency compared to RL baselines in both domains. Additionally, our approach outperforms from the beginning due to leveraging prior information. Specifically, in the OTD domain, DreamerV2 requires approximately 1,000 episodes to achieve the performance our method attains in just five episodes. In the more challenging STD domain, the numbers are 2,500 and 10, respectively. Among baselines, DreamerV2 performs best in OTD while DRQN performs best in STD. Regardless, these baselines sometimes act suboptimally at the end of the training (see spikes in their learning curves). For instance, in OTD, DreamerV2 still sometimes outputs a redundant *Get-Tool* action instead of delivering its current tool to the waiting human. And, in STD, DRQN occasionally prioritizes the slower human worker first. In contrast, R-PPO exhibits better convergence performance in both domains.

F. Ablation Studies in Simulation

To ablate our agent, we perform experiments in simulation with the OTD task with three tools.

Effect of Factored Models. We conducted an experiment where we provided the same initial training for the prior networks (all trained for 2,000 epochs) and compared the proposed approach (Ours) with its variants, namely, no factorizations used (*i.e.*, the original BADDr), factored transition model only (Trans), and factored observation model only (Obs). Figure 6a indicates that our method and Obs perform best. Obs likely performs well because O_r in OTD is relatively small, resulting in a slight performance improvement when using the factored observation model. Conversely, we anticipate a more substantial performance improvement in domains where O_r is a significant component.

Using Imperfect Observation Models. Here, we investigate how our approach performs against DRQN and DreamerV2 (best RL baselines in section V-E) when we do not assume a correct observation model. For this purpose, we add *stochasticity* while observing the currently carried tools t_{carry}, which is now only correctly observable with a probability $p_{correct} = 0.85$. However, the data for initially training the prior observation model is obtained with $p_{correct} = 0.5$.



(a) Effect of Factored Models (b) Imperfect Obs. Model

Fig. 6: Ablation studies in OTD with three tools.

In this case, fig. 6b still confirms the sample efficiency superiority of our approach over the baselines.

VI. CONCLUSIONS

To our knowledge, this work presents the first on-robot Bayesian RL method for partially observable scenarios. By employing factored dynamics and mixed observability assumptions, our method can rapidly acquire high-quality behavior in long-horizon, real-world tasks within a minimal number of episodes, outperforming pure RL approaches. Although our tasks are relatively simple, the results clearly demonstrate the potential of a Bayesian approach for effective learning directly on physical hardware. One limitation of the approach is a slow inference speed, as the agent needs time to search for the next action. This weakness can be overcome using an RL agent to mimic our agent's actions, as previously investigated in [32].

ACKNOWLEDGMENTS

We thank Trung-Hieu Nguyen, Minh Nguyen, Van Anh Tran, and Yunus Terzioglu for helping in the robot experiments. This work is supported by the Army Research Office under award number W911NF20-1-0265 and NSF grant 2024790.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[5] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[6] S. Katt, H. Nguyen, F. A. Oliehoek, and C. Amato, "Baddr: Bayes-adaptive deep dropout rl for pomdps," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 723–731.

[7] S. Ross, J. Pineau, B. Chaib-draa, and P. Kreitmann, "A bayesian approach for learning and planning in partially observable markov decision processes." *Journal of Machine Learning Research*, vol. 12, no. 5, 2011.

[8] S. C. Ong, S. W. Png, D. Hsu, and W. S. Lee, "Pomdps for robotic tasks with mixed observability." in *Robotics: Science and systems*, vol. 5, 2009, p. 4.

[9] S. Katt, F. A. Oliehoek, and C. Amato, "Learning in pomdps with monte carlo tree search," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1819–1827.

[10] ——, "Bayesian reinforcement learning in factored pomdps," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 7–15.

[11] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," *Advances in neural information processing systems*, vol. 23, 2010.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[13] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[14] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, "End-to-end robotic reinforcement learning without reward engineering," *arXiv preprint arXiv:1904.07854*, 2019.

[15] A. Zhan, P. Zhao, L. Pinto, P. Abbeel, and M. Laskin, "A framework for efficient robotic manipulation," *arXiv preprint arXiv:2012.07975*, 2020.

[16] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[17] D. Wang, M. Jia, X. Zhu, R. Walters, and R. Platt, "On-robot learning with equivariant models," in *Conference on robot learning*, 2022.

[18] X. Zhu, D. Wang, O. Biza, G. Su, R. Walters, and R. Platt, "Sample efficient grasp learning using equivariant models," *arXiv preprint arXiv:2202.09468*, 2022.

[19] L. Smith, I. Kostrikov, and S. Levine, "A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning," *arXiv preprint arXiv:2208.07860*, 2022.

[20] P. Wu, A. Escontrela, D. Hafner, K. Goldberg, and P. Abbeel, "Daydreamer: World models for physical robot learning," *arXiv preprint arXiv:2206.14176*, 2022.

[21] 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.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[23] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[24] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *International conference on machine learning*, 2016, pp. 1050–1059.

[25] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, vol. 79, 2008.

[26] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[28] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[29] X. Ma, P. Karkus, D. Hsu, W. S. Lee, and N. Ye, "Discriminative particle filter reinforcement learning for complex partial observations," *arXiv preprint arXiv:2002.09884*, 2020.

[30] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[31] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[32] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," *Advances in neural information processing systems*, vol. 27, 2014.