Scalable Bayesian Reinforcement Learning

Gilwoo Lee

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Siddhartha S. Srinivasa, Chair

Byron Boots

Debadeepta Dey

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Scalable Bayesian Reinforcement Learning

Gilwoo Lee

Chair of the Supervisory Committee:
Boeing Endowed Professor Siddhartha S. Srinivasa
Computer Science & Engineering

Informed and robust decision making in the face of uncertainty is critical for robots operating in unstructured environments. We formulate this problem as Bayesian Reinforcement Learning (BRL) over latent Markov Decision Processes (MDPs). While Bayes-optimality is theoretically the gold standard, existing algorithms scale poorly to continuous state and action spaces. This thesis proposes a set of BRL algorithms that scale to complex control tasks. Our algorithms build on the following insight: robotics problems have structural priors that we can use to produce approximate models and experts that the agent can leverage.

First, we propose an algorithm which improves a nominal model and policy with data-driven semi-parametric learning and optimal control. Then, we look into more general BRL tasks with complex latent models. We propose algorithms that combine batch reinforcement learning algorithms with experts to scale to complex latent tasks. Finally, through simulated and physical experiments, we demonstrate that our algorithms drastically outperform existing adaptive RL methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Sidd Srinivasa, who has been genuinely supportive throughout my journey, both as a mentor and a good friend. He has taught me to fearlessly ask questions, sharpen my thoughts, and explore new research problems. The critical thinking and passion I have learned from Sidd have become my most valuable asset.

I am grateful to my thesis committee members, Byron Boots, Debadeepta Dey, and Samuel A. Burden, for their input. Byron, thank you for sharing your knowledge and experience in machine learning. Dey, thank you for your insightful questions and for being supportive of my career path. Sam, thank you for your feedback on optimal control. I would also like to thank Leslie P. Kaelbling, Tomás Lozano-Pérez, and Matthew T. Mason for inviting me to the fields of robotics and machine learning.

Despite many rumors on how Ph.D. programs can be painful, I have enjoyed this journey, largely owing to my collaborators and the members of the Personal Robotics Lab. Thanks especially to Brian and Sanjiban, who have been not only great collaborators but also great friends. Thanks to Aditya, Rosario, Tapo, Sherdil, Schmittle, Chris M., Kay, William, Patrick, Ethan, J.S., Hanjun, Amal, Matthew R., and Stefanos with whom I share the joy of being in Seattle. Many thanks to Mike K. and Jen K. for patiently helping me through my first year of robot experiments. Thanks to Shushman, Rachel, Aaron J., Shervin, Laura, Chris D., Matt K., Zita, and Henny, with whom I spent my first two years at the Carnegie Mellon University.

I would like to thank my family, especially my mother, for the love and support throughout this journey. My final thanks go to my husband, Youngsun, who has always been supportive. None of this would have been possible without you.

DEDICATION

To Erin

FUNDING

This thesis contains work partially supported by the National Institute of Health R01 (#R01EB019335), National Science Foundation CPS (#1544797), National Science Foundation NRI (#1637748), the Office of Naval Research, the RCTA, Amazon, and Honda Research Institute USA. The author is partially supported by Kwanjeong Educational Foundation.

Chapter 1

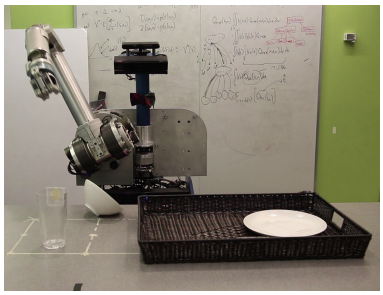
INTRODUCTION

Robots that are deployed in the real world must operate in the face of uncertainty. For example, a robot that manipulates objects must estimate the physical properties of the objects, such as mass, shape, and friction coefficients (Figure 1.1a). An autonomous vehicle must infer the latent goals of other cars to safely navigate around them (Figure 1.1b). A robot interacting with humans must infer and adapt to latent human intentions (Figure 1.1c). All of these properties, which define the task and dynamics, are often unobservable or provide only noisy observations.

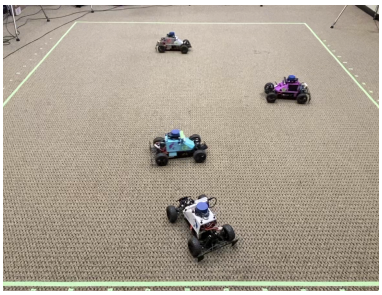
One could argue that some of these latent properties can be inferred with system identification, with which one can build models and generate optimal policies. This is true to some degree, as sensors and simulators have improved significantly in the past decade. One would expect simulation-trained policies, combined with high-precision sensors, to transfer gracefully to the real world.

However, we often observe that simulator-based policies perform poorly in the real world, even with good system identification. There are two fundamental challenges associated with the sim-to-real transfer. First, a simulator does not have the perfect model of the real world. Nonlinear stochasticity and wear and tear are hard to model and therefore often get neglected. Second, more critically, system identification itself requires active exploration in the real world, at the risk of damaging the robot or the environment. These challenges require the agent to handle model uncertainty online, while performing the task.

How do we prepare the agent with such capabilities? In fact, admitting that we will never know the perfect model is the key. We must prepare the agent as if we would prepare ourselves for a hiking trip, preparing for various scenarios that may arise on the road. This



(a) Object manipulation



(b) Autonomous Driving

(c) Assistive Feeding

Figure 1.1: Various robotic tasks that require inferring latent properties of the object, goals of other agents, or human intentions.

leads to the central tenet of our thesis:

Prepare the agent offline to handle uncertainty online optimally.

Depending on what the agent can afford in the real world, there are several strategies that the agent may pursue. The first case is when the agent can afford real-world failures. Such a scenario may arise when the agent repeatedly performs a task in a structured environment, such as pick-and-place in a warehouse. With a recovery system that allows the agent to drop the items occasionally, the agent can use the failure cases to improve its policy. We propose one such approach, GP-Iterative Linear Quadratic Control (Chapter 3), which combines semi-parametric learning with optimal control to improve the model and policy.

The second case is when the robot cannot afford such cost-free interactions. This is a more likely scenario for robots deployed in less-structured environments, such as autonomous driving agents on the street. Unfortunately, the agent does not know which latent MDP it interacts with, preventing it from acting optimally. At best, the agent can be *Bayes-optimal*, i.e., optimal with respect to the uncertainty. The agent must balance two objectives: (1) probing the system (*exploration*), and (2) maximizing an expected long-term reward (*exploitation*).

Model-based Bayesian Reinforcement Learning (Ghavamzadeh et al., 2015) elegantly

incorporates uncertainty into this exploration-exploitation dilemma. Here, the agent maintains a *belief*, a distribution over the latent parameters. Our prior knowledge over the candidate models can be represented as the prior distribution. A Bayes-optimal policy often includes robust and uncertainty-reducing actions, but only to the degree that they maximize task performance.

In this thesis, we focus on achieving Bayes-optimality in complex control tasks in continuous state and action spaces. While most control tasks require continuous actions, existing approaches (Kurniawati et al., 2008; Pineau et al., 2003b) often discretize actions, which must resort to coarse discretization as the action space grows. State-of-the-art algorithms that handle continuous actions require heavy computation at test time (Silver & Veness, 2010; Sunberg & Kochenderfer, 2017), making them unsuitable for control tasks that require fast and reactive actions. Our approaches find Bayes-optimal policies that are fast and reactive, making them suitable for application in robotics. Our algorithms are based on the following key insight:

Robotics problems have structural priors that we can leverage.

First, noting that our prior knowledge of the task domain is materialized as model priors, we build upon batch reinforcement learning algorithms, such as Proximal Policy Optimization (Schulman et al., 2017), to sample from the model prior and run the simulation in batches. By observing how certain actions collapse beliefs while some others result in undesirable states, the agent learns to take uncertainty-reducing and robust actions, even without auxiliary information-gathering reward or risk-sensitive cost terms. See Bayesian Policy Optimization (Chapter 4) and Bayesian Residual Policy Optimization (Chapter 7) for more discussion.

Second, we use experts derived from model priors to form initial policies. Such initial policies are substantially better than randomly initialized ones, and therefore significantly accelerate learning. By utilizing policies that give the agent a head start, the agent focuses on learning to be Bayesian instead of learning the task from scratch. Bayesian Residual

Q-Learning (Chapter 6) and Bayesian Residual Policy Optimization take this approach.

1.1 Contributions

This thesis develops a set of Bayesian Reinforcement Learning algorithms for continuous state and action spaces. Specifically, we contribute the following algorithms:

- GP-Iterative Linear Quadratic Control (GP-ILQG) combines semiparametric model learning and robust optimal control.
- Bayesian Policy Optimization (BPO) extends PPO to Bayes-Adaptive Markov Decision Processes (BAMDPs). We introduce two encoders, one for belief and the other for the state, to robustly handle large latent spaces.
- Bayesian CPACE (BCPACE) provides PAC-Bayes-optimality as well as a distance metric for BAMDPs.
- Bayesian Residual Q-Learning (BRQN) uses value-function experts to accelerate Q-Learning in BAMDPs.
- Bayesian Residual Policy Optimization (BRPO), combines experts' policies with PPO via residual learning. We prove that the agent's policy improves monotonically from the expert ensemble's performance.

Most of our algorithms are evaluated on simulated experiments, with the exception of BRPO. Simulated experiments are carefully selected to highlight common challenges for control tasks with model uncertainty. Qualitatively, we verify that the learned policies take robust and uncertainty-reducing actions. Quantitatively, we compare with belief-agnostic RL algorithms and adaptive RL algorithms and verify that our algorithms outperform the non-Bayesian ones. Further, we use discrete problems to compare with state-of-the-art discrete BRL algorithms and verify that our algorithms indeed reach Bayes-optimality. Finally, through physical experiments on the MuSHR racecar platform (Srinivasa et al., 2019), we show that the BRPO agent is well-suited for real-robot tasks that require fast and reactive policies.

Chapter 2

RELATED WORK

2.1 Overview

A long history of research addresses belief-space reinforcement learning. Here, we highlight the most relevant work and refer the reader to Ghavamzadeh et al. (2015), Shani et al. (2013), and Aberdeen (2003) for more comprehensive reviews of the Bayes-Adaptive and Partially Observable MDP literature.

Table 2.1 compares our algorithms with state-of-the-art algorithms that handle model uncertainty. SARSOP (Kurniawati et al., 2008) and POMDP-LITE (Chen et al., 2016), POMCPOW (Sunberg & Kochenderfer, 2017) are POMDP algorithms that can be used for Bayesian RL problems (Section 2.3). EPOpt (Rajeswaran et al., 2017) is a robust RL algorithm (Section 2.4). PSRL (Osband et al., 2013) is an online learning algorithm (Section 2.8).

The top row describes the criteria we believe are essential for Bayesian RL algorithms for control tasks. The first category is whether the algorithm can handle "continuous state and action" spaces, which we believe is necessary for control tasks. Many state-of-the-art algorithms for POMDPs require discretization, which limits their scalability for tasks with large continuous spaces.

The "multi-modal policies" category contains qualitative evaluation for policy gradient algorithms, such as EPOpt and BPO, of their empirical results. These algorithms can learn multi-modal policies in theory, but we often observe that they converge to a smooth, unimodal policy across multiple latent models.

The "offline learning" category applies to offline policy search algorithms. Policies from offline learning algorithms can perform poorly if the model prior used in the policy search is incorrect. However, this is not too concerning, as human experts can provide reasonable

Algorithm	Continuous State/Action	Multi-modal Policies	Offline Learning	Incorporates Uncertainty	Fast & Reactive
SARSOP	×	✓	✓	✓	✓
POMDP-LITE	×	✓	×	✓	×
POMCPOW	✓	✓	×	✓	×
EPOpt	✓	×	✓	×	✓
PSRL	✓	✓	×	×	✓
GP-ILQG	✓	×	×	✓	✓
BPO	✓	×	✓	✓	✓
BCPACE	✓	✓	✓	✓	×
BRQN	×	✓	✓	✓	✓
BRPO	✓	✓	✓	✓	✓

Table 2.1: Comparison of various algorithms that handles model uncertainty. See Section 2.1 for more discussion.

priors with domain-specific knowledge. While online learning algorithms can adapt their policies and underlying models even if the prior is incorrect, they can produce critical failures during the exploration. These failures are often not suitable for robot control tasks unless provided with a safe recovery system.

The "incorporates uncertainty" category applies to algorithms whose policies incorporate uncertainty in the form of belief distribution. Even when an algorithm makes use of model uncertainty, their policies may not be Bayes-optimal. For example, EPOpt produces policies for worst-case scenarios, and PSRL produces optimal policies for sampled models.

Finally, the "fast and reactive" catogory applies to algorithms that do not require much online computation, such as forward simulation or Bellman update. Often, this is a limiting factor for online algorithms such as POMCPOW for their application in robotics. Given a small online computation budget, these algorithms cannot produce Bayes-optimal policies.

2.2 Optimal Control

Most model-based reinforcement learning (MBRL) has both model learning (system identification) and policy optimization components (Kober et al., 2013). The data for a model comes either from real world or simulation, and is combined to construct a model via nonlinear function approximators such as Locally Weighted Regression (Atkeson et al., 1997), Gaussian Processes (Rasmussen, 2006), or Neural Networks (Narendra & Parthasarathy, 1990). Once the model is built, a typical policy gradient method computes the derivatives of the cost function with respect to control parameters (Deisenroth & Rasmussen, 2011; Levine & Koltun, 2013).

If an analytic model is given, e.g., via equations of motion or as a simulator[1], one can use classical optimal control techniques such as Differential Dynamic Programming (DDP) (Jacobson & Mayne, 1970), which compute a reference trajectory as well as linear feedback control law. For robustness, Iterative Linear Quadratic Gaussian Control (ILQG) (Todorov & Li, 2005) or H-∞ Control (Stoorvogel, 1993) can be used to incorporate multiplicative noise. Variants of ILQG have been used to generate guiding policies for data-driven RL methods (Levine & Koltun, 2013; Zhang et al., 2016). Recently, there have been some attempts to combine DDP or ILQG with data-driven models by replacing analytical models with locally linear models (Mitrovic et al., 2010; Yamaguchi & Atkeson, 2015) or nonlinear models (Pan & Theodorou, 2014; Pan et al., 2015; Yamaguchi & Atkeson, 2016; Pan & Theodorou, 2015) learned by Gaussian Processes or Neural Networks.

The goal of our algorithm, GP-ILQG in particular, is closely aligned with those of Zagal et al. (2004) and Abbeel et al. (2006). Zagal et al. (2004) has proposed a framework in which the robot maintains two controllers, one for the simulator and another for the real world, and aims to narrow the difference. Abbeel et al. (2006) assumes a deterministic real world, constructs an optimal policy based on the simulator's deterministic model, and evaluates its performance in the real world, while successively augmenting the simulator's model with

[1] We consider black-box simulators as analytical models, as derivatives can be taken by finite differencing.

time-dependent corrections based on real-world observations. GP-ILQG considers a stochastic system and the correction is not time-dependent.

GP-ILQG's online learning approach resonates with Ross & Bagnell (2012), which extends DAgger to the MBRL setting and iterates between model learning and optimal control-based exploration with an additional exploration policy. The key difference is that GP-ILQG focuses on safe exploration by incorporating uncertainty into the model learning and policy search, while Ross & Bagnell (2012) provides stronger guarantees of achieving near-optimal performance. It would be an interesting direction to consider whether the guarantee can extend if we incorporate model uncertainty into the algorithm.

2.3 Belief-Space Reinforcement Learning

Planning in belief space, where part of the state representation is a belief distribution, is intractable (Papadimitriou & Tsitsiklis, 1987). This is a consequence of the curse of dimensionality: the dimensionality of belief space over a finite set of variables equals the size of that set, so the size of belief space grows exponentially. Many approximate solvers focus on one or more of the following: 1) value function approximation, 2) compact, approximate belief representation, or 3) direct mapping of belief to an action. QMDP (Littman et al., 1995a) assumes full observability after one step to approximate Q-value. Point-based solvers, like SARSOP (Kurniawati et al., 2008) and PBVI (Pineau et al., 2003b), exploit the piecewise-linear-convex structure of POMDP value functions (under mild assumptions) to approximate the value of a belief state. Sampling-based approaches, such as BAMCP (Guez et al., 2012) and POMCP (Silver & Veness, 2010), combine Monte Carlo sampling and simple rollout policies to approximate Q-values at the root node in a search tree. Except for QMDP, these approaches target discrete POMDPs and cannot be easily extended to continuous spaces. Sunberg & Kochenderfer (2018) extend POMCP to continuous spaces using double progressive widening. Model-based trajectory optimization methods (Platt et al., 2010; van den Berg et al., 2012) have also been successful for navigation on systems like unmanned aerial vehicles and other mobile robots.

Neural network variants of POMDP algorithms are well suited for compressing high-dimensional belief states into compact representations. For example, QMDP-Net (Karkus et al., 2017) jointly trains a Bayes-filter network and a policy network to approximate Q-value. Deep Variational Reinforcement Learning (Igl et al., 2018) learns to approximate the belief using variational inference and a particle filter, and it uses the belief to generate actions. Our methods, BPO and BRPO, are closely related to Exp-GPOMDP (Aberdeen & Baxter, 2002), a model-free policy gradient method for POMDPs. We leverage model knowledge and revisit the underlying policy optimization method with recent advancements. Peng et al. (2018) use Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) to encode a history of observations to generate an action. The key difference between BPO and Peng et al. (2018) is that BPO explicitly utilizes the belief distribution, while in Peng et al. (2018) the LSTM must implicitly learn an embedding for the distribution. BRPO takes a step further and leverages experts to scale to complex Bayesian tasks. We believe that explicitly using a Bayes filter improves data efficiency and interpretability.

2.4 Robust (Adversarial) Reinforcement Learning

One can bypass the burden of maintaining belief and still find a robust policy by maximizing the return for worst-case scenarios. Commonly referred to as Robust Reinforcement Learning (Morimoto & Doya, 2001), this approach uses a min-max objective and is conceptually equivalent to H-infinity control (Başar & Bernhard, 2008) from classical robust control theory. Recent works have adapted this objective to train agents against various external disturbances and adversarial scenarios (Pinto et al., 2017; Bansal et al., 2018; Pattanaik et al., 2018). Interestingly, instead of training against an adversary, an agent can also train to be robust against model uncertainty with an ensemble of models. For example, Ensemble Policy Optimization (EPOpt) (Rajeswaran et al., 2017) trains an agent on multiple MDPs and strives to improve worst-case performance by concentrating rollouts on MDPs where the current policy performs poorly. Ensemble-CIO (Mordatch et al., 2015) optimizes trajectories across a finite set of MDPs.

While adversarial and ensemble model approaches have proven to be robust even to unmodeled effects, they may result in overly conservative behavior when the worst-case scenario is extreme. In addition, since these methods do not infer or utilize uncertainty, they perform poorly when explicit information-gathering actions are required. Our approaches are fundamentally different in that ours internally maintains a belief distribution. As a result, our policies outperform robust policies in many scenarios.

2.5 Adaptive Policy Methods

Some approaches can adapt to changing model estimates without operating in belief space. Adaptive-EPOpt (Rajeswaran et al., 2017) retrains an agent with an updated source distribution after real-world interactions. Posterior Sampling Reinforcement Learning (Osband et al., 2013) samples from a source distribution, executes an optimal policy for the sample for a fixed horizon, and then re-samples from an updated source distribution. These approaches can work well for scenarios in which the latent MDP is fixed throughout multiple episodes. Universal Policy with Online System Identification (UP-OSI) (Yu et al., 2017) learns to predict the maximum likelihood estimate ϕ_{MLE} and trains a universal policy that maps (s, ϕ_{MLE}) to an action. However, without a notion of belief, both PSRL and UP-OSI can over-confidently execute policies that are optimal for the single estimate, causing poor performance in expectation over different MDPs.

2.6 Residual Learning

Residual learning has its foundations in boosting (Freund & Schapire, 1999), which builds a strong ensemble by sequentially training weak learners that address the failures of their predecessors. Boosting with hand-designed policies or models allows priors to be injected into RL. Prior work has leveraged known but approximate models by learning the residual between the approximate dynamics and the discovered dynamics (Ostafew et al., 2014, 2015; Berkenkamp & Schoellig, 2015). GP-ILQG takes a similar approach and learning the residual between an approximate model and observed real-world dynamics, while iteratively improving

its policy. There has also been work on learning residual policies over hand-defined ones for solving long horizon (Silver et al., 2018) and complex control tasks (Johannink et al., 2019). Similarly, our approaches, BRQN and BRPO in particular, initialize with experts and learns to improve via Bayesian reinforcement learning.

Recent work by Cheng et al. (2020) introduces an imitation learning algorithm to learn from multiple suboptimal experts. While the objective is different from BRQN and BRPO in that our work focuses on learning the Bayes-optimality gap, the proposed approach can be used to construct a lightweight, high-performing ensemble of experts for BRPO.

2.7 Bayesian Meta-reinforcement Learning

Meta-reinforcement learning (MRL) approaches train sample-efficient learners by exploiting structure common to a distribution of MDPs. For example, MAML (Finn et al., 2017) trains gradient-based learners while RL2 (Duan et al., 2016b) trains memory-based learners. While meta-supervised learning has well established Bayesian roots (Baxter, 1998, 2000), it was only recently that meta-reinforcement learning was strongly tied to Bayesian Reinforcement Learning (BRL) (Ortega et al., 2019; Rabinowitz, 2019). Our work is more closely related to Bayesian MRL approaches. MAML-HB (Grant et al., 2018) casts MAML as hierarchical Bayes and improves posterior estimates. BMAML (Yoon et al., 2018) uses non-parametric variational inference to improve posterior estimates. PLATIPUS (Finn et al., 2018) learns a parameter distribution instead of a fixed parameter. PEARL (Rakelly et al., 2019) learns a data-driven Bayes filter across tasks. In contrast to these approaches, BRQN and BRPO use experts at test time, learning only to optimally correct them.

2.8 Bayesian Reinforcement Learning and Posterior Sampling

Posterior Sampling Reinforcement Learning (PSRL) (Osband et al., 2013) is an online RL algorithm that maintains a posterior over latent MDP parameters ϕ. However, the problem setting it considers and how it uses this posterior are quite different than what we consider here.

Our algorithms, BPO, BRQN, and BRPO in particular, focus on scenarios where the agent can only interact with the test MDP for a *single episode*; latent parameters are resampled for each episode. The PSRL regret analysis assumes MDPs with finite horizons and *repeated episodes* with the same test MDP, i.e. the latent parameters are fixed for all episodes.

Before each episode, PSRL samples an MDP from its posterior over MDPs, computes the optimal policy for the sampled MDP, and executes it on the fixed test MDP. Its posterior is updated after each episode, concentrating the distribution around the true latent parameters. During this exploration period, it can perform arbitrarily poorly. Furthermore, sampling a latent MDP from the posterior determinizes the parameters; as a result, there is no uncertainty in the sampled MDP, and the resulting optimal policies that are executed will never take sensing actions.

2.8.1 The Gap Between Bayes Optimality and Posterior Sampling

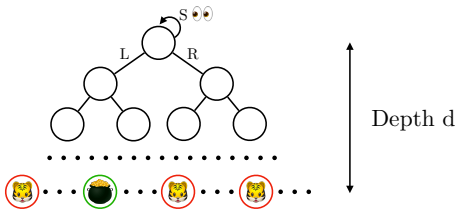


Figure 2.1: A tree-like MDP that highlights the distinction between BRL and PSRL.

Consider a deterministic tree-like MDP (Figure 2.1). Reward is received only at the terminal leaf states: one leaf contains a pot of gold ($R = 100$) and all others contain a dangerous tiger ($R = -10$). All non-leaf states have two actions, go left (L) and go right (R). The start state additionally has a sense action (S), which is costly ($R = -0.1$) but reveals the exact location of the pot of gold. Both algorithms are initialized with a uniform prior over the $N = 2^d$ possible MDPs (one for each possible location of the pot of gold).

To contrast the performance of the Bayes-optimal policy and posterior sampling, we consider the multi-episode setting where the agent repeatedly interacts with the same MDP. The MDP is sampled once from the uniform prior, and agents interact with it for T episodes. This is the setting typically considered by posterior sampling (PSRL) (Osband et al., 2013).

Before each episode, PSRL samples an MDP from its posterior over MDPs, computes the optimal policy, and executes it. After each episode, it updates the posterior and repeats. Sampling from the posterior determinizes the underlying latent parameter. As a result, PSRL will never produce sensing actions to reduce uncertainty about that parameter because the sampled MDP has no uncertainty. More concretely, the optimal policy for each tree MDP is to navigate directly to the gold *without sensing*; PSRL will never take the sense action. Thus, PSRL makes an average of $\frac{N-1}{2}$ mistakes before sampling the correct pot of gold location and the cumulative reward over T episodes is

$$-10 \underbrace{\left(\tfrac{N-1}{2}\right)}_{\text{mistakes}} + 100 \underbrace{\left(T - \tfrac{N-1}{2}\right)}_{\text{pot of gold}} \tag{2.1}$$

In the first episode, the Bayes-optimal first action is to sense. All subsequent actions in this first episode navigate toward the pot of gold, for an episode reward of $-0.1 + 100$. In the subsequent $T - 1$ episodes, the Bayes-optimal policy navigates directly toward the goal without needing to sense, for a cumulative reward of $100T - 0.1$. The performance gap between the Bayes-optimal policy and posterior sampling grows exponentially with depth of the tree d.

Chapter 3

GP-ITERATIVE LINEAR QUADRATIC CONTROL

In this chapter, we consider scenarios in which the agent can afford some real-world failures. We make two simplifying assumptions from the general Bayesian RL setup. First, we assume that the latent model does not change over episodes. Second, we assume that the latent model is approximately centered around a nominal model. These assumptions are particularly useful when the robot has unmodeled components such as wear and tear on its joints, which do not change quickly.

With these assumptions, we propose a data-efficient policy search algorithm, GP-Iterative Linear Quadratic Control (GP-ILQG), which iteratively improves initial policy by combining an optimal control technique with data-driven model learning. GP-ILQG is capable of correcting significant model errors and quickly converges to an optimal policy for the real-world model. Importantly, it incorporates model uncertainty into its policy and takes conservative actions when the uncertainty is high.

3.1 Introduction

As we aim to control more complex robotic systems autonomously, simulators are being more frequently used for training in model-based reinforcement learning (Kober et al., 2013). A simulator allows us to explore various policies without damaging the robot, and is also capable of generating a large amount of synthetic data with little cost and time.

However, we often observe that simulator-based policies perform poorly in real world, due to model discrepancy between the simulation and the real world. This discrepancy arises from two fundamental challenges: (1) system identification to match the simulation model with the real world requires the exploration of a large state space at the risk of damaging

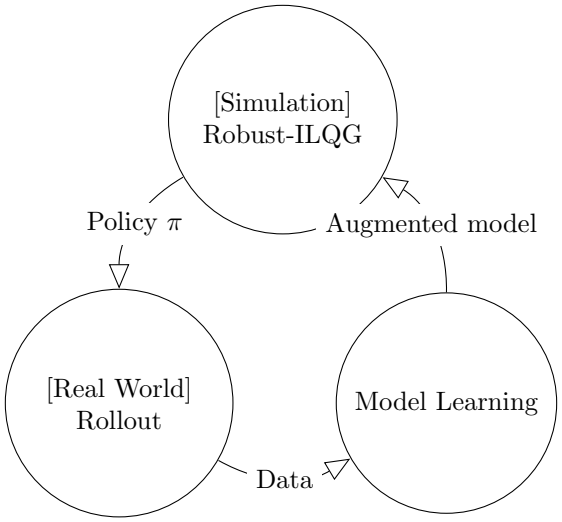


Figure 3.1: GP-ILQG overview

the robot, and (2) even with good system identification, there is still discrepancy due to the limitations of a simulator's ability to render real-world physics.

Stochastic optimal control algorithms attempt to partially address this issue by artificially injecting noise into the simulation during training (Huh & Todorov, 2009; Wang et al., 2010), or by explicitly modeling multiplicative noise (Todorov & Li, 2005; Stoorvogel, 1993).

If the task domain is predefined, exploration can be limited to task-specific trajectories, and system identification can be coupled with trajectory optimization (Tan et al., 2016). Some recent works have suggested policy training with multiple models, which results in a policy robust to model variance (Mordatch et al., 2016; Boeing & Bräunl, 2012). While these methods have shown some successful results, they are still limited by the expressiveness of the simulator's model. If the true model is outside of the simulator's model space, little can be guaranteed.

Thus, although these algorithms produce more robust policies, they fail to address the fundamental issue: there is unknown but structured model discrepancy between the simulation and the real world.

Our approach addresses both model bias and multiplicative noise. It is based on the

following key insight:

> *Improving upon known model prior and incorporating uncertainty in policy search enable data-efficient learning and robust control.*

Our algorithm iterates over simulation-based optimal control, real-world data collection, and model learning, as illustrated in Figure 3.1. Starting from a potentially incorrect model given by the simulator, we obtain a control policy, with which we collect data in the real world. This data feeds into model learning, during which we correct model bias and estimate our uncertainty of the correction. Both the correction and its uncertainty are incorporated into computing a robust optimal control policy, which then gets used to collect more data.

Our approach improves any simulator beyond the scope of its model space to match real-world observations and produces an optimal control policy robust to model uncertainty and multiplicative noise. The improved simulator uses previous real-world observations to infer the true model when it explores previously visited space, but when it encounters a new region, it relies on the simulator's original model. Due to this hybrid nature, our algorithm shows faster convergence to the optimal policy than a pure data-driven approach (Pan & Theodorou, 2014) or a pure simulation-based approach. Moreover, as it permanently improves the simulator, it shows even faster convergence in new tasks in similar task domain.

3.2 Approach

We work with a deterministic nonlinear dynamical system. We assume that we have only partial knowledge of the true system, which we represent as f. The unknown component is represented as ϵ. The dynamic system can then be written as

$$s' = f(s, a) + \epsilon(s, a) \tag{3.1}$$

where s is the state and a is the action.

Given a trajectory $\{s_0, a_0, \cdots s_T\}$, the total cost is given as $J(s_0) = \mathbb{E}\left[l_T(s_T) + \sum_{i=0}^{T-1} l(s_i, a_i)\right]$, where l_T is the final cost and l is the running cost. The objective is to fine a policy that

Algorithm 1 GP-ILQG

Require: Nominal model f, GP prior ϵ, Data D

$\pi \leftarrow \texttt{ILQG}(f+\epsilon)$
while π not converged **do**
  Collect $\{(s, a, s' - f(s, a)\}$ from trajectories executed with π
  $D \leftarrow D \cup \{(s, a, s' - f(s, a)\}$
  $\epsilon \leftarrow \texttt{GP}(D)$
  $\pi \leftarrow \texttt{ILQG}(f+\epsilon)$

minimize the cost function. We can find such a policy using optimal control methods such as ILQG if the dynamics is known.

We approximate the unknown component from data. From real-world trajectories, we collect a set of $(s, a, s' - f(s, a))$ tuples, where the last element corresponds to the unknown component, ϵ. We use a Gaussian Process (GP) (Rasmussen, 2006) to estimate this. Because GP provides state-action dependent Gaussian distribution, this formulation makes Equation 3.1 equivalent to a stochastic dynamical system used in ILQG, except that the Gaussian component now captures uncertainty instead of stochasticity. We refer to Todorov & Li (2005) for the full derivation of ILQG.

Our approach is summarized in Algorithm 1. Initially, we start by using ILQG to obtain a locally optimal policy for $f+\epsilon$, where the unknown component is a zero-mean GP prior. Then, with real-world data from the executed policy, we train a GP to estimate ϵ. We run ILQG again, but with the updated dynamics. This process is repeated until the policy converges.

One major advantage of our approach is that it is straightforward to use the improved model for a new task. Given observations from previous tasks, Algorithm 1 can start with a better estimate of ϵ. This results in a reduced learning time for subsequent tasks, as we verify empirically in the following section.

3.2.1 Gaussian Process with the Simulator as a Mean Function

Gaussian Process Regression is a nonlinear regression technique that has been successfully used in many model-based reinforcement learning approaches (Deisenroth & Rasmussen, 2011;

Pan & Theodorou, 2014). A Gaussian Process is a stochastic process in which any finite subset is jointly Gaussian. Given a set of inputs and corresponding outputs $\{x_i, y_i\}_{i=1}^n$ the distribution of $\{y_i\}_{i=1}^n$ is defined by the covariance given by a kernel function $k(x_i, x_j)$.

We use a variant of Gaussian Processes that uses a nonzero mean function. With $f : X \to Y$ as the mean function, the prediction for test input x becomes the following:

$$\mathbb{E}[y] = f(x) + k_{xX}^\top K^{-1}(Y - f(X))$$
$$var(y) = k_{xx} - k_{xX}^\top K^{-1} k_{xX}$$

where k_{xX} is the covariance between test input x and training set X, K is the covariance matrix of among the elements of X. In this formulation, the GP provides a posterior distribution given $f(x)$ and observations.

Using the simulator as the mean function for a Gaussian Process allows us to combine the simulator and real-world observations smoothly. Near previous observations, the simulator's prediction is corrected to match the target. When the test input is far from the previous observations, the predictions resort to the simulator.

As we have both s and a as the input, we define $\widetilde{s} = [s^\top a^\top]^\top$ to be the input and δs to be the output, and use f as the mean function. Then, the GP serves to correct the error, $\delta s - f\delta t$. We use the Automatic Relevance Determination squared exponential kernel:

$$k(\widetilde{s}_i, \widetilde{s}_j) = \sigma_f^2 \exp(-(\widetilde{s}_i - \widetilde{s}_j)^\top \Lambda^{-1} (\widetilde{s}_i - \widetilde{s}_j))$$

where σ_f^2 is the signal variance and Λ controls the characteristic length of each input dimension. These hyperparameters are optimized to maximize log-likelihood of data using conventional optimization methods. For multiple dimensions of Y, we train one GP per dimension and treat each dimension to be independent.

3.3 Experimental Results

We consider two simulated tasks: cart-pole swing-up and quadrotor control. For each task, one set of model parameters was used as the "simulator", and another set of model parameters

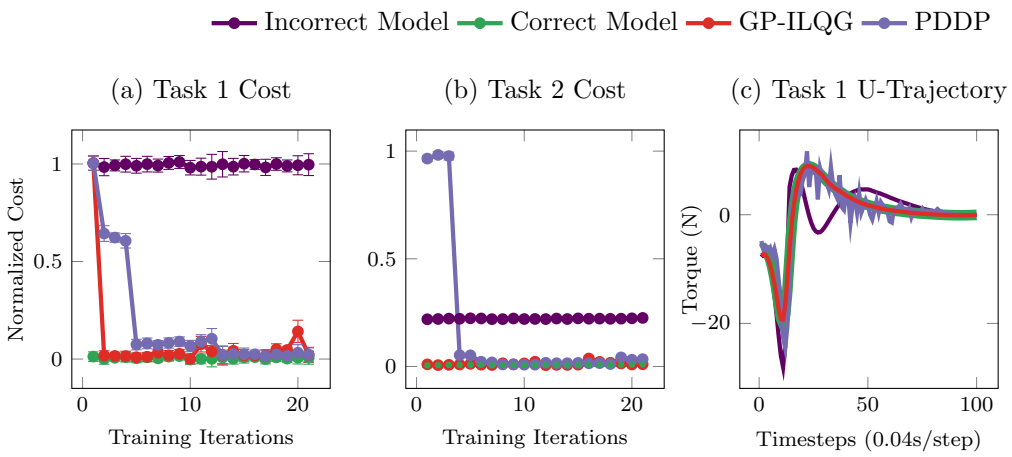


Figure 3.2: Cartpole swing-up.

was used as the "real-world." Multiplicative noise was added to dynamics and Gaussian noise was added to observation. We compare GP-ILQG's performance with three optimal control algorithms: (1) ILQG using the "real-world" model, (2) ILQG using the incorrect "simulator" model, (3) Probabilistic DDP (PDDP) (Pan & Theodorou, 2014), which is a variant of ILQG that relies only on data. For both GP-ILQG and PDDP, the same set of data and same GP implementation were used at each iteration.

As data gets large, computing Gaussian Process becomes computationally expensive, and thus it is common to use a subset of the data for computing the mean and covariance. In our experiments, we keep all observations and uniformly sub-sample 300 data points at each iteration[1]. GP hyperparameters are optimized with the Gaussian Process for Machine Learning Toolbox (Rasmussen & Nickisch, 2010). If the learner's prediction error for a validation set is higher than that of its previous learner, we re-sample and re-train.

[1]For better results, more advanced techniques such as Sparse Pseudo-input GP (Snelson & Ghahramani, 2006) or Sparse Spectral Gaussian Process (Quiñonero-Candela et al., 2010) can be used.

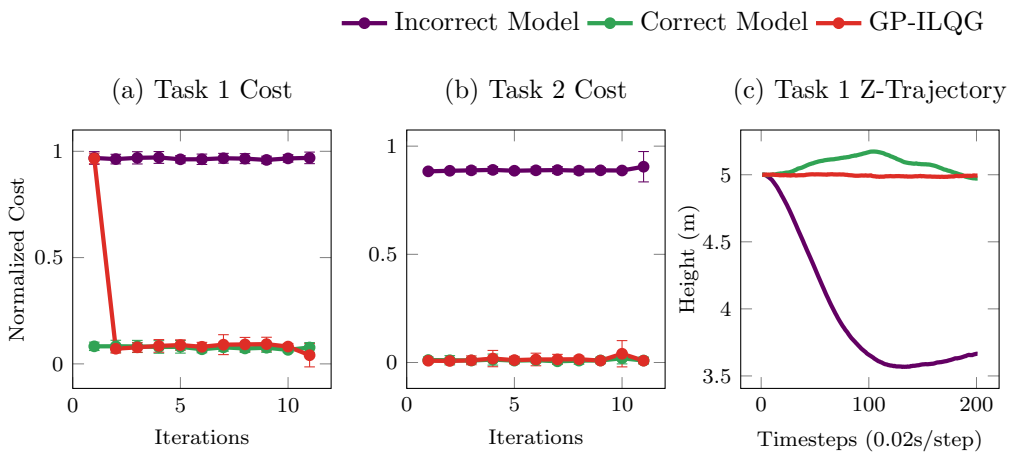


Figure 3.3: Quadrotor control

3.3.1 Cart-Pole Swing-Up

In the cart-pole problem, the state is defined as $[x, \dot{x}, \theta, \dot{\theta}]$, where x is the position of the cart along the $x-$axis and θ is the angle of the pole from the vertical downright position. Control input is the x-directional force (N). Our model for the mass of cart and pole is 1kg each. In the simulator we use 1m pole, while the real world model uses 1.3m one.

We run two tasks in this experiment. In the first task, the initial state is $[0, \pi/4, 0, 0]$. Figure 3.2(a) is the normalized cost for the first task. While both GP-ILQG and PDDP converges to the optimal performance, GP-ILQG is converges much quickly, within the first 2 iterations.

The difference between GP-ILQG and PDDP is more noticeable in the second task (Figure 3.2(b)), which starts from a different initial state $[0, -\pi/4, 0, 0]$. Both GP-ILQG and PDDP use the learner used in the previous task, but the initial cost for PDDP is significantly higher than GP-ILQG. We believe that this is because both algorithms explore an unexplored region in the first few iterations. While GP-ILQG relies on the simulator's inaccurate model in the unexplored region, PDDP has no information to make meaningful advancement until enough data is collected.

What is more noticeable is the improved performance of GP-ILQG over the simulator-based ILQG. The latter's suboptimal policy results in significantly higher cost in general. Figure 3.2(c) shows the control sequences generated by the final policies of the four algorithms. GP-ILQG's control sequence is almost identical to the optimal sequence, and PDDP closely follows the optimal sequence as well. However, the simulator-based ILQG's control sequence is quite different due to its incorrect model.

3.3.2 Quadrotor

We use the quadrotor model introduced in van den Berg (2016). The model has 12-dimensional state, $x = [p, v, r, w]^\top$, where p (m) and v (m/s) refers to the quadrotor's position and velocity in 3D space, r is orientation (rotation about axis r by angle $\|r\|$), and w (rad/s) is angular velocity. It has 4 control inputs, $u = [u_1, u_2, u_3, u_4]^\top$, which represent the force (N) exerted by the four rotors. The dynamics is given as the following:

$$
\begin{aligned}
\dot{p} &= v \\
\dot{v} &= -ge_3 + (\sum u_i)\exp([r]e_3 - k_v v)/m \\
\dot{r} &= w + \frac{1}{2}[r]w + (1 - \frac{1}{2}\|r\|/\tan(\frac{1}{2}\|r\|)[r]^2/\|r\|^2 \\
\dot{w} &= J^{-1}(\rho(u_2 - u_4)e_1 + \rho(u_3 - u_1)e_2 + k_m(u_1 - u_2 + u_3 - u_4)e_3 - [w]Jw)
\end{aligned}
$$

where e_i are the standard basis vectors, $g = 9.8m/s^2$ is gravity, k_v is the drag coefficient of rotors, m (kg) is the mass, J (kg m^2) is the moment of inertia matrix, and ρ (m) is the distance between the center of mass and the center of rotors, and k_m is a constant relating the force of rotors to its torque. $[\cdot]$ refers to the skew-symmetric cross product. The model parameters used are summarized in Table 3.1. The real-world model is 40% heavier than the simulator's model.

We evaluate the performance of two tasks. The quadrotor starts at a position near $(0m, 0m, 5m)$ with zero velocity. In the first task, the goal is to move the quadrotor forward to $(4m, 0m, 5m)$ in 4 seconds. The second task is to drive the quadrotor to $(2m, 1m, 7m)$ in 4

	Simulator	Real World
k_v	0.15	0.15
k_m	0.025	0.025
m	**0.5**	**0.7**
J	0.05 I	0.05 I
ρ	0.17	0.17

Table 3.1: k_v is a constant relating the velocity to an opposite force, caused by rotor drag and induced inflow. m (kg) is the mass, J (kg m^2) is the moment of inertia matrix, ρ (m) is the distance between the center of mass and the center of the rotors.

seconds. The cost function was set to track a straight line from the initial position to the goal, with higher cost for maintaining the height.

In this experiment, we were not able to run PDDP to convergence with the same data used in GP-ILQG. We believe that this arises from the same problem we saw in the second task of cart-pole: PDDP has insufficient data to infer the unexplored state-space. We note that the original PDDP algorithm requires random trajectories as its initial data set instead of random variations of a single nominal trajectory. While our experiment does not indicate that PDDP is incapable of this task[2], it highlights our algorithm's data efficiency. Even with the small set of task-specific data, GP-ILQG converges in the first two iterations in the initial task (Figure 3.3(a) and converges immediately to the optimal policy in the second task (Figure 3.3(b)). Figure 3.3(c) compares the trajectories generated by the three algorithms. It shows that while our algorithm closely tracks the desired height, the simulator's suboptimal controller fails to recover from the vertical drop due to its incorrect mass model.

[2]A similar experiment with quadrotor control was shown to be successful in Pan et al. (2015).

3.4 Discussion

GP-ILQG combines real-world data with a simulator's model to improve real-world performance of simulation-based optimal control. Our approach uses a Gaussian Process to correct a simulator's nonlinear model bias beyond the scope of its model space while incorporating the uncertainty of our estimate in computing a robust optimal control policy. Through simulated experiments, we have shown that our approach converges to the optimal performance within a few iterations and is capable of generalizing the learned dynamics for new tasks.

Although our algorithm is capable of correcting significant model errors, it is limited by the quality of the initial policy based on the simulator's incorrect model. For example, the simulator's model can be sufficiently different from the true model such that the initial policy results in catastrophic damage to the robot. Our algorithm is incapable of measuring this initial uncertainty, although it can be improved by providing an initial set of expert-generated trajectories.

Chapter 4

BAYESIAN POLICY OPTIMIZATION

In this chapter, we formulate the problem of model uncertainty as Bayes-Adaptive Markov Decision Processes (BAMDPs). In a BAMDP, the agent maintains a posterior distribution (belief) over latent model parameters, and maximizes its long-term reward given the belief.

Our algorithm, Bayesian Policy Optimization, builds on batch policy optimization algorithms to learn a universal policy that navigates the exploration-exploitation trade-off to maximize the Bayesian value function. Our method significantly outperforms algorithms that address model uncertainty without explicitly reasoning about belief distributions and is competitive with state-of-the-art Partially Observable Markov Decision Process solvers.

4.1 Introduction

The Bayes-Adaptive Markov Decision Process (BAMDP) framework (Ghavamzadeh et al., 2015) elegantly captures the exploration-exploitation dilemma that the agent faces. Here, the agent maintains a *belief*, which is a posterior distribution over the latent parameters ϕ given a history of observations. A BAMDP can be cast as a Partially Observable Markov Decision Process (POMDP) (Duff & Barto, 2002) whose state is (s, ϕ), where s corresponds to the observable world state. By planning in the belief space of this POMDP, the agent balances explorative and exploitative actions. In this chapter, we focus on BAMDP problems in which the latent parameter space is either a *discrete finite set* or a *bounded continuous set* that can be approximated via *discretization*. For this class of BAMDPs, the belief is a categorical distribution, allowing us to represent it using a vector of weights.

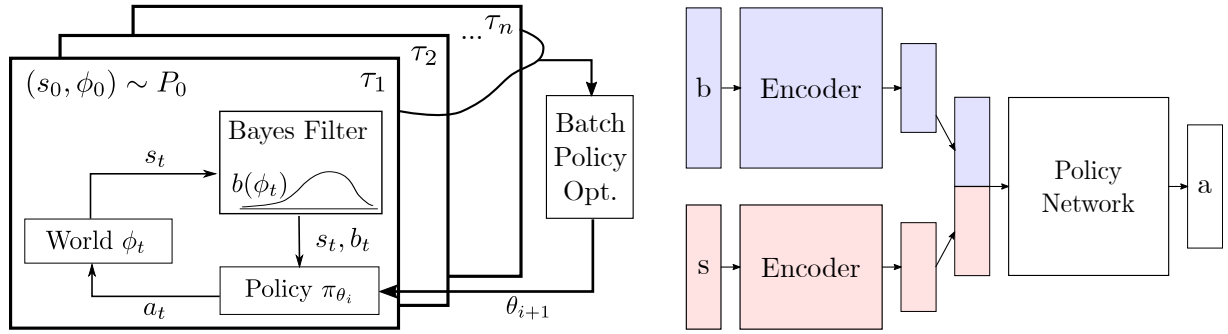
The core problem for BAMDPs with continuous state-action spaces is how to explore the reachable belief space. In particular, discretizing the latent space can result in an arbitrarily

large belief vector, which causes the belief space to grow exponentially. Approximating the value function over the reachable belief space can be challenging: although point-based value approximations (Kurniawati et al., 2008; Pineau et al., 2003b) have been largely successful for approximating value functions of discrete POMDP problems, these approaches do not easily extend to continuous state-action spaces. Monte-Carlo Tree Search approaches (Silver & Veness, 2010; Guez et al., 2012) are also prohibitively expensive in continuous state-action spaces: the width of the search tree after a single iteration is too large, preventing an adequate search depth from being reached.

Our key insight is that we can bypass learning the value function and directly learn a policy that maps beliefs to actions by leveraging the latest advancements in batch policy optimization algorithms (Schulman et al., 2015, 2017). Inspired by previous approaches that train learning algorithms with an ensemble of models (Rajeswaran et al., 2017; Yu et al., 2017), we examine model uncertainty through a BAMDP lens. Although our approach provides only locally optimal policies, we believe that it offers a practical and scalable solution for continuous BAMDPs.

Our method, Bayesian Policy Optimization (BPO), is a batch policy optimization method which utilizes a black-box Bayesian filter and augmented state-belief representation. During offline training, BPO simulates the policy on multiple latent models sampled from the source distribution (Figure 4.1a). At each simulation timestep, it computes the posterior belief using a Bayes filter and inputs the state-belief pair (s, b) to the policy. Our algorithm only needs to update the posterior along the simulated trajectory in each sampled model, rather than branching at each possible action and observation as in MCTS-based approaches.

Our key contribution is the following. We introduce a Bayesian policy optimization algorithm to learn policies that directly reason about model uncertainty while maximizing the expected long-term reward (Section 4.3). To address the challenge of large belief representations, we introduce two encoder networks that balance the size of belief and state embeddings in the policy network (Figure 4.1b). In addition, we show that our method, while designed for BAMDPs, can be applied to continuous POMDPs when a compact belief representation is



(a) Training procedure

(b) Network structure

Figure 4.1: An overview of Bayesian Policy Optimization. The policy is simulated on multiple latent models. At each timestep of the simulation, a black-box Bayes filter updates the posterior belief and inputs the state-belief to the policy (Figure 4.1a). Belief (b) and state (s) are independently encoded before being pushed into the policy network (Figure 7.2)

available (Section 4.3.2). Through experiments on classical POMDP problems and BAMDP variants of OpenAI Gym benchmarks, we show that BPO significantly outperforms algorithms that address model uncertainty without explicitly reasoning about beliefs and is competitive with state-of-the-art POMDP algorithms (Section 4.4).

4.2 Preliminaries: Bayesian Reinforcement Learning

The Bayes-Adaptive Markov Decision Process framework (Duff & Barto, 2002; Ross et al., 2008; Kolter & Ng, 2009) was originally proposed to address uncertainty in the transition function of an MDP. The uncertainty is captured by a latent variable, $\phi \in \Phi$, which is either directly the transition function, e.g. $\phi_{sas'} = T(s, a, s')$, or is a parameter of the transition, e.g. physical properties of the system. The latent variable is either fixed or has a known transition function. We extend the previous formulation of ϕ to address uncertainty in the reward function as well.

Formally, a BAMDP is defined by a tuple $\langle S, \Phi, A, T, R, P_0, \gamma \rangle$, where S is the observable

state space of the underlying MDP, Φ is the latent space, and A is the action space. T and R are the parameterized transition and reward functions, respectively. The transition function is defined as: $T(s, \phi, a', s', \phi') = P(s', \phi'|s, \phi, a') = P(s'|s, \phi, a')P(\phi'|s, \phi, a', s')$. The initial distribution over (s, ϕ) is given by $P_0 : S \times \Phi \to \mathbb{R}^+$, and γ is the discount.

Bayesian Reinforcement Learning (BRL) considers the long-term expected reward with respect to the uncertainty over ϕ rather than the true (unknown) value of ϕ. The uncertainty is represented as a *belief distribution* $b \in B$ over latent variables ϕ. BRL maximizes the following Bayesian value function, which is the expected value *given the uncertainty*:

$$\begin{aligned} V_\pi(s, b) &= R(s, b, a') + \gamma \sum_{s' \in S, b' \in B} P(s', b'|s, b, a')V_\pi(s', b') \\ &= R(s, b, a') + \gamma \sum_{s' \in S, b' \in B} P(s'|s, b, a')P(b'|s, b, a', s')V_\pi(s', b') \end{aligned} \tag{4.1}$$

where the action is $a' = \pi(s, b)$.[1]

The Bayesian reward and transition functions are defined in expectation with respect to ϕ: $R(s, b, a') = \sum_{\phi \in \Phi} b(\phi)R(s, \phi, a')$, $P(s'|s, b, a') = \sum_{\phi \in \Phi} b(\phi)P(s'|s, \phi, a')$. The belief distribution can be maintained recursively, with a black-box Bayes filter performing posterior updates given observations. We describe how to implement such a Bayes filter in Section 4.3.1.

The use of (s, b) casts the partially observable BAMDP as a fully observable MDP in belief space, which permits the use of any policy gradient method. We highlight that a reactive Bayesian policy in belief space is equivalent to a policy with memory in observable space (Kaelbling et al., 1998). In our work, the complexity of memory is delegated to a Bayes filter that computes a sufficient statistic of the history.

In partially observable MDPs (POMDPs), the states can be observed only via a noisy observation function. Mixed-observability MDPs (MOMDPs) (Ong et al., 2010) are similar to BAMDPs: their states are (s, ϕ), where s is observable and ϕ is latent. Although any BAMDP problem can be cast as a POMDP or a MOMDP problem (Duff & Barto, 2002),

[1] The state space S can be either discrete or continuous. The belief space B is always continuous, but we use $\sum$ notation for simplicity.

Algorithm 2 Bayesian Policy Optimization

Require: Bayes filter $\psi(\cdot)$, initial belief $b_0(\phi)$, P_0, policy π_{r_0}, horizon T, $n_{\text{itr}}, n_{\text{sample}}$

for $i = 1, 2, \cdots, n_{\text{itr}}$ **do**
  for $n = 1, 2, \cdots, n_{\text{sample}}$ **do**
    Sample latent MDP M: $(s_0, \phi_0) \sim P_0$
    $\tau_n \leftarrow$ `Simulate`$(\pi_{r_{i-1}}, b_0, \psi, M, T)$
  Update policy: $r_i \leftarrow$ `BatchPolicyOptimization`$(r_{i-1}, \{\tau_1, \cdots, \tau_{n_{\text{sample}}}\})$
return $\pi_{r_{best}}$

procedure SIMULATE(π, b_0, ψ, M, T)
  for $t = 1, \cdots, T$ **do**
    $a_t \leftarrow \pi(s_{t-1}, b_{t-1})$
    Execute a_t on M, observing r_t, s_t
    $b_t \leftarrow \psi(s_{t-1}, b_{t-1}, a_t, s_t)$
  return $(s_0, b_0, a_1, r_1, s_1, b_1, \cdots, a_H, r_H, s_H, b_H)$

the source of uncertainty in a BAMDP usually comes from the transition function, not the unobservability of the state as it does with POMDPs and MOMDPs.

4.3 Bayesian Policy Optimization

We propose Bayesian Policy Optimization, a simple policy gradient algorithm for BAMDPs (Algorithm 2). The agent learns a stochastic Bayesian policy that maps a state-belief pair to a probability distribution over actions $\pi : S \times B \rightarrow P(A)$. During each training iteration, BPO collects trajectories by simulating the current policy on several MDPs sampled from the prior distribution. During the simulation, the Bayes filter updates the posterior belief distribution at each timestep and sends the updated state-belief pair to the Bayesian policy. By simulating on MDPs with different latent variables, BPO observes the evolution of the state-belief throughout multiple trajectories. Since the state-belief representation makes the partially observable BAMDP a fully observable Belief-MDP, any batch policy optimization

algorithm (e.g., Schulman et al. (2015, 2017)) can be used to maximize the Bayesian Bellman equation (Equation 4.1).

One key challenge is how to represent the belief distribution over the latent state space. To this end, we impose one mild requirement, i.e., that the belief can be represented with a fixed-size vector. For example, if the latent space is discrete, we can represent the belief as a categorical distribution. For continuous latent state spaces, we can use Gaussian or a mixture of Gaussian distributions. When such specific representations are not appropriate, we can choose a more general uniform discretization of the latent space.

Discretizing the latent space introduces the curse of dimensionality. An algorithm must be robust to the size of the belief representation. To address the high-dimensionality of belief space, we introduce a new policy network structure that consists of two separate networks to independently encode state and belief (Figure 7.2). These encoders consist of multiple layers of nonlinear (e.g., ReLU) and linear operations, and they output a compact representation of state and belief. We design the encoders to yield outputs of the same size, which we concatenate to form the input to the policy network. The encoder networks and the policy network are *jointly* trained by the batch policy optimization. Our belief encoder achieves the desired robustness by learning to compactly represent arbitrarily large belief representations. In Section 7.3, we empirically verify that the separate belief encoder makes our algorithm more robust to large belief representations (Figure 4.2b).

As with most policy gradient algorithms, BPO provides only a locally optimal solution. Nonetheless, it produces robust policies that scale to problems with high-dimensional observable states and beliefs (see Section 7.3).

4.3.1 Bayes Filter for Bayesian Policy Optimization

Given an initial belief b_0, a Bayes filter recursively performs the posterior update:

$$b'(\phi'|s, b, a', s') = \eta \sum_{\phi \in \Phi} b(\phi) T(s, \phi, a', s', \phi') \tag{4.2}$$

where η is the normalizing constant, and the transition function is defined as $T(s, \phi, a', s', \phi') = P(s', \phi'|s, \phi, a') = P(s'|s, \phi, a')P(\phi'|s, \phi, a', s')$. At timestep t, the belief $b_t(\phi_t)$ is the posterior distribution over Φ given the history of states and actions, $(s_0, a_1, s_1, ..., s_t)$. When ϕ corresponds to physical parameters for an autonomous system, we often assume that the latent states are fixed.

Our algorithm utilizes a black-box Bayes filter to produce a posterior distribution over the latent states. Any Bayes filter that outputs a fixed-size belief representation can be used; for example, we use an extended Kalman filter to maintain a Gaussian distribution over continuous latent variables in the `LightDark` environment in Section 7.3. When such a specific representation is not appropriate, we can choose a more general discretization of the latent space to obtain a computationally tractable belief update.

For our algorithm, we found that uniformly discretizing the range of each latent parameter into K equal-sized bins is sufficient. From each of the resulting $K^{|\Phi|}$ bins, we form an MDP by selecting the mean bin value for each latent parameter. Then, we approximate the belief distribution with a categorical distribution over the resulting MDPs.

We approximate the Bayes update in Equation 4.2 by computing the probability of observing s' under each discretized $\phi \in \{\phi_1, \cdots, \phi_{K^{|\Phi|}}\}$ as follows:

$$b'(\phi|s, b, a', s') = \frac{b(\phi)p(s'|s, \phi, a')}{\sum_{i=1}^{K^{|\Phi|}} b(\phi_i)p(s'|s, \phi_i, a')}$$

where the denominator corresponds to η.

As we verify in Section 7.3, our algorithm is robust to approximate beliefs, which allows the use of computationally efficient approximate Bayes filters without degrading performance. A belief needs only to be accurate enough to inform the agent of its actions.

4.3.2 Generalization to POMDP

Although BPO is designed for BAMDP problems, it can naturally be applied to POMDPs. In a general POMDP where state is unobservable, we need only $b(s)$, so we can remove the state encoder network.

Knowing the transition and observation functions, we can construct a Bayes filter that computes the belief b over the hidden state:

$$b'(s') = \psi(b, a', o') = \eta \sum_{s \in S} b(s) T(s, a', s') Z(s, a', o')$$

where η is the normalization constant, and Z is the observation function, $Z(s, a', o') = P(o'|s, a')$, of observing o' after taking action a' at state s. Then, BPO optimizes the following Bellman equation:

$$V_\pi(b) = \sum_{s \in S} b(s) R(s, \pi(b)) + \gamma \sum_{b' \in B} P(b'|b, \pi(b)) V_\pi(b')$$

For general POMDPs with large state spaces, however, discretizing state space to form the belief state is impractical. We believe that this generalization is best suited for beliefs with conjugate distributions, e.g., Gaussians.

4.4 Experimental Results

We evaluate BPO on discrete and continuous POMDP benchmarks to highlight its use of information-gathering actions. We also evaluate BPO on BAMDP problems constructed by varying physical model parameters on OpenAI benchmark problems (Brockman et al., 2016). For all BAMDP problems with continuous latent spaces (`Chain`, `MuJoCo`), latent parameters are sampled in the continuous space in Step 3 of Algorithm 2, regardless of discretization.

We compare BPO to EPOpt and UP-MLE, robust and adaptive policy gradient algorithms, respectively. We also include BPO-, a version of our algorithm without the belief and state encoders; this version directly feeds the original state and belief to the policy network. Comparing with BPO- allows us to better understand the effect of the encoders. For UP-MLE, we use the maximum likelihood estimate (MLE) from the same Bayes filter used for BPO, instead of learning an additional online system identification (OSI) network as originally proposed by UP-OSI. This lets us directly compare performance when a full belief distribution is used (BPO) rather than a point estimate (UP-MLE). For the OpenAI

	Tiger	Chain	LightDark	MuJoCo
Max. episode length	100	100	15	200
Batch size	500	10000	400	500
Training iterations	1000	500	10000	200
Discount (γ)	0.95	1.00	1.00	0.99
Stepsize ($\overline{D}_{KL}$)	0.01	0.01	0.01	0.01
GAE λ	0.96	0.96	0.96	0.96

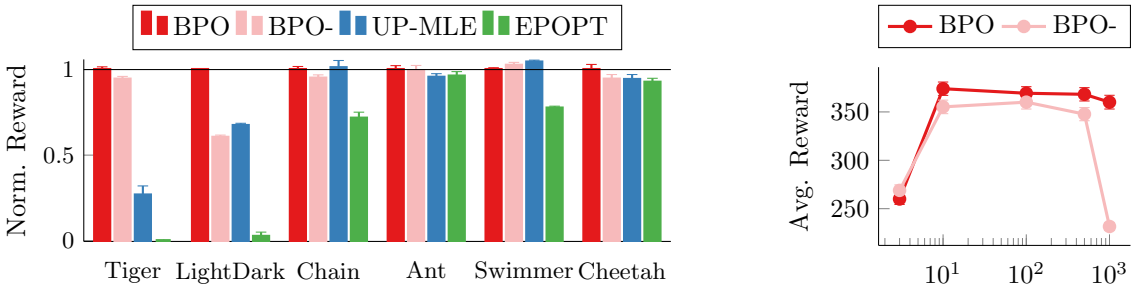
Table 4.1: Training parameters

BAMDP problems, we also compare to a policy trained with TRPO in an environment with the mean values of the latent parameters.

All policy gradient algorithms (BPO, BPO-, EPOpt, UP-MLE) use TRPO as the underlying batch policy optimization subroutine. See Table 4.1 for parameter details. For all algorithms, we compare the results from the seed with the highest mean reward across multiple random seeds. Although EPOpt and UP-MLE are the most relevant algorithms that use batch policy optimization to address model uncertainty, we emphasize that neither formulates the problems as BAMDPs.

The encoder networks and policy network are jointly trained with Trust Region Policy Optimization (Schulman et al., 2015). We used the implementation provided by Duan et al. (2016a) with the parameters listed in Table **??**.

As shown in Figure 7.2, the BPO network's state and belief encoder components are identical, consisting of two fully connected layers with N_h hidden units each and tanh activations ($N_h = 32$ for `Tiger`, `Chain`, and `LightDark`; $N_h = 64$ for `MuJoCo`). The policy network also consists of two fully connected layers with N_h hidden units each and tanh activations. For discrete action spaces (`Tiger`, `Chain`), the output activation is a softmax, resulting in a categorical distribution over the discrete actions. For continuous action spaces (`LightDark`, `MuJoCo`), we represent the policy as a Gaussian distribution.



(a) Benchmark (b) Discretization

Figure 4.2: (a) Comparison of BPO with belief-agnostic, robust RL algorithms. BPO significantly outperforms benchmarks when belief-awareness and explicit information gathering are necessary (`Tiger`, `LightDark`). It is competitive with UP-MLE when passive estimation or universal robustness is sufficient (`Chain`, `MuJoCo`). (b) Scalability of BPO with respect to latent state space discretization for the `Chain` problem.

Figure 4.2a illustrates the normalized performance for all algorithms and experiments. We normalize by dividing the total reward by the reward of BPO. For `LightDark`, which has negative reward, we first shift the total reward to be positive and then normalize.

Tiger (Discrete POMDP). In the `Tiger` problem, originally proposed by Kaelbling et al. (1998), a tiger is hiding behind one of two doors. An agent must choose among three actions: listen, or open one of the two doors; when the agent listens, it receives a noisy observation of the tiger's position. If the agent opens the door and reveals the tiger, it receives a penalty of -100. Opening the door without the tiger results in a reward of 10. Listening incurs a penalty of -1. In this problem, the optimal agent listens until its belief about which door the tiger is behind is substantially higher for one door vs. the other. Chen et al. (2016) frame `Tiger` as a BAMDP problem with two latent states, one for each position of the tiger.

Figure 4.2a demonstrates the benefit of operating in state-belief space when information gathering is required to reduce model uncertainty. Since the EPOPT policy does not maintain a belief distribution, it sees only the most recent observation. Without the full history of

observations, EPOpt learns only that opening doors is risky; because it expects the worst-case scenario, it always chooses to listen. UP-MLE leverages all past observations to estimate the tiger's position. However, without the full belief distribution, the policy cannot account for the confidence of the estimate. Once there is a higher probability of the tiger being on one side, the UP-MLE policy prematurely chooses to open the safer door. BPO significantly outperforms both of these algorithms, learning to listen until it is extremely confident about the tiger's location. In fact, BPO achieves close to the approximately optimal return found by SARSOP (19.0 ± 0.6), a state-of-the-art offline POMDP solver that approximates the optimal value function rather than performing policy optimization (Kurniawati et al., 2008).

Chain (Discrete BAMDP). To evaluate the usefulness of the independent encoder networks, we consider a variant of the `Chain` problem (Strens, 2000). The original problem is a discrete MDP with five states $\{s_i\}_{i=1}^5$ and two actions $\{A, B\}$. Taking action A in state s_i transitions to s_{i+1} with no reward; taking action A in state s_5 transitions to s_5 with a reward of 10. Action B transitions from any state to s_1 with a reward of 2. However, these actions are noisy: in the canonical version of `Chain`, the opposite action is taken with slip probability 0.2. In our variant, the slip probability is uniformly sampled from $[0, 1.0]$ at the beginning of each episode.[2] In this problem, either action provides equal information about the latent parameter. Since active information-gathering actions do not exist, BPO and UP-MLE achieve similar performance.

Figure 4.2b shows that our algorithm is robust to the size of latent space discretization. We discretize the parameter space with 3, 10, 100, 500, and 1000 uniformly spaced samples. At coarser discretizations (3, 10), we see little difference between BPO and BPO-. However, with a large discretization (500, 1000), the performance of BPO- degrades significantly, while BPO maintains comparable performance. The performance of BPO also slightly degrades when the discretization is too fine, suggesting that this level of discretization makes the problem unnecessarily complex. Figure 4.2a shows the best discretization (10).

[2] A similar variant was introduced in Wang et al. (2012).

	BPO	BEETLE	PERSEUS	MCBRL
`Chain-10` (tied)	364.5 ± 0.5	365.0 ± 0.4	366.1 ± 0.2	
`Chain-10` (semitied)	364.9 ± 0.8	364.8 ± 0.3	365.1 ± 0.3	321.6 ± 6.4

Table 4.2: For the `Chain` problem, a comparison of the 95% confidence intervals of average return for BPO vs. other benchmark algorithms. Values for BEETLE, MCBRL, and Perseus are taken from Wang et al. (2012), which does not report MCBRL performance in the "tied" setting.

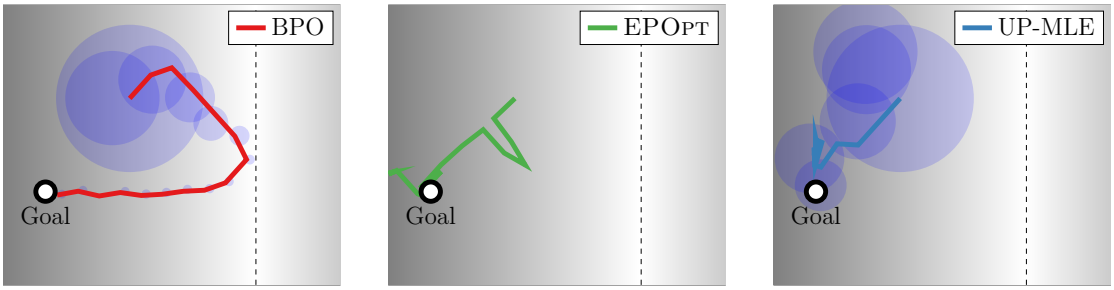


Figure 4.3: Visualization of different algorithms on the `LightDark` environment. The dashed line indicates the light source. Blue circles are one standard deviation for per-step estimates. The BPO policy moves toward the light to obtain a better state estimate before moving toward the goal.

In this discrete domain, we compare BPO to BEETLE (Poupart et al., 2006) and MCBRL (Wang et al., 2012), state-of-the-art discrete Bayesian reinforcement learning algorithms, as well as Perseus (Spaan & Vlassis, 2005), a discrete POMDP solver. In addition to our variant, we consider a more challenging version where the slip probabilities for both actions must be estimated independently. Poupart et al. (2006) refer to this as the "semi-tied" setting; our variant is "tied." BPO performs comparably to all of these benchmarks (Table 4.2).

Light-Dark (Continuous POMDP). We consider a variant of the `LightDark` problem proposed by Platt et al. (2010), where an agent tries to reach a known goal location while being

uncertain about its own position. At each timestep, the agent receives a noisy observation of its location. In our problem, the vertical dashed line is a light source; the farther the agent is from the light, the noisier its observations. The agent must decide either to reduce uncertainty by moving closer to the light, or to exploit by moving from its estimated position to the goal.

After each action, an agent receives a noisy observation of its location, which is sampled from a Gaussian distribution, $o \sim \mathcal{N}([x, y]^\top, w(x))$, where $[x, y]$ is the true location. The noise variance is a function of x and is minimized when $x = 5$: $w(x) = \frac{1}{2}(x-5)^2 + \text{const}$. There is no process noise.

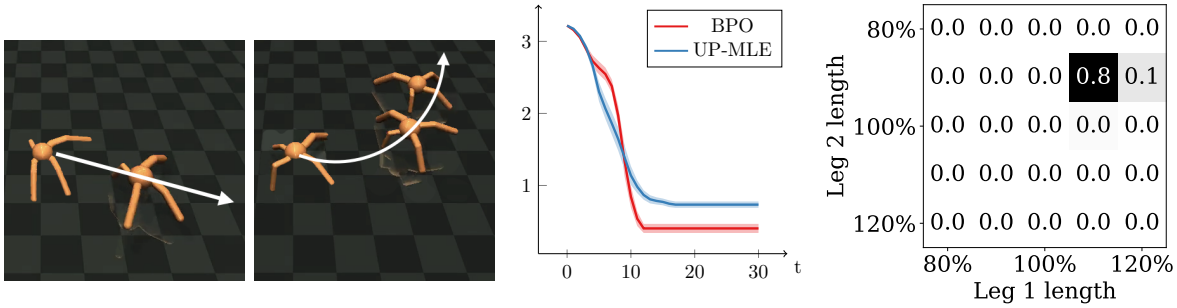
The reward function is $r(s, a) = -\frac{1}{2}(\|s-g\|^2 + \|a\|^2)$, where s is the true agent position and g is the goal position. A large penalty of $-5000\|s_T - g\|^2$ is incurred if the agent does not reach the goal by the end of the time horizon, analogous to the strict equality constraint in the original optimization problem (Platt et al., 2010).

The initial belief is $[x, y, \sigma^2] = [2, 2, 2.25]$. During training, we randomly sample latent start positions from a rectangular region $[2, -2] \times [4, 4]$ and observable goal positions from $[0, \text{-}2] \times [2, 4]$.

This example demonstrates how to apply BPO to general continuous POMDPs (Section 4.3.2). The latent state is the continuous pose of the agent. For this example, we parameterize the belief as a Gaussian distribution and perform the posterior update with an Extended Kalman Filter, as in Platt et al. (2010).

Figure 4.3 compares sample trajectories from different algorithms on the `LightDark` environment. Based on its initial belief, the BPO policy moves toward a light source to acquire less noisy observations. As it becomes more confident in its position estimate, it changes direction toward the light and then moves straight to the goal. Both EPOpt and UP-MLE move straight to the goal without initially reducing uncertainty.

MuJoCo (Continuous BAMDP). Finally, we evaluate the algorithms on three simulated benchmarks from OpenAI Gym (Brockman et al., 2016) using the MuJoCo physics simulator (Todorov et al., 2012): `HalfCheetah`, `Swimmer`, and `Ant`. Each environment has



(a) BPO vs. TRPO (b) Entropy for `Ant` (c) Belief at $t = 20$

Figure 4.4: (a) Comparison of BPO and TRPO trained on the nominal environment for a different environment. The task is to move to the right along the x-axis. However, the model at test time differs from the one TRPO trained with: one leg is 20% longer, another is 20% shorter. (b) Comparison of average entropy per timestep by BPO and UP-MLE. The belief distribution collapses more quickly under the BPO policy. (c) Belief distribution at $t = 20$ during a BPO rollout.

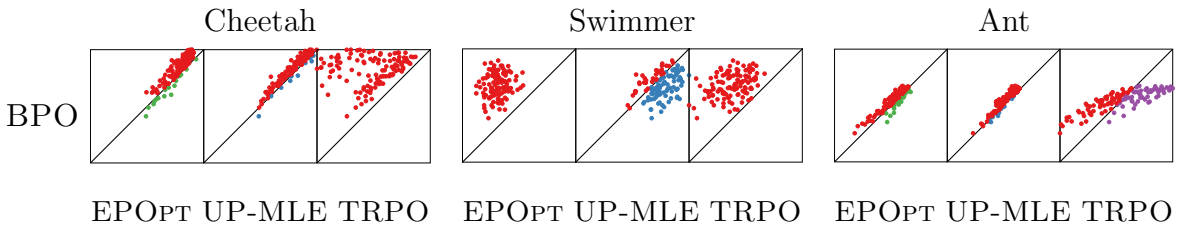


Figure 4.5: Pairwise performance comparison of algorithms on `MuJoCo` BAMDPs. Each point represents an MDP, and its (x, y)-coordinates correspond to the long-term reward by (baseline, BPO). The farther a point is above the line $y = x$, the more BPO outperforms that baseline. Colors indicate which algorithm achieved higher reward: BPO (red), EPOpt (green), UP-MLE (blue), or TRPO (purple).

several latent physical parameters that can be changed to form a BAMDP.

For ease of analysis, we vary two parameters for each environment. For `HalfCheetah`, the front and back leg lengths are varied. For `Ant`, the two front leg lengths are varied. `Swimmer` has four body links, so the first two link lengths vary together according to the first parameter, and the last two links vary together according to the second parameter. We chose to vary link lengths rather than friction or the damping constant because a policy trained on a single nominal environment can perform well across large variations in those parameters. All link lengths vary by up to 20% of the original length.

To construct a Bayes filter, the 2D-parameter space is discretized into a 5×5 grid with a uniform initial belief. We assume Gaussian noise on the observation, i.e. $o = f_\phi(s, a) + w$ with $w \sim \mathcal{N}(0, \sigma^2)$, with ϕ being the parameter corresponding to the center of each grid cell. It typically requires only a few steps for the belief to concentrate in a single cell of the grid, even when a large σ^2 is assumed.

The MuJoCo benchmarks demonstrate the robustness of BPO to model uncertainty. For each environment, BPO learns a universal policy that adapts to the changing belief over the latent parameters.

Figure 4.4 highlights the performance of BPO on `Ant`. BPO can efficiently move to the right even when the model substantially differs from the nominal model (Figure 4.4a). It takes actions that reduce entropy more quickly than UP-MLE (Figure 4.4b). The belief over the possible MDPs quickly collapses into a single bin (Figure 4.4c), which allows BPO to adapt the policy to the identified model.

Figure 4.5 provides a more in-depth comparison of the long-term expected reward achieved by each algorithm. In particular, for the `HalfCheetah` environment, BPO has a higher average return than both EPOPT and UP-MLE for most MDPs. Although BPO fares slightly worse than UP-MLE on `Swimmer`, we believe that this is largely due to random seeds, especially since BPO- matches UP-MLE's performance (Figure 4.2a).

Qualitatively, all three algorithms produced agents with reasonable gaits in most MDPs. We postulate two reasons for this. First, the environments do not require active information-

gathering actions to achieve a high reward. Furthermore, for deterministic systems with little noise, the belief collapses quickly (Figure 4.4b); as a result, the MLE is as meaningful as the belief distribution. As demonstrated by Rajeswaran et al. (2017), a universally robust policy for these problems is capable of performing the task. Therefore, even algorithms that do not maintain a history of observations can perform well.

4.5 Discussion

Bayesian Policy Optimization is a practical and scalable approach for continuous BAMDP problems. We demonstrate that BPO learns policies that achieve performance comparable to state-of-the-art discrete POMDP solvers. They also outperform state-of-the-art robust policy gradient algorithms that address model uncertainty without formulating it as a BAMDP problem. Our network architecture scales well with respect to the degree of latent parameter space discretization due to its independent encoding of state and belief. We highlight that BPO is agnostic to the choice of batch policy optimization subroutine. Although we used TRPO in this work, we can also use more recent policy optimization algorithms, such as PPO (Schulman et al., 2017), and leverage improvements in variance-reduction techniques (Weaver & Tao, 2001).

BPO outperforms algorithms that do not explicitly reason about belief distributions. Our Bayesian approach is necessary for environments where uncertainty must actively be reduced, as shown in Figure 4.2a and Figure 4.3. If all actions are informative (as with `MuJoCo`, `Chain`) and the posterior belief distribution easily collapses into a unimodal distribution, UP-MLE provides a lightweight alternative.

BPO scales to fine-grained discretizations of latent space. However, our experiments also suggest that each problem has an optimal discretization level, beyond which further discretization may degrade performance. As a result, it may be preferable to perform variable-resolution discretization rather than an extremely fine, single-resolution discretization. Adapting iterative densification ideas previously explored in motion planning (Gammell et al., 2015) and optimal control (Munos & Moore, 1999) to the discretization of latent space may

yield a more compact belief representation while enabling further improved performance.

An alternative to the model-based Bayes filter and belief encoder components of BPO is learning to directly map a history of observations to a lower-dimensional belief embedding, analogous to Peng et al. (2018). This would enable a policy to learn a meaningful belief embedding without losing information from our a priori choice of discretization. Combining a recurrent policy for unidentified parameters with a Bayes filter for identified parameters offers an intriguing future direction for research efforts.

Chapter 5

BAYESIAN CPACE

In this chapter, we present a Probably Approximately Correct (PAC) algorithm for Bayes-Adaptive Markov Decision Processes (BAMDPs) in continuous state and action spaces. Our key insight is to compute a near-optimal value function by covering the continuous state-belief-action space with a finite set of representative samples and exploiting the Lipschitz continuity of the value function. We prove the near-optimality of our algorithm and analyze several schemes that boost the algorithm's efficiency. Finally, we empirically validate our approach on discrete and continuous BAMDPs and show that the learned policy has consistently competitive performance against baseline approaches.

5.1 Introduction

Although BRL provides an elegant problem formulation for model uncertainty, PAC algorithms for continuous state and action space BAMDPs have been less explored, limiting possible applications in many robotics problems. In the discrete domain, there exist some efficient online, PAC optimal approaches (Kolter & Ng, 2009; Chen et al., 2016) and approximate Monte-Carlo algorithms (Guez et al., 2012), but it is not straightforward to extend this line of work to the continuous domain. State-of-the-art approximation-based approaches for belief space planning in continuous spaces (Sunberg & Kochenderfer, 2017; Guez et al., 2014) do not provide PAC optimality.

In this chapter, we present a PAC optimal algorithm for BAMDPs in continuous state and action spaces. The key challenge for PAC optimal exploration in continuous BAMDPs is that the same state will not be visited twice, which often renders Monte-Carlo approaches computationally prohibitive, as discussed in (Sunberg & Kochenderfer, 2017). However,

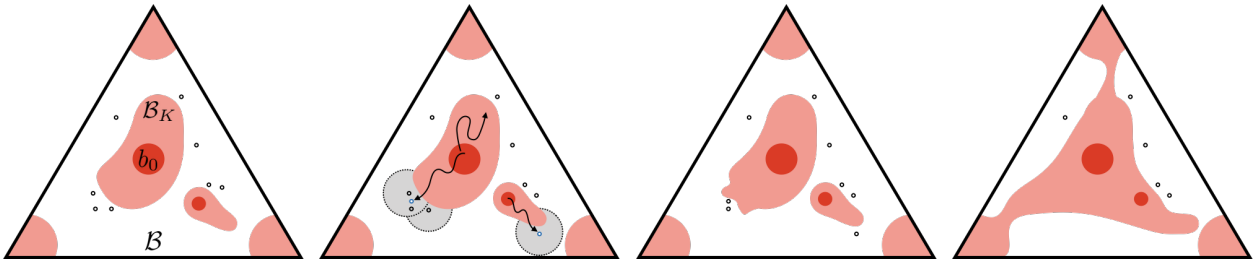


Figure 5.1: The BCPACE algorithm for BAMDPs. The vertices of the belief simplex correspond to the latent MDPs constituting the BAMDP model, for which we can precompute the optimal Q-values. During an iteration of BCPACE, it executes its greedy policy from initial belief b_0, which either never escapes the known belief MDP Belief MDP$_K$ or leads to an unknown sample. Adding the unknown sample to the sample set may expand the known set K and the known belief MDP Belief MDP$_K$. The algorithm terminates when the optimally reachable belief space is sufficiently covered.

if the value function satisfies certain smoothness properties, i.e. Lipschitz continuity, we can efficiently "cover" the reachable belief space. In other words, we leverage the following property:

> A set of representative samples is sufficient to approximate a Lipschitz continuous value function of the reachable continuous state-belief-action space.

Our algorithm, BCPACE (Figure 5.1) maintains an approximate value function based on a set of visited samples, with bounded optimism in the approximation from Lipschitz continuity. At each timestep, it greedily selects an action that maximizes the value function. If the action lies in an underexplored region of state-belief-action space, the visited sample is added to the set of samples and the value function is updated. Our algorithm adopts C-PACE (Pazis & Parr, 2013), a PAC optimal algorithm for continuous MDPs, as our engine for exploring belief space.

We make the following contributions:

1. We present a PAC optimal algorithm for continuous BAMDPs (Section 5.2).

2. We show how BAMDPs can leverage the value functions of latent MDPs to reduce the sample complexity of policy search, without sacrificing PAC optimality (Definitions 5.2.3 and 5.2.4).

3. We prove that Lipschitz continuity of latent MDP reward and transition functions is a sufficient condition for Lipschitz continuity of the BAMDP value function (Lemma 1).

4. Through experiments, we show that BCPACE has competitive performance against state-of-art algorithms in discrete BAMDPs and promising performance in continuous BAMDPs (Section 5.3).

5.2 BCPACE*: Continuous PAC Optimal Exploration in Belief Space*

In this section, we present BCPACE, an offline PAC-Bayes algorithm that computes a near-optimal policy for a continuous state and action BAMDP. BCPACE is an extension of C-PACE (Pazis & Parr, 2013), a PAC optimal algorithm for continuous state and action MDPs. Efficient exploration of a continuous space is challenging because that the same state-action pair cannot be visited more than once. C-PACE addresses this by assuming that the state-action value function is Lipschitz continuous, allowing the value of a state-action pair to be approximated with nearby samples. Similar to other PAC optimal algorithms (Strehl et al., 2009), C-PACE applies the principle of *optimism in the face of uncertainty*: the value of a state-action pair is approximated by averaging the value of nearby samples, inflated proportionally to their distances. Intuitively, this distance-dependent bonus term encourages exploration of regions that are far from previous samples until the optimistic estimate results in a near-optimal policy.

Our key insight is that C-PACE can be extended from continuous states to those augmented with finite-dimensional belief states. We derive sufficient conditions for Lipschitz continuity of the belief value function. We show that BCPACE is indeed PAC-Bayes and

bound the sample complexity as a function of the covering number of the reachable belief space from initial belief b_0. In addition, we also present and analyze three practical strategies for improving the sample complexity and runtime of BCPACE.

5.2.1 Definitions and Assumptions

We assume all rewards lie in $[0, R_{\max}]$ which implies $0 \leq Q_{\max}, V_{\max} \leq \frac{R_{\max}}{1-\gamma}$. We will first show that Assumption 5.2.1 and Assumption 5.2.2 are sufficient conditions for Lipschitz continuity of the value function.[1] Subsequent proofs do not depend on these assumptions as long as the value function is Lipschitz continuous.

Assumption 5.2.1 (Lipschitz Continuous Reward and Transition Functions)**.** *Given any two state-action pairs (s_1, a_1) and (s_2, a_2), there exists a distance metric $d(\cdot, \cdot)$ and Lipschitz constants L_R, L_P such that the following is true:*

$$|R(s_1, \phi, a_1) - R(s_2, \phi, a_2)| \leq L_R d_{s_1,a_1,s_2,a_2}$$
$$\sum_{s'} |P(s'|s_1, \phi, a_1) - P(s'|s_2, \phi, a_2)| \leq L_P d_{s_1,a_1,s_2,a_2}$$

where $d_{s_1,a_1,s_2,a_2} = d((s_1, a_1), (s_2, a_2))$

Assumption 5.2.2 (Belief Contraction)**.** *Given any two belief vectors b_1, b_2 and any tuple of (s, a, s'), the updated beliefs from the Bayes estimator $b_1' = \tau(b_1, s, a, s')$ and $b_2' = \tau(b_2, s, a, s')$ satisfy the following:*

$$||b_1' - b_2'||_1 \leq ||b_1 - b_2||_1$$

Assumption 5.2.1 and Assumption 5.2.2 can be used to prove the following lemma.

Lemma 1 (Lipschitz Continuous Value Function)**.** *Given any two state-belief-action tuples (s_1, b_1, a_1) and (s_2, b_2, a_2), there exists a distance metric $d(\cdot, \cdot)$ and a Lipschitz constant L_Q such that the following is true:*

$$|Q(s_1, b_1, a_1) - Q(s_2, b_2, a_2)| \leq L_Q d_{s_1,b_1,a_1,s_2,b_2,a_2}$$

[1] For all proofs, refer to supplementary material.

where $d_{s_1,b_1,a_1,s_2,b_2,a_2} = d\left((s_1, b_1, a_1), (s_2, b_2, a_2)\right)$

The distance metric $d\left((s_1, b_1, a_1), (s_2, b_2, a_2)\right)$ for state-belief-action tuples is a linear combination of the distance metric for state-action pairs used in Assumption 5.2.1 and the L_1 norm for belief

$$\alpha d\left((s_1, a_1), (s_2, a_2)\right) + ||b_1 - b_2||_1$$

for an appropriate choice of α, which is a function of $R_{\max}, L_R$, and L_P.

BCPACE builds an optimistic estimator $\tilde{Q}(s, b, a)$ for the value function $Q(s, b, a)$ using nearest neighbor function approximation from a collected sample set. Since the value function is Lipschitz continuous, the value for any query can be estimated by extrapolating the value of neighboring samples with a distance-dependent bonus. If the number of close neighbors is sufficiently large, the query is said to be "known" and the estimate can be bounded. Otherwise, the query is unknown and is added to the sample set. Once enough samples are added, the entire reachable space will be known and the estimate will be bounded with respect to the true optimal value function $Q^*(s, b, a)$. We define these terms more formally below.

Definition 5.2.1 (Known Query)**.** *Let* $L_{\tilde{Q}} = 2L_Q$ *be the Lipschitz constant of the optimistic estimator. A state-belief-action query* (s, b, a) *is said to be "known" if its* k^{th} *nearest neighbor in the sample set* (s_k, b_k, a_k) *is within* $\epsilon / L_{\tilde{Q}}$*.*

We are now ready to define the estimator.

Definition 5.2.2 (Optimistic Value Estimate)**.** *Assume we have a set of samples* C *where every element is a tuple* $(s_i, b_i, a_i, r_i, s'_i, b'_i)$*: starting from* (s_i, b_i)*, the agent took an action* a_i*, received a reward* r_i*, and transitioned to* (s'_i, b'_i)*. Given a state-belief-action query* (s, b, a)*, its* j^{th} *nearest neighbor from the sample set provides an optimistic estimate*

$$x_j = L_{\tilde{Q}} d\left((s, b, a), (s_j, b_j, a_j)\right) + \tilde{Q}(s_j, b_j, a_j). \tag{5.1}$$

The value is the average of all the nearest neighbor estimates

$$\tilde{Q}(s,b,a) = \frac{1}{k}\sum_{j=1}^{k} \min\left(x_j, \tilde{Q}_{\max}\right) \tag{5.2}$$

where $\tilde{Q}_{\max} = R_{\max} + \gamma Q_{\max}$ *is the upper bound of the estimate. If there are fewer than* k *neighbors,* $\tilde{Q}_{\max}$ *can be used in place of the corresponding* x_j*.*

Note that the estimator is a recursive function. Given a sample set C, value iteration is performed to compute the estimate for each of the sample points,

$$\tilde{Q}(s_i,b_i,a_i) = r_i + \gamma \max_a \tilde{Q}(s_i',b_i',a) \tag{5.3}$$

where $\tilde{Q}(s_i',b_i',a)$ is approximated via equation 5.2 using its nearby samples. This estimate must be updated every time a new sample is added to the set.

We introduce two additional techniques that leverage the Q-values of the underlying latent MDPs to improve the sample complexity of BCPACE.

Definition 5.2.3 (Best-Case Upper Bound)**.** *We can replace the constant* $\tilde{Q}_{\max}$ *in Definition 5.2.2 with* $\tilde{Q}_{\max}(s,b,a)$ *computed as follows:*

$$\tilde{Q}_{\max}(s,b,a) = \max_{\phi, b(\phi)>0} Q(s,\phi,a)$$

In general, any admissible heuristic U that satisfies $Q(s,b,a) \leq U(s,b,a) \leq \tilde{Q}_{\max}$ can be used. In practice, the Best-Case Upper Bound reduces exploration of actions which are suboptimal in all latent MDPs with nonzero probability.

We can also take advantage of $Q(s,\phi,a)$ whenever the belief distribution collapses. These exact values for the latent MDPs can be used to seed the initial estimates.

Definition 5.2.4 (Known Latent Initialization)**.** *Let* e_ϕ *be the belief distribution where* $P(\phi) = 1$*, i.e. a one-hot encoding. If there exists* ϕ *such that* $||b - e_\phi||_1 \leq \frac{\epsilon}{L_Q(1+\gamma)}$*, then we can use the following estimate:*

$$\tilde{Q}(s,b,a) = Q(s,\phi,a) \tag{5.4}$$

This extends Definition 5.2.1 for a known query to include any state-belief-action tuple where the belief is within $\frac{\epsilon}{L_Q(1+\gamma)}$ *of a one-hot vector.*

We refer to Proposition 5.2.1 for how this reduces sample complexity.

5.2.2 Algorithm

We describe our algorithm, BCPACE, in Algorithm 3. To summarize, at every timestep t the algorithm computes a greedy action a_t using its current value estimate $\tilde{Q}(s_t, b_t, a_t)$, receives a reward r_t, and transitions to a new state-belief (s_{t+1}, b_{t+1}) (Lines 6–8). If the sample is not known, it is added to the sample set C (Line 10). The value estimates for all samples are updated until the fixed point is reached (Line 11). Terminal condition G is met when no more samples are added and value iteration has converged for sufficient number of iterations. The algorithm invokes a subroutine for computing the estimated value function (Lines 13–23) which correspond to the operations described in Definition 5.2.2, 5.2.3, and 5.2.4.

5.2.3 Analysis of Sample Complexity

We now prove that BCPACE is PAC-Bayes. Since we adopt the proof of C-PACE, we only state the main steps and defer the full proof to supplementary material. We begin with the concept of a known belief MDP.

Definition 5.2.5 (Known Belief MDP)**.** *Let* Belief MDP *be the original belief MDP. Let K be the set of all known state-belief-action tuples. We define a known belief MDP* Belief MDP$_K$ *that is identical to* Belief MDP *on K (i.e. identical transition and reward functions) and for all other state-belief-action tuples, it transitions deterministically with a reward $R(s, b, a) = \tilde{Q}(s, b, a)$ to an absorbing state with zero reward.*

We can then bound the performance of a policy on Belief MDP with its performance on Belief MDP$_K$ and the maximum penalty incurred by escaping it.

Algorithm 3 BCPACE

Require: Bayes-Estimator τ, initial belief b_0, BAMDP, terminal condition G, horizon T

Ensure: Action value estimate $\tilde{Q}$

Initialize sample set $C \leftarrow \emptyset$
while G is false **do**
  Initialize BAMDP by resampling initial state and latent variable to $s_0, \phi_0 \sim P_0(s, \phi)$
  Reset belief to b_0
  for $t = 0, 1, 2, \cdots, T-1$ **do**
    $a_t \leftarrow \arg\max_a \tilde{Q}(s_t, b_t, a)$
    Execute a_t on BAMDP to receive r_t, s_{t+1}
    $b_{t+1} \leftarrow \tau(b_t, s_t, a_t, s_{t+1})$
    if (s_t, b_t, a_t) is not known **then**
      Add $(s_t, a_t, b_t, r_t, s_{t+1}, b_{t+1})$ to C
      Find fixed point of $\tilde{Q}(s_i, b_i, a_i)$ for C
Return $\tilde{Q}$

function $\tilde{Q}(s, b, a)$
  Find closest one-hot vector $\phi = \min_\phi d(b, e_\phi)$
  if $||b - e_\phi||_1 \leq \frac{\epsilon}{L_Q(1+\gamma)}$ **then**
    $\tilde{Q}(s, b, a) \leftarrow Q(s, \phi, a)$
  else
    Find k nearest neighbors $\{s_j, b_j, a_j, r_j, s'_j, b'_j\}$ in sample set C
    for $j = 1, \cdots, k$ **do**
      $d_j \leftarrow d((s, b, a), (s_j, b_j, a_j))$
      $x_j \leftarrow L_{\tilde{Q}} d_j + \tilde{Q}(s_j, b_j, a_j)$
    $\tilde{Q}(s, b, a) = \frac{1}{k} \sum_{j=1}^{k} \min\left(x_j, \tilde{Q}_{\max}(s, b, a)\right)$
  Return $\tilde{Q}(s, b, a)$

Lemma 2 (Generalized Induced Inequality, Lemma 8 in Strehl & Littman (2008))**.** *We are given the original belief MDP* Belief MDP*, the known belief MDP* Belief MDP$_K$*, a policy* π *and time horizon* T*. Let* $P(E_K)$ *be the probability of an escape event, i.e. the probability of*

sampling a state-belief-action tuple that is not in K when executing π on BELIEF MDP *from (s, b) for T steps. Let $V^{\pi}_{\text{BELIEF MDP}}$ be the value of executing policy on* BELIEF MDP*. Then the following is true:*

$$V^{\pi}_{\text{BELIEF MDP}}(s, b, T) \geq V^{\pi}_{\text{BELIEF MDP}_K}(s, b, T) - Q_{\max} P(E_K)$$

We now show one of two things can happen: either the greedy policy escapes from the known MDP, or it remains in it and performs near optimally. We first show that it can only escape a certain number of times before the entire reachable space is known.

Lemma 3 (Full Coverage of Known Space, Lemma 4.5 in Kakade et al. (2003))**.** *All reachable state-belief-action queries will become known after adding at most $k\mathcal{N}_{\text{BELIEF MDP}}(\epsilon/L_{\tilde{Q}})$ samples to C.*

Corollary 5.2.1 (Bounded Escape Probability)**.** *At a given timestep, let $P(E_K) > \frac{\epsilon}{Q_{\max}(1-\gamma)}$. Then with probability $1-\delta$, this can happen at most for $\frac{2Q_{\max}}{\epsilon}\left(k\mathcal{N}_{\text{BELIEF MDP}}(\epsilon/L_{\tilde{Q}}) + \log\left(\frac{1}{\delta}\right)\right)\log\frac{R_{\max}}{\epsilon}$ timesteps.*

We now show that when inside the known MDP, the greedy policy will be near optimal.

Lemma 4 (Near-optimality of Approximate Greedy (Theorem 3.12 of Pazis & Parr (2013)))**.** *Let $\tilde{Q}$ be an estimate of the value function that has bounded Bellman error $-\epsilon_- \leq \tilde{Q} - B\tilde{Q} \leq \epsilon_+$, where B is the Bellman operator. Let $\tilde{\pi}$ be the greedy policy on $\tilde{Q}$. Then the policy is near-optimal:*

$$V^{\tilde{\pi}}(s, b) \geq V^{*}(s, b) - \frac{\epsilon_- + \epsilon_+}{1-\gamma}$$

Let ϵ be the approximation error caused by using a finite number of neighbors in equation 5.2 instead of the Bellman operator. Then Lemma 4 leads to the following corollary.

Corollary 5.2.2 (Near-optimality on Known Belief MDP)**.** *If $\frac{\tilde{Q}^2_{\max}}{\epsilon^2}\log\left(\frac{2\mathcal{N}_{\text{BELIEF MDP}}(\epsilon/L_{\tilde{Q}})}{\delta}\right) \leq k \leq \frac{2\mathcal{N}_{\text{BELIEF MDP}}(\epsilon/L_{\tilde{Q}})}{\delta}$, i.e. the number of neighbors is large enough, then using Hoeffding's inequality we can show $-\epsilon \leq \tilde{Q} - B\tilde{Q} \leq 2\epsilon$. Then on the known belief MDP* BELIEF MDP$_K$*, the following can be shown with probability $1-\delta$:*

$$V^{\tilde{\pi}}_{\text{BELIEF MDP}_K}(s, b) \geq V^{*}_{\text{BELIEF MDP}_K}(s, b) - \frac{3\epsilon}{1-\gamma}$$

We now put together these ideas to state the main theorem.

Theorem 5 (BCPACE is PAC-Bayes)**.** *Let* Belief MDP *be a belief MDP. At timestep t, let $\tilde{\pi}_t$ be the greedy policy on $\tilde{Q}$, and let (s_t, b_t) be the state-belief pair. With probability at least $1-\delta$, $V^{\tilde{\pi}_t}(s_t, b_t) \geq V^*(s_t, b_t) - \frac{7\epsilon}{1-\gamma}$, i.e. the algorithm is $\frac{7\epsilon}{1-\gamma}$-close to the optimal policy for all but*

$$m = \frac{2Q_{\max}}{\epsilon}\left(k\mathcal{N}_{\text{Belief MDP}}(\epsilon/L_{\tilde{Q}}) + \log\frac{2}{\delta}\right)\log\left(\frac{R_{\max}}{\epsilon}\right)$$

steps when $k \in \left[\frac{\tilde{Q}_{\max}^2}{\epsilon^2}\log\frac{4\mathcal{N}_{\text{Belief MDP}}(\epsilon/L_{\tilde{Q}})}{\delta}, \frac{4\mathcal{N}_{\text{Belief MDP}}(\epsilon/L_{\tilde{Q}})}{\delta}\right]$ is used for the number of neighbors in equation 5.2.

Proof: At time t, we can form a known belief MDP Belief MDP$_K$ from the samples collected so far. Either the policy leads to an escape event within the next T steps or the agent stays within Belief MDP$_K$. Such an escape can happen at most m times with high probability; when the escape probability is low, $V^{\tilde{\pi}}$ is $\frac{7\epsilon}{1-\gamma}$-optimal. □

5.2.4 Analysis of Performance Enhancements

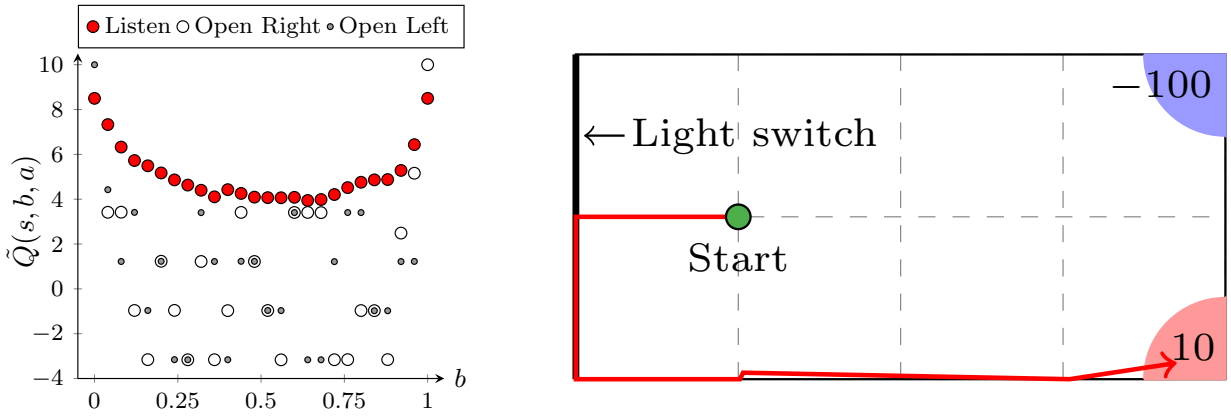
We can initialize estimates with exact Q values for the latent MDPs.This makes the known space larger, thus reducing covering number.

Proposition 5.2.1 (Known Latent Initialization)**.** *Let $\mathcal{N}'_{\text{Belief MDP}}(\epsilon/L_{\tilde{Q}})$ be the covering number of the reduced space $\left\{(s,b,a) \;\middle|\; \forall e_i, d\left(b, e_i\right) \geq \frac{\epsilon}{L_Q(1+\gamma)}\right\}$. Then the sample complexity reduces by a factor of $\frac{\mathcal{N}'_{\text{Belief MDP}}(\epsilon/L_{\tilde{Q}})}{\mathcal{N}_{\text{Belief MDP}}(\epsilon/L_{\tilde{Q}})}$.*

It is also unnecessary to perform value iteration until convergence.

Proposition 5.2.2 (Approximate Value Iteration)**.** *Let $0 < \beta < \tilde{Q}_{\max}$. Suppose the value iteration step (Line 11) is run only for $i = \lceil \log(\beta/\tilde{Q}_{\max})/\log\gamma \rceil$ iterations denoted by $\tilde{B}^i\tilde{Q}$ (instead of until convergence $\tilde{B}^\infty\tilde{Q}$). We can bound the difference between two functions as $\left|\left|\tilde{B}^i\tilde{Q} - \tilde{B}^\infty\tilde{Q}\right|\right|_\infty \leq \beta$. This results in an added suboptimality term in Theorem 5:*

$$V^{\tilde{\pi}_t}(s_t, b_t) \geq V^*(s_t, b_t) - \frac{7\epsilon + 2\beta}{1-\gamma} \tag{5.5}$$



(a) Value approximation for Tiger

(b) An optimal path taken by BCPACE

Figure 5.2: With greedy exploration, only best actions are tightly approximated (Figure 5.2a). BCPACE takes optimal actions for a continuous BAMDP (Figure 5.2b).

One practical enhancement is to collect new samples in a batch with a fixed policy before performing value iteration. This requires two changes to the algorithm: 1) an additional loop to repeat (Lines 5–11) n times, and 2) perform (Line 11) outside of the loop. This increases the sample complexity by a constant factor but has empirically reduced runtime by only performing value iteration when a large change is expected.

Proposition 5.2.3 (Batch Sample Update)**.** *Suppose we collect new samples from n rollouts with the greedy policy at time t before performing value iteration. This increases the sample complexity only by a constant factor of $O(n)$.*

5.3 Experimental Results

We compare BCPACE with QMDP, POMDP-LITE, and SARSOP for discrete BAMDPs and with QMDP for continuous BAMDPs. For discrete state spaces, we evaluate BCPACE on two widely used synthetic examples, Tiger (Kaelbling et al., 1998) and Chain (Strens, 2000). For both BCPACE and POMDP-LITE, the parameters were tuned offline for

	QMDP	P-Lite	SARSOP	BCPACE
`Tiger`	$16.5 \pm .8$	$11.8 \pm .6$	$\mathbf{17.8 \pm 1.9}$	$\mathbf{18.0 \pm 1.4}$
`Chain`	$12.9 \pm .5$	$13.0 \pm .1$	$13.4 \pm .1$	$\mathbf{14.3 \pm .1}$
`LightDark`	0	$15.1 \pm .3$	**29.0**	**29.0**
`LightDark` (cont.)	0	-	-	$\mathbf{25.4 \pm .1}$

Table 5.1: Benchmark results. `LightDark` (cont.) has continuous state space. BCPACE is competitive for both discrete and continuous BAMDPs).

best performance. For continuous state spaces, we evaluate on a variant of the Light-Dark problem (Platt et al., 2010).

While our analysis is applicable for BAMDPs with continuous state and action spaces, any approximation the greedy selection of an action is not guaranteed to be PAC-Bayes. Thus, we limit our continuous BAMDP experiments to discrete action spaces and leave the continuous action case for future work.

Tiger: Please see Section 4.4 for the description of the problem.

Table 5.1 shows that BCPACE performs as competitively as SARSOP and is better than QMDP or POMDP-lite. This is not surprising since both BCPACE and SARSOP are offline solvers.

Figure 5.2a visualizes the estimated values. Because BCPACE explores greedily, exploration is focused on actions with high estimated value, either due to optimism from under-exploration or actual high value. As a result, suboptimal actions are not taken once BCPACE is confident that they have lower value than other actions. Because fewer samples have been observed for these suboptimal actions, their approximated values are not tight. Note also that the original problem explores a much smaller subset of the belief space, so we have randomly initialized the initial belief from $[0, 1]$ rather than always initializing to 0.5 for this visualization, forcing BCPACE to perform additional exploration.

Chain: Please see Section 4.4 for the description of the canonical version of the problem.

In our variant, we allow the slip probability to be selected from $[0.2, 0.5, 0.8]$ with uniform probability at the beginning of each episode. These three latent MDPs form a BAMDP. Table 5.1 shows that BCPACE outperforms other algorithms.

Light-Dark Tiger: We consider a variant of the Light-Dark problem, which we call Light-Dark Tiger (Figure 5.2b). In this problem, one of the two goal corners (top-right or bottom-right) contains a tiger. The agent receives a penalty of -100 if it enters the goal corner containing the tiger and a reward of 10 if it enters the other region. There are four actions—Up, Down, Left, Right—which move one unit with Gaussian noise of $\mathcal{N}(0, \sigma^2)$. The tiger location is unknown to the agent until the left wall is reached. As in the original Tiger problem, this POMDP can be formulated as a BAMDP with two latent MDPs.

We consider two cases, one with zero noise and another with $\sigma = 0.01$. With zero noise, the problem is a discrete POMDP and the optimal solution is deterministic; the agent hits the left wall and goes straight to the goal location. When there is noise, the agent may not reach the left wall in the first step. Paths executed by BCPACE still take Left until the left wall is hit and goes to the goal (Figure 5.2b).

5.4 Discussion

We have presented the first PAC-Bayes algorithm for continuous BAMDPs whose value functions are Lipschitz continuous. While the practical implementation of BCPACE is limited to discrete actions, our analysis holds for both continuous and discrete state and actions. We believe that our analysis provides an important insight for the development of PAC efficient algorithms for continuous BAMDPs.

The BAMDP formulation is useful for real-world robotics problems where uncertainty over latent models is expected at test time. An efficient policy search algorithm must incorporate prior knowledge over the latent MDPs to take advantage of this formulation. As a step toward this direction, we have introduced several techniques that utilize the value functions of underlying latent MDPs without affecting PAC optimality.

One of the key assumptions BCPACE has made is that the cardinality of the latent

state space is finite. This may not be true in many robotics applications in which latent variables are drawn from continuous distributions. In such cases, the true BAMDP can be approximated by sampling a set of latent variables, as introduced in Wang et al. (2012). In future work, we will investigate methods to select representative MDPs and to bound the gap between the optimal value function of the true BAMDP and the approximated one.

Although it is beyond the scope of BCPACE, we would like to make two remarks. First, BCPACE can easily be extended to allow parallel exploration, similar to how Pazis & Parr (2016) extended the original C-PACE to concurrently explore multiple MDPs. Second, since we have generative models for the latent MDPs, we may enforce exploration from arbitrary belief points. Of course, the key to efficient exploration of belief space lies in exploring just beyond the optimally reachable belief space, so "random" initialization is unlikely to be helpful. However, if we can approximate this space similarly to sampling-based kinodynamic planning algorithms (Li et al., 2016), this may lead to more structured search in belief space.

Chapter 6

BAYESIAN RESIDUAL Q-LEARNING

In this chapter, we propose a Q-learning framework that jointly trains an ensemble of experts that propose action-value estimates and a residual network that learns the gap between the ensemble's proposal and the Bayes-optimal action-value. Our algorithm encompasses several alternate architectures and techniques for Bayesian Q-learning, which we compare empirically.

Our key insight is that finding an expert policy for each latent MDP, in the absence of model uncertainty, is much easier than finding a Bayes-optimal policy for the whole problem. This insight is materialized again in Chapter 7, with a batch policy optimization algorithm. In this chapter, we apply it to a Q-learning framework on discrete action spaces and focus on closing the gap between the experts and the optimal policy in action-values.

6.1 Introduction

As discussed in previous chapters, achieving the Bayes-optimal objective due to the curse of dimensionality. Looking at the challenge closer, we notice that there is more specific challenge to BAMDPs whose latent models require significantly different policies. In these problems, the agent must not only learn to solve each latent MDP, but also learn to be Bayesian over them. As the latent models get more diverse and complex, the simulatenous learning of the two becomes more computationally expensive. Deep reinforcement learning algorithms that utilize neural networks often fail to learning the multimodal policy, and end up converging to undesirable local optima.

We leverage the fact that training an expert policy for each MDP is much easier than training a single Bayes-optimal agent across multiple. For any given MDP, such an expert

policy could come from optimal control, demonstration, or another RL algorithm. Once these experts are provided, we can use their expected value under the belief as a baseline, which is a well-known upper-bound to the Bayes-optimal value function (Littman et al., 1995a). Since the expected value is optimal when the belief collapses to single MDP, the agent can focus learning in regions where the entropy is high. Formally, we decompose the Bayes-optimal value function as the following:

$$Q^*(s, b, a) = (1 - w(b)) \sum_k b(k) Q^{(k)}(s, a) + w(b) Q^r(s, b, a) \tag{6.1}$$

where $w(b)$ is the entropy, $b(k)$ is the belief of the kth MDP, and $Q^{(k)}$ is the value function of the corresponding expert and Q^r is the *residual*. We assume there is a known finite number of MDPs or that they have been clustered into a finite number of clusters.

Our algorithm is a Bayesian Q-learning method which simultaneously learns the residual and improves the expert values. Our key insight is that we can maintain distinct experts by only exposing each to the MDP that it is responsible for; because each expert is responsible only for a single MDP, the experts remain belief-agnostic. Meanwhile, the residual Q-network (Q^r) is belief-aware and is trained across all episodes.

We make the following contributions:

- An architecture for a Bayesian RL agent with distinct experts and a residual Q-network
- A family of algorithms that jointly train expert and residual Q-networks, warm-start learned experts, or use fixed experts
- Empirical evaluation that demonstrates the network is able to learn Bayes-optimal behaviors in a data-efficient manner

6.2 Bayesian Residual Q-Learning

We propose a Q-learning algorithm which combines a set of expert value functions in a Bayes-optimal manner by simultaneously learning a residual value function and expert value functions. We start from the idea that each expert can be responsible for one of the latent

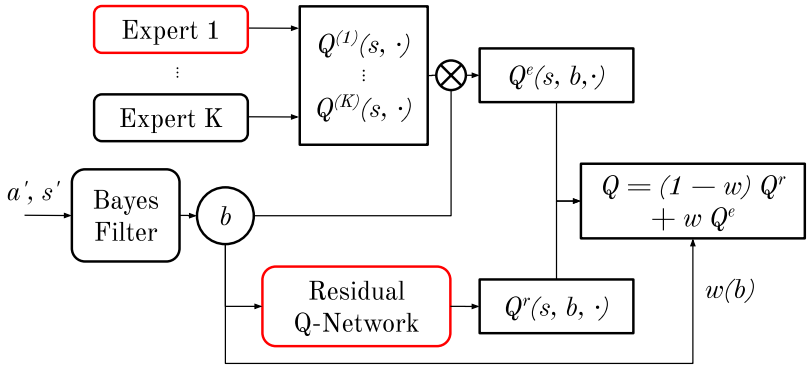


Figure 6.1: Network overview. State s is an input to every module, and s', a' are the state and action from the previous step. Red is one example of trained modules per sample batch. Samples from $\mathcal{M}^k$ are only used to train expert k, while the Residual Q-network is trained with samples from all MDPs.

MDPs, and combine the expert value functions with a residual value function (Equation 6.1). Assuming that the experts are indeed good experts for each MDP, we use entropy as w, i.e. $w(b) = -\alpha \sum_k b(k) \log(b(k))$ with α as a tuning parameter so that the residual function disappears when the entropy is low.

Now we need to make sure that the experts are actually trained to achieve high performance on their assigned MDP. The key insight is to expose expert k only to the episodes where $\mathcal{M}^k$ was the latent MDP, and train it in a belief-agnostic manner.

The algorithm is provided in Algorithm 4, which is a modified version of Deep Q Network (DQN) (Van Hasselt et al., 2016). Figure 7.1 shows the network structure. During training, we keep track of the latent MDPs and maintain target values for the final value function and expert value functions separately. Given a replay sample $(k, s_j, b_j, a_j, r_j, s_{j+1}, b_{j+1})$ generated from $\mathcal{M}^k$, we have

$$y_j^{(k)} = r_j + \gamma \max_{a'} Q_\theta^{(k)}(s_j, a') \tag{6.2}$$

corresponding to the target for the expert Q_θ^k and

$$y_j = r_j + \gamma \max_{a'} Q_\theta(s_j, b_j, a') \tag{6.3}$$

Algorithm 4 Bayesian Residual Q-Learning

Require: MDP distribution P_0, Bayes filter $\psi(\cdot)$, initial belief b_0, θ_0, horizon T, N, Replay memory $\mathcal{D}$

for episode$= 1, \cdots, N$ **do**
  Sample an MDP $\mathcal{M}_i \sim P_0$
  Initialize belief b_0
  for $t = 1, \cdots, T$ **do**
    With probability ϵ select a random action a_t, or
    $a_t = \arg\max_a Q_\theta(s_t, b_t, a)$
    Execute a_t and observe r_t, s_{t+1}
    Update $b_{t+1} = \psi(b_t, a_t, s_{t+1})$
    Store $(i, s_t, b_t, a_t, r_t, s_{t+1}, b_{t+1})$ in $\mathcal{D}$
  Sample $\mathcal{M}_k \sim P_0$
  Sample a random minibatch from $\mathcal{D}$ with k, i.e. $(k, s_j, b_j, a_j, r_j, s_{j+1}, b_{j+1})$
  Set $y_j^{(k)} = r_j + \gamma \max_{a'} Q_\theta^k(s_j, a')$
  Set $y_j = r_j + \gamma \max_{a'} Q_\theta(s_j, b_j, a')$
  Perform a gradient descent step on $(y_j^{(k)} - Q_\theta^k(s_j, a_j))^2$, update only θ_k for Q_θ^k
  Perform a gradient descent step on $(y_j - Q_\theta(s_j, b_j, a_j))^2$, update only θ_r for Q_θ^r

corresponding to the target for the weight-balanced sum of Q^r and $Q^{(1),\ldots,(k)}$ in Equation 6.1.

In certain cases, we may have access to near-optimal experts even during test time. In this case we propose to use a simpler version of our algorithm, which we refer to as RBQN-FixedExperts (BRQN-FE). In RBQN-FE, we perform the gradient descent steps from Equation 6.3 and update only θ_r. Through experiments we verify that this indeed results in a data-efficient, fast convergence to the optimal value. We refer to our original algorithm which jointly trains the experts as RBQN-LearnedExperts (BRQN-LE). If the

experts are accessible only during training, we can consider an intermediate approach where we warm-start the expert networks in BRQN-LE with expert value functions and jointly train them.

6.3 Experimental Results

6.3.1 Toy Example: Tiger

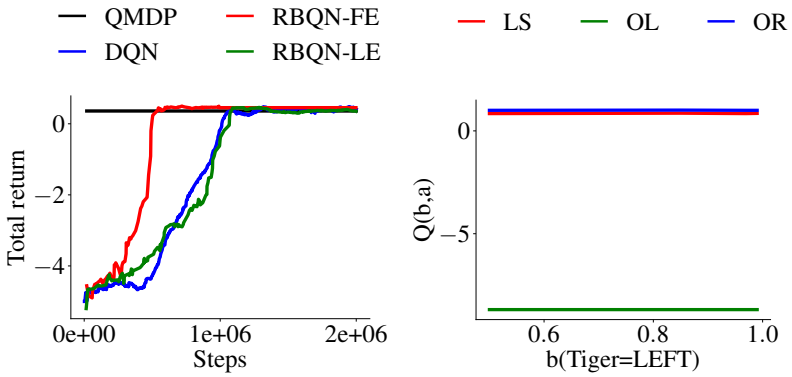
We first use a toy problem to verify our hypothesis that our algorithm produces experts that specialize in each MDP. We refer to Section 4.4 for the description of the problem. In this problem, a Bayes-optimal agent would listen until its belief about which door the tiger is behind is substantially higher for one door than the other.

Figure 6.2a shows two versions of our algorithm, RBQN-FixedExperts, RBQN-LearnedExperts, compared with a vanilla DQN trained on the belief MDP which provides belief of Tiger location as an input. We can see that the RBQN-FE has a much faster convergence as the value is near-optimal for one-hot beliefs.

Figure 6.2b shows that BRQN-LE is able to train each expert on one MDP, without mixing other MDPs. It shows the value function of one of the experts trained by BRQN-LE. The expert trained with the tiger behind the left door learns precisely that it can open the right door for all beliefs. Nonetheless, the final value function produced by RBQN-LearnedExperts is Bayes-optimal, as can be seen in Figure 6.2a.

6.3.2 RockSample

Next, we turn to the `RockSample` POMDP problem, which has also been shown to to be a BRL problem by Chen et al. (2016), to demonstrate how much our algorithm can improve beyond the experts. In this problem, the agent is in a grid world where rocks are positioned at a set of predetermined locations (Figure 6.3). The rocks are either GOOD or BAD, and the agent must sense the rock and use the noisy sensor information to determine which rock to sample. Sampling GOOD rocks provides +10 reward, while sampling BAD rocks result in



(a) Total rewards (b) BRQN-LE expert

Figure 6.2: `Tiger`. (a) BRQN-FE converges quickly to the optimum. (b) BRQN-LE produces experts specialized in one MDP, which opens one of the doors deterministically.

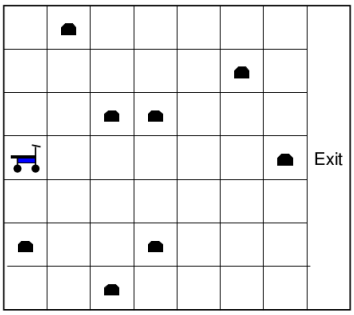


Figure 6.3: `RockSample`. Figure from Smith & Simmons (2004).

BRQN-FE	DQN	BPO	QMDP	SARSOP
19.34 ± 0.33	0.0 ± 0.0	7.35 ± 0.0	16.58 ± 0.99	21.47 ± 0.04

Table 6.1: `RockSample` with 95% confidence interval.

-10 penalty. This problem is very challenging because the agent has to associate the sparse rewards on the rocks with sensing actions which do not have immediate rewards. We use the setting with 8 rocks in a 7×7 grid.

Table 6.1 shows the comparison of BRQN-FE with other algorithms. Vanilla DQN was not able to learn this relationship, resulting in zero rewards. BPO learns to go straight to the goal.

In BRQN-FE, there are $2^8 = 256$ experts. Each expert takes the most value-maximizing path given a fixed rock configuration of GOOD and BAD. BRQN-FE combines the experts' value functions with the output of the residual network with a weight parameter of 0.01 given

to the residual network. The weight parameter was found by offline tuning. Our algorithm in fact learns to significantly outperform the QMDP agent, getting closer to the near-optimal value given by SARSOP, an offline appoximate POMDP solver (Kurniawati et al., 2008).

However, we were not able to get BRQN-LE to produce as good as performance as BRQN-FE, as BRQN-LE struggled to get each of the experts to the level comparable to optimal experts.

We believe that this shows our algorithm as a potential way to combine optimal control methods with Bayesian RL approaches to produce an agent which performs beyond what each optimal control expert can suggest, thus getting the best of both worlds.

6.4 Discussion

We have proposed an algorithm for Bayesian meta-reinforcement learning which simultaneously trains experts and a residual Q-network. Our algorithm trains each expert solely on the MDP it is responsible for while training the residual network to learn how to combine the experts Bayes optimally.

Our results show both limitations of and promising directions for our algorithm. Through the experiment on `Tiger` we have verified that the experts in the joint training scenario indeed learn to be single-MDP experts. From the experiment on `RockSample`, we show that the residual network is indeed capable of learning to perform better what the experts propose even when the experts are optimal with respect to each of the latent MDPs. However, jointly training the experts and the residual for this large-scale problem turned out to be quite challenging, even when we warm-started the Q functions with approximated Q values from the experts. This is in fact a phenomenon commonly observed in Imitation Learning algorithms, where even a small discrepancy between the learned Q function and the expert Q function results in the agent exploring unknown domains and therefore failing to learn. On the other hand, BRQN-FE sheds new light on how to combine experts which can be acquired from much simpler settings, to get an agent that can handle complex Bayesian RL problems.

Chapter 7

BAYESIAN RESIDUAL POLICY OPTIMIZATION

In this chapter, we focus on Bayesian RL problems with complex latent MDPs that may require multi-modal policies. Analogous to Chapter 6, we build on the following insight: in the absence of uncertainty, each latent MDP is easier to solve. We first obtain an ensemble of experts, one for each latent MDP, and fuse their advice to compute a baseline policy. Next, we train a Bayesian residual policy to improve upon the ensemble's recommendation and learn to reduce uncertainty. Our algorithm, Bayesian Residual Policy Optimization (BRPO), imports the scalability of policy gradient methods and task-specific expert skills. We prove that BRPO monotonically improve upon the expert ensemble, and empirically demonstrate that BRPO significantly improves the ensemble of experts and drastically outperforms existing adaptive RL methods, both in simulated and physical robot experiments.

7.1 Introduction

A Bayesian RL problem can be viewed as a large continuous belief MDP, which is computationally infeasible to solve directly (Ghavamzadeh et al., 2015). Solving Bayesian RL problems becomes even harder if the latent MDPs require vastly different policies to achieve high reward. For example, consider an autonomous vehicle which must safely navigate around pedestrians navigating to latent goals (Figure 7.1). Depending on the latent goals of the pedestrians, the agent may make drastic changes to its navigation policy. Robust RL methods (Rajeswaran et al., 2017; Tobin et al., 2017) often fail to recover that multi-modality.

We build upon a simple yet recurring observation (Choudhury et al., 2018; Osband et al., 2013): ignoring uncertainty by solving individual latent MDPs is much more tractable than solving the original belief MDP. If the path for each pedestrian is known, the autonomous

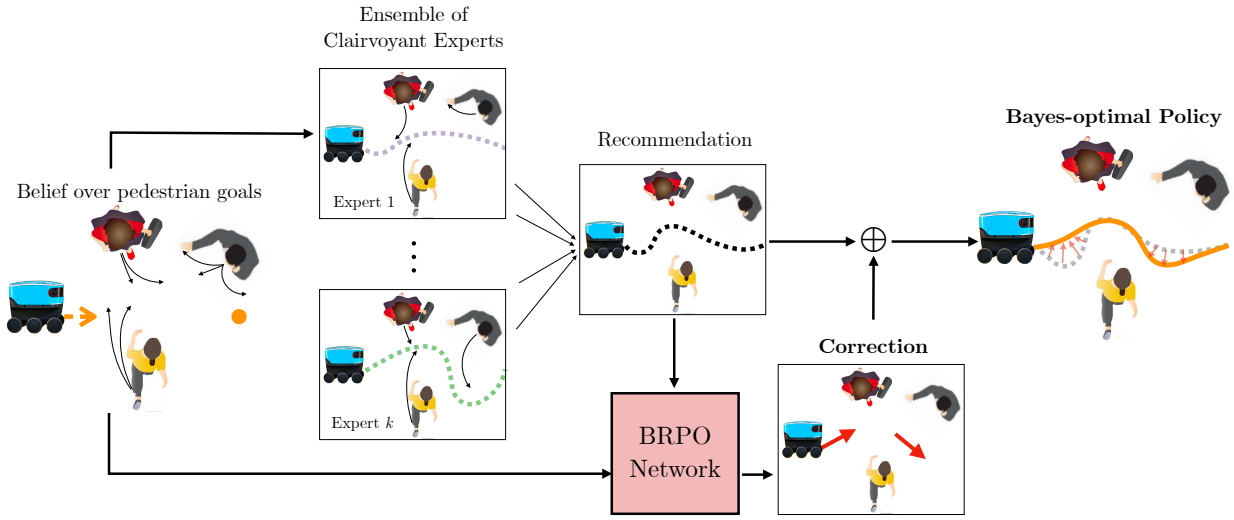


Figure 7.1: An overview of Bayesian Residual Policy Optimization. (a) Pedestrian goals are latent and tracked as a belief distribution. (b) Experts propose their solutions for a scenario, which are combined into a mixture of experts. (c) Residual policy takes in the belief and ensemble's proposal and returns a correction to the proposal. (d) The combined BRPO and ensemble policy is Bayes-optimal.

vehicle can invoke a motion planner to avoid collisions. We can think of these solutions as *clairvoyant experts*, i.e., experts that think they know the latent MDP and offer advice accordingly. An ensemble policy of these clairvoyant experts can be surprisingly effective, but since each expert is individually confident about which MDP the agent faces, the ensemble never prioritizes uncertainty-reducing or robust actions. Such actions can be critical for solving the original problem with model uncertainty.

Our algorithm, Bayesian Residual Policy Optimization (BRPO), computes a *residual* policy to augment an ensemble of clairvoyant experts (Figure 7.1). This is computed via policy optimization in a residual belief MDP, induced by the ensemble's actions on the original belief MDP. Because the ensemble is near-optimal when the entropy is low, BRPO can focus on learning to act safely in regions of high entropy. Moreover, the better initialization provided by the ensemble enables BRPO to learn much faster than methods starting from

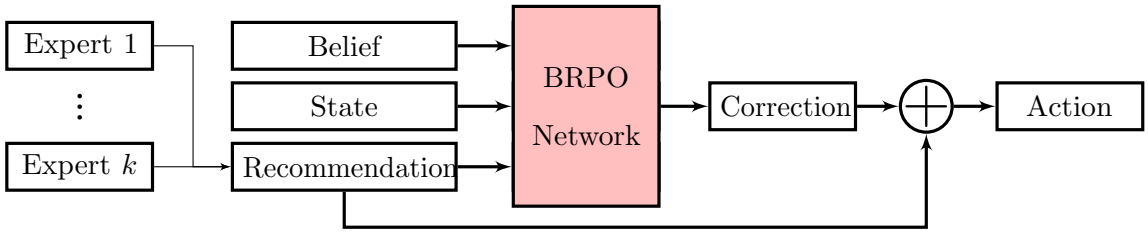


Figure 7.2: Bayesian residual policy network architecture.

scratch without experts.

Our key contribution is the following:

- We propose BRPO, a scalable Bayesian RL algorithm.
- We prove that BRPO monotonically improves upon the expert ensemble, converging to a Bayes-optimal policy.
- We experimentally demonstrate that BRPO outperforms both the ensemble and existing adaptive RL algorithms in simulation, and apply BRPO to a physical robot task.

*7.2 Bayesian Residual Policy Optimization (*BRPO*)*

Bayesian Residual Policy Optimization relies on an ensemble of clairvoyant experts where each expert solves a latent MDP. This is a flexible design parameter with three guidelines. First, the ensemble must be fixed before training begins. This freezes the residual belief MDP, which is necessary for theoretical guarantees (Section 7.2.3). Next, the ensemble should return its recommendation quickly since it will be queried online at test time. Practically, we have observed that this factor is often more important than the strength of the initial ensemble; even weaker ensembles can provide enough of a head start for residual learning to succeed. Finally, when the belief has collapsed to a single latent MDP, the resulting recommendation must follow the corresponding expert. In general, the ensemble should become more reliable as entropy decreases.

BRPO performs batch policy optimization in the residual belief MDP, producing actions that continuously correct the ensemble recommendations. Intuitively, BRPO enjoys improved data-efficiency because the correction can be small when the ensemble is effective (e.g., when uncertainty is low or when the experts are in agreement). When uncertainty is high, the agent learns to override the ensemble, reducing uncertainty and taking actions robust to model uncertainty.

7.2.1 Ensemble of Clairvoyant Experts

The ensemble policy maps the state and belief to a distribution over actions $\pi_e : S \times B \to P(A)$. It combines clairvoyant experts $\pi_1, \cdots, \pi_k$, one for each latent variable ϕ_i. Each expert can be computed via single-MDP RL or optimal control. There are various strategies to produce an ensemble from a set of experts. Following the maximum a posteriori (MAP) expert of the ensemble $\pi_e = \arg\max_{b(\phi)} \pi_\phi$ allows BRPO to solve tasks with infinitely many latent MDPs. The ensemble can also be a weighted sum of expert actions, which is the MAP action for Gaussian policies.

Maximum a Posteriori as an Ensemble of Experts One choice for the ensemble policy π_e is to select the maximum a posteriori (MAP) action, $a_{\text{MAP}} = \arg\max_a \sum_{i=1}^{k} b(\phi_i)\pi_i(a|s)$. However, computing the MAP estimate may require optimizing a non-convex function, e.g., when the distribution is multimodal. We can instead maximize the lower bound using Jensen's inequality.

$$\log \sum_{i=1}^{k} b(\phi_i)\pi_i(a|s) \geq \sum_{i=1}^{k} b(\phi_i) \log \pi_i(a|s) \tag{7.1}$$

This is much easier to solve, especially if $\log \pi_i(a|s)$ is convex. If each $\pi_i(a|s)$ is a Gaussian with mean μ_i and covariance Σ_i, the resultant action is the belief-weighted sum of mean actions:

$$a^* = \arg\max_a \sum_{i=1}^{k} b(\phi_i) \log \pi_i(a|s) = \left[\sum_{i=1}^{k} b(\phi_i)\Sigma_i^{-1}\right]^{-1} \sum_{i=1}^{k} b(\phi_i)\Sigma_i^{-1}\mu_i$$

7.2.2 Bayesian Residual Policy Learning

Our algorithm is summarized in Algorithm 5. In each training iteration, BRPO collects trajectories by simulating the current policy on several MDPs sampled from the prior distribution. At every timestep of the simulation, the ensemble is queried for an action recommendation (Line 9), which is summed with the correction from the residual policy network (Figure 7.2) and executed (Line 10-12). The Bayes filter updates the posterior after observing the resulting state (Line 13). The collected trajectories are the input to a policy optimization algorithm, which updates the residual policy network.

The BRPO agent effectively experiences a different MDP. In this new MDP, actions are always shifted by the ensemble recommendation. We formalize this correspondence between the residual and original belief MDPs in the next section, showing that BRPO inherits the monotonic improvement guarantee from existing policy optimization algorithms.

7.2.3 BRPO Inherits Motononic Improvement

BRPO guarantees monotonic improvement on the expected return of the mixture between the ensemble policy π_e and the initial residual policy π_{r_0}. First, we observe that π_r operates on its own residual MDP and show that the probability of any state-sequence for π_r in the residual MDP is equal to that of π in the original MDP. Then we observe that the monotonic guarantee from the underlying policy optimization algorithm holds for π_r in the residual MDP. Combining these, we transfer the guarantee for π_r in the residual MDP to π in the original MDP. The following arguments apply to all MDPs, not just belief MDPs; thus, we've omitted the belief from the state for clarity of exposition.

Let $\mathcal{M} = \langle S, A, T, R, P_0 \rangle$ be the original MDP. For simplicity, assume that R depends only on states. Every π_e for $\mathcal{M}$ induces a residual MDP $\mathcal{M}_r$ equivalent to $\mathcal{M}$ except for the transition function, T_r. For every residual action a_r, T_r marginalizes over all expert

Algorithm 5 Bayesian Residual Policy Optimization

Require: Bayes filter ψ, belief b_0, prior P_0, residual policy π_{r_0}, expert π_e, horizon T, $n_{\text{itr}}, n_{\text{sample}}$

for $i = 1, 2, \cdots, n_{\text{itr}}$ **do**
  for $n = 1, 2, \cdots, n_{\text{sample}}$ **do**
    Sample latent MDP $\mathcal{M}$: $(s_0, \phi_0) \sim P_0$
    $\tau_n \leftarrow \texttt{Simulate}(\pi_{r_{i-1}}, \pi_e, b_0, \psi, \mathcal{M}, T)$
  $\pi_{r_i} \leftarrow \texttt{BatchPolicyOpt}(\pi_{r_{i-1}}, \{\tau_n\}_{n=1}^{n_{\text{sample}}})$
return $\pi_{r_{best}}$

procedure SIMULATE($\pi_r, \pi_e, b_0, \psi, \mathcal{M}, T$)
  for $t = 1, \cdots, T$ **do**
    $a_{e_t} \sim \pi_e(s_t, b_t)$ // Expert recommendation
    $a_{r_t} \sim \pi_r(s_t, b_t, a_{e_t})$ // Residual policy
    $a_t \leftarrow a_{r_t} + a_{e_t}$
    Execute a_t on $\mathcal{M}$, receive r_{t+1}, observe s_{t+1}
    $b_{t+1} \leftarrow \psi(s_t, b_t, a_t, s_{t+1})$ // Belief update
  $\tau \leftarrow (s_0, b_0, a_{r_0}, r_1, s_1, b_1, \cdots, s_T, b_T)$
  return τ

recommendations.

$$T_r(s'|s, a_r) = \sum_{a_e} T(s'|s, a_e + a_r)\pi_e(a_e|s) \tag{7.2}$$

Let $\pi_r(a_r|s, a_e)$ be a residual policy. The final policy π executed on $\mathcal{M}$ is a mixture of π_r and π_e.

$$\pi(a|s) = \sum_{a_r} \pi_e(a - a_r|s)\pi_r(a_r|s, a - a_r) \tag{7.3}$$

First, we note that the probability of observing any sequence of states is equal in both MDPs. Let $\xi = (s_0, s_1, ..., s_{T-1})$ be a sequence of states. Let $\alpha = \{\tau\}$ be the set of all length T trajectories (state-action sequences) in $\mathcal{M}$ with ξ as the states, and $\beta = \{\tau_r\}$ be analogously

defined for a set of trajectories in $\mathcal{M}_r$. Note that each state-sequence ξ may have multiple corresponding state-action trajectories $\{\tau\}$.

Lemma 6. *The probability of ξ is equal when executing π on $\mathcal{M}$ and π_r on $\mathcal{M}_r$, i.e.,*

$$\pi(\xi) = \sum_{\tau \in \alpha} \pi(\tau) = \sum_{\tau_r \in \beta} \pi_r(\tau_r) = \pi_r(\xi)$$

Proof. We prove this by induction. The base case ($T = 0$) holds trivially since $\mathcal{M}$ and $\mathcal{M}_r$ share the same initial state distribution P_0. Assuming that it holds for $T = t$, pick any ξ and let its last element be s. Consider an s'-extended sequence $\xi' = (\xi, s')$. Conditioned on ξ, the probability of ξ' is equal in $(\pi, \mathcal{M})$ and $(\pi_r, \mathcal{M}_r)$, which we can see by marginalizing over all state-action sequences:

$$\sum_{\tau_r'} \pi_r(\tau_r'|\xi) = \sum_{a_r} \pi_r(a_r|s) T_r(s'|s, a_r) \tag{7.4}$$

$$= \sum_{a_r} \pi_r(a_r|s) \sum_{a} T(s'|s, a) \pi_e(a - a_r|s) \tag{7.5}$$

$$= \sum_{a} \sum_{a_r} \pi_r(a_r|s) \pi_e(a - a_r|s) T(s'|s, a) \tag{7.6}$$

$$= \sum_{a} \pi(a|s) T(s'|s, a) \tag{7.7}$$

$$= \sum_{\tau'} \pi(\tau'|\xi) \tag{7.8}$$

The transition from (7.4) to (7.5) comes from (7.2) and (7.6) to (7.7) comes from (7.3). It follows that,

$$\pi(\xi') = \pi(\xi) \sum_{\tau'} \pi(\tau'|\xi) = \pi_r(\xi) \sum_{\tau_r'} \pi_r(\tau_r'|\xi) = \pi_r(\xi'),$$

which proves the lemma. □

Lemma 6 immediately leads to the next theorem.

Theorem 7. *A residual policy π_r executed on $\mathcal{M}_r$ has the same expected return as the mixture policy π executed on $\mathcal{M}$.*

$$\mathbb{E}_{\tau \sim (\pi, \mathcal{M})}[R(\tau)] = \mathbb{E}_{\tau_r \sim (\pi_r, \mathcal{M}_r)}[R(\tau_r)]$$

Proof. Since reward depends only on the states, $R(\tau) = R(\tau_r) = R(\xi)$ for all $\tau \in \alpha, \tau_r \in \beta$. Hence, Lemma 6 implies that the performance of π_r on the residual MDP $\mathcal{M}_r$ is equivalent to the BRPO agent's performance on the original MDP $\mathcal{M}$. □

Finally, we observe that the residual policy π_r, when executed in $\mathcal{M}_r$, inherits the monotonic improvement guarantee from PPO (Schulman et al., 2017), the underlying policy optimization algorithm.

Lemma 8. BRPO *monotonically improves the expected return of* π_r *in* $\mathcal{M}_r$*, i.e.,*

$$J(\pi_{r_{i+1}}) \geq J(\pi_{r_i})$$

with $J(\pi_r) = \mathbb{E}_{\tau \sim (\pi_r, \mathcal{M}_r)}[R(\tau)]$*, where* $\tau \sim (\pi_r, \mathcal{M}_r)$ *indicates that* τ *is a trajectory with actions sampled from* π_r *and executed on* $\mathcal{M}_r$*.*

Proof. BRPO uses PPO for optimization (Schulman et al., 2017). PPO's clipped surrogate objective approximates the following objective,

$$\max_{\theta} \hat{\mathbb{E}} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t - \beta \cdot \text{KL}(\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)) \right], \tag{7.9}$$

where π_θ is a policy parameterized by θ and $\pi_{\theta_{\text{old}}}$ is the policy in the previous iteration, which correspond to the current and previous residual policies $\pi_{r_i}, \pi_{r_{i-1}}$ in Algorithm 5. $\hat{A}$ is the generalized advantage estimate (GAE) and KL is the Kullback–Leibler divergence between the two policy distributions. PPO proves monotonic improvement for the policy's expected return by bounding the divergence from the previous policy in each update. This guarantee only holds if both policies are applied to the same residual MDP, i.e. if the ensemble is fixed. □

Combining Theorem 7 with Lemma 8 transfers the monotonic improvement guarantee to the original MDP $\mathcal{M}$.

Theorem 9. BRPO *monotonically improves upon the mixture between ensemble policy* π_e *and initial residual policy* π_{r_0}*, eventually converging to a locally optimal policy.*

Proof. From Lemma 8, we have that π_r monotonically improves on the residual MDP $\mathcal{M}_r$. From Theorem 7, monotonic improvement of π_r on $\mathcal{M}_r$ implies monotonic improvement of the mixture policy π on the actual MDP $\mathcal{M}$. If the initial residual policy's actions are small, the expected return of the mixture policy π on $\mathcal{M}$ is close to that of the ensemble π_e. □

In summary, BRPO tackles RL problems with model uncertainty by building on an ensemble of clairvoyant experts and optimizing a policy on the residual MDP induced by the ensemble. Even suboptimal ensembles often provide a strong baseline, resulting in data-efficient learning and high returns. We empirically evaluate this hypothesis in Section 7.3.

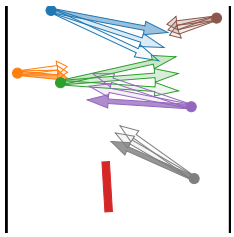
7.3 Experimental Results

We focus on problems highlighting common challenges for robots with model uncertainty. In these tasks, different latent MDPs require significantly different solutions and costly sensing is needed for disambiguation. Learned policies must balance robust actions in the face of uncertainty with uncertainty-reducing actions.

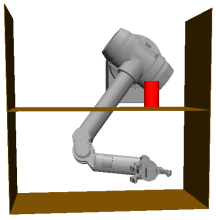
In all domains that we consider, BRPO improves on the ensemble's recommendation and significantly outperforms adaptive-RL baselines that do not leverage experts (Section 7.3.1). Qualitatively, robust Bayes-optimal behavior naturally emerges during training (Section 7.3.1). Our ablation studies demonstrate that both the belief and ensemble recommendation are valuable (Section 7.3.3) and that BRPO learns to reduce uncertainty without auxiliary information-gathering reward bonuses (Section 7.3.4). Finally, through physical experiments on the MuSHR racecar platform (Srinivasa et al., 2019), we demonstrate that BRPO agent significantly improves from a simple expert ensemble and is well-suited for real-robot tasks (Section 7.3.2).

7.3.1 Simulated Experiments

Crowd Navigation Inspired by Cai et al. (2019), an autonomous agent must quickly navigate past a crowd of people without collisions. Six people cross in front of the agent

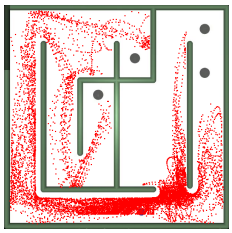


(a) `CrowdNav`

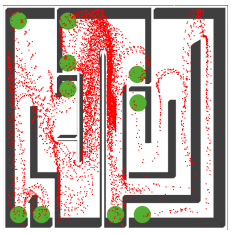


(b) `ArmShelf`

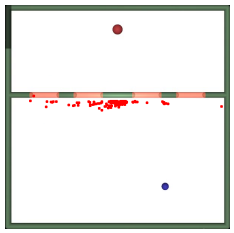
Figure 7.3: Setup for `CrowdNav` and `ArmShelf`. In `CrowdNav`, the goal for the agent (red) is to drive upward without colliding with pedestrians (other colors). In `ArmShelf`, the goal is to reach for the can.



(a) `Maze4`



(b) `Maze10`



(c) `Door4`

Figure 7.4: Sensing locations. In `Maze4` and `Maze10`, sensing is dense around the starting regions (bottom of `Maze4`, center of `Maze10`) and where multiple latent goals (gray, green) are nearby and must be disambiguated. In `Door4`, BRPO only senses when close to the doors, where the sensor is most accurate.

at fixed speeds, three from each side (Figure 7.3a). Each person noisily walks toward its latent goal on the other side, which is sampled uniformly from a discrete set of destinations. The agent observes each person's speed and position to estimate the belief distribution for each person's goal. There is a single expert which uses model predictive control: each walker is simulated toward a belief-weighted average goal position, and the expert selects cost-minimizing steering angle and acceleration.

At the beginning of the episode, initial pedestrian positions are sampled uniformly along the left and right sides of the environment. Speeds are sampled uniformly between 0.1 and 1.0 m/s. The agent observes each person's speed and position to estimate the goal distribution. The agent starts at the bottom of the environment, with initial speed sampled uniformly from 0 to 0.4 m/s. The agent controls acceleration and steering angle, bounded between ± 0.12 m/s^2 and ± 0.1 rad. Pedestrians are modeled as a 1m diameter circle. The agent is modeled as a rectangular vehicle of 0.5 m width and 2 m length. A collision results in a terminal cost of $100 \cdot (2v)^2 + 0.5$. Successfully reaching the top of the environment produces

terminal reward of 250, while navigating to the left or right side results in terminal cost of 1000. A per timestep penalty of 0.1 encourages the agent to complete the episode quickly.

Cartpole In this environment, the agent's goal is to keep the cartpole upright for as long as possible. The latent parameters are cart mass and pole length, uniformly sampled from $[0.5, 2.0]\text{kg} \times [0.5, 2.0]\text{m}$. The agent's estimator is a 3×3 discretization of the 2D continuous latent space, and the resulting belief is a categorical distribution over that grid. Each expert is a Linear-Quadratic Regulator (LQR) for the center of each grid square. The ensemble is the belief-weighted sum of experts.

The cartpole initializes with small initial velocity around the upright position. The environment terminates when the pole is more than 1.2 rad away from the vertical upright position or the cart is 4.0 m away from the center. The agent is rewarded by 1 for every step the cartpole survives. The environment has finite horizon of 500 steps.

Object Localization In the `ArmShelf` environment, the agent must localize an object without colliding with the environment or the object. The continuous latent variable is the object's pose, which is anywhere on either shelf of the pantry (Figure 7.3b). The agent receives a noisy observation of the object's pose upon sensing, which is less noisy as the end-effector approaches the object. The agent uses an Extended Kalman Filter to track the object's pose. The ensemble is the MAP expert which takes the MAP object pose and proposes a collision-free movement toward the object.

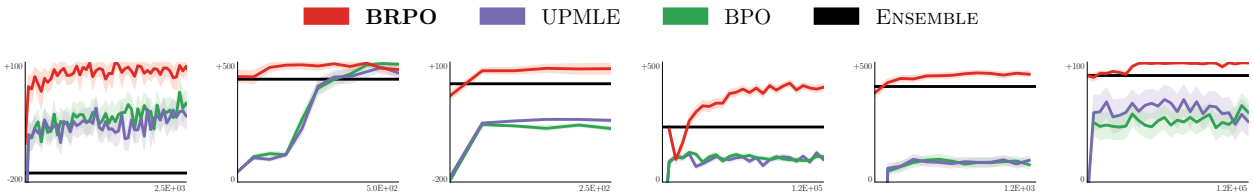
The agent can control the end-effector in the (x, y, z) directions. The goal is to move the hand to the object without colliding with the environment or object. The agent observes the top and bottom shelf poses, end-effector pose, arm configuration, and the noise scale. The noise scale is the standard deviation of the Gaussian noise on the agent's observation of the object's pose. Without sensing, the noise is very large: $w \sim \mathcal{N}(0, 5.0^2)$ where the width of the shelf is only 0.35 m, When sensing is invoked, the noise is reduced to $w \sim \mathcal{N}(0, d^2)$ where d is the distance between the object and the end-effector.

Latent Goal Mazes In the `Maze4` and `Maze10`, the agent must identify which latent goal is active. At the beginning of each episode, the latent goal is set to one of four or ten goals. The agent is rewarded for reaching the active goal and penalized for reaching an inactive goal. The agent receives a noisy measurement of the distance to the goal, with noise proportional to the true distance. Each expert proposes an action (computed via motion planning) that navigates to the corresponding goal. The ensemble recommends the belief-weighted sum of the experts' suggestions.

The agent observes its current position, velocity, and distance to all latent goals. If sensing is invoked, it also observes the noisy distance to the goal. In addition, the agent observes the categorical belief distribution over the latent goals. In `Maze4`, reaching the active goal provides a terminal reward of 500, while reaching an incorrect goal gives a penalty of 500. The task ends when the agent receives either the terminal reward or penalty, or after 500 timesteps. In `Maze10`, the agent receives a penalty of 50 and continues to explore after reaching an incorrect goal.

Doors There are four possible doors to the next room of the `Door4` environment. At the beginning of each episode, each door is opened or closed with 0.5 probability. To check the doors, the agent can either sense or crash into them (which costs more than sensing). Sensing returns a noisy binary vector for all four doors with exponentially-decreasing accuracy proportional to the distance to each door. Crashing returns an accurate indicator of the door it crashed into. Each expert navigates directly through the closest open door, and the ensemble recommends the belief-weighted sum of experts.

To check the doors, the agent can either sense (-1) or crash into them (-10). At every step, the agent observes its position, velocity, distance to goal, and whether it crashed or passed through a door. In addition, the agent observes the categorical distribution over the $2^4 = 16$ possible door configurations (from the Bayes filter) and the ensemble's recommendation. The agent receives a terminal reward of 100 if it reaches the goal within 300 timesteps.



(a) `CrowdNav` (b) `Cartpole` (c) `ArmShelf` (d) `Maze4` (e) `Maze10` (f) `Door4`

Figure 7.5: Training curves. BRPO dramatically outperforms agents that do not leverage expert knowledge (BPO, UP-MLE), and significantly improves the ensemble of experts.

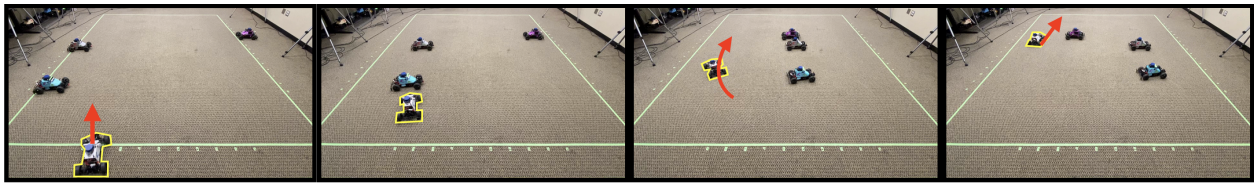
BRPO Improves Ensemble, Outperforms Adaptive Methods

We compare BRPO to adaptive RL algorithms that consider the belief over latent states: BPO and UP-MLE, a modification to (Yu et al., 2017) that augments the state with the Bayes filter's maximum likelihood estimate. Neither approach can incorporate experts.

We also compare with the ensemble of experts baselines, which does not take any sensing actions (as discussed in Section 7.2). For tasks requiring explicit sensing actions (`ArmShelf`, `Maze4`, `Maze10`, `Door4`), we strengthen the ensemble by sensing with probability 0.5 at each timestep. More sophisticated sensing strategies require more task-specific knowledge to design; see Section 7.3.5 for more discussion.

Figure 7.5 compares the training performance of all algorithms across the six environments. Note that BRPO's initial policy does not exactly match the ensemble: the random initialization for the residual policy network adds zero-mean noise around the ensemble policy, which may result in an initial drop relative to the ensemble (Figure 7.5c, Figure 7.5d).

On the wide variety of problems we have considered, BRPO agents perform dramatically better than BRPO and UP-MLE agents. BRPO and UP-MLE were unable to match the performance of BRPO, except on the simple `Cartpole` environment. This seems to be due to the complexity of the latent MDPs, discussed further in Section 7.3.6. In fact, for `Maze4` and `Maze10`, we needed to modify the reward function to encourage information-gathering



(a) Starts moving. (b) Waits for the blue car. (c) Takes slight detour. (d) Accelerates to the goal.

Figure 7.6: Rollout on `CarNav`, a modified `CrowdNav` for the physical MuSHR cars. The BRPO agent waits, detours, and accelerates around other cars to reach the goal quickly.

for BRPO and UP-MLE; without such reward bonuses, they were unable to learn any meaningful behavior. We study the effect that such a reward bonus would have on BRPO in Section 7.3.4. For `Cartpole`, both BRPO and UP-MLE learned to perform optimally but required much more training time than BRPO.

BRPO Learns Bayes-Optimal Behavior

For `Maze4`, `Maze10` and `Door4`, we have visualized where the agent invokes explicit sensing (Figure 7.4). For `Maze4` and `Maze10`, the BRPO agent learns to sense when goals must be distinguished, e.g. whenever the road diverges. For `Door4`, it senses only when that is most cost-effective: near the doors, where accuracy is highest. This results in a rather interesting policy in which the agent dashes to the wall, senses only once or twice, and drives through the closest open door. The BRPO agent avoids crashing in almost all scenarios. We refer the reader to Section 7.3.6 for more qualitative analysis, including keyframes of a few representative trajectories.

7.3.2 MuSHR Car Experiment

We modify `CrowdNav` to run an experiment with MuSHR cars (Srinivasa et al., 2019). The BRPO agent controls one, while three others represent pedestrians (reduced from `CrowdNav` due to space constraints). Each car is roughly 30 cm wide and 50 cm long, and is controlled by forward velocity and steering angle. Poses for all cars are tracked using an array of twelve

		BRPO	Ensemble
Real	Success Rate (%)	**96.6 (29/30)**	36.6 (11/30)
	Navigation Time (s)	$\mathbf{12.4 \pm 0.2}$	18.3 ± 0.8
Simulation	Success Rate (%)	$\mathbf{97.2 \pm 0.04}$	24.0 ± 0.2
	Navigation Time (s)	$\mathbf{6.3 \pm 0.2}$	10.5 ± 0.1

Table 7.1: Comparison of BRPO and the expert ensemble on the `CarNav` environment. In both simulation and on the physical system, BRPO succeeds much more often and requires less time to navigate because it accelerates when safe. Navigation time is only measured for successful trials.

OptiTrack PrimeX 22 cameras. As before, the agent aims to navigate past the "pedestrians" as they noisily move toward their latent goals (Figure 7.6). Section **??** contains further details.

We use a very simple ensemble to represent computational constraints that may be present with a physical robot. There is only one expert in this ensemble, which assumes that the pedestrians will remain static as it plans forward. It uses model predictive control to avoid these obstacles as it navigates toward its goal, and is restricted to a fixed forward velocity of 0.4 m/s and steering angle between $[-0.2, 0.2]$ radians. We train the agent in simulation and execute directly on the car.

BRPO improves on this very simple baseline, successfully completing the task without collisions in 29 of 30 real-world trials. Table 7.1 compares the performance of BRPO with the ensemble in both the real and simulated environments. In both cases, BRPO dramatically improves on the ensemble's success rate. Furthermore, the BRPO agent reduces the navigation time by 36.7% in simulation and 32.2% with the physical robot, indicating that it learns to navigate both safely and quickly.

Qualitatively, the BRPO agent often starts slowly. The pedestrians' latent goals are

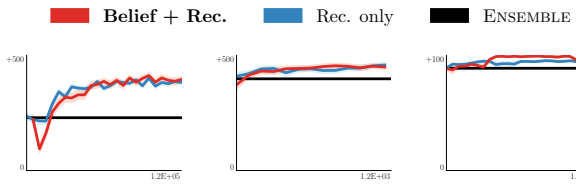
usually clearer by the time it passes the first car, at which point it can accelerate. Depending on the latent goals, the agent occasionally waits for pedestrians to pass or detours around them. Since the expert moves at a fixed velocity, all deceleration and acceleration emerges naturally from training with BRPO. Figure 7.6 shows snapshots from one rollout to illustrate some of this behavior; more recorded trials are available in the supplementary video.

MuSHR Car Experiment Details The "pedestrians" cross a region of 3.5 m width and 5 m height, randomly starting from either side. The y-coordinates of their initial positions and goals are randomly chosen between $[1.5, 3.8]$ m such that they can move straight to their goals without collision. The agent only observes their poses and velocities, with which it runs a Bayes filter over their latent goal positions. Poses and velocities for all cars are tracked with an array of twelve OptiTrack PrimeX 22 cameras. At the beginning of each episode, pedestrians start moving after a random delay between $[0, 1]$ seconds, with random speeds chosen between $[0.5, 0.6]$ m/s. The BRPO agent starts at the bottom row, $y = -0.1$ m. The goal is to navigate to $y = 4.5$ m as quickly as possible without hitting the pedestrians or passing the environment border, drawn in green in Figure 7.6.

Simulated Training Environment To mimic the physical robot's dynamics, the velocity is scaled by a factor of 0.8. Furthermore, because the car often stops when the commanded velocity drops below 0.3 m/s, the simulation rounds values below that threshold down to 0. The scaled velocity is clamped to the maximum velocity of 0.6 m/s. Gaussian noise is clamped at one standard deviation and added to the velocity and steering angle, with standard deviations of 0.1 m/s and 0.05 radians.

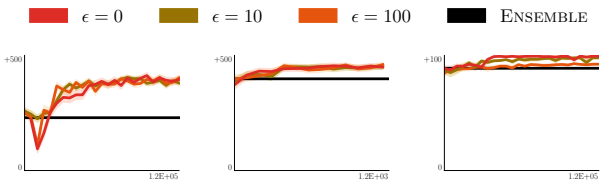
In training, the agent is rewarded 10 for successfully reaching the terminal condition $y = 4.5$. When the agent is within 0.65 m of any other cars or 0.5 m of the left or right borders, the agent receives a penalty of -10 and the episode terminates. Otherwise, the agent receives $-0.005 + (4.5 - y) - 0.05\|\theta\|$ where y is the car's y-coordinate and θ is its rotation, counterclockwise from y-axis. Unlike the ensemble, which has fixed velocity and

only chooses the steering angle, the BRPO agent outputs both velocity and steering angle corrections. The final commanded velocity is computed as above, and the steering angle is clamped between $[-0.3, 0.3]$ radians.



(a) `Maze4` (b) `Maze10` (c) `Door4`

Figure 7.7: Ablation study on input features. Including both belief and recommendation results in faster learning in `Door4`.



(a) `Maze4` (b) `Maze10` (c) `Door4`

Figure 7.8: Ablation study on information-gathering reward (Equation 7.10). BRPO is robust to information-gathering reward.

7.3.3 Ablation Study: Residual Policy Inputs

The BRPO policy takes the belief distribution, state, and ensemble recommendation as inputs (Figure 7.2). We considered two versions of BRPO with different inputs: only recommendation (which implicitly encodes belief), and one with both recommendation and belief.

Figure 7.7 shows that providing both belief and recommendation as inputs to the policy is important. Although BRPO with only the recommendation performs comparably to BRPO with both inputs on `Maze4` and `Maze10`, the one with both inputs learns faster on `Door4`.

7.3.4 Ablation Study: Information-Gathering Reward Bonuses

Because BRPO maximizes the Bayesian Bellman equation (Equation 4.1), exploration is incorporated into its long-term objective. As a result, auxiliary rewards to encourage exploration are unnecessary. However, existing work that does not explicitly consider the belief has suggested various auxiliary reward terms to encourage exploration, such as surprisal

rewards (Achiam & Sastry, 2017) or intrinsic rewards (Pathak et al., 2017). To investigate whether such rewards benefit the BRPO agent, we augment the reward function with the following auxiliary bonus from Chen et al. (2016):

$$\tilde{r}(s, b, a) = r(s, b, a) + \epsilon \cdot \mathbb{E}_{b'} \left[\|b - b'\|_1 \right] \tag{7.10}$$

where $\|b - b'\|_1 = \sum_{i=1}^{k} |b(\phi_i) - b'(\phi_i)|$ rewards change in belief.

Figure 7.8 summarizes the performance of BRPO when training with $\epsilon = 0, 10, 100$. Too much emphasis on information-gathering causes the agent to over-explore and therefore underperform. In `Door4` with $\epsilon = 100$, we qualitatively observe that the agent crashes into the doors more often. Crashing significantly changes the belief for that door; the huge reward bonus outweighs the penalty of crashing from the environment.

We find that BRPO and UP-MLE are unable to learn without an exploration bonus on `Maze4`, `Maze10`, and `Door4`. We used $\epsilon = 1$ for `Maze4` and `Door4`, and $\epsilon = 100$ for `Maze10`. Upon qualitative analysis, we found that the bonus helps BRPO and UP-MLE learn to sense initially, but the algorithms are unable to make further progress. We believe that this is because solving the latent mazes is challenging.

7.3.5 Ablation Study: Better Sensing Ensemble

The ensemble we used for training BRPO in Figure 7.5 randomly senses with probability 0.5. An ensemble baseline policy that senses more effectively could be designed manually, and used as the initial policy for the BRPO agent to improve on. Note that in general, designing such a policy can be challenging: it requires either task-specific knowledge, or solving an approximate Bayesian RL problem. We bypass these requirements by using BRPO.

On the `Maze10` environment, we have found via offline tuning that a more effective ensemble baseline agent senses only for the first 150 of 750 timesteps. Table 7.2 shows that BRPO results in higher average return and success rate. The performance gap comes from the suboptimality of the ensemble recommendation, as experts are unaware of the penalty for reaching incorrect goals.

	BRPO	RandomSensing	BetterSensing
Avg. Return	**465.7** ± 4.7	409.5 ± 10.8	416.3 ± 9.4
Success Rate	**100%**	-	96.3%

Table 7.2: Comparison of BRPO and ensembles on `Maze10`.

7.3.6 Qualitative Behavior Analysis

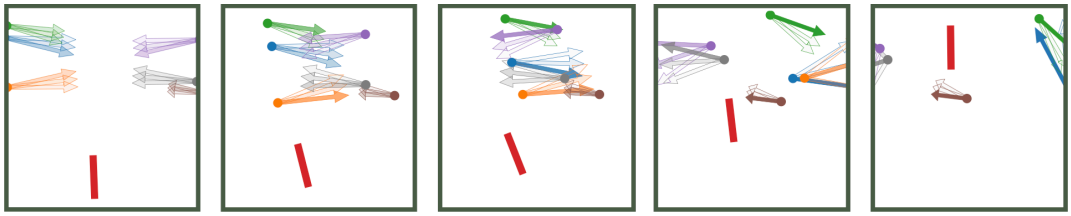
Figure 7.9 shows some representative trajectories taken by BRPO agents. Across multiple environments (`CrowdNav`, `Maze4`, `Maze10`), we see that the BRPO agent adapts to the evolving posterior. As the posterior over latent goals updates, the agent shifts directions. While this rerouting partly emerges from the ensemble policies as the posterior sharpens, BRPO's residual policy reduces uncertainty (`Maze4`, `Maze10`) and pushes the agent to navigate faster, resulting in higher performance than the ensembles.

7.4 Discussion

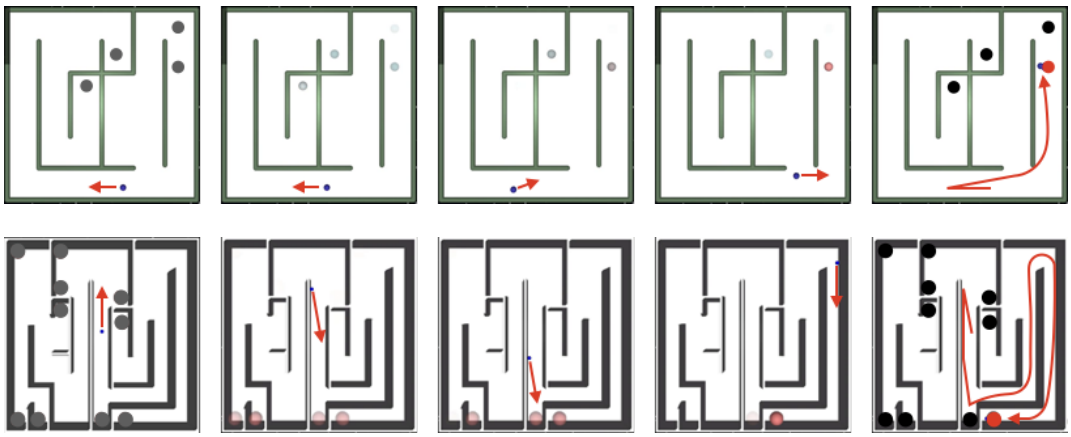
Our algorithm, Bayesian Residual Policy Optimization, builds on an ensemble of experts by operating within the resulting residual belief MDP. We prove that this strategy preserves guarantees, such as monotonic improvement, from the underlying policy optimization algorithm. The scalability of policy gradient methods, combined with task-specific expertise, enables BRPO to quickly solve a wide variety of complex problems, such as navigating through a crowd of pedestrians. BRPO improves on the original ensemble of experts and achieves much higher rewards than existing Bayesian RL algorithms by sensing more efficiently and acting more robustly.

Although out of scope for this work, a few key challenges remain. First is an efficient construction of an ensemble of experts, which becomes particularly important for continuous latent spaces with infinitely many MDPs. Infinitely many MDPs do not necessarily require

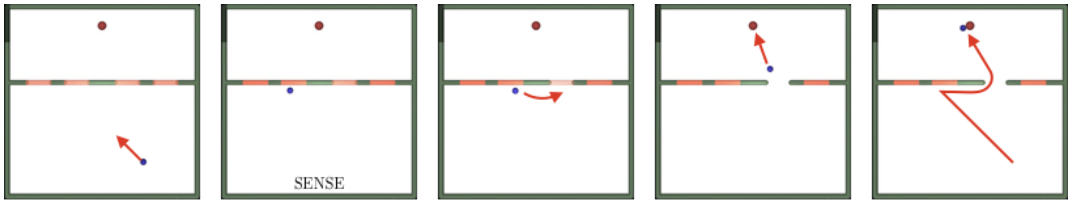
infinite experts, as many may converge to similar policies. An important future direction is subdividing the latent space and computing a qualitatively diverse set of policies. Another challenge is developing an efficient Bayes filter, which is an active research area. In certain occasions, the dynamics of the latent MDPs may not be accessible, which would require a learned Bayes filter. Combined with a tractable, efficient Bayes filter and an efficiently computed set of experts, we believe that BRPO will provide an even more scalable solution for BRL problems.



(a) `CrowdNav`. Arrows are the directions to discrete latent goals; transparency indicates the posterior probability of the corresponding goal, and its length indicates the speed. The agent changes direction as it anticipates collision.



(b) Latent goal mazes with four (`Maze4`) and ten (`Maze10`) possible goals. The agent senses as it navigates, changing its direction as goals are deemed less likely (more transparent). We have marked the true goal with red in the last frame for clarity.



(c) `Door4`. The agent senses only when it is near the wall with doors, where sensing is most accurate. The transparency of the red bars indicates the posterior probability that the door is blocked. With sensing, the agent notices that the third door is open.

Figure 7.9: BRPO policy keyframes. Best viewed in color.

Chapter 8

CONCLUSION

This thesis addresses the problem of solving complex control tasks in continuous state and action spaces under model uncertainty. We formulate this problem as Bayesian Reinforcement Learning, in which latent models are tracked as beliefs. To address the challenge of scalability, we present algorithms that leverage structural priors to improve data-efficiency and accelerate learning. Through mathematical analyses we bound the sample complexity of a PAC-Bayes algorithm, BCPACE, and prove that BRPO improves monotonically from an ensemble of experts. Through demonstrations in simulation and physical tasks, we show that our algorithms outperform adaptive RL algorithms and are well suited for real robot tasks that require fast and reactive controls.

The empirical success of our algorithms partially depends on a set of assumptions we have made. Mainly, we assume that (1) the distribution over latent states can be represented compactly, that (2) model-based Bayes filters are available, and that (3) the model prior approximately matches the real world. While these assumptions hold for many practical applications, relaxing them can extend our algorithms to a more diverse set of problems. In the remainder of this chapter, we discuss the assumptions and how one may relax them.

Efficient choice of latent models and experts Although BPO and BRPO can handle any belief representation and scale to fine-grained discretizations of latent space, our experiments suggest that each problem has an optimal discretization level, beyond which further discretization may degrade performance. For BRPO, the discretization extends to the construction of an ensemble of experts. For these algorithms, an efficient choice of latent models and experts can significantly reduce the computational burden. Even when

the model prior assumes infinitely many MDPs, it does not necessarily require an infinite number of nominal models or experts, as many may have similar dynamics and converge to similar policies. It may be preferable to perform variable-resolution discretization rather than single-resolution discretization. Adapting iterative densification ideas previously explored in motion planning (Gammell et al., 2015) and optimal control (Munos & Moore, 1999) to the discretization of latent space may yield a compact belief representation and improved performance.

Learned belief representations Across all our Bayesian RL algorithms, we have used model-based Bayes filters, which is commonly used in many POMDP algorithms. On certain occasions, however, the dynamics of the latent MDPs may not be accessible, which would require a learned Bayes filter. Combined with tractable, efficient Bayes filters, our algorithms will provide an even more scalable solution for BRL problems.

An alternative to the model-based Bayes filter is learning to directly map a history of observations to a lower-dimensional belief embedding, analogous to Peng et al. (2018). This would enable a policy to learn a meaningful belief embedding without losing information from our a priori choice of discretization. We could also utilize a likelihood-free posterior inference method introduced in Papamakarios & Murray (2016), which trains a Bayesian conditional density estimator that takes in trajectories and outputs a set of parameters that defines the distribution over the latent parameters.

Improving model priors While BPO and BRPO provide Bayes-optimal policies, they do not improve model priors nor capture the unmodeled part of the real world. Ideally, the model prior used by our agent should improve and adapt to match the real world. To achieve a Bayesian RL framework whose model prior improves over time, we can combine BPO and BRPO with model learning. The agent can iteratively improve the model prior from data and update its policy using the improved prior. Analogous to GP-ILQG, we can capture the unmodeled component of the real-world through semi-parametric model learning.

BIBLIOGRAPHY

Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In William W. Cohen and Andrew Moore (eds.), *Proceedings of the 23th International Conference on Machine Learning (ICML-06)*, pp. 1–8, 2006. URL http://www.machinelearning.org/proceedings/icml2006/001_Using_Inaccurate_Mod.pdf.

Douglas Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes. *National ICT Australia, Canberra, Australia*, 2003.

Douglas Aberdeen and Jonathan Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *International Conference on Machine Learning*, 2002.

Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

John Asmuth, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate. A bayesian sampling approach to exploration in reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence*, 2009.

Christopher G Atkeson, Andrew W Moorey, and Stefan Schaalz. Locally weighted learning. *Artif Intell Rev*, 11(1-5):11–73, 1997.

Haoyu Bai, David Hsu, and Wee Sun Lee. Integrated perception and planning in the continuous space: A pomdp approach. *The International Journal of Robotics Research*, 33 (9), 2014.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.

Tamer Başar and Pierre Bernhard. *H-infinity optimal control and related minimax design problems: a dynamic game approach*. Springer Science & Business Media, 2008.

Jonathan Baxter. Theoretical models of learning to learn. In *Learning to learn*, pp. 71–94. Springer, 1998.

Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

Felix Berkenkamp and Angela P Schoellig. Safe and robust learning control with gaussian processes. In *2015 European Control Conference (ECC)*, pp. 2496–2501. IEEE, 2015.

Adrian Boeing and Thomas Bräunl. Leveraging multiple simulators for crossing the reality gap. In *Control Automation Robotics & Vision (ICARCV), 2012 12th International Conference on*, pp. 1113–1119. IEEE, 2012.

Ronen I Brafman and Moshe Tennenholtz. R-max - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Panpan Cai, Yuanfu Luo, Aseem Saxena, David Hsu, and Wee Sun Lee. Lets-drive: Driving in a crowd by learning from tree search. *arXiv preprint arXiv:1905.12197*, 2019.

Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. POMDP-lite for robust robot planning under uncertainty. In *IEEE International Conference on Robotics and Automation*, 2016.

Ching-An Cheng, Andrey Kolobov, and Alekh Agarwal. Policy improvement from multiple experts. *arXiv preprint arXiv:2007.00795*, 2020.

Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *The International Journal of Robotics Research*, 37(13-14):1632–1672, 2018.

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *AAAI Conference on Artificial Intelligence*, 1998.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 2016a.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016b.

Michael O'Gordon Duff and Andrew Barto. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts at Amherst, 2002.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

Yoav Freund and Robert Schapire. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2, 2015.

Jonathan Gammell, Siddhartha Srinivasa, and Timothy Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation*, 2015.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6): 359–483, 2015.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

Arthur Guez, David Silver, and Peter Dayan. Efficient Bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, 2012.

Arthur Guez, Nicolas Heess, David Silver, and Peter Dayan. Bayes-adaptive simulation-based search with value function approximation. In *Advances in Neural Information Processing Systems*, 2014.

Iain Guilliard, Richard J Rogahn, Jim Piavis, and Andrey Kolobov. Autonomous thermalling as a partially observable markov decision process. In *Robotics: Science and Systems*, 2018.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pp. 5302–5311, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

David Hsu, Nan Rong, and Wee S Lee. What makes some pomdp problems easy to approximate? In *Advances in Neural Information Processing Systems*, 2008.

Dongsung Huh and Emanuel Todorov. Real-time motor control using recurrent neural networks. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pp. 42–49. IEEE, 2009.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. *arXiv preprint arXiv:1806.02426*, 2018.

David H Jacobson and David Q Mayne. Differential dynamic programming. 1970.

S. Javdani, S.S. Srinivasa, and J.A. Bagnell. Shared autonomy via hindsight optimization. In *Robotics: Science and Systems*, 2015.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029. IEEE, 2019.

Leslie Pack Kaelbling, Michael Littman, and Anthony Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Sham Kakade, Michael J Kearns, and John Langford. Exploration in metric state spaces. In *International Conference on Machine Learning*, 2003.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. PhD thesis, University College London (University of London), 2003.

R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-Net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, 2017.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3): 193–208, 2002.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Svetoslav Kolev and Emanuel Todorov. Physically consistent state estimation and system identification for contacts. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pp. 1036–1043. IEEE, 2015.

Zico Kolter and Andrew Ng. Near-Bayesian exploration in polynomial time. In *International Conference on Machine Learning*, 2009.

Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

Gilwoo Lee, Brian Hou, Aditya Mandalika, Jeongseok Lee, Sanjiban Choudhury, and Siddhartha S. Srinivasa. Bayesian policy optimization for model uncertainty. In *International Conference on Learning Representations*, 2019.

Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML (3)*, pp. 1–9, 2013.

Lihong Li. *A unifying framework for computational reinforcement learning theory*. PhD thesis, Rutgers University-Graduate School-New Brunswick, 2009.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems.

Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

Michael Littman, Anthony Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 1995a.

Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings*, pp. 362–370. 1995b.

CK Liu and S Jain. A short tutorial on multibody dynamics. *Georgia Institute of Technology, School of Interactive Computing, Tech. Rep. GIT-GVU-15-01-1*, 8, 2012.

Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI Conference on Artificial Intelligence*, 1999.

MATLAB. *version 9.1.0 (R2016b)*. The MathWorks Inc., Natick, Massachusetts, 2016.

Tad McGeer. Passive Dynamic Walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990. doi: 10.1177/027836499000900206. URL http://ijr.sagepub.com/content/9/2/62.abstract.

Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. *CoRR*, abs/1904.00956, 2019a.

Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. *arXiv preprint arXiv:1904.00956*, 2019b.

Djordje Mitrovic, Stefan Klanke, and Sethu Vijayakumar. Adaptive optimal feedback control

with learned internal dynamics models. In *From Motor Learning to Interaction Learning in Robots*, pp. 65–84. Springer, 2010.

Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 242–248. IEEE, 2016.

Jun Morimoto and Kenji Doya. Robust reinforcement learning. In *Advances in Neural Information Processing Systems*, 2001.

Remi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *International Joint Conference on Artificial Intelligence*, 1999.

Kumpati S Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks*, 1(1):4–27, 1990.

Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

Pedro A. Ortega, Jane X. Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alexander Pritzel, Pablo Sprechmann, Siddhant M. Jayakumar, Tom McGrath, Kevin Miller, Mohammad Gheshlaghi Azar, Ian Osband, Neil C. Rabinowitz, András György, Silvia Chiappa, Simon Osindero, Yee Whye Teh, Hado van Hasselt, Nando de Freitas, Matthew Botvinick, and Shane Legg. Meta-learning of sequential strategies. *arXiv preprint arXiv:1905.03030*, 2019.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, 2013.

Chris J Ostafew, Angela P Schoellig, and Timothy D Barfoot. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. In *IEEE International Conference on Robotics and Automation*, pp. 4029–4036. IEEE, 2014.

Chris J Ostafew, Angela P Schoellig, and Timothy D Barfoot. Conservative to confident: treating uncertainty robustly within learning-based control. In *IEEE International Conference on Robotics and Automation*, pp. 421–427. IEEE, 2015.

Yunpeng Pan and Evangelos Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*, pp. 1907–1915, 2014.

Yunpeng Pan and Evangelos A Theodorou. Data-driven differential dynamic programming using gaussian processes. In *American Control Conference (ACC), 2015*, pp. 4467–4472. IEEE, 2015.

Yunpeng Pan, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Scalable reinforcement learning via trajectory optimization and approximate gaussian process regression. *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2015.

Christos Papadimitriou and John Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

George Papamakarios and Iain Murray. Fast ϵ-free inference of simulation models with bayesian conditional density estimation, 2016.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *International Conference on Autonomous Agents and Multiagent Systems*, 2018.

Jason Pazis and Ronald Parr. Pac optimal exploration in continuous space markov decision processes. In *AAAI Conference on Artificial Intelligence*, 2013.

Jason Pazis and Ronald Parr. Efficient pac-optimal exploration in concurrent, continuous state mdps with delayed updates. In *AAAI Conference on Artificial Intelligence*, 2016.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation*, 2018.

Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference on Artificial Intelligence*, 2003a.

Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, 2003b.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, 2017.

Robert Platt, Russ Tedrake, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *International Conference on Machine Learning*, 2006.

Joaquin Quiñonero-Candela, Carl Edward Rasmussen, AnÃbal R Figueiras-Vidal, et al. Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11 (Jun):1865–1881, 2010.

Neil C. Rabinowitz. Meta-learners' learning dynamics are unlike learners'. *arXiv preprint arXiv:1905.01320*, 2019.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. EPOpt: Learning robust neural network policies using model ensembles. In *International Conference on Learning Representations*, 2017.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *Journal of Machine Learning Research*, 11(Nov):3011–3015, 2010.

Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. 2012.

Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptive POMDPs. In *Advances in Neural Information Processing Systems*, 2008.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Journal on Autonomous Agents and Multiagent Systems*, 27(1):1–51, 2013.

David Silver and Joel Veness. Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, 2010.

Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

Jonas Sjöberg, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre-Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, 1995.

Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 520–527. AUAI Press, 2004.

Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. In *UAI*, 2005.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *Advances in Neural Information Processing Systems*, 2013.

Edward J Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations research*, 26(2):282–304, 1978.

Matthijs TJ Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

Siddhartha S. Srinivasa, Patrick Lancaster, Johan Michalove, Matt Schmittle, Colin Summers, Matthew Rockett, Joshua R. Smith, Sanjiban Chouhury, Christoforos Mavrogiannis, and Fereshteh Sadeghi. MuSHR: A low-cost, open-source robotic racecar for education and research. *CoRR*, abs/1908.08031, 2019.

Bradly C. Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel,

and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

Anton A Stoorvogel. The h-infinity control problem: A state space approach. 1993.

Alexander L Strehl and Michael L Littman. Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems*, 2008.

Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *International Conference on Machine Learning*, 2006.

Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009.

Malcolm Strens. A Bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, 2000.

Zachary Sunberg and Mykel Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *International Conference on Automated Planning and Scheduling*, 2018.

Zachary Sunberg and Mykel J. Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. *preprint arXiv:1709.06196*, 2017.

Jie Tan, Zhaoming Xie, Byron Boots, and C Karen Liu. Simulation-based design of dynamic controllers for humanoid balancing. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 2729–2736. IEEE, 2016.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 6054–6061. IEEE, 2014.

Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pp. 300–306. IEEE, 2005.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

Jur van den Berg. Extended lqr: Locally-optimal feedback control for systems with non-linear dynamics and non-quadratic cost. In *Robotics Research*, pp. 39–56. Springer, 2016.

Jur van den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

Jack M Wang, David J Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, volume 29, pp. 73. ACM, 2010.

Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *International Conference on Machine Learning*, 2005.

Yi Wang, Kok Sung Won, David Hsu, and Wee Sun Lee. Monte Carlo Bayesian reinforcement learning. In *International Conference on Machine Learning*, 2012.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence*, 2001.

Akihiko Yamaguchi and Christopher G Atkeson. Differential dynamic programming with temporally decomposed dynamics. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pp. 696–703. IEEE, 2015.

Akihiko Yamaguchi and Christopher G Atkeson. Neural networks and differential dynamic programming for reinforcement learning problems. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 5434–5441. IEEE, 2016.

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pp. 7332–7342, 2018.

Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Robotics: Science and Systems*, 2017.

Juan Cristóbal Zagal, Javier Ruiz-del Solar, and Paul Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. In *IAV 2004 the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal*, 2004.

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 528–535. IEEE, 2016.