Focused Model-Learning and Planning for Non-Gaussian Continuous State-Action Systems

Zi Wang, Stefanie Jegelka, Leslie Pack Kaelbling, and Tomás Lozano-Pérez

MIT CSAIL, {ziw,stefje,lpk,tlp}@csail.mit.edu

Abstract. We introduce a joint framework for model leaning and planning in stochastic domains with continuous state and action spaces with non-Gaussian transition noise. It is efficient in large spaces with large amounts of data because (1) local models are estimated only when the planner requires them; (2) the planner focuses on the most relevant states to the current planning problem; and (3) the planner focuses on the most informative and/or high-value actions. Our theoretical analysis shows that the expected difference between the optimal value function of the original problem and the value of the policy we compute vanishes sub-linearly in the number of actions we test, under mild assumptions. We show empirically that multi-modal transition models are necessary if the underlying dynamics is not single-mode, and our algorithm is able to complete both learning and planning within minutes for a stochastic pushing problem in simulation given more than a million data points, as a result of focused planning.

1 Introduction

Most real-world domains are sufficiently complex that it is difficult to build an accurate deterministic model of the effects of actions. Even with highly accurate actuators and sensors, stochasticity still appears in basic manipulations such as pushing (Yu et al., 2016). The stochasticity may come from inaccurate execution of actions as well as from lack of detailed information about the underlying world state. For example, rolling a die is a deterministic process that depends on the forces applied, the surface of the table, etc.; however, we are not able to model the situation sufficiently accurately to plan reliable actions, nor to execute them repeatably if we could plan them. We can plan using a stochastic model of the system, but in many situations, such as pushing objects or rolling dice, the stochasticity is not modeled well by additive single-mode Gaussian noise, and so we need a more sophisticated model class.

In this paper, we address the problem of learning and planning for non-Gaussian stochastic systems in the practical setting of continuous state and action spaces. Instead of trying to construct a highly accurate physical simulator that will inevitably make inaccurate assumptions about the environment and the interaction between robots and objects, e.g. Coulomb friction law and uniform pressure distribution, we learn a stochastic model of the robot's interaction with the environment from data. Then, to make plans using the learned models, we propose a novel planning algorithm for non-Gaussian continuous state-action systems.

Our framework is modular, separating model-learning from planning. Doing so allows the learned models to be used for planning to achieve different objectives in the

same domain, as well as to be potentially transferred to related domains or even different types of robots. This strategy is in contrast to most reinforcement-learning approaches, which build the objective into the structure being learned. In addition, rather than constructing a single monolithic model of the entire domain, our method uses a memory-based learning scheme, and computes localized models on the fly, only when the planner requires them. To avoid constructing models that do not contribute to improving the policy, we design our planner to focus only on states relevant to the current planning problem, and actions that are informative and/or lead to high reward.

Given a stochastic model of state transitions in the continuous state-action space, we need a strategy for planning. Recently, Huynh et al. (2016) introduced the incremental Markov Decision process algorithm (iMDP), which incrementally generates a sequence of Markov Decision Processes to approximate the original continuous-time/space problem. iMDP assumes the process dynamics can be described by a Wiener process, in which transition noise is Gaussian with execution-time-dependent variance. We generalize the iMDP algorithm to apply to stochastic systems with transition noise that can be well-modeled using a mixture of Gaussians. In addition, we take two steps to focus its computation on the current problem instance, defined by the starting state and goal region. To focus on relevant states, we use real time dynamic programming (RTDP) (Barto et al., 1995) rather than value iteration; to focus selection of actions from a continuous space, we propose a new batch Bayesian optimization technique that greedily optimizes a submodular acquisition function to select action candidates to test in parallel.

2 Related Work

Learning The class of problems that we address can be seen as reinforcement-learning (RL) problems in observable continuous state-action spaces. One way to address the problem would be through model-free RL, which seeks to estimate a value function or policy directly through experience. Although the majority of work in RL addresses domains with discrete action spaces, there has been a thread of relevant work on value-function-based RL in continuous action spaces (Gullapalli et al., 1994; Baird and Klopf, 1993; Prokhorov and Wunsch, 1997; van Hasselt and Wiering, 2007; Nichols, 2015). An alternative approach is to do direct search in the space of policies (Deisenroth and Rasmussen, 2011; Jakab and Csató, 2012).

In continuous state-action spaces, a model-based RL approach, in which a model is estimated and used to optimize a policy, can often be more effective. Gaussian processes (GP) can be used to learn the dynamics (Deisenroth et al., 2008; Rottmann and Burgard, 2009; Nguyen-Tuong and Peters, 2011), which can be used by GP-based dynamic programming (Deisenroth et al., 2009; Rottmann and Burgard, 2009) to determine a continuous-valued closed-loop policy for the whole state space. We refer interested readers to an excellent survey on Gaussian processes and reinforcement learning by Ghavamzadeh et al. (2015).

Unfortunately, the uniform i.i.d Gaussian noise assumption on the dynamics is restrictive and may not hold in practice (Yu et al., 2016), and the transition model can be multi-modal; it may additionally be difficult to obtain a good GP prior. The basic GP al-

gorithm is not able to capture either the multi-modality or the heteroscedasticity. While more advanced GP algorithms may address these problems, they often suffer from high computational cost (Titsias and Lázaro-Gredilla, 2011; Yuan and Neubauer, 2009).

Moldovan et al. (2015) addressed the problem of multi-modality by using Dirichlet process mixture models (DPMMs) to learn the density of the transition models. Their strategies for planning using the learned model were limited by deterministic assumptions, appropriate for their domains of application, but potentially resulting in collisions in ours. Kopicki et al. (2009); Kopicki (2010); Kopicki et al. (2011) addressed the problem of learning to predict the behavior of rigid objects under manipulations such as pushing, using kernel density estimation. In this paper, we propose an efficient planner that can work with multi-modal stochastic models in continuous state-action space. Our learning method resembles DPMMs but we estimate the density on the fly when the planner queries a state-action pair. We were not able to compare our approach with DPMMs because we found DPMMs not computationally feasible for large datasets (e.g. more than a million data points).

Planning We are interested in domains for which many queries that specify a starting state and a goal set may be made, and in which the solution to any given query can be described using a policy that covers only a small fraction of the state space. For this reason, we do not need to solve for a policy over the entire state space, but rather to plan for a limited range of the space that the robot is likely to encounter as it traverses from the starting state to the goal region.

Our planning method is directly based on the iMDP method of Huynh et al. (2016), which uses sampling techniques from RRTs to create successively more accurate discrete MDP approximations of the original continuous MDP, ultimately converging to the optimal solution to the original problem. Their model assumes Wiener-process transitions, which are too limited to model our domains of interest, and finds a policy over a larger subspace than may be necessary. iMDP is, in turn, related to the stochastic motion roadmap (Alterovitz et al., 200), which also assumes Gaussian transition uncertainty.

Bayesian optimization There have been a number of applications of Bayesian optimization in optimal control, though to our knowledge, it has not been previously applied to action-selection in continuous-action MDPs. Frean and Boyle (2008) applied Bayesian optimization to find weights in a neural network controller; Brochu et al. (2010) used Bayesian optimization to solve for the parameters of a hierarchical MDP. More recently, Turchetta et al. (2016) adopted Gaussian process optimization to address the problem of safely exploring the states of finite MDPs.

3 Background

Let the state space $S \subset \mathbb{R}^{d_s}$ and the control space $U \subset \mathbb{R}^{d_u}$ both be bounded compact sets. We assume the state is fully observed (any remaining latent state will manifest as stochasticity in the transition models). Actions $a = (u, \Delta t)$ are composed of both a control on the robot and the duration for which it will be exerted, so the action space is $A = U \times \mathbb{R}_+$. The goal region is denoted as $\mathcal{G} \subset S$, and states in the goal region $\mathcal{G}$

are terminal states. The *transition model* has the form of a probability density function $p_{s'|s,a}$ on the resulting state s', given previous state s and action a, such that $\forall s' \in S, p_{s'|s,a}(s'|s,a) \geq 0, \int_S p(s'|s,a)\,\mathrm{d}s' = 1$.

Given a transition model, a goal region, and a cost function $C : S \times S \times A \to \mathbb{R}$, we can characterize the problem as a continuous state-action MDP $(S, A, p_{s'|s,a}, R, \gamma)$, where $R(s'|s,a) = -C(s'|s,a)$ is the immediate reward function and γ is the discount factor. Without loss of generality, we assume the same reward for all states in the goal region, and the same cost for all states on the boundary of the free space (e.g., states in which the robot collides).

It is common practice to approach the solution of continuous MDPs by constructing and solving approximations in the form of discrete state-action MDPs. (Huynh et al., 2016) pursued this approach for stochastic motion-planning domains, using RRTs with backward extension to sample states for successive discrete approximate MDPs, and obtaining probabilistic completeness by covering the possible paths with probability 1 while keeping the at least one goal state always in the state set. In this paper, we adopt a very similar strategy, but using RRT with forward extension to handle domains in which the dynamics are not reversible, and dealing with non-Gaussian transition noise.

Once we have a discrete-state continuous-action MDP $(\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$, we can solve for a deterministic policy via value iteration. Here $\tilde{S}$ is a finite set of states that satisfy $\tilde{S} \subset S$, $\hat{P}_{s'|s,a}(s'|s,a)$ is the probability mass function for the transition from state $s \in \tilde{S}$ and action $a \in A$ to a new state $s' \in \tilde{S}$, $R(s'|s,a)$ is the immediate reward of the transition for $s, s' \in \tilde{S}, a \in A$, and γ is the discount factor of the rewards, the same as in the continuous case.

Given a discrete MDP and policy π, the value of each state s at iteration n is

$$\tilde{V}^{\pi}(s) = \sum_{s' \in \tilde{S}} \hat{P}_{s'|s,a}(s'|s,\pi(s)) \left(R(s'|s,\pi(s)) + \gamma^{\Delta t^{\pi}} \tilde{V}^{\pi}(s') \right) \tag{1}$$

Note that Δt is part of $\pi(s)$. And for the optimal policy π^*,

$$\tilde{V}^{\pi^*}(s) = \underset{a=(u,\Delta t) \in A}{\arg\max} \sum_{s' \in \tilde{S}} \hat{P}_{s'|s,a}(s'|s,a) \left(R(s'|s,a) + \gamma^{\Delta t} \tilde{V}^{\pi^*}(s') \right) \tag{2}$$

The $\arg\max$ in Eq. (2) is not a trivial optimization problem since the dimension of A may be high, and it may be computationally expensive to obtain $\hat{P}_{s'|s,a}(\cdot|\cdot,\cdot)$. We use batch Bayesian optimization with caching to efficiently approximate this problem.

Beyond the fact that iMDP assumes a known model for the transition dynamics and we do not, there are two major algorithmic differences between our method and iMDP:

- iMDP computes the maximum over actions via uniformly random sampling, assuming that the Bellman equation can be accurately solved in the limit of infinitely many action samples; in our case, we are concerned about practical performance, and so perform a more utility-driven action-sampling strategy using Bayesian optimization.
- iMDP uses value iteration while our method use RTDP. We make this choice because the model can be expensive to compute, and we would like to compute as few models for state-action pairs as possible.

Other differences between our method and iMDP include the type of RRT for state sampling (we use RRT with forward extension to handle non-holonomic systems and because our learning method estimates a forward model, while iMDP uses backward extension), and the domain of interest for the stochastic models and planning problem (single-modal transitions, multiple starting states and single goal for iMDP and multi-modal transitions, multiple start-goal pairs for our method).

4 BOIDP

We describe our algorithm Bayesian Optimization Incremental-realtime Dynamic Programming (BOIDP) in this section. The algorithm takes as input: a data-set of previous experience characterizing the effects of its actions in free space, a map of the permanent obstacles in the domain, and a goal region in state space. Given a starting state of the system s_0, BOIDP first computes and then executes a closed-loop policy for the robot.

In a typical model-based learning approach, first a monolithic model is estimated from the data and then that model is used to construct a policy. Here, however, we aim to scale to large, complex spaces with non-Gaussian dynamics, a setting where it is very difficult to represent and estimate a single monolithic model. Hence, we take a different approach: we estimate local models on demand, as the planning process requires information about relevant states and actions. At the highest level, the algorithm proceeds as follows:

- A discrete set of states is sampled using an RRT-like process;
- A continuous-action, discrete-state MDP, defined on that state set, is solved, yielding a control policy for the robot.

The models are computed on demand during the solution of the MDP. To truly take advantage of incremental model estimation, we work to focus the MDP solver on relevant parts of the state and action spaces. We assume that the initial state is known, so it is only necessary to derive a policy that with high probability will cover the states reached by the robot during execution. Using the RTDP solution strategy (Barto et al., 1995) rather than value iteration, we focus attention on states likely to be encountered given the starting state. Each dynamic-programming update in RTDP requires a maximization over the action space; we cannot achieve this analytically and so must sample a finite set of possible actions. We use a recent Bayesian optimization technique (Wang et al., 2016) to focus action sampling on regions of the action space that are informative and/or likely to be high-value.

Both the state sampling and transition-distribution estimation processes assume a collision checker $\textsc{ExistsCollision}(s, a, s')$ that checks the path from s to s' induced by action a for collisions with permanent objects in the map.

4.1 Estimating local transition models

We assume that the basic underlying dynamics in free space with no obstacles are location invariant; that is, that the change in state Δs resulting from taking action a is independent of the state s in which a was executed. We are given a training dataset

$D = \{\Delta s_i, a_i\}_{i=0}^{N}$, where a_i is an action and Δs_i is the resulting state change, collected in the free space.

Now, given a new query for action a, we predict the distribution of Δs by looking at the subset $D' = \{\Delta s_j, a_j\}_{j=0}^{M} \subseteq D$ whose actions a_j are the most similar to a (in our experiments we use 1-norm distance to measure similarity), and fit a Gaussian mixture model on Δs_j using the EM algorithm, yielding an estimated continuous state-action transition model

$$p_{s'|s,a}(s + \Delta_s \mid s, a) = p_{\Delta s|a}(\Delta s \mid a) \ .$$

We use the Bayesian information criterion (BIC) to determine the number of mixture components.

Finally, given a discrete set of states $\tilde{S}$, starting state s and action a, we can compute the approximate discrete transition model $\hat{P}_{s'|s,a}$ as shown in Algorithm 1. We use the function HIGHPROBNEXTSTATES to select the set of states $\hat{S}$ that are the $\Theta(\log|\tilde{S}|)$ most probable next states, following iMDP (Huynh et al., 2016). Because $p_{s'|s,a}$ does not take obstacles into account, we have to check the path from state s to next state $s' \in \tilde{S}$ induced by action a for collisions, and model their effect in the approximate discrete transition model $\hat{P}_{s'|s,a}$. To achieve this, we add a dummy terminal state s_{obs}, which represents a collision, to the selected next-state set $\hat{S}$. Then, for any s, a, s' transition that generates a collision, we move the probability mass $\hat{P}_{s'|s,a}(s' \mid s, a)$ to the transition to the collision state $\hat{P}_{s'|s,a}(s_{obs} \mid s, a)$. Finally, $\hat{P}_{s'|s,a}(\hat{S} \mid s, a)$ is normalized and returned together with the selected set $\hat{S}$.

These learned discrete transition models can be indexed by state s and action a and cached for future use in tasks that use the same set of states $\tilde{S}$ and the same obstacle map. In addition, the learned Gaussian mixture models $p_{\Delta s|a}$ can be cached for future use in any task with the same free-space dynamics. The memory-based essence of our modeling strategy is similar to the strategy of non-parametric models such as Gaussian processes, which make predictions for new inputs via smoothness assumptions and similarity between the query point and training points in the data set.

Although we have presented one learning strategy here, a variety of other methods that can estimate a continuous conditional density, such as kernel density estimators, would also be applicable.

Algorithm 1 Transition model for discrete states

function TRANSITIONMODEL($s, a, \tilde{S}, p_{s'|s,a}$)
  $\hat{S} \leftarrow$ HIGHPROBNEXTSTATES($p_{s'|s,a}(\tilde{S} \mid s, a)) \cup \{s_{obs}\}$ ▷ s_{obs} is a terminal state
  $\hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow p_{s'|s,a}(\hat{S} \mid s, a)$
  for s' **in** $\hat{S}$ **do**
    if $s' \in S^o$ **and** EXISTSCOLLISION(s, a, s') **then**
      $\hat{P}_{s'|s,a}(s_{obs} \mid s, a) \leftarrow \hat{P}_{s'|s,a}(s_{obs} \mid s, a) + \hat{P}_{s'|s,a}(s' \mid s, a)$
      $\hat{P}_{s'|s,a}(s' \mid s, a) \leftarrow 0$
  $\hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow$ NORMALIZE($\hat{P}_{s'|s,a}(\hat{S} \mid s, a)$)
  return $\hat{S}, \hat{P}_{s'|s,a}(\hat{S} \mid s, a)$

Algorithm 2 BOIDP

1: **function** BOIDP($s_0, \mathcal{G}, S, p_{s'|s,a}, N_{\min}$)
2: $\tilde{S} \leftarrow \{s_0\}$
3: **loop**
4: $\tilde{S} \leftarrow$ SAMPLESTATES($N_{\min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
5: $\pi, V =$ RTDP($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
6: **until** CALLBACK(π, V)
7: **return** π, V

8: **function** CALLBACK(π, V)
▷ User-defined function to judge if π, V are good enough to return.

4.2 Planning in BOIDP

At the highest level, BOIDP operates in a loop, in which it samples a discrete set of states and attempts to solve the discrete-state, continuous-action MDP induced by that sampled state set. The input to BOIDP includes the starting state s_0, the goal region $\mathcal{G}$, the continuous state space S, the function handle that computes the next state density $p_{s'|s,a}$, and the minimum number of states, $N_{\min}$, to sample at each iteration. It may be that more than $N_{\min}$ states are sampled, because sampling must continue until at least one goal state is included in the resulting set $\tilde{S}$. If the value of the resulting policy computed from RTDP is satisfactory, it is returned, otherwise, additional state samples are added and the process is repeated. The user can define the function CALLBACK to specify the desired termination condition on the resulting policy.

Sampling states In order to generate a discrete state set for the MDP, we sample states both in the interior of S^o and on its boundary ∂S. Terminal states are the states in the goal region $\mathcal{G}$. We assume there exists a distance metric d that can measure the closeness between states in S.

To generate one interior state sample, we randomly generate a state s_{rand}, and find $s_{nearest}$ that is the nearest state to s_{rand} in the current sampled state set $\tilde{S}$. Then we uniformly randomly sample a set of actions from A, and choose the action a that gives us the $s_n \sim p_{s'|s,a}(s'|s_{nearest}, a)$ that is the closest to s_{rand}. To sample states on the boundary ∂S, we assume a uniform random generator for states on ∂S is available. If not, we can use something similar to SAMPLEINTERIORSTATES but only sample inside the obstacles uniformly in line 7 of Algorithm 3. Once we have a sample s_{rand} in the obstacle, we try to reach s_{rand} by moving along the path $s_{rand} \rightarrow s_n$ incrementally until a collision is reached.

IMDP suggested that the ratio of the number of samples in S^o and ∂S be $1:1$. However, we have found in practice that it is beneficial to have more states sampled in ∂S in the beginning, in order to provide the planner with a good representation of the parts of the space in which collisions can occur, so that they can be avoided. Algorithm 3 describes the state sampling procedures. Notice that we can always add more states by calling SAMPLESTATES.

Focusing on the relevant states via RTDP Our strategy will be to apply our algorithm with a known starting state s_0 and goal region $\mathcal{G}$. Hence, it is not necessary to compute a complete policy, and so we use Real Time Dynamic Programming (RTDP) (Barto et al.,

Algorithm 3 RRT states sampling for BOIDP

function SAMPLESTATES($N_{min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  $\tilde{S}^o \leftarrow$ SAMPLEINTERIORSTATES($\lceil N_{min}/2 \rceil, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  $\partial\tilde{S} \leftarrow$ SAMPLEBOUNDARYSTATES($\lceil N_{min}/2 \rceil, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  return $\tilde{S}^o \cup \partial\tilde{S}$

function SAMPLEINTERIORSTATES($N_{min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
  while $|\tilde{S}| < N_{min}$ or $\mathcal{G} \cap \tilde{S} = \emptyset$ **do**
    $s_{rand} \leftarrow$ UNIFORMSAMPLE(S)
    $s_{nearest} \leftarrow$ NEAREST($s_{rand}, \tilde{S}$)
    $s_n, a_n \leftarrow$ RRTEXTEND($s_{nearest}, s_{rand}, p_{s'|s,a}$)
    if found s_n, a_n **then**
      $\tilde{S} \leftarrow \tilde{S} \cup \{s_n\}$
  return $\tilde{S}$

function RRTEXTEND($s_{nearest}, s_{rand}, p_{s'|s,a}$)
  $d_n = \infty$
  while stopping criterion not reached **do**
    $a \leftarrow$ UNIFORMSAMPLE(A)
    $s' \leftarrow$ SAMPLE $\left(p_{s'|s,a}(\cdot|s_{nearest}, a)\right)$
    if (**not** EXISTSCOLLISION(s, s', a)) **and** $d_n > d(s_{rand}, s')$ **then**
      $d_n \leftarrow d(s_{rand}, s')$
      $s_n, a_n \leftarrow s', a$
  return s_n, a_n

1995) to compute a value function on the relevant state space and a policy that, with high probability, will reach the goal before it reaches a state for which an action has not been determined. We assume an upper bound of the values for each state s to be $h_u(s)$. One can approximate $h_u(s)$ via the shortest distance from each state to the goal region on the fully connected graph with vertices $\tilde{S}$. We show the pseudocode in Algorithm 4. When doing the recursion (TRIALRECURSE), we can save additional computation when maximizing $Q(s, a)$. Assume that the last time maximize$_a Q(s, a)$ was called, the result was a^* and the transition model tells us that $\bar{S}$ is the set of possible next states. The next time we call maximize$_a Q(s, a)$, if the values for $\bar{S}$ has not changed, we can just return a^* as the result of the optimization. This can be done easily by caching the current (optimistic) policy and transition model for each state.

Focusing on good actions via Bayesian optimization Algorithm 4 relies on a challenging optimization over a continuous and possibly high-dimensional action space. Queries to $Q_s(a) = \sum_{s'} \hat{P}(s'|s, a) \left(R(s'|s, a) + \gamma^{\Delta t}\tilde{V}(s')\right)$ can be very expensive because in many cases a new model must be estimated. Hence, we need to limit the number of points queried during the optimization. There is no clear strategy for computing the gradient of $Q_s(a)$, and random sampling is very sample-inefficient especially as the dimensionality of the space grows. We will view the optimization of $Q_s(a)$ as a black-box function optimization problem, and use batch Bayesian optimization to efficiently approximate the solution.

We first briefly review a sequential Gaussian-process optimization method, GP-EST (Wang et al., 2016), shown in Algorithm 5. For a fixed state s, we assume $Q_s(a)$ is a sample from a Gaussian process with zero mean and kernel κ. At iteration t, we select action a_t and observe the function value $y_t = Q_s(a_t) + \epsilon_t$, where ϵ_t is i.i.d.

Algorithm 4 RTDP for BOIDP

function RTDP($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  for s **in** $\tilde{S}$ **do**
    $V(s) = h_u(s)$ ▷ Compute the upper bound of the value for s
  while $V(\cdot)$ not converged **do**
    $\pi, V \leftarrow$ TRIALRECURSE($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  return π, V

function TRIALRECURSE($s, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  if reached cycle **or** $s \in \mathcal{G}$ **then**
    return
  $\pi(s) \leftarrow \arg\max_a Q(s, a, \tilde{S}, p_{s'|s,a})$ ▷ Max computed via Bayesian optimization
  $s' \leftarrow$ SAMPLE($\hat{P}_{s'|s,a}(\tilde{S} \mid s, \pi(s))$)
  TRIALRECURSE($s', \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
  $\pi(s) \leftarrow \arg\max_a Q(s, a, \tilde{S}, p_{s'|s,a})$ ▷ Max computed via Bayesian optimization
  $V(s) \leftarrow Q(s, \pi(s), \tilde{S}, p_{s'|s,a})$
  return π, V

function Q($s, a, \tilde{S}, p_{s'|s,a}$)
  if $\hat{P}_{s'|s,a}(\tilde{S} \mid s, a)$ has not been computed **then**
    $\hat{P}_{s'|s,a}(\tilde{S} \mid s, a) = \mathbf{0}$ ▷ $\hat{P}_{s'|s,a}$ is a matrix shared across the process
    $\hat{S}, \hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow$ TRANSITIONMODEL($s, a, \tilde{S}, p_{s'|s,a}$)
  return $R(s, a) + \gamma^{\Delta t} \sum_{s' \in \hat{S}} \hat{P}_{s'|s,a}(s' \mid s, a) V(s')$

Gaussian computational noise $\mathcal{N}(0, \sigma^2)$. Given the observations $\mathfrak{D}_t = \{(a_\tau, y_\tau)\}_{\tau=1}^t$ up to time t, we obtain the posterior mean and covariance of the $Q_s(a)$ function via the kernel matrix $\boldsymbol{K}_t = [\kappa(a_i, a_j)]_{a_i, a_j \in \mathfrak{D}_t}$ and $\boldsymbol{\kappa}_t(a) = [\kappa(a_i, a)]_{a_i \in \mathfrak{D}_t}$ (Rasmussen and Williams, 2006): $\mu_t(a) = \boldsymbol{\kappa}_t(a)^{\mathrm{T}}(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}_t$, and $\kappa_t(a, a') = \kappa(a, a') - \boldsymbol{\kappa}_t(a)^{\mathrm{T}}(\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{\kappa}_t(a')$. The posterior variance is given by $\sigma_t^2(a) = \kappa_t(a, a)$. We can then use the posterior mean function $\mu_t(\cdot)$ and the posterior variance function $\sigma_t^2(\cdot)$ to select which action to test in the next iteration. We here make use of the assumption that we have an upper bound $h_u(s)$ on the value $V(s)$. We select the action that is most likely to have a value greater than or equal to $h_u(s)$ to be the next one to evaluate.

Algorithm 5 is only able to deal with sequential tests of $Q_s(a)$, but given availability of parallel computation hardware, it may be much more effective to test $Q_s(a)$ for multiple values of a in parallel. This requires us to choose a diverse subset of actions that are expected to be informative and/or have good values. We here propose a new batch Bayesian optimization method that selects a query set that has large diversity and low values of the acquisition function $G_{s,t}(a) = \left(\frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)}\right)$. The key idea is to maximize a submodular objective function with a cardinality constraint on $B \subset$

Algorithm 5 Optimization of $Q_s(a)$ via sequential GP optimization

$\mathfrak{D}_0 \leftarrow \emptyset$
for $t = 1 \rightarrow T$ **do**
  $\mu_{t-1}, \sigma_{t-1} \leftarrow$ GP-predict($\mathfrak{D}_{t-1}$)
  $a_t \leftarrow \arg\min_{a \in A} \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)}$
  $y_t \leftarrow Q_s(a_t)$
  $\mathfrak{D}_t \leftarrow \mathfrak{D}_{t-1} \cup \{a_t, y_t\}$

Algorithm 6 Optimization of $Q_s(a)$ via batch GP optimization

$\mathfrak{D}_0 \leftarrow \emptyset$
for $t = 1 \rightarrow T$ **do**
  $\mu_{t-1}, \sigma_{t-1} \leftarrow$ GP-predict($\mathfrak{D}_{t-1}$)
  $B \leftarrow \emptyset$
  for $i = 1 \rightarrow M$ **do**
    $B \leftarrow B \cup \{\arg\max_{a \in A} F_s(B \cup \{a\}) - F_s(B)\}$
  $\boldsymbol{y}_B \leftarrow Q_s(B)$ ▷ Test Q_s in parallel.
  $\mathfrak{D}_t \leftarrow \mathfrak{D}_{t-1} \cup \{B, \boldsymbol{y}_B\}$

$A, |B| = M$ that captures these two properties:

$$F_s(B) = \log\det \boldsymbol{K}_B - \lambda \sum_{a \in B} \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{3}$$

where $\boldsymbol{K}_B = [\kappa(a_i, a_j)]_{a_i, a_j \in B}$ and λ is a trade-off parameter for diversity and quality. If λ is large, F_s will prefer actions with lower $G_{s,t(a)}$, which means a better chance of having high values. If λ is low, $\log\det \boldsymbol{K}_B$ will dominate F_s and a more diverse subset B is preferred. One can choose λ by cross-validation. We can optimize the heuristic function F_s via greedy optimization which yield a $1 - \frac{1}{e}$ approximation to the optimal solution. We describe the batch GP optimization in Algorithm 6.

The greedy optimization can be efficiently implemented using the following property of the determinant:

$$F_s(B \cup \{a\}) - F_s(B) = \log\det \boldsymbol{K}_{B \cup \{a\}} - \log\det \boldsymbol{K}_B - \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{4}$$

$$= \log(\kappa_a - \boldsymbol{\kappa}_{Ba}^{\mathrm{T}} \boldsymbol{K}_B^{-1} \boldsymbol{\kappa}_{Ba}) - \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{5}$$

where $\kappa_a = \kappa(a, a), \boldsymbol{\kappa}_{Ba} = [\kappa(a_i, a)]_{a_i \in B}$.

5 Theoretical analysis

In this section we bound the error of BOIDP with Bayesian optimization for optimizing $Q_s(\cdot)$, but using the standard synchronous value iteration rather than RTDP, for simplicity. An analogous analysis will extend our results to bounded RTDP (McMahan et al., 2005; Smith and Simmons, 2006). Without loss of generality, we assume $\min \Delta t = 1$. It has been shown that, the optimal value function $\tilde{V}^*$ computed by iMDP (Huynh et al., 2016) will become arbitrarily close to the optimal value function V^* of the original continuous state-action MDP as $|\tilde{S}|$ sampled by the RRT goes to infinity (Huynh et al., 2016), under the strict condition that the Bellman equation is perfectly solved. Based on iMDP, we further show in the following theorem that with finitely many actions selected by Bayesian optimization, the L_∞ norm of the difference between V^* and the value function $\hat{V}$ of the policy computed by BOIDP decreases sub-linearly in the number of actions selected for optimizing $Q_s(\cdot)$.

Theorem 1 (Error bound of synchronous value iteration (SVI) for BOIDP**).**
Let V^* be the optimal value function for the original continuous state-action MDP $(S, A, p_{s'|s,a}, R, \gamma)$. At iteration k of the SVI, we define $\hat{V}_k$ to be the value function for $(\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$ approximated by BOIDP *and assume that*

$$Q_{s,k}(a) = \sum_{s' \in \tilde{S}} \hat{P}_{s'|s,a}(s'|s,a) \left(R(s'|s,a) + \gamma^{\Delta t} \hat{V}_{k-1}(s') \right)$$

is a function sampled from a Gaussian process with known priors and and zero-mean i.i.d Gaussian noise with standard deviation σ for any state $s \in \tilde{S}$. If we allow Bayesian optimization for $Q_{s,k}(a)$ to sample T actions for each state and run SVI until converge with $\mathfrak{K}$ iterations, in expectation,

$$\lim_{|\tilde{S}| \to \infty} |\hat{V}_{\mathfrak{K}}(\cdot) - V^*(\cdot)|_\infty \leq \frac{\nu}{1-\gamma} \sqrt{\frac{2\eta_T}{T \log(1+\sigma^2)}},$$

where η_T is the maximum information gain of the selected actions (Srinivas et al., 2010, Theorem 5), $\nu = \max_{s,t,k} \min_{a \in A} G_{s,t,k}(a)$, and $G_{s,t,k}(\cdot)$ is the acquisition function in Wang et al. (2016, Theorem 3.1) for state $s \in \tilde{S}$, iteration $t = 1, 2, \cdots, T$ in Algorithm 5 or 6, and iteration $k = 1, 2, \cdots, \mathfrak{K}$ in the SVI.

We prove Thm. 2 in the appendix.

6 Experiments

We tested our approach in a very simple quasi-static problem, in which a robot pushes a circular object through a planar workspace with obstacles. We assume the action is a quasi-static push, represented by the robot's initial relative position x to the object, the direction of the push z, and the duration of the push Δt, which are illustrated in Fig. 1.

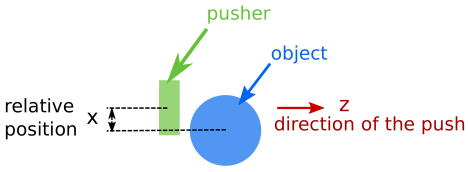

Fig. 1: We consider pushing a circular object with a rectangle pusher.

We used a pseudo-random Halton generator (Halton, 1960; LaValle, 2006) for state sampling in the RRT. To reduce the number of sampled states very close to obstacles, we randomly rejected 50% of states that collided with obstacles.

6.1 Importance of learning accurate models

Our method was specifically designed to be appropriate for use in systems whose dynamics are not well modeled with uni-modal Gaussian noise. The experiments in this section explore the question of whether a uni-modal model could work just as well,

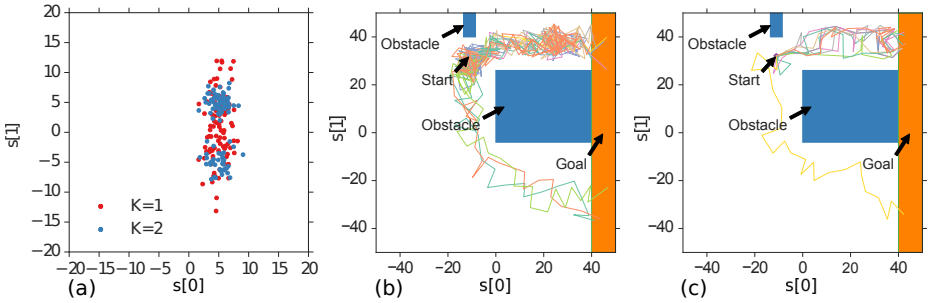

Fig. 2: (a) Samples from the single-mode Gaussian transition model ($K = 1$) and the two component mixture of Gaussians transition model ($K = 2$) when $a = 0$. (b) Samples of 10 trajectories for planning with $K = 1$. (c) Samples of 10 trajectories for planning with $K = 2$. Using the correct number of components for the transition model improves the planner's performance.

using a simple domain with known dynamics $s' = s + T(a)\rho$, where the relative position $x = 0$ and duration $\Delta t = 1$ are fixed, the action is the direction of motion, $a = z \in [0, 2\pi)$, $T(a)$ is the transformation matrix $\begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix}$, and the noise is

$$\rho \sim 0.6\mathcal{N}(\begin{bmatrix} 5.0 \\ 5.0 \end{bmatrix}, \begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}) + 0.4\mathcal{N}(\begin{bmatrix} 5.0 \\ -5.0 \end{bmatrix}, \begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}). \tag{6}$$

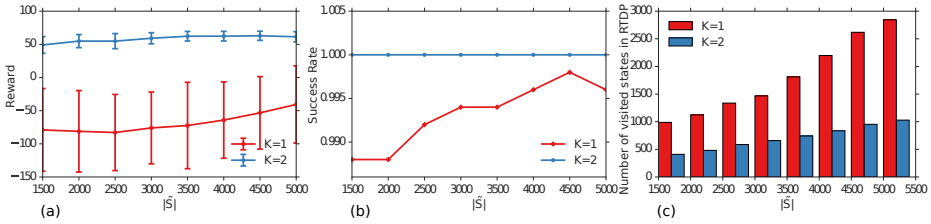

Fig. 3: Using two components ($K = 2$) performs much better than using one component ($K = 1$) and visits many fewer states in RTDP, because it matches the true model of the system. ((a): reward. (b): success rate. (c): number of visited states in RTDP.)

We sample ρ from its true distribution and fit a Gaussian ($K = 1$) and a mixture of Gaussians ($K = 2$). The samples from $K = 1$ and $K = 2$ are shown in Fig. 2(a). We plan with both models where each action has an instantaneous reward of -1, hitting an obstacle has a reward of -10, and the goal region has a reward of 100. The discount factor $\gamma = 0.99$. To show that the results are consistent, we use Algorithm 3 to sample 1500 to 5000 states to construct $\tilde{S}$, and plan with each of them using 100 uniformly discretized actions within 1000 iterations of RTDP. To compute the Monte Carlo reward, we simulated 500 trajectories for each computed policy with the true model dynamics,

and for each simulation, at most 500 steps are allowed. We show 10 samples of trajectories for both $K = 1$ and $K = 2$ with $|\tilde{S}| = 5000$, in Fig 2(b,c). Planning with the correct model tends to find better trajectories, while because $K = 1$ puts density on many states that the true model does not reach, the policy of $K = 1$ in (b) causes the robot to do extra maneuvers or even choose a longer trajectory to avoid obstacles that it actually has very low probability of hitting. As a result, the reward and success rate for $K = 2$ are both higher than $K = 1$, as shown in Fig. 3(a,b). Furthermore, because the single-mode Gaussian estimates the noise to have a large variance, it causes RTDP to visit more states than necessary, as shown in Fig. 3(c).

6.2 Focusing on the good actions

We denote using Bayesian optimization in Lines 10 and 13 in Algorithm 4 as BO and using random as Rand. We first demonstrate why BO is better than random for optimizing $Q_s(a)$ with the simple example in Sec. 6.1. We plot the $Q_s(a)$ in the first iteration of RTDP where $s = [-4.3, 33.8]$, and let random and BO in Algorithm 5 each pick 10 actions to evaluate sequentially. We use the GP implementation and the default Matern52 kernel implemented in the GPy module (GPy, since 2012) and optimize its kernel parameters every 5 selections. The first point for both BO and Rand is fixed to be $a = 0.0$. We observe that BO is able to focus its action selections in the high value region, and BO is also able to explore informative actions if it has not found a good value or if it has finished exploiting a good region (see selection 10). Random action selection wastes choices on regions that have already been determined to be bad.

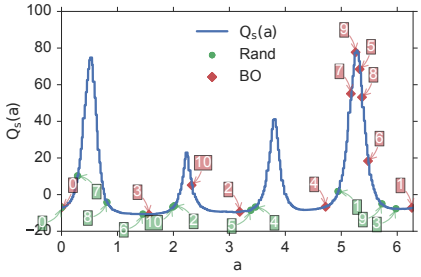

Fig. 4: BO selects actions more strategically than Rand.

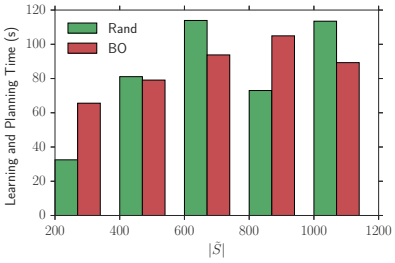

Fig. 5: Learning and planning time of BO and Rand.

Next we consider a more complicated problem in which the action is the high level control of a pushing problem $a = (z, x, \Delta t)$, $z \in [0, 2\pi), x \in (-1, 1), \Delta t \in (0, 3)$ as illustrated in Fig. 1. The instantaneous reward is -1 reward for each free-space motion, -10 for hitting an obstacle, and 100 for reaching the goal; $\gamma = 0.99$.

We collected 1.2×10^6 data points of the form $(a, \Delta s)$ with $z = 0$ in the Box2D simulator (Catto, 2011) where noise comes from variability of the executed action. We make use of the fact that the object is cylindrical (with radius 1.0), and reuse the data by

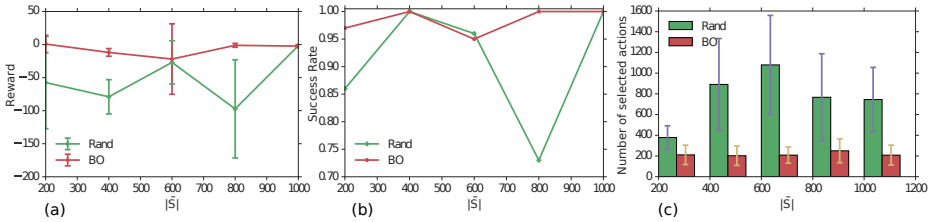

Fig. 6: (a) Reward. (b) Success rate. (c) Average number of actions selected per visited state. BO achieves better reward and success rate, with many fewer actions.

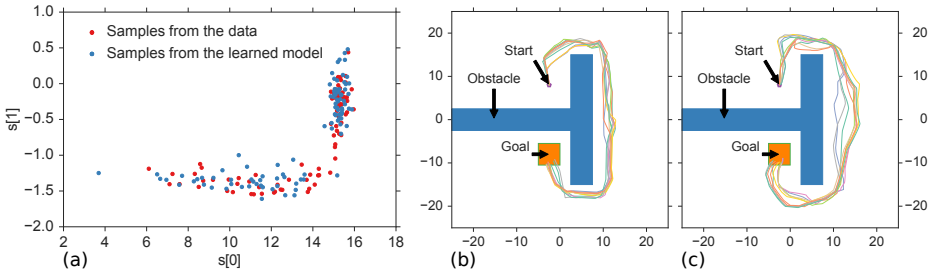

Fig. 7: (a) The conditional distribution of Δs given $a = (0.0, 0.3, 2.0)$ is a multi-modal Gaussian. (b) 10 samples of trajectories generated via Rand iwth 1000 states. (c) 10 samples of trajectories generated via BO with 1000 states.

applying the transformation matrix $T(z)$. An example of the distribution of Δs given $a = (0.0, 0.3, 2.0)$ is shown in Fig. 7(a). We compare policies found by Rand and BO with approximately the same amount of total computation time. They are both able to compute the policy in $30 \sim 120$ seconds, which is shown in Fig. 5. In more realistic domains, it is possible that the learning of the transition model will take longer and dominate the action-selection computation. We simulate 100 trajectories in the Box2D simulator for each planned policy with a maximum of 200 seconds. We show the result of the reward and success rate in Fig. 6(a,b), together with the average number of actions selected for visited states in Fig. 6(c). In our simulations, BO consistently performs approximately the same or better than Rand in terms of reward and success rate while BO selects fewer actions than Rand. We show 10 simulated trajectories for Rand and BO with $|\tilde{S}| = 1000$ in Fig. 7(b,c) and an animation of the trajectory resulting from a BO policy in the supplementary material.

7 Conclusion

An important class of robotics problems involves a vehicle or more complex system that has continuous unknown transition dynamics that are constant within free space, but are impacted by the presence of obstacles. We have provided a method for estimating those dynamics from data and using them to plan effective policies in complex domains with obstacles, and support multiple queries with different start and goal states, or even different obstacle maps. We achieve efficiency by focusing on relevant subsets of state and action spaces, but retain guarantees of asymptotic optimality.

References

R. Alterovitz, T. Siméon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and Systems III*, 200.

L. C. Baird, III and A. H. Klopf. Reinforcement learning with high-dimensional continuous actions. Technical report, Wright Laboratory, Wright Patterson Air Force Base, 1993. URL `http://leemon.com/papers/1993bk3.pdf`.

A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

E. Catto. Box2D, a 2D physics engine for games. `http://box2d.org`, 2011.

M. Deisenroth and C. 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)*, pages 465–472, 2011.

M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Model-based reinforcement learning with continuous states and actions. In *ESANN*, pages 19–24, 2008.

M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7):1508–1524, 2009.

M. Frean and P. Boyle. Using Gaussian processes to optimize expensive functions. In *Australasian Joint Conference on Artificial Intelligence*, pages 258–267. Springer, 2008.

M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 8(5–6):359–483, 2015.

GPy. GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

V. Gullapalli, J. A. Franklin, and H. Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems*, 14(1):13–24, 1994.

J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

V. A. Huynh, S. Karaman, and E. Frazzoli. An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research*, 35 (4):305–333, 2016.

H. S. Jakab and L. Csató. Reinforcement learning with guided policy search using Gaussian processes. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2012.

M. Kopicki. *Prediction learning in robotic manipulation*. PhD thesis, University of Birmingham, 2010.

M. Kopicki, J. Wyatt, and R. Stolkin. Prediction learning in robotic pushing manipulation. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6. IEEE, 2009.

M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, and J. Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5722–5729. IEEE, 2011.

S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd international conference on Machine learning*, pages 569–576. ACM, 2005.

T. M. Moldovan, S. Levine, M. I. Jordan, and P. Abbeel. Optimism-driven exploration for nonlinear systems. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3239–3246. IEEE, 2015.

D. Nguyen-Tuong and J. Peters. Model learning for robot control: A survey. *Cognitive processing*, 12(4):319–340, 2011.

B. D. Nichols. Continuous action-space reinforcement learning methods applied to the minimum-time swing-up of the acrobot. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2015.

D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007, 1997.

C. E. Rasmussen and C. K. Williams. Gaussian processes for machine learning. *The MIT Press*, 2006.

A. Rottmann and W. Burgard. Adaptive autonomous control using online value iteration with Gaussian processes. In *IEEE International Conference on Robotics and Automation*, pages 2106–2111. IEEE, 2009.

T. Smith and R. Simmons. Focused real-time dynamic programming for mdps: Squeezing more out of a heuristic. In *AAAI*, pages 1227–1232, 2006.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

M. K. Titsias and M. Lázaro-Gredilla. Variational heteroscedastic Gaussian process regression. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 841–848, 2011.

M. Turchetta, F. Berkenkamp, and A. Krause. Safe exploration in finite Markov decision processes with Gaussian processes. *arXiv preprint arXiv:1606.04753*, 2016.

H. van Hasselt and M. A. Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

Z. Wang, B. Zhou, and S. Jegelka. Optimization as estimation with Gaussian processes in bandit settings. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016. arXiv preprint arXiv:1604.04038.

C. Yuan and C. Neubauer. Variational mixture of Gaussian process experts. In *Advances in Neural Information Processing Systems*, pages 1897–1904, 2009.

A Proof of Thm. 1

Theorem 2 (Error bound of synchronous value iteration (SVI) for BOIDP).

Let V^* be the optimal value function for the original continuous state-action MDP $(S, A, p_{s'|s,a}, R, \gamma)$. At iteration k of the SVI, we define $\hat{V}_k$ to be the value function for $(\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$ approximated by BOIDP and assume that

$$Q_{s,k}(a) = \sum_{s' \in \tilde{S}} \hat{P}_{s'|s,a}(s'|s,a) \left(R(s'|s,a) + \gamma^{\Delta t} \hat{V}_{k-1}(s') \right)$$

is a function sampled from a Gaussian process with known priors and and zero-mean i.i.d Gaussian noise with standard deviation σ for any state $s \in \tilde{S}$. If we allow Bayesian optimization for $Q_{s,k}(a)$ to sample T actions for each state and run SVI until converge with $\mathfrak{K}$ iterations, in expectation,

$$\lim_{|\tilde{S}| \to \infty} |\hat{V}_{\mathfrak{K}}(\cdot) - V^*(\cdot)|_\infty \leq \frac{\nu}{1-\gamma} \sqrt{\frac{2\eta_T}{T \log(1+\sigma^2)}},$$

where η_T is the maximum information gain of the selected actions (Srinivas et al., 2010, Theorem 5), $\nu = \max_{s,t,k} \min_{a \in A} G_{s,t,k}(a)$, and $G_{s,t,k}(\cdot)$ is the acquisition function in Wang et al. (2016, Theorem 3.1) for state $s \in \tilde{S}$, iteration $t = 1, 2, \cdots, T$ in Algorithm 5 or 6, and iteration $k = 1, 2, \cdots, \mathfrak{K}$ in the SVI.

Proof. Our proof build on the proof of Huynh et al. (2016, Theorem 8), which bound the error of the value function $\tilde{V}^*$ computed by iMDP and the optimal value function V^*. We will first show that $|\tilde{V}^*_{\mathfrak{K}} - \hat{V}_{\mathfrak{K}}|$ is bounded via standard BO results (Wang et al., 2016, Theorem 3.1) and then combine with Huynh et al. (2016, Theorem 8) to show that $|V^* - \hat{V}_{\mathfrak{K}}|$ is also bounded.

Assume the optimal value functions from iterations 1 to $\mathfrak{K}$ in the SVI for the discrete state continuous action MDP $(\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$ are $\{\tilde{V}^*_1, \cdots, \tilde{V}^*_{\mathfrak{K}}\}$ respectively, and the value functions computed by BOIDP are $\{\hat{V}_1, \cdots, \hat{V}_{\mathfrak{K}}\}$. Let ϵ_k be the L_∞ norm between $\hat{V}_k$ and $\tilde{V}^*_k = \max_{a \in A} Q_{s,k}(a)$ at iteration k. According to Wang et al. (2016, Theorem 3.1), in expectation, for any fixed iteration k, the following inequality holds:

$$\epsilon_k \leq \nu_k \sqrt{\frac{2\eta_T}{T \log(1+\sigma^2)}},$$

where

$$\nu_k = \max_{s \in S, t \in [1,T]} \min_{a \in A} G_{s,t,k}(a),$$

and

$$G_{s,t,\cdot}(a) = \frac{h_u(s) - \mu_{t-1}(a)}{\sigma_{t-1}(a)}$$

is the acquisition function in Wang et al. (2016, Theorem 3.1), which makes use of the assumed upper bound $h_u(\cdot)$ on the value function. Let $\nu = \max_{k \in [1,\mathfrak{K}]} \nu_k$, we have $\forall k$,

$$\epsilon_k \leq \nu \sqrt{\frac{2\eta_T}{T \log(1+\sigma^2)}} = \epsilon,$$

So our optimization introduces error of at most ϵ to the optimization of the Bellman equation in every iteration of the SVI. For approximate synchronous value iteration, the following inequalities hold:

$$
\begin{aligned}
&|\hat{V}_0 - \tilde{V}_0^*|_\infty \leq \epsilon, \\
&|\hat{V}_1 - \tilde{V}_1^*|_\infty \leq \epsilon + \gamma\epsilon, \\
&\qquad \cdots, \\
&|\hat{V}_\mathfrak{K} - \tilde{V}_\mathfrak{K}^*|_\infty \leq \sum_{k=0}^{\mathfrak{K}} \gamma^k \epsilon < \frac{\epsilon}{1-\gamma} = \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}, \quad \forall \mathfrak{K} = 0, \cdots, \infty.
\end{aligned}
$$

Hence, for the iteration $\mathfrak{K}$ that the SVI converge for BOIDP, we have

$$
|\hat{V}_\mathfrak{K} - \tilde{V}^*|_\infty \leq \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}
$$

Next we use the bound on $|\tilde{V}^* - V^*|$ (Huynh et al., 2016, Theorem 8) to bound $|\hat{V}_\mathfrak{K} - V^*|$. By Huynh et al. (2016, Theorem 8), with probability 1,

$$
\lim_{|\tilde{S}|\to\infty} |\tilde{V}^* - V^*| = 0
$$

By the triangle inequality of L_∞, we have

$$
\begin{aligned}
\lim_{|\tilde{S}|\to\infty} |\hat{V}_\mathfrak{K} - V^*| &\leq \lim_{|\tilde{S}|\to\infty} |\hat{V}_\mathfrak{K} - \tilde{V}^*| + \lim_{|\tilde{S}|\to\infty} |\tilde{V}^* - V^*| \\
&\leq \frac{\nu}{1-\gamma}\sqrt{\frac{2\eta_T}{T\log(1+\sigma^2)}}
\end{aligned}
$$

holds in expectation. □