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

Zi Wang Stefanie Jegelka Leslie Pack Kaelbling Tomás Lozano-Pérez

***Abstract*— We introduce a framework for model learning and planning in stochastic domains with continuous state and action spaces and non-Gaussian transition models. It is efficient 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 the validity and asymptotic optimality of the proposed approach. Empirically, we demonstrate the effectiveness of our algorithm on a simulated multi-modal pushing problem.**

I. 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 frequently appears in basic manipulation, especially when it is non-prehensile [1]. 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, air resistance, 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 rolling dice or pushing a can as shown in Fig. 1, the stochasticity is not modeled well by additive single-mode Gaussian noise, and a more sophisticated model class is necessary.

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. Our framework learns transition models that can 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 which could be difficult to represent, 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, the planner focuses only on states relevant to the current planning problem, and actions that can lead to high reward.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA 02139. {ziw,stefje,lpk,tlp}@csail.mit.edu. We gratefully acknowledge support from NSF CAREER award 1553284, NSF grants 1420927 and 1523767, from ONR grant N00014-14-1-0486, and from ARO grant W911NF1410433. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

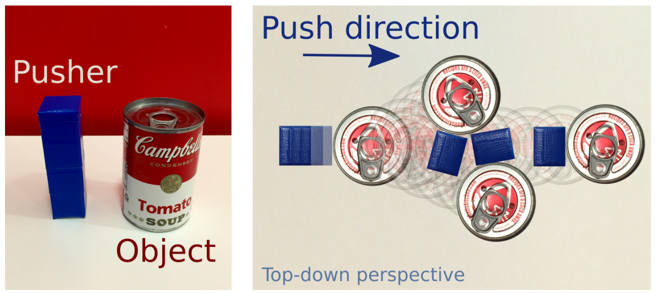

Fig. 1: A quasi-static pushing problem: the pusher has a velocity controller with low gain, resulting in non-Gaussian transitions. We show trajectories for object and pusher resulting from the same push velocity.

We propose a closed-loop planning algorithm that applies to stochastic continuous state-action systems with arbitrary transition models. It is assumed that the transition models are represented by a function that may be expensive to evaluate. Via two important steps, we focus the 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) [2] on a set of states strategically sampled by a rapidly-exploring random tree (RRT) [3], [4]. To focus selection of actions from a continuous space, we develop a new batch Bayesian optimization (BO) technique that selects and tests, in parallel, action candidates that will lead most quickly to a near-optimal answer.

We show theoretically that the expected accumulated 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. Finally, we evaluate our approach empirically on a simulated multi-modal pushing problem, and demonstrate the effectiveness and efficiency of the proposed algorithm.

II. Related Work

a) Learning: The class of problems that we address may be viewed as reinforcement-learning (RL) problems in observable continuous state-action spaces. It is possible to address the problem through model-free RL, which estimates a value function or policy for a specific goal directly through experience. Though 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 [5], [6], [7], [8], [9]. An alternative approach is to do direct search in the space of policies [10], [11].

In continuous state-action spaces, model-based RL, where a model is estimated to optimize a policy, can often be more effective. Gaussian processes (GP) can help to learn the dynamics [12], [13], [14], which can then be used by GP-based dynamic programming [15], [13] to determine a continuous-valued closed-loop policy for the whole state space. More details can be found in the excellent survey [16].

Unfortunately, the common assumption of i.i.d Gaussian noise on the dynamics is restrictive and may not hold in practice [1], and the transition model can be multi-modal. It may additionally be difficult to obtain a good GP prior. The basic GP model can capture neither the multi-modality nor the heteroscedasticity of the noise. While more advanced GP algorithms may address these problems, they often suffer from high computational cost [17], [18].

Moldovan et al. [19] addressed the problem of multi-modality by using Dirichlet process mixture models (DP-MMs) to learn the density of the transition models. Their strategies for planning were limited by deterministic assumptions, appropriate for their domains of application, but potentially resulting in collisions in ours. Kopicki et al. [20], [21], [22] 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 arbitrary, especially multi-modal, stochastic models in continuous state-action spaces. Our learning method in the experiment 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.

b) Planning: We are interested in domains for which queries are made by specifying a starting state and a goal set, and in which the solution to the given query can be described by a policy that covers only a small fraction of the state space that the robot is likely to encounter.

Planning only in the fraction of the state-action space that the robot is likely to encounter is, in general, very challenging [23]. The iMDP method [4], which is most related to our work, 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 method assumes the ability to solve the Bellman equation optimally (e.g. for a simple stochastic LQR problem), the availability of the backward transition models, and that the dynamics is modeled by a Wiener process, in which the transition noise is Gaussian with execution-time-dependent variance. However, the assumptions are too restrictive to model our domains of interest where the dynamics is non-closed-form, costly to evaluate, non-reversible, and non-Gaussian. Furthermore, iMDP is designed for stochastic control problems with multiple starting states and a single goal, while we are interested in multiple start-goal pairs.

Our work builds on the idea of constructing a sequence of MDPs from iMDP [4], and aims at practically resolving the challenges of state/action selection.

c) Bayesian optimization: There have been a number of applications of BO in optimal control, although to our knowledge, it has not been previously applied to action-selection in continuous-action MDPs. BO has been used to find weights in a neural network controller [24], to solve for the parameters of a hierarchical MDP [25], and to address safe exploration in finite MDPs [26].

III. Problem formulation

Let the state space $S \subset \mathbb{R}^{d_s}$ with metric d and the control space $U \subset \mathbb{R}^{d_u}$ both be compact and measurable sets. The interior of the state space S is S^o and the boundary is ∂S. For the control space U, there exists an open set U^o in R^{d_u} such that U is the closure of U^o. 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 $A = U \times [T_{min}, T_{max}]$, where $T_{min}, T_{max} \in \mathbb{R}_+ \setminus \{\infty\}$ are the minimum and the maximum amount of duration allowed. The action space A is also a compact set. The starting state is s_0, and the goal region is $\mathcal{G} \subset S$, in which all states are terminal states. We assume $\mathcal{G}$ has non-zero measure, and S has finite measure. The *transition model* has the form of a continuous probability density function $p_{s'|s,a}$ on the resulting state s', given previous state s and action a, such that $\forall s' \in S, p_{s'|s,a}(s' \mid s, a) \geq 0, \int_S p(s' \mid s, a)\,\mathrm{d}s' = 1$.

Given a transition model and a cost function $C : S \times S \times A \to \mathbb{R}$, we can formulate the problem as a continuous state-action MDP $(S, A, p_{s'|s,a}, R, \gamma)$, where $R(s' \mid s, a) = -C(s' \mid s, a)$ is the immediate reward function and γ is the discount factor. A high reward is assigned to the states in the goal region $\mathcal{G}$, and a cost is assigned to colliding with obstacles or taking any action. We would like to solve for the optimal policy $\pi : S \to A$, for which the value of each state s is

$$V^\pi(s) = \max_{a \in A} \int_{s' \in S} p_{s'|s,a}(s'|s,a) \left(R(s'|s,a) + \gamma^{\Delta t} V^\pi(s') \right) \mathrm{d}s'.$$

IV. Our method: BOIDP

We describe our algorithm Bayesian Optimization Incremental-realtime Dynamic Programming (BOIDP) in this section. At the highest level, BOIDP in Alg. 1 operates in a loop, in which it samples a discrete set of states $\tilde{S} \subset S$ and attempts to solve the discrete-state, continuous-action MDP $\tilde{\mathcal{M}} = (\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$. Here $\hat{P}_{s'|s,a}(s' \mid s, a)$ is the probability mass function for the transition from state $s \in \tilde{S}$ using action $a \in A$ to a new state $s' \in \tilde{S}$. The value function for the optimal policy of the approximated MDP $\tilde{\mathcal{M}}$ is $V(s) = \max_{a \in A} Q_s(a)$, where

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

Algorithm 1 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 stopping criteria reached
7:     EXECUTEPOLICY($\pi, \tilde{S}, \mathcal{G}$)

8: function EXECUTEPOLICY($\pi, \tilde{S}, \mathcal{G}$)
9:     loop
10:        $s_c \leftarrow$ current state
11:        $\tilde{s} \leftarrow \arg\min_{s \in \tilde{S}} d(s, s_c)$
12:        Execute $\pi(\tilde{s})$
13:    until current state is in $\mathcal{G}$
```

If the value of the resulting policy is satisfactory according to the task-related stopping criterion[1], we can proceed; otherwise, additional state samples are added and the process is repeated. Once we have a policy π on $\tilde{S}$ from RTDP, the robot can iteratively obtain and execute the policy for the nearest state to the current state in the sampled set $\tilde{S}$ by the metric d.

There are a number of challenges underlying each step of BOIDP. First, we need to find a way of accessing the transition probability density function $p_{s'|s,a}$, which is critical for the approximation of $\hat{P}_{s'|s,a}(s' \mid s, a)$ and the value function. We describe our "lazy access" strategy in Sec. IV-A. Second, we must find a way to compute the values of as few states as possible to fully exploit the "lazy access" to the transition model. Our solution is to first use an RRT-like process [3], [4] to generate the set of states that asymptotically cover the state space with low dispersion (Sec. IV-B), and then "prune" the irrelevant states via RTDP [2] (Sec. IV-C). Last, 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 develop a new batch BO algorithm to focus action sampling on regions of the action space that are informative and/or likely to be high-value, as described in Sec. IV-D.

Both the state sampling and transition estimation processes assume a collision checker EXISTSCOLLISION(s, a, s') that checks the path from s to s' induced by action a for collisions with permanent objects in the map.

A. *Estimating transition models in* BOIDP

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 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 via "lazy access" to the model: we estimate local models on demand, as the planning process requires information about relevant states and actions.

[1]For example, one stopping criterion could be the convergence of the starting state's value $V(s_0)$.

Algorithm 2 Transition model for discrete states

```
1: function TRANSITIONMODEL($s, a, \tilde{S}, p_{s'|s,a}$)
2:     $\hat{S} \leftarrow$ HIGHPROBNEXTSTATES($p_{s'|s,a}(\tilde{S} \mid s, a)$) $\cup \{s_{obs}\}$
       ▷ $s_{obs}$ is a terminal state
3:     $\hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow p_{s'|s,a}(\hat{S} \mid s, a)$
4:     for $s'$ in $\hat{S}$ do
5:         if $s' \in S^o$ and EXISTSCOLLISION($s, a, s'$) then
6:             $\hat{P}_{s'|s,a}(s_{obs} \mid s, a) \leftarrow \hat{P}_{s'|s,a}(s_{obs} \mid s, a) + \hat{P}_{s'|s,a}(s' \mid s, a)$
7:             $\hat{P}_{s'|s,a}(s' \mid s, a) \leftarrow 0$
8:     $\hat{P}_{s'|s,a}(\hat{S} \mid s, a) \leftarrow$ NORMALIZE($\hat{P}_{s'|s,a}(\hat{S} \mid s, a)$)
9:     return $\hat{S}, \hat{P}_{s'|s,a}(\hat{S} \mid s, a)$
```

We assume a dataset $D = \{s_i, a_i, s'_i\}_{i=0}^N$ for the system dynamics and the dataset is large enough to provide a good approximation to the probability density of the next state given any state-action pair. If a stochastic simulator exists for the transition model, one may collect the dataset dynamically in response to queries from BOIDP. The "lazy access" provides a flexible interface, which can accommodate a variety of different density-estimation algorithms with asymptotic theoretical guarantees, such as kernel density estimators [27] and Gaussian mixture models [28]. In our experiments, we focus on learning Gaussian mixture models with the assumption that $p_{s'|s,a}(s' \mid s, a)$ is distributed according to a mixture of Gaussians $\forall (s, a) \in S \times A$.

Given a discrete set of states $\tilde{S}$, starting state s and action a, we compute the approximate discrete transition model $\hat{P}_{s'|s,a}$ as shown in Algorithm 2. We use the function HIGHPROBNEXTSTATES to select the largest set of next states $\hat{S} \subseteq \tilde{S}$ such that $\forall s' \in \hat{S}, p_{s'|s,a}(s' \mid s, a) > \epsilon$. The parameter ϵ is a small threshold, e.g. we can set $\epsilon = 10^{-5}$. If $p_{s'|s,a}$ 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 approximated 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. 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.

For the case where the dynamics model $p_{s'|s,a}$ is given, computing the approximated transition $\hat{P}_{s'|s,a}$ could still be computationally expensive because of the collision checking. Our planner is designed to alleviate the high computation

Algorithm 3 RRT states sampling for BOIDP

```
 1: function SampleStates($N_{min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
 2:     $\tilde{S}^o \leftarrow$ SampleInteriorStates($\lceil N_{min}/2 \rceil, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
 3:     $\partial\tilde{S} \leftarrow$ SampleBoundaryStates($\lceil N_{min}/2 \rceil, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
 4:     return $\tilde{S}^o \cup \partial\tilde{S}$

 5: function SampleInteriorStates($N_{min}, \tilde{S}, \mathcal{G}, S, p_{s'|s,a}$)
 6:     while $|\tilde{S}| < N_{min}$ or $\mathcal{G} \cap \tilde{S} = \emptyset$ do
 7:         $s_{rand} \leftarrow$ UniformSample($S$)
 8:         $s_{nearest} \leftarrow$ Nearest($s_{rand}, \tilde{S}$)
 9:         $s_n, a_n \leftarrow$ RRTExtend($s_{nearest}, s_{rand}, p_{s'|s,a}$)
10:         if found $s_n, a_n$ then
11:             $\tilde{S} \leftarrow \tilde{S} \cup \{s_n\}$
12:     return $\tilde{S}$

13: function RRTExtend($s_{nearest}, s_{rand}, p_{s'|s,a}$)
14:     $d_n = \infty$
15:     while stopping criterion not reached do
16:         $a \leftarrow$ UniformSample($A$)
17:         $s' \leftarrow$ Sample $(p_{s'|s,a}(\cdot \mid s_{nearest}, a))$
18:         if (not ExistsCollision($s, s', a$)) and $d_n > d(s_{rand}, s')$ then
19:             $d_n \leftarrow d(s_{rand}, s')$
20:             $s_n, a_n \leftarrow s', a$
21:     return $s_n, a_n$
```

Algorithm 4 RTDP for BOIDP

```
 1: function RTDP($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
 2:     for $s$ in $\tilde{S}$ do
 3:         $V(s) = h_u(s)$              ▷ Compute the value upper bound
 4:     while $V(\cdot)$ not converged do
 5:         $\pi, V \leftarrow$ TrialRecurse($s_0, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
 6:     return $\pi, V$

 7: function TrialRecurse($s, \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
 8:     if reached cycle or $s \in \mathcal{G}$ then
 9:         return
10:     $\pi(s) \leftarrow \arg\max_a Q(s, a, \tilde{S}, p_{s'|s,a})$      ▷ Max via BO
11:     $s' \leftarrow$ Sample($\hat{P}_{s'|s,a}(\tilde{S}|s, \pi(s))$)
12:     TrialRecurse($s', \mathcal{G}, \tilde{S}, p_{s'|s,a}$)
13:     $\pi(s) \leftarrow \arg\max_a Q(s, a, \tilde{S}, p_{s'|s,a})$      ▷ Max via BO
14:     $V(s) \leftarrow Q(s, \pi(s), \tilde{S}, p_{s'|s,a})$
15:     return $\pi, V$

16: function Q($s, a, \tilde{S}, p_{s'|s,a}$)
17:     if $\hat{P}_{s'|s,a}(\tilde{S}|s, a)$ has not been computed then
18:         $\hat{P}_{s'|s,a}(\tilde{S}|s, a) = \mathbf{0}$      ▷ $\hat{P}_{s'|s,a}$ is a shared matrix
19:         $\hat{S}, \hat{P}_{s'|s,a}(\hat{S}|s, a) \leftarrow$ TransitionModel($s, a, \tilde{S}, p_{s'|s,a}$)
20:     return $R(s, a) + \gamma^{\Delta t} \sum_{s' \in \hat{S}} \hat{P}_{s'|s,a}(s'|s, a) V(s')$
```

in $\hat{P}_{s'|s,a}$ by focusing on the relevant states and actions, as detailed in the next sections.

B. Sampling states

Algorithm 3 describes the state sampling procedures. The input to SampleStates in Alg. 3 includes the minimum number of states, $N_{\min}$, to sample at each iteration of BOIDP. It may be that more than $N_{\min}$ states are sampled, because sampling must continue until at least one terminal goal state is included in the resulting set $\tilde{S}$. To generate a discrete state set, we sample states both in the interior of S^o and on its boundary ∂S. Notice that we can always add more states by calling SampleStates.

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 sample a set of actions from A, for each of which we sample the next state s_n from the dataset D given the state-action pair $s_{neareast}, a$ (or from $p_{s'|s,a}$ if given). We choose the action a that gives us the s_n 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 $s_{rand} \to s_n$ incrementally until a collision is reached.

C. Focusing on the relevant states via RTDP

We 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 can use RTDP [2] to compute a value function focusing 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 $\arg\max_a Q_s(a)$ 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 $\arg\max_a Q_s(a)$, if the values for $\bar{S}$ have 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.

D. Focusing on good actions via BO

RTDP in Algorithm 4 relies on a challenging optimization over a continuous and possibly high-dimensional action space. Queries to $Q_s(a)$ in Eq. (1) 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 BO to efficiently approximate the solution and make full use of the parallel computing resources.

We first briefly review a sequential Gaussian-process optimization method, GP-EST [29], 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 $y_t = Q_s(a_t) +$

ϵ_t, where $\epsilon_t \sim \mathcal{N}(0,\sigma^2)$. 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 $\boldsymbol{\kappa}_t(a) = [\kappa(a_i,a)]_{a_i\in\mathfrak{D}_t}$ [30]:

$$\begin{aligned}\mu_t(a) &= \boldsymbol{\kappa}_t(a)^{\mathrm{T}}(\boldsymbol{K}_t+\sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}_t\\ \kappa_t(a,a') &= \kappa(a,a') - \boldsymbol{\kappa}_t(a)^{\mathrm{T}}(\boldsymbol{K}_t+\sigma^2\boldsymbol{I})^{-1}\boldsymbol{\kappa}_t(a')\ .\end{aligned}$$

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 relies on sequential tests of $Q_s(a)$, but 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 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 $B \subset A, |B| = M$ that characterize both diversity and quality:

$$F_s(B) = \log\det \boldsymbol{K}_B - \lambda \sum_{a\in B} \frac{h_u(s)-\mu_{t-1}(a)}{\sigma_{t-1}(a)} \tag{2}$$

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. λ can be chosen by cross-validation. We 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:

$$F_s(B\cup\{a\}) - F_s(B) \tag{3}$$

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

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

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

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

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

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

```
1: $\mathfrak{D}_0 \leftarrow \emptyset$
2: for $t = 1 \rightarrow T$ do
3:     $\mu_{t-1}, \sigma_{t-1} \leftarrow$ GP-predict($\mathfrak{D}_{t-1}$)
4:     $B \leftarrow \emptyset$
5:     for $i = 1 \rightarrow M$ do
6:         $B \leftarrow B \cup \{\arg\max_{a\in A} F_s(B\cup\{a\}) - F_s(B)\}$
7:     $\boldsymbol{y}_B \leftarrow Q_s(B)$                ▷ Test $Q_s$ in parallel
8:     $\mathfrak{D}_t \leftarrow \mathfrak{D}_{t-1} \cup \{B, \boldsymbol{y}_B\}$
```

V. Theoretical analysis

In this section, we characterize the theoretical behavior of BOIDP. Thm. 1 establishes the error bound for the value function on the $\hat{\pi}^*$-*relevant* set of states [2], where $\hat{\pi}^*$ is the optimal policy computed by BOIDP. A set $B \subseteq S$ is called π-*relevant* if all the states in B are reachable via finite actions from the starting state s_0 under the policy π. We denote $|\cdot|_B$ as the L_∞ norm of a function $\cdot$ over the set B. Without loss of generality, we assume $\min \Delta t = 1$.

Under mild conditions on $Q_s(a)$ specified in Thm. 1, we show that with finitely many actions selected by BO, the expected accumulated error expressed by the difference between the optimal value function V^* and the value function $\hat{V}$ of the policy computed by BOIDP in Alg. 1 on the $\hat{\pi}^*$-*relevant* set decreases sub-linearly in the number of actions selected for optimizing $Q_s(\cdot)$ in Eq. (1).

Theorem 1 (Error bound for BOIDP)**.** *Let $D = \{s_i, a_i, s'_i\}_{i=0}^N$ be the dataset that is collected from the true transition probability $p_{s'|s,a}$, $\forall(s,a) \in S\times A$. We assume that the transition model $p_{s'|s,a}$ estimated by the density estimator asymptotically converges to the true model. $\forall s \in S$, we assume $Q_s(a) = \int_{s'\in S} p_{s'|s,a}(s' \mid s,a)\left(R(s'\mid s,a) + \gamma^{\Delta t}V^*(s')\right)\mathrm{d}s'$ is a function locally continuous at $\arg\max_{a\in A} Q_s(a)$, where $V^*(\cdot) = \max_{a\in A} Q_\cdot(a)$ is the optimal value function for the continuous state-action MDP $\mathcal{M} = (S, A, p_{s'|s,a}, R, \gamma)$. $V^*(\cdot)$ is associated with an optimal policy whose relevant set contains at least one state in the goal region $\mathcal{G}$. At iteration k of RTDP in Alg. 4, we define $\hat{V}_k$ to be the value function for $\tilde{\mathcal{M}} = (\tilde{S}, A, \hat{P}_{s'|s,a}, R, \gamma)$ approximated by* BOIDP, *$\hat{\pi}_k$ to be the policy corresponding to $\hat{V}_k$, and B_k to be the $\hat{\pi}_k$-*relevant set. *We assume that*

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

is a function sampled from a Gaussian process with known priors and i.i.d Gaussian noise $\mathcal{N}(0,\sigma)$. If we allow Bayesian optimization for $Q_{s,k}(a)$ to sample T actions for each state and run RTDP in Alg. 4 until it converges with respect to the Cauchy's convergence criterion [31] with $\mathcal{K} < \infty$ iterations, in expectation,

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

where η_T is the maximum information gain of the selected actions [32, Theorem 5], $\nu = \max_{s,t,k}\min_{a\in A} G_{s,t,k}(a)$, and

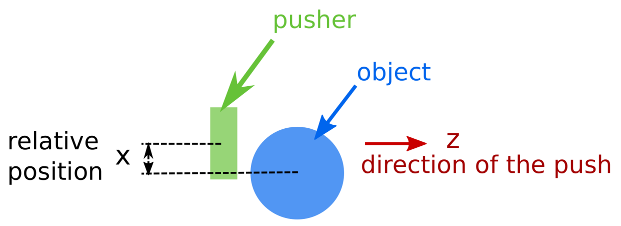

Fig. 2: Pushing a circular object with a rectangle pusher.

$G_{s,t,k}(\cdot)$ is the acquisition function in [29, Theorem 3.1] for state $s \in \tilde{S}$, iteration $t = 1, 2, \cdots, T$ in Alg. 5 or 6, and iteration $k = 1, 2, \cdots, \mathcal{K}$ of the loop in Alg. 4.

The detailed proof can be found in [33].

VI. Implementation and Experiments

We tested our approach in a quasi-static problem, in which a robot pushes a circular object through a planar workspace with obstacles in simulation[2]. We represent the action by the robot's initial relative position x to the object (its distance to the object center is fixed), the direction of the push z, and the duration of the push Δt, which are illustrated in Fig. 2. The companion video shows the behavior of this robot, controlled by a policy derived by BOIDP from a set of training examples.

In this problem, 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 $a = (u, \Delta t)$ 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 in a simulator. 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.

A. Importance of learning accurate models

Our method was 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, 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 rotation matrix for angle, and the noise is

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

[2]All experiments were run with Python 2.7.6 on Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 64GB memory.

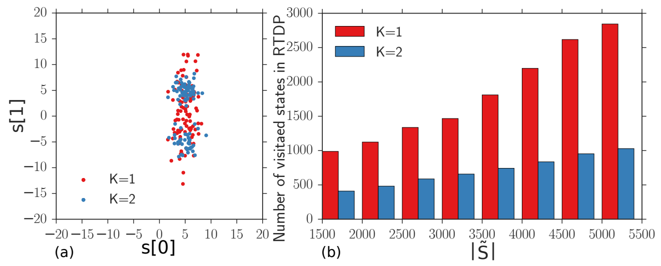

Fig. 3: (a) Samples from the single-mode Gaussian transition model ($K = 1$) and the two-component Gaussian mixture transition model ($K = 2$) in the free space when $a = 0$. (b) The number of visited states (y-axis) increases with the number of sampled states $|\tilde{S}|$ (x-axis). Planning with $K = 2$ visits fewer states in RTDP than with $K = 1$.

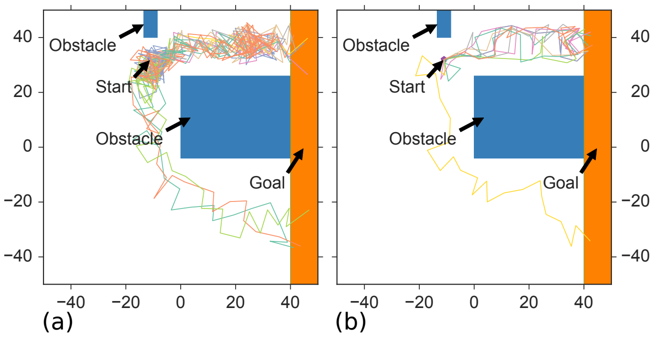

Fig. 4: (a) Samples of 10 trajectories with $K = 1$. (b) Samples of 10 trajectories with $K = 2$. Using the correct number of components for the transition model improves the quality of the trajectories.

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. 3 (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 4. Planning with the right model, $K = 2$, 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 Fig 4 (a) 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. 5. Furthermore, because the single-mode Gaussian estimates the noise to have a large variance, it causes RTDP to visit many more states than necessary, as shown in Fig. 3 (b).

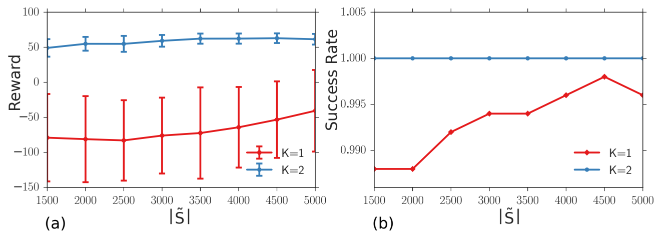

Fig. 5: (a): Reward. (b): Success rate. Using two components ($K = 2$) performs much better than using one component ($K = 1$) in terms of reward and success rate.

B. Focusing on the good actions and states

In this section we demonstrate the effectiveness of our strategies for limiting the number of states visited and actions modeled. We denote using Bayesian optimization in Lines 10 and 13 in Algorithm 4 as BO and using random selections as Rand.

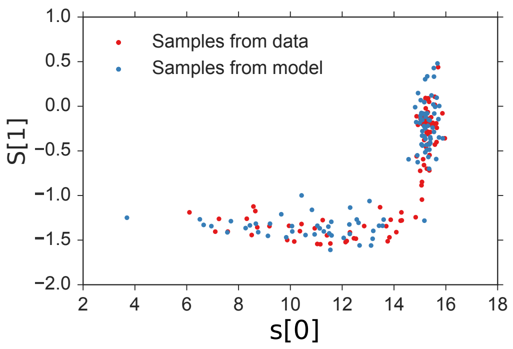

Fig. 6: The conditional distribution of Δs given $a = (z, x, \Delta t) = (0.0, 0.3, 2.0)$ is a multi-modal Gaussian.

We first demonstrate why BO is better than random for optimizing $Q_s(a)$ with the simple example from Sec. VI-A. 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 as shown in Fig 7 (a). We use the GP implementation and the default Matern52 kernel implemented in the GPy module [34] 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.

Next we consider a more complicated problem in which the action is the high level control of a pushing problem $a = (z, x, \Delta t)$, $z \in [0, 2\pi], x \in [-1.0, 1.0], \Delta t \in [0.0, 3.0]$ as illustrated in Fig. 2. The instantaneous reward is -1 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 x and Δt as variables in the Box2D simulator [35] where noise comes from variability of the executed action. We make use of the fact that the object is cylindrical (with radius 1.0) to reuse data. An example of the distribution of Δs given $a = (0.0, 0.3, 2.0)$ is shown in Fig. 6.

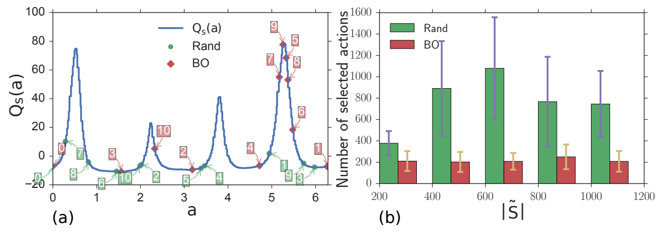

Fig. 7: (a) We optimize $Q_s(a)$ with BO and Rand by sequentially sampling 10 actions. BO selects actions more strategically than Rand. (b) BO samples fewer actions than Rand in the pushing problem for all settings of $|\tilde{S}|$.

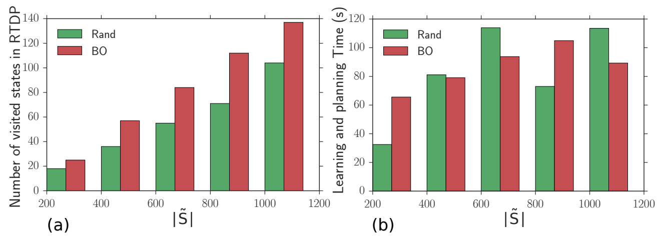

Fig. 8: (a) Number of visited states in RTDP. Both of Rand and BO consistently focus on about 10% states for planning. (b) Learning and planning time of BO and Rand.

We compare policies found by Rand and BO with the same set of sampled states ($|\tilde{S}| = 200, 400, 600, 800, 1000$) within approximately the same amount of total computation time. They are both able to compute the policy in $30 \sim 120$ seconds, as shown in Fig. 8 (b). In more realistic domains, it is possible that learning 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. 9, and the average number of actions selected for visited states in Fig. 7(b). 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. 10.

From Fig. 8 (a), it is not hard to see that RTDP successfully controlled the number of visited states to be only a small fraction of the whole sampled set of states. Interestingly, BO was able to visit slightly more states with RTDP and as a result, explored more possible states that it is likely to encounter during the execution of the policy, which may be a factor that contributed to its better performance in terms of reward and success rate in Fig. 9. We did not compare with pure value iteration because the high computational cost of computing models for all the states made it infeasible.

BOIDP is able to compute models for only around 10% of the sampled states and about 200 actions per state. If

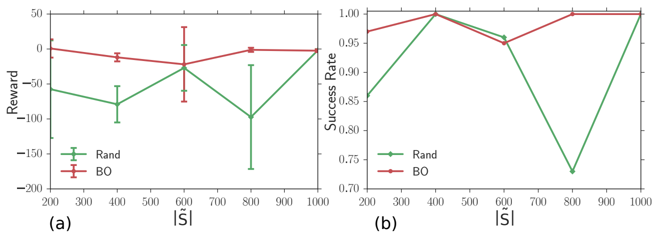

Fig. 9: (a) Reward. (b) Success rate. BO achieves better reward and success rate, with many fewer actions and slightly more visited states.

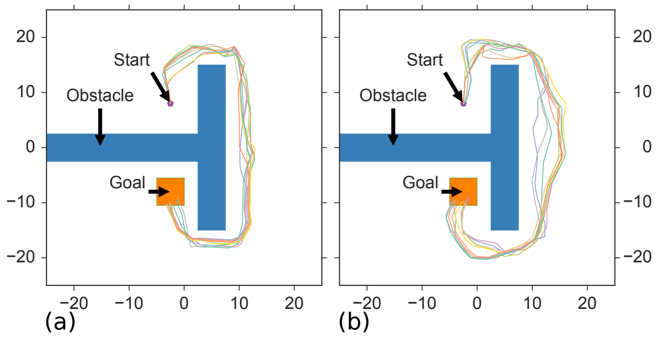

Fig. 10: (a) 10 samples of trajectories generated via Rand with 1000 states. (b) 10 samples of trajectories generated via BO with 1000 states.

we consider a naive grid discretization for both action (3 dimension) and state (2 dimension) with 100 cells for each dimension, the number of models we would have to compute is on the order of 10^{10}, compared to our approach, which requires only 10^4.

VII. Conclusion

An important class of robotics problems are intrinsically continuous in both state and action space, and may demonstrate non-Gaussian stochasticity in their dynamics. We have provided a framework to plan and learn effectively for these problems. We achieve efficiency by focusing on relevant subsets of state and action spaces, while retaining guarantees of asymptotic optimality.

References

[1] 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 *IROS*, 2016.

[2] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1, pp. 81–138, 1995.

[3] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. TR 98-11, 1998.

[4] V. A. Huynh, S. Karaman, and E. Frazzoli, "An incremental sampling-based algorithm for stochastic optimal control," *IJRR*, vol. 35, no. 4, pp. 305–333, 2016.

[5] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Control Systems*, vol. 14, no. 1, pp. 13–24, 1994.

[6] L. C. Baird, III and A. H. Klopf, "Reinforcement learning with high-dimensional continuous actions," Wright Laboratory, Wright Patterson Air Force Base, Tech. Rep., 1993.

[7] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[8] H. van Hasselt and M. A. Wiering, "Continuous action spaces," in *ADPRL*, 2007.

[9] B. D. Nichols, "Continuous action-space reinforcement learning methods applied to the minimum-time swing-up of the acrobot," in *ICSMC*, 2015.

[10] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *ICML*, 2011.

[11] H. S. Jakab and L. Csató, "Reinforcement learning with guided policy search using Gaussian processes," in *IJCNN*, 2012.

[12] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Model-based reinforcement learning with continuous states and actions." in *ESANN*, 2008.

[13] A. Rottmann and W. Burgard, "Adaptive autonomous control using online value iteration with Gaussian processes," in *ICRA*, 2009.

[14] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[15] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7, pp. 1508–1524, 2009.

[16] M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar *et al.*, "Bayesian reinforcement learning: A survey," *Foundations and Trends in Machine Learning*, vol. 8, no. 5–6, pp. 359–483, 2015.

[17] M. K. Titsias and M. Lázaro-Gredilla, "Variational heteroscedastic Gaussian process regression," in *ICML*, 2011.

[18] C. Yuan and C. Neubauer, "Variational mixture of Gaussian process experts," in *NIPS*, 2009.

[19] T. M. Moldovan, S. Levine, M. I. Jordan, and P. Abbeel, "Optimism-driven exploration for nonlinear systems," in *ICRA*, 2015.

[20] M. Kopicki, J. Wyatt, and R. Stolkin, "Prediction learning in robotic pushing manipulation," in *ICAR*, 2009.

[21] M. Kopicki, "Prediction learning in robotic manipulation," Ph.D. dissertation, University of Birmingham, 2010.

[22] M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, and J. Wyatt, "Learning to predict how rigid objects behave under simple manipulation," in *ICRA*, 2011.

[23] T. Yee, V. Lisy, and M. Bowling, "Monte Carlo tree search in continuous action spaces with execution uncertainty," in *IJCAI*, 2016.

[24] M. Frean and P. Boyle, "Using Gaussian processes to optimize expensive functions," in *Australasian Joint Conference on Artificial Intelligence*, 2008.

[25] 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," University of British Columbia, Tech. Rep. TR-2009-023, 2009.

[26] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite Markov decision processes with Gaussian processes," in *NIPS*, 2016.

[27] D. Wied and R. Weißbach, "Consistency of the kernel density estimator: a survey," *Statistical Papers*, vol. 53, no. 1, pp. 1–21, 2012.

[28] A. Moitra and G. Valiant, "Settling the polynomial learnability of mixtures of Gaussians," in *FOCS*, 2010.

[29] Z. Wang, B. Zhou, and S. Jegelka, "Optimization as estimation with Gaussian processes in bandit settings," in *AISTATS*, 2016.

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

[31] A. L. Cauchy, *Cours d'analyse de l'Ecole Royale Polytechnique*. Imprimerie royale, 1821, vol. 1.

[32] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *ICML*, 2010.

[33] Z. Wang, S. Jegelka, L. P. Kaelbling, and T. Lozano-Pérez, "Focused model-learning and planning for non-Gaussian continuous state-action systems," *arXiv preprint arXiv:1607.07762*, 2016.

[34] GPy, "GPy: a Gaussian process framework in python," http://github.com/SheffieldML/GPy, since 2012.

[35] E. Catto, "Box2D, a 2D physics engine for games," http://box2d.org, 2011.