Scalable Reinforcement Learning via Trajectory Optimization and Approximate Gaussian Process Regression

Yunpeng Pan[1,2], **Xinyan Yan**[1,3], **Evangelos Theodorou**[1,2], **and Byron Boots**[1,3]

[1]Institute for Robotics and Intelligent Machines, Georgia Institute of Technology
[2]School of Aerospace Engineering, Georgia Institute of Technology
[3]School of Interactive Computing, Georgia Institute of Technology
{ypan37,xyan43,evangelos.theodorou}@gatech.edu, bboots@cc.gatech.edu

1 Introduction & Related Work

Over the last decade, reinforcement learning (RL) has begun to be successfully applied to robotics and autonomous systems. While model-free RL has demonstrated promising results [1, 2, 3], it requires human expert demonstrations and relies on lots of direct interactions with the physical systems. In contrast, model-based RL was developed to address the issue of sample inefficiency by learning dynamics models explicitly from data, which helps to provide better generalization [4, 5]. However, model-based methods suffer from two issues: 1) classical value function approximation methods [6, 7] and modern global policy search methods [5] are computationally inefficient for moderate to high-dimensional problems; and 2) model errors significantly degrade the performance.

In order to design an efficient RL algorithm, we combine the attractive characteristics of two approaches: local trajectory optimization and random feature approximations. Local trajectory optimization, such as Differential Dynamic Programming (DDP) [8] are a class of approaches for solving nonlinear optimal control problems. These methods generate locally optimal control policies along with an optimal trajectory. Compared to global approaches, DDP shows superior computational efficiency and scalability to high-dimensional problems [9, 10]. In all of the variations of DDP [11, 12, 13, 14, 15], the principal limitation is that it relies on accurate and explicit representation of the dynamics, which is generally challenging to obtain due to the complexity of the relationships between states, controls and observations in autonomous systems. In this work we take a nonparametric approach to learn the dynamics based on Gaussian processes (GPs). GPs have demonstrated encouraging performance in modeling dynamical systems [16, 17, 18, 19]. However, standard GP regression is computationally expensive and does not scale to moderate/large datasets. While a number of approximation methods exist, a recent method sparse spectrum Gaussian process regression (SSGPR) [20, 21], stands out with a superior combination of efficiency and accuracy compared to approximation strategies such as local GPR [22]. SSGPR is based on kernel function approximation using finite dimensional random feature mappings, introduced in [23].

The proposed method is related to a number of recently developed model-based RL approaches that use GPs to represent dynamics models [19, 24, 25]. While featuring impressive data-efficiency, most of these methods are computation-intensive and do not scale to moderate/large size of datasets (i.e., a few thousands data points). Therefore they are not suitable for data-intensive applications under real-time or computational power constraints. Furthermore, they do not adaptively update models or re-optimize policies "on the fly" (during interactions with the physical systems) due to the significant computational burden. This results in lack of robustness and generalizability that restrict their applicability in uncertain environment. By combining the benefits of both DDP and SSGPR, we will show that our approach is able to scale to high-dimensional dynamical systems and moderate to large datasets.

2 Proposed Approach

We consider a general unknown dynamical system described by the following differential equation

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u})\mathrm{d}t + \mathrm{d}\xi, \qquad \mathbf{x}(t_0) = \mathbf{x}_0, \qquad \mathrm{d}\xi \sim \mathcal{N}(0, \sigma_\xi), \tag{1}$$

with state $\mathbf{x} \in \mathbb{R}^n$, control $\mathbf{u} \in \mathbb{R}^m$, unknown transition dynamics $\mathbf{f}$, time t and standard Brownian motion noise $\xi \in \mathbb{R}^p$. The RL problem is defined as finding a sequence of controls that minimizes the expected cost $J^\pi(\mathbf{x}(t_0)) = \mathbb{E}_{\mathbf{x}}\big[\underbrace{q(\mathbf{x}(T))}_{\textbf{Terminal cost}} + \int_{t_0}^{T} \underbrace{\mathcal{L}(\mathbf{x}(t), \pi(\mathbf{x}(t)), t)}_{\textbf{Running cost}} \mathrm{d}t\big]$, where $\mathbf{u}(t) = \pi(\mathbf{x}(t))$ is the control policy. The cost $J^\pi(\mathbf{x}(t_0))$ is defined as the expectation of the total cost accumulated from t_0 to T. For the rest of our analysis, we denote $\mathbf{x}_k = \mathbf{x}(t_k)$ in discrete-time where $k = 0, 1, ..., H$ is the time step, we use this subscript rule for other variables as well.

2.1 Model learning via incremental sparse spectrum Gaussian process regression

Learning a continuous functional mapping from state-control pair $\tilde{\mathbf{x}} = (\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n+m}$ to state transition $\mathrm{d}\mathbf{x} \in \mathbb{R}^n$ can be viewed as an probabilistic inference with the goal of inferring $\mathrm{d}\mathbf{x}$ given $\tilde{\mathbf{x}}$. In this subsection, we introduce an approximate Gaussian processes (GP) approach to learning the dynamics model in Eq. 1. In standard GP regression (GPR), the prior distribution of the underlying function is defined as $f(\tilde{\mathbf{x}}) \sim \mathcal{GP}(0, k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j))$, where k is the covariance or the kernel function. We consider the popular Squared Exponential (SE) covariance function with Automatic Relevance Determination (ARD) distance measure $k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) = \sigma_f^2 \exp(-\frac{1}{2}(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)^\mathsf{T}\mathbf{P}^{-1}(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j))$, $\mathbf{P} = \mathrm{diag}([\ l_1^2 \quad ... \quad l_{n+m}^2\ ])$. The hyperparameters of the kernel consist of the signal variance σ_f^2 and the length scales for input space $\mathbf{l} = [l_1, ..., l_{n+m}]$. Given a sequence of N state-control pairs $\tilde{\mathbf{X}} = \{(\mathbf{x}_1, \mathbf{u}_1), ..., (\mathbf{x}_N, \mathbf{u}_N)\}$ and the corresponding state transition $\mathrm{d}\mathbf{X} = \{\mathrm{d}\mathbf{x}_1, ..., \mathrm{d}\mathbf{x}_N\}$, the prior joint distribution of the output of a test state-control pair $\tilde{\mathbf{x}}^* = (\mathbf{x}^*, \mathbf{u}^*)$ and the observed outputs can be written as $\begin{pmatrix} \mathrm{d}\mathbf{X} \\ \mathrm{d}\mathbf{x}^* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{G} & \mathbf{k}^* \\ \mathbf{k}^{*\mathsf{T}} & k^* \end{bmatrix}\right)$, where $\mathbf{G} = \mathbf{K} + \sigma_n^2\mathbf{I}$, $\mathbf{K} = [k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)]_{i,j=1}^{N,N}$, $\mathbf{k}^* = [k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}^*)]_{i=1}^{N}$, and $k^* = k(\tilde{\mathbf{x}}^*, \tilde{\mathbf{x}}^*)$. The posterior distribution of the state transition at $\tilde{\mathbf{x}}^*$ is derived as $\mathrm{d}\mathbf{x}^*|\tilde{\mathbf{X}}, \mathrm{d}\mathbf{X}, \tilde{\mathbf{x}}^* \sim \mathcal{N}(\mathbf{k}^{*\mathsf{T}}\mathbf{G}^{-1}\mathrm{d}\tilde{\mathbf{X}}, \quad k^* - \mathbf{k}^{*\mathsf{T}}\mathbf{G}^{-1}\mathbf{k}^*)$. Unfortunately, GPR exhibits significant practical limitations for learning and inference on large datasets due to its $O(N^3)$ computation and $O(N^2)$ space complexity, which is a direct consequence of having to store and invert the matrix $\mathbf{G}$. This computational inefficiency is a bottleneck for applying GP-based RL in real-time.

Sparse Spectrum Gaussian Process Regression (SSGPR)[20] is a recent approach that provides a principled approximation of GPR by employing a random Fourier feature approximation of the kernel function[23]. Based on *Bochner's theorem* [26], any shift-invariant kernel functions can be represented as the Fourier transform of a unique measure

$$k(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j) = \int_{\mathbb{R}^n} e^{i\boldsymbol{\omega}^\mathsf{T}(\mathbf{x}_i - \mathbf{x}_j)} p(\boldsymbol{\omega}) d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega}}[\boldsymbol{\phi}_{\boldsymbol{\omega}}(\tilde{\mathbf{x}}_i)\boldsymbol{\phi}_{\boldsymbol{\omega}}(\tilde{\mathbf{x}}_j)],\ \boldsymbol{\phi}_{\boldsymbol{\omega}}(\tilde{\mathbf{x}}) = [\cos(\boldsymbol{\omega}^\mathsf{T}\tilde{\mathbf{x}})\ \sin(\boldsymbol{\omega}^\mathsf{T}\tilde{\mathbf{x}})]. \tag{2}$$

We can, therefore, unbiasedly approximate the SE kernel function by drawing r random samples from the distribution $\mathrm{p}(\boldsymbol{\omega})$. More precisely $k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \approx \sum_{i=1}^{r} \boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\tilde{\mathbf{x}}_i)^\mathsf{T}\boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\tilde{\mathbf{x}}_j) = \boldsymbol{\phi}(\tilde{\mathbf{x}}_i)^\mathsf{T}\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$, where $\boldsymbol{\phi}(\tilde{\mathbf{x}}) = \frac{\sigma_f}{\sqrt{r}}[\ \cos(\boldsymbol{\Omega}^\mathsf{T}\tilde{\mathbf{x}}) \quad \sin(\boldsymbol{\Omega}^\mathsf{T}\tilde{\mathbf{x}})\ ]^\mathsf{T}$, $\boldsymbol{\Omega} \sim \mathcal{N}(\mathbf{0}, \mathbf{P}^{-1})_{n \times r}$. With this feature mapping, the function from state-control pair to state transition can be represented as a weighted sum of the basis or the feature functions $\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\tilde{\mathbf{x}}^*)$. Assuming the prior distribution of weights of the features $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_p)$, the posterior distribution of $\mathrm{d}\mathbf{x}^*$ can be derived as in the standard Bayesian linear regression

$$\mathrm{d}\mathbf{x}^*|\tilde{\mathbf{X}}, \mathrm{d}\mathbf{X}, \tilde{\mathbf{x}}^* \sim \mathcal{N}(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}^*, \sigma_n^2(1 + \boldsymbol{\phi}^{*\mathsf{T}}\mathbf{A}^{-1}\boldsymbol{\phi}^*)), \tag{3}$$

where $\boldsymbol{\phi}(\tilde{\mathbf{x}}^*) = \boldsymbol{\phi}^*, \mathbf{w} = \mathbf{A}^{-1}\boldsymbol{\Phi}\mathrm{d}\mathbf{X}, \mathbf{A} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma_n^2\boldsymbol{\Sigma}_p^{-1}, \boldsymbol{\Phi} = [\ \boldsymbol{\phi}(\tilde{\mathbf{x}}_1) \quad ... \quad \boldsymbol{\phi}(\tilde{\mathbf{x}}_N)\ ]$. Thus the computation complexity becomes $O(Nr^2 + r^3)$, which is significantly more efficient than GPR with $O(N^3)$ time complexity when the number of random features is much smaller than the number of training samples. To make this method incremental, so that weights $\mathbf{w}$ are updated given a new sample, we do not store or invert $\mathbf{A}$ explicitly. Instead, we keep track of its upper triangular Cholesky factor $\mathbf{A} = \mathbf{R}^\mathsf{T}\mathbf{R}$ as in [21]. Given a new sample, the Cholesky factor $\mathbf{R}$ is updated through rank-1 update, with computation complexity $O(r^2)$. Weights $\mathbf{w}$ can be recovered anytime using back-substitution with time $O(r^2)$.

2.2 Local approximation and control policy learning

In order to perform trajectory optimization based on the learned model, we create a local model along a nominal trajectory $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$. Based on the analysis in section 2.1, the SSGPR representation of the dynamics can be rewritten as $\mathbf{x}_{k+1} = \mathcal{F}(\mathbf{x}_k, \mathbf{u}_k)$. where $\mathcal{F}$ is an explicit function derived from the predictive mean of SSGPR (3). Define the control and state variations $\delta\mathbf{x}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k$ and $\delta\mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k$. In this work we consider the first-order approximation of the dynamics. More precisely we have $\delta\mathbf{x}_{k+1} = \mathcal{F}_k^x \delta\mathbf{x}_k + \mathcal{F}_k^u \delta\mathbf{u}_k$, where the Jacobian matrices $\mathcal{F}_k^x$ and $\mathcal{F}_k^u$ are computed using chain-rule $\mathcal{F}_k^x = \nabla_{\mathbf{x}_k}\mathcal{F} = \frac{\partial \mathbf{x}_{k+1}}{\partial \mathrm{d}\mathbf{x}_k}\frac{\partial \mathrm{d}\mathbf{x}_k}{\partial \tilde{\mathbf{x}}_k}\frac{\partial \tilde{\mathbf{x}}_k}{\partial \mathbf{x}_k}$ and $\mathcal{F}_k^u = \nabla_{\mathbf{u}_k}\mathcal{F} = \frac{\partial \mathbf{x}_{k+1}}{\partial \mathrm{d}\mathbf{x}_k}\frac{\partial \mathrm{d}\mathbf{x}_k}{\partial \tilde{\mathbf{x}}_k}\frac{\partial \tilde{\mathbf{x}}_k}{\partial \mathbf{u}_k}$. Based on the GP representation the partial derivative $\frac{\partial \mathrm{d}\mathbf{x}_k}{\partial \tilde{\mathbf{x}}_k}$ is specified as $\frac{\partial \mathrm{d}\mathbf{x}_k}{\partial \tilde{\mathbf{x}}_k} \approx \frac{\partial \mathbb{E}(\mathrm{d}\mathbf{x}_k)}{\partial \tilde{\mathbf{x}}_k} = \mathbf{w}^\mathsf{T}\left(\begin{bmatrix}\boldsymbol{\Omega}^\mathsf{T}\\ \boldsymbol{\Omega}^\mathsf{T}\end{bmatrix} \circ \left(\begin{bmatrix}-\mathbf{s}_k\\ \mathbf{c}_k\end{bmatrix}\mathbf{1}^\mathsf{T}\right)\right)$, where $\mathbf{s}_k = \frac{\sigma_f}{\sqrt{r}}\sin(\boldsymbol{\Omega}^\mathsf{T}\tilde{\mathbf{x}}_k)$, $\mathbf{c}_k = \frac{\sigma_f}{\sqrt{r}}\cos(\boldsymbol{\Omega}^\mathsf{T}\tilde{\mathbf{x}}_k)$, $\circ$ is the matrix element-wise multiplication, and $\mathbf{1}$ is a column vector with n ones. Other partial derivatives can be trivially obtained [24]. We can, therefore, compute the Jacobian matrices analytically without using numerical methods.

The Bellman equation for the value function in discrete-time is specified as $V(\mathbf{x}_k, k) = \min_{\mathbf{u}_k}\mathbb{E}\Big[\underbrace{\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + V\Big(\mathcal{F}(\mathbf{x}_k, \mathbf{u}_k), k+1\Big)}_{Q(\mathbf{x}_k, \mathbf{u}_k)} | \mathbf{x}_k\Big]$.We create a quadratic local model of the value function by expanding the Q-function up to the second order $Q_k(\bar{\mathbf{x}}_k + \delta\mathbf{x}_k, \bar{\mathbf{u}}_k + \delta\mathbf{u}_k) \approx Q_k^0 + Q_k^x\delta\mathbf{x}_k + Q_k^u\delta\mathbf{u}_k + \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_k\\ \delta\mathbf{u}_k\end{bmatrix}^\mathrm{T}\begin{bmatrix}Q_k^{xx} & Q_k^{xu}\\ Q_k^{ux} & Q_k^{uu}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_k\\ \delta\mathbf{u}_k\end{bmatrix}$,where the superscripts of the Q-function indicate derivatives. For instance, $Q_k^x = \nabla_x Q_k(\mathbf{x}_k, \mathbf{u}_k)$. For the rest of the paper, we will use this superscript rule for $\mathcal{L}$ and V as well. To find the optimal control policy, we compute the local variations in control $\delta\hat{\mathbf{u}}_k$ that maximize the Q-function

$$\delta\hat{\mathbf{u}}_k = \arg\max_{\delta\mathbf{u}_k}\Big[Q_k(\bar{\mathbf{x}}_k + \delta\mathbf{x}_k, \bar{\mathbf{u}}_k + \delta\mathbf{u}_k)\Big] = \underbrace{-(Q_k^{uu})^{-1}Q_k^u}_{\mathbf{I}_k}\underbrace{-(Q_k^{uu})^{-1}Q_k^{ux}}_{\mathbf{L}_k}\delta\mathbf{x}_k = \mathbf{I}_k + \mathbf{L}_k\delta\mathbf{x}_k. \quad (4)$$

The optimal control can be found as $\hat{\mathbf{u}}_k = \bar{\mathbf{u}}_k + \delta\hat{\mathbf{u}}_k$. The quadratic expansion of the value function is backward propagated based on the equations that follow $Q_k^x = \mathcal{L}_k^x + V_k^x\mathcal{F}_k^x, Q_k^u = \mathcal{L}_k^u + V_k^x\mathcal{F}_k^u, Q_k^{xx} = \mathcal{L}_k^{xx} + (\mathcal{F}_k^x)^\mathrm{T}V_k^{xx}\mathcal{F}_k^x, Q_k^{ux} = \mathcal{L}_k^{ux} + (\mathcal{F}_k^u)^\mathrm{T}V_k^{xx}\mathcal{F}_k^x, Q_k^{uu} = \mathcal{L}_k^{uu} + (\mathcal{F}_k^u)^\mathrm{T}V_k^{xx}\mathcal{F}_k^u, V_{k-1} = V_k + Q_k^u\mathbf{I}_k, V_{k-1}^x = Q_k^x + Q_k^u\mathbf{L}_k, V_{k-1}^{xx} = Q_k^{xx} + Q_k^{xu}\mathbf{L}_k$. The second-order local approximation of the value function is propagated backward in time. We then generate a locally optimal trajectory by propagating the GP-based dynamics forward in time. Successive application of this scheme would lead to an optimal control policy as well as state-control trajectory.

A distinctive feature of our RL scheme is that we adaptively update the model and re-optimize the policy during interactions with the physical system. At each time step we initialize with the policy obtained on the previous run, and update the learned model by incorporating data point collected in the previous step. This online learning scheme with "warm-start" lead to very efficient computation and improved performance compared to offline learning. A high-level summary of the proposed algorithm is shown in **Algorithm** 1.

Algorithm 1 Model-based RL via DDP and SSGPR (1-3: **offline learning**, 4-8: **online learning**)

1: **Initialization:** Collect data by applying pre-specified or random controls to the system (1),
2: **Model learning:** Train GP hyperparameters. Sample random features and compute their weights (sec.2.1).
3: **Trajectory optimization:** Perform DDP based on the learned GP dynamics model (sec.2.2).
4: **for** $k = 0{:}H\text{-}1$ **do**
5: **Interaction:** Apply one-step control $\hat{\mathbf{u}}_k$ to the system and move one step forward. Record data.
6: **Model adaptation:** Incorporate data and update random features' weights (sec.2.1).
7: **Trajectory optimization:** Perform DDP with updated model and planning horizon $H - k$. Initialize with the previously optimized policy/trajectory (sec.2.2).
8: **end for**
9: **return** Optimal control sequence $\hat{\mathbf{u}}_{0,..,H}$.

3 Experiments and Analysis

We consider 2 simulated RL tasks: quadrotor flight and PUMA-560 robotic arm reaching. Quadrotor is an underactuated rotorcraft which rely on symmetry in order to fly in a conventional, stable fight

regime. With 12 dimensional states, 6 degrees of freedom and 4 rotors to control them, the quadrotor's task is to fly to 10 different targets from the same initial position in 30 time steps. PUMA-560 is a 3D robotic arm that has 12 state dimensions, 6 degrees of freedom with 6 actuators on each joint. The task is to steer the end-effector to the desired position and orientation in 50 time steps. We performed experiment#1 and experiment#2 on the quadrotor and PUMA-560, respectively. In experiment#1, first, we obtained training data by performing 150 rollouts (4500 data points) around a pre-specified trajectory (different from the tasks). For offline learning we sampled 1000 random features. Second, we performed trajectory optimization based on the learned model, and obtained the optimal control policies. Fig.2a shows the results by applying policies to the true dynamics. Third, we re-sampled 300, 600, and 1000 random features respectively and re-optimized the policies and trajectories obtained in the previous time step, shown in fig. 2b, In experiment#2 we followed similar steps but used the same sampled random features for both offline and online learning (without re-sampling of random features). We collected 5000 data points offline from random explorations (applying random control commands). And sampled 100, 300, 800 random features for both offline and online learning. Results are shown in fig.3. Computation time for online learning is shown in fig.1. From both experiments we observed that 1) online learning improves cost reduction performance compared to offline learning thanks to model adaptation and re-optimization; 2) the performance as well as computation time increase with the number of random features. The selection of random features number would depend on particular applications with emphasis on accuracy or efficiency.

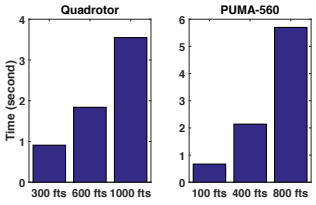
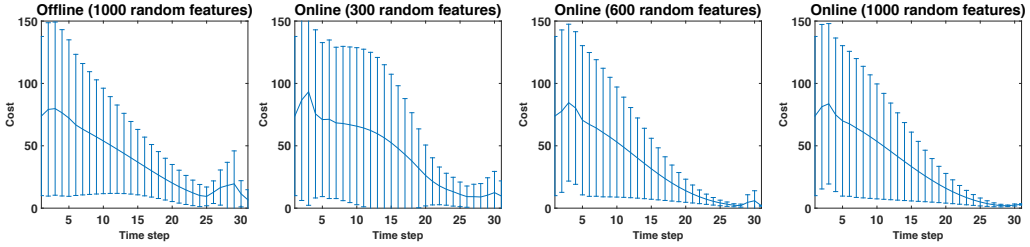
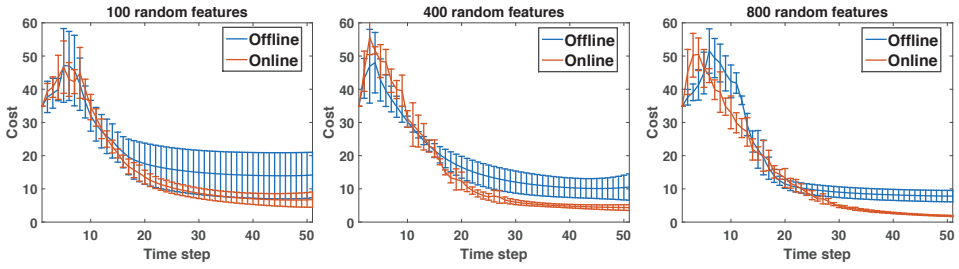


Figure 1: Average online computation time (second) per time step



(a) Offline learning using 1000 random features (b) Online learning using 300 random features (c) Online learning using 600 random features (d) Online learning using 1000 random features

Figure 2: Experiment 1. Each figure show the mean and standard deviation of final trajectory costs over 10 independent trials using offline and online learning.



(a) 100 random features (b) 400 random features (c) 800 random features

Figure 3: Experiment 2. Each figure shows the mean and standard deviation of final trajectory costs over 10 independent trials for offline and online learning using different number of random features.

4 Conclusion and Future Works

We introduced a model-based reinforcement learning algorithm based on trajectory optimization and sparse spectrum Gaussian process regression. Our approach is able to scale to high-dimensional dynamical systems and large datasets. In addition, our method updates the learned models in an incremental fashion and re-optimizes the control policies during interactions with the physical systems via successive local approximations. Future works will focus on extending the applicability of this method, for instance, using predictive covariances to guide exploration and exploitation.

References

[1] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[2] E. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *The Journal of Machine Learning Research*, 11:3137–3181, 2010.

[3] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 281–288. ACM, 2012.

[4] C.G. Atkeson and J.C. Santamaria. A comparison of direct and model-based reinforcement learning. In *In International Conference on Robotics and Automation*. Citeseer, 1997.

[5] M.P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[6] D.P. Bertsekas and J.N. Tsitsiklis. Neuro-dynamic programming (optimization and neural computation series, 3). *Athena Scientific*, 7:15–23, 1996.

[7] A.G. Barto, W. Powell, J. Si, and D.C. Wunsch. Handbook of learning and approximate dynamic programming. 2004.

[8] D. Jacobson and D. Mayne. Differential dynamic programming. 1970.

[9] Y. Tassa, T. Erez, and W. D. Smart. Receding horizon differential dynamic programming. In *NIPS*, 2007.

[10] Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. *ICRA 2014*.

[11] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005*, pages 300–306. IEEE, 2005.

[12] E. Theodorou, Y. Tassa, and E. Todorov. Stochastic differential dynamic programming. In *American Control Conference (ACC), 2010*, pages 1125–1132. IEEE, 2010.

[13] D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal feedback control with learned internal dynamics models. In *From Motor Learning to Interaction Learning in Robots*, pages 65–84. Springer, 2010.

[14] J. Van Den Berg, S. Patil, and R. Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[15] Y. Pan, K. Bakshi, and E.A. Theodorou. Robust trajectory optimization: A cooperative stochastic game theoretic approach. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[16] J. Ko and D. Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

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

[18] P. Hemakumara and S. Sukkarieh. Learning uav stability and control derivatives using gaussian processes. *IEEE Transactions on Robotics*, 29:813–824, 2013.

[19] M. Deisenroth, D. Fox, and C. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transsactions on Pattern Analysis and Machine Intelligence*, 27:75–90, 2015.

[20] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal. Sparse spectrum gaussian process regression. *The Journal of Machine Learning Research*, 99:1865–1881, 2010.

[21] A. Gijsberts and G. Metta. Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*, 41:59–69, 2013.

[22] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[23] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2007.

[24] Y. Pan and E. Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1907–1915, 2014.

[25] Andras Kupcsik, Marc Peter Deisenroth, Jan Peters, Ai Poh Loh, Prahlad Vadakkepat, and Gerhard Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 2014.

[26] W. Rudin. *Fourier analysis on groups*. Interscience Publishers, New York, 1962.