SOLAR-GP: Sparse Online Locally Adaptive Regression Using Gaussian Processes for Bayesian Robot Model Learning and Control

Brian Wilcox and Michael C. Yip

***Abstract*—Machine learning methods have been widely used in robot control to learn inverse mappings. These methods are used to capture the entire non-linearities and non-idealities of a system that make geometric or phenomenological modeling difficult. Most methods employ some form of off-line or batch learning where training may be performed prior to a task, or in an intermittent manner, respectively. These strategies are generally unsuitable for teleoperation, where commands and sensor data are received in sequential streams and models must be learned on-the-fly. We combine sparse, local, and streaming methods to form Sparse Online Locally Adaptive Regression using Gaussian Processes (SOLAR–GP), which trains streaming data on localized sparse Gaussian Process models and infers a weighted local function mapping of the robot sensor states to joint states. The resultant prediction of the teleoperation command is used for joint control. The algorithm was adapted to perform on arbitrary link manipulators including the Baxter robot, where modifications to the algorithm are made to run training and prediction in parallel so as to keep consistent, high-frequency control loop rates. This framework allows for a user-defined cap on complexity of generated local models while retaining information on older regions of the explored state-space.**

***Index Terms*—Model learning for control, learning and adaptive systems, neural and fuzzy control.**

I. Introduction

MANY non-trivial robotic scenarios involve operations under noisy, biased, and/or uncertain robot kinematics and environmental influences. This includes surgery, search and rescue, underwater navigation, and human-robot interactions. Many of these robots, due to the sensitivities of their environments and the objects and actors in those environments, often have mechanical compliance in their system for safety. Normally, control for precision tasks involves accurate modeling and feedback control, yet complex robot mechanics can be overwhelming to model sufficiently and accurately for precision tracking and control. For precise control, these models must be accurate across large operating state spaces and multiple dimensions, and may need to describe non-stationary and dynamically changing phenomenon. Similarly the environmental effects, including heat, fluid, non-prehensile contact and obstruction may change the effective control mappings from joints to end-effector pose. Furthermore, in cases of teleoperation, inaccurate models can result in confusion to the operator, reduced or unstable tracking performance and significant detrimental effects into their ability to perform tasks [1].

Rather than phenomenological modeling all of the robot and environmental effects, model-free machine learning methods have been explored to learning the inverse mapping directly from sensor outputs to joint inputs. Most methods employed fixed parameter non-linear regressors such as neural networks or Gaussian mixture models [2] that are not suitable for online learning [3] nor teleoperation in unknown and new scenarios (unknown robot manipulator, unknown enviroments, or both). Adaptive control, an early class of online learning strategies, provides robust control but under strong assumptions on the robot kinematics, dynamics, and environment [4], and is thus not entirely model-free. Model-free, Jacobian-based controllers [1] provide online learning under very few assumptions but do not build a global model. Recently, Gaussian Process regression (GPR) has shown promise in modeling the inverse dynamics of robotic systems [5]. Its intuitive structure and Bayesian framework allow for the uncertainties associated with model regression and prediction, as well as noise, to be explicitly expressed.

In this letter, we consider Gaussian Processes for online learning of a Bayesian model for control during teleoperation. However, the computational cost of training and prediction for GPs easily reaches above 1 second and several hundred milliseconds for real-time control even with a small number (under 100) of free parameters, making it a challenge for use in control loops. Prediction involves inversion of the covariance matrix of order $\mathcal{O}(N^3)$, where N is the number of training data points [6]. Sparse Gaussian Processes reduce the computational complexity of regression (and optimization); by selecting $M \ll N$ support points, this complexity can be reduced to $\mathcal{O}(NM^2)$. Various sparse methods for Gaussian processes have been introduced in the literature, with large focus on selection of the support points [7], [8]. Online sparse methods exist, which take advantage of incremental updates to the model and training while forgetting old data [9]. However, global sparse

Manuscript received September 10, 2019; accepted January 14, 2020. Date of publication February 17, 2020; date of current version February 26, 2020. This letter was recommended for publication by Associate Editor S. Weiss and Editor P. Rocco upon evaluation of the reviewers' comments. This work was supported by the NSF Grant 1935329. *(Corresponding author: Michael C. Yip.)*

The authors are with the Department of Electrical & Computer Engineering, University of California, San Diego, La Jolla, CA 92093 USA (e-mail: bpwilcox@ucsd.edu; yip@ucsd.edu).

Digital Object Identifier 10.1109/LRA.2020.2974432

methods rely on an assumption of stationary data and a global distance metric. Additionally, the pre-set size of the sparse model may not appropriately handle a growing training set, as one would expect during teleoperation tasks. Local Gaussian process methods provide an approach to handle non-stationary data while maintaining low computation cost by focusing on regression in local regions of the data [10], [11]. Such models rely on partitioning the contribution of new data to a local model and performing prediction under these new local neighborhoods. Selection of neighborhood size becomes either a challenging tuning or optimization problem.

For online GPR, incremental updates to models and hyper-parameters have been proposed [10], [12], while others have suggested schemes such as forgetting rates to reduce computation. In data retrieved from trajectories with correlated paths, drifting Gaussian process models have been suggested to keep track of a sliding window of data [13]. These approaches, though low in computation cost, may be inefficient when locations in the state-space are being revisited since the regression must be performed again from scratch. However, recent work in the GPR literature has shown promising results for sparse online implementations in the streaming setting for continual learning [14], [15].

Robot teleoperation task assumes a streaming setting of data coming in sequentially of the form $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{N}$, where training pair $(\mathbf{x}_t, y_t)$ is now generally taken to be ordered in the data streams such that $(\mathbf{x}_{t+1}, y_{t+1})$ is only available after $(\mathbf{x}_t, y_t)$. In teleoperation, this entails that not only the training data, but also the next test signal, $\mathbf{x}_{test}$, comes in streams to the model for prediction. Though many works in online GP focus on tracking problems, we move our focus to active control using the model's prediction to generate the next training input-output pair. More interestingly, we are concerned with from-scratch learning, where the initial model is presumed not to cover the regions of input state-space that test signals will uncover throughout the teleoperation. Therefore, for this task, we desire an approach with the following properties: data-efficient, low memory-usage, iterative, real-time, and retains past information.

In order to handle the streaming case for teleoperation and control, we formulate a framework called Sparse Online Locally Adaptive Regression using Gaussian Processes (SOLAR–GP). The major contribution of this letter is a strategy for *online* non-parametric Bayesian model learning for *real-time* robot control. SOLAR-GP parallelizes strategies of GP sparsification, continual learning and local optimization to learn inverse models for control, leverage old models when revisiting previously explored state spaces, and adapting local models to new data (Fig. 1). Experiments are presented from various complexities of simulation environment and a real-life Baxter robot making no phenomenological assumptions. These strategies are thus useful for robots with challenging mechanics (i.e. soft-jointed or soft robots), those that are affected by environment effects, and those systems that uncertainty and noise that affect their performance; using a model-free strategy such as SOLAR–GP also means transferabilty of the framework between robot systems and applications with ease.

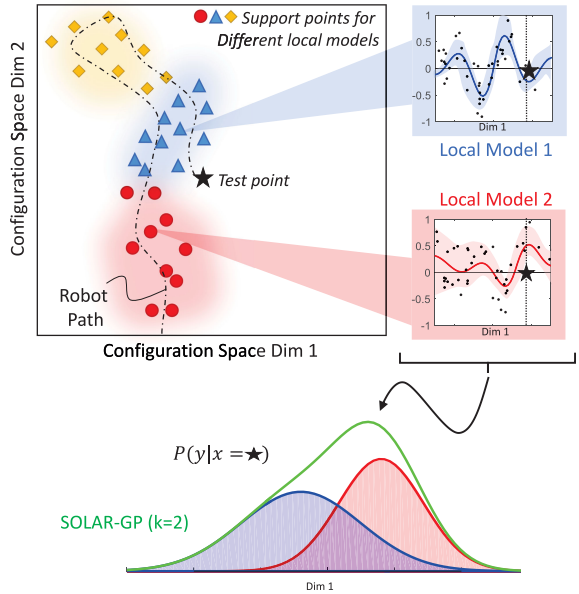


Fig. 1. SOLAR-GP is a Gaussian Process framework for efficient robot model learning and control. It generates local, streaming, sparse Gaussian Process models from input-output data during robot navigation of its configuration state-space, and uses k local models for prediction, resulting in a fast-to-compute and update mixture of Gaussians.

II. Background

SOLAR–GP builds, and modifies a set of works for computationally efficient GPs: sparsification [7], local partitioning [11], and streaming [14]. Below we describe these core works in the Gaussian Process literature.

A. Gaussian Processes

A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. In another view, GPs represent a distribution over functions, characterized by a mean function $m(\mathbf{x})$ and a covariance function $V(\mathbf{x})$:

$$\begin{aligned} f(\mathbf{x}) &\sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \\ m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned} \tag{1}$$

The mean function is the a priori expectation of the unknown function. It is often assumed as zero without any prior knowledge. The covariance function, or kernel, is typically selected by design as a measure of similarity between data points. A popular kernel choice is the squared exponential, also known as the radial basis function or Gaussian kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_s^2 \exp\left(-\tfrac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \mathbf{W}(\mathbf{x} - \mathbf{x}')\right) \tag{2}$$

In this kernel, the signal variance σ_s^2 and the characteristic length-scale or width $\mathbf{W}$ are free parameters, or hyperparameters, that can be varied (or learned) in the GP model. The characteristic length-scales (one for each input dimension), for example, affects how quickly the function values change, and thus selection of these hyperparameters is crucial to good data

fit. An $N \times N$ kernel matrix, K, is constructed from this kernel function for every pairwise point in the N training data points.

Regression in a GP amounts to inferring the conditional probability of the function output given the known data and a test point. This forms a posterior distribution of the model with the following mean and covariance:

$$m_*(x) = K_*^\top (\sigma_n^2 I + K)^{-1} y \tag{3}$$

$$V_*(x_*) = k_{**} - K_*^\top (\sigma^2 I + K)^{-1} K_* \tag{4}$$

where K is the covariance matrix, $K(X, X)$, formed by the full training data using the kernel function from (2), K_* is the covariance matrix, $K(X, x_*)$, between the test point and the full training data, k_{**} is the kernel function evaluated at the test point, $k(x_*, x_*)$, σ_n^2 is the noise variance, and y is the observed output. Here, y can be thought of as the observed underlying function values plus some noise. The noise variance of the data, σ_n^2, is another hyper-parameter to be learned in the model. As seen in (3) and (4), this regression requires the inversion of the kernel matrix, which has cubed computational complexity and is thus very costly as N becomes large.

Learning in GPs involves optimizing the hyper-parameters of the kernel with the data. This is achieved by maximizing the log marginal likelihood of the data:

$$\log p(y) = \log[\mathcal{N}(y|0, \sigma^2 I + K)] \tag{5}$$

After hyperparameter optimization in (5), prediction can be performed which robustly fits an arbitrary function. However, due to the $\mathcal{O}(N^3)$ efficiency, the standard GPR is not suitable for online implementations. To increase this efficiency, sparse methods are usually employed.

B. Sparse Gaussian Processes

Several methods in previous literature have addressed the expensive computational complexity of GP regression, of which many propose a sparse subset approach where $M \ll N$ support (or inducing) points are used to approximate the function space. If we augment the posterior distribution $p(f|y)$ with function values, u, of the inducing points, then we can obtain an augmented posterior distribution [7]:

$$p(f, u|y) = \frac{p(f|u)p(u)p(y|f)}{p(y)} \tag{6}$$

We note that the augmented joint distribution $p(f, u, y) = p(f|u)p(u)p(y|f)$, which is the same as the joint distribution of f and y when marginalized over u. In [7], Titsias proposes another distribution, $q(f, u)$, which may approximate the original posterior. To accomplish this approximation, a distribution with free variational parameters is proposed and are fitted to match the original posterior by minimizing the Kullback-Liebler divergence, $KL(q(f, u)||p(f, u|y))$. From the KL-divergence, a lower bound on the marginal likelihood may be determined such that:

$$KL(q(f,u)||p(f,u|y)) = \log p(y) + \int q(f,u) \log \frac{q(f,u)}{p(f,u,y)} dudf \tag{7}$$

The evidence lower bound, ELBO, on the log marginal likelihood is the second part of the equation. Further, [7] chooses the approximate distribution, $q(f, u)$, to be of the following form.

$$q(f, u) = p(f|u)q(u) \tag{8}$$

By replacing $q(f|u)$ with $p(f|u)$, the prior conditional distribution, the only free parameter is $q(u)$, whose optimal distribution can be analytically obtained. The lower bound, $\mathcal{F}(q(u))$, then becomes:

$$\mathcal{F}(q(u)) = \int p(f|u)q(u) \log \frac{p(u)p(y|f)}{q(u)} dudf \tag{9}$$

With the bound in terms of the variational parameter, the optimal form of $q(u)$ can be derived. Thus, given the optimal choice of $q(u)$, the bound on the log marginal likelihood is

$$\mathcal{F}(q(u)) = \log \mathcal{N}(y|0, \sigma^2 I + K_{NM} K_{MM}^{-1} K_{MN}) - \frac{1}{2\sigma^2} Tr[K_{NN} - K_{NM} K_{MM}^{-1} K_{MN}] \tag{10}$$

where K_{NM} is the kernel matrix between the M support points and N training points, and K_{MM} is the kernel matrix between the M support points. Therefore, learning and optimization under sparse GPs via variational inference equates to maximizing the bound in (10) for both the M support point locations of u and the hyper-parameters of the kernel, which reduces the computational complexity to $O(NM^2)$, allowing for low computation cost while maintaining accuracy close to the full GP as M increases.

C. Local Gaussian Processes

As described in [11], local GP models are separate Gaussian models defined by local partitions of the training data into clusters, each defined by a model center location. Data is separated into the closest local model until a point exceeds a threshold distance, w_{gen}, at which point a new model is created. Prediction of a query point becomes a weighted average of the local models according to the distance of the query point to each model center.

A distance metric, w_k, is defined between the location of each data point $\mathbf{x}$ and the center of each kth local model, $\mathbf{c}_k$:

$$w_k = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_k)^\top \mathbf{W} (\mathbf{x} - \mathbf{c}_k)\right) \tag{11}$$

where $\mathbf{W}$ is the kernel width parameter that is learned from the squared exponential kernel. After the addition of each new data point to a local model, the model center is recomputed as the mean of all the data in its local partition.

After the data has been partitioned, the kernel matrix is updated for each local GP model. In order to deal with the ever-growing number of training examples, a limit on the number of data points in each local model is enforced. Additionally, [16] proposes an incremental update to the covariance matrix and prediction vector via updating the Cholesky decomposition of the Gram matrix. Regression and prediction under these local models is a weighted average of each local model's predicted mean, y_k, for a query point with the distance metric w_k of that point from each model center. Optimization of the hyper-parameters of the kernel function is typically done on a subsample of the the full training data.

D. Streaming Gaussian Processes

While other GP formulations may naively append new data and retrain, [14] proposed a framework to eloquently leverage previously trained model information into the next sequential updates for the streaming data case.

In the streaming setting where data arrives sequentially, new data $(\mathbf{x}_{new}, \mathbf{y}_{new})$ will be added to the old data, $(\mathbf{x}_{old}, \mathbf{y}_{old})$. Via variational inference, assuming only direct access to the new data, the old data must be encoded into the previous posterior distribution. The old posterior approximation, $q_{old}(f) \approx p(f|\mathbf{y}_{old})$, and the new posterior approximation, $q_{new}(f) \approx p(f|\mathbf{y}_{old}, \mathbf{y}_{new})$, are of the forms

$$q_{old}(f) = \frac{1}{\mathcal{Z}_1(\theta_{old})} p(f|\theta_{old}) p(\mathbf{y}_{old}|f) \tag{12}$$

$$q_{new}(f) = \frac{1}{\mathcal{Z}_2(\theta_{new})} p(f|\theta_{new}) p(\mathbf{y}_{old}|f) p(\mathbf{y}_{new}|f), \tag{13}$$

where the exact posterior distributions are parameterized by the kernel hyperparameters, θ. Here we recognize that though the new exact posterior, $p(f|\mathbf{y}_{old}, \mathbf{y}_{new})$, would incorporate the new data, they must find an approximation for $p(\mathbf{y}_{old}|f)$ since it cannot be obtained directly. By approximating as $p(\mathbf{y}_{old}|f) \approx \mathcal{Z}_1(\theta_{old}) q_{old}(f)/p(f|\theta_{old})$, we have

$$p(f|\mathbf{y}_{old}, \mathbf{y}_{new}) = \frac{\mathcal{Z}_1(\theta_{old})}{\mathcal{Z}_2(\theta_{new})} p(f|\theta_{new}) p(\mathbf{y}_{new}|f) \frac{q_{old}(f)}{p(f|\mathbf{y}_{old})} \tag{14}$$

However, $p(f|\mathbf{y}_{old}, \mathbf{y}_{new})$ is intractable, thus [14] introduces the variational distribution using inducing point locations which are allowed to change after each step. The previous approximate posterior then becomes $q_{old}(f) = p(f_{\neq \mathbf{a}}|\mathbf{a}, \theta_{old}) q_{old}(\mathbf{a})$ with $\mathbf{a} = f(\mathbf{z}_{old})$ evaluated at the previous $\mathbf{z}$ inducing point locations, representing the approximate distribution of the latent function given the induced function values at their respective input locations. Likewise of similar form, the new approximate posterior distribution is of the form $q_{new}(f) = p(f_{\neq \mathbf{b}}|\mathbf{b}, \theta_{new}) q_{new}(\mathbf{b})$, where $\mathbf{b} = f(\mathbf{z}_{new})$. With the variational parameter, $q_{new}(\mathbf{b})$, from the new approximate posterior distribution, $q_{new}(f)$, a new lower bound on the log marginal likelihood can be found from the KL-divergence,

$$KL[q_{new}(f)||p(f|\mathbf{y}_{old}, \mathbf{y}_{new})] = \log \frac{\mathcal{Z}_1(\theta_{old})}{\mathcal{Z}_2(\theta_{new})} + \int df q_{new}(f) \left[\log \frac{p(\mathbf{a}|\theta_{old}) q_{new}(\mathbf{b})}{p(\mathbf{b}|\theta_{new}) q_{old}(\mathbf{a}) p(\mathbf{y}_{new}|f)} \right] \tag{15}$$

Note that the previous approximate posterior distribution is assumed to be $q_{old}(\mathbf{a}) = \mathcal{N}(\mathbf{a}|\mathbf{m}_\mathbf{a}, \mathbf{S}_\mathbf{a})$. This implies that it can be resolved just by the previous inducing point locations and the kernel hyperparameters.

III. SOLAR–GP: Sparse Online Locally Adaptive Regression Using Gaussian Processes

SOLAR–GP comprises a model initialize, a model partition function, a predictor/generator, a trainer, and the tester (i.e. operator). A block diagram is shown in Fig. 2.

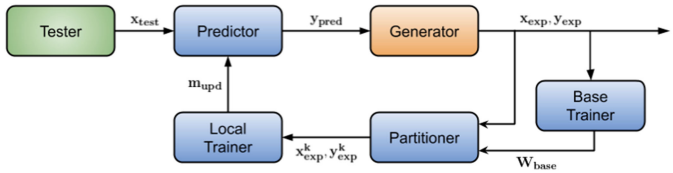


Fig. 2. Block diagram of control flow for SOLAR–GP. The forward pass operates at the max control loop rate, while the feedback path may be at slower speeds (e.g. as-fast-as-possible) using parallelization.

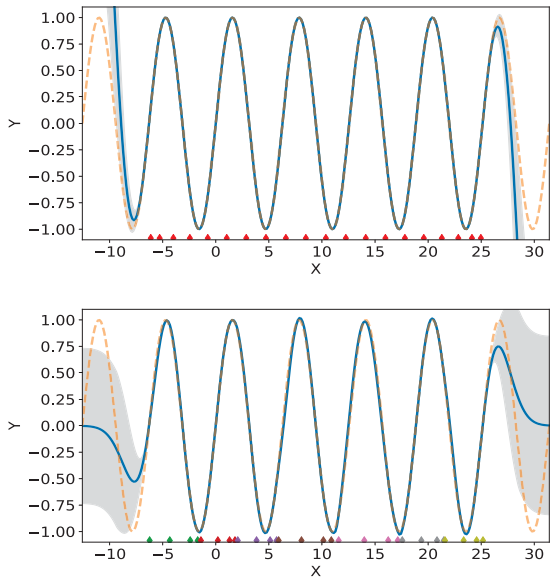


Fig. 3. **Top**: Sparse GP fit on sin(x). Red marks represent inducing point locations. **bottom**: SOLAR–GP fit on sin(x). Different colored marks represent inducing points for unique local models.

1) Model Initialization: Before regression, a sparse GP model with N_{init} points is initialized. For robot teleoperation, this initialization can come from random jitters of the robot's joints or from some sampled, idealized model of the system. Then, variational inference [16] may be used to train a sparse local GP of size M, where $M < N_{init}$.

2) Local Model Partitioning: Describing a local model partition scheme requires a distance metric to determine when a new model should be generated given new training inputs. Local GPR [11] computes this distance metric for each model center using (11). Since in the streaming-from-scratch context such as teleoperation we assume no initial access to a reasonable subsample of the taskspace, SOLAR–GP continuously computes the kernel width, $\mathbf{W}_{base}$, computed from a separately trained base model, a single streaming sparse model which incorporates new data into its model across the entire task state-space.

After finding the nearest model or following a new local model initialization, the new training pair is incorporated into the nearest model. In contrast to the previous work's strategy of limiting memory complexity by retaining no more than a fixed number of local model data points before deleting points, SOLAR–GP leverages the sparse streaming model [14] as an analogue to the information gain strategy, which enables updates with only the inducing points of the prior sparse model and the new streaming data.

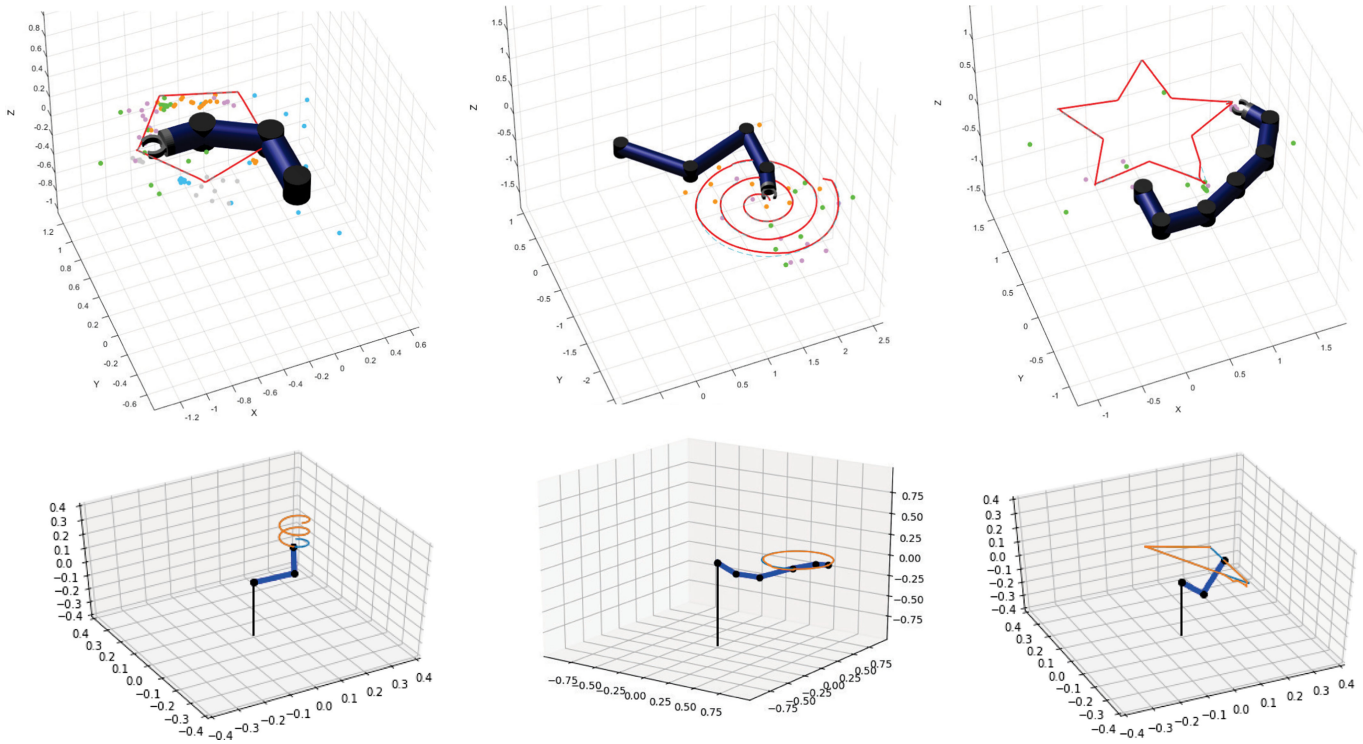


Fig. 4. Traces of a simulated planar 3, 4, 6 DOF manipulator arm recreating a pentagon, spiral, and star trajectory, as well as 3, 6, and 3 DOF 3D serial link arms recreating a helix, circle, and rectangle from simulated teleoperation test points. Red (top row) and orange (bottom row) show the end effector trace and blue shows the commanded trajectories. The colored dots on the image (top row) represent locations of the support points optimized from the SOLAR–GP.

3) Training: Training occurs online in order to keep the local models up-to-date with the most recent training data. After a new training experience pair $(\mathbf{x}_{new}, y_{new})$ is received, two updates occur. First, the training pair is added to the base model and trained via streaming GP optimization [15] via variational inference. This step updates the kernel width to be used for partitioning the data into its respective local models as well as for prediction. Then, once the base model is trained and the experience pair partitioned into a local model, m_k, it is trained, optimizing the kernel hyperparameters and inducing point locations. For training, the model is seeded with the previous m_k model, taking advantage of its past state. Typically, the M inducing inputs are initialized with those from the previous model; however, incorporating new data as part of the initial support points can help bias towards newer data.

For new model initialization, we seed it with the current base model and allow the inducing inputs to quickly bias to newer points. By setting the max number of inducing points for each local model, the complexity of each model is fixed even as regression persists for an indefinite amount of time. Whereas the training complexity for sparse models is $\mathcal{O}(nM^2)$, under the local model scheme the cost for training one local model becomes $\mathcal{O}(n_k M_k^2)$. If we consider n training examples partitioned equally into K local models, $n_k = n/K > 1$, with $M_k = M/K > 1$ inducing inputs, the overall complexity of training the local models is lower (since $\mathcal{O}(K n_k M_k^2) = \mathcal{O}(n\frac{M^2}{K^2})$) than in the batch case.

4) Prediction: A prediction over a set of local K models for a test input becomes a Mixture of Gaussians $g(\mathbf{x})$,

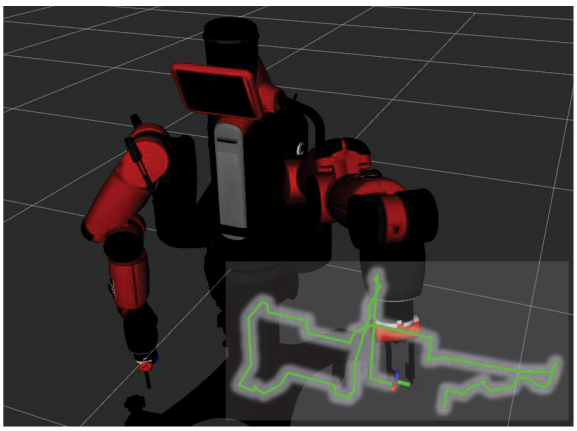
$$g(\mathbf{x}) = \sum_{i=0}^{K-1} \mathcal{N}(\mu_i, \Sigma_i) = \sum_{i=0}^{K-1} \frac{\exp\left(-\frac{1}{2}(x - c_i)^\top V_i^{-1}(x - c_i)\right)}{\sqrt{(2\pi)^d |V_i|}} \tag{16}$$

where $c_i, V_i \sim GP_i$ and d is the state-space dimension. For control, one may calculate the expected values of the first and second moments where $\mathbb{E}[g(x)] = \int_{-\infty}^{\infty} x g(x) dx$, though in practice the integral is computationally intractable. The simplest method may be to use the nearest local model. Alternatively, a weighted prediction on test inputs is used for the local models, where the kernel width is taken from the trained base model when computing the distance metric from the test input, x_{test}, to each model's center, c_k. The weighted predictions are then:

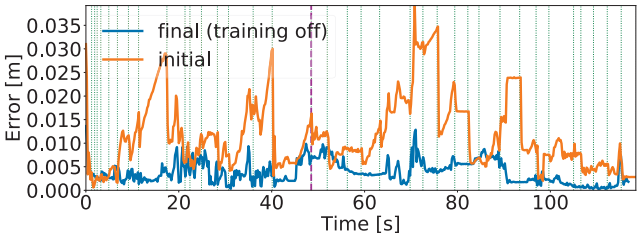
$$y_{pred} = \frac{\sum_{k=1}^{K} w_k y_k e^{-V_k}}{\sum_{k=1}^{K} w_k e^{-V_k}} \tag{17}$$

The variance of each local model's prediction is used as an additional, independent weight. This modification holds the advantage of weighting the contribution of each local model with not only its proximity to the queried test signal but also the uncertainty associated with each model.

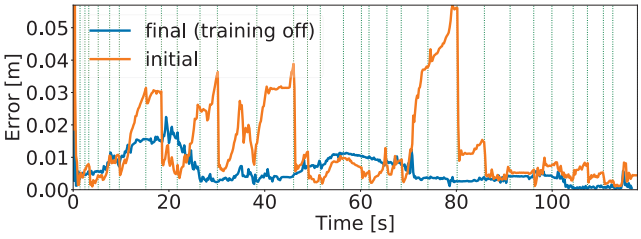
In contrast to other GP variants, now prediction retain high predictive power due to use of local models and training data



(a)



(b)



(c)

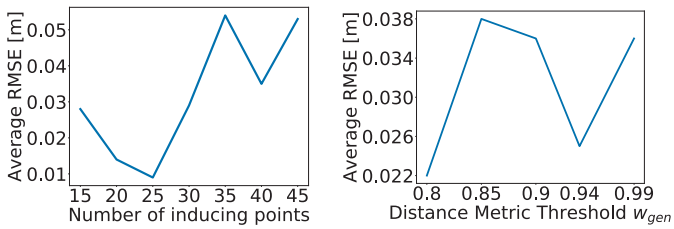
Fig. 5. (a) Test path for Baxter simulation covering large workspace for its left arm (b) Plot of path error $[m]$ vs time for SOLAR–GP with 25 inducing inputs per local model. Green vertical lines represent locations of training updates. The magenta vertical line indicates that a new local model has been added. (c) Plot of path error $[m]$ vs time for Streaming Sparse GP with 40 inducing inputs. The initial (orange) trajectory path errors and final (blue) after 3 training passes are shown for each plot.

partitioning, while having low computational complexity due to the local models being sparse.

5) Tester: The tester represents the streaming, commanded trajectory of the system, e.g. a teleoperator that comes following model initialization.

A. Online Robot Teleoperation Using ROS-Enabled SOLAR-GP

Per our model, we can describe the control flow of the algorithm as Tester, Predictor, Generator, Partitioner, and Trainer (base and local), where for a robot, the Generator is analogous to the robot's controller and sensing, which is responsible for taking a prediction and generating its relevant experience. For online robot teleoperation, we formulate the problem for parallel implementation using the Robot Operating System (ROS).



(a) (b)

Fig. 6. (a) Plot of number of inducing points vs average RMSE, (b) Plot of number of w_{gen} vs average RMSE.

Because local model partitioning, generation, and optimization are slower processes than affordable within a robot control loop (which requires only the predictor), parallelization of all the above models are implemented in SOLAR–GP, and is necessary for real-time teleoperation. Under a parallel framework, prediction of robot joint angles for joint position control can be consistently computed at a higher frequency while data collection and training can occur at a different frequency. Since it is expected that the robot moves and experiences regions of the state-space even while the Trainer is still optimizing the model on the previous points, an experience buffer node accumulates the experiences of the robot. Utilizing the ROS node graph and messaging for asynchronous input/output streams, we can implement the SOLAR–GP algorithm across separate nodes with independent, as-fast-as-possible loop rates and allow them to run in parallel.

IV. Experiments and Results

We tested SOLAR–GP on sample n-link robots in the sequential position trajectory tracking as well as demonstrate the capability of our method to run live teleoperation on both a simulated and real Baxter robot, without using its kinematic model for inverse kinematics mapping. We leverage the GPy library for its GP data structures and optimization solutions [17]; we also adapted the streaming Gaussian Process method from [14] to work within the GPy framework.

A. Toy Example

As an initial demonstration of the SOLAR–GP algorithm, we employed training and prediction on a simple toy problem in Fig. 3, a sinusoidal wave, and compare it to the sparse Gaussian process method from Titsias [7].

When the input space grows, as in the simple sinusoidal series data, more inducing inputs or support points are required to approximate the full model posterior distribution. Though one can select a larger number of support points for the data, coverage is bound by this predetermined limit, and the regression erodes. With SOLAR–GP, new models are produced to cover growing input spaces, allowing for each local model's inducing point limit to be reduced, an advantage for learning in continual, growing or undetermined input spaces (i.e. a robot exploring its configuration space).

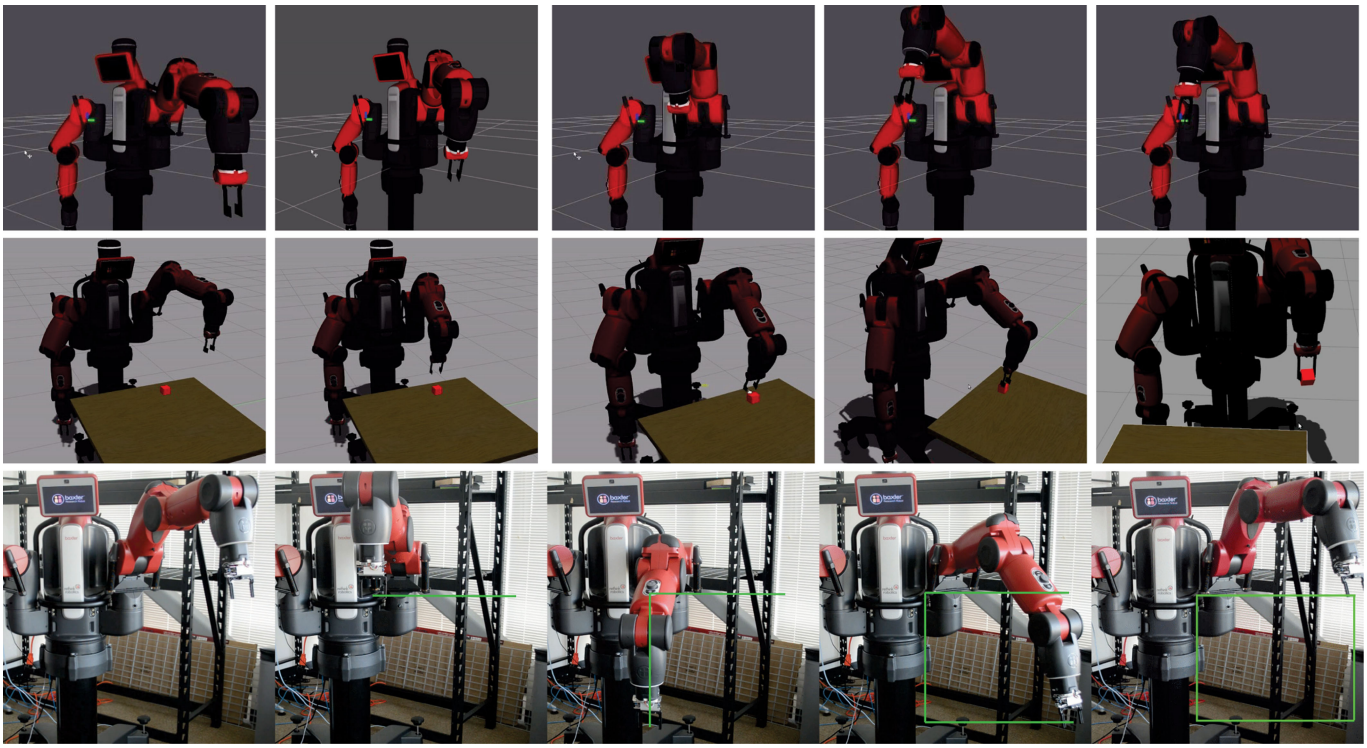


Fig. 7. Motion path progression of Baxter arm performing real-time teleoperation demos for a simulated query point (*top row*), simulated pickup task (*middle row*), and a rectangular path (*bottom row*) on the real Baxter.

TABLE I
RMSE FOR n-LINK ROBOT TELEOPERATION EXPERIMENTS

	pent	spiral	star	helix	circle	triangle
n	3	4	6	3	6	3
rmse [m]	0.0045	0.0077	0.0084	0.0068	0.0086	0.0091
# models	5	4	2	3	4	3
# sup pts	25	25	25	30	40	30

B. Sequential Robot Teleoperation for n-Link Manipulators

In these experiments, an inverse kinematic learning problem for n-link manipulators are presented, where the end-effector position represents the $\mathbf{x}$ input state, and the joint positions represent the output, $\mathbf{y}$. No geometric information is provided to SOLAR–GP and test signals arrive sequentially. Fig. 4 shows the resultant paths of concurrent model learning and online prediction on three planar robots with 3, 4, and 6 links. Support points for each local model cover the path as the task space is explored. Furthermore, trajectories for 3- and 6-link manipulators in 3D space is also demonstrated. Tracking errors are shown in Table I.

C. Teleoperation Control on a 7-DOF Robot Arm

The following describe several demonstrations using SOLAR-GP for teleoperation on a 7-DOF robot manipulator on the Baxter Robot (Rethink Robotics). Just as in the previous n-link manipulator experiments, we are learning the inverse kinematics for Baxter's arm.

1) Baxter Simulator Teleoperation: In these experiments, we perform real-time teleoperation on Baxter using a Microsoft Xbox controller. We use ROS to support parallel teleoperation, prediction, and training. Joint positions are determined from the SOLAR–GP model and sent to Baxter's position controller. As opposed to the sequential n-link experiments, the Baxter robot used a parallelized version of SOLAR–GP. In our tests, the position of the end effector was commanded to track a teleoperated input from an Xbox controller. The prediction loop ran at a publish rate of 100 Hz, while the training loop, dominated by the model optimization time, effectively operated between 0.3–2 Hz.

Fig. 5(a) shows a trajectory commanded via teleoperation. In order to demonstrate improved performance after learning the path, the trajectory is looped two times after the first pass while still training online followed by a final pass with training disabled. Fig. 5(b) shows a run of SOLAR-GP with 25 inducing points per local model with the absolute errors of the path on both the first pass through the trajectory and on the final pass after training. We compare to an example of the streaming sparse method [14] alone with 40 inducing points in Fig. 5(c), where no local models have been generated during the entire run. We generally found better performance with 40 inducing points than at greater numbers, where the additional training computation time may affect performance in this parallel training/prediction setting.

The first pass in Fig. 5(c) shows less than a 1 cm root-mean-squared (RMS) deviation from the desired position on average, with maximum 3 cm deviation before recovering. Peak performance of 4.4 mm was reached on the final trial after training updates stopped. In the Streaming Sparse GP run, it had an average RMS of 2.6 cm on the first pass, with a maximum deviation of about 6 cm. On its final pass, the peak performance reached

7.7 mm RMS. Though the Streaming Sparse GP algorithm by itself performed well, SOLAR–GP was able to out-perform the Sparse GP method.

Fig. 6 shows a summary of 140 test runs while varying key hyperparameters to show sensitivity to the parameters. A suitable range of hyperparameters to test were chosen after moving the robot manipulator over a large range of workspace and selecting the number of inducing points and the model generation distance threshold that fit well to the arm for performance during the tests. Using 25 inducing points had the lowest average RMSE on the test trajectory, with $w_{gen} = 0.8$ being the best on average distance threshold. Though one might expect a positive trend with more inducing points and more models created to resolve in lower $RMSE$, in the streaming online prediction, there are other factors. Since the robot generates experience from predictions, the training data will be partitioned based on how close the actual pose is to the model centers. The partitions are thus biased from the initial model created from a random jitter as well as on delayed training updates which may lag good predictions.

Fig. 7 shows a set of experiments with the Baxter system, initialized with jitter and then commanded. Setpoint control (top row) is where the SOLAR–GP model estimates the inverse kinematic solution repeatedly as it minimizes its error to the setpoint. It shows rapid convergence to the goal with minimal deviation from a straight-line path. Robot pick-up (middle row) shows a similar learning pattern but for a sequence of setpoints, with a grasp in-between to pick up the block. Finally, live teleoperation (bottom row) shows a Baxter being fed a live, interpolated sequence of setpoints to create a smooth square trajectory.

V. Conclusion

In this letter, we present SOLAR–GP, which combines the strengths of Local Gaussian Process Regression with those of Streaming Sparse Gaussian Processes in a principled manner. As a result, SOLAR–GP can learn continually with a lower fixed computational complexity and memory storage of either method alone and retain info on the whole work space through sparse local models. Experiments and demonstrations on n-link robots showed the versatility of the method for online prediction, training, and control, and several experiments on a Baxter robot highlighted the real-time performance using parallel training and predictions.

Our method maintains a competitive edge since training and predictions are done per local model's induced parameters instead of over the entire set of inducing points. Thus, since we can maintain $M_{local} < M_{batch}$, our method can be more flexible to cover larger areas of the input state space without incurring larger penalties in complexity.

For online robot control, SOLAR–GP performs like open loop control with adaptive parameters. Since these "parameter" updates occur after the robot has generated new experience for the next batch, the robot's estimate approaches the truth when more data closer to the target is trained, learning until the predictions converge. Thus, we see in Fig. 5 that even after the resultant pose from a joint prediction is far from the desired pose, that new experience pushes the model's next prediction closer to the truth. An advantage of the parallel use-case in ROS is that all intermediate points can be buffered for training while the algorithm continuously makes predictions.

An immediate area of interest for future work on the SOLAR–GP method lies in optimizing the weighted distance threshold, w_{gen}, for a learning setting, since the choice of w_{gen} affects when new models are created. Additionally, though SOLAR–GP maintains low computational complexity and memory, there is no implicit regulation on the number of local models that are created throughout a learning session. It is inherent that for very large input spaces, many models are possibly produced. Future work may consider how to redistribute models, potentially combining and re-partitioning as needed to satisfy both computational constraints and some optimality criterion.

References

[1] M. C. Yip and D. B. Camarillo, "Model-less feedback control of continuum manipulators in constrained environments," *IEEE Trans. Robot.*, vol. 30, no. 4, pp. 880–889, Aug. 2014.

[2] J. Peters, D. D. Lee, J. Kober, D. Nguyen-Tuong, J. A. Bagnell, and S. Schaal, "Robot learning," in *Springer Handbook of Robotics*. Berlin, Germany: Springer, 2016, pp. 357–398.

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

[4] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*, vol. 199. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[5] R. C. E. and C. K. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: The MIT Press, 2006.

[6] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," in *Proc. Advances Neural Inf. Process. Syst.*, vol. 18, 2005, pp. 1257–1264.

[7] M. K. Titsias, "Variational learning of inducing variables in sparse Gaussian processes," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2009, pp. 567–574.

[8] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," in *Proc. Conf. Uncertainty Artif. Intell.*, 2013, pp. 282–290.

[9] L. Csato and M. Opper, "Sparse on-line Gaussian processes," *Neural Comput.*, vol. 14, no. 3, pp. 614–68, 2002.

[10] F. Meier, P. Hennig, and S. Schaal, "Incremental local Gaussian regression," in *Proc. Advances Neural Inf. Process. Syst.*, 2014, pp. 972–980.

[11] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Local Gaussian process regression for real time online model learning and control," in *Proc. Advances Neural Inf. Process. Syst.*, 2008, pp. 1193–1200.

[12] C.-A. Cheng and B. Boots, "Incremental variational sparse Gaussian process regression," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 4410–4418.

[13] F. Meier and S. Schaal, "Drifting Gaussian processes with varying neighborhood sizes for online model learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2016.

[14] T. D. Bui, C. V. Nguyen, and R. E. Turner, "Streaming sparse Gaussian process approximations," in *Advances Neural Inf. Process. Syst.*, 2017, pp. 264–269.

[15] T. D. Bui, C. V. Nguyen, and R. E. Turner, "Partitioned variational inference: A unified framework encompassing federated and continual learning," Nov. 2018, *arXiv:1811.11206*.

[16] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Model learning with local Gaussian process regression," *Adv. Robot.*, vol. 23, no. 15, pp. 2015–2034, 2009.

[17] S. M. L. Group, "Sheffieldml/gpy: Gaussian processes framework in python," 2012. [Online]. Avilable: https://github.com/SheffieldML/GPy. Accessed on: Mar. 15, 2019.