Online Model Learning and Control from Streaming Data using Gaussian Processes for Robot Teleoperation

Brian Wilcox, University of California at San Diego, MS, Electrical and Computer Engineering
bpwilcox@ucsd.edu

***Abstract*— Machine learning methods have been explored more recently for robotic control, though learning the inverse mapping from sensor outputs to joint inputs in real time remains challenge for real-time control. In the task of teleoperation, commands and sensor data are received in sequential streams, thus, in a data-driven controller, the robot model should learn and adapt to these streams of data, while also making accurate predictions about the desired motor joint inputs at pace with the operator. This paper aims to achieve this goal of online model-learning and control for a teleoperation task by using Sparse Online Local Gaussian Process Regression (SOLGPR) to infer a local function mapping of the robot sensor states to joint states and perform a prediction of the teleoperation command for joint control. Local Gaussian process models are learned and sparsified via Variational Inference with user-tuned bounds on model size and complexity. An optimization scheme involving periodic optimization of a drifting Gaussian process helps to reduce computation time in conjunction with the sparse local models. This framework provides a basis for a tradeoff between model complexity and performance.**

I. Introduction

Robot modeling and control, though a well studied field, grows in challenge as systems become more and more complex. From humanoid robots, surgical robots, to terrain robots, the task of modeling systems with high dimensionality, unknown environment affects, nonlinearities, and adaptive behavior becomes ever more difficult, and furthermore, leads to model inaccuracies. For control, low model fidelity puts extra effort on the controller. Machine learning methods have been explored more recently for robotic control, though learning the inverse mapping from sensor outputs to joint inputs remains a challenge for real-time control [8]. In the task of teleoperation, commands and sensor data are received in sequential streams, thus, in a data-driven controller, the robot model should learn and adapt to these streams of data, while also making accurate predictions about the desired motor joint inputs at pace with the operator. To this end, work in model-learning for control needs to focus on low computational-complexity and learning time while maintaining high accuracy appropriate for the task environment.

In this paper, we introduce an approach for learning the inverse kinematics or dynamics of a robot manipulator in real-time, and producing joint input predictions for control. The framework is formulated to handle data incoming in streams from sensors and desired commands from a teleoperator. The main contribution of this work is in formulating a trade-off between sparsification strategies and local regression using Sparse Online Local Gaussian Process Regression for the use in teleoperation tasks.

In section II, we will discuss related work. Relevant method are described in Section III. Results are presented from a simulation environment in Section IV with a discussion of this work in Section V.

II. Related Work

Gaussian process regression (GPR) has shown promise in modeling the inverse dynamics of robotic systems [4]. Its intuitive structure and probabilistic framework allow for the uncertainties associated with model regression and prediction to be explicitly expressed. GPR enables robust training of hyper-parameters to fit data of growing size, however, the computational cost of training and prediction is inefficient for real-time control. Sparse Gaussian Processes reduce the computational complexity of regression (and optimization), dominated by an inversion of the covariance matrix on the order of $O(N^3)$, where N is the number of training data points [5]. By selecting $M << N$ support points, this complexity can be reduced to $O(NM^2)$, with stochastic methods reducing even further. Various sparse methods for Gaussian processes have been introduced in the literature, with large focus on selection of the support points. Online sparse methods exist, which take advantage of incremental updates to the model and training while forgetting old data [3]. However, global sparse 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. 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 [1][2]. Such models rely on partitioning the contribution of new data to a local model and performing prediction under these new local neighborhoods. Like in sparse methods, selection of neighborhood size becomes either a challenging tuning or optimization problem.

For online implementation of GPR, many methods have utilized incremental updates to models and hyper-parameters [1][6], 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 at a time [7]. These approaches, though low in computation cost, may be inefficient when paths are being revisited since the regression must be performed again from scratch. We aim to achieve online performance

of model learning by utilizing local Gaussian process models in conjunction with sparse methods to allow for a growing workspace.

III. Methods

In this work, we utilized Gaussian process regression to train and learn an inverse mapping of robot inverse kinematics (or dynamics). Particularly local models are created and used for a weighted prediction, while these models are sparsified via variational inference to allow for a tradeoff between model size and performance for online learning and control. An optimization schedule using a drifting Gaussian process is also introduced to lower computation cost and accomodate non-stationary modeling of data along a trajectory.

A. Gaussian Process Regression

Gaussian processes represent a distribution over functions, characterized by a mean function and a covariance function:

$$f(x) \sim GP(m(x), k(x, x')) \tag{1}$$

The mean function $m(x)$ is the a priori expectation of the unknown function. It is often assumed as zero without any prior knowledge. The covariance function $k(x, x')$, or kernel, is typically selected by design as a measure of similarity between data points. The most common kernel, especially in robotics is the squared exponential kernel, also known as the radial basis function or Gaussian kernel.

$$k(x, x') = \sigma_s^2 exp(-\frac{1}{2}(x - x')^T W(x - x')) \tag{2}$$

In this kernel, the signal variance σ_s^2 and the charateristic length scale or width W are hyperparameters to be learned for the Gaussian process model. 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 Gaussian process 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:

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

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

Where K is the kernel matrix of the N training input points, $k(X, X)$, K_* is the kernel matrix between the test point and the full training data, $k(X, x_*)$, 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 can be costly as N becomes very large. Learning in a Gaussian processes involves optimizing the hyper-parameters of the kernel with the data. This is achieved by maximizing the log marginal likelihood of the data:

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

After optimization in (5) for the hyperparameters of the GP model, model prediction can be performed which robustly fits an arbitrary function. However, due to the $O(N^3)$ efficiency, the standard GPR isn't suitable for online implementations. To increase this efficiency, sparse methods are usually employed.

B. Sparse Variational Inference

One popular approach toward sparse methods for Gaussian processes is to select a subset of points from the full training data to represent the model. We want to choose $M << N$ support points which will provide a sufficient representation of the full data. Selection of these M support points can be done by Variational Inference, which maximizes the similarity between an induced posterior distribution of the Gaussian process and the original posterior distribution via the Kullback-Leibler divergence. In [9], the hidden function of these M support points, f_m, is introduced, and an approximate posterior distribution $q(f, f_m)$, is induced, where

$$q(f, f_m) = p(f|f_m)q(f_m) \tag{6}$$

Here, $q(f, f_m)$ is an approximation of the true augmented posterior distribution, $p(f, f_m|y)$, and $q(f_m)$ is chosen to be a free variational Gaussian distribution such that

$$q(f_m) \sim p(f_m|y) \tag{7}$$

but in general $q(f_m) \neq p(f_m|y)$.

The aim in selecting the M support points then is to minimize the KL-divergence, $KL(q(f, f_m)||p(f, f_m|y))$, of the induced posterior distribution and the augmented true posterior distribution. This is achieved by maximizing the variational lower bound of the true marginal log likelihood as follows:

$$\begin{aligned} L_{bound} &= \int q(f, f_m) log \frac{p(y|f)(p(f|f_m)p(f_m)}{q(f, f_m)} df df_m \\ &= log\mathcal{N}(y|0, \sigma^2 I + K_{NM} K_{MM}^{-1} K_{MN}) \\ &\quad - \frac{1}{2\sigma^2} tr[K_{NN} - K_{NM} K_{MM}^{-1} K_{MN}] \end{aligned} \tag{8}$$

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 Gaussian processes via variational inference equates to maximizing the bound in (8) for both the M support point locations and the hyper-parameters of the kernel. As seen in (8), now the operation requires the inversion of an $M \times M$ kernel matrix of the induced points, which reduces the computational complexity to $O(NM^2)$. Using this sparse method picks optimal support locations to represent the full training data, which allows for low computation cost while maintaining accuracy close to the full GP as M increases.

C. Local Gaussian Processes

As described in [2], local Gaussian process 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, 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 x and the center of each local model, c_k, as follows:

$$w_k = exp(-\frac{1}{2}(x - c_k)^T W (x - c_k)) \tag{9}$$

where W is the width parameter that is learned from the squared exponential kernel. The key idea behind partitioning the training data is that the model with the closest center to any training point will update this data point to its local region. 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. A threshold, w_{gen}, is created, such that when the maximum w_k of a training point is greater than this threshold, i.e. $max(w_k) > w_{gen}$, a new local region is created with this point as the new model center. Thus, the number of local models, L, can increase as the workspace is explored.

After the data has been partitioned, the kernel matrix is updated for each local Gaussian process model. 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 as follows:

$$y_p = \frac{\sum_{k=1}^{M} w_k y_k}{\sum_{k=1}^{M} w_k} \tag{10}$$

The predictions of each local model is performed as in (3). Local models closer to the query point have a larger contribution to the overall predicted mean, y.

The regression using local GP models is more tractable than the standard GP because the operation is performed over smaller local regions rather than the entire batch of data. Furthermore, optimization of the hyper-parameters is typically done on a subsample of the the full training data.

D. Sparse Online Local Gaussian Process Regression

In this work we combine the methods presented above into a framework suitable for online model-learning for robot control, called Sparse Online Local Gaussian Process Regression (SOLGPR). In a teleoperation task, we assume that data is incoming sequentially to the model-learning process, both from sensors on the robot as well as the command signals from the teleoperator. This entails that in a single loop, new data points are updated to the current model, model parameters are learned, and prediction on the teleoperation signal is performed.

SOLGPR handles streaming data by first partitioning each training data input/output pair (x_i, y_i) into local regions via (9). These local regions are then trained using sparse variational inference for GP to select M inducing points for each local model. Predictions of the function output y_p are made from the command signal x_{tel} with the trained local models according to a modified version of (10) as follows:

$$y_p = \frac{\sum_{k=1}^{M} w_k y_k e^{-V_k}}{\sum_{k=1}^{M} w_k e^{-V_k}} \tag{11}$$

Where V_k represents the variance computed at each local model's prediction. This modification holds the advantage of weighting the contribution of each local model with not only its proximity to the queried teleoperation signal but also the uncertainty associated with each model. Ideally, both close and confident models will have the largest contribution to the overall prediction. In practice, only the top two or three best local models are used for prediction.

The prediction, y_{p+1}, is used as the control input for robot controller (i.e. joint torques, velocity, or angle). The resulting state of the robot, x_{i+1}, becomes the new experience along with the control prediction (or the measured control output) for the next loop.

To manage the size of the growing neighborhoods of the local models, a threshold of K local points is set as an upper bound of each region's size. After K local points of a model have been partitioned, a random data point (or oldest point) is deleted from the region when a new point is updated. For optimization, we take inspiration form [7], and employ an optimization schedule using a drifting Gaussian process. A drift parameter, D, selects how many data points to use in this drifting GP, and the optimization procedure is scheduled at set intervals. Below is a summary of the procedure for SOLGPR on a teleoperation task.

Summary of SOLGPR procedure:

1) **Initialize:** jitter robot to create initial GP models
2) **Teleoperate:** receive current command signal
3) **Control:** predict joint command and control robot
4) **Experience:** receive current robot state
5) **Partition:** Partition new experience into local regions
6) **Train:** Update and train sparse local models
7) **Learn:** Optimize drifting GP
8) **Loop:** Repeat 2-7

Under this procedure, SOLGPR enables the learning and control of a robot manipulator when the data arrives sequentially.

IV. Evaluation

A. Experimental Setup

The SOLGPR method in this paper was evaluated using a simulated teleoperation on a planar two-link robot manipulator. In this case, the task was to learn the inverse kinematics of a path generated by a teleoperator. The GPy library (Python) by the Sheffield Machine Learning Group was used for the creation and optimization of Gaussian

process models [10]. This library also provides tools for performing sparse variational inference on GP models. The Robot Operating System (ROS) was utilized for the the parallel node structure for streaming data and the simulation of a two link arm, visualized in $rviz$. Teleoperation paths were generated offline and fed to the SOLGPR controller sequentially.

B. Initial Test

In the first initial test of the SOLGPR algorithm, prediction was performed in Python without a simulation of a two-link arm in ROS, using the forward kinematics to compute the resultant 2D position. of the robot end effector.. Figure 1 shows the results of prediction on a circlular path. The teleoperation signals into the controller are the 2D desired position of the end effector. In every loop, the training data consists of the sensed 2D position of the robot end effector paired with the joint positions of the robot which produce that end position. In this case, a smaller trajectory (about 100) points was tested, with no need to sparsify the local models. The proximity threshold, w_{gen}, was set as 0.975.

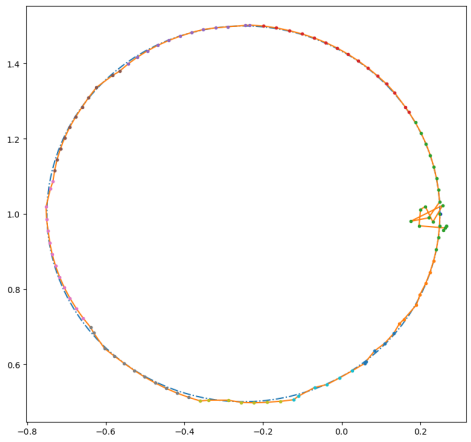

Fig. 1. Trace of end effector path produced by the joint angle predictions from the SOLGPR algorithm. Different color dots represent a separate local GP model. The procedure begins by initializing the GP model with a random jitter of the robot joint angles.

C. Teleoperation Simulation

For the simulation of a teleoperation task, a ROS node sampled a new point from a predefined trajectory in each iteration of the SOLGPR algorithm. Following the procedure outlined in Section III, subsection D, online regression and control was performed on a two-link manipulator. The schematic of ROS node communication is shown in Figure 2, whereby the controller node for the SOLGPR sends the predicted joint angle to the robot state publisher to be simulated and visualized in $rviz$.

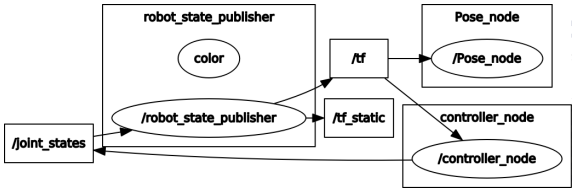

Fig. 2. ROS node graph showing communication scheme for online model-learning and control. The tf output represents the end effector transformation to simulate the sensor data input into the SOLGPR controller as experience

The number of inducing points for the sparse models was set to be 15 for the circle trajectory, while the upper bound on the K local points of each local model was capped at 100. A drift parameter D of 10 training points was set for the drifting Gaussian process used for optimization and scheduled to fit hyper-parameters every 10-20 iterations. Again, the proximity threshold, w_{gen}, was set as 0.975. Figure 3 shows an image of the robot manipulator learning to draw a circle.

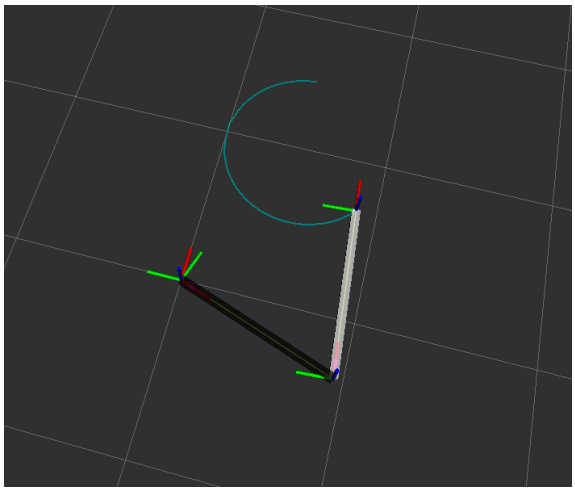

Fig. 3. Image of two-link manipulator tracing a circle in $Rviz$. The path was sampled from a predefined trajectory to be presented to the controller in real-time in a sequential update.

As a test of the computation cost as the data size increases, the training and prediction time for 1000 iterations was calculated and presented in figure 4. Here, the optimization schedule was modified to fit parameters every 50 iterations of the data.

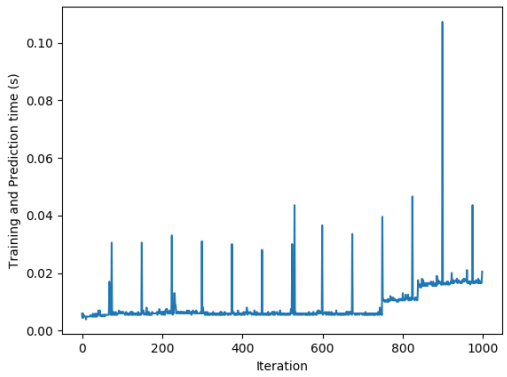

Fig. 4. Time for algorithm to perform training and prediction per iteration of teleoperation signal. Spikes in the graph represent the optimization schedule for this run of the controller.

V. Discussion

The Online Local Sparse Gaussian Process Regression method utilized in this work realized online model learning and control from streaming/sequential data. The performance of the controller appeared to be good on simple geometric trajectories like the circular path shown in this paper. Some more arbitrary paths, not shown here, result in more poor trajectories and performance by the SOLGPR controller. A major drawback of the local Gaussian processes method is the manual tuning of the proximity threshold, w_{gen}. Some paths benefit from certain partitions of the data, therefore the same proximity threshold may not result in the most accurate local models for an arbitrary trajectory. Perhaps, a new strategy or metric for partitioning the data would offer better results. One idea may be to use the distance between a query point and the convex hull of the neighboring local regions as a metric for partitioning. Computation time, however, was kept low by implementing the strategies to tradeoff local model size and sparsity and optimizing a drifting GP periodically. It is evident that the time to learn the inducing parameters for sparse GP can be longer than a standard optimization when the number of training points in a model is already low. A sparsification scheme that adapts this sparse regression for growing data size should likely benefit the overall performance. The optimization scheme under the drifting Gaussian process helped to maintain accuracy of the regression, however the tuning of the drift parameter (i.e. how many of the last data points to include) is a challenge of trial and error. While SOLGPR showcases a cohesive framework to handle streaming and sequential data, the choice of key tuning parameters for the algorithm remains future work. The tradeoff of sparsity, chosen by M support points, and local model size, chosen by an upper bound of K points with L local models, is well framed by this work, however the optimization of these parameters, along with optimization of the proximity threshold w_{gen} and drift parameter D, are the next steps to be taken in improving the current algorithm. This optimization could be formulated in terms of a cost function weighing the cost of performance errors and computation time. Another future step is to modify the simulation to incorporate telecoperation signals from a live operator using a controller (ex. Xbox 360 controller) integrated with ROS. In future work, improvements to the algorithm will be tested using clear benchmarks created to evaluate performance during teleoperation tasks.

VI. References

References

[1] F. Meier, P. Hennig, and S. Schaal, “Incremental Local Gaussian Regression,” in NIPS, 2014.

[2] D. Nguyen-Tuong, J. R. Peters, and M. Seeger, “Local Gaussian process regression for real time online model learning,” in Advances in Neural Information Processing Systems, 2008, pp. 1193–1200.

[3] L. Csato and M. Opper, “Sparse on-line gaussian processes.” Neural computation, vol. 14, no. 3, pp. 641–68, Mar. 2002.

[4] C. E. Rasmussen and C. K. Williams, Gaussian Processes for Machine Learning. MIT Press, 2006.

[5] E. Snelson and Z. Ghahramani, “Sparse Gaussian processes using pseudo-inputs,” Advances in neural information processing systems, vol. 18, p. 1257, 2006.

[6] C. Cheng and B. Boots, ”Incremental Variational Sparse Gaussian Process Regression”, in NIPS, 2016.

[7] F. Meier and S. Schaal, ”Drifting Gaussian processes with varying neighborhood sizes for online model learning”, International Conference on Robotics and Automation, Stockholm, 2016.

[8] D. Nguyen-Tuong, J. R. Peters, ”Model learning for robot control: A survey”, Cognitive Pressing, 2011.

[9] M. K. Titsias, “Variational learning of inducing variables in sparse Gaussian processes,” in International Conference on Artificial Intelligence and Statistics, 2009, pp. 567–574.

[10] ”SheffieldML/GPy: Gaussian processes framework in python”, [Online]: https://github.com/SheffieldML/GPy. [Accessed: 14-Jun-2017]