UNIVERSITY OF CALIFORNIA SAN DIEGO

Sparse Online Locally Adaptive Regression using Gaussian Processes for Continual Robot Model Learning and Control

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Electrical and Computer Engineering (with a specialization in Intelligent Systems, Robotics, and Control)

by

Brian Wilcox

Committee in charge:

Professor Michael Yip, Chair
Professor Nikolay Atanasov
Professor Ken Kreutz-Delgado

2019

The Thesis of Brian Wilcox as it is listed on UC San Diego Academic is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2019

DEDICATION

To my sister Kayla, whom has inspired and comforted me in this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to knowledge Professor Michael Yip for his support in this project and as the chair of my committee. His guidance has helped me shape my work into what it is.

ABSTRACT OF THE THESIS

Sparse Online Locally Adaptive Regression using Gaussian Processes for Continual Robot Model Learning and Control

by

Brian Wilcox

Master of Science in Electrical and Computer Engineering (with a specialization in Intelligent Systems, Robotics, and Control)

University of California San Diego, 2019

Professor Michael Yip, Chair

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, particularly in the task of teleoperation, where commands and sensor data are received in sequential streams. In this thesis, we explore recent advances in Gaussian Processes and aim to achieve online model-learning and control for a teleoperation task on a real robot. 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. 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.

Chapter 1

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 [1]. In this thesis, we focus on the machine learning subdomain of Gaussian Processes and look at its potential in the use case of online teleoperation of robot manipulators.

1.1 Gaussian Processes for Robot Modeling and Control

Gaussian process regression (GPR) has shown promise in modeling the inverse dynamics of robotic systems [2]. 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 [3]. 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 [4][5]. Online sparse methods exist, which take advantage of incremental updates to the model and training while forgetting old data [6]. 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 [7][8]. 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 [7][9], 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 [10]. These approaches, though low in computation cost, may be inefficient when paths 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 [11][12].

1.2 Robot Teleoperation

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.

As the main aim of this thesis work, the robot teleoperation task assumes a streaming setting of data coming in sequentially (or in small batches) 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 in the data streams such that $(\mathbf{x}_{t+1}, y_{t+1})$ is only available after $(\mathbf{x}_t, y_t)$. Since we aim to perform regression on this data 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. Thus, in a loop, new data points are updated to the current regression model, the model is trained, and prediction on the next test signal is performed. 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:** Since the only training data comes in streams which are resultant from the observed output of model predictions, the model must be data-efficient insofar as inference must be based on limited data

- **Low memory usage:** In the streaming setting, the size of the training data, N, may grow indefinitely as the model build, so because we can't assume infinite computational resources, our approach should keep memory usage low as more data is incorporated

- **Iterative:** As opposed to naively re-training models from scratch, it is more resourceful (and less computationally expensive) to iterate on new information and leveraging past information

- **Real-time:** Given that control signals need to be predicted in order to teleoperate the robot live, the algorithm needs to be fast enough to handle training and prediction in real-time.

- **Retains past information:** Though training and prediction when uncovering new regions of the input state-space is vital to success, the approach should also be able to leverage information about previously explored spaces without forgetting

In order to handle this streaming case for teleoperation and control, we formulate a framework called Sparse Online Locally Adaptive Regression using Gaussian Processes (SOLAR_GP).

In chapter 2, we will detail background relevant to our work. Chapter 3 will describe the method of SOLAR_GP. Results are presented from a simulation environment and real Baxter robot in chapter 4 with a discussion and conclusion of this work in chapter 5.

Chapter 2

Background

In a regression problem, one aims to produce a mapping between an input, $\mathbf{x}$, and the output of an underlying latent function, $f(\mathbf{x})$. For real data, we assume access only to observed values, y, of the target latent function, those values being affected by some noise, ε, such that

$$\begin{aligned} y &= f(\mathbf{x}) + \varepsilon \\ \varepsilon &\sim \mathcal{N}(\mathbf{0}, \sigma_n^2) \end{aligned} \tag{2.1}$$

For the regression tasks in this work, we will assume a training format for supervised learning given training data in the form $\mathcal{D} = \{(\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)\}$, where training pair $(\mathbf{x}_i, y_i)$ represents a single input vector of dimension d and its observed output. Given a batch of training data, we will use $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ for an input data matrix and output vector of N training observation points.

In general, the form of these models can be categorized as parametric vs non-parametric. In a parametric approach, the form of the model presumes a particular structure with a finite number of parameters, thus limiting the behavior of its representative capability. For example, in linear regression, a model of the form $y = \mathbf{x}^T \mathbf{w}$, with weight vector of parameters $\mathbf{w}$, presumes a linear relationship between input and output. Though this representation may hold accurate for

many problems, this limitation of model structure will perform poorly on other problems, such as highly non-linear data. While one may approach this issue by utilizing a parameterized model structure more suitable to dealing with a particular data set, this may prove exhaustive to find and potentially intractable. In contrast, non-parametric models do not assume any particular structure on the data and grow in complexity with the available data. In the task of learning inverse robot models, the relationship between inputs and outputs are highly nonlinear and grow in complexity with the robot's degree of freedom. Thus, in this thesis, we only focus on non-parametric models, in particular, models based Gaussian Processes.

After training a regression model to map these inputs to outputs, we aim to perform inference on new data.

2.1 Gaussian Processes

A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. In another view, Gaussian processes represent a distribution over functions, characterized by a mean function and a covariance function:

$$\begin{aligned} f(\mathbf{x}) &\sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \\ m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x}], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned} \tag{2.2}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \sigma_s^2 \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{W}(\mathbf{x} - \mathbf{x}')) \tag{2.3}$$

In this kernel, the signal variance σ_s^2 and the charateristic length-scale or width W are free parameters, or hyperparameters, than can be varied (or learned) in the Gaussian process 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 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:

$$m_*(x) = K_*^T(\sigma_n^2 I + K)^{-1} y \tag{2.4}$$

$$V_*(x_*) = k_{**} - K_*^T(\sigma^2 I + K)^{-1} K_* \tag{2.5}$$

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:

$$logp(y) = log[N(y|0, \sigma^2 I + K)] \tag{2.6}$$

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.

2.2 Sparse Gaussian Processes via Variational Inference

Given the expensive computational complexity of Gaussian process regression, it is often considered intractable for large datasets (e.g. thousands of points). Several methods have been explored in previous literature to address this issue, of which many propose a sparse subset approach where $M << N$ support (or inducing) points are used to approximate the function space and hence reduce the computational complexity. If we augment the posterior distribution $p(f|y)$ with function values, u of the inducing points, then we can obtain the augmented posterior distribution:

$$p(f,u|y) = \frac{p(f|u)p(u)p(y|f)}{p(y)} \tag{2.7}$$

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 [4], Titsias proposes another distribution, $q(f,u)$, which may approximate the original posterior. To accomplish this approximation, we use the variational bayesian methods to propose a distribution with free variational parameters and fit those parameters to be close to the original posterior according to 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:

$$KL(q(f,u)||p(f,u|y)) = \log p(y) + \int q(f,u) \log \frac{q(f,u)}{p(f,u,y)} dudf \tag{2.8}$$

From (2.8), the evidence lower bound, ELBO, on the log marginal likelihood is the second part of the equation. This bound may also be obtained using Jensin's inequality on the log

marginal likelihood. Further, Titsias chooses the approximate distribution, $q(f,u)$, to be of the following form.

$$q(f,u) = p(f|u)q(u) \tag{2.9}$$

By replacing $q(f|u)$ with $p(f|u)$, the prior conditional distribution, the only free parameter is $q(u)$ to affect the approximate distribution, whose optimal distribution can be analytically obtained. Substituting equation (2.9), the lower bound, $\mathcal{F}(q(u))$, becomes:

$$\mathcal{F}(q(u)) = \int p(f|u)q(u)\log\frac{p(u)p(y|f)}{q(u)}dudf \tag{2.10}$$

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

$$\mathcal{F}(q(u)) = log\mathcal{N}(y|0,\sigma^2 I + K_{NM}K_{MM}^{-1}K_{MN}) - \frac{1}{2\sigma^2}Tr[K_{NN} - K_{NM}K_{MM}^{-1}K_{MN}] \tag{2.11}$$

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 (2.10) for both the M support point locations and the hyper-parameters of the kernel. As seen in (2.11), 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.

2.3 Local Gaussian Processes

As described in [8], 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:

$$w_k = exp(-\frac{1}{2}(x - c_k)^T W (x - c_k)) \tag{2.12}$$

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. 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. Additionaly, [13] 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 as follows:

$$y_p = \frac{\sum_{k=1}^{M} w_k y_k}{\sum_{k=1}^{M} w_k} \tag{2.13}$$

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.

2.4 Streaming Sparse Gaussian Processes

Though several approaches have been proposed to minimize the computational complexity of GPR dealing with large datasets, there exist a smaller subset of methods to handle data which comes in sequential streams in an online fashion. Whilst other GP formulations may naively append new data and retrain, most lack a proper framework to eloquently leverage previously trained model information into the next sequential updates. [11] proposed a framework which aims to address these concerns specific to Gaussian processes in the streaming case.

Since in the streaming setting data arrives sequentially, new data (x_{new}, y_{new}) will be added to the old data, (x_{old}, y_{old}). As the goal of variational inference, the aim is to approximate the posterior distribution as well as the marginal likelihood such that we can estimate the inducing points and the kernel hyperparameters online. Assuming only access to the new data is available directly, the old data must be encoded into the previous posterior distribution. The old posterior approximation, $q_{old}(f)$, and the new posterior approximation, $q_{new}(f)$, are of the forms,

$$q_{old}(f) \approx p(f|\mathbf{y}_{old}) = \frac{1}{Z_1(\theta_{old})} p(f|\theta_{old}) p(\mathbf{y}_{old}|f) \tag{2.14}$$

$$q_{new}(f) \approx p(f|\mathbf{y}_{old}, \mathbf{y}_{new}) = \frac{1}{\mathcal{Z}_2(\theta_{new})} p(f|\theta_{new}) p(\mathbf{y}_{old}|f) p(\mathbf{y}_{new}|f) \tag{2.15}$$

Here they recognize that though the new exact posterior, $p(f|y_{old}, y_{new})$, would incorporate the new data, they must find an approximation for $p(y_{old}|f)$ since it cannot be obtained directly. By approximating the new exact posterior as $p(\mathbf{y}_{old}|f) \approx \mathcal{Z}_1(\theta_{old}) q_{old}(f)/p(f|\theta_{old})$ and substituting into (2.15), they find

$$p(f|\mathbf{y}_{old}, \mathbf{y}_{new}) = \frac{\mathcal{Z}_1(\theta_{old})}{\mathcal{Z}_2(\theta_{new})} p(f|\theta_{new}) p(\mathbf{y}_{new}|f) \frac{q_{old}(f)}{p(f|\mathbf{y}_{old})} \tag{2.16}$$

However, $p(f|\mathbf{y}_{old}, \mathbf{y}_{new})$ is intractable, thus they introduce the variational distribution using inducing point locations which are allowed to change after each step. Now, the previous approximate posterior becomes $q_{old}(f) = p(f_{\neq \mathbf{a}}|\mathbf{a}, \theta_{old}) q_{old}(\mathbf{a})$ with $\mathbf{a} = f(\mathbf{z}_{old})$, 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, $q_{new}(b)$, from the new approximate posterior distribution, $q_{new}(f)$, a new lower bound on the log marginal likelihood can be found from the Kullback-Liebler divergence,

$$\begin{aligned} KL[q_{new}(f)||p(f|\mathbf{y}_{old}, \mathbf{y}_{new})] = \log \frac{\mathcal{Z}_1(\theta_{old})}{\mathcal{Z}_2(\theta_{new})} \\ + \int df q_{new}(f) \left[\log \frac{p(\mathbf{a}|\theta\_old) q_{new}(\mathbf{b})}{p(\mathbf{b}|\theta_{new} q_{old}(\mathbf{a}) p(\mathbf{y}_{new}|f)} \right] \end{aligned} \tag{2.17}$$

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. Again as in the batch case, the optimal form of $q_{new}(\mathbf{b})$ can be derived to create a tractable lower bound which propagates the old data via the previous approximate posterior distribution, $q_{old}(\mathbf{a})$. A more detailed derivation of the resultant lower

bound and posterior distribution for prediction is given in the appendix of [11].

The contributions of streaming sparse Gaussian processes enables a much more principled and efficient manner to handle data that comes in streams. As we'll describe in the next chapter, this work provides a basis for the main method of this thesis.

Chapter 3

Methods

In this chapter, we present the methods and contributions of this thesis work. In chapter 2, we presented background on relevant methods in literature which have become the building blocks for the algorithm framework explained in this chapter. In particular, the algorithm developed represents a selective combination of sparse Gaussian processes and local Gaussian processes tailored toward the problem of teleoperation of real robots. Section 3.1 will describe the algorithm in full detail under the non-parallel setting, while section 3.2 explores modifications of the method for use in a real-time control context using robot position-control.

3.1 Sparse Online Locally Adaptive Regression using Gaussian Processes

In this section, the main components of this work's algorithm, Sparse Online Locally Adaptive Regression using Gaussian Processes, (SOLAR_GP), are discussed in detail. The algorithm combines recent work in sparse Gaussian processes for streaming data with local Gaussian process models in a form suitable for teleoperation.

3.1.1 Model Initialization

Before any test signals are provided for regression, we first initialize a sparse Gaussian process model with N_{init} points. If training on time-series data, this may be the first N_{init} points of the data. In the case of robot teleoperation, this initialization would come from a random jitter of the robot's joints. For this initialization, we provide a short procedure to iterate through N_{init} chosen at random between a selected range of the joint's movement. Considering a robot initialized at its starting position, we would typically choose a small range of motion to jitter the robot. With the observed input and output values generated from the initial jitter protocol, a sparse Gaussian process is initialized and optimized. We use methods from Titsias to train the initial inducing points along with the hyperparameters of the kernel (in this paper we use the radial basis function) and noise variance.

Once this initial model has been trained, it is used initially and henceforth as the basis for partitioning of training data into local Gaussian Process models, based on the work from Nguyen. In particular, we'll show later how we use the trained kernel hyperparameters of this "base" model to both partition the data as well as perform prediction.

3.1.2 Local Model Partitioning

In Nguyen's work [8] on Local Gaussian Process Models, they partitioned training inputs into separate models based on their "distance" or "similarity" to previous model centers. The original method computes the distance metric for each model center and new training input, w_k, by using the kernel width, $\mathbf{W}$, determined offline by performing maximum likelihood optimization on a subset of the training data. However, in the streaming, "from-scratch" setting, we assume that no such subset exists yet to learn just one appropriate kernel width. Therefore, we use the kernel width as computed from the "base" model, which incorporates new data into its model in the same sparse streaming framework, as training data is generated throughout the task across the entire

state-space. It is worth noting that the behavior of the kernel width from the base model may differ from the original method, because depending on the number of inducing points in the model, the base model may bias more recent data. Though this may be potentially reasonable given that the kernel width reflects recent trends in the length-scales of the data, we may optionally create a strategy to actively add inducing points to the base model as the precision of certain regions of the input-space are beyond some threshold.

After finding the nearest model (if over the user-defined threshold, w_{gen}, lest a new model is initialized), we incorporate the new training pair, except instead of appending new data with old data, we only need to use the new data to be incorporated into its respective local model. We're able to toss away any old training data because we are using the sparse streaming Gaussian process formulation as described in [11]. In contrast to the previous work's strategy of conserving memory and limiting complexity by retaining no more than a fixed number of local model data points before deleting points, or finding the information gain of new points, we may relieve those concerns under the sparse streaming model, where only the newest data point is required. This advantage will allow local models to stay fixed in computational complexity and in memory.

3.1.3 Training

The training updates and optimization of hyperparameters are crucial in a task such as teleoperation. Since predictions are computed on every new test input at a certain frequency, training occurs online in order to keep the local models up-to-date with the most recent training data.

Training occurs in two phases after a new training experience pair $(\mathbf{x}_{new}, y_{new})$ is received. First, the training pair is added to the base model and trained via streaming Gaussian process optimization. This step is necessary to update the kernel width to be used for partitioning the data into its respective local models as well as for prediction. Once the base model is trained and the experience pair partitioned into a local model, m_k, it is trained. For training, the model

is seeded with the previous m_k model, taking advantage of its past state. Given the structure of the streaming sparse GP model, it is also possible to increase or decrease the inducing points of the model as well as to initialize the inducing point positions, Z, before optimizing for their location. 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 bias the starting point for optimization of the model. Though not extensively explored in this thesis, we have proposed a potential initialization schema that compares the variance of the current inducing inputs and the new batch of training data and distributes a proportion .of inducing points out of the new batch.

If the nearest model to the new training experience has a distance metric less than the user-selected threshold, $w_k < w_{gen}$, then the new pair will be initialized into a new model. For this initialization, we seed it initially with the current base model and allow the inducing inputs to quickly bias to newer points. This process allows for smooth transitioning in newly added models as the input space is explored. Effectively, the local models trained are retaining data-efficient inference even as the algorithm accumulates data very far away. 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 $O(nM^2)$, under the local model scheme the cost for training one local model becomes $O(n_k M_k^2)$. If we consider n training examples partitioned equally into K local models, $(n_k = n/K)$) with $M_k = M/K$ inducing inputs, the overall complexity of training the local models is lower (since $O(Kn_k M_k^2) = O(n\frac{M^2}{K^2})$) than in the batch case. In maintaining a manageable and fixed computational complexity while retaining information in local and adaptive models, SOLAR_GP achieves many of the properties sought for teleoperation tasks.

3.1.4 Prediction

Prediction on test inputs follows the same method as described in section 2.3, where again the kernel width is taken from the trained base model when computing the distance metric from

Algorithm 1 Sparse Online Locally Adaptive Regression using Gaussian Processes

procedure SOLAR_GP
  initialize: Seed initial experience to create initial base sparse streaming GP model
    $m_{base} \leftarrow (\mathbf{x}_{init}, y_{init})$
loop:
  Tester: send next test signal x_{test}
  Predictor: predict, y_{pred}, using latest local models and use as generator input.
    $y_{pred} = \frac{\sum_{k=1}^{M} w_k y_k e^{-V_k}}{\sum_{k=1}^{M} w_k e^{-V_k}}$
  Generator: receive prediction and generate experience input-output pair (x_{exp}, y_{exp})
  BaseTrainer: Update and train base streaming sparse GP model
    $m_{base} \leftarrow (\mathbf{x}_{exp}, y_{exp})$
  Partitioner: partition new experience into nearest of K local regions* .
    if $max(w_k) > w_{gen}$:
      $L_{k_*} \leftarrow (\mathbf{x}_{exp}, y_{exp})$
    else:
      $L_{K+1} \leftarrow (\mathbf{x}_{exp}, y_{exp})$
    *Local regions, $\mathcal{L}$, are analogous to their local GP models, but contain summary of respective models including model centers latest experience pair (cleared after trained)
  Local Trainer: Update and train sparse local GP models $m_k \leftarrow (\mathbf{x}^k_{exp}, y^k_{exp})$
until finished

the test input, x_{test}, to each model's center, c_k. In contrast to the original method, now prediction is of lower computational complexity due to the local models being sparse. Another notable modification is adding the variance of each local model's prediction into the weightings according to:

$$y_p = \frac{\sum_{k=1}^{M} w_k y_k e^{-V_k}}{\sum_{k=1}^{M} w_k e^{-V_k}} \tag{3.1}$$

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 test 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.

For robot teleoperation, the prediction, y_{pred}, is used as the control input for robot controller (i.e. joint torques, velocity, or angle). The resulting state of the robot, x_{exp}, y_{exp},

becomes the new experience for the next loop of training. The full algorithm for online training and prediction is presented in Algorithm 1. Similar to training, now the prediction complexity could potentially be reduced for a single query input. In the batch sparse GP case, prediction on a single point is of $O(M^2)$ complexity. Assuming M inducing points were equally distributed across K local models, $M_k = M/K$, the cost for weighted prediction would be of $O(KM_k^2) = O(\frac{M^2}{K})$. If we were to only choose predictions for the top $K_* < K$ models, then the complexity could be further reduced.

3.2 Online Robot Teleoperation Under Position-Control

In the previous section, we described in detail the components of the SOLAR_GP algorithm. The main objective of this thesis work was to apply this algorithm in robot teleoperation. Previously we described the major components of the algorithm as Tester, Predictor, Generator, Partitioner, and Trainer (base and local). Note that in the Figure 3.1 that for robot teleoperation, the Tester block would be replaced with the Teleoperator, which sends goal test inputs in the trajectory periodically to the Predictor. The Generator is responsible for taking a prediction and generating its relevant experience. 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. We'll note that in order to resolve angle wrap-around issues for joint angles, we encode the joint angles before adding to the Trainer, such that $\theta \rightarrow [\sin\theta, \cos\theta]$. These encodings can be decoded before sent to the controller. Though the presented SOLAR_GP handles training and prediction in the sequential setting, i.e. where all steps in the procedure of Algorithm 1 are performed (and ordered) sequentially, that is not an optimal formulation for online robot teleoperation of a real robot. In this section, we discuss implementation of SOLAR_GP for real-time robot teleoperation using position-control.

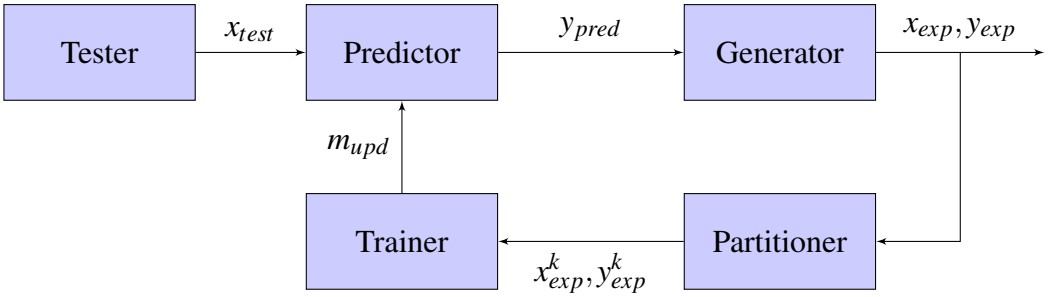

Figure 3.1: Block diagram of control flow for SOLAR_GP. Note that the output experience and the prediction output may not be the same due to both noise and generator variance.

3.2.1 Parallel implementation with Robot Operating System (ROS)

In an ideal setup (such as in pure simulation), Algorithm 1 could be implemented in the sequential streaming setting such that the Predictor on test inputs wait on the latest training update on the previous data experience sending prediction to the robot for position control. However, it is more likely the case that the frequency at which test predictions need to be made is faster than the training time on the last (or last batch) of experienced training data. Parallel implementation of the SOLAR_GP algorithm is an important step toward 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. In this manner, we could control the rate at which the teleoperation task executes, forgiving some potential lag in training. To enact robot teleoperation using the SOLAR_GP method, we implement the algorithm in the Robot Operating System (ROS). ROS is a flexible robotics middleware which enables a diversity of robotics software by providing cross-platform support, multiple-language support, package management, and message passing between processes. In particular, a ROS node is a process which can be connected on a graph to communicate with other ROS nodes via services and messages. By utilizing this node graph properly, we can implement the various steps of the SOLAR_GP algorithm as separate nodes and

allow them to run in parallel.

To best take advantage of the ROS node system, the Tester, Predictor, Generator, Partitioner, and Trainer, have been coordinated into node processes. Specifically, the main nodes that represent the SOLAR_GP algorithm are the *teleop node, predict node, control node, train node*, and the *experience buffer node*. ROS provides a message-passing protocol which allows these nodes to talk with one another across topics or service calls. We've coordinated our node processes to subscribe to certain messages and publish others which enable our algorithm to perform in parallel and operate at different desired frequencies. Below we list the responsibilities and message interaction of the SOLAR_GP for ROS implementation.

- *teleop node*: Analagous to the Tester step, sends next goal pose to the Predictor node
  - Subscription: raw teleoperator device outputs
  - Publish: goal pose of robot end effector
- *predict node*: Analagous to the Predictor node, predicts joint angles of goal pose using the latest available local GP models
  - Subscription: goal pose of robot end effector, local sparse GP model parameters
  - Publish: joint angle prediction
- *control node*: One aspect in the Generator step, uses robot's internal position controller to reach predicted joint position
  - Subscription: joint angle prediction
- *train node*: Analogous to the combined Trainer and Partitioner, this process both partitions and trains new batches of experience
  - Subscription: experience buffer of input-ouput pairs

  - Publish: local sparse GP model parameters

- *experience buffer node*: Second aspect of the Generator step, this node creates a buffer of the next N data points experienced and sends to the training node when ready

  - Subscription: current robot end effector position, current robot joint position
  - Publish: experience buffer of input-output pairs

The parallel node context allows for the predictions of the joint angle to be provided to the robot position controller at a fixed rate while the training node works to update the model with the last batch of points. 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 at a desired rate. In order to ensure that redundant experiences are given to the trainer, particularly when the robot is staying still, a threshold on the joint movement must be met before being added to the buffer of experiences. Once the Trainer receives the next batch from the experience buffer, the buffer is cleared and begins collecting data for the next batch. Figure 3.2 shows the ROS node graph of the system running SOLAR_GP with the Baxter robot simulation running. An expected consequence of the parallel training updates is that when training updates lag behind the Predictor, the predictions are expected to perform worse. especially if the currently available model has poor inference in the queried region. However, as we'll describe in our experiments, this parallel configuration works still works well with operating a robot at a desired frequency when training updates are not too slow.

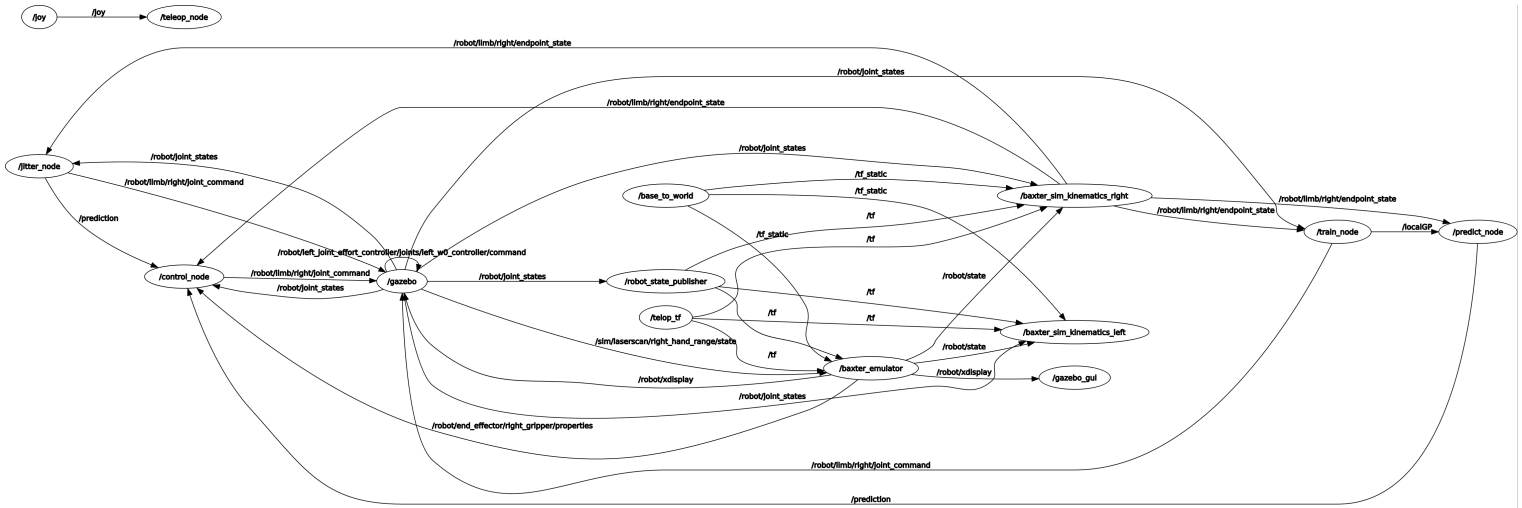

Figure 3.2: ROS node graph showing communication scheme for online model-learning and control.

Chapter 4

Experiments and Results

In this chapter, we present experiments and results aimed to highlight the behavior of the SOLAR_GP method. We'll show SOLAR_GP on sample n-link robots in the sequential teleoperation setting. Finally, we demonstrate the capability of our method to run live teleoperation on both a simulated and real Baxter robot. We build Gaussian Process models using the GPy library from the Sheffield Machine Learning Github repository [14]. We also adapt the streaming Gaussian Process method from the author's Github repository [15] to work in the GPy framework.

4.1 Sequential Robot Teleoperation for n-Link Manipulators

One of the primary motivations behind this work was to learn inverse kinematics/dynamics models for arbitrary robot models. In particular, we aim to learn under a forward prediction setting, where it is implied that test signals are streamed to the robot model sequentially. We are interested in "from-scratch" continual learning, therefore, after an initialization, the model must perform training and prediction online. In these experiments, we focus the robot model learning problem on inverse kinematics.

Inverse kinematics solutions map the robot end effector poses (position and orientation)

to joint configurations:

$$\theta = f^{-1}(\mathbf{x}_e) \tag{4.1}$$

The inverse kinematics for some robot models may be derived analytically, though other numerical solutions are often used for complex robot models. Since as robots become more complex, inverse solutions become more intractable, it is a suitable target function for SOLAR_GP.

4.1.1 Setup

For this base experiment for "simulated teleoperation", we created forward kinematics models for arbitrary *n*-Link manipulators. This forward model will be treated as a replacement for the real robot's position controller and sensing system, the Generator in our model structure, by taking joint prediction and providing the "experience" as the resultant forward kinematics solution. To create the forward kinematics models, we use the Denavit-Hartenberg (DH) parameter formulation [16] to easily build serial link manipulators. Under this framework, we can generate our forward kinematics solutions to the end effector pose by computing a series of homogeneous transformation matrices from the base joint to the end effector,

$$\overset{base}{ee}\mathrm{T} = \overset{0}{1}\mathrm{T}\overset{1}{2}\mathrm{T}\ldots\overset{n-1}{n}\mathrm{T} \tag{4.2}$$

Each transformation matrix, T, is parameterized using the DH-parameters. A more thorough explanation of the parameters and the DH formulation can be found in [16].

The Tester in this experiment to simulate the teleoperated signal is replaced by a geometric trajectory creator which feeds the next test point in the trajectory to the Predictor. For this base scenario, test signals arrive sequentially and in order only after the model is ready to receive another input after training. Later we'll handle the scenario when test signals arrive parallel to the model's training.

As described in the methods, the model is initialized using a random jitter of the robot's

joints. Once initialized the robot is ready to perform online, where the model's prediction output is used for "control" and the output experience for training. Since the trajectory is pre-computed, the model must quickly learn and adapt to accurately perform.

4.1.2 *n*-Link Results

We use SOLAR_GP to learn a the inverse kinematics mapping of a pre-defined trajectory of end effector poses streamed to an *n*-link manipulator arm. First, we show the results on planar robot models. Figure 4.1 shows the resultant paths of the online prediction on three planar robots with 3,4, and 6 links, respectively. Note that the support points for each local model cover the path as the task space is explored.

Figure 4.1: Images of a planar 3, 4, 6 DOF manipulator arm recreating a pentagon, spiral, and star trajectory from simulated teleoperation test points. The colored dots on the image (green, and pink), represent locations of the support points optimized from the SOLAR_GP.

We next extend into 3D poses and robots. In figure 4.2, paths for a 3-link and 6-link manipulator are shown in 3D space.

Table 4.1 shows the root mean squared error for the experiment trajectories and the actual generated poses, summarizing the result. Though ideally the errors would be computed on the prediction itself, because the predictions are used as joint inputs to the robot to generate the

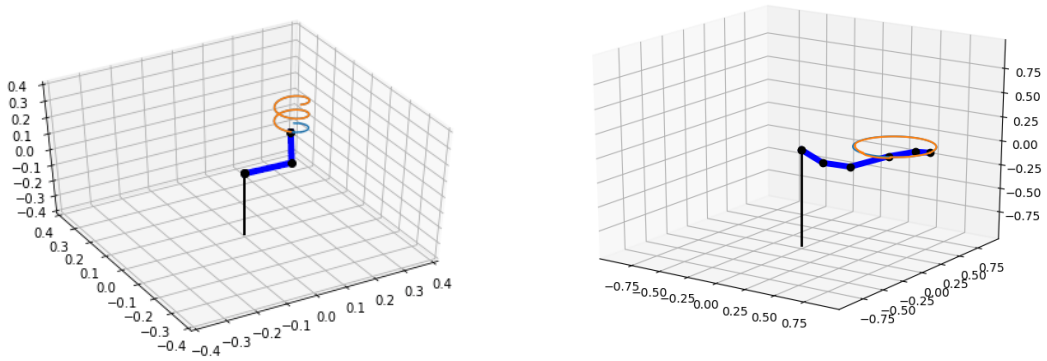

Figure 4.2: Images of a 3 DOF and 6 DOF manipulator arm recreating a 3D helix and circle trajectory from simulated teleoperation test points.

Table 4.1: Comparison of RMSE for sequential robot teleoperation experiments

	3 DOF pent	4 DOF spiral	6 DOF star	3 DOF helix	6 DOF circle
rmse	0.0045	0.0077	0.0084	0.0068	0.0086
# local models	5	4	2	3	4
# support pts	25	25	25	30	40

experience, it satisfies to compare the resulting path. Since the ”true” inverse kinematics model may not be known, errors between the generated experience with the desired result is a suitable metric for self-learning.

4.2 Teleoperation using Position Control on Baxter Robot

We aim to show the performance of SOLAR_GP on the Baxter robot, demonstrating its strengths on a realistic live teleoperation and control environment. Baxter is an industrial robot with two 7-degree-of-freedom arms created and developed by ReThink Robotics. Its use of series elastic actuators (SEA) and safety design has made it a popular robot for human collaborative use in both industry and research. We take advantage of the open source Baxter repository from ReThink Rootics to perform both simulation and real experiments on the Baxter robot.

4.2.1 Baxter Simulator Teleoperation

In these experiments, we perform real-time teleoperation on Baxter using a Microsoft Xbox controller (though not used in the experiments presented in this thesis, we have also supported teleoperation via the Touch Optic Device by 3D Systems). In section 3.2.1, we discussed the ROS structure utilized to support parallel teleoperation, prediction, and training. For these experiments, we computed joint positions which were sent to Baxter's position controller. Even during training, the ROS version of the SOLAR_GP method continuously makes predictions on the input teleoperation signal while accumulating training data. This parallel interface relieves the algorithm from longer training times and missing data while the Trainer is optimizing the most recent batch. As opposed to the sequential n-link experiments from section 4.1, the Baxter robot is able to stream experiences of its current pose and joint configuration throughout its control, so intermediate points are all available to increase the density of training data. An advantage of our method here is that this continuously growing dataset does not affect the the complexity of the algorithm's training and prediction.

In our tests, only position (excluding orientation) was used for the input robot pose into SOLAR_GP. We chose to fix the 3 twist joints on Baxter's elbow and wrists in order to reduce redundancy from the twist joint at Baxter's shoulder. For each teleoperation experiment, we started the robot in a neutral position for the Baxter arm (we use the left arm, though our algorithm is agnostic to either), and then randomly jittered the active robot joints to generate the initial model. Because training and predictions are online and "from-scratch", much of the workspace begins unexplored, therefore poor predictive performance is expected when test poses are far away. Since teleoperation in this learning setting is dependent on good predictive performance, we ensured that the teleoperator deliberately sent test poses to the prediction node by pressing a button on the controller. So that we could create automated tests for later playback, we added functionality to save and clear paths sent from the teleoperator. With the saved trajectory, we are able to send it to the SOLAR_GP algorithm as though they were live.

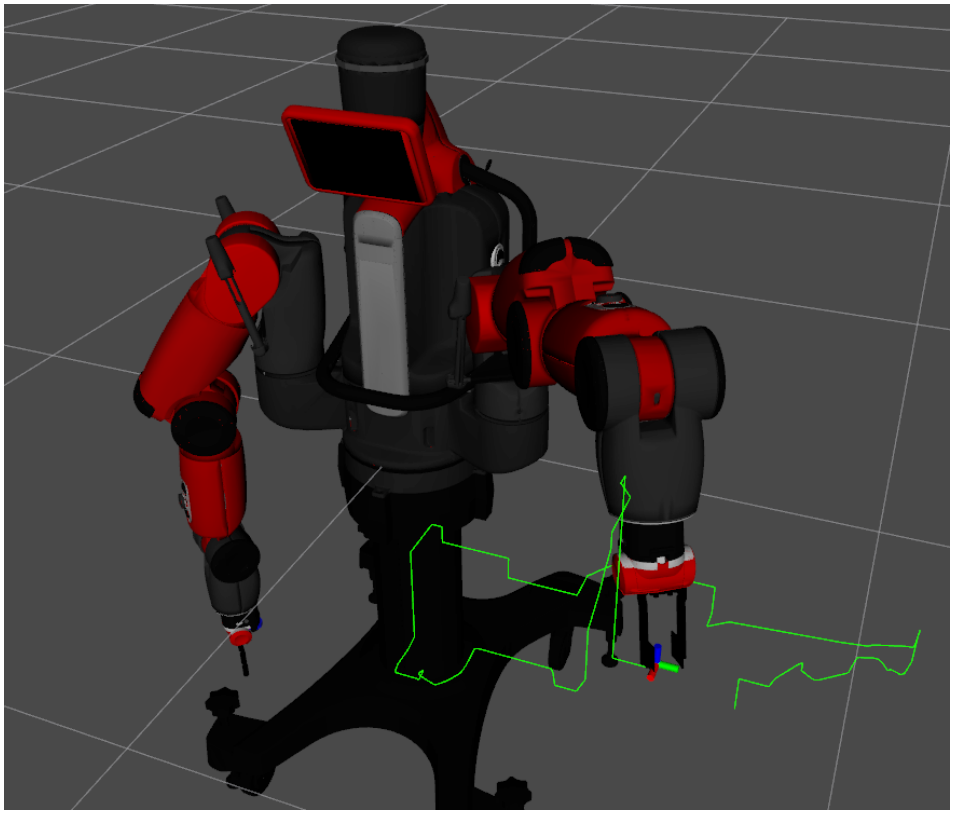

Figure 4.3: Image of Baxter robot during a teleoperation experiment to follow a test trajectory

After saving a live trajectory, shown in Figure 4.3, from a real-time teleoperation of Baxter using the Xbox controller, we played back the trajectory for Baxter to learn under varying user-defined parameters of the SOLAR_GP algorithm. We tested the SOLAR_GP algorithm on this path for two main user-defined parameters, number of inducing points per local model, and the weighted distance threshold, w_{gen}, the parameter that controls the threshold before a new model is generated. 140 runs of the trajectory were recorded for a range of [15, 45] for the number of inducing points and [0,8, 0.985] for the w_{gen}. 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 fourth final pass with training disabled. The final pass is tested to see that the trained model still performs well without new updates, thus not forgetting previous paths. Though only seldom reached, we capped the number of newly generated models at 10.

Figure 4.4 a) shows the squared error between the test pose and the real pose along the trajetory path in time when the model is defined with 25 inducing points and a weighted distance threshold, w_{gen}, of 0.939. We clearly see that the squared error remained low for the majority of the test. Interpretation of the error is simple, given that the root mean squared error corresponds to the average distance error across entire path. Thus, for the first pass through the teleoperated trajectory, the end effector pose of the Baxter robot was on average less than a centimeter from the desired position. At its worse in the first pass, the actual pose deviated up to about 3 centimeters before quickly recovering. As the repeated trials passed, the predictive performance increased until reaching the peak performance with an average distance error (RMSE) of 44 millimeters from the desired teleoperator position after training updates stopped.

We see from Figure 4.5 that on average using 25 inducing points had the lowest RMSE on the test trajectory. We also observe that, though not the highest average among weighted distance thresholds, the best performance on the trajectory came from a w_{gen} of 0.985 with 40 inducing points that resulted in 8 models being created across the the teleoperation path.

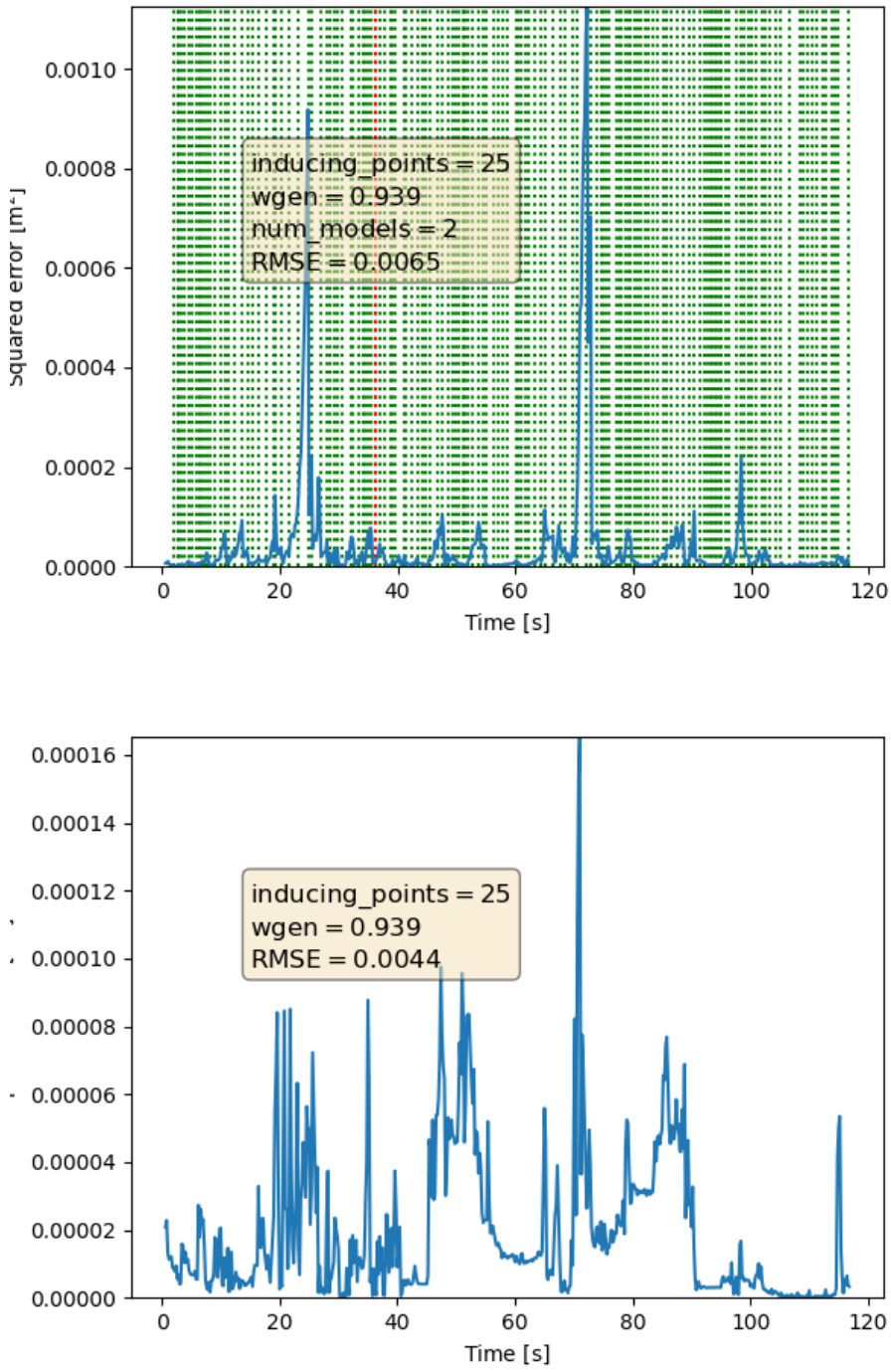

Figure 4.4: **a)** top: Plot of squared error $[m^2]$ of test and actual pose vs time for first pass through trajectory. Green vertical lines represent locations of training updates. The red vertical line indicates that a new local model has been added. **b)** bottom: Plot of squared error $[m^2]$ of test and actual pose vs time for pass through trajectory without training updates

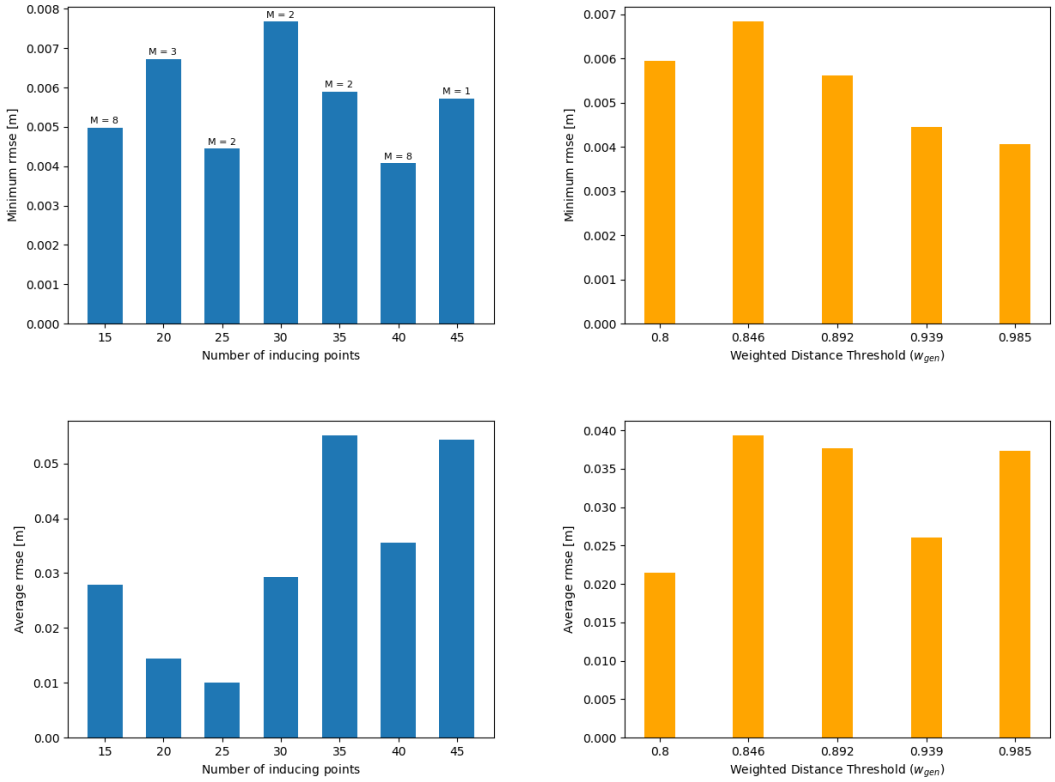

Figure 4.5: **a)** top left: Plot of number of inducing points vs minimum root mean squared error (RMSE) **b)** top right: Plot of number of weighted distance threshold (w_{gen}) vs minimum RMSE **c)** bottom left: Plot of number of inducing points vs average RMSE **d)** bottom right: Plot of number of w_{gen} vs average RMSE

4.2.2 Baxter Pick up Demo

As a test of coordination and successful learning for teleoperation tasks, we demonstrate using SOLAR_GP to learn how to move the robot for a pick up goal. In this test case, the teleoperator's job is to direct the robot end effector to pick up an object on a table. This task highlights the dexterity of the learning algorithm to achieve fine motor skills online, suitable for real user objectives. The same initialization procedure was followed and immediately live teleoperation was provided to bring the robot towards the block object for grasping. A button on the xbox controller was mapped to grab the object when the teleoperator is ready. Figure 4.6 shows the successful grasp of the block object in the Gazebo simulation.

We should state here that though successful pick up of an object was achieved in the demonstration, part of the challenge in this teleoperation task is to coordinate visual and spatial awareness of object position with the robot end effector's pose. For this reason, tangent to this work, in order to take advantage of learning algorithms for teleoperation tasks, one benefits from rich visual (or potentially haptic) feedback to send "good" command signals.

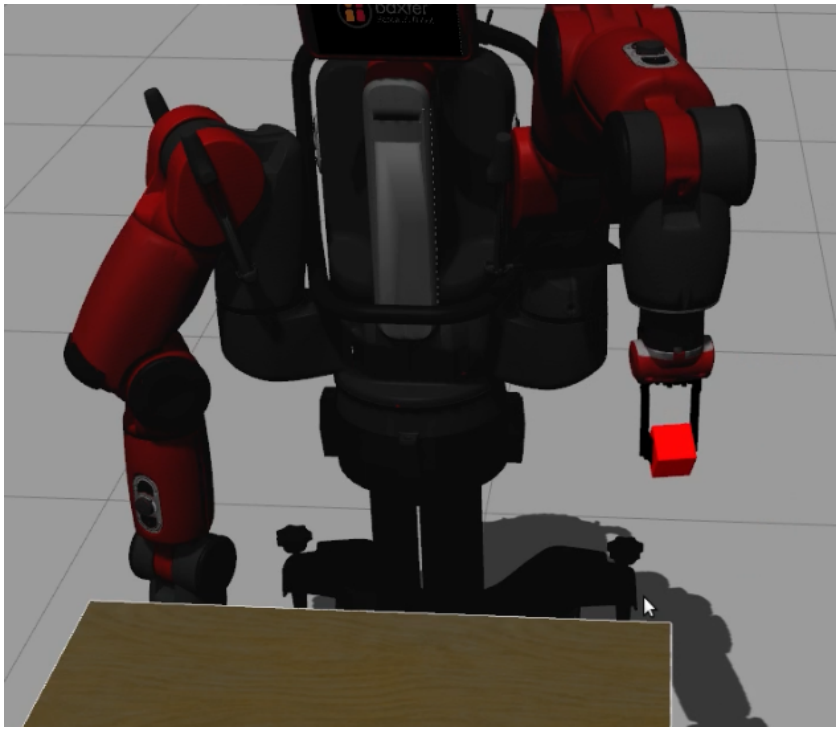

Figure 4.6: Image of Baxter robot during a teleoperation experiment to pick up a block from a table

4.2.3 Real Baxter Teleoperation Demo

The Baxter simulator in the ROS Gazebo physics environment uses the same API as does the real Baxter robot. Therefore, setting up our methods on the real Baxter were trivial. We demonstrate the real-time position control of Baxter from teleoperated signals using SOLAR_GP by directing the its arm's end effector position to draw the path of a square, as illustrated in figure 4.7.

We note that, though almost the same, our real Baxter has additional safety limits and settings as well as a very light "shaking" of its wrist at the end effector. This, however, did not fundamentally change anything about our application or results.

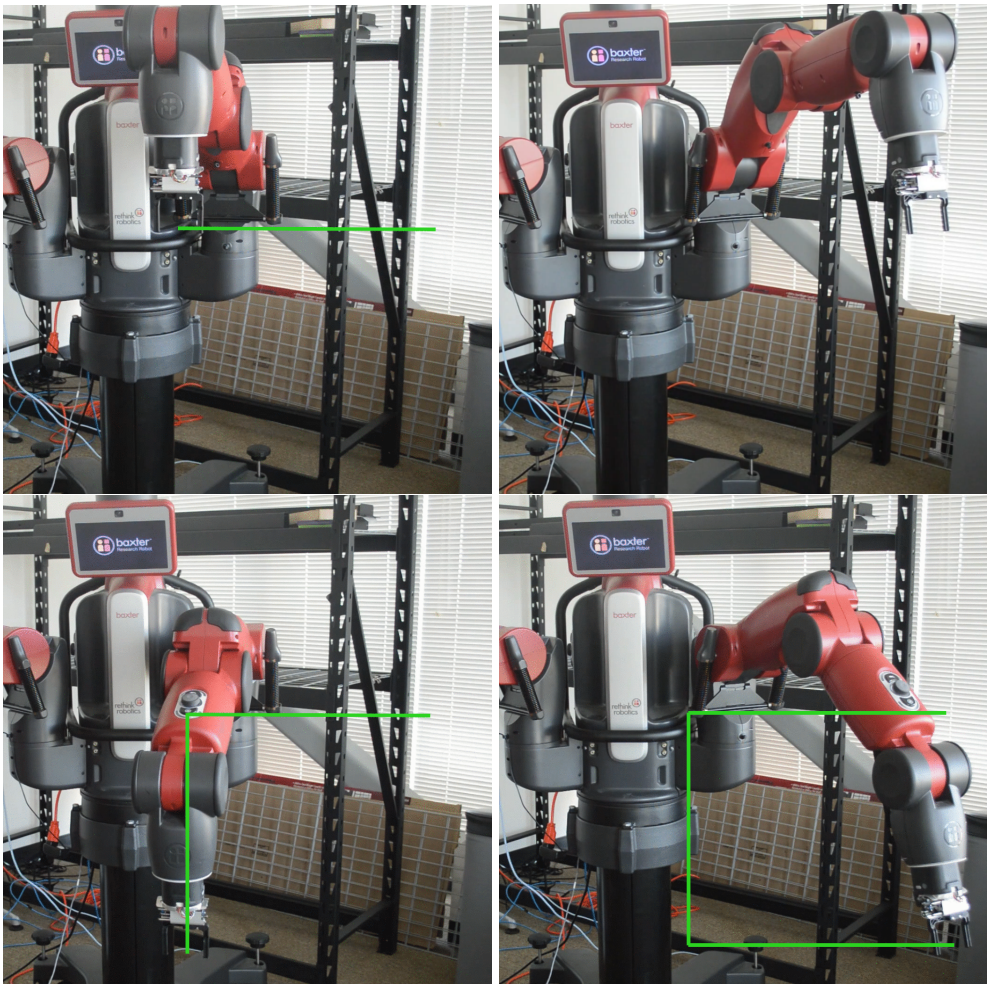

Figure 4.7: Images of real Baxter robot following a live teleoperation of a square trajectory

Chapter 5

Conclusion

5.1 Discussion

In section 4, we presented experimental results for the teleoperated streaming setting of robot inverse kinematics model learning. For the non-parallel, sequential order version of the SOLAR_GP algorithm, implemented on n-Link manipulators, the results exhibited high precision with reasonable settings for choice of number of inducing points and the weighted distance threshold. Though not thoroughly explored in this thesis, the number of inducing points needed for a large dataset under standard sparse Gaussian process regression compared with that of SOLAR_GPR would have been insightful. However, though likely that the number of total inducing points across all local models may exceed that in the single sparse model case after creation of many local models, 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.

Effectively, 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 naturally approaches the truth when more data closer to the target is trained. Thus, we see in the plots of squared error vs time as in figure 4.4 that even after the resultant pose from a joint prediction is far from the desired pose, the experience from driving toward and including that new experience aids pushes the model's next prediction closer to the truth. As an example, if the teleoperator requests a desired pose far from the model's current input distribution, then the poor joint predictions can lead to better future predictions so long as the robot model can provide additional training data from the last attempt. 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. The model's ability to continue learning for a single test query extends until the predictions converge.

Section 4.2.1 also explored multiple parameter configurations for learning inverse kinematics on the Baxter simulator. Out of the 140 test runs, we found that the best number of inducing points per local model on average was 25. Though other configurations could yield precise predictions, they have high variability in predictive performance based on the chosen distance threshold, w_{gen} for creation of new models. 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 case, that is not always the rule. 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 late training updates which may lag good predictions. Generally, however, we should expect that as training updates and initialization are equal that models with large number of inducing points perform more accurate predictions. This is evidenced by the one single best result of 0.0044 root mean squared error on the test trajectory using 40 inducing points with a high weighted distance threshold for new models. It is also the case that the streaming sparse GP models will start to drift towards more recent points, meaning forward predictions tend to do well, but when predicting with online training turned off as in the final run of each parameter

configuration, the SOLAR_GP models with more local models tended to perform better because they ”remember” the older data.

5.2 Summary

In this thesis, we demonstrated that SOLAR_GP performs well in the streaming data setting suitable for long-term continual learning. For the demands of robot model learning for teleoperated tasks in real time, our method was able to combine the strengths of Local Gaussian Process Regression with the 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, while the ROS version of SOLAR_GP with the Baxter robot highlighted the significant performance increase of parallel training, and predictions. We were able to demonstrate this thesis' main goal to show that the algorithm was suitable for real time position control of a real robot from scratch given live test trajectories from a teleoperator.

5.3 Future Work

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. This was a lingering question in the work of [8], since the choice of w_{gen} affects when new models are created. One may consider this distance metric to a model's center, w to be a naive discriminant between models in the sense that a model's capacity to predict with low uncertainty, i.e. small predictive variance. A potential metric could then be to use the predictive variance of the training point to determine whether any model has a high enough confidence to incorporate the new training point. This choice of metric

would already conform to the modification made in our method for weighting the predictions. Another suggestion would be to instead consider the distance to the convex hull of a model's data. If the new data point is far enough from the convex hull of all, it can seed a new model. Since we use a sparse streaming model and do not retain all training data, perhaps the convex hull of the induced points would provide interesting results.

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. Though we've seen that for reasonable values of w_{gen}, the number of models won't blow up, it is inherent that for very large input spaces (for example in long time series data), many models are possibly produced. Since, as mentioned, the choice of w_{gen} is not optimal, it is unknown a priori whether all new models are necessary. Future work may consider how to redistribute models, potentially combining and re-partitioning as needed to satisfy both computational constraints and some optimality criterion.

SOLAR_GP was primarily explored in the teleoperation setting, however future work could focus on its application in path planning and decision making. As more aspects of robot path planning, control, decision making have moved into the research of machine learning, application of end-to-end machine learning solutions are the likely future of robotics. One particular interesting application of SOLAR_GP would be in model-based reinforcement learning. In this setting, the RL-algorithm uses the model learned from SOLAR_GP in order to estimate the optimal policy. Looking at some recent works in online reinforcement learning, it would be interesting to explore a fully online implementation for robot decision making and path planning.

Appendix A

SOLAR_GP for ROS Source Code

Reference source code for the SOLAR_GP code used in the ROS-based experiments with the Baxter robot may be found at https://github.com/bpwilcox/SOLAR_GP-ROS.

The supporting package for teleoperation can be found at https://github.com/bpwilcox/teleop_utils.

For use on the Baxter robot, packages from the Rethink Robotics public Github repository are required [17].

Bibliography

[1] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," *Cognitive Pressing*, April 2011.

[2] R. C. E. and C. K. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[3] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in Neural Information Processing System*, vol. 18, p. 1257, 05 2005.

[4] M. K. Titsias, "Variational learning of inducing variables in sparse gaussian processes," in *International Conference on Artificial Intelligence and Statistics*, pp. 567–574, 2009.

[5] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

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

[7] F. Meier, P. Hennig, and S. Schaal, "Incremental local gaussian regression," in *Advances in Neural Information Processing Systems*, 2014.

[8] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Local gaussian process regression for real time online model learning and control," in *Advances in Neural Information Processing Systems*, 2008.

[9] C.-A. Cheng and B. Boots, "Incremental variational sparse gaussian process regression," in *Advances in Neural Information Processing Systems*, 2016.

[10] F. Meier and S. Schaal, "Drifting gaussian processes with varying neighborhood sizes for online model learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[11] T. D. Bui, C. V. Nguyen, and R. E. Turner, "Streaming sparse gaussian process approximations," in *Advances in Neural Information Processing Systems*, 2017.

[12] T. D. Bui, C. V. Nguyen, and R. E. Turner, "Partitioned variational inference: A unified framework encompassing federated and continual learning," *arXiv:1811.11206*, 11 2018.

[13] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[14] S. M. L. Group, "Sheffieldml/gpy: Gaussian processes framework in python." https://github.com/SheffieldML/GPy. Accessed: 2019-03-15.

[15] T. Bui, "thangbui/streaming_sparse_gp: Streaming sparse gaussian process approximations." https://github.com/thangbui/streaming_sparse_gp. Accessed: 2019-03-15.

[16] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*. John Wiley and Sons, 1989.

[17] R. R. Inc., "Rethink robotics inc.." https://github.com/RethinkRobotics. Accessed: 2019-03-15.