# Efficient Bayesian Local Model Learning for Control

Franziska Meier[1], Philipp Hennig[2] and Stefan Schaal[1,2]

***Abstract*— Model-based control is essential for compliant control and force control in many modern complex robots, like humanoid or disaster robots. Due to many unknown and hard to model nonlinearities, analytical models of such robots are often only very rough approximations. However, modern optimization controllers frequently depend on reasonably accurate models, and degrade greatly in robustness and performance if model errors are too large. For a long time, machine learning has been expected to provide automatic empirical model synthesis, yet so far, research has only generated feasibility studies but no learning algorithms that run reliably on complex robots. In this paper, we combine two promising worlds of regression techniques to generate a more powerful regression learning system. On the one hand, locally weighted regression techniques are computationally efficient, but hard to tune due to a variety of data dependent meta-parameters. On the other hand, Bayesian regression has rather automatic and robust methods to set learning parameters, but becomes quickly computationally infeasible for big and high-dimensional data sets. By reducing the complexity of Bayesian regression in the spirit of local model learning through variational approximations, we arrive at a novel algorithm that is computationally efficient and easy to initialize for robust learning. Evaluations on several datasets demonstrate very good learning performance and the potential for a general regression learning tool for robotics.**

## I. INTRODUCTION

Besides functionality and precision, computational cost is a crucial design criterion for machine learning algorithms in real-time settings, such as control problems. An example is the problem of building a model for robot kinematics or dynamics: The sensors in a robot can produce thousands of data points per second, quickly amassing a locally dense covering of the input domain of a particular task. In such situations, robust and fast exploitation of this local knowledge and compression for fast lookup is more important than long-range generalization – when entering a new part of the workspace, the robot will quickly create enough new data to learn about this new part. A learning method should produce a good model from the large number $N$ of data points, using a comparably small number $M$ of model parameters. Since this has to happen in real time, complexity must be low not just in $N$, but also in $M$.

From this viewpoint, learning with local models fulfills these requirements. For instance, locally weighted regression (LWR) [1] decomposes the learning problem by training $M$ *independent, local* regression models. The number $M$ is not fixed and can grow with the amount of data; the local regression models are usually simple, e.g. low order polynomials [2]. In the spirit of a Taylor series expansion, the idea of LWR is that simple models with few parameters may give a good local description, while it may be difficult to find sufficiently expressive nonlinear features to capture the entire function. A lot of good local models can be combined to form a good global one.

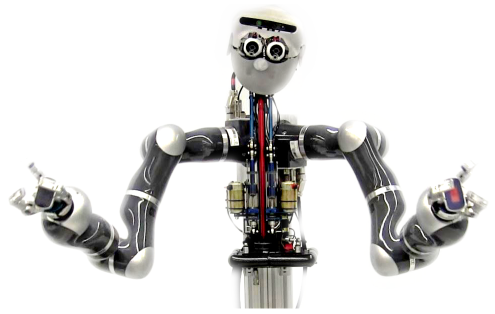
Fig. 1. One of our experimental platforms: The robot *Apollo* is a dual arm manipulation platform consisting of two 7 DoF KUKA light weight arms, two Barrett hands and a SARCOS humanoid head.

The key to LWRs computational efficiency is that each local model is trained independently. The resulting low cost profile has made LWR popular in robot learning. But, while advanced versions of LWR [3] have only linear cost $\mathcal{O}(NM)$, they require several tuning parameters, whose optimal values can be highly data dependent, such that robustness and learning accuracy are not always easily obtained. This is at least partly a result of the strongly localized training strategy, which does not allow models to "coordinate", or to benefit from other local models in their vicinity.

In contrast, Gaussian Process regression (GPR) [4] is a popular global regression method because it offers principled inference for hyperparameters. However, due to high computational cost applying GPR to problems within the domain of learning control has been difficult in the past. Recently, the sparsification of Gaussian processes [5], [6], [7] and online versions of such sparse GPs [8], [9] produced variants of GPR suitable for real-time model learning problems [10]. Nevertheless, it has not been demonstrated yet, that these variants are efficient enough for online learning of hyperparameters, as well. Instead, in [10] the distance metric is learned offline. Online performance then depends on how well the offline training data represents the full state-space of the robot. Furthermore, typically one global distance metric is learned, making it difficult to fit the non-stationary data

[1]Computational Learning and Motor Control Lab, University of Southern California, Los Angeles, CA 90089, USA. [2]Max-Planck-Institute for Intelligent Systems, 72076 Tübingen, Germany. Email: `fmeier@usc.edu`

This research was supported in part by National Science Foundation grants ECS-0326095, IIS-0535282, IIS-1017134, CNS-0619937, IIS-0917318, CBET-0922784, EECS-0926052, CNS-0960061, the DARPA program on Autonomous Robotic Manipulation, the Army Research Office, the Okawa Foundation, the ATR Computational Neuroscience Laboratories, and the Max-Planck-Society.

encountered in robotics. Alternatively, previous work has addressed these issues with local probabilistic regression approaches [11], [12], [13]. However, they did so from bottom up – constructing locally valid models independently of each other, which are then combined at prediction time.

Here, we propose a probabilistically motivated alternative to LWR – *local Gaussian regression* (LGR), a linear regression model that explicitly encodes localization. The well-known probabilistic Gaussian regression framework with radial basis function (RBF) features for nonlinear function approximation forms the basis and starting point of this work. A key insight of our work is that RBF feature functions can be reinterpreted as localizers of constant feature functions. This allows us to capture the notion of local models within the Gaussian regression framework. In its exact form, this model has the cubic cost in $M$ typical of Gaussian models, arising because observations induce correlation between all local models in the posterior. From the top down, we start with the globally optimal Bayesian linear regression framework and show step by step how to transform it into a localized inference procedure employing variational approximations to reduce computational complexity. This procedure leads to a local learning algorithm that combines the best of two worlds, i.e., the efficiency of LWR and the accuracy and cleanliness of Bayesian regression with local model coupling in the posterior. To our knowledge, this is the first top-down construction, starting from a globally optimal training procedure, to find approximations giving a localized regression algorithm similar in spirit to LWR.

## II. Bayesian Local Gaussian Regression

We start out with the classic Bayesian linear regression model with a Gaussian likelihood for regression parameters $\boldsymbol{w}$

$$p(\boldsymbol{y}\,|\,\boldsymbol{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n\,|\,\boldsymbol{w}^T\boldsymbol{\phi}^n, \beta_y^{-1}) \tag{1}$$

and adopt the idea of automatic relevance determination [14], [15] using a factorizing prior

$$p(\boldsymbol{w}\,|\,\boldsymbol{\alpha}) = \prod_{m=1}^{M} \mathcal{N}(w_m\,|\,\alpha_m^{-1}) \tag{2}$$

where $M$ is the dimensionality of feature vector $\boldsymbol{\phi}^n = \boldsymbol{\phi}(\boldsymbol{x}_n)$. Here we assume that our features are formed by $M$ spatially localized basis functions like the Radial basis function (RBF) kernel. The $m^{th}$ RBF feature

$$\eta_m(\boldsymbol{x}) = \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_m)\Lambda_m^{-1}(\boldsymbol{x}-\boldsymbol{c}_m)^\top\right]. \tag{3}$$

is parametrized by its center $\boldsymbol{c}_m \in \mathbb{R}^D$ and the positive definite metric $\Lambda_m \in \mathbb{R}^{D,D}$, both of which need to be estimated from observed data $D = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$. Here we assume a diagonal distance metric $\Lambda_m = \operatorname{diag}(\lambda_{m1}^{-2}, \dots, \lambda_{mD}^{-2})$. Learning within this framework can be done by Expectation-Maximization with hidden variables $\boldsymbol{w}$ and parameters $\theta = \{\beta_y^{-1}, \{\alpha_m, \boldsymbol{\lambda}_m\}_{m=1}^M\}$, assuming that the number of basis functions $M$ and the centers $\boldsymbol{c}_m$ are known. Here, we depart from the classic Bayesian inference framework to move towards a computationally efficient, local inference procedure. To arrive at a localized learning algorithm it is helpful to reinterpret the RBF features as localizing functions applied to a constant function $\xi_m$. Thus, the $m^{\text{th}}$ feature $\phi_m^n = \xi_m\eta_m^n$ can be thought of as a local constant model, such that each local model consists of its own set of parameters $\{w_m, \alpha_m, \lambda_m, c_m\}$. Using this interpretation, we see that in the standard EM learning framework inference over regression parameters $w_m$ of all $M$ local models is coupled resulting in a E-Step which has cubic cost in $M$. Furthermore, optimization of the hyperparameters is coupled over all local models, resulting in an expensive maximization step. In the following, we show how to modify the above regression model, such that the learning of each local model's parameters is decoupled from the rest and thus results in local computations. Next we explore the re-interpretation of the RBF kernel as a localizer of general functions $\xi_m$, to arrive at more expressive local models. Finally, we will show how to incrementally place the locals models.

### A. From Global to Local Computations

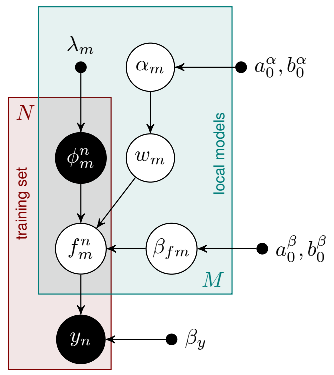

Fig. 2. Graphical model for Local Gaussian Regression (LGR)

Intuitively, using localizing feature functions renders local models almost independent because correlation between far away models is approximately zero. However, traditional inference schemes treat all local models as one big system, which is solved for as if all local models are fully correlated with each other. Here, we want to exploit that intuition of "almost independence" to arrive at a learning scheme in which algorithmically local models are actually being treated almost independently.

In order to do so, we first adopt the idea of Bayesian backfitting [16], and introduce a hidden variable $f_m^n$ for each local model $m$ and data point $n$. This hidden variable can be interpreted as a virtual target for each local model, such that all local models targets $f_m^n$ sum up to the actual target $y_n$. The complete data likelihood of the modified model is

$$\begin{aligned} p(\boldsymbol{y}, f\,|\,\theta) & \\ &= \prod_n \mathcal{N}(y^n\,|\,\mathbf{1}^\top\boldsymbol{f}^n, \beta_y^{-1}) \prod_m \mathcal{N}(f_m^n\,|\,w_m\phi_m^n, \beta_{fm}^{-1}) \end{aligned} \tag{4}$$

and we place Gaussian priors on the regression parameters $w_m$

and gamma priors on the precision parameters $\{\beta_{fm}, \alpha_m\}_{m=1}^M$

$$p(\boldsymbol{w}, \boldsymbol{\beta_f}, \boldsymbol{\alpha}) = \prod_{m=1}^{M} p(w_m \mid \alpha_m) p(\beta_{fm}) \prod_{k=1}^{K} p(\alpha_{mk}) \quad (5)$$

$$= \prod_m \mathcal{N}(w_m \mid 0, \alpha_m^{-1}) \mathcal{G}(\beta_{fm} \mid a_0^\beta, b_0^\beta) \quad (6)$$

$$\prod_{k=1}^{K} \mathcal{G}(\alpha_{mk} \mid a_0^\alpha, b_0^\alpha) \quad (7)$$

To summarize, our model, visualized in Figure 2, has hidden variables $z = \{w_m, \{f_m^n\}_{n=1}^N\}_{m=1}^M$ and parameters $\theta = \{\beta_y, \{\beta_{fm}\}_{m=1}^M, \{\alpha_{mk}\}_m^M, \{\lambda_m\}_{m=1}^M\}$.
Performing exact inference in this model is intractable, but even if it were tractable, we are looking for an approximation that renders all updates on the local models independent. Imposing the following factorized approximation on the posterior $q$

$$q(\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{\beta}_f, \boldsymbol{\alpha}) = q(\boldsymbol{f}) q(\boldsymbol{w}) q(\boldsymbol{\beta}_f) q(\boldsymbol{\alpha}) \quad (8)$$

achieves exactly that. Note, that simply assuming the factorization between $\boldsymbol{f}$ and $\boldsymbol{w}$ automatically results in factorized posterior distributions over all $w_m$. It can be shown [17] that the approximate posterior distributions are of the form

$$\begin{aligned}
\log q(\boldsymbol{f}^n) &= \mathbb{E}_{\boldsymbol{w}, \boldsymbol{\beta}_f}[\log p(y^n \mid \boldsymbol{f}^n, \beta_y) \quad (9)\\
&\quad + \log p(\boldsymbol{f}^n \mid \boldsymbol{w}^T \boldsymbol{\phi}^n, \boldsymbol{\beta_f})]\\
&= \log \mathcal{N}(\boldsymbol{f}^n \mid \mu_{f^n}, \Sigma_f)\\
\log q(w_m) &= \mathbb{E}_{\boldsymbol{f}, \beta_{fm}, \boldsymbol{\alpha}_m}[\sum_n \log \mathcal{N}(f_m^n \mid w_m^T \phi_m^n, \beta_{fm}^{-1}) \quad (10)\\
&\quad + \log \mathcal{N}(w_m \mid 0, \boldsymbol{\alpha}_m^{-1})]\\
&= \log \mathcal{N}(w_m \mid \mu_{w_m}, \Sigma_{w_m})\\
\log q(\beta_{fm}) &= \mathbb{E}_{f_m, w_m}[\sum_n \log \mathcal{N}(f_m^n \mid w_m^T \phi_m, \beta_{fm}^{-1}) \quad (11)\\
&\quad + \mathcal{G}(\beta_{fm} \mid a_n^\beta, b_N^\beta)]\\
&= \log \mathcal{G}(\beta_{fm} \mid a_n^\beta, b_N^\beta)\\
\log q(\alpha_m) &= \mathbb{E}_{\boldsymbol{w}}[\log \mathcal{N}(w_m \mid 0, \operatorname{diag} \boldsymbol{\alpha}_m{}^{-1}) \quad (12)\\
&\quad + \mathcal{G}(\alpha_m \mid a_0^\alpha, b_0^\alpha)]\\
&= \log \mathcal{G}(\alpha_m \mid a_N^\alpha, b_{Nm}^\alpha)
\end{aligned}$$

This results in the following variational Bayes update equations for the posterior mean and covariance of each local models target $\boldsymbol{f}^n = (f_1^n, \ldots, f_M^n)$

$$\begin{aligned}
\Sigma_f &= B^{-1} - B^{-1}\mathbf{1}(\beta_y^{-1} + \mathbf{1}^T B^{-1}\mathbf{1})^{-1}\mathbf{1}^T B^{-1}\\
&= B^{-1} - \frac{B^{-1}\mathbf{1}\mathbf{1}^T B^{-1}}{\beta_y^{-1} + \mathbf{1}^T B^{-1}\mathbf{1}} \quad (13)\\
\mu_{f_m^n} &= \mathbb{E}_{\boldsymbol{w}}[w_m^T]\phi_m(\boldsymbol{x}_n) + \quad (14)\\
&\quad \frac{1}{\beta_y^{-1} + \mathbf{1}^{\top} B^{-1}\mathbf{1}} B^{-1}\mathbf{1}\left(y_n - \sum_{m=1}^{M} \mathbb{E}[w_m]^T \phi_m^n\right)
\end{aligned}$$

where $B = \mathbb{E}_{\boldsymbol{\beta}_f} \operatorname{diag}(\hat{\beta}_{f1}, \ldots, \hat{\beta}_{fM})$, with $\hat{\beta}_{fm} = \frac{a_N^\beta}{b_{Nm}^\beta}$. Notice, how this step combines all local models predictions for data point $\boldsymbol{x}_n$ and then distribute the error to the each local models fake target. Thus, this is the step where all local models "communicate" with each other to reduce the prediction error. Given these fake targets, all local models updates can be updated independently – in parallel. The posterior mean and covariance of each models regression parameters are given by

$$\Sigma_{w_m} = (\hat{\beta}_{fm} \sum_n \phi_m^n \phi_m^{n\,T} + \hat{A}_m)^{-1} \quad (15)$$

$$\mu_{w_m} = \hat{\beta}_{fm} \Sigma_{w_m} \sum_n \phi_m^n \mu_{f_m^n} \quad (16)$$

where $\hat{A}_m = \frac{a_N^\alpha}{b_{Nm}^\alpha}$. Furthermore, the coefficients of the posterior gamma distributions are updated as

$$\begin{aligned}
a_N^\alpha &= a_o^\alpha + 0.5 \quad (17)\\
b_{Nm}^\alpha &= b_0^\alpha + 0.5(\mu_{w_m}^2 + \Sigma_{w_m}) \quad (18)\\
a_N^\beta &= a_0^\beta + 0.5N \quad (19)\\
b_{N,m}^\beta &= b_0^\beta + 0.5\sum_n(\mathbb{E}[w_m]^T \phi_m^n - \mathbb{E}_{\boldsymbol{f}}[f_m^n])^2) \quad (20)\\
&= b_0^\beta + 0.5(\sum_n(\mu_{w_m}^T \phi_m^n - \mu_{f_m^n})^2 + N\sigma_{fm}^2\\
&\quad + \operatorname{Tr}\{\phi_m \Sigma_{w_m} \phi_m^T\})
\end{aligned}$$

Finally, to optimize the length scales $\lambda_m$ we maximize the expected complete log likelihood under the variational bound

$$\boldsymbol{\lambda}^{opt} = \arg\max_{\boldsymbol{\lambda}} \mathbb{E}_{\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}_f}[\log p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}_f \mid \Phi, \boldsymbol{\lambda})]$$

which nicely factorizes into independent maximization problems for each local model

$$\boldsymbol{\lambda}_m^{opt} = \arg\max_{\boldsymbol{\lambda}_m} = \mathbb{E}_{\boldsymbol{f}, \boldsymbol{w}}[\sum_{n=1}^{N} \mathcal{N}(f_m^n \mid w_m \phi_m^n, \beta_{fm}^{-1})] \quad (21)$$

which are optimized via gradient ascent. Note, that while we can derive update equations for precision variable $\beta_y$, we have found that keeping it fixed at a large value, avoids potential ambiguities between $\boldsymbol{\beta}_f$ and $\beta_y$.
As a result, all update equation are local, with the exception of $\mu_{f^n}$ the posterior mean of the hidden variable. Note also, that all updates have low computational cost including the updates of the posterior mean and covariance of the fake targets. Furthermore, prediction at a test point $\boldsymbol{x}_*$ also has a low computational complexity of $\mathcal{O}(MK)$. To compute the predictive distribution at test point $\boldsymbol{x}^*$ we first marginalize over hidden variable $\boldsymbol{f}$

$$\begin{aligned}
&\int \mathcal{N}(y_*; \mathbf{1}^T \boldsymbol{f}_*, \beta_y^{-1}) \mathcal{N}(\boldsymbol{f}_*; W^T \boldsymbol{\phi}(x_*), B^{-1}) d\boldsymbol{f_*}\\
&= \mathcal{N}(y_*; \sum_m w_m \phi_m^*, \beta_y^{-1} + \mathbf{1}^T B^{-1}\mathbf{1}) \quad (22)
\end{aligned}$$

and then $\boldsymbol{w}$

$$\begin{aligned}
&\int \mathcal{N}(y_*; \boldsymbol{w}^T \boldsymbol{\phi}^*, \beta_y^{-1} + \mathbf{1}^T B^{-1}\mathbf{1}) \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_w, \Sigma_w) d\boldsymbol{w}\\
&= \mathcal{N}\left(y_*; \sum_m^M \mu_{w_m} \phi_m^*, \sigma^2(x^*)\right) \quad (23)
\end{aligned}$$

resulting in the posterior predictive mean $\sum_{m=1}^M w_m^T \phi_m^*$, which is the sum of all local models predictions. The predictive variance $x^*$ is given by

$$\sigma^2(x^*) = \beta_y^{-1} + \sum_{m=1}^{M} \hat{\beta}_{fm}^{-1} + \sum_{m=1}^{M} \phi_m^{*\,T} \Sigma_{w_m} \phi_m^*. \quad (24)$$

Thus far we have considered each local model to be a constant (one dimensional) localized model, each with its own precision and length scale parameters. This already mimics two important advantages of LWR: The possibility to fit functions with spatially varying noise levels and to locally adapt to varying output distributions. We now show how to increase the degrees of freedom per local model.

## B. Localized Feature Functions

In LWR, the form of each local model is free to be chosen as any parametric model. Typically, linear models are considered to be the best trade-off between variance and bias of the resulting regression estimates. Thus, we

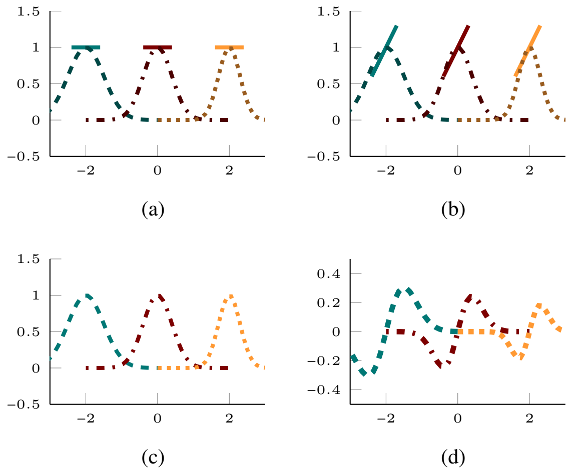

Fig. 3. Feature functions. (top) local constant (a) and linear (b) models together with their respective localizing RBF kernels. (bottom) the resulting features when both localizer and local constant model in c) and local linear model in d) are combined.

extend our notion of local constant models to allow for local parametric models. For this, we go back to the insight that RBF features can be viewed as a constant function $\xi$ times the RBF function, $\phi_m(\boldsymbol{x}) = \xi_m \eta_m(\boldsymbol{x})$. In this interpretation the constant functions $\xi_m$ are weighted by the RBF kernel, and thus we can view RBF features as locally weighted constant models. This representation allows us to create more complicated, localized, models by introducing general local models $\xi(\boldsymbol{x})$ and denoting the feature function to be $\phi_m(\boldsymbol{x}) = \xi_m(\boldsymbol{x})\eta_m(\boldsymbol{x})$. For instance we can now define the local linear models $\xi(\boldsymbol{x}) = \boldsymbol{x}$ and arrive at a form of locally weighted regression with linear local models. See Figure 3, for a comparison of RBF features with constant functions and RBF features with linear functions. This representation allows to create a variety of different local models, however from now on we will work with local linear functions $\xi(\boldsymbol{x}) = \boldsymbol{x}$. This generalization does not alter the form of the update equations but the dimensionality of the posterior mean and covariance of $w_m$ changes from one dimensional to $\boldsymbol{w}_m \in \mathbb{R}^{K,1}$ and $\Sigma_{w_m} \in \mathbb{R}^{K,K}$ respectively, where K is the dimensionality of the linear function – typically the input dimension plus a bias term. Furthermore, one can decide to put an independent prior precision variable $\alpha_{mk}$ on each dimension of $\boldsymbol{w}_m$, enabling the pruning of dimensions of local models. With this we have increased the flexibility of each local model. While that slightly increases the computational update cost per local model we empirically show that this pays off in terms of increased predictive performance.

## C. LGR vs LWR

So far we have focused on showing similarities between local Gaussian regression and locally weighted regression, especially in terms of computational cost. However, it is instructional to look at the differences between both methods. The first key difference is the way of how locality is used to decompose the function approximation problem into a series of independent and local model updates. In LWR regression parameters are fit on a data set weighted by the RBF kernel, essentially creating a different data set for each local model. In our model, the locality is achieved by weighting each local function $\xi(\boldsymbol{x})$ by the RBF kernel to determine the local models contribution to the target $y$. Thus, optimization of the length scales $\boldsymbol{\lambda}_m$ results in different behaviors. In LWR, because both input and output are weighted, reducing the length scale generally improves the fit. In our model, length scales are only reduced if it actually improves the overall fit of the fake targets. However, if it improves the fit length scales also can grow such that the local models actually cover larger areas of the input space. On average, we thus produce larger local models than LWR.

The second key difference is that variations of LWR exist that work on streaming data and incrementally allocate new local models when necessary [3]. Incoming data is immediately processed to update the affected local models, and is then immediately discarded. Our algorithm as presented here does not work yet under the assumption of streaming data, but rather assumes that a batch of data is available. However, we take a first step into the direction of a fully incremental model in the next section where we present an algorithm to incrementally place local models.

## III. Incremental Allocation of Local Models

Thus far, we have assumed that the number $M$ and locations $\boldsymbol{c}$ of local models were known. To incrementally add local models at new locations $\boldsymbol{c}_{M+1}$ we extend LGR analogous to the incremental learning of the relevance vector machine [18]. As a result, the incremental algorithm starts with one local model, and per iteration adds one local model before the variational E-Step. Conversely, existing local models for which all components $\alpha_{mk} \to \infty$ are pruned out. This works well in practice, with the caveat that the number local models $M$ can grow fast initially before the pruning becomes effective. Thus we check for each selected location $\boldsymbol{c}_{M+1}$ whether any of the existing local models centered at $\boldsymbol{c}_{1:M}$ produces a localizing weight $\eta_m(\boldsymbol{c}_{M+1}) \geq w_{\text{gen}}$, where $w_{\text{gen}}$ is a parameter between 0 and 1 and regulates how many models are added. An overview of the final algorithm is given in Algorithm 1.

**Algorithm 1** LGR with incremental allocation of local models

```
Require: ${x_n, y_n}_{n=1}^N, a_0^\alpha, b_0^\alpha, a_0^\beta, b_0^\beta, \lambda^{init}, w_{\text{gen}}$
 1: $a_N^\alpha = a_0^\alpha + 0.5,\ a_N^\beta = a_0^\beta + 0.5N$
    // initialize first local model with center $c_1$ and initial length scale $\lambda_1$
 2: $M = 1$
 3: $LM_1 = \{c_1 = x_1, \lambda_1 = \lambda^{init}, \hat{\beta}_1 = a_0^\beta / b_0^\beta, \hat{A}_1 = a_0^\alpha / b_0^\alpha\}$
    // iterate over all data points
 4: for $n = 2 \dots N$ do
 5:   if $\eta_m(x_n) < w_{\text{gen}}, \forall m = 1, \dots, M$ then
 6:     $M = M + 1$
 7:     $c_M = x_n, \lambda_M = \lambda^{init}, \hat{\beta_{fm}} = \frac{a_0^\beta}{b_0^\beta}, \hat{A}_M = \frac{a_0^\alpha}{b_0^\alpha}$
 8:     $LM_M = \{c_M, \lambda_M, \hat{\beta_M}, \hat{A}_M\}$
 9:   end if
10:   $B = \mathrm{diag}(\beta_{f1}, \dots, \beta_{fM})$
11:   $s = \beta_y^{-1} + \sum_m \beta_{fm}$
      // variational E-Step
12:   $\Sigma_f = B^{-1} - B^{-1}\mathbf{1}\mathbf{1}^T B^{-1} / s$
13:   $\mu_{f_m^n} = \mu_{w_m} \boldsymbol{\phi}_m^n + 1/s B^{-1}\mathbf{1}\left(y_n - \sum_{m=1}^M \mu_{w_m}^T \boldsymbol{\phi}_m^n\right)$
14:   for all $m$ do
15:     $\Sigma_{w_m} = (\hat{\beta}_{fm} \sum_n \phi_m^n {\phi_m^n}^T + \hat{A}_m)^{-1}$
16:     $\mu_{w_m} = \hat{\beta}_{fm} \Sigma_{w_m} \sum_n \phi_m^n \mu_{f_m^n}$
        // variational M-Step
17:     $b_{Nm}^\alpha = b_0^\alpha + 0.5(\mu_{w_m}^2 + \Sigma_{w_m})$
18:     $b_{N,m}^\beta = b_0^\beta + 0.5(\sum_n (\mu_{w_m}^T \phi_m^n - \mu_{f_m^n})^2 + N\sigma_{fm}^2$
19:                  $+ \mathrm{Tr}\{\phi_m \Sigma_{w_m} \phi_m^T\})$
20:     $\hat{A}_m = a_N^\alpha / b_{Nm}^\beta,\ \hat{\beta}_{fm} = a_N^\beta / b_{Nm}^\beta$
        // pruning
21:     if $A_m > 1e3$ then $M = M - 1; LM_m = []$ end if
22:   end for
23: end for
```

## IV. Experiments

We evaluate and compare LGR to locally weighted projection regression (LWPR) – an extension of LWR suitable for regression in high dimensional space [3] – on three data sets. For the first set of experiments data is drawn from the "cross function" in Figure 4, often used to demonstrate locally weighted learning on a problem with strongly varying spatial nonlinearities. For the second set of evaluations we compare both methods on the task of inverse dynamics learning on data collected from two robotic platforms: The SARCOS anthropomorphic robot arm with seven degrees of freedom and the right arm on a dual-arm manipulation platform with two KUKA lightweight arms, two Barrett hands and a SARCOS head, as shown in Figure 1.

In both experiments, LWPR performed multiple iterations through the data sets. The local Gaussian regression implementation is executed based on Algorithm 1 and are allowed an additional 1000 iterations to reach convergence.

### A. Synthetic Data

We used $2{,}000$ uniformly distributed training inputs, with zero mean Gaussian noise of variance $(0.2)^2$. The test set is a regular grid of 1641 edges without noise and is used to evaluate how well the underlying function is captured. The initial length scale was set to $\lambda_m = 0.3, \forall m$, and we ran each method with length scale learning (LSL) for $w_{\text{gen}} = 0.1, 0.2, \dots, 0.9$. All results presented here are results averaged over 5 randomly seeded runs. To understand the role of parameter $w_{\text{gen}}$ better, we summarize the normalized mean squared error as a function of parameter $w_{\text{gen}}$ for both methods in Figure 5. The key message of this graph is that the parameter $w_{\text{gen}}$ does not affect the performance of LGR or LWPR greatly. However, a significant difference between the performances of both algorithms is the amount of local models allocated as $w_{\text{gen}}$ increases. LWPR increasingly allocates more and more local models with larger values for $w_{\text{gen}}$ but we only see a slight increase in the number of local models for LGR, indicating that LGRs pruning mechanism works

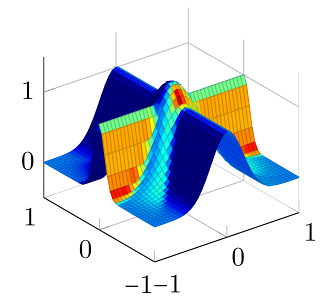

Fig. 4. (a) 2D cross function.

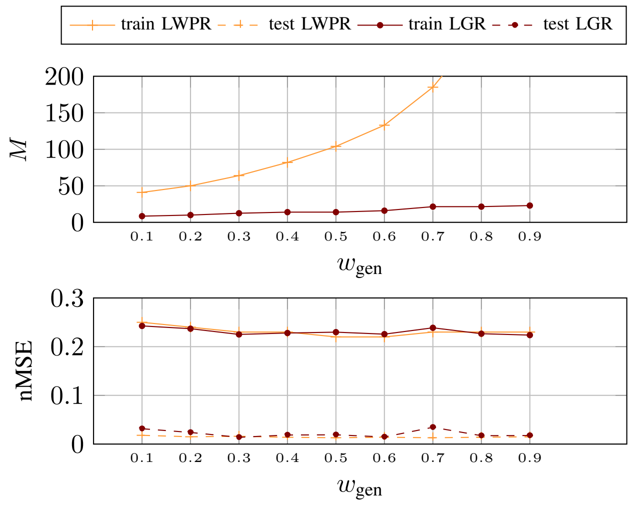

Fig. 5. normalized mean squared error (bottom) and # of learned local models (top) on 2D cross data with length scale learning.

TABLE I
PREDICTIVE PERFORMANCE ON 2D CROSS DATA

| | nMSE | # LM |
|---|---|---|
| **LWPR** | 0.016 | 60 |
| **GPR** | 0.013 | $\infty$ |
| **LGR constant** | 0.048 | 14.0 |
| **LGR linear** | 0.014 | 13.6 |

TABLE II
PREDICTIVE PERFORMANCE ON SARCOS INVERSE DYNAMICS

| | LWPR | | | LGR | | |
|---|---|---|---|---|---|---|
| Joint | nMSE | (MSE) | # LM | nMSE | (MSE) | # LM |
| 1 | 0.045 | (19.16) | 419 | 0.021 | (8.76) | 467 |
| 2 | 0.039 | (8.94) | 493 | 0.021 | (4.62) | 567 |
| 3 | 0.034 | (3.42) | 483 | 0.016 | (1.58) | 328 |
| 4 | 0.024 | (4.55) | 384 | 0.017 | (3.29) | 378 |
| 5 | 0.064 | (0.06) | 514 | 0.02 | (0.02) | 368 |
| 6 | 0.075 | (0.22) | 519 | 0.017 | (0.05) | 353 |
| 7 | 0.03 | (0.20) | 405 | 0.019 | (0.13) | 344 |

TABLE III
PREDICTIVE PERFORMANCE ON KUKA INVERSE DYNAMICS

| | LWPR | | | LGR | | |
|---|---|---|---|---|---|---|
| Joint | nMSE | (MSE) | # LM | nMSE | (MSE) | # LM |
| 1 | 0.14 | (4.39) | 1258 | 0.071 | (2.11) | 804 |
| 2 | 0.075 | (4.603) | 1378 | 0.049 | (2.98) | 704 |
| 3 | 0.151 | (0.9366) | 1257 | 0.079 | (0.49) | 761 |
| 4 | 0.076 | (0.922) | 1324 | 0.056 | (0.67) | 769 |
| 5 | 0.071 | (0.034) | 1331 | 0.051 | (0.0245) | 767 |
| 6 | 0.084 | (0.086) | 1407 | 0.047 | (0.048) | 615 |
| 7 | 0.093 | (0.068) | 1406 | 0.029 | (0.022) | 650 |

very well. This suggests, that LGR uses its local models more effectively to solve the function approximation task. Finally, we also evaluate the performance of LGR with local constant models vs local linear models in Table I. Again, the results were averaged over 5 randomly seed trials with parameters $w_{\text{gen}} = 0.3$ and initial length scale of $\lambda = 0.3$. With approximately the same number of local models, the LGR with linear local models outperforms LGR with constant models. As a baseline comparison, we provide results obtained from Gaussian process regression, where we have used the publicly available GPML toolbox [19].

### *B. Inverse Dynamics Learning*

We use the publicly available SARCOS data [20], which consists of rhythmic motions and contains $44,484$ training data points and $4,449$ test data points. For the inverse dynamics learning of the KUKA light weight arm we collected $100,000$ training data points and $21,000$ test data points of rhythmic motions at various speeds. Both arms have seven degrees of freedom, thus for both learning tasks we have 21 input variables representing joint positions, velocities and accelerations for the 7 joints. The task is to predict the 7 joint torques. In Table II we show the predictive performance of LWPR and LGR on the SARCOS data when trained with initial length scale $\lambda = 0.3$ and with $w_{\text{gen}} = 0.3$. On the KUKA lightweight arm an initial length scale of $0.3$ would generate thousands of local models, thus we start out with a slightly larger length scale of $0.4$ and $w_{\text{gen}} = 0.1$. Both evaluations show that on average LGR uses less local models than LWPR to achieve high accuracy. This supports our findings on the synthetic data indicating that LGR uses local models more effectively than LWPR.

## V. CONCLUSION

We have taken a top-down approach to developing a probabilistic localized regression algorithm: We start with the generative model of Gaussian linear regression, which amounts to a fully connected graph and thus has cubic inference cost. To break down the computational cost of inference, we first reinterpret RBF functions as localized feature functions – local models – and argue that because of the localization these local models are approximately independent. We exploit that fact through a variational approximation that reduced computational complexity to local computations. Empirical evaluation suggests that LGR can achieve on par or better performance than LWPR. More importantly our results indicate that LGR uses its local models more effectively than LWPR. A final step left for future work is to re-formulate our algorithm into an incremental version that can deal with a continuous stream of incoming data.

## REFERENCES

[1] Stefan Schaal and Christopher G Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[2] T. Hastie and C. Loader. Local regression: Automatic kernel carpentry. *Statistical Science*, 8:120–143, 1993.

[3] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: Incremental real time learning in high dimensional space. In *ICML*, pages 1079–1086, 2000.

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

[5] Michalis K Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574, 2009.

[6] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

[7] Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum Gaussian process regression. *JMLR*, 11:1865–1881, 2010.

[8] Marco F Huber. Recursive Gaussian process: On-line regression and learning. *Pattern Recognition Letters*, 45:85–91, 2014.

[9] Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural computation*, 14(3):641–668, 2002.

[10] Arjan Gijsberts and Giorgio Metta. Real-time model learning using incremental sparse spectrum Gaussian process regression. *Neural Networks*, 41:59–69, 2013.

[11] Jo-Anne Ting, Mrinal Kalakrishnan, Sethu Vijayakumar, and Stefan Schaal. Bayesian kernel shaping for learning control. *Advances in neural information processing systems*, 6:7, 2008.

[12] Duy Nguyen-Tuong, Jan R Peters, and Matthias Seeger. Local Gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems*, pages 1193–1200, 2008.

[13] Narayanan U Edakunni, Stefan Schaal, and Sethu Vijayakumar. Kernel carpentry for online regression using randomly varying coefficient model. In *IJCAI*, 2007.

[14] R.M. Neal. *Bayesian learning for neural networks*. Springer Verlag, 1996.

[15] Michael E Tipping. Sparse Bayesian learning and the relevance vector machine. *The Journal of Machine Learning Research*, 1:211–244, 2001.

[16] Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. The Bayesian backfitting relevance vector machine. In *ICML*, 2004.

[17] Zoubin Ghahramani. Graphical models and variational methods. *Advanced Mean Field Method—Theory and Practice*, 2000.

[18] Joaquin Quiñonero-Candela and Ole Winther. Incremental Gaussian processes. In *NIPS*, 2002.

[19] Carl Edward Rasmussen and Hannes Nickisch. Gpml Gaussian Processes for Machine Learning toolbox, 2013.

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