Model Learning with Local Gaussian Process Regression

Duy Nguyen-Tuong Matthias Seeger Jan Peters

Max Planck Institute for Biological Cybernetics

Spemannstr. 38, 72076 Tübingen

{*duy.nguyen-tuong, matthias.seeger, jan.peters*}*@tuebingen.mpg.de*

Abstract

Precise models of the robot inverse dynamics allow the design of significantly more accurate, energy-efficient and more compliant robot control. However, in some cases the accuracy of rigid-body models does not suffice for sound control performance due to unmodeled nonlinearities arising from hydraulic cable dynamics, complex friction or actuator dynamics. In such cases, estimating the inverse dynamics model from measured data poses an interesting alternative. Nonparametric regression methods, such as Gaussian process regression (GPR) or locally weighted projection regression (LWPR), are not as restrictive as parametric models and, thus, offer a more flexible framework for approximating unknown nonlinearities. In this paper, we propose a local approximation to the standard GPR, called local GPR (LGP), for real-time model online-learning by combining the strengths of both regression methods, i.e., the high accuracy of GPR and the fast speed of LWPR. The approach is shown to have competitive learning performance for high-dimensional data while being sufficiently fast for real-time learning. The effectiveness of LGP is exhibited by a comparison with the state-of-the-art regression techniques, such as GPR, LWPR and ν-SVR. The applicability of the proposed LGP method is demonstrated by real-time online-learning of the inverse dynamics model for robot model-based control on a Barrett WAM robot arm.

Keywords: Robotics, Inverse dynamics, Model-based control, Machine learning, Nonparametric regression

1 Introduction

Precise models of technical systems can be crucial in technical applications. In robot tracking control, only a well-estimated inverse dynamics model allow both high accuracy and compliant, low-gain control. For complex robots such as humanoids or light-weight arms, it is often hard to analytically model the system sufficiently well and, thus, modern regression methods can offer a viable alternative [1, 2]. However, highly accurate regression methods such as Gaussian process regression (GPR) suffer from high computational cost, while fast real-time learning algorithms such as locally weighted projection regression (LWPR) are not straightforward to use, as they require manual adjustment of many data dependent parameters.

In this paper, we attempt to combine the strengths of both approaches, i.e., the high accuracy and comfortable use of GPR with the fast learning speed of LWPR [3]. We will proceed as follows: firstly, we briefly review both model-based control as well as two nonparametric learning approaches, i.e., standard GPR and LWPR. We will discuss the necessity of estimating the inverse dynamics model and further discuss the advantages of both regression methods in learning this model. Subsequently, we describe our local Gaussian process models (LGP) approach and related work. We show that LGP inherits both the precision of GPR and a higher speed similar to LWPR.

In Section 4, the learning accuracy and performance of the presented LGP approach will be compared with several relevant regression methods, e.g., standard GPR [4], ν-support vector regression (ν-SVR) [5], sparse online GP (OGP) [6] and LWPR [1, 2]. The applicability of the LGP for low-gain model-based tracking control and real-time learning is demonstrated on a Barrett whole arm manipulator (WAM) [7]. We can show that its tracking performance exceeds analytical models [8] while remaining fully compliant.

1.1 Background

Model-based control, e.g., computed torque control [9] as shown in Figure 1, enables high speed and compliant robot control while achieving accurate control with small tracking errors for sufficiently precise robot models. The controller is supposed to move the robot that is governed by the system dynamics [9]

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \boldsymbol{\epsilon}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{u}\,, \tag{1}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ are joint angles, velocities and accelerations of the robot, respectively, $\mathbf{u}$ denotes the applied torques, $\mathbf{M}(\mathbf{q})$ the inertia matrix of the robot and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ Coriolis and centripetal forces, $\mathbf{G}(\mathbf{q})$ gravity forces and $\boldsymbol{\epsilon}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ represents nonlinearities of the robot which are not part of the rigid-body dynamics due to hydraulic tubes, friction, actuator dynamics, etc.

The model-based tracking control law determines the joint torques $\mathbf{u}$ necessary for following a desired trajectory $\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\ddot{\mathbf{q}}_d$ using a dynamics model while employing feedback in order to stabilize the system. For example, the dynamics model of the robot can be used as a feed-forward model that predicts the joint torques $\mathbf{u}_{\mathrm{FF}}$ required to perform the desired trajectory while a feedback term $\mathbf{u}_{\mathrm{FB}}$ ensures the stability of the tracking control with a resulting control law of $\mathbf{u} = \mathbf{u}_{\mathrm{FF}} + \mathbf{u}_{\mathrm{FB}}$. The feedback term can be a linear control law such as $\mathbf{u}_{\mathrm{FB}} = \mathbf{K}_p\mathbf{e} + \mathbf{K}_v\dot{\mathbf{e}}$, where $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$ denotes the tracking error and $\mathbf{K}_p, \mathbf{K}_v$ position-gain and velocity-gain, respectively. If an accurate model in the form of Equation (1) can be obtained, e.g., for negligible unknown nonlinearities, the resulting feedforward term $\mathbf{u}_{\mathrm{FF}}$ will largely cancel the robots nonlinearities [9].

1.2 Problem Statement

For complex robots such as humanoids or light-weight arms, it is often hard to model the system sufficiently well using the rigid body dynamics. Unknown nonlinearities $\boldsymbol{\epsilon}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ such as flexible hydraulic tubes, complex friction, gear boxes, etc, couple several degrees of freedom together and result

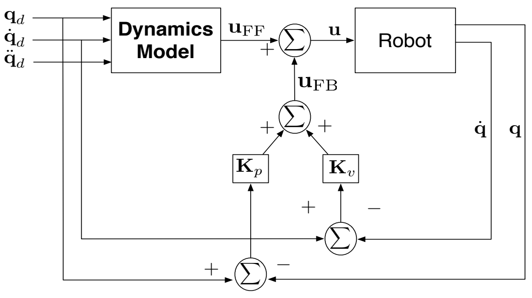


Figure 1: Schematic showing computed torque robot control

in highly altered dynamics. In particular, for the Barrett WAM several degrees of freedom are jointly actuated in a differential setup, and as a result, there is a complex friction function. Additionally, several spinning drives are in different reference frames from the actuated joint while only one can be measured resulting in effects such as reflective inertias.Thus, the dynamics can no longer be fully captured by standard rigid-body dynamics [10]. Such unknown nonlinearities can dominate the system dynamics and deteriorate the analytical model [11]. The resulting tracking error needs to be compensated using large gains [9]. High feedback gains prohibit compliant control and, thus, make the robot less safe for the environment while causing many practical problems such as actuator saturation, excitation of unmodeled dynamics, may result in large tracking errors in presence of noise, increase energy consumption, etc. To avoid high-gain feedback, it is essential to improve the accuracy of the dynamics model for predicting $\mathbf{u}_{\mathrm{FF}}$. Since $\mathbf{u}_{\mathrm{FF}}$ is a function of $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, it can be obtained with supervised learning using measured data. The resulting problem is a regression problem that can be solved by learning the mapping $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$ on sampled data [12–14] and, subsequently, using the resulting mapping for determining the feedforward motor commands. As trajectories and corresponding joint torques are sampled directly from the real robot, learning the mapping will include all nonlinearities and not only the ones described in the rigid-body model.

1.3 Challenges in Real-time Learning

Due to high computational complexity of nonlinear regression techniques, inverse dynamics models are frequently only learned offline for pre-sampled desired trajectories [12,14]. In order to take full advantage of a learning approach, online learning is absolute necessity as it allows the adaption to changes in the robot dynamics, load or the actuators. Furthermore, a training data set will never suffice for most robots with a large number of degrees of freedom and, thus, fast online learning is necessary if the trajectory leads to new parts of the state-space. However, for most real-time applications online model learning poses a difficult regression problem due to three constraints, i.e., firstly, the learning and prediction process should be very fast (e.g., learning needs to take place at a speed of 20-200Hz and prediction may take place at 200Hz up to 5kHz). Secondly, the learning system needs to be capable at dealing with large amounts of data (i.e., with data arriving at 200Hz, less than ten minutes of runtime will result in more than a million sampled data points). And, thirdly, the data arrives as a continuous stream, thus, the model has to be continuously adapted to new training examples over time.

2 Nonparametric Regression Methods

As any realistic inverse dynamics is a well-defined functional mapping of continuous, high-dimensional inputs to outputs of the same kind, we can view it as a regression problem. Given the input $\mathbf{x} \in \mathbb{R}^n$ and the target $\mathbf{y} \in \mathbb{R}^n$, the task of regression algorithms is to learn the mapping describing the relationship from input to target using samples. In this section, we will review the locally weighted projection regression (LWPR) and the Gaussian process regression (GPR). Locally-weighted projection regression is currently the standard real-time learning method in robot control applications and has been shown to scale into very high-dimensional domains [1, 2, 15]. However, it also requires skillful tuning of the meta parameters for the learning process in order to achieve competitive performance. Gaussian process regression on the other hand achieves a higher performance [4, 16] with very little tuning but also suffers of a significantly higher computational complexity.

2.1 Regression with LWPR

LWPR predicts the target values by approximating them with a combination of M individually weighted locally linear models. The weighted prediction $\hat{y}$ is then given by $\hat{y} = \mathbb{E}\{\bar{y}_k|\mathbf{x}\} = \sum_{k=1}^{M} \bar{y}_k p(k|\mathbf{x})$. According to the Bayesian theorem, the probability of the model k given query point $\mathbf{x}$ can be expressed as

$$p(k|\mathbf{x}) = \frac{p(k,\mathbf{x})}{p(\mathbf{x})} = \frac{p(k,\mathbf{x})}{\sum_{k=1}^{M} p(k,\mathbf{x})} = \frac{w_k}{\sum_{k=1}^{M} w_k}. \tag{2}$$

Hence, we have

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{M} w_k \bar{y}_k(\mathbf{x})}{\sum_{k=1}^{M} w_k}, \tag{3}$$

with $\bar{y}_k = \bar{\mathbf{x}}_k^T \hat{\boldsymbol{\theta}}_k$ and $\bar{\mathbf{x}}_k = [(\mathbf{x} - \mathbf{c}_k)^T, 1]^T$, where w_k is the weight or attributed responsibility of the model, $\hat{\boldsymbol{\theta}}_k$ contains the estimated parameters of the model and $\mathbf{c}_k$ is the center of the k-th linear model. The weight w_k determines whether a data point $\mathbf{x}$ falls into the region of validity of model k, similar to a receptive field, and is usually characterized with a Gaussian kernel

$$w_k = \exp\left(-\frac{1}{2}\left(\mathbf{x} - \mathbf{c_k}\right)^T \mathbf{D_k} \left(\mathbf{x} - \mathbf{c_k}\right)\right), \tag{4}$$

where $\mathbf{D_k}$ is a positive definite matrix called the distance matrix. During the learning process, both the shape of the receptive fields $\mathbf{D_k}$ and the parameters $\hat{\boldsymbol{\theta}}_k$ of the local models are adjusted such that the error between the predicted values and the observed targets is minimal. The regression parameter $\hat{\boldsymbol{\theta}}_k$ can be computed incrementally and online using the partial least squares method [1, 15]. The distance matrix $\mathbf{D_k}$ determines the size and shape of each local model; it can be updated incrementally using leave-one-out cross validation [2].

2.2 Regression with standard GPR

A powerful alternative for accurate function approximation in high-dimensional space is Gaussian process regression (GPR) [4]. Given a set of n training data points $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we would like to learn a function $f(\mathbf{x}_i)$ transforming the input vector $\mathbf{x}_i$ into the target value y_i given a model $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i is Gaussian noise with zero mean and variance σ_n^2 [4]. As a result, the observed targets can also be described by a Gaussian distribution $\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I}\right)$, where $\mathbf{X}$ denotes the set containing all input points $\mathbf{x}_i$ and $\mathbf{K}(\mathbf{X},\mathbf{X})$ the covariance matrix computed using a given covariance function. Gaussian kernels are probably the frequently used covariance functions [4] and are given by

$$k\left(\mathbf{x}_p, \mathbf{x}_q\right) = \sigma_s^2 \exp\left(-\frac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^T \mathbf{W} (\mathbf{x}_p - \mathbf{x}_q)\right) , \tag{5}$$

where σ_s^2 denotes the signal variance and $\mathbf{W}$ represents the widths of the Gaussian kernel. Other choices for possible kernels can be found in [4, 5]. The joint distribution of the observed target values and predicted value $f(\mathbf{x}_*)$ for a query point $\mathbf{x}_*$ is given by

$$\begin{bmatrix} \mathbf{y} \\ f(\mathbf{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I} & \boldsymbol{k}(\mathbf{X},\mathbf{x}_*) \\ \boldsymbol{k}(\mathbf{x}_*,\mathbf{X}) & k(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}\right) . \tag{6}$$

Conditioning the joint distribution yields the predicted mean value $f(\mathbf{x}_*)$ with the corresponding variance $V(\mathbf{x}_*)$

$$\begin{aligned} f(\mathbf{x}_*) &= \boldsymbol{k}_*^T \left(\mathbf{K} + \sigma_n^2\mathbf{I}\right)^{-1} \mathbf{y} = \boldsymbol{k}_*^T \boldsymbol{\alpha} , \\ V(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - \boldsymbol{k}_*^T \left(\mathbf{K} + \sigma_n^2\mathbf{I}\right)^{-1} \boldsymbol{k}_* , \end{aligned} \tag{7}$$

with $\boldsymbol{k}_* = \boldsymbol{k}(\mathbf{X}, \mathbf{x}_*)$, $\mathbf{K} = \mathbf{K}(\mathbf{X},\mathbf{X})$ and $\boldsymbol{\alpha}$ denotes the so-called prediction vector. The hyperparameters of a Gaussian process with Gaussian kernel are given by $\boldsymbol{\theta} = [\sigma_n^2, \sigma_f^2, \mathbf{W}]$ and remain the only open parameters. Their optimal value for a particular data set can be automatically estimated by maximizing the log marginal likelihood using standard optimization methods such as Quasi-Newton methods [4].

2.3 Comparison of these Approaches

The major drawback of LWPR is the currently necessary manual adjustment of the metaparameters[1] required for the update of the kernel width $\mathbf{D_k}$ and the regression vector $\hat{\boldsymbol{\theta}}_k$. These values are highly data dependent making it difficult to find an optimal set of parameters. Furthermore, as linear models are used in LWPR, a large number of local models may be required to achieve competitive prediction accuracy, since only relatively small regions can be fit using such linear models. Nevertheless, LWPR is the fastest and most task-appropriate real-time learning algorithm for inverse dynamics to date; currently, it can be considered the state of the art in real-time learning. On the other hand, GPR is more comfortable to apply while often achieving a higher prediction accuracy. All open parameters of a Gaussian process model, i.e., the hyperparameters $\boldsymbol{\theta}$, can be automatically adjusted by maximizing the marginal likelihood. As a result, GPR is relatively easy and flexible to use. However, the main limitation

[1]Current work by Ting et al. [17] indicates that automatic metaparameter estimation may be possible on the long run.

of standard GPR is that computational complexity scales cubically with the number of training examples. This drawback prevents standard GPR from applications which need large amounts of training data and require fast computation, e.g., model online learning for robot control.

3 Local Gaussian Process Regression

Model learning with GPR suffers from the expensive computation of the inverse matrix $(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}$ which yields a cost of $\mathcal{O}(n^3)$, see Equation (7). Inspired by locally weighted regression [1,2], we propose a method for speed-up the training and prediction process by partitioning the training data in local regions and learning an independent Gaussian process model (as given in Section 2.2) for each region. The number of data points in the local models is limited, where insertion and removal of data points can be treated in a principled manner. The prediction for a query point is performed by weighted average similar to LWPR. For partitioning and weighted prediction we use a kernel as similarity measure. Thus, our algorithm consists out of three stages: (i) clustering of data, i.e., insertion of new data points into the local models, (ii) learning of corresponding local models and (iii) prediction for a query point.

3.1 Partitioning of Training Data

Clustering input data can be performed efficiently using a similarity measure between the input point $\mathbf{x}$ and the centers of the respective local models. From a machine learning point of view, the similarity or proximity of data points can be defined in terms of a kernel. Kernel functions represent the dot product between two vectors in the feature space and, hence, naturally incorporate the similarity measure between data points. The clustering step described in this section results from the basic assumption that nearby input points are likely to have similar target values. Thus, training points that belong the same local region (represented by a center) are informative about the prediction for query points next to this local region.

A specific characteristic in this framework is that we take the kernel for learning the Gaussian process model as similarity measure w_i for the clustering process. If a Gaussian kernel is employed for learning the model, the corresponding measure will be

$$w_i\left(\mathbf{x},\mathbf{c}_i\right)=\exp\left(-\frac{1}{2}\left(\mathbf{x}-\mathbf{c}_i\right)^T\mathbf{W}\left(\mathbf{x}-\mathbf{c}_i\right)\right), \tag{8}$$

where $\mathbf{c}_i$ denotes the center of the i-th local model and $\mathbf{W}$ a diagonal matrix represented the kernel width. Note that this measure will result in the same weighting as in LWPR, see Equation (4). It should be emphasized that for learning the Gaussian process model any admissible kernel can be used. Thus, the similarity measure for the clustering process can be varied in many ways, and, for example, the commonly used Matern kernel [16] could be used instead of the Gaussian one. For the hyperparameters of the measure, such as $\mathbf{W}$ for Gaussian kernel, we use the *same* training approach as introduced in Section 2.2. Since the hyperparameters of a Gaussian process model can be achieved by likelihood optimization, it is straightforward to adjust the open parameters for the similarity measure. For example, we can

Algorithm 1: Partitioning the training data with incremental model learning.

```
Input: new data point $\{\mathbf{x}_{\text{new}}, y_{\text{new}}\}$.
for $i=1$ to number of local models do
  Compute proximity to the $i$-th local model:
    $w_i = k(\mathbf{x}_{\text{new}}, \mathbf{c}_i)$
end for
Take the nearest local model:
  $v = \max_i w_i$
if $v > w_{\text{gen}}$ then
  Insert $\{\mathbf{x}_{\text{new}}, y_{\text{new}}\}$ into the nearest local model:
    $\mathbf{X}_{\text{new}} = [\mathbf{X}, \mathbf{x}_{\text{new}}]$, $\mathbf{y}_{\text{new}} = [\mathbf{y}, y_{\text{new}}]$
  Update the corresponding center:
    $\mathbf{c}_{\text{new}} = \text{mean}(\mathbf{X}_{\text{new}})$
  Update the Cholesky matrix and the
  prediction vector of local model:
    Compute $\boldsymbol{l}$ and $l_*$
    Compute $\mathbf{L}_{\text{new}}$
    If the maximum number of data points is reached
    delete another point by permutation.
    Compute $\boldsymbol{\alpha}_{\text{new}}$ by back-substitution
else
  Create new model:
    $\mathbf{c}_{i+1} = \mathbf{x}_{\text{new}}$, $\mathbf{X}_{i+1} = [\mathbf{x}_{\text{new}}]$, $\mathbf{y}_{i+1} = [y_{\text{new}}]$
  Initialize of new Cholesky matrix $\mathbf{L}$ and
  new prediction vector $\boldsymbol{\alpha}$.
end if
```

subsample the available training data and, subsequently, perform the standard optimization procedure.

After computing the proximity between the new data point $\mathbf{x}_{new}$ and all available centers, the data point will be included to the *nearest* local model, i.e., the one with the maximal value of w_i. As the data arrives incrementally over time, a new model with center $\mathbf{c}_{i+1}$ is created if all similarity measures w_i fall below a threshold w_{gen}. The new data point is then used as new center $\mathbf{c}_{i+1}$ and, thus, the number of local models will increase if previously unknown parts of the state space are visited. When a new data point is assigned to a particular i-th model, i.e., $\max_i w_i(\mathbf{x}) > w_{\text{gen}}$ the center $\mathbf{c}_i$ will be updated to the mean of corresponding local data points.

Algorithm 2: Prediction for a query point.

```
Input: query data point $\mathbf{x}$, $M$ .
Determine $M$ local models closest to $\mathbf{x}$.
for $i = 1$ to $M$ do
    Compute proximity to the $i$-th local model:
        $w_i = k(\mathbf{x}, \mathbf{c}_i)$
    Compute local prediction using the $k$-th local model:
        $\bar{y}_i(\mathbf{x}) = \mathbf{k}_i(\mathbf{x})^T \boldsymbol{\alpha}_i$
end for
Compute weighted prediction using $M$ local models:
    $\hat{y}(\mathbf{x}) = \sum_{i=1}^{M} w_i \bar{y}_i(\mathbf{x}) / \sum_{k=1}^{M} w_i$ .
```

3.2 Incremental Update of Local Models

During online learning, we have to deal with an endless stream of data (e.g., at a 500 Hz sampling rate we get a new data point every 2 ms and have to treat 30 000 data points per minute). In order to cope with the real-time requirements, the maximal number of training examples needs to limited so that the local models do not end up with the same complexity as a standard GPR regression. Since the number of acquired data points increases continuously over time, we can enforce this limit by incrementally deleting old data points when newer and better ones are included. Insertion and deletion of data points can be achieved using first order principles, for example, maximizing the information gain while staying within a budget (e.g., the budget can be a limit on the number of data points). Nevertheless, while the update of the target vector $\mathbf{y}$ and input matrix $\mathbf{X}$ can be done straightforwardly, the update of the covariance matrix (and implicitly the update of the prediction vector $\boldsymbol{\alpha}$, see Equation (7)) is more complicated to derive and requires thorough analysis given here.

The prediction vector $\boldsymbol{\alpha}$ can be updated incrementally by directly adjusting the Cholesky decomposition of the Gram matrix $(\mathbf{K} + \sigma_n^2 \mathbf{I})$ as suggested in [18]. For doing so, the prediction vector can be rewritten as $\mathbf{y} = \mathbf{L}\mathbf{L}^T \boldsymbol{\alpha}$, where the lower triangular matrix $\mathbf{L}$ is a Cholesky decomposition of the Gram matrix. Incremental insertion of a new point is achieved by adding an additional row to the matrix $\mathbf{L}$.

Proposition 3.1 *If $\mathbf{L}$ is the Cholesky decomposition of the Gram matrix $\mathbf{K}$ while $\mathbf{L}_{new}$ and $\mathbf{K}_{new}$ are obtained by adding additional row and column, such that*

$$\mathbf{L}_{new} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \boldsymbol{l}^T & l_* \end{bmatrix}, \quad \mathbf{K}_{new} = \begin{bmatrix} \mathbf{K} & \boldsymbol{k}_{new}^T \\ \boldsymbol{k}_{new} & k_{new} \end{bmatrix}, \tag{9}$$

with $\boldsymbol{k}_{new} = \boldsymbol{k}(\mathbf{X}, \mathbf{x}_{new})$ and $k_{new} = k(\mathbf{x}_{new}, \mathbf{x}_{new})$, then $\boldsymbol{l}$ and l_* can be computed by solving

$$\mathbf{L}\boldsymbol{l} = \boldsymbol{k}_{new} \tag{10}$$

$$l_* = \sqrt{k_{new} - \|\boldsymbol{l}\|^2} \tag{11}$$

Proof Multiply out the equation $\mathbf{L}_{new}\mathbf{L}_{new}^T = \mathbf{K}_{new}$ and solve for $\boldsymbol{l}$ and l_*. ∎

Since $\mathbf{L}$ is a triangular matrix, $\boldsymbol{l}$ can be determined from Equation (10) by substituting it back in after computing the kernel vector $\boldsymbol{k}_{new}$. Subsequently, l_* and the new prediction vector $\boldsymbol{\alpha}_{new}$ can be determined from Equation (11), where $\boldsymbol{\alpha}_{new}$ can be achieved by twice back-substituting while solving $\mathbf{y}_{new} = \mathbf{L}_{new}\mathbf{L}_{new}^T\boldsymbol{\alpha}_{new}$. If the maximal number of training examples is reached, an old data point has to be deleted every time when a new point is being included. The deletion of the m-th data point can be performed efficiently using a permutation matrix $\mathbf{R}$ and solving $\mathbf{y}_{new} = \mathbf{R}\ \mathbf{L}_{new}\mathbf{L}_{new}^T\mathbf{R}\ \boldsymbol{\alpha}_{new}$, where $\mathbf{R} = \boldsymbol{I} - (\boldsymbol{\delta}_m - \boldsymbol{\delta}_n)(\boldsymbol{\delta}_m - \boldsymbol{\delta}_n)^T$ and $\boldsymbol{\delta}_i$ is a zero vector whose i-th element is one [18]. In practice, the new data point is inserted as a first step to the *last* row (n-th row) according to Equation (9) and, subsequently, the m-th data point is removed by adjusting $\mathbf{R}$. The partitioning and learning process is summarized in Algorithm 1. The incremental Cholesky update is very efficient and can be performed in a numerically stable manner as discussed in detail in [18].

Due to the Cholesky update formulation, the amount of computation for training can be limited due to the incremental insertion and deletion of data points. The main computational cost for learning the local models is dominated by the incremental update of the Cholesky matrix which yields $\mathcal{O}(N_l^2)$, where N_l presents the number of data points in a local model. Importantly, N_l can be set in accordance with the computational power of the available real-time computer system.

3.3 Prediction using Local Models

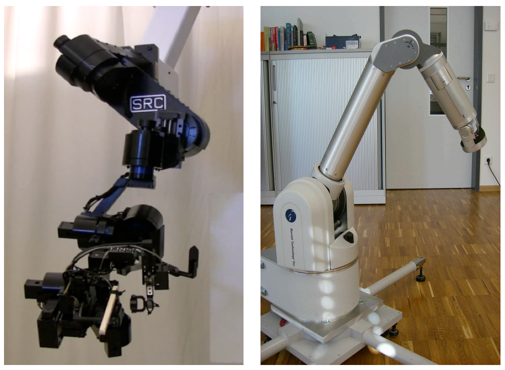
The prediction for a mean value $\hat{y}$ is performed using weighted averaging over M local GP predictions $\bar{y}_i$ for a query point $\mathbf{x}$ similar to LWPR. The weighted prediction $\hat{y}$ is then given by

$$\hat{y} = \frac{\sum_{i=1}^{M} w_i \bar{y}_i}{\sum_{i=1}^{M} w_i}\,. \tag{12}$$

Thus, each local GP prediction $\bar{y}_i = \boldsymbol{k}(\mathbf{X}_i, \mathbf{x})^T\boldsymbol{\alpha}_i$ is additionally weighted by the similarity $w_i\left(\mathbf{x}, \mathbf{c}_i\right)$ between the corresponding center $\mathbf{c}_i$ and the query point $\mathbf{x}$. The search for M local models can be quickly done by evaluating the proximity between the query point $\mathbf{x}$ and all model centers $\mathbf{c}_i$. The prediction procedure is summarized in Algorithm 2.

3.4 Relation to Previous Work

Many attempts have been made to reduce the computational cost of GPR, mostly of them follow either the strategy of creating (i) sparse Gaussian processes (SGP) or follow a (ii) mixture of experts (ME) approach. In a SGP, the whole input space is approximated using a smaller set of "inducing inputs" [6, 19, 20]. Here, the difficulty lies in the choice of the appropriate set of inducing inputs that essentially summarizes the original input space [4]. In contrast to SGP, the ME approach divides the whole input space in smaller subspaces employing a gating network which activates responsible Gaussian process experts [21, 22]. Thus, the computational cost of the matrix inversion is significantly reduced due to a much smaller number of data points within an expert. The ME performance depends largely on the number of experts for a particular data set. To reduce the impact of this problem, [21] allows the



(a) **SARCOS arm** (b) **Barrett WAM**

Figure 2: Robot arms used for data generation and experiments.

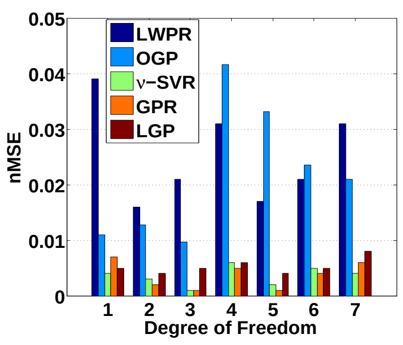
learning process to infer the required number of experts for a given data set using a Dirichlet process to determine the appropriate gating network. The proposed algorithm has approximately a complexity of $\mathcal{O}(n^3/M)$ for training and $\mathcal{O}(n^2 d)$ for adapting the gating network parameters, where M denotes the number of experts, n the total number of training data and d the dimension of input vector.

The presented LGP approach is loosely related to the ME approximation. However, the gating network requires competition between the models for the data points while the locality approach allows cooperation [23]. Particularly important is the fact that we re-use the kernel measure as similarity measure which results in two advantages, firstly, the metric parameters can be derived directly by optimization procedure which makes it more comfortable and flexible for using. Secondly, the evaluation of the metric can be performed very fast enabling it for real-time application. However, it shows that clustering in higher dimensional space is not always straightforward to perform with the kernel similarity metric. In our experience, partitioning of training data can be done quite well up to 20 dimensions. Since we can localize the training data in much lower spaces than learning the model, this obstacle can often be circumvented. We will discuss this issue in more detail in Section 4.

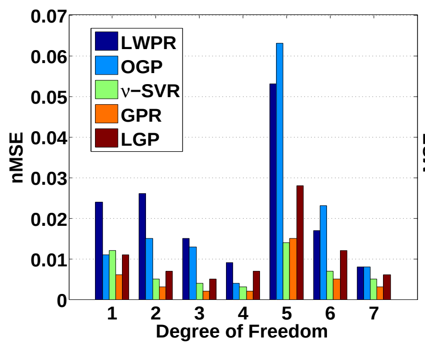
Compared with LWPR, one major advantage is that we use Gaussian process model for training the local models instead of linear models. Thus, we need significantly fewer local models to be competitive in learning accuracy. Gaussian process models have also shown to generalize the training data well and are easier to train as the open parameter can be obtained straightforwardly from the log marginal likelihood. However, a major drawback in comparison to LWPR is that the more complex models result in an increased computational cost.

4 Learning Inverse Dynamics

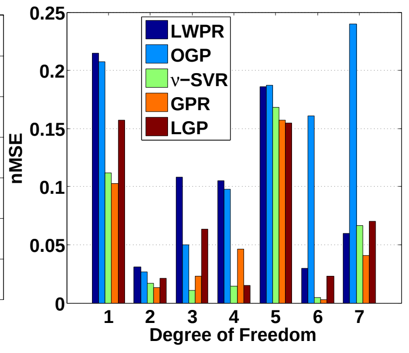
Learning models for control of high-dimensional systems in real-time is a difficult endeavor and requires extensive evaluation. For this reason, we have evaluated our algorithm using high-dimensional data taken from two real robots, e.g., the 7 degree-of-freedom (DoF) anthropomorphic SARCOS master



(a) **Approximation Error on SL data (SARCOS model)**



(b) **Approximation Error on SARCOS data**



(c) **Approximation Error on Barrett WAM data**

Figure 3: The approximation error is represented by the normalized mean squared error (nMSE) for each DoF (1–7) and shown for (a) simulated data from physically realistic SL simulation, (b) real robot data from an anthropomorphic SARCOS master arm and (c) measurements from a Barrett WAM. In all cases, LGP outperforms LWPR and OGP in learning accuracy while being competitive to ν-SVR and standard GPR. The small variances of the output targets in the Barrett data results in a nMSE that is a larger scale compared to SARCOS; however, this increase has no practical meaning and only depends on the training data.

arm and 7-DoF Barrett WAM both shown in Figure 2, as well as physically realistic simulation using SL [24]. We compare the learning performance of LGP with the state-of-the-art in nonparametric regression, e.g., LWPR, ν-SVR, standard GPR and online Gaussian Process Regression (OGP) in the context of approximating inverse robot dynamics. For evaluating ν-SVR and GPR, we have employed the libraries [25] and [26], respectively. The code for LGP contained also parts of the library [26].

4.1 Learning Accuracy Comparison

For comparing the prediction accuracy of our proposed method in the setting of learning inverse dynamics, we use three data sets, (i) SL simulation data (SARCOS model) as described in [14] (14094 training points and 5560 test points), (ii) data from the SARCOS master arm (13622 training points and 5500 test points) [2] as well as (iii) a data set generated from our Barrett arm (13572 training points, 5000 test points). Given samples $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as input, where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ denote the joint angles, velocity and acceleration, respectively, and using the corresponding joint torques $\mathbf{y} = [\mathbf{u}]$ as targets, we have a well-defined, proper regression problem. The considered seven degrees of freedom (DoF) robot arms result in 21 input dimensions (i.e., for each joint, we have an angle, a velocity and an acceleration) and seven target or output dimensions (i.e., a single torque for each joint). The robot inverse dynamics model can be estimated separately for each DoF employing LWPR, ν-SVR, GPR, OGP and LGP, respectively.

The training examples for LGP can be partitioned either in the same input space where the local models are learned or in a subspace that has to be physically consistent with the approximated function.

In the following, we localize the data depending on the position of the robot. Thus, the partitioning of training data is performed in a seven dimensional space (i.e., consisting of the seven joint angles). The localization should be performed in a low dimensional space, since with increasing input dimensions the partitioning of data may be difficult having negative effects on the learning performances. After determining the similarity metric w_k for all k local models in the partitioning space, the input point will be assigned to the *nearest* local model, i.e., the local model with the maximal value of distance measure w_k. For computing the localization, we will use the Gaussian kernel as given in Equation (5) and the corresponding hyperparameters are optimized using a subset of the training set.

Note that the choice of the limit value w_{gen} during the partitioning step is crucial for the performance of LGP and, unfortunately, is an open parameter requiring manual tuning. If w_{gen} is too small, a large number of local models will be generated with small number of training points. As these small models receive too little data for a stable GPR, they do not generalize well to unknown neighboring regions of the state space. If w_{gen} is large, the local models will include too many data points which either results in over-generalization or, if the number of admitted data points is enlarged as well, it will increase the computational complexity. Here, the training data is clustered in about 30 local regions ensuring that each local model has a sufficient amount of data points for high accuracy (in practice, roughly a hundred data points for each local model suffice) while having sufficiently few that the solution remains feasible in real-time (e.g., on the test hardware, an Intel Core Duo at 2GHz, that implies the usage of up to a 1000 data points per local model). On average, each local model includes approximately 500 training examples, i.e., some models will not fill up while others actively discard data. This small number of training data points enables a fast training for each local model using the previously described fast Cholesky matrix updates.

Figure 3 shows the normalized mean squared error (nMSE) of the evaluation on the test set for each of the three evaluated scenarios, i.e., a physically realistic simulation of the SARCOS arm in Figure 3 (a), the real anthropomorphic SARCOS master arm in Figure 3 (b) and the Barrett WAM arm in Figure 3 (c). Here, the normalized mean squared error is defined by nMSE = Mean squared error/Variance of target. During the prediction on the test set using LGP, we take the most activated local models, i.e., the ones which are next to the query point.

When observing the approximation error on the test set shown in Figure 3(a-c), it can be seen that LGP generalizes well to the test data during prediction. In all cases, LGP outperforms LWPR and OGP while being close in learning accuracy to of the offline-methods GPR and ν-SVR. The mean prediction for GPR is determined according to Equation (7) where we pre-computed the prediction vector $\boldsymbol{\alpha}$ from training data. When a query point appears, the kernel vector $\mathbf{k}_*^T$ is evaluated for this particular point.

4.2 Comparison of Computation Speed for Prediction

The computation requirements of kernel-based regression can even be problematic for prediction in real-time, thus, it is an essential component of the LGP that it results in a substantial reduction of prediction

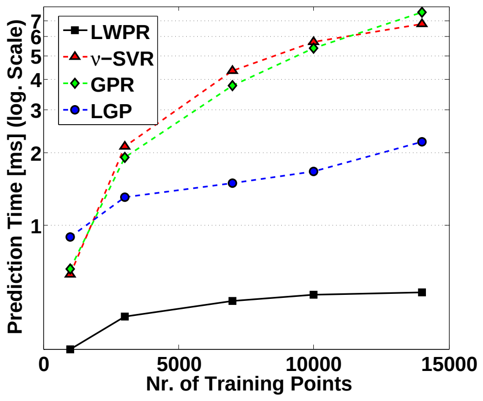


Figure 4: Average time in millisecond needed for prediction of 1 query point. The computation time is plotted logarithmically with respect to the number of training examples. The time as stated above is the required time for prediction of all 7 DoF performed sequentially. Here, LWPR presents the fastest method due to simple regression models. Compared to global regression methods such as standard GPR and ν-SVR, local GP makes significant improvement in term of prediction time. For this experiment, 3 local models are taken each time for prediction with LGP.

latency rendering online prediction feasible even for large data sets. The duration of a prediction of the LGP significantly lower than the one GPR and ν-SVR as only a small amount of local models in the vicinity of the current input data is needed during prediction. Thus, the complexity of the prediction operation is $\mathcal{O}(n)$ for a standard GPR (ν-SVR does not differ in complexity), it will become $\mathcal{O}(NM)$ for LGP, where n denotes the total number of training points, M number of local models used in the prediction and N number of data points in a local model. Note that usually $n >> NM$. The comparison of prediction speed is shown in Figure 4. Here, we train LWPR, ν-SVR, GPR and LGP on 5 different data sets with increasing training examples (1065, 3726, 7452, 10646 and 14904 data points, respectively). Subsequently, using the trained models we compute the average time needed to make a prediction for a query point for all 7 DoF. For LGP, we take the same number of local models in the vicinity for prediction as in last experiment. Since assuming a minimal prediction rate at 100 Hz (10 ms) in order to ensure system stability, data sets with more than 15000 points cannot be used with standard GPR or ν-SVR on an Intel Core Duo at 2GHz due to high computation demands for the prediction. In recent time, there are also approaches to speed up the prediction time for standard GPR [27,28]. In [27], for example, KD-trees are applied to find training data points next to the query point for prediction.

The results given in Figure 4 show that the computation time requirements of ν-SVR and GPR rises very fast with the size of training data set as expected. LWPR remains the best method in terms of computational complexity only increasing at a very low pace with the number of data points. However, as shown in Figure 4, the cost for LGP is significantly lower than the one for ν-SVR and GPR and increases at a much lower rate. The LGP prediction latency can be bounded by setting the number of local models needed for prediction, i.e., the parameter M. In practice, we need around 1000 data points in the neighborhood of the query point for prediction resulting in the usage of 2 or 3 local models. As shown by the results, LGP represents a compromise between learning accuracy and computational complexity. For large data sets (e.g., more than 5000 training examples), LGP reduces the prediction cost considerably in comparison to standard methods while still having a good learning performance.

5 Application in Model-based Robot Control

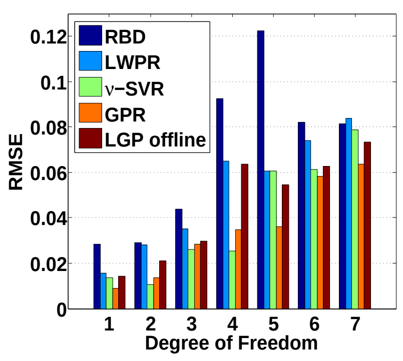
In this section, we use the inverse dynamics models learned in Section 4.1 for a model-based tracking control task [8] in the setting shown in Figure 1. Here, the model is used for predicting the feedforward torques $\mathbf{u}_{\mathrm{FF}}$ necessary to execute a given desired trajectory $[\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d]$. First, we compare standard rigid-body dynamics (RBD) models with several models learned offline on training data sets. For this offline learning comparison, we use LWPR, ν-SVR, standard GPR as well as our LGP as compared learning methods. We show that our LGP is competitive when compared with its alternatives. Second, we demonstrate that online learning is highly beneficial. During online learning, the local GP models are updated in real-time, and the online improvement during a tracking task outperforms the fixed offline model in comparison. Our goal is to achieve compliant tracking in robots without exception handling or force sensing but purely based on using low control gains. Our control gains are three orders of magnitude smaller than the manufacturers in the experiments and we can show that using good, learned inverse dynamics models we can still achieve compliant control. The accuracy of the model has a stronger effect on the tracking performance in this setting and, hence, a more precisely learned model will also results in a significantly lower tracking error.

5.1 Tracking using Offline Trained Models

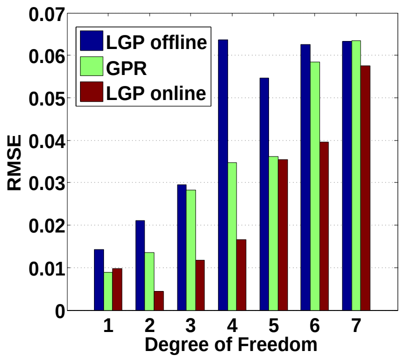
For comparison with the learned models, we also compute the feedforward torque using rigid-body (RB) formulation which is a common approach in robot control [8]. The control task is performed in real-time on the Barrett WAM, as shown in Figure 2. As desired trajectory, we generate a test trajectory which is similar to the one used for learning the inverse dynamics models in Section 4.1. Figure 5 (a) shows the tracking errors on test trajectory for 7 DoFs. The error is computed as root mean square error (RMSE) which is a frequently used measure in time series prediction and tracking control. Here, LGP provides a competitive control performance compared to GPR while being superior to LWPR and the state-of-the art rigid-body model.

5.2 Online Learning of Inverse Dynamics Models

In this section, we show that the LGP is capable of online adaptation while being used for predicting the required torques. Since the number of training examples in each local model is limited, the update procedure is sufficiently fast for real-time application. For doing so, we employ the joint torques $\mathbf{u}$ and the resulting robot trajectories $[\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as samples which are added to the LGP models online as



(a) **Tracking errors on Barrett: comparison of offline-learned models**



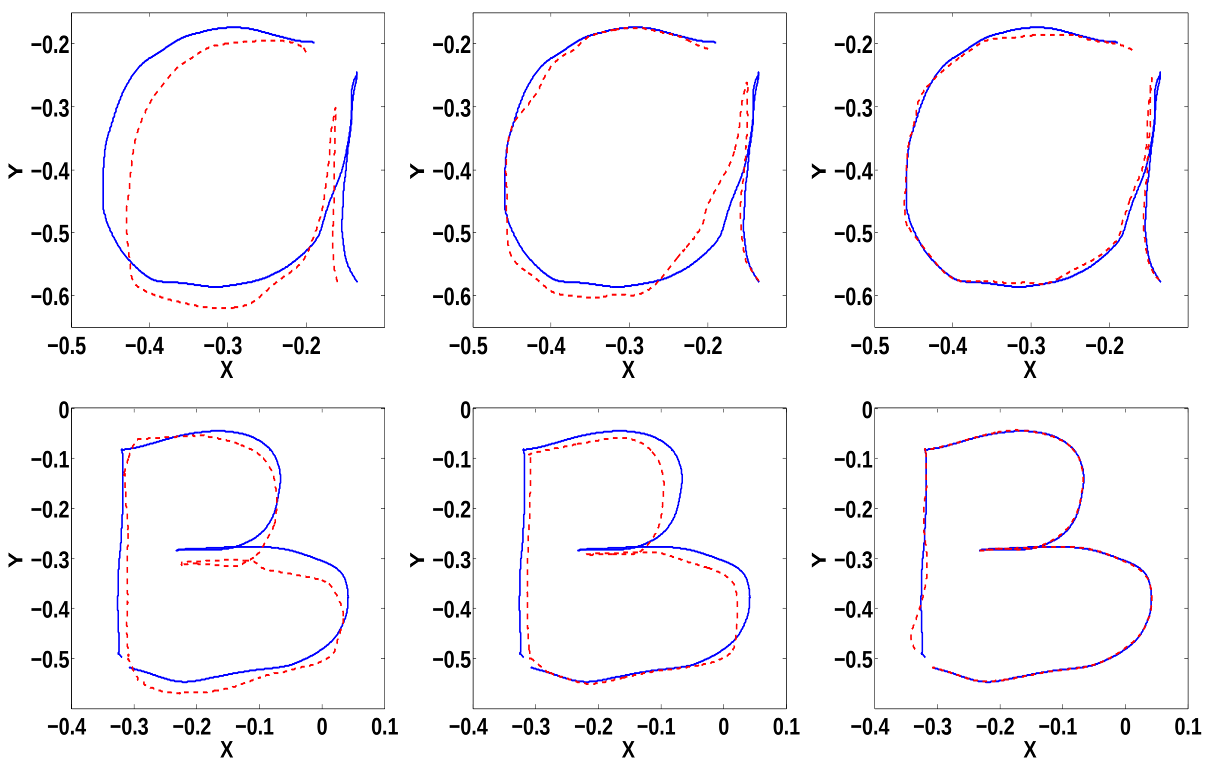
(b) **Tracking errors on Barrett: full GPR vs. offline and online-learned models with LGP**

Figure 5: **(a)** and **(b)** show the tracking errors (RMSE) on the Barrett WAM. For offline-learned models, LGP is competitive with full GPR and ν-SVR while being better than LWPR and rigid-body model. When employing online-updates, LGP can largely improve the tracking results outperforming the offline-learned models using full GPR. The reported results are computed for a test trajectory executed on the robot.

described in Section 3.2. New data points are added to the local models until these fill up and, once full, new points replace previously existing data points. The insertion of new data point is performed with information gain [29] while for the deletion we randomly take an old point from the corresponding local model. A new data point is inserted to the local model, if its information gain is larger than a given threshold value. In practice, this value is set such that the model update procedure can be maintained in real-time (the larger the information gain threshold, the more updates will be performed). Figure 5 (b) shows the tracking error after online learning with LGP in comparison with offline learned models. It can be seen that the errors are significantly reduced for LGP with online updates when compared to both standard GPR and LGP with offline learned models.

Here, we create a more complex test case for tracking with inverse dynamics models, i.e., we take the Barrett WAM by the end-effector and guide it along several trajectories which are subsequently used both in learning and control experiments. In order to make these trajectories straightforward to comprehend for humans, we draw all 26 characters of the alphabet in an imaginary plane in task space. An illustration for this data generation process is shown in Figure 7 (a). During the imagined writing, the joint trajectories are sampled from the robot. Afterwards, it will attempt to reproduce that trajectory, and the reproductions can be used to generate training data. Subsequently, we used several characters as training examples (e.g., characters from **D** to **O**) and others, e.g., **A** and **B**, as test examples. This setup results in a data set with 10845 samples for training and 1599 for testing.

Similar as in Section 4.1, we learn the inverse dynamics models using joint trajectories as input and joint torques as targets. The robot arm is then controlled to perform the joint-space trajectories



(a) **Tracking test-characters using rigid-body model**

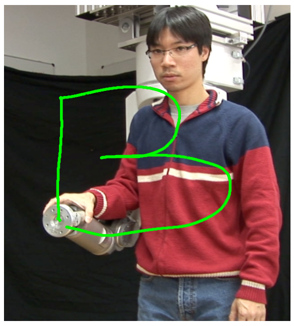
(b) **Tracking test-characters using offline-learned GP model**

(c) **Tracking test-characters after online-learning with LGP**

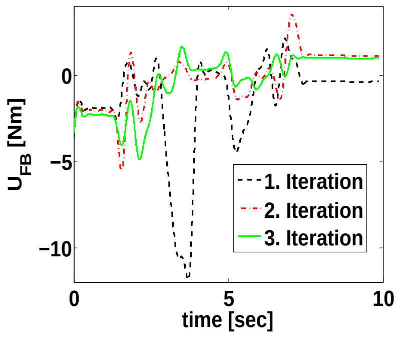
Figure 6: Compliant tracking performance on Barrett WAM for two test characters **A** and **B** where the controlled trajectory lies in joint-space while our visualization is in task space for improved comprehensibility. We compare the corresponding rigid body model, an offline trained GP model and an online learning LGP. The thick, blue line denotes the desired trajectory, while the dashed, red line represents the robot trajectory during the compliant tracking task. The results indicate that online learning with LGP outperforms the offline-learned model using full GPR as well as the rigid-body dynamics.

corresponding to the test characters using the learned models. For LGP, we additionally show that the test characters can be learned online by updating the local models, as described in Section 5. The Figure 6 shows the tracking results using online-learning with LGP in comparison to the offline trained model with standard GPR and a traditional rigid body model. It can be observed that the offline trained models (using standard GPR) can generalize well to unknown characters often having a better tracking performance than the rigid-body model. However, the results can be improved even further if the dynamics model is updated online – as done by LGP. The LGP results are shown in Figure 6 and are achieved after three trials on the test character.

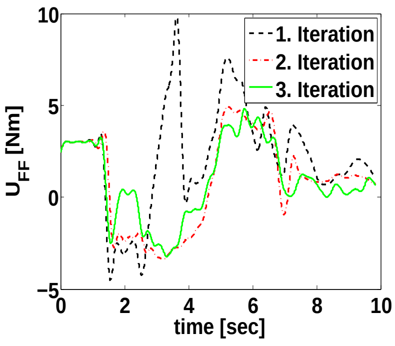
In practice, it is shown that a good tracking performance can already be achieved after 2 iterations of the unknown characters. During the first iteration, the tracking error is large (due to suboptimal prediction in presence of unknown trajectory) resulting in a large correcting feedback term $\mathbf{u}_{\mathrm{FB}}$, see



(a) **Character acquisition**



(b) **Magnitude of the feedback torque**



(c) **Magnitude of the feedforward torque**

Figure 7: This figure illustrates **(a)** Data generation for the learning task. **(b)** and **(c)** show as example the feedback $\mathbf{u}_{\mathrm{FB}}$ and feedforward $\mathbf{u}_{\mathrm{FF}}$ term of the 1. DoF after each iteration of the test character, e.g., **A**. The feedback term degreases gradually, as the model learns to make the required feedforward torques. The feedforward toques $\mathbf{u}_{\mathrm{FF}}$ does not change significantly after 2 iterations. For running through the complete trajectory, e.g., **A**, about 8 seconds are necessary.

Figure 7 (b). In the following iterations, the feedback term gradually degreases, as the model 'learns' to make a correct prediction, i.e, the optimal feedforward torque $\mathbf{u}_{\mathrm{FF}}$ required for the given characters, by using the online sampled input-torques $\mathbf{u}$ as learning target. The feedforward torque $\mathbf{u}_{\mathrm{FF}}$ converges already after 2 or 3 iterations, as shown in Figure 7 (c), i.e., after that the feedforward torques do not change significantly.

6 Conclusion

The local Gaussian process regression LGP combines the strength of fast computation as in local regression with the potentially more accurate kernel regression methods. As a result, we obtain a realtime capable regression method which is relatively easy to tune and works well in robot application. When compared to locally linear methods such as LWPR, the LGP achieves higher learning accuracy while having less computational cost compared to state of the art kernel regression methods such as GPR and ν-SVR. The reduced complexity allows the application of the LGP for online model learning which is necessary for realtime adaptation of model errors or changes in the system. Model-based tracking control using online learned LGP models achieves a superior control performance for low gain control in comparison to rigid body models as well as to offline learned models.

Future research will focus on several important extensions such as finding kernels which are most appropriate for clustering and prediction, and how the choice of a similarity can affect the LGP performance. Partitioning in higher dimension space is still a challenging problem, a possible solution is to perform dimensionality reduction during the partitioning step. It is also interesting to investigate how to infer an optimal value for w_{gen} from data. Furthermore, alternative criteria for insertion and deletion of data points need to be examined more closely. This operation is crucial for online learning as not every new data point is informative for the current prediction task, and on the other hand deleting an old but informative data point may degrade the performance. It also interesting to investigate further applications of the LGP in humanoid robotics with 35 of more DoFs and learning other types of the control such as operational space control.

REFERENCES

[1] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "Scalable techniques from nonparametric statistics for real-time robot learning," *Applied Intelligence*, pp. 49–60, 2002.

[2] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Computation*, no. 12, pp. 2602–2634, 2005.

[3] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Local gaussian processes regression for real-time model-based robot control," in *International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, 2008.

[4] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning.* Massachusetts Institute of Technology: MIT-Press, 2006.

[5] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond.* Cambridge, MA: MIT-Press, 2002.

[6] L. Csato and M. Opper, "Sparse online gaussian processes," *Neural Computation*, 2002.

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

[8] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Prentice Hall, 2004.

[9] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Dynamics and Control.* New York: John Wiley and Sons, 2006.

[10] W.Townsend, *Inertial Data for the Whole-Arm-Manipulator (WAM) Arm*, 2007.

[11] J. Nakanishi, J. A. Farrell, and S. Schaal, "Composite adaptive control with locally weighted statistical learning," *Neural Networks*, no. 1, pp. 71–90, 2005.

[12] E. Burdet, B. Sprenger, and A. Codourey, "Experiments in nonlinear adaptive control," in *International Conference on Robotics and Automation (ICRA)*, vol. 1, Albuquerque, NM, USA, 1997, pp. 537–542.

[13] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "Real-time robot learning with locally weighted statistical learning," in *International Conference on Robotics and Automation*, San Francisco, CA, USA, 2000.

[14] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Computed torque control with nonparametric regression models," in *Proceedings of the 2008 American Control Conference (ACC 2008)*, Seattle, Washington, USA, 2008.

[15] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space," in *International Conference on Machine Learning, Proceedings of the Sixteenth Conference*, Bled, Slovenia, 2000.

[16] M. Seeger, "Gaussian processes for machine learning," *International Journal of Neural Systems*, 2004.

[17] J. Ting, M. Kalakrishnan, S. Vijayakumar, and S. Schaal, "Bayesian kernel shaping for learning control," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2008.

[18] M.Seeger, "Low rank update for the cholesky decomposition," University of California at Berkeley, Tech. Rep., 2007. [Online]. Available: http://www.kyb.tuebingen.mpg.de/bs/people/seeger/

[19] J. Q. Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, 2005.

[20] D. H. Grollman and O. C. Jenkins, "Sparse incremental learning for interactive robot control policy estimation," in *IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, 2008.

[21] C. E. Rasmussen and Z. Ghahramani, "Infinite mixtures of gaussian process experts," in *Advances in Neural Information Processing Systems*, Cambridge, MA, USA, 2002.

[22] E. Snelson and Z. Ghahramani, "Local and global sparse gaussian process approximations," San Juan, Puerto Rico, 2007.

[23] S. Schaal. and C. G. Atkeson, "From isolation to cooperation: An alternative of a system of experts," in *Advances in Neural Information Processing Systems*, Denver, CO, USA, 1996.

[24] S. Schaal, "The SL simulation and real-time control software package," university of southern california, Tech. Rep., 2006. [Online]. Available: http://www-clmc.usc.edu/publications/S/schaal-TRSL.pdf

[25] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[26] M. Seeger, *LHOTSE: Toolbox for Adaptive Statistical Model*, 2007, http://www.kyb.tuebingen.mpg.de/bs/people/seeger/lhotse/.

[27] Y. Shen, A. Y. Ng, and M. Seeger, "Fast gaussian process regression using kd-trees," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2005.

[28] T. Suttorp and C. Igel, "Approximation of gaussian process models after training," in *European Symposium on Artificial Neural Networks*, Bruges, Belgium, 2008.

[29] M.Seeger, "Bayesian gaussian process models: Pac-bayesian generalisation error bounds and sparse approximations," Ph.D. dissertation, University of Edinburgh, 2005.