2008 IEEE/RSJ International Conference on Intelligent Robots and Systems
Acropolis Convention Center
Nice, France, Sept, 22-26, 2008

Local Gaussian Process Regression for Real-time Model-based Robot Control

Duy Nguyen-Tuong, Jan Peters
Max Planck Institute for Biological Cybernetics, Spemannstraße 38, 72076 Tübingen

***Abstract*— High performance and compliant robot control requires accurate dynamics models which cannot be obtained analytically for sufficiently complex robot systems. In such cases, machine learning offers a promising alternative for approximating the robot dynamics using measured data. This approach offers a natural framework to incorporate unknown nonlinearities as well as to continually adapt online for changes in the robot dynamics. However, the most accurate regression methods, e.g. Gaussian processes regression (GPR) and support vector regression (SVR), suffer from exceptional high computational complexity which prevents their usage for large numbers of samples or online learning to date. Inspired by locally linear regression techniques, we propose an approximation to the standard GPR using local Gaussian processes models inspired by [1], [2]. Due to reduced computational cost, local Gaussian processes (LGP) can be applied for larger sample-sizes and online learning. Comparisons with other nonparametric regressions, e.g. standard GPR, ν-SVR and locally weighted projection regression (LWPR), show that LGP has higher accuracy than LWPR and close to the performance of standard GPR and ν-SVR while being sufficiently fast for online learning.**

I. Introduction

Model-based control, e.g. computed torque control [3] 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 which is governed by the system dynamics [3]

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

where $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ are joint angles, velocities and accelerations of the robot, $\mathbf{u}$ denotes the inputs to the system, $\mathbf{M}(\mathbf{q})$ 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 generated joint torques or motor commands $\mathbf{u}$ such that the robot follows a desired trajectory $\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\ddot{\mathbf{q}}_d$. The dynamics model of the robot can be used as a feed-forward model to predict the joint torques $\mathbf{u}_{\mathrm{FF}}$ required to follow the given desired trajectory while a feedback term $\mathbf{u}_{\mathrm{FB}}$ ensures the stability of the trajectory 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, the rigid-body model for negligible unknown nonlinearities, the resulting feedforward term $\mathbf{u}_{\mathrm{FF}}$ will largely cancel the robots nonlinearities [3].

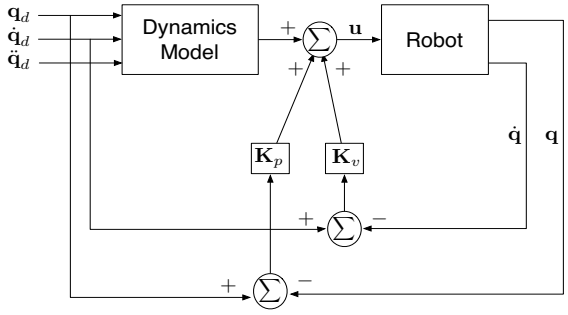


Fig. 1: Schematic showing computed torque robot control

In real, physical systems, there are frequently many unknown nonlinearities $\boldsymbol{\epsilon}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ such as hydraulic tubes, complex friction, gear boxes, etc, which couple several degrees of freedom together. Such unknown nonlinearities can dominate the system dynamics which deteriorates the model [4] and the resulting tracking error needs to be compensated using large gains [3]. High feedback gains prohibit compliant control making the robot less safe for the environment and causing many problems such as saturation of the actuators, excitation of the unmodeled dynamics, large error in presence of noise, large energy consumption etc., and thus should be avoided in practice. To avoid high-gain feedback, a more accurate dynamics model for computation of $\mathbf{u}_{\mathrm{FF}}$ is necessary. Since $\mathbf{u}_{\mathrm{FF}}$ is a function of $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, it can be obtained using supervised learning. The resulting problem is a regression problem which can be solved by learning the mapping $\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}} \rightarrow \mathbf{u}$ on sampled data [5], [6] 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.

Due to high computational complexity of nonlinear regression techniques, dynamics models are frequently only learned offline for pre-sampled desired trajectories [5], [6]. In order to take full advantage of a learning approach, online learning is absolute necessity as it allows the adaption due 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. Recently, there has been a lot of progress in online learning

resulting into sophisticated online learning techniques such as locally weighted projection regression (LWPR) [1], [7]. Here, the dynamics are approximated by locally linear functions covering the relevant state-space and online learning became computationally feasible due to low computational demands of the local projection regression which can be performed in real-time.

From our experience, the major drawback of LWPR is the required manual tuning of many highly data-dependent metaparameters. Furthermore, for complex data, large numbers of local models are necessary in order to achieve a competitive approximation. In contrast to LWPR, Gaussian process regression (GPR) need less data-specific tuning while yielding high learning accuracy. However, these advantages result into a tremendously higher computational cost both during the prediction step as well as during learning. As a result, this approach is not used for real-time application to date. In this paper, we combine the basic idea behind both approaches attempting to get as close as possible to the speed of local learning while having a comparable accuracy to Gaussian processes regression. This results in an approach inspired by [1], [2] which uses many local GPs in order to obtain a significant reduction of the computational cost during both prediction and learning step, thus, allowing the application of online learning. The remainder of the paper is organized as follows: first we give a short review of standard GPR. Subsequently, we describe our local Gaussian process models (LGP) approach and discuss how it inherits the advantages of both GRP and LWPR. Finally, our LGP method is evaluated for learning dynamics models of real robots for accurate tracking control. The tracking task is performed in real-time using computed torque control as shown in Figure 1. The learning accuracy and performance of our LGP approach will be compared with the most important standard methods LWPR, standard GPR [8] and ν-support vector regression (ν-SVR) [9], respectively. The evaluations are performed with both physically realistic simulations based on SL [10] as well as two different real robots, i.e., the SARCOS anthropomorphic master arm and BARRETT whole arm manipulator.

II. Regression with standard GPR

Given a set of n training data points $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we intend to discover the latent function $f_i(\mathbf{x}_i)$ which transforms the input vector $\mathbf{x}_i$ into a target value y_i given by $y_i = f_i(\mathbf{x}_i)+\epsilon_i$, where ϵ_i is Gaussian noise with zero mean and variance σ_n^2 [8]. As a result, the observed targets can also be described by $\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I}\right)$, where $\mathbf{K}(\mathbf{X},\mathbf{X})$ denotes the covariance matrix. As covariance function, a Gaussian kernel is frequently taken [8]

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

where σ_s^2 denotes the signal variance and $\mathbf{W}$ the width of the Gaussian kernel. To make a prediction $\bar{f}_*(\mathbf{x}_*)$ for a new input vector $\mathbf{x}_*$, the joint distribution of the observed target values and predicted function value is given by

$$\begin{bmatrix} \mathbf{y} \\ \bar{f}_*(\mathbf{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I} & \mathbf{k}(\mathbf{X},\mathbf{x}_*) \\ \mathbf{k}(\mathbf{x}_*,\mathbf{X}) & k(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}\right). \quad (3)$$

The conditional distribution yields the predicted mean value $\bar{f}_*(\mathbf{x}_*)$ with the corresponding variance $V(\mathbf{x}_*)$ [8]

$$\begin{aligned} \bar{f}_*(\mathbf{x}_*) &= \mathbf{k}_*^T \left(\mathbf{K} + \sigma_n^2\mathbf{I}\right)^{-1} \mathbf{y} = \mathbf{k}_*^T \boldsymbol{\alpha}, \\ V(\mathbf{x}_*) &= k(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}_*^T \left(\mathbf{K} + \sigma_n^2\mathbf{I}\right)^{-1} \mathbf{k}_*, \end{aligned} \quad (4)$$

with $\mathbf{k}_* = \mathbf{k}(\mathbf{X},\mathbf{x}_*)$, $\mathbf{K} = \mathbf{K}(\mathbf{X},\mathbf{X})$ and $\boldsymbol{\alpha}$ denotes the so-called prediction vector. The hyperparameters of a Gaussian process with Gaussian kernel are $\boldsymbol{\theta} = [\sigma_n^2, \sigma_f^2, \mathbf{W}]$ and their optimal value for a particular data set can be derived by maximizing the log marginal likelihood using common optimization procedures, e.g., quasi-Newton methods [8].

III. Local Gaussian Processes Regression

The major drawback of GPR is the expensive computation of the inverse matrix $(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}$ which yields a cost of $\mathcal{O}(n^3)$. Many attempts have been made to reduce this computational obstacle but they mostly follow two strategies: (i) sparse Gaussian processes (SGP), (ii) mixture of experts (ME). In SGP, the whole input space is approximated by a smaller set of so-called inducing inputs [11]–[13]. Here, the difficulty is to choose an appropriate set of inducing inputs which essentially summarize the original input space [8]. In contrast to SGP, ME divide the whole input space in smaller subspaces by a gating network, within which a Gaussian process expert, i.e. Gaussian local model, is trained [2], [14]. The computation cost for matrix inversion is then significantly reduced due to much smaller size of data points within a local model. However, the ME performance depends largely on the number of experts for a particular data set. To avoid this problem, [14] allows the learning process to infer the required number of experts for a given data set by employing a gating network related to Dirichlet process. 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 and d the dimension of input vector.

In the broader sense, the gating network of ME can be considered as a clustering process, where the main intention is to group the whole input data in a meaningful way. Inspired by locally weighted learning [1], [2], we propose a method enabling further reduction of computational cost by using a Gaussian kernel as distance measure for clustering the input data. The algorithm assigns each input point to the corresponding local model, i.e. specific expert, for which an inverse covariance matrix is determined. The mean prediction for a query point is subsequently made by weighted averaging over local mean predictions using local models in the neighborhood. Thus, the algorithm consists out of two stages: (i) localization of data, i.e. allocation of new input points and learning of corresponding local models, (ii) prediction for a query point.

Algorithm 1 Data Allocation and Model Learning

Input: new data point $\{\mathbf{x},\, y\}$.
for $k=1$ **to** all local models **do**
  Compute distance to the k-th local model:
    $w_k = \exp(-0.5(\mathbf{x}-\mathbf{c}_k)^T\mathbf{D}_k(\mathbf{x}-\mathbf{c}_k))$
end for
Take the nearest local model:
  $v = \max(w_k)$
if $v > w_{gen}$ **then**
  Insert $\{\mathbf{x},\, y\}$ to nearest local model:
    $\mathbf{X}_{\text{new}} = [\mathbf{X}, \mathbf{x}]$, $\mathbf{y}_{\text{new}} = [\mathbf{y}, y]$
  Update corresponding center:
    $\mathbf{c}_{new} = \text{mean}(\mathbf{X}_{new})$
  Compute inverse covariance matrix and prediction vector of local model:
    $\mathbf{K}_{\text{new}} = \mathbf{K}(\mathbf{X}_{\text{new}}, \mathbf{X}_{\text{new}})$
    $\boldsymbol{\alpha}_{\text{new}} = (\mathbf{K}_{\text{new}} + \sigma^2\mathbf{I})^{-1}\mathbf{y}_{\text{new}}$
  Update the kernel width $\mathbf{D}_k$, if the number of new input examples reaches a certain quantity:
    Optimize $\mathbf{D}_k$ by maximizing log likelihood using a subsample of all training data.
else
  Create new model:
    $\mathbf{c}_{k+1} = \mathbf{x}$, $\mathbf{X}_{k+1} = [\mathbf{x}]$, $\mathbf{y}_{k+1} = [y]$
  Initialization new inverse covariance matrix and new prediction vector.
end if

A. Localization of Data and Learning of Local Models

Clustering input data is performed by considering a distance measure of the input point $\mathbf{x}$ to the centers of all local models. The distance measure w_k is given by

$$w_k = \exp\left(-\frac{1}{2}\left(\mathbf{x}-\mathbf{c}_k\right)^T\mathbf{D}_k\left(\mathbf{x}-\mathbf{c}_k\right)\right), \tag{5}$$

where $\mathbf{c}_k$ denotes the center of the k-th local model and $\mathbf{D}_k$ a diagonal matrix represented the particular kernel width. It should be noted, that we use the *same* kernel width for computing w_k as well as for training of local GP models as given in Section II. The kernel width $\mathbf{D}_k$ and $\mathbf{W}$, respectively, is obtained by maximizing the log likelihood on a subset of the whole training data points. For doing so, we subsample the training data and, subsequently, perform an optimization procedure. Since $\mathbf{D}_k$ and $\mathbf{W}$ depend on training data which arrives as a stream, the kernel width has to be updated regularly as new data points are inserted to local models. Determining the distance measure w_k, we consider only the joint angles and velocities of the robot, i.e. $\mathbf{x}=[\mathbf{q}, \dot{\mathbf{q}}]^T$. Thus, the localization is performed in a 14-dim space (7 times joint angle and velocity). After determining w_k for all k local models, the input point $\mathbf{x}$ will be assigned to the *nearest* local model, i.e. the local model with the maximal value of distance measure w_k.

During the localization process, a new center $\mathbf{c}_{k+1}$, i.e. a new model, is created, if all distance measures w_k fall below a limit value w_{gen}. The new data point $\mathbf{x}$ is then set as new center $\mathbf{c}_{k+1}$. Thus, the number of local models is allowed to increase as the trajectories become more complex. Otherwise, if a new point is assigned to a particular k-th model, the center $\mathbf{c}_k$ is updated as mean of local data points. With the new assigned input point, the inverse covariance matrix of the corresponding local model can be computed. The localization procedure is summarized in Algorithm 1. Here, learning of new data takes place by adapting the local center, updating the kernel width and computing the inverse covariance matrix of local model.

Considering the algorithm, we mainly have a computational cost of $\mathcal{O}(N^3)$ for inverting the local covariance matrix, where N presents the number of data points in a local model. Furthermore, we can control the complexity by defining a limit number for data points in a local model. Since the number of local data points increases continuously over time, we can keep this limit by deleting an old data point as a new one is included. Insertion and deletion of data points can be decided by evaluating the information gain of the operation. The cost for inverting the local covariance matrix can be further reduced, as we need only to update the inverse matrix once it is computed. The update can be performed using Sherman-Morrison formula which has a complexity of $\mathcal{O}(N^2)$.

Algorithm 2 Weighted Prediction

Input: query data point $\mathbf{x}$, M.
Determine M local models next to $\mathbf{x}$.
for $k=1$ **to** M **do**
  Compute distance to the k-th local model:
    $w_k = \exp(-0.5(\mathbf{x}-\mathbf{c}_k)^T\mathbf{D}_k(\mathbf{x}-\mathbf{c}_k))$
  Compute local mean using the k-th local model:
    $\bar{y}_k = \mathbf{k}_k^T\boldsymbol{\alpha}_k$
end for
Compute weighted prediction using M local models:
  $\hat{y} = \sum_{k=1}^{M} w_k\bar{y}_k / \sum_{k=1}^{M} w_k$.

B. Weighted Prediction for a Query Point

The prediction for a mean value $\hat{y}$ is performed by weighted averaging over M local predictions $\bar{y}_k$. Hence, we have

$$\hat{y} = \frac{\sum_{k=1}^{M} w_k\bar{y}_k}{\sum_{k=1}^{M} w_k}\,. \tag{6}$$

Equation (6) is the solution of the cost function $J = \sum_{k=1}^{M} w_k(\bar{y}_k - \hat{y})^2$ and can also be motivated probabilistically [15]. Thus, our prediction $\hat{y}$ is a weighted least-square solution of M local predictions $\bar{y}_k$ using local models in the neighborhood. Each local prediction $\bar{y}_k$ is determined using Equation (4) and additionally weighted by the distance w_k between the corresponding center $\mathbf{c}_k$ and the query point $\mathbf{x}$. The search for M local models can be quickly done using nearest-neighbour-search algorithm. The prediction procedure is summarized in Algorithm 2.

(a) **SARCOS master arm** (b) **BARRETT WAM**

Fig. 2: Robot arms for generation of real data.

IV. Evaluations

We have evaluated our algorithm using high-dimensional robot data taken from real robots, e.g., the 7 degree-of freedom (DoF) anthropomorphic SARCOS master arm and 7-DoF BARRETT whole arm manipulator shown in Figure 2, as well as a physically realistic SL simulation [10]. First, we compare the learning performance of LGP with the state-of-the-art in nonparametric regression, e.g., LWPR, ν-SVR and standard GPR in the context of approximating robot dynamics. Subsequently, we use the learned dynamics model for a real-time control task. For evaluating ν-SVR and GPR, we have employed the libraries [16] and [17].

A. Dynamics Learning Accuracy Comparison

For the comparison of the accuracy of our method in the setting of learning inverse dynamics, we use three data sets, (i) simulation data as described in [6], (ii) data from the SARCOS master arm [1], (iii) a new data set generated from our BARRETT arm. Given samples $\mathbf{x_i} = [\mathbf{q_i}, \dot{\mathbf{q}}_\mathbf{i}, \ddot{\mathbf{q}}_\mathbf{i}]$ as input and using the corresponding torques $\mathbf{y} = \mathbf{u}$ as targets, we have a proper regression problem. For the considered seven degrees of freedom robot arms, we, thus, have data with 21 inputs (for each joint, we have an angle, a velocity and an acceleration) and 7 targets (a torque for each joint). Training takes place for each DoF separately employing LWPR, ν-SVR, GPR and LGP, respectively.

Figure 3 gives the normalized mean squared error (nMSE) in percent of the evaluation on the test set for each of the three evaluated scenarios, i.e., the simulated arm in (a), the SARCOS arm in (b) and the BARRETT arm in (c). Here, the normalized mean squared error is defined as: nMSE = Mean squared error/Variance of target. During the prediction on the test set, we take the most activated local models, i.e. the ones with the shortest distance to the query point (e.g., $M = 4$). It should be noted 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. If w_{gen} is too small, a lot of local models will be generated with small number of training points. It turns out that these small local models do not perform well in generalization for unknown data. If w_{gen} is large, the local models become also large which increase the computational complexity.

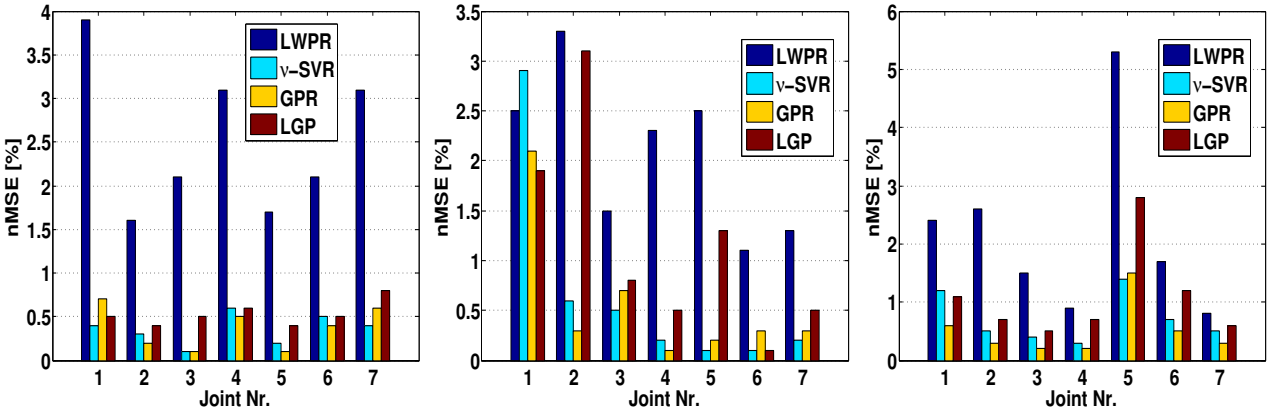
Joint Nr.	nMSE [%]				
	Lin. Regr.	LWPR	ν-SVR	GPR	Local GPR
1	19.2	2.4	1.2	0.6	1.1
2	182.5	2.6	0.5	0.3	0.7
3	19.8	1.5	0.4	0.2	0.5
4	65.6	0.9	0.3	0.2	0.7
5	295.5	5.3	1.4	1.5	2.8
6	27.3	1.7	0.7	0.5	1.2
7	31.9	0.8	0.5	0.3	0.6

TABLE I: **Approximation error** as nMSE (in percent) for each DoF using real SARCOS data. Standard GPR, ν-SVR and LGP show a good learning performance providing higher learning accuracy compared to LWPR. In contrast to nonparametric methods, traditional linear regression using analytical model yields large approximation errors, especially for 2. and 5. DoF. This fact indicates that for those DoF the analytical model fails to explain the data.

Here, the training set is clustered in few local regions (for example, around 50 for the simulated data and 30 local models for the SARCOS data and BARRETT data, respectively) ensuring that each local model has a sufficient amount of data points for high accuracy (in practice, this requirement results in more than 50 data points) while having sufficiently few that the solution remains feasible in real-time (on our current hardware, a Core Duo at 2GHz, that means less than 900 data points). On average, each local model has approximately 300 training examples. This small number of training inputs enables a fast training for each local model, i.e. the matrix inversion and determination of hyperparameters.

Considering the approximation error on the test set shown in Figure 3(a-c), it can be seen that LGP generalizes well even when using only few local models for prediction. In all cases, LGP outpreforms LWPR while being close in learning accuracy to GPR and ν-SVR. The mean-prediction is determined according to Equation (4) where we precomputed 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. The operation of mean-prediction has then the order of $\mathcal{O}(n)$ for standard GPR and ν-SVR, respectively, and $\mathcal{O}(NM)$ for LGP, where n denotes the total number of training points, M number of local models and N number of data points in a local model. In this example, as we take 4 local models for LGP prediction we have to deal with about 1500 data points each time.

The results, as shown in Figure 3 (a-c), indicate that LGP performs better than LWPR in all cases and is able to generalize the learned information well. Table I additionally gives a comparison of nonparametric regression methods employed above with the common linear regression using analytical rigid-body model. The results reported in Table I are computed using SARCOS data. Here, the linear regression yields very large approximation error, especially for the 2. and 5. DoF. Apparently, for these DoF the nonlinearities (e.g., hydraulic cables, complex friction) cannot be approximated well. This example shows the difficulty using the analytical model for control in practice. The imprecise dynamics model will result in poor control performance for real system, e.g., large tracking error.



(a) **Approximation Error using SL data** (b) **Approximation Error using BARRETT data** (c) **Approximation Error using SARCOS data**

Fig. 3: Approximation error as nMSE (in percent) for each DoF. The error is computed after prediction on the test sets with simulated data from SL-model, real robot data from BARRETT and SARCOS master arm, respectively. In all cases, LGP outperforms LWPR in learning accuracy while being competitive to ν-SVR and standard GPR.

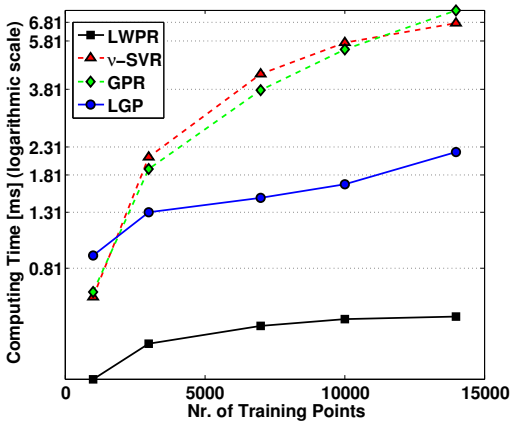


Fig. 4: Average time in millisecond needed for prediction of 1 query point. The computation time is plotted logarithmic in respect of the number of training examples. The time as stated above is the required time for prediction of all 7 DoF. Here, LWPR presents the fastest method due to simple regression models. Compared to global regression methods such as standard GPR and ν-SVR, local GP makes significant improvement in term of computation time.

B. Comparison of Computation Speed for Prediction

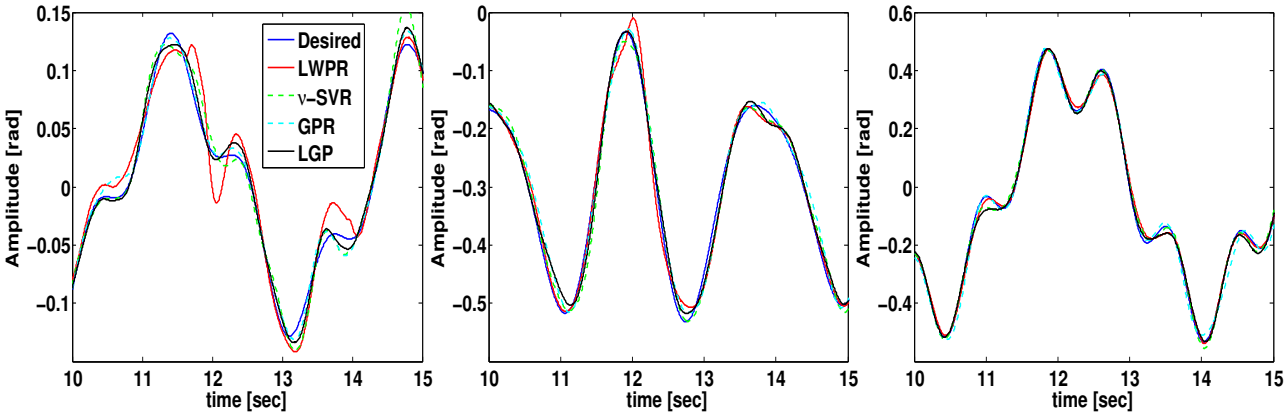
Due to the fact that only a small amount of local models in the vicinity are needed during prediction for LGP, the computation time is reduced significantly compared to GPR and ν-SVR. The comparison of prediction speed is shown in Figure 4. Here, we train LWPR, ν-SVR, GPR and LGP on 5 different data 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.

The results 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 speed. However, as shown in Figure 4, the cost for LGP is significantly lower than the one ν-SVR and GPR and increases at a much lower rate. In practice, we can also curb the computation demands of single models by deleting old data points, if a new ones are assigned to the model. As approach to deleting and inserting data points, we can use the information gain of the corresponding local model as a principled measure. It can be seen from the results that 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 while keeping a good learning performance.

C. Low-Gain Model-based Control

In this section, we use the dynamics models trained in Section IV-A for a computed torque tracking control task in the setting shown in Figure 1. Again, we follow the real-time setup in [6] using the test trajectories described in [6] as desired trajectories. During the control experiment we set the feedback gains, i.e., $\mathbf{K}_p$ and $\mathbf{K}_v$, to very low values to take the aim of compliant control into account. As a result, the dynamics model has a stronger effect on computing the predicted torque and, hence, a better learning performance of each method results in a lower tracking error. The control task is performed in real-time using a physically realistic simulation of the SARCOS arm in SL [10]. Table II shows the tracking error, i.e. $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$, as nMSE for all 7 DoF. The tracking performance over time of joints 1, 2 and 3 are shown in Figure 5 for example, other joints are similar.

As comparison we also compute a simple linear controller with gravity compensation which is a common approach in industrial robot control [3]. The robot system is sampled by 480 Hz. For LWPR, we are able to compute the controller command $\mathbf{u}$ for *every* sample-step, since the torque prediction is very fast, i.e. less than 1 ms for a prediction for all 7 DoF (see Figure 4). As GPR and ν-SVR require much longer for a prediction due to more involved computation, i.e. about 7 ms for a prediction, we can only update the controller command



(a) **Tracking of 1. Joint** (b) **Tracking of 2. Joint** (c) **Tracking of 3. Joint**

Fig. 5: Tracking performance of LWPR, ν-SVR, GPR and LGP for joint 1, 2 and 3 as example. Other joints show similar tracking performance.

Joint Nr.	nMSE [%]				
	g. comp. PD	LWPR	ν-SVR	GPR	Local GPR
1	27.7	2.5	1.8	1.3	0.9
2	15.1	1.1	1.5	1.7	0.8
3	3.3	0.3	0.3	0.4	0.3
4	12.4	4.8	2.9	3.2	2.2
5	3.6	0.4	0.2	0.4	0.2
6	4.9	0.5	0.4	0.6	0.3
7	13.9	1.8	1.3	1.4	0.8

TABLE II: **Tracking error** as nMSE (in percent) for each DoF using test trajectories. The error is computed after a real-time tracking task over 60 sec. The traditional PD controller with gravity compensation is inferior to the model-based method. GPR and ν-SVR provide slightly better results compared to LWPR. LGP gives the best tracking performance in all cases.

for every 4-th sample-step. In contrast, the controller update can be done for every 2-th sample-step in case of LGP, whereas we use 4 local models for prediction each time.

Considering the tracking error reported in Table II, it can be seen that all computed torque control algorithms clearly outperform the state-of-the-art PD controller with gravity compensation. Compared LWPR with global regression methods, i.e. GPR and ν-SVR, the global regression provides only a slightly better result in spite of higher learning accuracy. The reason is that the controller command $\mathbf{u}$ is updated at every sample-step for LWPR instead of every 4-th sample-step for GPR and ν-SVR. Hence, using LWPR the robot can react much faster towards changes in the trajectories. This advantage of fast computing compensates the inferior learning performance of LWPR. As LGP incorporates the strength of both local and global regressions, i.e. a tradeoff of learning accuracy and computation complexity, it provides the best tracking results as shown in Table II.

V. Conclusion and Future Prospects

Using local GP, we combine the fast computation of local regression with more accurate regression methods with less manual tuning. LGP achieves higher learning accuracy compared to locally linear methods such as LWPR while having less computational cost compared to GPR and ν-SVR. The results also show that our approach is promising for an application in model online-learning which is necessary to generalize the dynamics model for all trajectories.

References

[1] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Computation*, 2005.

[2] E. Snelson and Z. Ghahramani, "Local and global sparse gaussian process approximations," *Artificial Intelligence and Statistics*, 2007.

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

[4] J. Nakanishi, J. A. Farrell, and S. Schaal, "Composite adaptive control with locally weighted statistical learning," *Neural Networks*, 2005.

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

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

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

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

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

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

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

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

[13] 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.

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

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

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

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