The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Sparse Online Model Learning for Robot Control with Support Vector Regression

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

Abstract— **The increasing complexity of modern robots makes it prohibitively hard to accurately model such systems as required by many applications. In such cases, machine learning methods offer a promising alternative for approximating such models using measured data. To date, high computational demands have largely restricted machine learning techniques to mostly offline applications. However, making the robots adaptive to changes in the dynamics and to cope with unexplored areas of the state space requires online learning. In this paper, we propose an approximation of the support vector regression (SVR) by sparsification based on the linear independency of training data. As a result, we obtain a method which is applicable in real-time online learning. It exhibits competitive learning accuracy when compared with standard regression techniques, such as ν-SVR, Gaussian process regression (GPR) and locally weighted projection regression (LWPR).**

I. Introduction

In recent years, model learning has become a popular tool in many robotics applications such as model-based control [1], [2], terrain modeling [3], sensor modeling [4] and many other applications. The reason for this rising interest is that with the increasing complexity in modern robots, hand-crafted models of such systems are often not sufficiently accurate. Model learning can be an useful alternative in such cases, as the model can be obtained directly using measured data. Unknown nonlinearites are directly taken in account, while they are neglected either by the standard modeling techniques, or by hand-crafted approximation using measurements. However, excessive computational complexity still hinders a widespread application of the more accurate learning techniques in robotics. Due to high computational demands, models are mostly approximated offline on pre-sampled data to date. Models that are learned offline can only approximate the model correctly in the area of the state space that is covered by the sampled data. In order to cope with unknown state space regions, online model learning is necessary. It also allows the adaption to changes in the robot dynamics, unforeseen load or time-variant torque generation of the actuators. However, real-time online model learning poses three major challenges, i.e., firstly, the learning and prediction process needs to be sufficiently fast. Secondly, the learning system needs to deal with very large amounts of data. And, thirdly, the data arrives as a continuous stream, thus, the model has to be continuously adapted to new training examples over time. Previously, several approaches for real-time model learning for robotics have been introduced, such as locally weighted projection regression (LWPR) [5] or local Gaussian process regression [2]. In these methods, the state space is partitioned in local regions for which local models are approximated and, thus, these methods will not make proper use of the global behavior of the embedded functions. As the proper allocation of relevant areas of the state space is essential, appropriate online clustering becomes a central problem of these approaches. For high dimensional data, partitioning of the state space is well-known to be a complicated problem [2], [5]. To circumvent this online clustering [2], an alternative is to find a sparse representation of the *known* data [6], [7]. By using the sparse set for model learning, the computational complexity can be reduced. However, determining an appropriate sparsification method applicable for a real-time online learning is a major challenge tackled here.

In this paper, we propose an online sparsification method applicable for online model learning with support vector regression (SVR). Our approach is an extension of the work in [8]–[10] and we show that it is appropriate for real-time application. The proposed sparsification is performed using a test of linear independency to select a sparse subset of the training data points, often called *dictionary*. The resulting framework allows us to derive criteria for incremental insertion and deletion of dictionary data points which are the two essential operations in the online learning scenario. As evaluation, the proposed approach is applied for online learning of the inverse dynamics model. The rest of the paper will be organized as follows: firstly, we review the problem of learning inverse dynamics for model-based control, secondly, the SVR method. Subsequently, we present a sparsification approach enabling online model learning with SVR, called SOSVR. The efficiency of the proposed approach is exhibited by a comparison of learning inverse dynamics models with well-established regression methods, i.e., ν-support vector regression (ν-SVR) [11], Gaussian process regression (GPR) [6], locally weighted projection regression (LWPR) [5] and online Gaussian process regression (OGP) [12].

In Section I-A we will review support vector regression, and in Section I-B inverse dynamics model for model-based control.

A. Support Vector Regression

Support vector regression attempts to find a hyperplane fitting the data in a high-dimensional feature space [7]. Thus, the model is given by

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}) + b\,, \tag{1}$$

where $\mathbf{x}$ is an input point projected to the feature space by $\mathbf{\Phi}$, $\mathbf{w}$ and b denote the weights and bias of the hyperplane in

that space, respectively. Given the training data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, $\mathbf{w}$ (and, subsequently, b) can be solved by minimizing the following constraint optimization problem

$$\min_w \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{n} \sum_{i=1}^{n} (\xi_i + \xi_i^*) \,, \tag{2}$$

$$\text{subject to} \quad \begin{aligned} &\left(\mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_i) + b\right) - y_i \leq \epsilon + \xi_i \\ &y_i - \left(\mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_i) + b\right) \leq \epsilon + \xi_i^* \\ &\xi_i^{(*)} \geq 0 \,, \end{aligned}$$

where $\xi_i^{(*)}$ denote slack variables, C a penalty factor for the deviation of data points, ϵ represents the width of a tube around the hyperplane where deviations are tolerated. Equation (2) can be solved using its dual form [7] yielding $\mathbf{w} = \sum_{i=1}^n (\alpha_i^* - \alpha_i) \mathbf{\Phi}(\mathbf{x}_i)$ with the Lagrange multipliers $\alpha_i^{(*)}$. Thus, the solution for Equation (1) is given by

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\alpha_i^* - \alpha_i) \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b \,, \tag{3}$$

where the kernel $\mathrm{k}(\mathbf{x}_i, \mathbf{x})$ represents the inner-product of two feature vectors $\mathbf{\Phi}$. A frequently used kernel is a Gaussian kernel, for example, $\mathrm{k}(\mathbf{x}_i, \mathbf{x}) = \exp(-0.5(\mathbf{x}_i - \mathbf{x})^T \mathbf{W} (\mathbf{x}_i - \mathbf{x}))$, with $\mathbf{W}$ represents the width of the Gaussian kernel. For solving the optimization problem in Equation (2), the parameter ϵ has to be chosen in accordance with the noise level in the data. A straightforward way to adapt ϵ is implemented by the ν-SVR that uses an open parameter ν in order to automatically adjust ϵ [11].

B. Learning Inverse Dynamics for Control

Model-based tracking control [13] has many potential advantages such as compliance, better performance during high speed movements, reduced energy consumption and improved tracking accuracy. The model-based tracking control law determines the joint torques $\mathbf{u}$ that are required to follow a desired trajectory $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$. In computed torque control, the motor command $\mathbf{u}$ consists out of two parts, a feedforward term $\mathbf{u}_{\mathrm{FF}}$ to achieve the movement and a feedback term $\mathbf{u}_{\mathrm{FB}}$ to ensure stability of the tracking. 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 $\mathbf{e}$ denotes the tracking error with position gain $\mathbf{K}_p$ and velocity gain $\mathbf{K}_v$. The feedforward term $\mathbf{u}_{\mathrm{FF}}$ is determined using an inverse dynamics model and, traditionally, the analytical rigid-body model is employed [13]. If a sufficiently precise inverse dynamics model can be approximated, the resulting control law $\mathbf{u} = \mathbf{u}_{\mathrm{FF}}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d) + \mathbf{u}_{\mathrm{FB}}$ will drive the robot along the desired trajectory accurately.

One possibility to obtain an accurate inverse dynamics model is to learn it directly from measured data. The resulting problem is a regression problem which can be solved by learning the inverse dynamics $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}_{\mathrm{FF}}$ on sampled data [1], [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 inverse dynamics will include all nonlinearities of the system.

II. Real-time Online Support Vector Regression

Classical SVR does not support incremental updates of the model that are necessary for online learning as the data arrives sequentially over time. The continuously increasing online data set and the real-time performance requirement prohibit the straightforward application of standard SVR in the online mode. There have been many attempts to develop an online algorithm for SVR, for an overview see [7]. However, to our best knowledge only few of them are potentially applicable in a real-time learning system (e.g., for online learning with model updates at 50 Hz or faster). A promising approach is given in [8]–[10], where [10] provides a sequential SVR training procedure and [8] applies a linear independence test for the sparsification of SVR. However, the approach given in [8] is not yet suitable for application in fast real-time learning as the sparse data set is not bounded and, thus, can become arbitrarily large during online learning. In this paper, we extend the idea of sparsification using linear independence tests [8] and develop a framework for incremental insertion & deletion of data points which can deal online with constantly arriving new data points.

A. Test of Linear Independency and Training of SVR

At any point in time, our algorithm maintains a dictionary $\{\mathbf{x}_i, y_i\}_{i=1}^m$. To test whether a new point $\{\mathbf{x}_{m+1}, y_{m+1}\}$ should be inserted into the dictionary, we firstly need to ensure that it can not be approximated in the feature space using the current data set. This test can be performed using [7], [8]

$$\delta = \left\| \sum_{i=1}^{m} a_i \mathbf{\Phi}(\mathbf{x}_i) - \mathbf{\Phi}(\mathbf{x}_{m+1}) \right\|^2 , \tag{4}$$

where the a_i denote the coefficients of the linear dependency. The coefficients a_i can be determined by minimizing δ, which yields the optimal coefficient vector $\boldsymbol{a}_m = \boldsymbol{K}_m^{-1} \boldsymbol{k}_m$. Here, $\boldsymbol{K}_m = \mathbf{k}(\mathbf{X}_m, \mathbf{X}_m)$ is the kernel matrix evaluated for the dictionary data set $\mathbf{X}_m$ and $\boldsymbol{k}_m = \mathbf{k}(\mathbf{X}_m, \mathbf{x}_{m+1})$ is the kernel vector. After substituting the optimal value $\boldsymbol{a}_m$ into Equation (4), δ becomes

$$\delta = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1}) - \boldsymbol{k}_m^T \boldsymbol{a}_m \,. \tag{5}$$

The value δ is an independency measure indicating how well a new data point $\mathbf{x}_{m+1}$ can be approximated by a given data set in the feature space. Thus, the larger the value δ is, the more *independent* is $\mathbf{x}_{m+1}$ from the dictionary set $\mathbf{X}_m$, and the more *informative* is $\mathbf{x}_{m+1}$ for the learning procedure. Using δ we can decide whether to insert new data points into the dictionary. However, it is necessary to set a budget for the dictionary in order to cope with the computational complexity. Thus, we have to delete dictionary points, if the given limit is reached. The procedure for inserting and deleting of dictionary points will be discussed in detail in Sections II-B and II-C. Due to the lack of space, we will focus on the main results. After updating the dictionary, the support vector machine can be sequentially trained on the new dictionary [10], i.e., by re-estimating the Lagrange multipliers for the prediction step. The training and update procedure are summarized in Algorithm 1.

Algorithm 1: Independency Test with Online Update of Dictionary and Sequential Training of SVR.

Input: new point $\{\mathbf{x}_{m+1}, y_{m+1}\}$, η, MaxNr.
Compute
$\boldsymbol{a}_m = \boldsymbol{K}_m^{-1}\boldsymbol{k}_m$
$\boldsymbol{k}_m = \mathrm{k}(\mathbf{X}_m, \mathbf{x}_{m+1})$
Compute δ
$\delta = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1}) - \boldsymbol{k}_m^T \boldsymbol{a}_m$
if $\delta > \eta$ **then**
  if number of dictionary points $<$ MaxNr **then**
    Update dictionary according to Algorithm 2
    by inserting $\{\mathbf{x}_{m+1}, y_{m+1}\}$.
  else
    Update dictionary according to Algorithm 3
    by inserting $\{\mathbf{x}_{m+1}, y_{m+1}\}$
    and deleting an old dictionary point.
  end if
  Sequential Training of SVR
  using the new dictionary $\{\mathbf{x}_i, y_i\}_{i=1}^{m+1}$.
end if
Compute prediction for a query point $\mathbf{x}$
$f(\mathbf{x}) = (\boldsymbol{\alpha}_i^* - \boldsymbol{\alpha}_i)\, \mathrm{k}(\mathbf{X}_{m+1}, \mathbf{x}) + b$

B. Insert a New Point to the Dictionary

The independency measure δ can be used as a criterium for the selection of new data points by defining a threshold value η. If $\delta > \eta$, the new point will be included to the dictionary, otherwise not. The larger η is chosen, the smaller is the potential number of dictionary points. However, a small dictionary size may lead to a bad generalization ability. On the other hand, large dictionary will be prohibitive expensive in terms of computational complexity. To cope with this problem, we set η sufficiently small and define an upper bound on the dictionary size according to the computational power of the system. If the bound is reached, we delete old dictionary points depending on their independency from the remainder of the dictionary. For doing so, we have to constantly update the independency variable δ^k for every dictionary point k after inserting a new data point, as changing the dictionary implies a change for each δ^k.

The independency measure δ, as given in Equation (5), can be incrementally updated for *each* old dictionary point after inserting a new point $\mathbf{x}_{m+1}$. This update for an existing dictionary point $\mathbf{x}_k$ is done by adjusting the corresponding coefficient vector $\boldsymbol{a}_m^k$. However, updating $\boldsymbol{a}_m^k$ implies an update of $(\boldsymbol{K}_m^k)^{-1}$ and $\boldsymbol{k}_m^k$. Here, inserting a new point will extend $\boldsymbol{K}_m^k$ by a row/column and $\boldsymbol{k}_m^k$ by a value, respectively, such that

$$\boldsymbol{K}_{m+1}^k = \begin{bmatrix} \boldsymbol{K}_m^k & \boldsymbol{k}^k \\ \boldsymbol{k}^{kT} & k_{mm} \end{bmatrix}, \quad \boldsymbol{k}_{m+1}^k = \begin{bmatrix} \boldsymbol{k}_m^k \\ k_{km} \end{bmatrix}, \tag{6}$$

where $\boldsymbol{k}^k = \mathbf{k}(\mathbf{X}_m^k, \mathbf{x}_{m+1})$, $k_{mm} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1})$ and $k_{km} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_k)$. The incremental update of the inverse matrix $(\boldsymbol{K}_{m+1}^k)^{-1}$ can be given as

Algorithm 2: Update of dictionary by insertion of a new data point.

Input: new dictionary point $\{\mathbf{x}_{m+1}, y_{m+1}\}$.
Update dictionary $\{\mathbf{x}_i, y_i\}_{i=1}^{m+1}$
for $k = 1$ **to** number of dictionary points **do**
  Get $\mathbf{x}_k$ and $\mathbf{X}_m^k$
  Compute
  $\boldsymbol{k}^k = \mathbf{k}(\mathbf{X}_m^k, \mathbf{x}_{m+1})$
  $k_{mm} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1})$
  $k_{km} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_k)$
  Compute
  $\boldsymbol{\alpha}_k = \left(\boldsymbol{K}_m^k\right)^{-1} \boldsymbol{k}^k$
  $\gamma_k = k_{mm} - \boldsymbol{k}^{kT} \boldsymbol{\alpha}_k$
  Update δ^k as given in Equation (8).
  Update $(\boldsymbol{K}_{m+1}^k)^{-1}$ as given in Equation (7).
end for

$$\left(\boldsymbol{K}_{m+1}^k\right)^{-1} = \frac{1}{\gamma_k} \begin{bmatrix} \gamma_k (\boldsymbol{K}_m^k)^{-1} + \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^T & -\boldsymbol{\alpha}_k \\ -\boldsymbol{\alpha}_k^T & 1 \end{bmatrix}. \tag{7}$$

This derivation leads to the update rule for the linear independency value δ^k for the k-th dictionary point

$$\begin{aligned} \delta^k &= \mathrm{k}(\mathbf{x}_k, \mathbf{x}_k) - \boldsymbol{k}_{m+1}^{kT} \boldsymbol{a}_{m+1}^k, \\ \boldsymbol{a}_{m+1}^k &= \frac{1}{\gamma_k} \begin{bmatrix} \gamma_k \boldsymbol{a}_m^k + \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^T \boldsymbol{k}_m^k - k_{km} \boldsymbol{\alpha}_k \\ -\boldsymbol{\alpha}_k^T \boldsymbol{k}_m^k + k_{km} \end{bmatrix}. \end{aligned} \tag{8}$$

The variables γ_k and $\boldsymbol{\alpha}_k$ are computed as

$$\begin{aligned} \boldsymbol{\alpha}_k &= (\boldsymbol{K}_m^k)^{-1} \boldsymbol{k}^k, \\ \gamma_k &= k_{mm} - \boldsymbol{k}^{kT} \boldsymbol{\alpha}_k. \end{aligned}$$

The procedure for insertion of new data points with δ-update is summarized in Algorithm 2.

C. Insert a New Point and Delete a Dictionary Point

As the data arrives continuously in an online setting, it is necessary to limit the number of dictionary points so that the computational power of the system is not exceeded. Thus, it is essential to delete old points if the limit is reached after inserting new data points. The deletion can be performed straightforwardly by deleting a dictionary point with a *minimal* value of δ. The idea is to delete points which are most *dependent* on other dictionary points, i.e., the corresponding δ value is minimal. Insertion and additional deletion of dictionary points also change the independency values of other dictionary points which have to be updated.

Insertion of a new point with an additional deletion of the j-th dictionary point imply a manipulation of the j-th row/column of $\boldsymbol{K}_m^k$ and the j-th value of $\boldsymbol{k}_m^k$, i.e., by

$$\boldsymbol{K}_{m+1}^k = \begin{bmatrix} \boldsymbol{K}_{j-m}^k & \boldsymbol{k}_j^{kT} & \boldsymbol{K}_{j+m}^k \\ \boldsymbol{k}_j^k & k_{mm} & \boldsymbol{k}_{j+m}^k \\ \boldsymbol{K}_{j+m}^k & \boldsymbol{k}_{j+m}^{kT} & \boldsymbol{K}_{j-m}^k \end{bmatrix}, \boldsymbol{k}_{m+1}^k = \begin{bmatrix} \boldsymbol{k}_{j-m}^k \\ k_{km} \\ \boldsymbol{k}_{j+m}^k \end{bmatrix}. \tag{9}$$

The values $\boldsymbol{k}^k$, k_{mm} and k_{km} are determined as shown in Section II-B. The incremental update of the independency

Algorithm 3: Update of dictionary by insertion of a new data point and additional deletion of an old data point.

Input: new dictionary point $\{\mathbf{x}_{m+1}, y_{m+1}\}$.
Find dictionary point with minimal δ
$j = \min_i \delta$
Update dictionary by overwriting point j:
$\{\mathbf{x}_j, y_j\} = \{\mathbf{x}_{m+1}, y_{m+1}\}$
for $k=1$ **to** number of dictionary points **do**
Get $\mathbf{x}_k$ and $\mathbf{X}_m^k$
Compute
$\boldsymbol{k}^k = \mathrm{k}(\mathbf{X}_m^k, \mathbf{x}_{m+1})$
$k_{mm} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1})$
$k_{km} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_k)$
Compute
$\boldsymbol{k}_{m+1}^k = [\ \boldsymbol{k}_{j-m}^k \quad k_{km} \quad \boldsymbol{k}_{j+m}^k\ ]^T$
$\boldsymbol{d} = \boldsymbol{k}^k - row_j[\boldsymbol{K}_m^k]^T$
Update δ^k as given in Equation (10).
Update $(\boldsymbol{K}_{m+1}^k)^{-1} = \boldsymbol{A}$.
end for

measure δ^k for every k-th dictionary point can directly be performed by applying the matrix inversion. Hence,

$$\begin{aligned} \delta^k &= \mathrm{k}(\mathbf{x}_k, \mathbf{x}_k) - \boldsymbol{k}_{m+1}^{kT} \boldsymbol{a}_{m+1}^k \,, \\ \boldsymbol{a}_{m+1}^k &= \boldsymbol{A}\, \boldsymbol{k}_{m+1}^k \,, \end{aligned} \tag{10}$$

where $\boldsymbol{A}$ is computed as

$$\boldsymbol{A} = \boldsymbol{A}^* - \frac{\mathrm{row}_j[\boldsymbol{A}^*]^T \boldsymbol{d}^T \boldsymbol{A}^*}{1 + \boldsymbol{d}^T \mathrm{row}_j[\boldsymbol{A}^*]^T} \,,$$

$$\boldsymbol{A}^* = (\boldsymbol{K}_m^k)^{-1} - \frac{(\boldsymbol{K}_m^k)^{-1} \boldsymbol{d}\, \mathrm{row}_j[(\boldsymbol{K}_m^k)^{-1}]}{1 + \boldsymbol{d}^T \mathrm{row}_j[(\boldsymbol{K}_m^k)^{-1}]^T} \,.$$

Here, $\boldsymbol{d}$ is determined by $\boldsymbol{d} = \boldsymbol{k}^k - \mathrm{row}_j[\boldsymbol{K}_m^k]^T$ and row_j denotes the j-th row of a given matrix. The update of $(\boldsymbol{K}_{m+1}^k)^{-1}$ can be given as $(\boldsymbol{K}_{m+1}^k)^{-1} = \boldsymbol{A}$. The complete procedure is summarized in Algorithm 3.

D. Relation to Previous Work

Our algorithm was inspired by the ideas proposed in [8]. However, there are several important differences which need to be discussed here. In [8], they introduce the so called 'reduced' variables for the training of the SVR. Here, *all* data points which do not belong to the dictionary are represented as a linear combination of the dictionary points. The support vector prediction is then computed as $f(\mathbf{x}) = (\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T \boldsymbol{\Lambda} \mathbf{k}(\mathbf{X}, \mathbf{x}) + b$, where $\boldsymbol{\Lambda}$ denotes a $n \times m$ matrix storing n training points and m dictionary points. It turns out that an update of $\boldsymbol{\Lambda}$ is necessary, if dictionary points have to be deleted. Since the number of data points n increases continuously online, this approach is not appropriate for online application, as $\boldsymbol{\Lambda}$ is growing without any bound.

In order to avoid the problems faced by [8], we train the SVR *directly* from the dictionary points instead of using reduced variables. This approach allows us to formulate an efficient insertion and deletion procedure for the dictionary, making SVR applicable for fast online learning. However, our method requires a sufficiently large dictionary for the training of the SVR. Since the dictionary is permanently updated for every data point, i.e., by inserting and deleting dictionary points, the algorithm also allows fast model adaptation for unknown data. Considering the computational cost we need $\mathcal{O}(m)$ for prediction, where m is the number of dictionary points. For training, we require $\mathcal{O}(m^3)$, i.e., m rank-one updates which have a complexity of $\mathcal{O}(m^2)$.

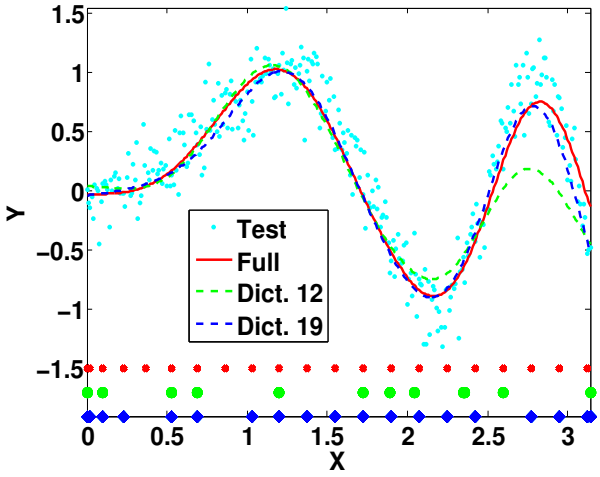
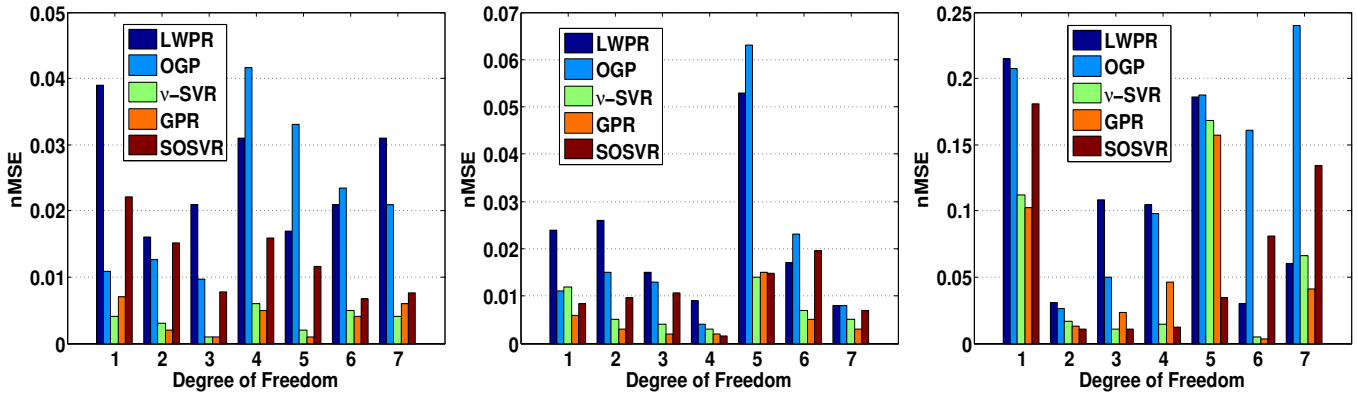


Fig. 1: Prediction performance using the toy data set. The prediction results for the toy example using the full dictionary (red thick line), dictionary with a budget of 12 data points (green dashed line) and dictionary with a budget of 19 data points (blue dashed line). The dots show the positions of the corresponding dictionary points in the input space. Using a fix budget, the most relevant regions in the input space are covered with dictionary points.

E. Toy Example

In this section, we will show the learning performance of the algorithm using a toy data set, in order to highlight the effects of different dictionary size on the generalization ability. As a toy example, we generate a noisy data set where the relation between the target $\mathbf{y}$ and the input $\mathbf{x}$ is given by $y_i = \sin(x_i^2) + \varepsilon_i$. The data consists out of 315 points, ε_i is white Gaussian noise with standard deviation 0.2 and x_i ranges from 0 to π.

The prediction results are shown in Figure 1, where $\eta = 0.001$ and a Gaussian kernel with width 0.4 is being used. Figure 1 shows the performance using the full dictionary (red line), i.e., *no* dictionary points are deleted. With the given threshold η, the algorithm finds 23 dictionary points out of 315 training examples, equally covered the input space $\mathbf{x}$. Since η directly corresponds to a distance in input space, dictionary points are sequentially included, if this distance between sample points in the input space is exceeded. In practice, the number of dictionary points is not known beforehand for a given η. The dictionary size can be arbitrarily large during online learning, if unknown parts of the state space have to be discovered, and, thus, has to be limited. The results of prediction performed using a dictionary with a fix budget are also shown in Figure 1. Through the incremental deletion and insertion process, the



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

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

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

Fig. 2: 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, SOSVR is competitive to standard global batch regression methods such as ν-SVR and GPR, state-of-the-art local learning LWPR and sparse Gaussian process approximation OGP. Note that in **(c)** the small variances of the output targets in the Barrett data result in a larger nMSE when compared to SARCOS data in **(b)**.

Fig. 3: Barrett Whole-Arm-Manipulator (WAM)

dictionary points are mostly concentrated on the state space regions which are 'difficult' to learn, e.g., $1.5 < \mathbf{x} < 2.7$. As shown in the figure, the dictionary size represents a trade-off between learning accuracy and computational complexity.

III. Evaluations

In this section, we evaluate our sparse online SVR (SOSVR) in several different experimental settings. Firstly, we will evaluate our algorithm in the context of learning inverse dynamics. The learning accuracy of SOSVR will be compared with other standard regression methods, i.e., LWPR, GPR, ν-SVR and the online method OGP. For the evaluation in inverse dynamics learning, we employ 3 data sets as described in [1], [2]. These data sets include synthetic data as well as real robot data generated from the 7 degrees of freedom (DoF) anthropomorphic SARCOS arm and the 7-DoF Barrett WAM, shown in Figure 3. Finally, SOSVR is applied for online learning of an inverse dynamics model for robot computed torque control, following the setting in [1]. For the robot control task, we use a model of our 7-DoF Barrett WAM simulated with the real-time simulation package SL [15].

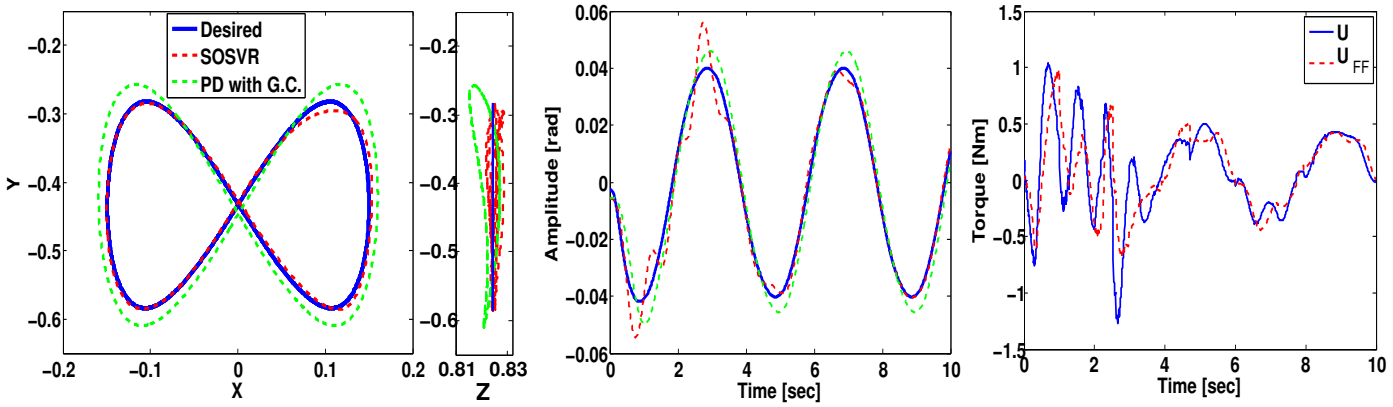
A. Comparison in Learning Inverse Dynamics

For comparing the learning accuracy in the setting of learning inverse dynamics, we use three data sets: (i) SL simulation data for the SARCOS arm model with 14094 training points and 5560 test points [1], (ii) data from the SARCOS master arm with 13622 training points and 5500 test points [5] as well as (iii) a data set generated from our Barrett arm containing 13572 training points and 5000 test points [2]. 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 regression problem. The considered 7 DoF robot arms result in 21 input dimensions (i.e., for each joint, we have an angle, a velocity and an acceleration) and 7 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 SOSVR, respectively.

Figure 2 shows the prediction error on the test sets evaluated using the normalized mean square error (nMSE) defined as the fraction of mean squared error and the variance of target. For all three data sets the dictionary size is limited to 3000. Figure 2 shows the results of the comparison. It can be seen that SOSVR is competitive in learning accuracy.

B. Model Online Learning in Computed Torque Control

In this section, we apply SOSVR for the online learning of the inverse dynamics model of our Barrett WAM for torque prediction as described in [2]. For doing so, the trajectory $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ and the corresponding joint torques $\mathbf{u}$ are sampled online. The inverse dynamics model is learned during the tracking task using the trajectory as input and the joint torque as target, starting with an empty dictionary. Here, the dictionary size is set to be 100, $\eta = 0.0001$ and a Gaussian kernel is used. For the experiment, the desired joint space trajectory is generated such that the robot's end-effector follows a 8-figure in the task space, as shown in



(a) **Tracking Error in Task-space. Left: X-Y view; Right: Z-Y view**

(b) **Tracking Error in Joint-space (3. DoF) during the first 10 sec**

(c) **Feedforward Torque (u_{FF}) and Joint Torque (u) during Learning**

Fig. 4: **(a)** Tracking performance in task space after 20 sec. Computed torque control with model online learning achieves almost perfect tracking outperforming common PD-controller with gravity compensation. **(b)** Tracking performance in joint space for the 3. DoF during the first 10 sec (other DoFs are similar). The result shows that the model converges already after 9 sec with online learning. **(c)** Predicted feedforward torque and joint torque of the 3. DoF during the first 10 sec. The predicted torque $\mathbf{u}_{\mathrm{FF}}$ converges to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning – as a result, the controls need to rely significantly less on feedback.

Figures 4 (a). The robot is controlled in joint space and, thus, large joint space errors indicate large task space errors. The tracking control task is performed in the SL simulation environment [15].

The results of the tracking performance are shown in Figure 4. Here, we compare the computed torque controller using model online learning with a common PD-controller with gravity compensation. The tracking control task is simulated for 20 sec. During this time, the dictionary is first filled up and, subsequently, updated about 900 times. Figure 4 (a) reports the tracking performance in task space after 20 sec. Figure 4 (b) and 4 (c) show exemplarily the tracking performance in joint space and the corresponding joint torque for the 3. DoF during the first 10 sec. As shown by the results, the tracking performance is improved significantly after 20 sec online learning while the model learning process converges already after 9 sec. As seen in Figure 4 (c), the prediction for the feedforward torque during online learning consistently converges to the joint torque $\mathbf{u}$ – as a result, the controls need to rely significantly less on feedback .

IV. Conclusion and Future Prospects

In this paper, we propose a sequential sparsification method enabling a potential application in real-time online model learning with SVR. Our approach provides a framework for efficient insertion and deletion of dictionary points taking in account the required fast computation during model online learning. SOSVR is shown to be competitive in comparison with other state-of-the-art nonparametric regression methods while being sufficiently efficient for an online application. The implementation of this approach on a real robot system is in progress. Our future research will be focused on further examinations such as the impact of different kernel types on the prediction and learning performances.

References

[1] 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.

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

[3] T. Lang, C. Plagemann, and W. Burgard, "Adaptive non-stationary kernel regression for terrain modeling," *Robotics: Science and Systems (RSS)*, 2007.

[4] C. Plagemann, K. Kersting, P. Pfaff, and W. Burgard, "Heteroscedastic gaussian process regression for modeling range sensors in mobile robotics," *Snowbird learning workshop*, 2007.

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

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

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

[8] Y. Engel, S. Mannor, and R. Meir, "Sparse online greedy support vector regression," *European Conference on Machine Learning*, 2002.

[9] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola, "Input space versus feature space in kernel-based methods," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[10] S. Vijayakumar and S. Wu, "Sequential support vector classifiers and regression," *International Conference on Soft Computing*, 1999.

[11] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett, "New support vector algorithms," *Neural Computation*, 2000.

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

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

[14] E. Burdet and A. Codourey, "Evaluation of parametric and nonparametric nonlinear adaptive controllers," *Robotica*, vol. 16, no. 1, pp. 59–73, 1998.

[15] 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