Incremental Online Sparsification for Model Learning in Real-time Robot Control

Duy Nguyen-Tuong and Jan Peters

Max Planck Institute for Biological Cybernetics, Spemannstraße 38, 72076 Tübingen

Abstract

For many applications such as compliant, accurate robot tracking control, dynamics models learned from data can help to achieve both compliant control performance as well as high tracking quality. Online learning of these dynamics models allows the robot controller to adapt itself to changes in the dynamics (e.g., due to time-variant nonlinearities or unforeseen loads). However, online learning in real-time applications – as required in control – cannot be realized by straightforward usage of off-the-shelf machine learning methods such as Gaussian process regression or support vector regression. In this paper, we propose a framework for online, incremental sparsification with a fixed budget designed for fast real-time model learning. The proposed approach employs a sparsification method based on an independence measure. In combination with an incremental learning approach such as incremental Gaussian process regression, we obtain a model approximation method which is applicable in real-time online learning. It exhibits competitive learning accuracy when compared with standard regression techniques. Implementation on a real Barrett WAM robot demonstrates the applicability of the approach in real-time online model learning for real world systems.

Keywords: Sparse Data, Machine Learning, Real-time Online Model Learning, Inverse Dynamics, Robot Control

1. Introduction

In recent years, model learning has become an important tool in a variety of robotics applications such as terrain modeling [5], sensor evaluation [11], model-based control [8, 13] and many others. The reason for this rising interest is that accurate analytical models are often hard to obtain due to the increasing complexity of modern robot systems. Model learning can be a useful alternative as the model is estimated directly from measured data. Unknown nonlinearites are directly taken in account, while they are neglected either by the standard physics-based modeling techniques or approximated by hand-crafted models. Nevertheless, the excessive computational complexity of the more advanced regression techniques still hinders their widespread application in robotics. Models that have been learned offline can only approximate the model correctly in the area of the state space that is covered by the sampled data, and often do not generalize beyond that region. Thus, in order to cope with unknown state space regions online model learning is essential. Furthermore, it also allows the adaptation of the model to changes in the robot dynamics, for example, due to unforeseen loads or time-variant nonlinearities such as backlash, complex friction and stiction.

However, real-time online model learning poses three major challenges: first, the learning and prediction processes need to be sufficiently fast; second, the learning system needs to deal with large amounts of data; third, the data arrives as a continuous stream and, thus, the model has to be continuously adapted to new training exam-

ples. A few approaches for real-time model learning for robotics have been introduced in the machine learning literature, such as locally weighted projection regression [18] or local Gaussian process regression [10]. In these methods, the state space is partitioned in local regions for which local models are approximated and, thus, these methods will not make 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 for these approaches. For high dimensional data, partitioning of the state space is well-known to be a difficult issue [18, 10]. To circumvent this online clustering, an alternative is to find a sparse representation of the *known* data [12, 15, 6]. For robot applications, however, it requires finding an incremental sparsification method applicable in real-time online learning – a major challenge tackled in this paper.

Inspired by the work in [15, 3], we propose a method for incremental, online sparsification which can be integrated into several existing online regression methods, making them applicable for model learning in real-time. The suggested sparsification is performed using a test of linear independence to select a sparse subset of the training data points, often called the *dictionary*. The resulting framework allows us to derive criteria for incremental insertion and deletion of dictionary data points, which are two essential operations in such an online learning scenario. For evaluation, we combine our sparsification framework with an incremental approach for Gaussian process regression (GPR) as described in [10]. The resulting algorithm is applied in online learning of the inverse dynamics model for robot control [17, 9].

The rest of the paper will be organized as follows: first, we present our sparsification approach which enables real-time online model learning. In Section 3, the efficiency of the proposed approach in combination with an incremental GPR update is demonstrated by an offline comparison of learning inverse dynamics models with well-established regression methods, i.e., ν-support vector regression [16], standard Gaussian process regression [12], locally weighted projection regression [18] and local Gaussian process regression [10]. Finally, the capability of incremental GPR using online sparsification for real-time model learning will be illustrated by online approximation of inverse dynamics models for real-time tracking control on a Barrett WAM. A conclusion will be given in Section 4.

2. Incremental Sparsification for Real-time Online Model Learning

In this section, we introduce a sparsification method which – in combination with an incremental kernel regression – enables fast, real-time model learning. The proposed sparsification approach is formulated within the framework of kernel methods. Therefore, we first present the basic intuition behind the kernel methods and motivate the need of online sparsification. Subsequently, we describe the proposed sparsification method in details.

2.1. Model Learning with Kernel Methods

By learning a model, we want to approximate a mapping from the input set $\mathbf{X}$ to the target set $\mathbf{Y}$. Given n training data points $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we intend to discover the latent function $f(\mathbf{x}_i)$ which transforms the input vector $\mathbf{x}_i$ into a target value y_i given by the model $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i represents a noise term. In general, it is assumed that $f(\mathbf{x})$ can be parametrized as $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}$, where $\boldsymbol{\phi}$ is a feature vector mapping the input $\mathbf{x}$ into some high dimensional space and $\mathbf{w}$ is the corresponding weight vector [15, 12]. The weight $\mathbf{w}$ can be represented as a linear combination of the input vectors in the feature space, i.e., $\mathbf{w} = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$ with α_i denoting the linear coefficients. Using these results, the prediction $\hat{y}$ of a query point $\mathbf{x}$ can be given as

$$\begin{aligned} \hat{y} = \hat{f}(\mathbf{x}) &= \textstyle\sum_{i=1}^n \alpha_i \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle \,, \\ &= \textstyle\sum_{i=1}^n \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}) \,. \end{aligned} \tag{1}$$

As indicated by Equation (1), the inner product of features vectors $\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle$ can be represented as a kernel value $\mathrm{k}(\mathbf{x}_i, \mathbf{x})$ [15]. Thus, instead of finding a feature vector, only appropriate kernels

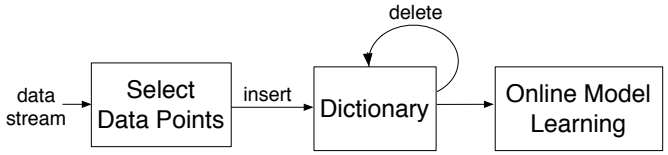


Figure 1: Sparsification for online model learning.

need to be determined. An often used kernel is, for example, the Gaussian kernel

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

where $\mathbf{W}$ denotes the kernel widths [15, 12]. For employing kernel methods in model learning, however, one needs to estimate the linear coefficients α_i using training examples. State-of-the-art kernel methods such as kernel ridge regression, support vector regression (SVR) or Gaussian process regression (GPR), differ in the approaches for estimating α_i [15, 12, 4]. While support vector regression estimates the linear coefficients by optimization using training data [15], kernel ridge regression and Gaussian process regression basically solve the problem by matrix inversion [4, 12] (see the Appendix for a short review of GPR). The complexity of model learning with kernel methods, i.e., the estimation of α_i, depends largely on the number of training examples. In GPR, for example, the computational complexity is $\mathcal{O}(n^3)$, if the model is obtained in batch learning.

However, online model learning requires incremental updates, e.g., incremental estimation of α_i, as the data arrives sequentially. There have been many attempts to develop incremental, online algorithms for kernel methods, such as incremental SVM [2], sequential SVR [19], recursive kernel learning with NARX form [6] or the kernel recursive least-squares algorithm [3], for an overview see [15]. However, most incremental kernel methods do not scale to online learning in real-time, e.g., for online learning with model updates at 50 Hz or faster. The main reason is that they are neither sparse [2, 19], as they use the complete data set for model training, nor do they restrict the size of the sparse set [3]. To overcome these shortcomings, we propose the setup illustrated in Figure 1.

To ensure real-time constraints, we train the model using a dictionary with a fixed budget. The budget of the dictionary, i.e., the sparse set, needs to be determined from the intended learning speed and available computational power. To efficiently make use of the stream of continuously arriving data, we select only the most informative data points for the dictionary. If the budget limit is reached, dictionary points will need to be deleted. Finally, for the model training using dictionary data, most incremental kernel regression methods can be employed, e.g., incremental GPR as described in [10], sequential SVR [19] or incremental SVM [2].

Inspired by the work in [3, 14], we use a linear independence measure to select the most informative points given the current dictionary. Based on this measure, we derive criteria to remove data points from the dictionary, if a given limit is reached. The following sections describe the proposed approach in detail.

2.2. Sparsification using Linear Independence Test

The main idea in our sparsification approach is that we intend to cover the relevant state space at the best, given a limited number of dictionary points. At any point in time, our algorithm maintains a dictionary $\mathcal{D} = \{\boldsymbol{d}_i\}_{i=1}^m$ where m denotes the current number of dictionary points $\boldsymbol{d}_i$. The choice of the dictionary element $\boldsymbol{d}_i$ might be crucial for particular application and will be discussed in Section 2.5. To test whether a new point $\boldsymbol{d}_{m+1}$ should be inserted into the dictionary, we need to ensure that it can not be approximated in the feature space spanned by the current dictionary set. This test can be performed using a measure δ defined as

$$\delta = \left\| \sum_{i=1}^{m} \mathrm{a}_i \boldsymbol{\phi}(\boldsymbol{d}_i) - \boldsymbol{\phi}(\boldsymbol{d}_{m+1}) \right\|^2 , \quad (3)$$

(see, e.g., [14, 15] for background information), where a_i denote the coefficients of linear dependence. Equation (3) can be understood as a distance of the new point $\boldsymbol{d}_{m+1}$ to the linear plane spanned by the dictionary set $\mathcal{D}$ in the feature space as illustrated in Figure 2. Thus, the value

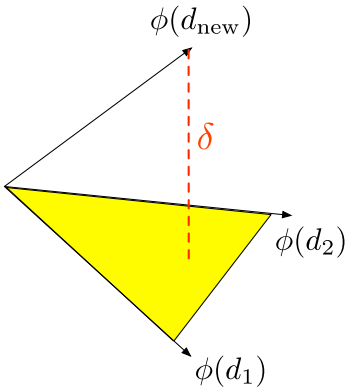


Figure 2: Geometrical interpretation of the independence measure δ. Here, the dictionary consists of two data points $\{\boldsymbol{d}_1, \boldsymbol{d}_2\}$ which span a linear plane in the feature space. The independence measure δ for a new data point $\boldsymbol{d}_{new}$ can be interpreted as the distance to this plane.

δ can be considered as an independence measure indicating how well a new data point $\boldsymbol{d}_{m+1}$ can be approximated in the feature space of a given data set. Thus, the larger the value of δ is, the more *independent* is $\boldsymbol{d}_{m+1}$ from the dictionary set $\mathcal{D}$, and the more *informative* is $\boldsymbol{d}_{m+1}$ for the learning procedure.

The coefficients a_i from Equation (3) can be determined by minimizing δ. Formulated in matrix form, the minimization of δ can be given as

$$\boldsymbol{a} = \min_{\mathbf{a}} \left[\mathbf{a}^T \boldsymbol{K} \mathbf{a} - 2\mathbf{a}^T \boldsymbol{k} + k\right] , \qquad (4)$$

where $\boldsymbol{K} = \mathrm{k}(\mathcal{D}, \mathcal{D})$ represents the kernel matrix, $\boldsymbol{k} = \mathrm{k}(\mathcal{D}, \boldsymbol{d}_{m+1})$ is the kernel vector and $k = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$ denotes a kernel value. Note that in Equation (4) we make use of the property that inner products of feature vectors can be represented as kernel values. Minimizing Equation (4) yields the optimal coefficient vector $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$. The parameter $\boldsymbol{a}$ from Equation (4) can be further regularized taking in account problems such as outliers. The regularization can be controlled by a regularization-parameter which can be chosen to alleviate the outlier's contribution to the sparsication process. Usually, this parameter can be obtained from training data by cross-validation. However, in this paper we omitted this regularization as we did not observe any serious problems due to outliers. After substituting the optimal value $\boldsymbol{a}$ into Equation (3), δ becomes

$$\delta = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1}) - \boldsymbol{k}^T \boldsymbol{a} \,. \qquad (5)$$

Using δ we can decide whether to insert new data points into the dictionary. The sparsification procedure is summarized in Algorithm 1.

Algorithm 1 Independence test with online updates of the dictionary.

Input: new point $\boldsymbol{d}_{m+1}$, threshold η, $\mathrm{N}_{\max}$.
Compute: $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$ with $\boldsymbol{k} = \mathrm{k}(\mathcal{D}, \boldsymbol{d}_{m+1})$,
Compute: $\delta = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1}) - \boldsymbol{k}^T \boldsymbol{a}$,
if $\delta > \eta$ **then**
  if number of dictionary points $< \mathrm{N}_{\max}$ **then**
    Insert $\boldsymbol{d}_{m+1}$ into $\mathcal{D}$ and update the dictionary using Algorithm 2.
  else
    Insert $\boldsymbol{d}_{m+1}$ into $\mathcal{D}$ and update the dictionary using Algorithm 3, while replacing an old dictionary point from $\mathcal{D}$.
  end if
  Incremental online model training using the new dictionary $\mathcal{D} = \mathcal{D} \cup \boldsymbol{d}_{m+1}$.
end if

The independence measure δ as given in Equation (5), can be used as a criterion for selecting new data points by thresholding, see Algorithm 1. The threshold parameter η implicitly controls the level of sparsity. However, for a given threshold value η, the number of dictionary points selected from the online data stream is *not* known beforehand and, thus, can be very large. Large dictionaries are prohibitively expensive in terms of computational complexity and as a result not real-time capable. To cope with this problem, we need to define an upper bound on the dictionary size and delete old dictionary points if this limit is reached.

For deleting old dictionary points, we consider the independence measures δ^i of every dictionary point i as illustrated in Figure 3. The value δ^i indicates, how well the dictionary point i can be approximated by the remainder of the dictionary. The score δ^i has to be updated, when a new data point is inserted into the dictionary or an old data point is removed from the dictionary. In Sections

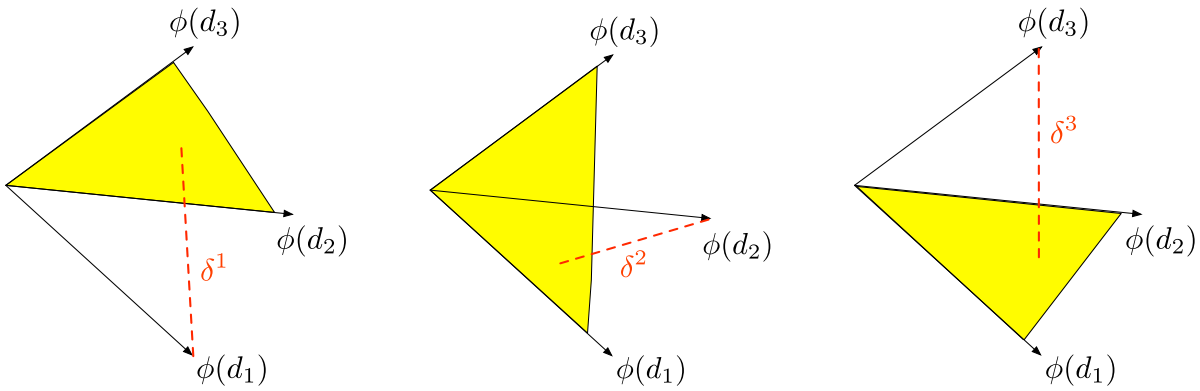


Figure 3: Geometrical interpretation of individual independence measures $\delta^i, i=1..3$, for the dictionary $\mathcal{D}=\{\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3\}$. The dictionary point with a *minimal* δ^i is more likely to be deleted. After every dictionary operation (e.g., insertion and deletion of dictionary points), the individual independence measures δ^i have to be incrementally updated.

2.3 and 2.4, we will discuss an efficient, incremental way of updating the value δ^i applicable for real-time online computation.

2.3. Dictionary Update for Inserting New Points

For inserting a new data point $\boldsymbol{d}_{m+1}$ into the dictionary, we have to incrementally update the independence measure δ^i and corresponding coefficients for *every* dictionary point i, as changing the dictionary implies a change for δ^i. This update for an existing dictionary point $\boldsymbol{d}_i$ is achieved by adjusting the corresponding coefficient vector $\boldsymbol{a}^i$. Updating $\boldsymbol{a}^i$ implies an update of $(\boldsymbol{K}^i)^{-1}$ and $\boldsymbol{k}^i$. Here, inserting a new point will extend $\boldsymbol{K}^i$ by a row/column and $\boldsymbol{k}^i$ by a value, respectively, such that

$$\boldsymbol{K}^i_{\text{new}} = \begin{bmatrix} \boldsymbol{K}^i_{\text{old}} & \boldsymbol{k}_{m+1} \\ \boldsymbol{k}^T_{m+1} & k_{m+1} \end{bmatrix}, \qquad \boldsymbol{k}^i_{\text{new}} = \begin{bmatrix} \boldsymbol{k}^i_{\text{old}} & k_{i,m+1} \end{bmatrix}^T, \tag{6}$$

where $k_{m+1}=\mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$, $k_{i,m+1}=\mathrm{k}(\boldsymbol{d}_i, \boldsymbol{d}_{m+1})$, $\boldsymbol{k}_{m+1} = \mathrm{k}(\mathcal{D}^i, \boldsymbol{d}_{m+1})$ with $\mathcal{D}^i = \mathcal{D}\backslash\{\boldsymbol{d}_i\}$. Using the Equation (6), the incremental update of the inverse matrix $(\boldsymbol{K}^i_{\text{new}})^{-1}$ is given by

$$\left(\boldsymbol{K}^i_{\text{new}}\right)^{-1} = \frac{1}{\gamma_i} \begin{bmatrix} \gamma_i (\boldsymbol{K}^i_{\text{old}})^{-1} + \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^T & -\boldsymbol{\alpha}_i \\ -\boldsymbol{\alpha}_i^T & 1 \end{bmatrix}. \tag{7}$$

This result leads to the update rule for the linear independency value δ^i for the i-th dictionary point given by

$$\delta^i = \mathrm{k}(\boldsymbol{d}_i, \boldsymbol{d}_i) - \boldsymbol{k}^{iT}_{\text{new}} \boldsymbol{a}^i_{\text{new}}, \quad \text{with}$$
$$\boldsymbol{a}^i_{new} = \frac{1}{\gamma_i} \begin{bmatrix} \gamma_i \boldsymbol{a}^i_{\text{old}} + \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^T \boldsymbol{k}^i_{\text{old}} - k_{i,m+1} \boldsymbol{\alpha}_i \\ -\boldsymbol{\alpha}_i^T \boldsymbol{k}^i_{\text{old}} + k_{i,m+1} \end{bmatrix}. \tag{8}$$

The variables γ_i and $\boldsymbol{\alpha}_i$ are determined by $\boldsymbol{\alpha}_i = (\boldsymbol{K}^i_{\text{old}})^{-1}\boldsymbol{k}_{\text{m+1}}$ and $\gamma_i = k_{m+1} - \boldsymbol{k}^T_{m+1}\boldsymbol{\alpha}_i$. The procedure for the dictionary update after insertion of new data points is summarized in Algorithm 2.

Algorithm 2 Update the dictionary after inserting a new data point.

Input: new dictionary point $\boldsymbol{d}_{m+1}$.
Update dictionary $\mathcal{D} = \{\boldsymbol{d}_i\}_{i=1}^{m+1}$.
for $i=1$ **to** m **do**
  Compute
    $\boldsymbol{k}_{m+1} = \mathrm{k}(\mathcal{D}^i, \boldsymbol{d}_{m+1})$,
    $k_{m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$,
    $k_{i,m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_i)$.
  Compute
    $\boldsymbol{\alpha}_i = \left(\boldsymbol{K}^i_{\text{old}}\right)^{-1} \boldsymbol{k}_{m+1}$,
    $\gamma_i = k_{m+1} - \boldsymbol{\alpha}_i^T \boldsymbol{k}_{m+1}$.
  Update δ^i as given in Equation (8).
  Update $(\boldsymbol{K}^i_{new})^{-1}$ as given in Equation (7).
end for

2.4. Dictionary Update for Replacing Points

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 and the real-time constraints are not violated. The individual independence value δ^i – as illustrated in Figure 3 – gives rise to a straightforward deletion rule: the *smaller* the independence value δ^i is, the more likely the corresponding dictionary point is to be deleted. The idea is to delete points that are more *dependent* on other dictionary points, i.e., where the corresponding independence value δ^i is small. For the deletion of dictionary points, we additionally consider a temporal allocation of each dictionary point by imposing a time-dependent forgetting rate $\lambda^i \in [0,1]$. Thus, we take the independency value δ^i weighted by the forgetting value λ^i as a deleting score. The role of λ^i will be discussed in detail in Section 2.5.

Insertion and additional deletion of dictionary points also change the independence values of other dictionary points which have to be updated subsequently. Insertion of a new point with an additional deletion of the j-th dictionary point implies a manipulation of the j-th row/column of $\boldsymbol{K}^i$ and the j-th value of $\boldsymbol{k}^i$ given by

$$\boldsymbol{K}^i_{new} = \begin{bmatrix} \boldsymbol{K}^i_{\text{old}(1:j)} & \boldsymbol{k}_{m+1(1:j)} & \boldsymbol{K}^i_{\text{old}(j:m)} \\ \boldsymbol{k}^T_{m+1(1:j)} & k_{m+1} & \boldsymbol{k}^T_{m+1(j:m)} \\ \boldsymbol{K}^{iT}_{\text{old}(j:m)} & \boldsymbol{k}_{m+1(j:m)} & \boldsymbol{K}^i_{\text{old}(j:m)} \end{bmatrix}, \quad (9)$$

$$\boldsymbol{k}^i_{\text{new}} = \left[\boldsymbol{k}^i_{\text{old}(1:j)}\, k_{i,m+1}\, \boldsymbol{k}^i_{\text{old}(j:m)}\right]^T.$$

The values $\boldsymbol{k}_{m+1}$, k_{m+1} and $k_{i,m+1}$ are determined as shown in Section 2.3. The incremental update of the independence measure δ^i for every i-th dictionary point can be performed directly using an incremental matrix inverse update. Hence,

$$\delta^i = \mathrm{k}(\boldsymbol{d}_i, \boldsymbol{d}_i) - \boldsymbol{k}^{iT}_{\text{new}} \boldsymbol{a}^i_{\text{new}} \text{ and } \boldsymbol{a}^i_{\text{new}} = \boldsymbol{A}\, \boldsymbol{k}^i_{\text{new}}, \quad (10)$$

where $\boldsymbol{A}$ is computed by the update rule

$$\boldsymbol{A} = \boldsymbol{A}^* - \frac{\mathrm{row}_j[\boldsymbol{A}^*]^T \boldsymbol{r}^T \boldsymbol{A}^*}{1 + \boldsymbol{r}^T \mathrm{row}_j[\boldsymbol{A}^*]^T} \quad \text{with}$$

$$\boldsymbol{A}^* = (\boldsymbol{K}^i_{\text{old}})^{-1} - \frac{(\boldsymbol{K}^i_{\text{old}})^{-1} \boldsymbol{r}\, \mathrm{row}_j[(\boldsymbol{K}^i_{\text{old}})^{-1}]}{1 + \boldsymbol{r}^T \mathrm{row}_j[(\boldsymbol{K}^i_{\text{old}})^{-1}]^T}.$$

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

Algorithm 3 Replace the least important data point in the dictionary by a new one.

Input: new dictionary point $\boldsymbol{d}_{m+1}$.
Compute λ^i as given in Equation (12) and, subsequently, find the dictionary point with minimal δ^i weighted by the forgetting rate λ^i:
$j = \min_i(\lambda^i \delta^i)$.
Update dictionary $\mathcal{D}$ by overwriting point j:
$\boldsymbol{d}_j = \boldsymbol{d}_{m+1}$.
for $i = 1$ **to** m **do**
Compute
$\boldsymbol{k}_{m+1} = \mathrm{k}(\mathcal{D}^i, \boldsymbol{d}_{m+1})$,
$k_{m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{m+1})$,
$k_{i,m+1} = \mathrm{k}(\boldsymbol{d}_{m+1}, \boldsymbol{d}_i)$.
Compute
$\boldsymbol{r} = \boldsymbol{k}_{m+1} - \mathrm{row}_j[\boldsymbol{K}^i_{\text{old}}]^T$.
Update δ^i as given in Equation (10).
Update $(\boldsymbol{K}^i_{\text{new}})^{-1} = \boldsymbol{A}$ in Equation (10).
end for

2.5. Characterization of the Dictionary Space and Temporal Allocation

In previous sections, we described the procedures for incremental insertion and deletion of dictionary points appropriate for real-time computation. The basic idea is that we attempt to approximate the dictionary space (characterized by the dictionary vector $\boldsymbol{d}$) at best using a limited sparse data set $\mathcal{D}$. However, the choice of the dictionary space is a crucial step which may depend on particular applications. As we are dealing with regression in this paper, it is reasonable to choose the dictionary vector as

$$\boldsymbol{d} = \left[\, \boldsymbol{x} \quad \boldsymbol{y} \,\right]^T, \quad (11)$$

where $\boldsymbol{x}$ is the input vector and $\boldsymbol{y}$ represents the target values. In so doing, our dictionary space will include the input as well as target distributions. In practice, it shows that input and target distributions both incorporate relevant information for model approximation by regression. For other applications such as learning classification models, considering the input space only might be sufficient.

In addition to the spatial allocation of the dictionary space as achieved by employing the inde-

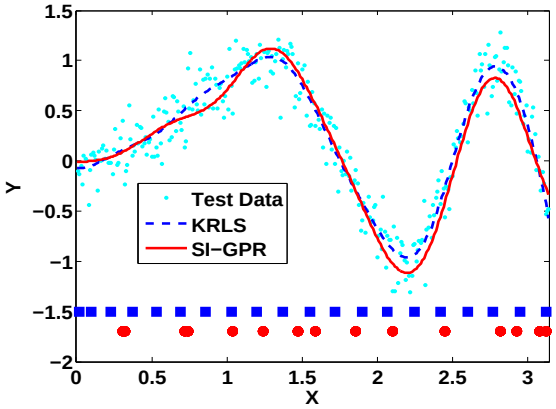
pendence measure δ, temporal allocation can be taken into account by introducing a time-variant forgetting factor for every dictionary point i. Here, we consider the forgetting rate

$$\lambda^i(t) = \exp\left(-\frac{(t-t_i)^2}{2h}\right), \qquad (12)$$

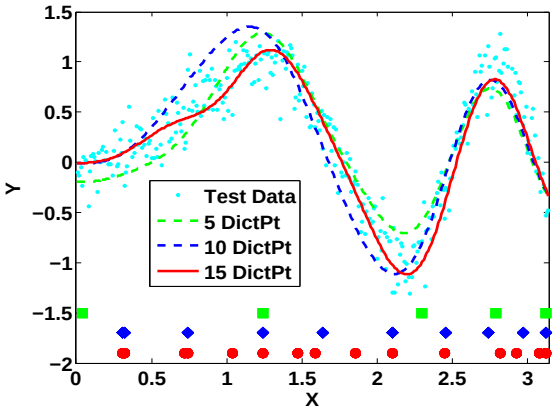
where t_i is the time when the dictionary point i is included into the sparse set and h represents the intensity of the forgetting rate. For deleting a point from the dictionary, the deletion score is the independence value weighted by the corresponding forgetting value $\lambda^i \in [0,1]$, as shown in Algorithm 3. By imposing a temporal weight on the independence values, we make sure that temporal information encoded in the data is sufficiently taken in account for the model learning. The forgetting score λ (controlled by the parameter h) represents a trade-off between temporal and spatial covering. If the h value is very small, the temporal effects will gain more importance and the sparsication will behave similar to a time window in common online algorithms.

2.6. Comparison to Previous Work

Our work was inspired by the work in [3] where the authors also use a linear independence measure to select dictionary points. However, they do not remove dictionary points again but instead show that the dictionary size is bounded for a given threshold η if the data space is assumed to be compact. In practice, for a given threshold value the actual dictionary size is data dependent and unknown beforehand, thus, the dictionary can be very large. In order to cope with the computational constraints in real-time applications, the dictionary size has to be limited. In contrast to the algorithm in [3], our approach allows us to formulate an efficient insertion and deletion procedure for a given budget. In [3], the spatial allocation is performed in the input space only, resulting in an uniform covering of the complete state space which might be suboptimal for model approximation by regression. As we additionally employ a temporal allocation, the resulting online learning algorithm is able to adapt the model to temporal effects which is an important issue for online learning on real systems.



(a) **Comparison with KRLS**



(b) **SI-GPR prediction with different budgets**

Figure 4: An illustrative example of the prediction after a single sweep through a toy data set. The dots show the positions of the corresponding dictionary points in the input space. **(a)** The performance of SI-GPR is quite similar to KRLS while the selection of the sparse data sets poses the main difference. SI-GPR selects 15 dictionary points and KRLS 21 data points, respectively. **(b)** Using a fixed budget, e.g., 5, 10 or 15 dictionary points, the most relevant regions in the input space are covered with data points.

An Illustrative Toy-Example

In the following section, we compare the kernel recursive least square (KRLS) [3] with our approach. For the model training, we combine the sparsification with an incremental GPR learning [10], called sparse incremental GPR (SI-GPR). A quick review for GPR and its incremental updates can be found in the Appendix. It should be noted that other incremental model learning methods can also be used in combination with our incremental sparsification, such as sequential SVR or incremental SVM [19, 2].

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 set consists of 315 points, where ε_i is white Gaussian noise with standard deviation 0.2 and x_i ranges from 0 to π. Here, the data is incrementally fed to the algorithms and the prediction is computed after a single sweep through the data set. The results are shown in Figure 4, where $\eta = 0.3$ and a Gaussian kernel with width 0.3 is being used. In Figure 4 (a), it can be seen that the prediction performance of SI-GPR using dictionary is quite similar to KRLS. Here, the selection of the sparse data points presents the main difference. While KRLS uniformly fills up the complete input space (resulting in 21 dictionary points), the sparsification used by SI-GPR selects the most relevant data points as dictionary points, where the limit of the dictionary is set to be 15. Figure 4 (b) shows the performance of SI-GPR for different dictionary sizes, where the prediction improves as expected with increasing dictionary size.

3. Evaluations

In this section, we evaluate our sparsification approach used in combination with the incremental GPR (SI-SVR) in several different experimental settings with a focus on inverse dynamics modeling for robot tracking control.

First, we give a short review of learning dynamics models for control. Subsequently, we evaluate the algorithm in the context of learning inverse dynamics. The learning accuracy of SI-GPR will be compared with other regression methods, i.e., LWPR [18], GPR [12], ν-SVR [16] and LGP [10]. For this evaluation in inverse dynamics learning, we employ 2 data sets including synthetic data, as well as real robot data generated from the 7 degrees-of-freedom (DoF) Barrett WAM, shown in Figure 6 (a). In Section 3.3, SI-GPR is applied for real-time online learning of inverse dynamics models for robot computed torque control (state-of-the-art batch regression methods can not be applied for online model learning). Finally, in Section 3.4, we demonstrate the capability of the approach in online learning of a dynamics which changes in presence of different loads.

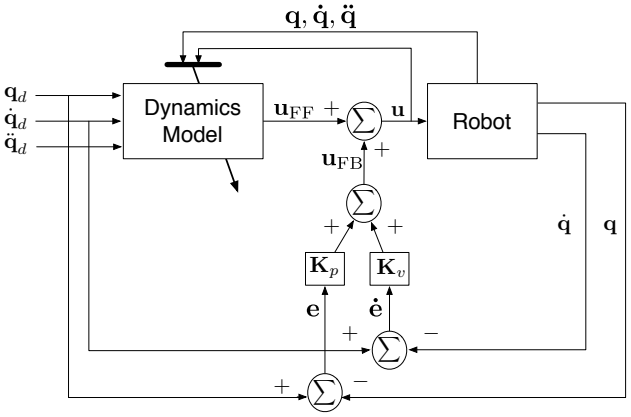
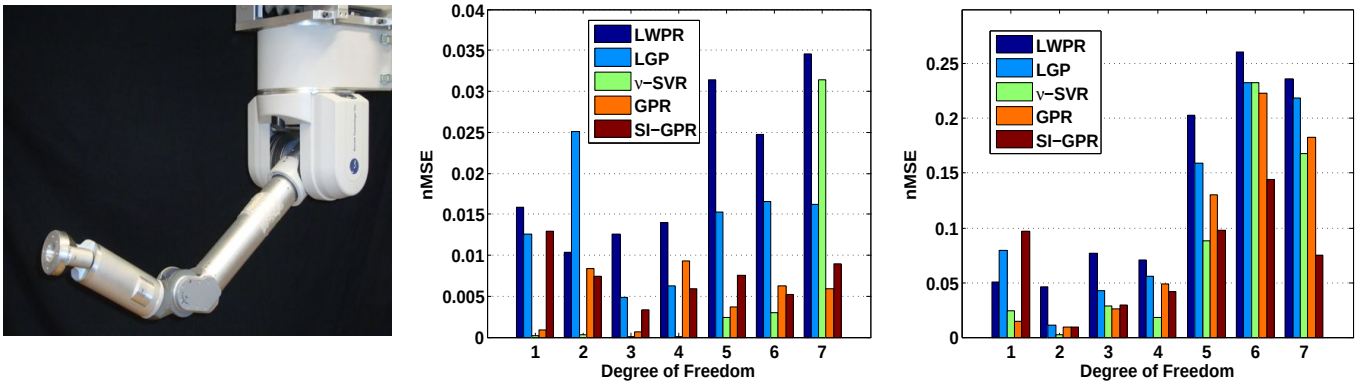


Figure 5: Feedforward control with online model learning.

3.1. Learning Dynamics Models for Control

Model-based tracking control laws [17] determine the joint torques $\mathbf{u}$ that are required for the robot to follow a desired trajectory $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, where $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$ denote the desired joint angles, velocity and acceleration. In feedforward control, the motor command $\mathbf{u}$ consists 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, as shown in Figure 5. 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 [17].

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 accurately drive the robot along the desired trajectory. Due to the high complexity of modern robot systems such as humanoids or service robots, traditional analytical rigid-body model often cannot provide a sufficiently accurate inverse dynamics model. The lack of model precision has to be compensated by increasing the tracking gains $\mathbf{K}_p$ and $\mathbf{K}_v$ making the robot stiff and less safe for the environment [9]. Thus, to fulfill both requirements of compliant control, i.e., having low tracking gains and high tracking accuracy, more precise models are necessary. One possibility to obtain an accurate inverse dynamics model is to learn it *directly* from measured data. The resulting problem is a



(a) **Barrett WAM** (b) **Error on simulation data** (c) **Error on real WAM data**

Figure 6: **(a)** 7-DoF Barrett WAM, used for evaluations. **(b)** Error (nMSE) on simulated data from a robot model for every DoF. **(c)** Error (nMSE) on real robot data for every DoF. As shown by the results, SI-GPR is competitive to standard batch regression methods such as ν-SVR and GPR, local learning methods such as LWPR and LGP. For model learning with SI-GPR, the dictionary size is limited to be 2000 data points.

regression problem by approximating the inverse dynamics model $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$ from sampled data [1]. The resulting mapping can be subsequently applied for predicting the appropriate feedforward motor commands $\mathbf{u}_{\text{FF}}$. As trajectories and corresponding joint torques are sampled directly from the real robot, learning the inverse dynamics will include all nonlinearities of the system encoded by the data.

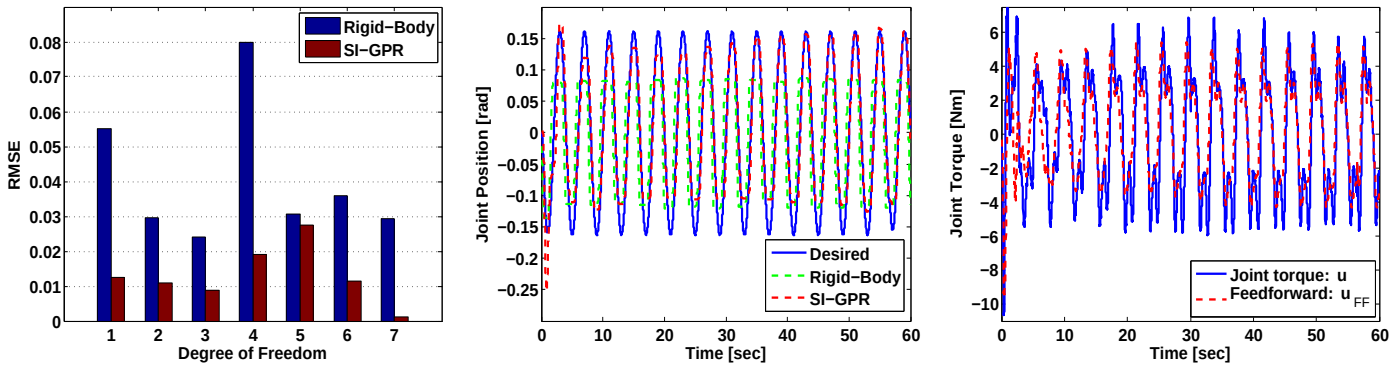
If the dynamics model can be learned online (see Figure 5), the robot controller can adapt itself to changes in the robot dynamics, e.g., due to unforeseen load or time-variant disturbances. Online learning of dynamics models using sampled data realized in a setup as in Figure 5 can be considered as a self-supervised learning problem.

3.2. Offline Comparison in Learning Inverse Dynamics

For the generation of two data sets, i.e., one with simulation data and one with real robot data, we sample joint space trajectories and corresponding torques from an analytical model of the Barrett WAM, as well as from the real robot. This results in two test scenarios, each having 12000 training points and 3000 test points. Given samples $\mathbf{x}=[\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as input and using the corresponding joint torques $\mathbf{y}=\mathbf{u}$ as targets, we have a regression problem with 21 input dimensions and 7 output dimensions (i.e., a single desired torque for each motor joint). The robot inverse dynamics model is estimated separately for each DoF employing LWPR, ν-SVR, GPR, LGP and SI-GPR.

Employing SI-GPR for learning the inverse dynamics model, we first sparsify the full data sets (i.e., 12000 data points) as described in Section 2, and subsequently apply incremental GPR for an offline approximation of the model. For all data sets, the dictionary size is limited to be 2000 points, where the parameter η is set to be 0.1. For the sparsification process and the incremental GPR, we employ the Gaussian kernel, whose hyperparameters are obtained by optimizing the marginal likelihood [12].

Figures 6 (b) and (c) show the offline approximation errors on the test sets evaluated using the normalized mean square error (nMSE) which is defined as the fraction of mean squared error and the variance of target. It can be observed from the results that SI-GPR using sparsification is competitive in learning accuracy despite the smaller amount of training examples (i.e., dictionary points). In practice, it also shows that the models are easier to train using SI-GPR compared to local learning methods such as LWPR and LGP, as the latter require an appropriate clustering of the state space which is not straightforward to perform for many data sets.



(a) **Tracking error on Barrett WAM in joint space for all 7 DoFs**

(b) **Tracking error on Barrett WAM in joint space for the 1st DoF**

(c) **Feedforward $\mathbf{u}_{\text{FF}}$ and joint torque u of the 1st DoF during SI-GPR online learning**

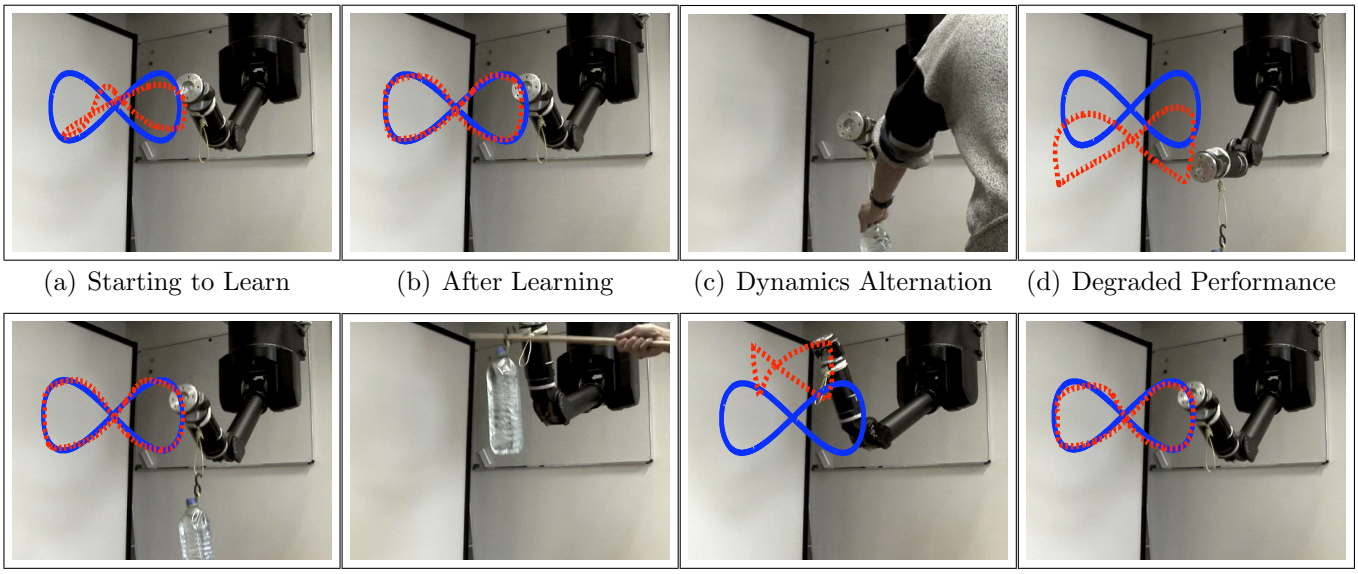
Figure 7: **(a)** Compliant tracking performance in joint space after 60 sec computed as RMSE. Computed torque control with online learned model (SI-GPR) outperforms common analytical rigid-body model. **(b)** Tracking performance in joint space for the 1st DoF, e.g., the shoulder flexion extension, (other DoFs are similar) during the first 60 sec. **(c)** Predicted feedforward torque $\mathbf{u}_{\text{FF}}$ and joint torque $\mathbf{u}$ of the 1st DoF. The predicted torque $\mathbf{u}_{\text{FF}}$ gradually converges to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning.

3.3. Model Online Learning in Computed Torque Control

In this section, we apply SI-GPR for the real-time online learning of the inverse dynamics model for torque prediction in robot tracking control, as introduced in Section 3.1. The control task with model online learning is performed with 500 Hz sample rate (i.e., we get a new test point every 2 ms) on the real Barrett WAM. In so doing, the trajectory $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ and the corresponding joint torques $\mathbf{u}$ are sampled online as shown in Figure 5. The inverse dynamics model, i.e., the mapping $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$, is learned in a self-supervised manner during the tracking task using the trajectory as input and the joint torque as target, starting with an empty dictionary. As the learned inverse dynamics model is subsequently applied to compute the feedforward torques $\mathbf{u}_{\text{FF}}$, it can be observed that $\mathbf{u}_{\text{FF}}$ gradually converges against $\mathbf{u}$. In this experiment, we set the tracking gains (see Section 3.1) to very low values satisfying the requirement of compliant control. In so doing, inaccurate inverse dynamics models, however, will result in large tracking errors.

In this online learning experiment, the maximal dictionary size is set to be 300, $\eta = 0.01$ and a Gaussian kernel is used whose parameters are determined by optimizing the marginal likelihood. For the forgetting score λ, a forgetting rate $h = 0.4$ is chosen by cross-validation. The desired tracking trajectory is generated such that the robot's end-effector follows a 8-figure in the task space. For comparison, we also apply an analytical rigid-body model for torque prediction [17]. The tracking results are shown in Figure 7, where tracking control task is performed for 60 sec on Barrett WAM. During this time, the dictionary is first incrementally filled up and subsequently updated about 700 times. The inverse dynamics model is incrementally learned online using the dictionary.

Figure 7 (a) compares the tracking performance of the online learned model and the analytical rigid-body model in joint space for all 7 DoFs, where the tracking error is computed as root mean square error (RMSE). It can be observed that the tracking accuracy can be significantly improved, if the inverse dynamics model is approximated online. For several DoFs such as the 4th DoF (i.e, the elbow flexion), the rigid-body model fails to described the true dynamics resulting in a large tracking error as shown in Figure 7 (a). This example demonstrates the difficulty in analytically modeling complex systems which may be alleviated by directly learning the model.



(a) Starting to Learn (b) After Learning (c) Dynamics Alternation (d) Degraded Performance

(e) After Learning (f) Dynamics Alternation (g) Degraded Performance (h) After Learning

Figure 8: Tracking experiment with online model learning. The blue, thick lines illustrate the desired trajectory and the red, dotted lines show the robot trajectory in task space. **(a)** As the model is started to be learned online (beginning with an empty dictionary), the robot first shows a transient behavior. **(b)** With a successfully learned model, the robot is able to follow the trajectory well. **(c)** The original dynamics is changed by hanging a heavy water bottle to the arm. **(d)** Due to the modified dynamics, the robot fails to track the desired initial trajectory. Therefore, the robot starts to learn the modified dynamics online. **(e)** As the online model learning converges, the robot gradually moves back to the desired initial position. **(f)** The dynamics is modified again by removing the water bottle. **(f)** The learned model is no more accurate, i.e., the predicted torques are now too large. The modified dynamics is subsequently adapted online. **(g)** As the dynamics model is successfully adapted, the robot returns to the initial desired position. A video showing the experiment can be seen at "http://www.robot-learning.de/".

Figures 7 (b,c) show the tracking performance in joint space for the 1st DoF, i.e., the shoulder flexion extension, during the first 60 sec, other DoFs are similar. It can be seen that the predicted torque $\mathbf{u}_{\text{FF}}$ (for 1st DoF) consistently converges to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning.

3.4. Online Learning for Changing Dynamics

In this section, we demonstrate the capability of the algorithm for online model learning and self-adaption to changes in the dynamics in a more complex task. Figures 8 (a)-(h) show the progress of the experiment. First, we learn the robot inverse dynamics online starting with an empty dictionary. We apply SI-GPR with the same parameter setting as given in Section 3.3. Here, we also first incrementally fill up the dictionary and subsequently update for new data points. Subsequently, we modify the robot dynamics by attaching a heavy water bottle to the arm (beside the changing load, the swinging bottle additionally introduces a time-variant nonlinearity). Due to the change in the dynamics, the robot cannot follow the given trajectory in joint space. As in Section 3.3, the desired joint space trajectory is defined such that the robot draw a figure-8 in the task space. The end-effector velocity of the robot during the tracking in task space is about 0.6 m/sec, where the displacement ranges from 0.2 to 0.5 m. The modified dynamics can now be learned online, as shown in the Figures 8 (a)-(h).

As the online model learning converges, the modified dynamics is taken into account by the predicted feedforward torques $\mathbf{u}_{\text{FF}}$. As an example, Figure 9 shows the joint angle and corresponding torques of the 4th DoF (i.e., the robot's elbow flexion) during the experiment. With the attached water bottle, the predicted feedforward torques $\mathbf{u}_{\text{FF}}$ gradually becomes more precise as the model converges. The decreasing model error results in accurate tracking performance. By

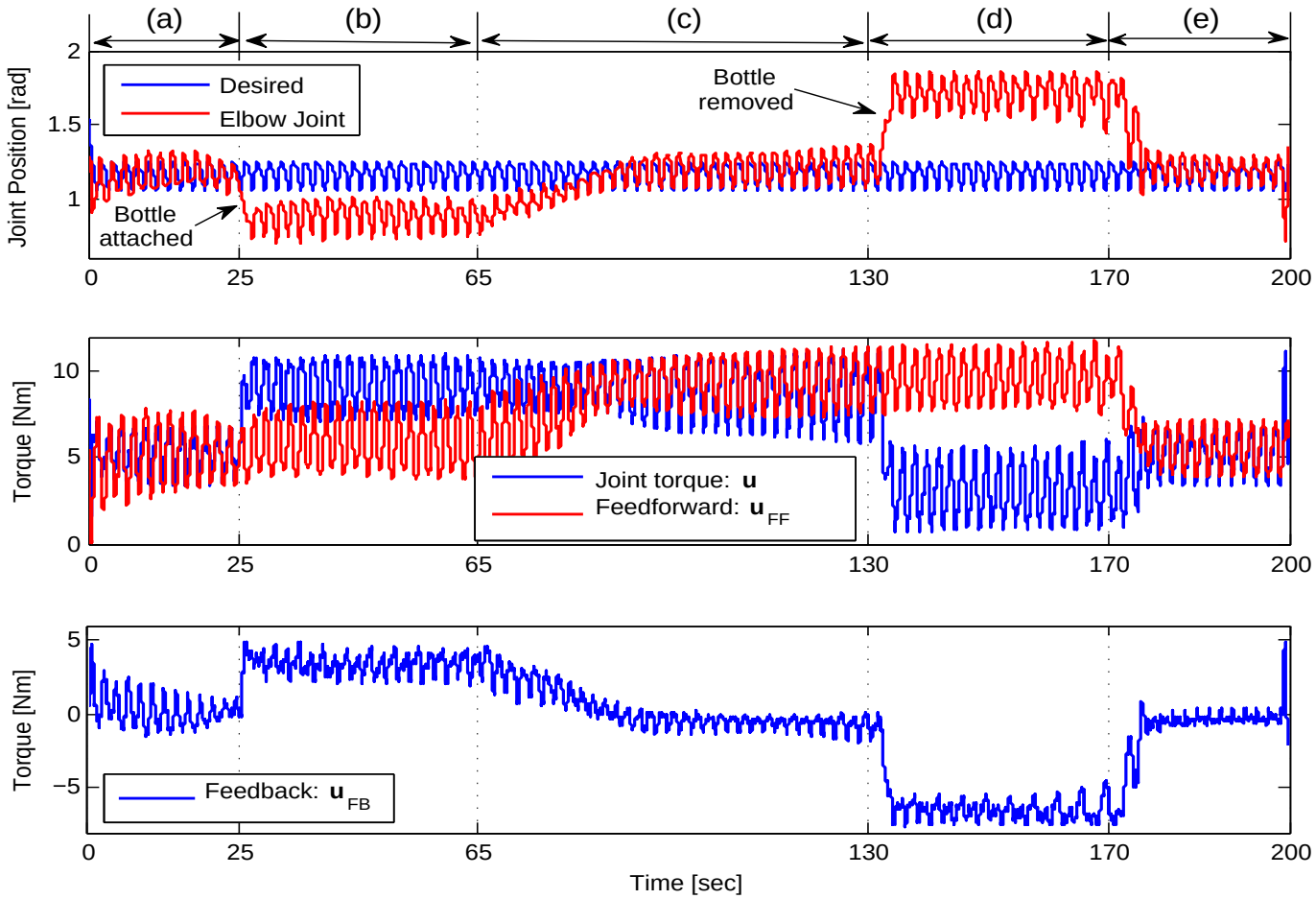


Figure 9: Elbow joint angle (4th DoF) and corresponding torques of the Barrett WAM during the experiment. **(a)** The robot starts with an unmodified dynamics. **(b)** As the water bottle is attached to the arm, it causes a jump in the feedback torque $\mathbf{u_{FB}}$ and joint torque $\mathbf{u}$. Due to the compliant tracking mode, the change in the feedback torque $\mathbf{u_{FB}}$ is not sufficient to compensate the resulting tracking error. **(c)** As the dynamics model is learned online, the new dynamics can be incorporated in the prediction of the feedforward torque $\mathbf{u}_{\mathrm{FF}}$. As the online model learning converges, the resulting tracking error is also gradually reduced, i.e., the robot returns to the desired position. Note how the feedback torque $\mathbf{u_{FB}}$ is decreasing, as the model, i.e., the torque prediction, becomes more accurate. **(d)** The online modification of the dynamics, i.e., removing the water bottle, leads to changes in the feedback and joint torques. **(e)** As the adaptation is successfully done and the feedforward torque $\mathbf{u_{FF}}$ converges, the robot moves back to the desired trajectory and the feedback torque decreases again.

continuously updating the dictionary, i.e., by insertion and eventual deletion of dictionary points, the model can adapt to dynamical changes in the environment. This effect can further be demonstrated by removing the water bottle. As observed from Figure 9, the predicted torque $\mathbf{u}_{\mathrm{FF}}$ is subsequently reduced, since smaller torques are now required for proper tracking of the desired trajectory.

In this experiment, we employ very low tracking gains. As a result, the model errors will seriously degrade the tracking accuracy unless a more precise inverse dynamics model is learned. Figure 9 shows that the feedback torques $\mathbf{u}_{\mathrm{FB}}$ are not able to compensate the model error due to the low tracking gains. As the inverse dynamics model becomes more accurate during the online learning, the feedback torques $\mathbf{u}_{\mathrm{FB}}$ decrease, and, thus, the joint torques $\mathbf{u}$ mainly depend on the feedforward term $\mathbf{u}_{\mathrm{FF}}$. As the torque generation relies more on the inverse dynamics model, we can achieve both compliant control (by using low tracking gains) and high tracking accuracy at the same time.

4. Conclusion and Future Work

Motivated by the need of fast online model learning in robotics, we have developed an incremental sparsification framework which can be used in combination with an online learning algorithm enabling an application in real-time online model learning. The proposed approach provides a way to efficiently insert and delete dictionary points taking in account the required fast computation during model online learning in real-time. The implementation and evaluation on a physical Barrett WAM robot emphasizes the applicability in real-time online model learning for real world systems. Our future research will be focused on further extensions such as including a database enabling online learning for large data sets.

Acknowledgments

We would like to thank Bernhard Schölkopf from Max Planck Institute for Biological Cybernetics for helpful discussions and for providing us with relevant machine learning literature of reduced set methods.

References

[1] Burdet, E., Codourey, A., 1998. Evaluation of parametric and nonparametric nonlinear adaptive controllers. Robotica 16 (1), 59–73.

[2] Cauwenberghs, G., Poggio, T., 2000. Incremental and decremental support vector machine learning. Advances in Neural Information Processing Systems.

[3] Engel, Y., Mannor, S., Meir, R., 2004. The kernel recursive least-square algorithm. IEEE Transaction on signal processing 52.

[4] Hastie, T., Tibshirani, R., Friedman, J., 2001. The Elements of Statistical Learning. Springer, New York.

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

[6] Liu, Y., Wang, H., Yu, J., Lia, P., 2009. Selective recursive kernel learning for online identification of nonlinear systems with NARX form. Journal of Process Control, 181–194.

[7] M.Seeger, 2007. Low rank update for the cholesky decomposition. Tech. rep., University of California at Berkeley.

[8] Nakanishi, J., Schaal, S., 2004. Feedback error learning and nonlinear adaptive control. Neural Networks.

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

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

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

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

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

[14] Schölkopf, B., Mika, S., Burges, C. J. C., Knirsch, P., Müller, K.-R., Rätsch, G., Smola, A. J., 1999. Input space versus feature space in kernel-based methods. IEEE Transactions on Neural Networks 10 (5), 1000–1017.

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

[16] Schölkopf, B., Smola, A., Williamson, R., Bartlett, P., 2000. New support vector algorithms. Neural Computation.

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

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

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

Appendix

Gaussian Process Regression

A powerful probabilistic method for model learning is Gaussian process regression (GPR). Given a set of n training data points $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we intend to discover the latent function $f(\mathbf{x}_i)$ which transforms the input vector $\mathbf{x}_i$ into a target value y_i given by the model $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i is Gaussian noise with zero mean and variance σ_n^2 [12]. A Gaussian process model is determined by its covariance function and mean function which is

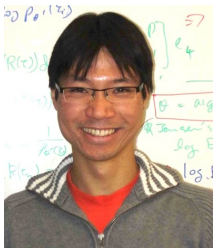
often assumed to be zero. In practice, the covariance function can be chosen by the users such as a Gaussian kernel as given in Equation (2). The resulting prediction $\bar{f}(\mathbf{x}_*)$ and the corresponding variance $\mathbf{V}(\mathbf{x}_*)$ of a query input vector $\mathbf{x}_*$ can be given as

$$\begin{aligned}\bar{f}(\mathbf{x}_*) &= \mathbf{k}_*{}^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{y}\,, \\ &= \mathbf{k}_*{}^T \boldsymbol{\alpha}\,, \\ \mathbf{V}(\mathbf{x}_*) &= \mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*{}^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{k}_*\,,\end{aligned} \tag{13}$$

Here, $\mathbf{K} = \mathrm{k}(\mathbf{X}, \mathbf{X})$ denotes the covariance matrix evaluated on the training input data $\mathbf{X}$, $\mathbf{k}_* = \mathrm{k}(\mathbf{X}, \mathbf{x}_*)$ is the covariance vector evaluated on $\mathbf{X}$ and the query point $\mathbf{x}_*$, and $\boldsymbol{\alpha} = (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1}\mathbf{y}$ is the so-called prediction vector. The open parameters of GPR, i.e., the hyperparameters, can be optimized from training data. The usual practice is to maximize the log marginal likelihood using common optimization procedures, e.g., quasi-Newton methods [12].

The GPR model as introduced here can also be obtained incrementally for online applications [10]. Essentially, the matrix $(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1}$ in Equation (13) has to be incrementally updated when including a new point [10]. This approach is equivalent to a rank-one update which yields a complexity of $\mathcal{O}(m^2)$, where m denotes the current number of training data points [7]. For real-time applications, it is necessary to define an upper bound for m in order to cope with the limited computational power. Thus, it is essential to develop a method for incrementally selecting a limited number of *informative* data points making the regression real-time capable.

The Authors



Duy Nguyen-Tuong has been pursuing his Ph.D. since 2007 at the Max Planck Institute for Biological Cybernetics in the department of Bernhard Schölkopf supervised by Jan Peters. Before doing so, he studied control and automation engineering at the University of Stuttgart and the National University of Singapore. His main research interest is the application of machine learning techniques in control and robotics.

Jan Peters is a senior research scientist and heads the Robot Learning Lab (RoLL) at the Max Planck Institute for Biological Cybernetics in Tuebingen, Germany. He graduated from University of Southern California (USC) with a Ph.D. in Computer Science. He holds two German M.S. degrees in Informatics and in Electrical Engineering (from Hagen University and Munich University of Technology) and two M.S. degrees in Computer Science and Mechanical Engineering from USC. Jan Peters has been a visiting researcher at the Department of Robotics at the German Aerospace Research Center (DLR) in Oberpfaffenhofen, Germany, at Siemens Advanced Engineering (SAE) in Singapore, at the National University of Singapore (NUS), and at the Department of Humanoid Robotics and Computational Neuroscience at the Advanded Telecommunication Research (ATR) Center in Kyoto, Japan. His research interests include robotics, nonlinear control, machine learning, reinforcement learning, and motor skill learning.