Online Kernel-based Learning for Task-Space Tracking Robot Control

Duy Nguyen-Tuong, Jan Peters

***Abstract*—Task-space control of redundant robot systems based on analytical models is known to be susceptive to modeling errors. Here, data driven model learning methods may present an interesting alternative approach. However, learning models for task-space tracking control from sampled data is an ill-posed problem. In particular, the same input data point can yield many different output values, which can form a non-convex solution space. Because the problem is ill-posed, models cannot be learned from such data using common regression methods. While learning of task-space control mappings is *globally* ill-posed, it has been shown in recent work that it is *locally* a well-defined problem. In this paper, we use this insight to formulate a local, kernel-based learning approach for online model learning for task-space tracking control. We propose a parametrization for the local model which makes an application in task-space tracking control of redundant robots possible. The model parametrization further allows us to apply the kernel-trick and, therefore, enables a formulation within the kernel learning framework. For evaluations, we show the ability of the method for online model learning for task-space tracking control of redundant robots.**

***Index Terms*—Kernel Methods, Online Learning, Real-Time Learning, Task-Space Tracking, Robot Control**

I. Introduction

Control of redundant robots in operational space, especially task-space tracking control, is an essential ability needed in robotics [1], [2]. Here, the robot's end-effector follows a desired trajectory in task-space, while distributing the resulting forces onto the robot's joints. Analytical formulation of task-space control laws requires given kinematics and dynamics models of the robot. However, modeling the kinematics and dynamics is susceptive to errors. For example, accurate analytical dynamics models are hard to obtain for complex robot systems, due to many unknown nonlinearities resulting from friction or actuator dynamics [3]. One promising possibility to overcome such inaccurate hand-crafted models is to learn them from data. From a machine learning point of view, learning of such models can be understood as a regression problem. Given input and output data, the task is to learn a model describing the input to output mapping.

Using standard regression techniques, such as Gaussian process regression [4], support vector regression [5] or locally weighted regression [6], a model can be approximated to describe a *single-valued* mapping (i.e., one-to-one) between the input and output data. The single-valued property requires that the same input point should always yield the same single output value, resulting in a well-defined learning problem. However, this situation changes when learning a torque prediction model for task-space tracking control of redundant robots. Here, we are confronted with the problem of learning *multi-valued* or one-to-many mappings. In this case, standard regression techniques can not be applied. Naively learning such multi-valued mappings from sampled data using standard regression will average over multiple output values in a potentially non-convex solution space [7]. Thus, the resulting model will output degenerate predictions, which lead to poor control performance and may cause damage to the redundant robot system.

However, despite being a globally ill-posed problem, learning such task-space control mappings is locally well-defined [7], [8]. Learning a mapping from data can be understood as an averaging over input regions [8]. Employing local learning, the data averaging is performed over local regions. Predictions can subsequently be done by combining learned local models. When the data averaging is performed over small local regions, the approximation is locally consistent. If the local learning is performed online and, thus, time and space variant, different output solutions (depending on time and space) for the same inputs can be approximated using different local models. In this paper, we employ this insight to formulate an online local learning approach, appropriate for learning models that allow prediction with such multi-valued mappings. The key idea is to localize a model in configuration space, while continuously updating this model online by including new data points and, eventually, removing old points. Here, local data points are inserted or removed based on a kernel distance measure. By doing so, we have a consistent spatial and temporal local data set. Due to the local consistency, a prediction model can be learned. The resulting proposed model hence differs from the classical regression setup. Due to the local configuration pre-filtering is not a general regression approach but involves a filtering or prediction step. The proposed model parametrization allows us to apply the kernel-trick and, therefore, enables a formulation within the kernel learning framework [5]. Kernel methods have been shown to be a flexible and powerful tool for learning general nonlinear models [9]–[13]. In task-space tracking control of redundant robots, the model parametrization enables a projection of the joint-space stabilization torques into the task's null-space.

The remainder of the paper will be organized as follows: first, we give a brief overview of task-space control and provide a review of related work. In Section II, we describe our approach to learn task-space tracking control. The proposed method will be evaluated on redundant robot systems, e.g., a

D. Nguyen-Tuong and J. Peters are with the Department Empirical Inference of the Max Planck Institute for Biological Cybernetics. Spemannstraße 38, 72076 Tübingen, Germany. Email: {duy.nguyen-tuong,jan.peters}@tuebingen.mpg.de.

simulated 3-DoF robot and 7-DoF Barrett WAM, for task-space tracking control in Section III. The most important lessons from this research project will be summarized in Section IV.

A. Problem Statement

To obtain an analytical task-space control law, we first need to model the robot's kinematics [1]. The relationship between the task-space and the joint-space of the robot is usually given by the forward kinematics model $\boldsymbol{x} = f(\boldsymbol{q})$. Here, $\boldsymbol{q} \in \mathbb{R}^m$ denotes the robot's configuration in the joint-space and $\boldsymbol{x} \in \mathbb{R}^d$ represents the task-space position and orientation. For redundant robot systems, it is necessary that $m > d$. The task-space velocity and acceleration are

$$\dot{\boldsymbol{x}} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \quad \text{and} \quad \ddot{\boldsymbol{x}} = \dot{\mathbf{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} + \mathbf{J}(\boldsymbol{q})\ddot{\boldsymbol{q}}\,,$$

where $\mathbf{J}(\boldsymbol{q}) = \partial f / \partial \boldsymbol{q}$ is the Jacobian. For computing the joint torques necessary for the robot to follow the task-space trajectory, a model of the robot dynamics is required. A typical dynamics model can be given in the form of

$$\boldsymbol{u} = \mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,,$$

see [14] for more details. Here, $\boldsymbol{u}$ denotes the joint torque, $\mathbf{M}(\mathbf{q})$ is the generalized inertia matrix of the robot, and $\mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a vector containing forces, such as gravity, centripetal and Coriolis forces. Combining the dynamics model with the kinematics model yields one possible operational space control law

$$\boldsymbol{u} = \mathbf{M}\mathbf{J}_w^{\dagger}(\ddot{\boldsymbol{x}}_{\text{ref}} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \mathbf{F}\,, \qquad (1)$$

where $\mathbf{J}_w^{\dagger}$ denotes the weighted pseudo-inverse of $\mathbf{J}$, as described in [3], [15], [16]. In Equation (1), a task-space attractor $\ddot{\boldsymbol{x}}_{\text{ref}}$ is employed for tracking the actual task-space acceleration $\ddot{\boldsymbol{x}}$ [3]. Here, the task-space attractor is formulated as $\ddot{\boldsymbol{x}}_{\text{ref}} = \ddot{\boldsymbol{x}}_{\text{des}} + \mathbf{G}_{vv}(\dot{\boldsymbol{x}}_{\text{des}} - \dot{\boldsymbol{x}}) + \mathbf{G}_{pp}(\boldsymbol{x}_{\text{des}} - \boldsymbol{x})$, where $\boldsymbol{x}_{\text{des}}$, $\dot{\boldsymbol{x}}_{\text{des}}$ and $\ddot{\boldsymbol{x}}_{\text{des}}$ denote the desired task-space trajectory. $\mathbf{G}_{vv}$ and $\mathbf{G}_{pp}$ are positive task-space gain matrices.

To ensure stable tracking in the joint-space of redundant robots, the controller command $\boldsymbol{u}$ in Equation (1) is usually extended by a null-space controller term $\boldsymbol{u}_0$. Thus, the total joint torque command $\boldsymbol{u}_{\text{joint}}$ is given as

$$\boldsymbol{u}_{\text{joint}} = \boldsymbol{u} + \left(\mathbf{I} - \mathbf{J}_w^{\dagger}\mathbf{J}\right)\boldsymbol{u}_0\,. \qquad (2)$$

The term $\boldsymbol{u}_0$ can be interpreted as joint-space stabilizing torques which are only effective in the task's null-space and, thus, do not interfere with the task achievement [3]. The null-space controller command $\boldsymbol{u}_0$ can be chosen such that the redundant robot is pulled towards a desired rest posture $\boldsymbol{q}_{\text{rest}}$, i.e., $\boldsymbol{u}_0 = -\mathbf{G}_v\dot{\boldsymbol{q}} - \mathbf{G}_p(\boldsymbol{q} - \boldsymbol{q}_{\text{rest}})$, where $\mathbf{G}_p$ and $\mathbf{G}_v$ are positive joint-space gain matrices.

As indicated by Equations (1, 2), an analytical formulation for task-space control requires given analytical kinematics and dynamics models. As modeling these relationships can be inaccurate in practice [3], [17], model learning presents a promising alternative. In the task-space tracking problem shown in Equation (1), we want to learn mappings from inputs $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}})$ to targets $\boldsymbol{u}$. However, this mapping is one-to-many [7], [8], as there can be many torques $\boldsymbol{u}$ which correspond to the same task-space acceleration $\ddot{\boldsymbol{x}}$ given $\boldsymbol{q}, \dot{\boldsymbol{q}}$. Thus, naively learning a task-space control model for redundant robots from sampled data may result in a degenerate mapping. In practice, such degenerate models will provide inconsistent torque predictions.

B. Related Work

Learning multi-valued mappings has previously been investigated in the field of neural motor control [18], [19]. In [18], the multi-valued relationship is resolved for a particular output solution by jointly approximating the forward and inverse mapping. Here, it is assumed that the inverse mapping is multi-valued while the corresponding forward mapping is unique. The basic idea is to use the unique forward mapping to resolve the ambiguities in the inverse mapping. Originally, the introduced forward-inverse learning principle has been formulated in the framework of neural networks [18]. In a neural networks based implementation, the forward model is chained with the multi-valued inverse model, where the prediction errors made by the forward model are used to adapt the weight values of the inverse model for a given output solution. However, training such neural networks is well-known to be problematic due to local minima, instability and difficulties in selecting the network structures. Nevertheless, this framework of learning forward and inverse models initiated a number of follow-up research projects, such as [19], [20]. For example, in [20] considerable evidence was presented indicating that the forward-inverse models approach may explain human motor control. In [19], the authors approximate pairwise forward-inverse models for different motor control tasks and, subsequently, combine them for prediction.

In the broader sense, the pairwise forward-inverse model approach [19] can be understood as a *local* learning method, where the data is first partitioned into local regions for which local forward-inverse model pairs are subsequently approximated. Here, different solutions in the output space will be approximated with different local forward-inverse models. These local models can be combined to make consistent predictions for query points. Due to the local consistency, these models can be learned straightforwardly [19], [21]. A local learning approach is also employed in [22] and [8] to learn models for robot inverse kinematics, where locally weighted regression techniques are used. The locally weighted regression approach has been further extended for learning operational space robot control [7]. While Peters et al. [7] attempt to learn a direct mapping for predicting the joint torques for control, Salaun et al. [23] first learn a forward kinematics model and invert the learned model afterwards. Subsequently, they combine it with an inverse dynamics model to generate the required torque command.

Compared to previous local learning approaches, we attempt to learn a *single* localized model, while continuously updating this local model depending on the robot's current configuration. One advantage of our approach is that it is not necessary to partition the data space beforehand, as done in previous local approaches [7], [8]. For high dimensional data, partitioning of the data space is well-known to be a difficult issue [17], [24]. Due to the local consistency, our model learning problem is well-defined. We propose a model parameterization which enables kernel-based learning of torque prediction models for task-space tracking control. The model parametrization also allows a null-space projection, which is necessary to stabilize the robot in the joint-space without interfering with the task-space performance.

II. Learning Task-Space Tracking with Kernels

In this paper, we want to learn the mapping from inputs $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}})$ to outputs $\boldsymbol{u}$, similar to the one described by Equation (1). This mapping is subsequently used for predicting the outputs for given query points. As such one-to-many mappings are locally well-defined [7], [8], they can be approximated with a local kernel learning approach. Here, our model will be localized in the robot's joint position space. The local data is incrementally updated, as the robot moves to new state space regions. Every local data point is additionally weighted by its distance to the most recent joint position. Thereby, we ensure that the local data points form a well-defined set that is appropriate for model learning. Using the weighted local data points, the model's parameters can be obtained by minimizing a cost function. To place the model into the kernel learning framework, we propose a model parametrization appropriate for the application of the kernel-trick. The parametrization is also suitable for task-space tracking control of redundant robots.

The approach we employed in this work can be understood as smoothing over a local region. When employing the approach to locally learn a dynamic function whose output solutions can change with time and space, different output solutions can be temporarily approximated. In the following sections, we will describe how the model is localized and updated in an online setting. We present the parametrization of the local model and show how the corresponding parameters can be obtained from data. Subsequently, we show how the learned local model can be used in online learning for task-space robot control.

A. Model Localization

For learning task-space tracking, we use a *single* local model for torque prediction, where this model is localized in the robot's joint position space. This local data set needs to be continuously updated in the online setting, as the robot frequently moves to new state space regions. In this section, we describe the measures needed to localize and update the model during online learning. The procedure includes insertion of new data points into the local data set and removal of old ones. By continuously online update, we have a consistent spatial and temporal local data set appropriate for model learning. Here, the spatial allocation is taken into account by using kernel-based measures, while the temporal property is achieved by continuously insertion and removal of data points.

1) Insertion of New Data Points: For deciding whether to insert a new data point into the local data set, we consider the distance measure δ as proposed in [25] and [26]. A brief motivation for this measure is given in the Appendix. This measure is defined by

$$\delta(\boldsymbol{q}^*) = k(\boldsymbol{q}^*, \boldsymbol{q}^*) - \boldsymbol{k}^T \boldsymbol{K}_a^{-1} \boldsymbol{k}\,. \tag{3}$$

where $k(\cdot,\cdot)$ denotes a kernel, $\boldsymbol{K}_a = k(\boldsymbol{L}, \boldsymbol{L})$ is the kernel matrix evaluated for the local joint positions $\boldsymbol{L} = \{\boldsymbol{q}_i\}_{i=1}^N$ and $\boldsymbol{k} = k(\boldsymbol{L}, \boldsymbol{q}^*)$ is the kernel vector [26]. The value δ describes the distance of a point $\boldsymbol{q}^*$ to the surface defined by $\boldsymbol{L}$ in the joint-space. This value increases with the distance of $\boldsymbol{q}^*$ from the surface $\boldsymbol{L}$ [26]. It should be noted that the measure given in Equation (3) has been used in various contexts, such as sparsification [27], [28] and learning dynamical systems [26]. In this paper, we employed this measure as a criterion for localization. While the sparsification and pruning deal with the problem of data reduction, the localization is related to the problem of selecting neighboring data points.

Using Equation (3), we can make decisions for inserting new data points. If the δ values of new data points exceed a given threshold η, we will insert these points into the local model. The employed measure δ ensures that new data points will be included into the local set, when the robot moves to new joint-space regions.

2) Removal of Old Data Points: For removing data points from the local set, we select the point which is the *farthest* from the most recent joint position $\boldsymbol{q}$. Here, we employ a Gaussian kernel as a distance measure between $\boldsymbol{q}$ and other local data points $\boldsymbol{q}_i$

$$k\left(\boldsymbol{q}, \boldsymbol{q}_i\right) = \exp\left(-\frac{1}{2}(\boldsymbol{q}-\boldsymbol{q}_i)^T \mathbf{W} (\boldsymbol{q}-\boldsymbol{q}_i)\right), \tag{4}$$

where $\mathbf{W}$ denotes the kernel width. Removing the farthest local data point implies that its kernel measure $k(\cdot,\cdot)$ is the smallest. By continuously inserting and removing local data points, we make sure that the local data set is suitable for the current region of the state space. It is worth noting that it is sufficient to localize the local set in the robot's joint position space. Due to the smoothness of the function being learned, spatially and temporarily nearby input data points will have similar output values. Hence, the localization will form a consistent spatial and temporal local set.

B. Model Parametrization

The described insertion and removal operations in preceding section result in a data set localized in the joint position space. Due to the local consistency, model learning using this data is well-defined. Given the sampled local data set $\boldsymbol{D} = \{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{x}}_i, \boldsymbol{u}_i\}_{i=1}^N$, we can now learn a model for torque prediction for task-space control.

From the physical model described in Equation (1), we can see that the joint torque $\boldsymbol{u}$ is *linear* in the task-space acceleration $\ddot{\boldsymbol{x}}$, while it is *nonlinear* in the joint position $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$. Using this insight, we propose the following parametrization for the local model

$$\boldsymbol{u} = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\theta}_0^T \ddot{\boldsymbol{x}}\,, \tag{5}$$

where $\boldsymbol{\phi}$ is a vector containing nonlinear functions projecting $[\boldsymbol{q}, \dot{\boldsymbol{q}}]$ into some high-dimensional spaces. Generally, $\ddot{\boldsymbol{x}}$ can have d dimensions and $\boldsymbol{u}$ is a m dimensional vector. Following the representer theorem [5], the coefficients $\boldsymbol{\theta}, \boldsymbol{\theta}_0$ in Equation (5) can be expanded in term of N local data points. Hence, we have

$$\boldsymbol{\theta} = \textstyle\sum_{i=1}^{N} \alpha_i \boldsymbol{\phi}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)\,, \quad \boldsymbol{\theta}_0 = \sum_{i=1}^{N} \alpha_0^i \ddot{\boldsymbol{x}}_i\,,$$

where α_i, α_0^i are the corresponding linear expansion coefficients. Inserting the linear expansions into Equation (5) and re-writing it in term of N sample data points yields

$$\boldsymbol{U} = \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{P}\boldsymbol{\alpha}_0\,. \tag{6}$$

Here, the elements $[\boldsymbol{K}]_{ij} = \langle \boldsymbol{\phi}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i), \boldsymbol{\phi}(\boldsymbol{q}_j, \dot{\boldsymbol{q}}_j)\rangle$ are the pairwise inner-products of the feature vectors. Thus, $[\boldsymbol{K}]_{ij}$ can be represented with kernels [5], i.e., $[\boldsymbol{K}]_{ij} = \tilde{k}([\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i], [\boldsymbol{q}_j, \dot{\boldsymbol{q}}_j])$. The matrix $\boldsymbol{K}$ is thus a kernel matrix evaluated at the joint position and velocity employing the kernel $\tilde{k}(\cdot,\cdot)$. Using this so-called kernel-trick, only the kernel function $\tilde{k}$ needs to be determined instead of an explicit feature mapping $\boldsymbol{\phi}$ [5]. Similarly, the elements $[\boldsymbol{P}]_{ij} = \langle \ddot{\boldsymbol{x}}_i, \ddot{\boldsymbol{x}}_j\rangle$ represent the pairwise inner-products of the task-space acceleration $\ddot{\boldsymbol{x}}$. Thus, $\boldsymbol{P}$ can be understood as a kernel matrix where linear kernels are applied. In Equation (6), the matrix $\boldsymbol{U}$ is given by $\boldsymbol{U} = \{\boldsymbol{u}_i\}_{i=1}^{N}$.

C. *Online Learning of Local Model*

Learning requires the estimation of the expansion parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ in Equation (6) from the local data set. Employing the learned model, we can predict the output for a query point. In particular, for online learning the expansion parameters have to be estimated incrementally, as the data arrives as a stream over time.

1) Estimation of Model Parameters: Using the model parametrization in Section II-B, the expansion parameters can be estimated from data by minimizing an appropriate cost function $\mathcal{L}$ given by

$$\begin{aligned}\mathcal{L} = {} & \frac{\gamma}{2}\left(\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\alpha}_0^T\boldsymbol{P}\boldsymbol{\alpha}_0\right) \\ & + \frac{1}{2}\left(\boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{P}\boldsymbol{\alpha}_0 - \boldsymbol{U}\right)^T \mathbf{N} \left(\boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{P}\boldsymbol{\alpha}_0 - \boldsymbol{U}\right).\end{aligned} \tag{7}$$

The first term in Equation (7) acts as regularization, while the second term represents a squared-loss based data-fit. In Equation (7), the parameter γ controls the regularization and the diagonal matrix $\mathbf{N}$ denotes the *weight* for each data point in the local set. The minimization of $\mathcal{L}$ w.r.t. $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ yields the analytical solution

$$\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} + \gamma\mathbf{N}^{-1} & \boldsymbol{P} \\ \boldsymbol{K} & \boldsymbol{P} + \gamma\mathbf{N}^{-1} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{U} \end{bmatrix}. \tag{8}$$

A derivation for Equation (8) is given in the Appendix. The weighting metric $\mathbf{N}$ incorporates a distance measure of each local data point to the most recent point in the local set. Here, we employ a kernel distance measure in the joint position space, as given in Equation (4). The weighting metric $\mathbf{N}$ ensures that the local data will form a well-defined set appropriate for the model learning step.

Algorithm 1 Online learning of the local model.

```
Given: local data set $\boldsymbol{D} = \{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{x}}_i, \boldsymbol{u}_i\}_{i=1}^{N}$, $N_{\max}$, threshold value $\eta$.
Input: new input $\{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}\}$ and output $\boldsymbol{u}$.

Evaluate the distance of $\boldsymbol{q}$ to the surface defined by $\boldsymbol{L} = \{\boldsymbol{q}_i\}_{i=1}^{N}$ based on the measure $\delta(\boldsymbol{q})$ from Equation (3).
if $\delta(\boldsymbol{q}) > \eta$ then
    for $i = 1$ to $N$ do
        Compute: $\mathbf{N}(i,i) = k(\boldsymbol{q}, \boldsymbol{q}_i)$ using Equation (4).
    end for
    if $N < N_{\max}$ then
        Include the new point: $\boldsymbol{D}_{N+1} = \{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}, \boldsymbol{u}\}$.
    else
        Find the farthest point: $j = \min_i \mathbf{N}(i,i)$.
        Replace the $j$-th local data point by the query point: $\boldsymbol{D}_j = \{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}, \boldsymbol{u}\}$.
    end if
    Update the expansion parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ incrementally using Equation (8), while re-weighting every local data point with the new distance metric $\mathbf{N}$.
end if
```

2) Online Model Learning: As the data arrives continuously in the online setting, Equation (8) has to be updated incrementally. Such incremental updates require adjusting the corresponding row and column of the inverse matrix, i.e., a rank-one update of the inverse matrix [29]. In this paper, we employ the well-known Sherman-Morrison formular to update the corresponding rows and columns [30]. Additionally, every data point in the local set has to be re-weighted by its distance to the most current point after every insertion and removal step. In practice, we initialize the inverse matrix in Equation (8) as a diagonal matrix, where the number $N_{\max}$ of local data points is fixed. During online learning, the inverse matrix is first updated $N_{\max}$ times while filling up the local data set. Subsequently, old data points have to be removed when new points are inserted. Removing an old point is performed by overwriting the corresponding row and column by new ones. The complete procedure for learning the local model is summarized in the Algorithm 1.

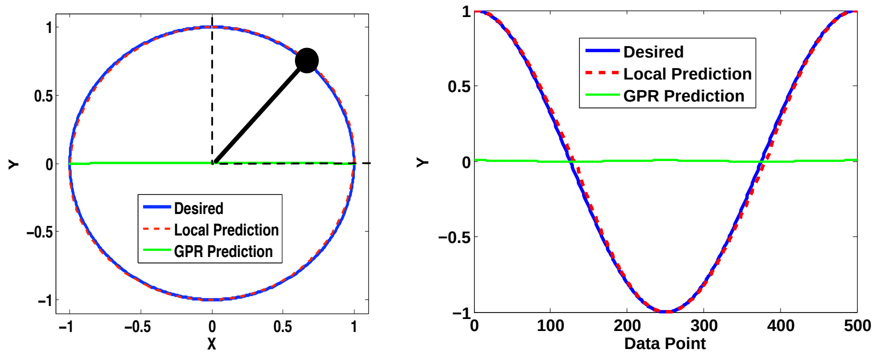
3) Prediction: With the optimization results from Equation (8), the prediction $\hat{\boldsymbol{u}}$ for a query point $[\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_{\mathrm{ref}}]$ can be computed as

$$\hat{\boldsymbol{u}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_{\mathrm{ref}}) = \boldsymbol{\alpha}^T\tilde{k}(\boldsymbol{Q}, [\boldsymbol{q}, \dot{\boldsymbol{q}}]) + \boldsymbol{\alpha}_0^T\langle \ddot{\boldsymbol{X}}, \ddot{\boldsymbol{x}}_{\mathrm{ref}}\rangle\,, \tag{9}$$

where $\ddot{\boldsymbol{X}} = \{\ddot{\boldsymbol{x}}_i\}_{i=1}^{N}$ denotes the set of sampled task-space acceleration and $\boldsymbol{Q} = \{\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i\}_{i=1}^{N}$ contains robot's joint positions and velocities.

D. *Using Local Model for Task-Space Control*

Up to now, we have learned a well-defined local model to predict the joint torques required to drive the robot along a desired task-space trajectory. To ensure the local consistency, this local model is continuously updated depending on the current robot's configuration. However, even after obtaining



(a) **Trajectory of a pendulum in the x–y space**

(b) **Prediction of the outcome y**

Fig. 1: An example of learning a non-unique function. For the pendulum shown in (a), we have for each input position x two possible output values. Naively learning a global mapping $\boldsymbol{x} \to \boldsymbol{y}$ using GPR [4] results in an average over multiple output solutions, as shown in (b). However, when the mapping is learned locally within the vicinity of the query point in an online setting, the model learning problem is well-defined resulting in a proper prediction.

Algorithm 2 Online prediction for task-space control.

```
**Given:** a rest posture $q_{\text{rest}}$, local data $\ddot{X} = \{\ddot{x}_i\}_{i=1}^N$ and $Q = \{q_i, \dot{q}_i\}_{i=1}^N$, expansion parameters $\alpha$ and $\alpha_0$.
**Input:** query point $\{q, \dot{q}, x_{\text{des}}, \dot{x}_{\text{des}}, \ddot{x}_{\text{des}}\}$.

Compute null-space control torque $u_0$.
Compute null-space projection matrix $H = \alpha_0^T \ddot{X}$.
Compute task-space attractor $\ddot{x}_{\text{ref}}$.
Compute joint torque control $u_{\text{joint}}$ as given in Equation (10).
```

a perfect prediction of the necessary torques, it is not clear whether the robot will be stable in the joint-space. Thus, we need to explore ways to stabilize the robot in the joint-space without interfering the task-space performance, as done in analytical task-space control (see Equation (2)). Here, the key idea is to *project* the stabilizing torques $\boldsymbol{u}_0$ into the null-space of the "task relevant" part.

From Equation (9) for the prediction, it can be seen that the second term is the task relevant part, as this term explicitly depends on $\ddot{\boldsymbol{x}}_{\text{ref}}$. Therefore, for the robot joint-space stabilization, we can project the stabilization torques $\boldsymbol{u}_0$ into the null-space of this term. Hence, the total joint torque controller command $\boldsymbol{u}_{\text{joint}}$ can be computed as

$$\boldsymbol{u}_{\text{joint}} = \boldsymbol{\alpha}^T \tilde{k}(\boldsymbol{Q}, [\boldsymbol{q}, \dot{\boldsymbol{q}}]) + \boldsymbol{\alpha}_0^T \langle \ddot{\boldsymbol{X}}, \ddot{\boldsymbol{x}}_{\text{ref}} \rangle + (\mathbf{I} - \boldsymbol{H}(\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T)\boldsymbol{u}_0 \,. \tag{10}$$

The null-space projection is then given by the matrix $\boldsymbol{H} = \boldsymbol{\alpha}_0^T \ddot{\boldsymbol{X}}$. The resulting null-space projection allows joint-space stabilization based on $\boldsymbol{u}_0$ without interfering the task performance. The procedure for online torque prediction in task-space tracking control is summarized in the Algorithm 2.

III. Robot Evaluations

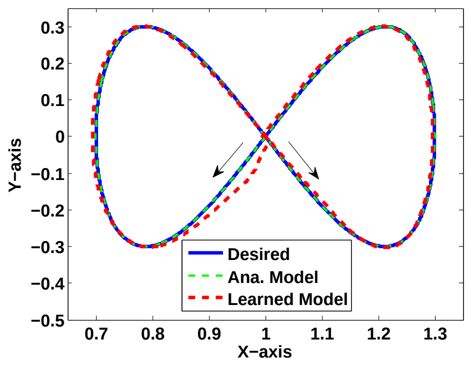
In this section, we evaluate the proposed approach for learning task-space control, as described in Section II. First, we show for a toy example how a non-unique function can be learned in the online setting using this local learning approach. This example further illustrates the basic idea behind the local learning principle when used for approximating a multi-valued mapping. Subsequently, we show the ability of our method in learning torque prediction models for task-space tracking control of redundant robot systems. The control experiments are performed with both simulated 3-DoF robot and 7-DoF anthropomorphic Barrett arm as shown in Figure 2 (c).

A. Online Learning of a Non-unique Function

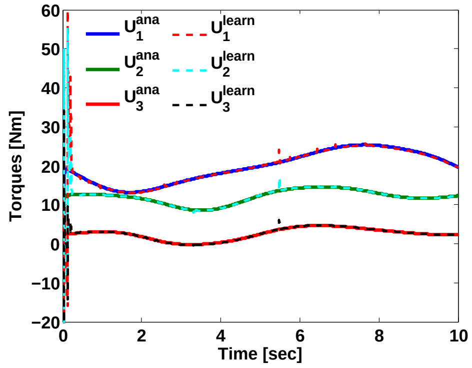
As benchmark example, we create a one-dimensional non-unique function shown in Figure 1 (a). In this example, there is a pendulum that can rotate in the $\boldsymbol{x}$–$\boldsymbol{y}$ plane. For a circular rotation, the trajectory of $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by $x_i = \sin(t_i)$ and $y_i = \cos(t_i)$ for t_i ranging from 0 to 2π. For the experiment, we sample 500 data points from the generated trajectory. If we employ $\boldsymbol{x}$ as input and $\boldsymbol{y}$ as the target output, we will have a non-unique prediction problem.

In this example, the parametrization of the local model is given by $\boldsymbol{y} = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x})$. While the model is localized in the $\boldsymbol{x}$ space, we incrementally update the local data set and learn the model in the online setting, as described in Section II. For online model learning, we incrementally feed the data to the algorithm. Figure 1 shows the results after one sweep through the data set. To highlight the difficulty in learning such multi-valued mappings from data, the well-known Gaussian process regression (GPR) [4] is employed to globally approximate the mapping $\boldsymbol{x} \to \boldsymbol{y}$. The comparison between the two methods is given in Figure 1. Figure 1 (a) shows the trajectory of the pendulum in the $\boldsymbol{x} - \boldsymbol{y}$ space when using local learning and GPR for the prediction of $\boldsymbol{y}$ given $\boldsymbol{x}$. Figure 1 (b) shows the corresponding prediction outputs $\boldsymbol{y}$.

In the experiment, the size of the local data set $N_{\max}$ is chosen to be 10. The choice of the size also depends on the complexity of the function being approximated. As the computing cost is $\mathcal{O}(4N^2)$ due to the incremental update of the inverse matrix in Equation (8), the number of the



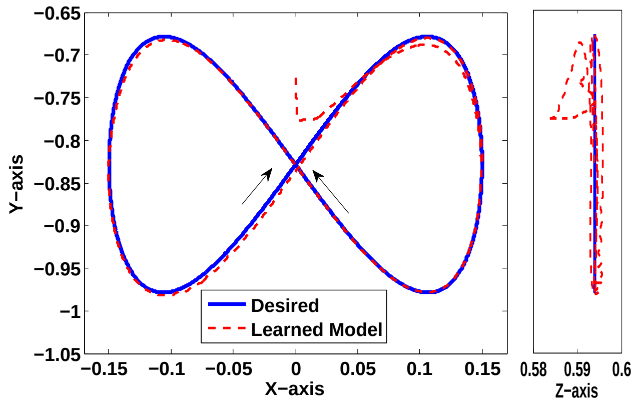
(a) **Task-space tracking**



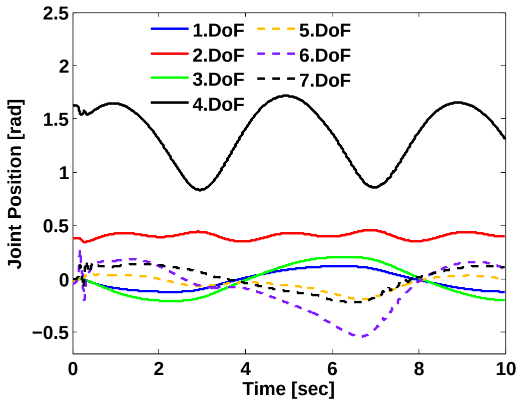
(b) **Control torque for each DoF**

(c) **Anthropomorphic Barrett WAM**

Fig. 2: Task-space tracking by a 3-DoF robot. Here, we compared task-space tracking using perfect analytical model with one that was learned online. As a perfect model is used, the analytical task-space tracking control yields a perfect tracking, as shown in (a). During online learning, the learned task-space controller continuously improves the task-space tracking performance. As shown in (b), the learned task-space control torques $\boldsymbol{u}_{\text{joint}}$ converge to the perfect analytical torques after a short transient phase. (c) 7-DoF Barrett whole arm manipulator used for task-space tracking control.

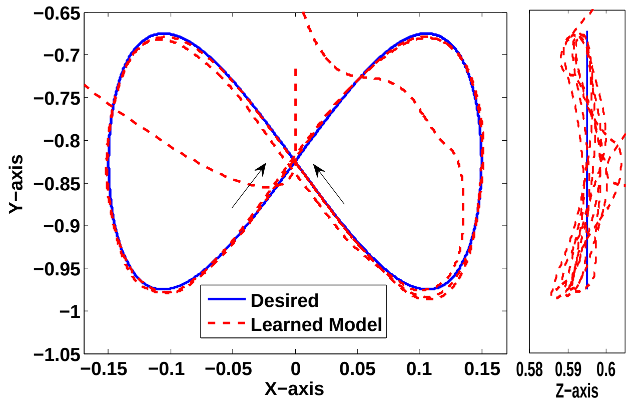


(a) **Task-space tracking performance**

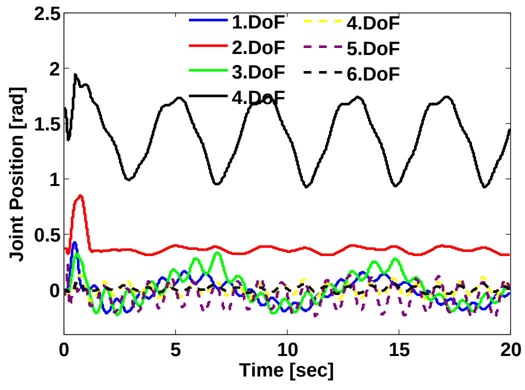


(b) **Joint trajectory**

Fig. 3: Task-space tracking control of a simulated 7-DoF Barrett WAM with online learned model. (a) During online model learning, the task-space controller is able to compute the required torques to follow the task-space trajectory. (b) Joint-space trajectory during the online learning. It can be seen that the joint trajectory is kept in the neighborhood of the rest posture. Here, the rest posture is given by $\boldsymbol{q}_{\text{rest}} = [0.0, 0.5, 0.0, 1.9, 0.0, 0.0, 0.0]^T$.



(a) **Task-space tracking with null-space perturbation**



(b) **Joint trajectory with null-space perturbation**

Fig. 4: Task-space tracking control of a simulated 7-DoF Barrett WAM with additional null-space perturbation. (a) The null-space perturbation leads to an initial deviation from the desired trajectory. Despite the continuous perturbation in the null-space, the online learned controller is able to track the task-space trajectory sufficiently well. Here, the learned controller is able to stabilize the robot in the joint-space. (b) Joint-space trajectory during the online learning with additional perturbation. The perturbation creates oscillations in the joint-space movements.

local data points N also implies a trade-off between learning performance and computational complexity. During the online learning and prediction, we first incrementally fill the local set up and, subsequently, update the local model online by insertion and removal. We employ the Gaussian kernel for the localization step, as well as for model learning. The kernel width $\mathbf{W}$ is optimized by cross-validation, and the threshold η is set to be 0.001.

In practice, the two parameters η and $N_{\max}$ are open-parameters which need to be chosen by the user. The parameter $N_{\max}$ determines the size of the local region, and η determines the frequency of the updates. The smaller the parameter η is, the more frequently the local model is updated. While small η-values enable an accurate approximation of the dynamics, they also lead to higher computational complexity. Thus, η should be chosen according to the available computational power. Large $N_{\max}$ might lead to inconsistency in the data, on the other hand small $N_{\max}$-values can deteriorate the learning performance. While there is a connection between the variability of the function being approximated, the sampling rate and the choice of $N_{\max}$, it is difficult to give general rules for the choice of this parameter. However, a rule of thumb might be useful: the more variable the data is, the smaller should be $N_{\max}$; and the denser the data is sampled, the larger $N_{\max}$ can be chosen. In practice, $N_{\max}$ and η can be chosen beforehand by cross-validation on sampled data. By so doing, optimal settings for a given function being learned can be determined.

As the farthest point in the input space is removed when a new point is inserted, one can observe that the local data set always covers a region in the vicinity of the recent query point. Due to the locality of the data set, i.e., only a local region of the input-output space is covered with data at any time, the local learning approach does not suffer from the ambiguity in the outputs. Thus, we have a local model which moves along with the data stream observed online. The locality is further enforced by the weighting metric $\boldsymbol{N}$ introduced in Equation (7), which ensures that the local set forms a convex data space. Thus, a local model can be learned and results in a proper online prediction of the targets $\boldsymbol{y}$, shown in Figure 1.

B. Online Model Learning for Task-Space Tracking Control

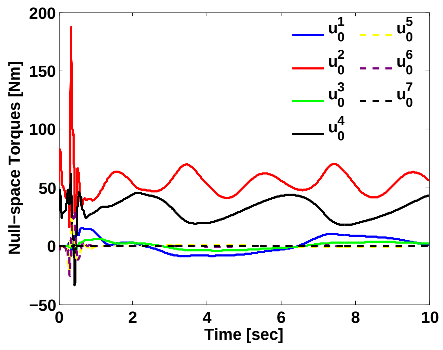
In this section, we apply the proposed method to online learning of torque prediction models for task-space control of a simulated 3-DoF robot and the simulated 7-DoF Barrett WAM. In the experiments, the models are learned online, while the robots are controlled to track a task-space trajectory. Here, the task-space trajectory is given by the positions of the end-effector in Cartesian space. Thus, the task-space dimension is 2 for the 3-DoF robot and 3 for the 7-DoF Barrett arm. The tracking results in task-space for the 3-DoF robot and the Barrett WAM are shown in Figures 2 and 3, respectively. The figures show the tracking performance during the first 10 seconds.

In the experiment using the 3-DoF robot model shown in Figure 2, we compare the task-space tracking control performance, when employing the online learned model and the perfect analytical model. Using the *perfect* analytical model knowledge, the joint torques are computed as given in Equation (2). Thus, the robot performs perfect task-space tracking, as shown in Figure 2 (a). In this example, the rest posture is set to be $\boldsymbol{q}_{\mathrm{rest}} = [-\pi/3, \pi/3, \pi/3]^T$.

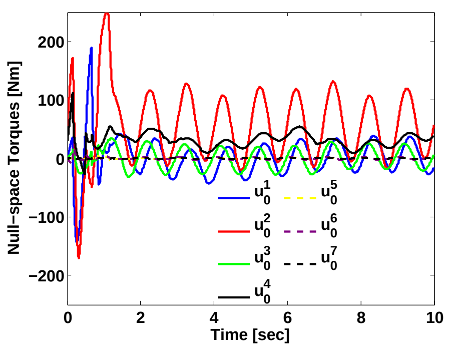
For the online learning of the task-space control model, the torque prediction is computed as given in Equation (10). The size of the local set is determined to be 30 and $\eta = 0.01$. Here, we employ a Gaussian kernel, where the kernel width $\mathbf{W}$ is optimized beforehand. As shown in Figure 2 (b), the learned joint torques converge to the torques computed by the perfect analytical model after a short transient phase. As a result, the robot achieves good task-space tracking performance after a few seconds of online model learning. This comparison shows that the online learned local model is able to approximate the physical properties of the robot. In the next experiment, we employ the proposed approach to control the more complex 7-DoF Barrett WAM in simulation. Similar to the previous experiment, the robot is controlled to follow a figure-8 in task-space while learning the torque prediction model online. Here, the local set consists of 150 data points, $\eta = 0.05$ and a Gaussian kernel is used. During online learning, the model is incrementally updated 300 times. The results for the first 10 seconds are shown in Figure 3. It can be observed that the robot is able to follow the task-space trajectory well, while keeping the joint-space trajectory in the vicinity of the rest posture $\boldsymbol{q}_{\mathrm{rest}} = [0.0, 0.5, 0.0, 1.9, 0.0, 0.0, 0.0]^T$.

In task-space tracking control, it is desirable that the joint-space stabilization $\boldsymbol{u}_0 = -\mathbf{G}_v\dot{\boldsymbol{q}} - \mathbf{G}_p(\boldsymbol{q} - \boldsymbol{q}_{\mathrm{rest}})$ does not significantly affect the task-space performance. To achieve this goal, the joint-space stabilization torque $\boldsymbol{u}_0$ is projected into the null-space of the task relevant part, as shown in Equation (10). In order to show the decoupling of $\boldsymbol{u}_0$ from the task performance, we additionally perturbate the null-space movements. In particular, we add a perturbation to the first 4 DoFs of the Barrett WAM (i.e., the shoulder DoFs and the elbow DoF). For this experiment, we define a time-dependent rest posture as $\boldsymbol{q}_{\mathrm{rest}}(t) = [P(t), 0.5 + P(t), P(t), 1.9 + P(t), 0.0, 0.0, 0.0]^T$, with $P(t) = 0.5\sin(2\pi t)$ and t is the running time. Figure 4 shows the results for the first 20 sec.

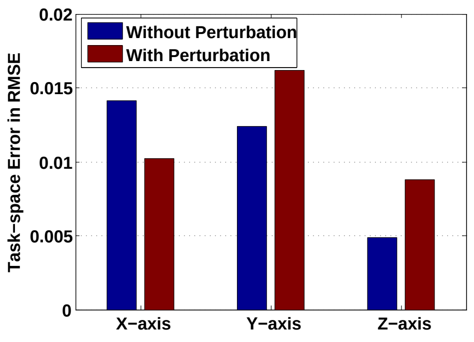
It can be seen that the time-dependent perturbation in the rest posture $\boldsymbol{q}_{\mathrm{rest}}(t)$ leads to an initial deviation from the desired trajectory. During online learning, the robot gradually moves back to the desired trajectory, as the model learning is converging. Although the perturbation is added to the first 4 DoFs, the null-space movements are distributed on all 7 DoFs, as shown by the Figure 4 (b). Due to the perturbation, the stabilization torque $\boldsymbol{u}_0$ starts oscillating as shown in Figure 5 (b). The perturbation of $\boldsymbol{u}_0$ also becomes noticeable through oscillations in the joint trajectory (see Figure 4). The decoupling of the joint-space stabilization $\boldsymbol{u}_0$ from the task performance is showed by the circumstances that the task-space tracking remains sufficiently accurate despite different joint-space stabilization. Figure 5 shows $\boldsymbol{u}_0$ (a) and (b) for cases with and without perturbation. Despite the different stabilization torques, the task-space performance is hardly affected. The task-space tracking errors are similar for both cases, as shown in Figure 5 (c).



(a) **Null-space torque u_0 for each DoF**



(b) **Null-space torque u_0 with perturbation**



(c) **Task-space tracking error for each axis**

Fig. 5: Figures (a) and (b) show the joint-space stabilization torques $\boldsymbol{u}_0$ during the first 10 sec of the task-space tracking. (a) Stabilization torque *without* perturbation. (b) Stabilization torque *with* perturbation. Due to the projection of $\boldsymbol{u}_0$ into the null-space of the task, different joint-space stabilization torques do not significantly affect the task performance. In both cases, the task-space tracking errors remain similar, as shown in Figure (c). The tracking errors are computed as root-mean-square-error (RMSE) after 10 sec online learning.

IV. Conclusion

In this paper, we employed local, kernel-based learning for the online approximation of a multi-valued mapping. This approach is based on the key insight that an approximation of such mappings from data is globally an ill-posed problem, while it is locally well-defined. Our proposed method uses an online procedure for updating the local model by inserting and removing data points. We further proposed a parametrization for the local model that allows learning task-space tracking control. The update procedures and the model are formulated in the kernel framework, where the resulting parameters can be incrementally learned online. The resulting framework has open parameters based on the smoothness for dynamics mapping. To determine a measures of this smoothness is an important open issue which needs to be addressed in future, follow-up research projects. As evaluation, we showed that the approach was able to learn torque prediction models for task-space tracking of redundant robots in several setups. The experiments are performed both on a simulated 3-DoF robot and the simulated 7-DoF Barrett WAM. The results show that the presented kernel-based approach can be used to approximate multi-valued mappings for task-space tracking control.

References

[1] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *Journal of Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[2] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *International Journal of Humanoid Robotics*, vol. 2, no. 4, pp. 505–518, 2005.

[3] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: a theoretical and emprical comparison," *International Journal of Robotics Research*, vol. 27, no. 6, pp. 737–757, 2008.

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

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

[6] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1–5, pp. 11–73, 1997.

[7] J. Peters and S. Schaal, "Learning to control in operational space," *International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 2008.

[8] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *IEEE International Conference on Intelligent Robots and Systems*, 2001.

[9] S. V. Vaerenbergh, I. Santamara, W. Liu, and J. C. Principe, "Fixed budget kernel recursive least squares," in *International Conference on Acoustics, Speech and Signal Processing*, 2010.

[10] H. Ning, X. Jing, and L. Cheng, "Online identification of nonlinear spatiotemporal systems using kernel learning approach," *Neural Networks, IEEE Transactions on*, no. 22, pp. 1381 – 1394, 2011.

[11] P. Bouboulis, K. Slavakis, and S. Theodoridis, "Adaptive learning in complex reproducing kernel hilbert spaces employing wirtinger's subgradients," *Neural Networks and Learning Systems, IEEE Transactions on*, no. 23, pp. 425 – 438, 2012.

[12] B. Chen, S. Zhao, P. Zhu, and J. Principe, "Quantized kernel least mean square algorithm," *Neural Networks and Learning Systems, IEEE Transactions on*, no. 23, pp. 22 – 32, 2012.

[13] K. Slavakis, P. Bouboulis, and S. Theodoridis, "Adaptive multiregression in reproducing kernel hilbert spaces: The multiaccess mimo channel case," *Neural Networks and Learning Systems, IEEE Transactions on*, no. 23, pp. 60 – 276, 2012.

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

[15] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*. New York: McGraw-Hill, 1996.

[16] J. Peters, M. Mistry, F. E. Udwadia, J. Nakanishi, and S. Schaal, "A unifying methodology for robot control with redundant dofs," *Autonomous Robots*, vol. 24, no. 1, pp. 1–12, 2008.

[17] 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.

[18] I. Jordan and D. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Science*, vol. 16, pp. 307–354, 1992.

[19] D. M. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, pp. 1317–1329, 1998.

[20] N. Bhushan and R. Shadmehr, "Evidence for a forward dynamics model in human adaptive motor control," *Advances in Neural Information Processing Systems*, 1999.

[21] M. Haruno, D. M. Wolpert, and M. Kawato, "Mosaic model for sensorimotor learning and control," *Neural Computation*, vol. 13, no. 10, pp. 2201–2220, 2001.

[22] G. Tevatia and S. Schaal, "Efficient inverse kinematics algorithms for high-dimensional movement systems," *University of Southern California*, 2008.

[23] C. Salaun, V. Padois, and O. Sigaud, "Control of redundant robots using learned models: an operational space control approach," in *Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems*, 2009.

[24] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Computation*, vol. 12, no. 11, pp. 2602 – 2634, 2005.

[25] 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.

[26] D. Nguyen-Tuong and J. Peters, "Incremental sparsification for real-time online model learning," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

[27] W. Liu, I. Park, and J. C. Principe, "An information theoretic approach of designing sparse kernel adaptive filters," *IEEE Transactions on Neural Networks*, no. 20, pp. 1950 – 1961, 2009.

[28] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*. Wiley, 2010.

[29] D. Nguyen-Tuong and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[30] G. H. Golub and C. F. V. Loan, *Matrix Computation*. Baltimore: John Hopkins University Press, 1996.

Appendix

A. Linear Independence Measure

To test whether a new point $\boldsymbol{q}^*$ should be inserted, we need to ensure that it can not be approximated in the feature space spanned by the current local set $\boldsymbol{L}=\{\boldsymbol{q}_i\}_{i=1}^N$. This test can be performed using a measure δ defined as

$$\delta = \left\| \sum_{i=1}^{m} \mathrm{a}_i \boldsymbol{\psi}(\boldsymbol{q}_i) - \boldsymbol{\psi}(\boldsymbol{q}^*) \right\|^2 , \tag{11}$$

(see, e.g., [5], [25], [27] for more background information), where a_i denote the coefficients of linear dependence and $\boldsymbol{\psi}$ is a feature function. Equation (11) can be understood as a distance of the new point $\boldsymbol{q}^*$ to the linear plane spanned by the local set $\boldsymbol{L}$ in the feature space. Thus, the value δ can be considered as a measure indicating how well a new data point $\boldsymbol{q}^*$ can be approximated in the feature space of a given data set. The larger the value of δ is, the more *informative* is $\boldsymbol{q}^*$ for the local set $\boldsymbol{L}$.

The coefficients a_i from Equation (11) can be determined by minimizing δ. Formulated in matrix form, the minimization of δ can be given as

$$\boldsymbol{a} = \min_{\mathbf{a}} \left[\mathbf{a}^T \boldsymbol{K}_a \mathbf{a} - 2\mathbf{a}^T \boldsymbol{k} + b \right] , \tag{12}$$

where $\boldsymbol{K}_a = k(\boldsymbol{L}, \boldsymbol{L})$ represents the kernel matrix, $\boldsymbol{k} = k(\boldsymbol{L}, \boldsymbol{q}^*)$ is the kernel vector and $b = k(\boldsymbol{q}^*, \boldsymbol{q}^*)$. Note that in Equation (12) we make use of the property that inner products of feature vectors can be represented as kernel values [5]. Minimizing Equation (12) yields the optimal coefficient vector $\boldsymbol{a} = \boldsymbol{K}_a^{-1}\boldsymbol{k}$. The parameter $\boldsymbol{a}$ from Equation (12) 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 selection process. After substituting the optimal value $\boldsymbol{a}$ into Equation (11), δ becomes

$$\delta(\boldsymbol{q}^*) = b - \boldsymbol{k}^T \boldsymbol{a} . \tag{13}$$

Using δ we can decide whether to insert new data points into the local set.

B. Estimation of the Expansion Parameters

The derivatives of the cost function $\mathcal{L}$ in Equation (7) w.r.t. $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ and are given by

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} &= \gamma \boldsymbol{\alpha} + \mathbf{N} \left(\boldsymbol{K\alpha} + \boldsymbol{P\alpha}_0 - \boldsymbol{U} \right) &= \mathbf{0} \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}_0} &= \gamma \boldsymbol{\alpha}_0 + \mathbf{N} \left(\boldsymbol{K\alpha} + \boldsymbol{P\alpha}_0 - \boldsymbol{U} \right) &= \mathbf{0} . \end{aligned} \tag{14}$$

Rewriting Equation (14) yields

$$\begin{aligned} \boldsymbol{U} &= \left(\gamma \mathbf{N}^{-1} + \boldsymbol{K} \right) \boldsymbol{\alpha} + \boldsymbol{P\alpha}_0 \\ \boldsymbol{U} &= \left(\gamma \mathbf{N}^{-1} + \boldsymbol{P} \right) \boldsymbol{\alpha}_0 + \boldsymbol{K\alpha} . \end{aligned} \tag{15}$$

Solving Equation (15) for $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ yields

$$\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} + \gamma \mathbf{N}^{-1} & \boldsymbol{P} \\ \boldsymbol{K} & \boldsymbol{P} + \gamma \mathbf{N}^{-1} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{U} \end{bmatrix} . \tag{16}$$

For online learning, the inverse matrix in Equation (16) need to be incrementally updated which requires a cost of $\mathcal{O}(4N^2)$ for computation and memory. As the computation can be expensive for large N, the online update of the inverse matrix can be implemented in an independent process. Here, the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ can be updated independently from the joint torque controller (showed in Equation (10)). While the controller command $\boldsymbol{u}_{joint}$ need to be computed for each sampling step in real-time, e.g., every 2 ms for the Barrett WAM, it is sufficient to update $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_0$ a much slower rate.