# Distributed Nonlinear Regression Using In-Network Processing With Multiple Gaussian Kernels

Ban-Sok Shin*, Henning Paul*, Masahiro Yukawa† and Armin Dekorsy*
*Department of Communications Engineering, University of Bremen, Germany
Email: {shin, paul, dekorsy}@ant.uni-bremen.de
†Dept. of Electronics and Electrical Engineering, Keio University, Japan
Email: yukawa@elec.keio.ac.jp

***Abstract*—In this paper, we propose the use of multiple Gaussian kernels for distributed nonlinear regression or system identification tasks by a network of nodes. By employing multiple kernels in the estimation process we increase the degree of freedom and thus, the ability to reconstruct nonlinear functions. For this, we extend the so-called KDiCE algorithm, which allows a distributed regression of nonlinear functions but uses a single kernel only, to multiple kernels. We corroborate our proposed scheme by numerical evaluations for the reconstruction of nonlinear functions both static and time-varying. We achieve performance gains for both cases, in particular for the tracking of a time-varying nonlinear function.**

***Index Terms*—In-network processing, distributed regression, kernel least-squares, multiple kernels**

## I. Introduction

We study the task of distributed nonlinear regression/system identification with multiple reproducing kernels by a network of nodes. In real-world applications considering a network of sensors or base stations this task is relevant, e.g., to reconstruct physical environments such as the spatial distribution of temperature, gas, sound intensity or the path loss in mobile communications [1]. Due to the nonlinearity present in the underlying physical model, a satisfying reconstruction requires nonlinear estimation techniques. Furthermore, in most of these applications distributed approaches employing in-network processing (INP) techniques are desired such that the network itself is able to monitor and act based on its learned knowledge. In INP applications, no central unit or fusion center is available and the reconstruction is performed within the network. Thus, each node has to rely on information exchange with neighboring nodes to achieve a satisfying reconstruction. In order to combine nonlinear estimation techniques with INP, we utilize kernel methods. These methods provide the possibility to reconstruct nonlinear functions and have been applied to various well-known estimation algorithms [2]. Adaptive filters such as the least-mean-squares (LMS), affine projection algorithm (APA), recursive least squares (RLS) have been extended to nonlinear versions in [3], [4]. One difficulty is the selection of an appropriate kernel function. If the choice of this function does not match the underlying, unknown system, the performance will degrade severely. To address this issue, multiple kernels have been introduced and utilized to extend current kernel adaptive filters by the framework developed in [5], [6]. The usage of multiple kernels lowers the requirement of exact a-priori knowledge about the correct kernel function or a manual tuning of the kernel parameters. It introduces a higher flexibility into the estimation process and improves the reconstruction performance in particular for functions with distinct features as, e.g., high and low frequency components.

The task of distributed nonlinear regression/system identification using kernel methods has been addressed in the past by [7], [8] and investigated for mobile sensor networks in [9]. However, in these approaches only a single kernel function is considered. Our contribution in this paper is the extension of the kernel distributed consensus-based estimation (KDiCE) algorithm from [7] to multiple kernels to improve flexibility and performance. By this, we obtain the multikernel distributed consensus-based estimation (MKDiCE) algorithm. We illustrate its performance by numerical evaluations on the reconstruction of a static and a time-varying nonlinear function based on a diffusion field model.

## II. Preliminaries

### A. System Model

We consider a network of $N_\mathrm{n}$ nodes described by an undirected graph $\mathcal{G} = (\mathcal{J}, \mathcal{E})$ with a set of nodes $\mathcal{J}$ and a set of edges $\mathcal{E}$. The set $\mathcal{E}$ represents the connections among neighboring nodes in the network over which an information exchange is possible. We assume that the graph $\mathcal{G}$ is connected, i.e., each node can be reached by any other node in the network over single or multiple hops. Furthermore, to each node $j$ belongs a set of neighbors $\mathcal{N}_j$ containing all nodes connected to it. We assume that the network performs measurements of an arbitrarily nonlinear function $f : \mathcal{X} \to \mathbb{R}$ mapping samples from the input space $\mathcal{X} \subset \mathbb{R}^{N_\mathrm{i}}$ into the output space $\mathbb{R}$. Let us denote $d_j$ as the scalar measurement of $f(\boldsymbol{x})$ taken by node $j$ with its individual $N_\mathrm{i} \times 1$ regression vector $\boldsymbol{x}_j \in \mathcal{X}$. The measurement by node $j$ is then given by

$$d_j = f(\boldsymbol{x}_j) + n_j. \tag{1}$$

We denote by $n_j$ zero mean white Gaussian noise of variance $\sigma_n^2$. If we assume one regression vector $\boldsymbol{x}_j$ per node $j$, in total there are $N_\mathrm{n}$ input-output data pairs $\{\boldsymbol{x}_j, d_j\}_{j=1}^{N_\mathrm{n}}$ available in the network. Based on these data pairs the objective is to perform a nonlinear regression of the unknown function $f(\boldsymbol{x})$.

978-1-5090-3009-5/17/$31.00 ©2017 IEEE

### B. Problem Formulation

To reconstruct the unknown nonlinear function $f$ we employ multiple reproducing kernels by adopting the approach of multikernel adaptive filtering proposed originally in [5]. A kernel $\kappa_m : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive-definite function mapping input samples $\boldsymbol{x} \in \mathcal{X}$ to functions $\kappa_m(\boldsymbol{x}, \cdot)$ in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}_m$ [2]. When multiple kernels are employed many such RKHSs $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_{N_\mathrm{k}}$ are induced, correspondingly, each by its individual kernel function. To exploit these RKHSs, we approximate $f$ by a linear combination of $N_\mathrm{k}$ kernel functions such that its approximation $\tilde{f}$ lies in the sum space of these RKHSs [6]

$$\tilde{f} \in \mathcal{H}^+ := \mathcal{H}_1 + \mathcal{H}_2 + \ldots + \mathcal{H}_{N_\mathrm{k}}. \tag{2}$$

By this, we embed a higher flexibility into the reconstruction process compared to using a single kernel only. This is particularly useful if the function $f$ comprises several distinct elements such as high and low frequency components. A typical kernel function is the Gaussian kernel

$$\kappa_m(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{2\zeta_m^2}\right), \quad \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}, \tag{3}$$

where $\zeta_m$ is the kernel bandwidth controlling the width of the Gaussian shape. To express the approximation of $f$ by $N_\mathrm{k}$ kernels, let us define the set $\mathcal{D} = \{\bar{\boldsymbol{x}}_j\}_{j=1}^{N_\mathrm{d}} \subseteq \{\boldsymbol{x}_j\}_{j=1}^{N_\mathrm{n}}$ to be the dictionary selecting a subset of the $N_\mathrm{n}$ available regression vectors of all nodes in the network[1]. Then, we can approximate the output of the unknown function $f(\boldsymbol{x})$ by

$$\tilde{f}(\boldsymbol{x}) = \sum_{m=1}^{N_\mathrm{k}} \sum_{j=1}^{N_\mathrm{d}} w_j^{(m)} \kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_j), \tag{4}$$

with $w_j^{(m)}$ being the weighting coefficient for dictionary entry $j$ and kernel $\kappa_m$. We can express (4) in terms of an inner product by stacking the coefficients $w_j^{(m)}$ and corresponding kernel evaluations $\kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_j)$ for one kernel $\kappa_m$ into vectors of dimension $N_\mathrm{d} \times 1$ as

$$\boldsymbol{w}^{(m)} = \left[w_1^{(m)}, w_2^{(m)}, \ldots, w_{N_\mathrm{d}}^{(m)}\right]^\mathrm{T}, \tag{5a}$$

$$\boldsymbol{\kappa}_m(\boldsymbol{x}) = \left[\kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_1), \kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_2), \ldots, \kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_{N_\mathrm{d}})\right]^\mathrm{T}. \tag{5b}$$

Vector $\boldsymbol{\kappa}_m(\boldsymbol{x})$ contains kernel evaluations between $\boldsymbol{x}$ and each dictionary element $\bar{\boldsymbol{x}}_j$ using the kernel function $\kappa_m$. Hence, we can reformulate (4) by

$$\tilde{f}(\boldsymbol{x}) = \sum_{m=1}^{N_\mathrm{k}} \sum_{j=1}^{N_\mathrm{d}} w_j^{(m)} \kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_j) = \sum_{m=1}^{N_\mathrm{k}} (\boldsymbol{w}^{(m)})^\mathrm{T} \boldsymbol{\kappa}_m(\boldsymbol{x}). \tag{6}$$

Let us further stack $\boldsymbol{w}^{(m)}$ and $\boldsymbol{\kappa}_m$ for all $N_\mathrm{k}$ kernels into vectors of dimension $N_\mathrm{k}N_\mathrm{d} \times 1$:

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}^{(1)} \\ \boldsymbol{w}^{(2)} \\ \vdots \\ \boldsymbol{w}^{(N_\mathrm{k})} \end{bmatrix}, \qquad \boldsymbol{\kappa}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\kappa}_1(\boldsymbol{x}) \\ \boldsymbol{\kappa}_2(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{\kappa}_{N_\mathrm{k}}(\boldsymbol{x}) \end{bmatrix}. \tag{7}$$

[1]The selection of the dictionary samples can be done e.g. by the *coherence criterion* as in [5].

Finally, we can express (4) by

$$\begin{aligned} \tilde{f}(\boldsymbol{x}) &= \sum_{m=1}^{N_\mathrm{k}} \sum_{j=1}^{N_\mathrm{d}} w_j^{(m)} \kappa_m(\boldsymbol{x}, \bar{\boldsymbol{x}}_j) = \sum_{m=1}^{N_\mathrm{k}} (\boldsymbol{w}^{(m)})^\mathrm{T} \boldsymbol{\kappa}_m(\boldsymbol{x}) \\ &= \boldsymbol{w}^\mathrm{T} \boldsymbol{\kappa}(\boldsymbol{x}). \end{aligned} \tag{8}$$

Based on this representation we formulate a least squares (LS) problem with respect to (w.r.t.) the weight vector $\boldsymbol{w}$ for a nonlinear regression of $f(\boldsymbol{x})$ exploiting all data pairs $\{\boldsymbol{x}_j, d_j\}_{j=1}^{N_\mathrm{n}}$ in the network:

$$\boldsymbol{w}^\star = \underset{\boldsymbol{w} \in \mathbb{R}^{N_\mathrm{k}N_\mathrm{d}}}{\arg\min} \sum_{j=1}^{N_\mathrm{n}} (d_j - \boldsymbol{w}^\mathrm{T} \boldsymbol{\kappa}(\boldsymbol{x}_j))^2 \tag{9}$$

This problem can be solved by centralized estimators as in [7], [10] where all data pairs $\{\boldsymbol{x}_j, d_j\}_{j=1}^{N_\mathrm{n}}$ are available at a single node. However, here we focus on solving (9) in a distributed way.

## III. Proposed Algorithm

### A. Multikernel Distributed Consensus-Based Estimation

To achieve a distributed solution of (9), we introduce individual weight vectors $\boldsymbol{w}_j \in \mathbb{R}^{N_\mathrm{k}N_\mathrm{d}}$ for each node $j$ in the network as an estimate of the optimal weight vector $\boldsymbol{w}^\star$. By this, each node is able to compute its own local weight vector $\boldsymbol{w}_j$. We formulate a distributed LS problem on the set of weight vectors $\{\boldsymbol{w}_j\}_{j \in \mathcal{J}}$:

$$\{\boldsymbol{w}_j^\star | j \in \mathcal{J}\} = \underset{\{\boldsymbol{w}_j | j \in \mathcal{J}\}}{\arg\min} \sum_{j=1}^{N_\mathrm{n}} (d_j - \boldsymbol{w}_j^\mathrm{T} \boldsymbol{\kappa}(\boldsymbol{x}_j))^2 \tag{10a}$$

$$\text{s.t. } \boldsymbol{w}_j = \boldsymbol{w}_i \quad \forall j \in \mathcal{J}, i \in \mathcal{N}_j, \tag{10b}$$

where we added the consensus constraint (10b) in order to guarantee that all weight vectors in the network converge to the same solution. However, due to a direct coupling between weight vectors $\boldsymbol{w}_j$ and $\boldsymbol{w}_i$ among neighboring nodes, a parallel processing at the nodes cannot be achieved. Thus, in order to enable parallel computations of the weight vectors in the network, we decouple (10b) by an intermediate variable $\boldsymbol{z}_j$ per node $j$ and express (10b) equivalently by $\boldsymbol{w}_j = \boldsymbol{z}_i, \boldsymbol{z}_j = \boldsymbol{w}_j, \forall j \in \mathcal{J}, i \in \mathcal{N}_j$. To solve (10a) together with the decoupled constraints we apply the alternating direction method of multipliers (ADMM) [11] following [7], [12] to obtain a parallel distributed . Eventually, we achieve iterative update equations with iteration index $k$ analog to [7] but extended for multiple kernels:

$$\boldsymbol{z}_j^{k+1} = \frac{\mu}{|\mathcal{N}_j'|} \sum_{i \in \mathcal{N}_j'} \frac{1}{\mu} \boldsymbol{w}_i^k - \boldsymbol{\lambda}_{ij}^k \tag{11a}$$

$$\boldsymbol{\lambda}_{ji}^{k+1} = \boldsymbol{\lambda}_{ji}^k - \frac{1}{\mu} \left(\boldsymbol{w}_j^k - \boldsymbol{z}_i^{k+1}\right) \tag{11b}$$

$$\begin{aligned} \boldsymbol{w}_j^{k+1} = &\left(\boldsymbol{\kappa}(\boldsymbol{x}_j)\boldsymbol{\kappa}(\boldsymbol{x}_j)^\mathrm{T} + \frac{|\mathcal{N}_j'|}{\mu} \boldsymbol{I}_{N_\mathrm{k}N_\mathrm{d}}\right)^{-1} \cdot \\ &\left(d_j \boldsymbol{\kappa}(\boldsymbol{x}_j) + \sum_{i \in \mathcal{N}_j'} \frac{1}{\mu} \boldsymbol{z}_i^{k+1} + \boldsymbol{\lambda}_{ji}^{k+1}\right). \end{aligned} \tag{11c}$$

where $\mu$ is a positive step size and $\mathcal{N}_j' = \mathcal{N}_j \cup \{j\}$ is the set of neighbors including the node $j$ itself. Moreover, variable $\boldsymbol{\lambda}_{ji}$ represents the Lagrange multiplier stemming from the optimization method used. It considers how well the consensus constraint between $\boldsymbol{w}_j$ and $\boldsymbol{z}_i$ among neighboring nodes $j$ and $i$ is fulfilled. The iterative equations (11a)-(11c) build the proposed MKDiCE algorithm for a distributed computation of the optimal weight vector $\boldsymbol{w}^\star$ in (9). The variables of the algorithm are initialized as $\boldsymbol{w}_j^0 = \boldsymbol{\lambda}_{ij}^0 = \mathbf{0}$ for all nodes $j \in \mathcal{J}$. Note that the KDiCE [7] is a special case of the MKDiCE algorithm if we use one kernel function only. Furthermore, the dimension of the vectors $\boldsymbol{w}_j^{k+1}, \boldsymbol{z}_j^{k+1}, \boldsymbol{\lambda}_{ji}^{k+1}$ is scaled by $N_\mathrm{k}$ compared to the KDiCE due to the use of multiple kernels. This might come with an increased communication overhead in the network due to a higher dimension of the vectors to be exchanged. However, the required number of dictionary samples $N_\mathrm{d}$ can be reduced significantly by using multiple kernels such that, eventually, the dimension of the vectors $N_\mathrm{k}N_\mathrm{d}$ is scaled down.

With each node $j$ possessing its own weight vector $\boldsymbol{w}_j^k$, it can estimate the output of the unknown function $f(\boldsymbol{x})$ per iteration $k$ for arbitrary input samples $\boldsymbol{x}$ using the approximation (8):

$$\tilde{f}_j^k(\boldsymbol{x}) = (\boldsymbol{w}_j^k)^\mathrm{T} \boldsymbol{\kappa}(\boldsymbol{x}) \tag{12}$$

Note that for the evaluation of the complete $N_\mathrm{k}N_\mathrm{d} \times 1$ vector $\boldsymbol{\kappa}(\boldsymbol{x}) = [\kappa_1(\boldsymbol{x}, \bar{\boldsymbol{x}}_1), \ldots, \kappa_2(\boldsymbol{x}, \bar{\boldsymbol{x}}_1), \ldots, \kappa_{N_\mathrm{k}}(\boldsymbol{x}, \bar{\boldsymbol{x}}_{N_\mathrm{d}})]^\mathrm{T}$ each node $j$ requires full knowledge of the dictionary $\mathcal{D}$.

### B. Communication Overhead

In order to quantify the overhead of the algorithms we first consider the total number of transmissions in the network. This number will be the same for the MKDiCE and KDiCE since only the dimension of the vectors to be exchanged differs. For both algorithms, the variables $\boldsymbol{w}_j, \boldsymbol{z}_j$ are transmitted in a broadcast fashion by each node $j$ in the network per iteration $k$ resulting in $2N_\mathrm{n}$ transmissions. The Lagrange multipliers $\boldsymbol{\lambda}_{ji}$ have to be exchanged in a unicast fashion since they are specifically related to node $j$ and node $i$. Hence, there are two transmissions per edge in the network resulting in $2|\mathcal{E}|$ transmissions and the total number of transmissions by all nodes per iteration $k$ is given by $2|\mathcal{E}| + 2N_\mathrm{n}$. We consider the total number of scalar entries of the vectors to be exchanged in the network as the communication overhead $N_\mathrm{o}$. Then, for each algorithm the overhead is given by the number of transmissions scaled by the dimension of the exchanged vectors:

$$\text{KDiCE:} \qquad N_\mathrm{o} = (2|\mathcal{E}| + 2N_\mathrm{n}) \cdot N_\mathrm{d} \tag{13a}$$

$$\text{MKDiCE:} \qquad N_\mathrm{o} = (2|\mathcal{E}| + 2N_\mathrm{n}) \cdot N_\mathrm{d}N_\mathrm{k} \tag{13b}$$

The communication overhead for MKDiCE is scaled by $N_\mathrm{k}$ compared to KDiCE. As mentioned above, in MKDiCE this increase can be compensated by a smaller required dictionary size $N_\mathrm{d}$.

## IV. Numerical Results

For the numerical evaluations, we assume a network of $N_\mathrm{n} = 100$ nodes randomly deployed on a unit-square $\mathcal{X} = [0, 1]^2$ sampling the scalar function $f(\boldsymbol{x})$. Thus, the input sample $\boldsymbol{x}$ is the Cartesian position with $\boldsymbol{x} = [x_1, x_2]^\mathrm{T} \in \mathcal{X}$. Nodes having a distance $r \leq 0.3$ to each other are connected by an edge and the noise variance for the measurement of the nodes is $\sigma_n^2 = 0.01$. For the reconstruction of $f$, we apply the MKDiCE and the KDiCE algorithm and choose the Gaussian kernel (3) as kernel function. The Cartesian position $\boldsymbol{x}_j$ of each node $j$ is used as regression vector and the function value at the node's position builds the corresponding measurement $d_j$ as in (1). As error measure, we consider the $\mathrm{MSE}^k$ at iteration $k$ between the true and the reconstructed function averaged over the whole unit-square $\mathcal{X}$ which is uniformly sampled by $N_\mathrm{g}$ spatial grid points and over all $N_\mathrm{n}$ nodes:

$$\mathrm{MSE}^k = \frac{1}{N_\mathrm{n}} \frac{1}{N_\mathrm{g}} \sum_{j=1}^{N_\mathrm{n}} \sum_{l=1}^{N_\mathrm{g}} |f(\boldsymbol{x}_l) - \tilde{f}_j^k(\boldsymbol{x}_l)|^2 \tag{14}$$

For numerical evaluations, $\mathrm{MSE}^k$ is averaged over 100 independent trials with a new realization of network topology in each trial.

### A. Multiple Gaussian Impulses

We consider the superposition of two Gaussian impulses with different widths as the unknown nonlinear function $f(\boldsymbol{x})$ to be reconstructed:

$$f(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{p}_1||^2}{2 \cdot 0.1^2}\right) + \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{p}_2||^2}{2 \cdot 0.3^2}\right),$$

with $\boldsymbol{p}_1 = [0.3, 0.3]^\mathrm{T}, \boldsymbol{p}_2 = [0.8, 0.6]^\mathrm{T}$. The widths of the impulses are chosen such that two distinct shapes are generated. For both algorithms, we set the step size to $\mu = 1$. To build the dictionary $\mathcal{D}$ we apply the coherence criterion over all node positions $\boldsymbol{x}_j$ in the network. Hence, a regression vector or node position $\boldsymbol{x}_j$ is added into the dictionary $\mathcal{D}$ if the following condition is satisfied:

$$\max_{m=1,\ldots,N_\mathrm{k}} \max_{\ell=1,\ldots,N_\mathrm{d}} |\kappa_m(\boldsymbol{x}_j, \bar{\boldsymbol{x}}_\ell)| \leq \tau, \tag{15}$$

with $\tau > 0$ being the coherence threshold which controls the dictionary size $N_\mathrm{d}$. We set $\tau$ such that all algorithms have the same dictionary size $N_\mathrm{d}$. This can be done by considering the highest kernel bandwidth $\zeta$ among all kernel functions since this value is significant for the coherence criterion when employing multiple kernels. If $\zeta_a$ is the highest bandwidth in a set of kernel functions and $\zeta_b$ is the highest bandwidth in another set of kernel functions, the same dictionary size can be achieved if $\tau_a^{\zeta_a^2} = \tau_b^{\zeta_b^2}$ [5]. By this, we guarantee a fair comparison between the algorithms. Note, that the generation of the dictionary $\mathcal{D}$ is done beforehand in each trial and stays fixed for the specific algorithm. For the MKDiCE algorithm we choose two and three Gaussian kernels with different bandwidths, respectively. We assume the same dictionary for each kernel. We compare the performance to the KDiCE with

TABLE I
PARAMETER VALUES FOR EXPERIMENT IN IV-A

| Algorithm | Bandwidth | Coherence |
|---|---|---|
| KDiCE$_1$ | $\zeta = 0.1$ | $\tau = 0.39$ |
| KDiCE$_2$ | $\zeta = 0.2$ | $\tau = 0.79$ |
| MKDiCE$_1$ | $\zeta_1 = 0.1$<br>$\zeta_2 = 0.2$<br>$\zeta_3 = 0.3$ | $\tau = 0.9$ |
| MKDiCE$_2$ | $\zeta_1 = 0.1$<br>$\zeta_2 = 0.2$ | $\tau = 0.79$ |

bandwidths $\zeta = 0.1$ and $\zeta = 0.2$, respectively. The former is chosen as an exemplary mismatch of the kernel bandwidth whereas the latter is well matched to function $f$. The coherence threshold $\tau$ is set such that the resulting average dictionary size for all algorithms is $\bar{N}_\mathrm{d} = 27$. Table I lists the chosen parameters.

Figure 1 depicts the reconstruction mean square error (MSE) over iteration $k$ for both algorithms. We can observe that MKDiCE$_1$ and MKDiCE$_2$ outperform both single kernel approaches in terms of steady-state error and convergence speed. Furthermore, two kernels are sufficient since there is no significant performance improvement by MKDiCE$_1$ compared to MKDiCE$_2$. Although $\zeta_2 = 0.2$ of MKDiCE$_2$ does not match the bandwidth of the second impulse in $f(\boldsymbol{x})$, the MKDiCE is still able to reconstruct the function successfully.

In order to analyze the impact of multiple kernels on the overhead in the network we consider the MSE over the communication overhead in the network caused by the algorithms. This is shown in Figure 2. We find that although the dimension of the vectors to be exchanged is scaled by $N_\mathrm{k}$ the MKDiCE still outperforms the KDiCE consuming a lower overhead for the same reconstruction performance. E.g. to achieve an MSE of about $-23$ dB only half of the overhead is consumed by MKDiCE$_2$ compared to KDiCE$_2$. Comparing MKDiCE$_1$ to MKDiCE$_2$ the effect of an additional kernel can be observed by a slightly increased overhead in the first iterations.

Figure 3 shows the MSE over different average dictionary sizes $\bar{N}_\mathrm{d}$ where MKDiCE$_1$ is omitted due to its similar performance to MKDiCE$_2$. The MSE value for a specific dictionary size is generated by averaging over the last 200 values of the reconstruction MSE. We observe that MKDiCE$_2$ outperforms both KDiCE$_1$ and KDiCE$_2$. In particular, for dictionary sizes $\bar{N}_\mathrm{d} > 20$ MKDiCE$_2$ starts to outperform KDiCE$_2$ while the performance of KDiCE$_1$ is significantly lower over the whole range. For a dictionary size of approximately $\bar{N}_\mathrm{d} > 45$ no relevant improvement is visible for MKDiCE$_2$.

### B. Dynamic Diffusion Field

In the following, we investigate the tracking performance of the MKDiCE for a time-varying function. To this end, we consider a dynamic diffusion field generated by $L = 2$ instantaneous and localized sources [13]. Each diffusion source $l$ has intensity $c_l$, activation time $t_l$ and Cartesian position $\boldsymbol{p}_l$.

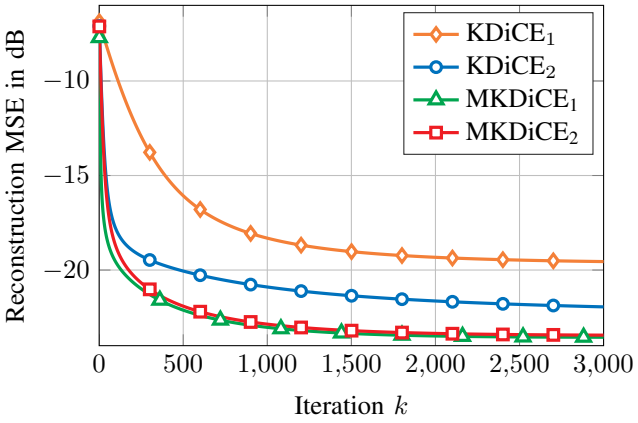


Fig. 1. MSE over iteration $k$ for reconstruction of two Gaussian impulses with an average dictionary size of $\bar{N}_\mathrm{d} = 27$.

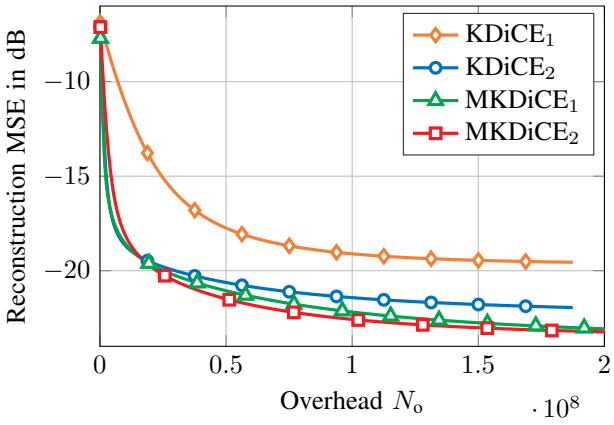


Fig. 2. MSE over communication overhead $N_\mathrm{o}$ for reconstruction of two Gaussian impulses with an average dictionary size of $\bar{N}_\mathrm{d} = 27$.

The function at time $t$ and coordinate $\boldsymbol{x}$ is given by

$$f(\boldsymbol{x}, t) = \sum_{l=1}^{L} \frac{c_l}{4\pi\nu(t - t_l)} \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{p}_l||^2}{4\nu(t - t_l)}\right) \cdot h(t - t_l).$$

with source intensities $c_1 = 1, c_2 = 0.7$, positions $\boldsymbol{p}_1 = [0.3, 0.3]^\mathrm{T}, \boldsymbol{p}_2 = [0.8, 0.6]^\mathrm{T}$ and activation times $t_1 = 0, t_2 = 30$. The function $h(t - t_l)$ is the Heaviside-function and $\nu$ is the diffusion constant of the medium chosen as $\nu = 0.01$. To track the diffusion field, each node samples the field at its position $\boldsymbol{x}_j$ every $\Delta t = 0.1$ time instant. Based on each measured sample $d_j$ the algorithms perform one full iteration such that sampling index and iteration index are equivalent to each other. For both algorithms we choose the step size as $\mu = 1$. For the KDiCE we use one Gaussian kernel with bandwidth $\zeta = \sqrt{2\nu}$ which is matched to the diffusion constant to give the best performance. Furthermore, we choose the coherence threshold $\tau = 0.95$ resulting in an average dictionary size of $\bar{N}_\mathrm{d} = 76$. For the MKDiCE we use three Gaussian kernels with bandwidths $\zeta_1 = 0.1, \zeta_2 = 0.2, \zeta = 0.3$ and coherence threshold $\tau = 0.85$ resulting in an average dictionary size of $\bar{N}_\mathrm{d} = 20$. As reference we include a centralized

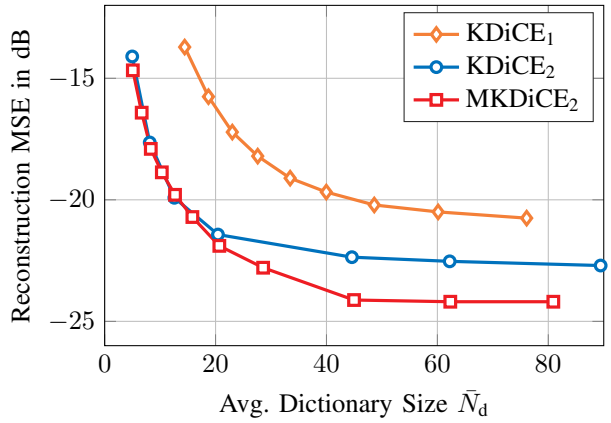


Fig. 3. MSE over average dictionary size $\bar{N}_\mathrm{d}$ for reconstruction of two Gaussian impulses.

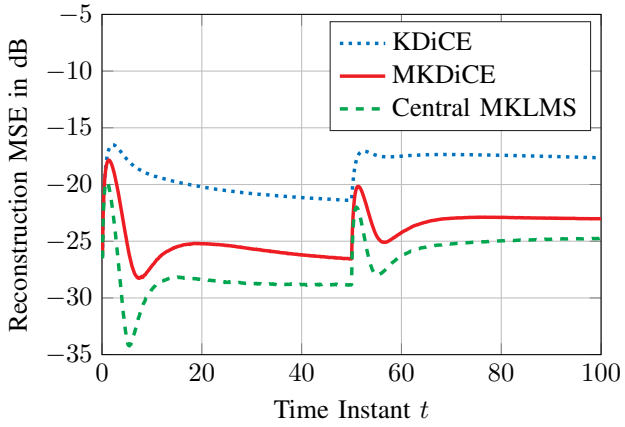


Fig. 4. MSE over time instant $t$ for tracking of two diffusive sources activated at different time instances and expanding over time.

tracking algorithm namely the multikernel least-mean-squares (MKLMS) exploiting all data pairs in the network based on the new measurement $d_j^{k+1}$ in each iteration:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \mu \sum_{j \in \mathcal{J}} \left( d_j^{k+1} - (\boldsymbol{w}^k)^\mathrm{T} \boldsymbol{\kappa}(\boldsymbol{x}_j) \right) \boldsymbol{\kappa}(\boldsymbol{x}_j) \quad (16)$$

We use the same bandwidths and same coherence threshold as in MKDiCE. The step size is chosen as $\mu = 10^{-3}$. Note that for all algorithms the kernel bandwidths are not adapted over time but stay fixed.

Figure 4 depicts the MSE over iteration $k$. It is clearly visible that the MKDiCE outperforms the KDiCE significantly in terms of tracking speed and performance. By employing multiple kernels the algorithm is able to combine these kernels in a flexible way and thus, to track the expansion of the diffusive sources with higher accuracy. Furthermore, the MKDiCE performs close to the centralized MKLMS with a gap of approximately 3 dB. In contrast, the KDiCE fails at tracking the sources with satisfying performance, especially after the activation of the second source. This is due to the limitation by one kernel function since the bandwidth does not suffice to model the superposition of both sources properly. Regarding the MKDiCE we observe a higher steady-state error after the superposition as well. However, this loss is limited as multiple kernels can compensate for the mismatch caused by the activation of the second source. Note that in MKDiCE no exact a-priori knowledge of the diffusion constant $\nu$ is required to choose appropriate kernel bandwidths in contrast to the KDiCE. Based on Figure 4 it is obvious that in terms of communication overhead the MKDiCE will consume less exchanges to achieve the same performance as KDiCE.

## V. Conclusion

We addressed the task of nonlinear distributed regression with multiple kernels by a network of nodes. To this end, we combined the KDiCE with multiple kernels. We tested the resulting MKDiCE algorithm on static and time-varying nonlinear functions and could observe performance gains versus the KDiCE especially for the tracking of a diffusion field. Future work involves the integration of dictionary learning and an appropriate kernel selection into the distributed procedure.

## Acknowledgment

The work leading to this publication was partially funded by the German Research Foundation (DFG) under grant Pa2507/1. M. Yukawa is thankful to JSPS Grants-in-Aid (15K06081, 15K13986, 15H02757).

## References

[1] M. Kasparick, R. L. G. Cavalcante, S. Valentin, S. Stańczak, and M. Yukawa, "Kernel-based adaptive online reconstruction of coverage maps with side information," *IEEE Trans. on Vehicular Technology*, vol. 65, no. 7, pp. 5461–5473, 2016.

[2] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[3] W. Liu, J. C. Príncipe, and S. Haykin, *Kernel Adaptive Filtering*. John Wiley & Sons, 2010.

[4] W. Gao, J. Chen, C. Richard, J. Huang, and R. Flamary, "Kernel LMS algorithm with forward-backward splitting for dictionary learning," in *IEEE ICASSP*, 2013, pp. 5735–5739.

[5] M. Yukawa, "Multikernel adaptive filtering," *IEEE Trans. on Signal Processing*, vol. 60, pp. 4672–4682, 2012.

[6] ——, "Adaptive learning in cartesian product of reproducing kernel hilbert spaces," *IEEE Trans. on Signal Processing*, vol. 63, pp. 1–18, 2015.

[7] B.-S. Shin, H. Paul, and A. Dekorsy, "Distributed kernel least squares for nonlinear regression applied to sensor networks," in *EUSIPCO*, September 2016.

[8] W. Gao, J. Chen, C. Richard, and J. Huang, "Diffusion adaptation over networks with kernel least-mean-square," in *IEEE CAMSAP*, 2015.

[9] B.-S. Shin, H. Paul, and A. Dekorsy, "Spatial field reconstruction with distributed kernel least squares in mobile sensor networks," in *Int. Conf. on Systems, Communications and Coding*, February 2017.

[10] F. Tobar and D. Mandic, "Multikernel least squares estimation," *Sensor Signal Processing for Defence*, 2012.

[11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010.

[12] H. Paul, J. Fliege, and A. Dekorsy, "In-network-processing: Distributed consensus-based linear estimation," *IEEE Commun. Lett.*, vol. 17, no. 1, pp. 59–62, 2013.

[13] Y. M. Lu, P. L. Dragotti, and M. Vetterli, "Localizing point sources in diffusion fields from spatiotemporal samples," in *Int. Conf. on Sampling Theory and Applications*, May 2011.