Extended Kernel Recursive Least Squares Algorithm

Weifeng Liu, *Member, IEEE*, Il (Memming) Park, Yiwen Wang, *Member, IEEE*, and José C. Príncipe, *Fellow, IEEE*

***Abstract*—This paper presents a kernelized version of the extended recursive least squares (EX-KRLS) algorithm which implements for the first time a general linear state model in reproducing kernel Hilbert spaces (RKHS), or equivalently a general nonlinear state model in the input space. The center piece of this development is a reformulation of the well known extended recursive least squares (EX-RLS) algorithm in RKHS which only requires inner product operations between input vectors, thus enabling the application of the kernel property (commonly known as the kernel trick). The first part of the paper presents a set of theorems that shows the generality of the approach. The EX-KRLS is preferable to 1) a standard kernel recursive least squares (KRLS) in applications that require tracking the state-vector of general linear state-space models in the kernel space, or 2) an EX-RLS when the application requires a nonlinear observation and state models. The second part of the paper compares the EX-KRLS in nonlinear Rayleigh multipath channel tracking and in Lorenz system modeling problem. We show that the proposed algorithm is able to outperform the standard KRLS and EX-RLS in both simulations.**

***Index Terms*—Extended recursive least squares, Kalman filter, kernel methods, reproducing kernel Hilbert spaces.**

I. Introduction

SOLID mathematical foundation, and widely successful applications are making kernel methods very popular. By the famed kernel trick, many linear methods have been recast in high dimensional reproducing kernel Hilbert spaces (RKHS) to yield more powerful nonlinear extensions, including support vector machines [1], principal component analysis [2], least-mean-square [3], affine projection algorithms [4], Adaline [5], etc. More recently, there have been many efforts to study the general formulation of recursive least squares in RKHS, such as kernel recursive least squares (KRLS) [6]–[8] and sliding-window KRLS (SW-KRLS) [9]. Moreover, due to the close relationship between the extended recursive least squares and the Kalman filter [10], [11], it will be of great significance to study the possibility of deriving an extended RLS in RKHS, because it may lead to a new research line to formulate a nonlinear Kalman filter. There are many nonlinear extensions of the classical Kalman filter in the Bayesian inference framework such as extended Kalman filter [11], [12], unscented Kalman filter [13], [14], particle filter [15] and cubature Kalman filter [16]. All of them approximate the state's posterior distributions by either linearization or sampling at each iteration and hence they are suboptimal in nature. However, as will be evidenced later, our model does not employ any approximate estimation strategy because it is based on a functional analysis framework (solving a set of linear equations identical to the Kalman filter in a space that is nonlinearly related to the input space).

In this line of work, a kernel Kalman filter is described in [17], [18], which uses a very simple observation model. Furthermore its recursion relies on subspace approximation through the kernel principal component analysis (which requires all the training data available before the recursion). By contrast, our algorithm is designed to learn the weight vector of a kernel regressor using a Kalman filter with an explicit state transition process and a more complicated observation model. And our algorithm is online, i.e., the learning is accomplished sample by sample. Although our current work is motivated to improve the tracking ability of the standard KRLS, it provides a promising framework to deal with general nonlinear state estimation problems with a wide applications in auto-piloting, brain-computer interface, dynamic positioning, forecasting, etc.

The organization of the paper is as follows. In Section II, related algorithms including recursive least squares, kernel recursive least squares and extended recursive least squares are briefly reviewed. Next in Section III, we focus on general formulation and properties of the extended kernel recursive least squares algorithm. Then tracking model based and subspace assumption based extended kernel recursive least squares are discussed in Sections IV and V respectively. Two experiments are studied in Section VI to support our theory. Finally Section VII summarizes the conclusions and future lines of research.

Manuscript received September 02, 2008; accepted March 30, 2009. First published May 02, 2009; current version published September 16, 2009. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Gerald Matz. This work was partially supported by NSF grant ECS-0601271.

W. Liu is with the Amazon.com, Seattle, WA 98104 USA (e-mail: weifeng@amazon.com).

I. Park is with the Department of Biomedical Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: memming@cnel.ufl.edu).

Y. Wang is with Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong (e-mail: eewangyw@ust.hk).

J. C. Príncipe is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: principe@cnel.ufl.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2009.2022007

II. A Review of Recursive Least Squares Techniques

We start with a review on recursive least squares, kernel recursive least squares, extended recursive least squares and general kernel methods. This section serves to establish the notations and help distinguish our contributions.

A. Recursive Least Squares

With a sequence of training data $\{\mathbf{u}(j), d(j)\}_{j=1}^{i-1}$ up to time $i-1$, the recursive least squares algorithm estimates the weight $\mathbf{w}(i-1)$ by minimizing the following cost

$$\min_{\mathbf{w}(i-1)} \sum_{j=1}^{i-1} \left| d(j) - \mathbf{u}(j)^T \mathbf{w}(i-1) \right|^2 + \lambda \left\| \mathbf{w}(i-1) \right\|^2 \quad (1)$$

Here $\mathbf{u}(j)$ is the $L \times 1$ regressor input, $d(j)$ is the desired response (which is assumed a scalar but extensions to multi-dimensional output is quite straightforward), and λ is the regularization parameter. When a new input-output pair $\{\mathbf{u}(i), d(i)\}$ becomes available, the weight estimate $\mathbf{w}(i)$ which is the minimizer of

$$\min_{\mathbf{w}(i)} \sum_{j=1}^{i} \left|d(j) - \mathbf{u}(j)^T \mathbf{w}(i)\right|^2 + \lambda \left\|\mathbf{w}(i)\right\|^2 \tag{2}$$

can be calculated recursively from the previous estimate $\mathbf{w}(i-1)$ without solving (2) directly. The standard recursive least squares (RLS) is summarized in Algorithm 1 [11].

Algorithm 1: Recursive Least Squares (RLS)

```
Initialize $\mathbf{w}(0) = 0, \mathbf{P}(0) = \lambda^{-1}\mathbf{I}$
Iterate for $i \geq 1$
    $r_e(i) = 1 + \mathbf{u}(i)^T \mathbf{P}(i-1)\mathbf{u}(i)$
    $\mathbf{k}_p(i) = \mathbf{P}(i-1)\mathbf{u}(i)/r_e(i)$
    $e(i) = d(i) - \mathbf{u}(i)^T \mathbf{w}(i-1)$
    $\mathbf{w}(i) = \mathbf{w}(i-1) + \mathbf{k}_p(i)e(i)$
    $\mathbf{P}(i) = \left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)/r_e(i)\right]$   (3)
```

In (3), $\mathbf{P}(i)$ can be interpreted as the inverted regularized data autocorrelation matrix, $\mathbf{k}_p(i)$ is called the gain vector, and $e(i)$ is the prediction error. Also notice that $\mathbf{P}$ is $L \times L$, where L is the dimensionality of the input $\mathbf{u}$. Therefore, the time and space complexities are both $O(L^2)$ at each iteration.

The RLS distributes the computational load evenly in each iteration, which is very appealing in adaptive signal processing, adaptive controls and communications [19] where data are available sequentially over time.

B. Kernel Methods

A kernel [20] is a continuous, symmetric, positive-definite function $\kappa : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$. $\mathbb{U}$ is the input domain, a compact subset of $\mathbb{R}^L$. The commonly used kernels include the Gaussian kernel (4) and the polynomial kernel (5):

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\left(-a\|\mathbf{u} - \mathbf{u}'\|^2\right) \tag{4}$$

$$\kappa(\mathbf{u}, \mathbf{u}') = (\mathbf{u}^T\mathbf{u}' + 1)^p. \tag{5}$$

Mercer theorem [20], [21] states that any kernel $\kappa(\mathbf{u}, \mathbf{u}')$ can be expanded as follows:

$$\kappa(\mathbf{u}, \mathbf{u}') = \sum_{i=1}^{\infty} \varsigma_i \phi_i(\mathbf{u})\phi_i(\mathbf{u}') \tag{6}$$

where ς_i and ϕ_i are the eigenvalues and the eigenfunctions respectively. The eigenvalues are non-negative. Therefore, a mapping $\boldsymbol{\varphi}$ can be constructed as

$$\begin{aligned} &\boldsymbol{\varphi} : \mathbb{U} \to \mathbb{F} \\ &\boldsymbol{\varphi}(\mathbf{u}) = \left[\sqrt{\varsigma_1}\phi_1(\mathbf{u}), \sqrt{\varsigma_2}\phi_2(\mathbf{u}), \cdots\right]. \end{aligned} \tag{7}$$

By construction, the dimensionality of $\mathbb{F}$ is determined by the number of strictly positive eigenvalues, which can be infinite in the Gaussian kernel case. In the machine learning literature, $\boldsymbol{\varphi}$ is usually treated as the feature mapping and $\boldsymbol{\varphi}(\mathbf{u})$ is the transformed feature vector lying in the feature space $\mathbb{F}$ (which is an inner product space). By doing so, an important implication is

$$\boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}'). \tag{8}$$

$\mathbb{F}$ is isometric-isomorphic to the RKHS induced by the kernel, which is a functional space. By slightly abusing notation, we do not distinguish these two spaces in this paper if no confusion is involved.

The main idea of kernel methods can be summarized as follows: transform the input data into a high-dimensional feature space via a positive definite kernel such that the inner product operation in the feature space can be computed efficiently through the kernel evaluation (8). Then appropriate linear methods are subsequently applied on the transformed data. As long as we can formulate the algorithm in terms of inner product (or equivalent kernel evaluation), we never have to explicitly compute in the high dimensional feature space.

C. Kernel Recursive Least Squares

The kernel recursive least squares is systematically studied in [8] and we provide a brief review here to better distinguish and appreciate our contribution. First, Mercer theorem is used to transform the data $\mathbf{u}(i)$ into the feature space $\mathbb{F}$ as $\boldsymbol{\varphi}(\mathbf{u}(i))$ (denoted as $\boldsymbol{\varphi}(i)$). Then the recursive least squares algorithm is formulated on the example sequence $\{d(1), d(2), \ldots\}$ and $\{\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), \ldots\}$. At each iteration, the weight vector $\boldsymbol{\omega}(i)$, which is the minimizer of

$$\min_{\boldsymbol{\omega}(i)} \sum_{j=1}^{i} \left|d(j) - \boldsymbol{\omega}(i)^T \boldsymbol{\varphi}(j)\right|^2 + \lambda \left\|\boldsymbol{\omega}(i)\right\|^2 \tag{9}$$

needs to be solved recursively as in (3). However, (3) cannot be directly applied here because the dimensionality of $\boldsymbol{\varphi}(j)$ is so high (can be even infinite) that it is not feasible in practice.

By introducing

$$\begin{aligned} \mathbf{d}(i) &= [d(1), \cdots, d(i)]^T \\ \boldsymbol{\Phi}(i) &= [\boldsymbol{\varphi}(1), \cdots, \boldsymbol{\varphi}(i)] \end{aligned}$$

one has

$$\boldsymbol{\omega}(i) = \left[\lambda\mathbf{I} + \boldsymbol{\Phi}(i)\boldsymbol{\Phi}(i)^T\right]^{-1} \boldsymbol{\Phi}(i)\mathbf{d}(i) \tag{10}$$

further by the matrix inversion lemma [19],

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\left[\lambda\mathbf{I} + \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)\right]^{-1} \mathbf{d}(i). \tag{11}$$

We must emphasize the significance of the change from (10) to (11) here. First, $\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)$ is computable by the kernel trick (8) and second, the weight is explicitly expressed as a linear combination of the transformed input data $\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\mathbf{a}(i)$.

Denoting

$$\mathbf{Q}(i) = \left[\lambda\mathbf{I} + \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)\right]^{-1} \tag{12}$$

one has

$$\mathbf{Q}(i)^{-1} = \begin{bmatrix} \mathbf{Q}(i-1)^{-1} & \mathbf{h}(i) \\ \mathbf{h}(i)^T & \lambda + \boldsymbol{\varphi}(i)^T\boldsymbol{\varphi}(i) \end{bmatrix} \tag{13}$$

where $\mathbf{h}(i) = \boldsymbol{\Phi}(i-1)^T\boldsymbol{\varphi}(i)$. Hence, updating the inverse of this growing matrix can be quite efficient [19]

$$\mathbf{Q}(i) = r(i)^{-1} \begin{bmatrix} \mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix} \tag{14}$$

where

$$\begin{aligned} \mathbf{z}(i) &= \mathbf{Q}(i-1)\mathbf{h}(i) \\ r(i) &= \lambda + \boldsymbol{\varphi}(i)^T\boldsymbol{\varphi}(i) - \mathbf{z}(i)^T\mathbf{h}(i). \end{aligned} \tag{15}$$

Therefore the expansion coefficients of the weight are

$$\begin{aligned} \mathbf{a}(i) &= \mathbf{Q}(i)\mathbf{d}(i) \\ &= \begin{bmatrix} \mathbf{Q}(i-1) + \mathbf{z}(i)\mathbf{z}(i)^T r(i)^{-1} & -\mathbf{z}(i)r(i)^{-1} \\ -\mathbf{z}(i)^T r(i)^{-1} & r(i)^{-1} \end{bmatrix} \\ &\quad \times \begin{bmatrix} \mathbf{d}(i-1) \\ d(i) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}(i)r(i)^{-1}e(i) \\ r(i)^{-1}e(i) \end{bmatrix} \end{aligned} \tag{16}$$

where $e(i)$ is the prediction error computed by the difference between the desired signal and the prediction $f_{i-1}(\mathbf{u}(i))$:

$$\begin{aligned} f_{i-1}(\mathbf{u}(i)) &= \mathbf{h}(i)^T\mathbf{a}(i-1) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa(\mathbf{u}(j), \mathbf{u}(i)) \\ e(i) &= d(i) - f_{i-1}(\mathbf{u}(i)). \end{aligned}$$

As we can see, the structure of KRLS is similar to a radial basis function (RBF) network at any time i [22] (of course KRLS is not restricted to RBF kernels and theoretically any positive definite kernel can be used). $\mathbf{a}_j(i-1)$ is the jth component of $\mathbf{a}(i-1)$ and all the previous data serve as the centers. The coefficients $\mathbf{a}(i)$ and the centers $\mathcal{C}(i)$ should be stored in the computer during training. The updates needed for KRLS at time i is

$$\mathbf{a}_i(i) = r(i)^{-1}e(i) \tag{17}$$

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) - r(i)^{-1}e(i)\mathbf{z}_j(i),\ j = 1, \cdots, i-1 \tag{18}$$

$$\mathcal{C}(i) = \{\mathcal{C}(i-1), \mathbf{u}(i)\}. \tag{19}$$

KRLS is summarized in Algorithm 2. The time and space complexities are both $O(i^2)$ at iteration i. The complexity increases as more training data become available, so sparsification techniques are usually used to curtail the growth [8], which will be discussed later.

Algorithm 2: Kernel Recursive Least Squares (KRLS)

```
Initialize $\mathbf{Q}(1) = (\lambda + \kappa(\mathbf{u}(1), \mathbf{u}(1)))^{-1}$, $\mathbf{a}(1) = \mathbf{Q}(1)d(1)$
Iterate for $i > 1$:
    $\mathbf{h}(i) = [\kappa(\mathbf{u}(i), \mathbf{u}(1)), \cdots, \kappa(\mathbf{u}(i), \mathbf{u}(i-1))]^T$
    $\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{h}(i)$
    $r(i) = \lambda + \kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{z}(i)^T\mathbf{h}(i)$
    $\mathbf{Q}(i) = r(i)^{-1} \begin{bmatrix} \mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix}$
    $e(i) = d(i) - \mathbf{h}(i)^T\mathbf{a}(i-1)$
    $\mathbf{a}(i) = \begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}(i)r(i)^{-1}e(i) \\ r(i)^{-1}e(i) \end{bmatrix}$    (20)
```

D. Extended Recursive Least Squares

The problem for RLS (and KRLS) is that it has a poor tracking performance. From the viewpoint of state-space model, RLS implicitly assumes that the data satisfy [11]

$$\begin{aligned} \mathbf{x}(i+1) &= \mathbf{x}(i) \\ d(i) &= \mathbf{u}(i)^T\mathbf{x}(i) + v(i) \end{aligned} \tag{21}$$

that is, the state $\mathbf{x}(i)$ is fixed over time and with no surprise the optimal estimate of the state $\mathbf{w}(i)$ cannot track variations.

To improve its tracking ability, several techniques can be employed. For example, the data can be exponentially-weighted over time or a truncated window can be applied on the training data (which also can be viewed as a special weighting). The work for a sliding-window KRLS has already been done in [9] and we focus on a formulation of extended kernel recursive least squares with an explicit state transition process including model uncertainty, which includes the exponentially-weighted scheme as a special case [19].

As is known from sequential estimation, a more general linear state-space model is

$$\begin{aligned} \mathbf{x}(i+1) &= \mathbf{A}\mathbf{x}(i) + \mathbf{n}(i) \quad \text{(state model)} \\ d(i) &= \mathbf{u}(i)^T\mathbf{x}(i) + v(i) \quad \text{(observation model)} \end{aligned} \tag{22}$$

with $\mathbf{A}$ as the state transition matrix, $\mathbf{n}(i)$ the state noise and $v(i)$ the observation noise (both assumed Gaussian distributed, white spatially and temporally for simplicity).

Kalman [10] proposed a two step sequential estimation algorithm to update the state estimate. At the core of this procedure is the recursive-least-squares update of the observation model. Indeed, the solution of this state-space estimation problem amounts to solving the following least squares cost function [19]:

$$\begin{aligned} \min_{\{\mathbf{x}(1),\mathbf{n}(1),\cdots,\mathbf{n}(i)\}} &\left[\sum_{j=1}^{i} \beta^{i-j} \left| d(j) - \mathbf{u}(j)^T\mathbf{x}(j) \right|^2 \right. \\ &\left. + \lambda\beta^i \|\mathbf{x}(1)\|^2 + q^{-1}\sum_{j=1}^{i} \beta^{i-j} \|\mathbf{n}(j)\|^2 \right], \\ \text{subject to} \quad &\mathbf{x}(j+1) = \mathbf{A}\mathbf{x}(j) + \mathbf{n}(j) \end{aligned} \tag{23}$$

where β is the exponential weighting factor on the past data, λ is the regularization parameter to control the initial state-vector norm and q provides trade-off between the modeling variation and measurement disturbance. We must emphasize that (23) is a much harder quadratic optimization problem with linear constraints compared to the constraint-free least squares problem (2) which is the basis of both the RLS and the KRLS. When the input data are transformed into a high dimensional feature space via a kernel mapping, this problem gets just much harder, as we will discuss in the following section.

The extended RLS recursion is quoted in Algorithm 3 [19] for easy comparison.

Algorithm 3: Extended Recursive Least Squares (EX-RLS)

```
Initialize $\mathbf{w}(0) = 0$, $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$
Iterate for $i \geq 1$
    $r_e(i) = \beta^i + \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)$
    $\mathbf{k}_p(i) = \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r_e(i)$
    $e(i) = d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)$
    $\mathbf{w}(i) = \mathbf{A}\mathbf{w}(i-1) + \mathbf{k}_p(i)e(i)$
    $\mathbf{P}(i) = \mathbf{A}\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)/r_e(i)\right] \times \mathbf{A}^T + \beta^i q\mathbf{I}$    (24)
```

III. Extended Kernel Recursive Least Squares

To derive a kernelized EX-RLS, the input is transformed into the feature space $\mathbb{F}$ as in KRLS. The state-transition model and observation model become

$$\begin{aligned}\mathbf{x}(i+1) &= \mathbf{A}\mathbf{x}(i) + \mathbf{n}(i)\\ d(i) &= \boldsymbol{\varphi}(i)^T\mathbf{x}(i) + v(i)\end{aligned} \tag{25}$$

which is similar to (22) except that the input is $\boldsymbol{\varphi}(i) \in \mathbb{F}$ instead of $\mathbf{u}(i) \in \mathbb{U}$. Notice that $\mathbf{x}(i) \in \mathbb{F}$ now lies in a possibly infinite dimensional feature space and $\mathbf{A} : \mathbb{F} \to \mathbb{F}$ becomes an operator, rigorously speaking.

The first question is: how expressive can this model be? The following theorem, which answers the question, is one of the main contributions of the paper.

Theorem 1: Assume a general nonlinear state-space model

$$\begin{aligned}\mathbf{s}(i+1) &= g(\mathbf{s}(i))\\ d(i) &= h(\mathbf{u}(i), \mathbf{s}(i)) + v(i)\end{aligned} \tag{26}$$

where $\mathbf{s} \in \mathbb{S}$ is the original state vector, $g : \mathbb{S} \to \mathbb{S}$ and $h : \mathbb{U} \times \mathbb{S} \to \mathbb{R}$ are general continuous nonlinear functions. There exists a transformed input vector $\boldsymbol{\varphi}(\mathbf{u})$, a transformed state vector $\mathbf{x}(\mathbf{s})$ and a linear operator $\mathbf{A}$, such that the following equations hold:

$$\begin{aligned}\mathbf{x}(\mathbf{s}(i+1)) &= \mathbf{A}\mathbf{x}(\mathbf{s}(i))\\ d(i) &= \boldsymbol{\varphi}(\mathbf{u}(i))^T\mathbf{x}(\mathbf{s}(i)) + v(i)\end{aligned} \tag{27}$$

and

$$\boldsymbol{\varphi}(\mathbf{u})^T\boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}') \tag{28}$$

with κ a suitable kernel function.

Proof: It is equivalent to prove that there exist $\boldsymbol{\varphi}(\mathbf{u})$, $\mathbf{x}(\mathbf{s})$ and $\mathbf{A}$ such that

$$\begin{aligned}h(\mathbf{u}, \mathbf{s}) &= \boldsymbol{\varphi}(\mathbf{u})^T\mathbf{x}(\mathbf{s})\\ \mathbf{x}(g(\mathbf{s})) &= \mathbf{A}\mathbf{x}(\mathbf{s}).\end{aligned} \tag{29}$$

Assume the Gaussian kernel $\kappa : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$ is chosen here. To simplify the demonstration, we assume the dimensionality of the induced RKHS is J, which is ∞ in the case of the Gaussian kernel. And construct the transformed input as

$$\boldsymbol{\varphi}(\mathbf{u}) = [\sqrt{\varsigma_1}\phi_1(\mathbf{u}), \sqrt{\varsigma_2}\phi_2(\mathbf{u}), \ldots, \sqrt{\varsigma_J}\phi_J(\mathbf{u}), 0, 0, \ldots]^T$$

where ς_j and ϕ_j are defined in (6). 0's are appended to make the total dimension of the transformed vector $2J$.

First of all, by Mercer theorem, we have $\boldsymbol{\varphi}(\mathbf{u})^T\boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}')$. Then it is proved by [23] that the RKHS induced by the Gaussian kernel has the universal approximation capability, that is, by treating $\mathbf{s}$ as the parameter and $h(\mathbf{u}, \mathbf{s})$ as a function of $\mathbf{u}$, we can represent it as a linear combination of $\{\sqrt{\varsigma_j}\phi_j(\mathbf{u})\}_{j=1}^J$, which form a basis of the function space

$$h(\mathbf{u}, \mathbf{s}) = \sum_{j=1}^{J} \sqrt{\varsigma_j}\phi_j(\mathbf{u})\mathbf{x}_j(\mathbf{s})$$

where $\mathbf{x}_j$ is a scalar function of $\mathbf{s}$.

Now we can construct the transformed state vector as

$$\mathbf{x}(\mathbf{s}) = [\mathbf{x}_1(\mathbf{s}), \mathbf{x}_2(\mathbf{s}), \ldots, \mathbf{x}_J(\mathbf{s}), \phi_1(\mathbf{s}), \phi_2(\mathbf{s}), \ldots, \phi_J(\mathbf{s})]^T.$$

Note that we append 0's at the end of $\boldsymbol{\varphi}(\mathbf{u})$ to make it has the same dimension as $\mathbf{x}$ and more importantly

$$\boldsymbol{\varphi}(\mathbf{u})^T\mathbf{x}(\mathbf{s}) = h(\mathbf{u}, \mathbf{s}).$$

Apparently, $\{\phi_j(\mathbf{s})\}_{j=1}^J$ form a basis for another RKHS (with the function domain defined on $\mathbb{S}$) and by the same argument, the composition functions $\mathbf{x}_j \circ g$ and $\phi_j \circ g$ (which are all functions defined from $\mathbb{S}$ to $\mathbb{R}$) can be written as linear combinations of the basis. Putting this fact into matrix form, we have a $2J \times 2J$ matrix $\mathbf{A}$ (a linear operator) such that

$$\mathbf{x}(g(\mathbf{s})) = \mathbf{A}\mathbf{x}(\mathbf{s}).$$

Therefore the existence is proved by construction. ■

Notice that we purposely omit the state noise in (26) and (27), since the noise term makes the proof very complicated without providing more insights. In general, the existence of state noise requires linear approximation around operating points and the approximation error can be modeled as part of $\mathbf{n}(i)$. For example, if $\mathbf{s}(i+1) = g(\mathbf{s}(i)) + \mathbf{n}_\mathbf{s}(i)$, we can define $\mathbf{n}(i) = \mathbf{x}(g(\mathbf{s}(i)) + \mathbf{n}_\mathbf{s}(i)) - \mathbf{x}(g(\mathbf{s}(i)))$ such that the same proof can be used. By so doing, $\mathbf{n}(i) \to 0$ as $\mathbf{n}_\mathbf{s}(i) \to 0$. And $\mathbf{n}(i)$ is now correlated to the state $\mathbf{x}(i)$.

This theorem effectively shows for any nonlinear state-space model given by (26), there exists an equivalent representation of a linear state-space model in RKHS (27). This result is of great significance, providing a theoretic framework to tackle general nonlinear state space model estimation problems. With this in mind, a natural question is how to solve the general state model (25) noting that many variables are in very high dimensional spaces. And the answer to this question is another main contribution of the paper.

By slightly abusing notation, we view operators as infinite dimensional matrices to treat the recursions in the input space and feature space consistently. Therefore, the recursion to estimate $\mathbf{x}(i)$ in the RKHS would look like:

Start with $\boldsymbol{\omega}(0) = 0, \mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, and iterate for $i \geq 1$,

$$\begin{aligned} r_e(i) &= \beta^i + \boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)\boldsymbol{\varphi}(i) \\ \mathbf{k}_p(i) &= \mathbf{A}\mathbf{P}(i-1)\boldsymbol{\varphi}(i)/r_e(i) \\ e(i) &= d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1) \\ \boldsymbol{\omega}(i) &= \mathbf{A}\boldsymbol{\omega}(i-1) + \mathbf{k}_p(i)e(i) \\ \mathbf{P}(i) &= \mathbf{A}\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)/r_e(i)\right] \\ &\quad \times \mathbf{A}^T + \beta^i q\mathbf{I}. \end{aligned} \tag{30}$$

However, this approach is not feasible in practice because the transformed data $\boldsymbol{\varphi}(i)$ and the state vector $\mathbf{x}(i)$ now lie in a possibly infinite dimensional space $\mathbf{F}$ (consequently the $\mathbf{P}$ matrix could be $\infty \times \infty$ and $\boldsymbol{\omega}$ $\infty \times 1$). We cannot use the matrix inversion lemma as in the KRLS because of the complicated constrained least squares cost function (23). The utilization of the kernel trick requires a reformulation of the recursion solely in terms of inner product operations between transformed input vectors.

While it would be most desirable to have such a general state-space model in the RKHS, it turns out to be at best very difficult. In this work we focus on two special cases of the state transition operator $\mathbf{A}$ in the following two sections.

IV. Extended Kernel Recursive Least Squares for Tracking Models

For the first special case, we assume the state transition operator has the form $\mathbf{A} = \alpha\mathbf{I}$, then we have the following tracking model:

$$\begin{aligned} \mathbf{x}(i+1) &= \alpha\mathbf{x}(i) + \mathbf{n}(i) \\ d(i) &= \boldsymbol{\varphi}(i)^T\mathbf{x}(i) + v(i) \end{aligned} \tag{31}$$

where α is a scaling factor. α indicates a uni-directional change in state, either attenuating or amplifying. If the change is very slow over time, α is very close to 1. For example, this model is particularly suitable to track slow fading channels in communications and it provides a theoretical foundation of the exponentially weighted kernel recursive least squares [19]. Furthermore, introducing the state noise is also very important, since it provides a framework to study random walk state-space model in RKHS, which is here studied for the first time in the literature to the best knowledge of the authors.

Theoretically, (31) can be used to solve the following problem:

$$\begin{aligned} \mathbf{s}(i+1) &= \tilde{\alpha}\mathbf{s}(i) + \mathbf{n}(i) \\ d(i) &= h\left(\mathbf{u}(i), \mathbf{s}(i)\right) + v(i) \end{aligned} \tag{32}$$

for any nonlinear function h and with $\tilde{\alpha}$ very close to 1. Since $\tilde{\alpha}$ is very close to 1 and the nonlinear mapping between $\mathbf{s}$ and $\mathbf{x}$ is continuous, a single scalar α is sufficient to model this slow change. The recursion (30) becomes:

Start with $\boldsymbol{\omega}(0) = 0, \mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, and iterate for $i \geq 1$,

$$\begin{aligned} r_e(i) &= \beta^i + \boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)\boldsymbol{\varphi}(i) \\ \mathbf{k}_p(i) &= \alpha\mathbf{P}(i-1)\boldsymbol{\varphi}(i)/r_e(i) \\ e(i) &= d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1) \\ \boldsymbol{\omega}(i) &= \alpha\boldsymbol{\omega}(i-1) + \mathbf{k}_p(i)e(i) \\ \mathbf{P}(i) &= |\alpha|^2\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)/r_e(i)\right] \\ &\quad + \beta^i q\mathbf{I}. \end{aligned} \tag{33}$$

By carefully observing the recursion (33), one can conclude that the expression $\boldsymbol{\varphi}(j)^T\mathbf{P}(k)\boldsymbol{\varphi}(i)$ (for any k, i, j), which can be $(1 \times \infty) \times (\infty \times \infty) \times (\infty \times 1)$, plays an important role in the recursion. Making this expression computable in the input space is crucial and is demonstrated by the following theorem.

Theorem 2: The matrices $\mathbf{P}(j)$ in (33) are of the following form:

$$\mathbf{P}(j) = \rho(j)\mathbf{I} - \mathbf{H}(j)\mathbf{Q}(j)\mathbf{H}(j)^T \tag{34}$$

where $\rho(j)$ is a scalar and $\mathbf{Q}(j)$ is a $j \times j$ real-valued matrix with $\mathbf{H}(j) = [\boldsymbol{\varphi}(1), \cdots, \boldsymbol{\varphi}(j)]$, for all $j > 0$.

Proof: First notice that by (33)

$$\begin{aligned} \mathbf{P}(0) &= \lambda^{-1}\mathbf{I}, \\ r_e(1) &= \beta + \lambda^{-1}\kappa\left(\mathbf{u}(1), \mathbf{u}(1)\right), \\ \mathbf{P}(1) &= \left[\frac{|\alpha|^2}{\lambda} + \beta q\right]\mathbf{I} - \boldsymbol{\varphi}(1)\left[\frac{|\alpha|^2\lambda^{-2}}{\beta + \lambda^{-1}\kappa\left(\mathbf{u}(1), \mathbf{u}(1)\right)}\right]\boldsymbol{\varphi}(1)^T \end{aligned} \tag{35}$$

so the claim is valid for $j = 1$, namely,

$$\begin{aligned} \rho(1) &= |\alpha|^2\lambda^{-1} + \beta q, \\ \mathbf{Q}(1) &= \frac{|\alpha|^2\lambda^{-2}}{\beta + \lambda^{-1}\kappa\left(\mathbf{u}(1), \mathbf{u}(1)\right)}. \end{aligned} \tag{36}$$

Then by induction, the proof for all j follows. Assume it is true for $j = i - 1$,

$$\mathbf{P}(i-1) = \rho(i-1)\mathbf{I} - \mathbf{H}(i-1)\mathbf{Q}(i-1)\mathbf{H}(i-1)^T. \tag{37}$$

By substituting into the last equation of (33), one has

$$\begin{aligned} \mathbf{P}(i) &= |\alpha|^2\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)r_e^{-1}(i)\right] \\ &\quad + \beta^i q\mathbf{I} \\ &= \left(|\alpha|^2\rho(i-1) + \beta^i q\right)\mathbf{I} \\ &\quad - |\alpha|^2 r_e^{-1}(i)\mathbf{H}(i) \\ &\quad \times \begin{bmatrix} \mathbf{Q}(i-1)r_e(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\rho(i-1)\mathbf{z}(i) \\ -\rho(i-1)\mathbf{z}(i)^T & \rho^2(i-1) \end{bmatrix}\mathbf{H}(i)^T \end{aligned} \tag{38}$$

where $\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{H}(i-1)^T\boldsymbol{\varphi}(i)$. Notice that $r_e(i)$ is computable by $r_e(i) = \beta^i + \boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)\boldsymbol{\varphi}(i)$. Therefore,

$$\begin{aligned} \rho(i) &= |\alpha|^2\rho(i-1) + \beta^i q \\ \mathbf{Q}(i) &= \frac{|\alpha|^2}{r_e(i)}\begin{bmatrix} \mathbf{Q}(i-1)r_e(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\rho(i-1)\mathbf{z}(i) \\ -\rho(i-1)\mathbf{z}(i)^T & \rho^2(i-1) \end{bmatrix} \\ \mathbf{P}(i) &= \rho(i)\mathbf{I} - \mathbf{H}(i)\mathbf{Q}(i)\mathbf{H}(i)^T \end{aligned} \tag{39}$$

■

By Theorem 2, the calculation $\boldsymbol{\varphi}(j)^T\mathbf{P}(k)\boldsymbol{\varphi}(i)$ only involves inner product operations between transformed input vectors, and we can use the kernel trick readily. Furthermore, we can express the weight vector (which lies in a high dimensional feature space $\mathbb{F}$) as a linear combination of the previous transformed input vectors as shown in the next theorem.

Theorem 3: The optimal state estimate in (33) is a linear combination of the past transformed input vectors

$$\boldsymbol{\omega}(j) = \mathbf{H}(j)\mathbf{a}(j) \tag{40}$$

where $\mathbf{a}(j)$ is a $j \times 1$ real-valued vector with $\mathbf{H}(j) = [\boldsymbol{\varphi}(1), \cdots, \boldsymbol{\varphi}(j)]$, for all $j > 0$.

Proof: Notice that by (33)

$$\begin{aligned}\boldsymbol{\omega}(0) &= 0,\\ \boldsymbol{\omega}(1) &= \frac{\alpha\lambda^{-1}d(1)\boldsymbol{\varphi}(1)}{\beta + \lambda^{-1}\kappa(\mathbf{u}(1),\mathbf{u}(1))}\end{aligned} \tag{41}$$

so

$$\mathbf{a}(1) = \frac{\alpha\lambda^{-1}d(1)}{\beta + \lambda^{-1}\kappa(\mathbf{u}(1),\mathbf{u}(1))}. \tag{42}$$

Thus the claim is valid for $j = 1$. Then we use induction to prove it is valid for any j. Assume it is true for $i-1$. By the recursion (33) and the result from Theorem 2, we have

$$\begin{aligned}\boldsymbol{\omega}(i) &= \alpha\boldsymbol{\omega}(i-1) + k_p(i)e(i)\\ &= \alpha\mathbf{H}(i-1)\mathbf{a}(i-1) + \alpha\mathbf{P}(i-1)\boldsymbol{\varphi}(i)e(i)/r_e(i)\\ &= \alpha\mathbf{H}(i-1)\mathbf{a}(i-1) + \alpha\rho(i-1)\boldsymbol{\varphi}(i)e(i)/r_e(i)\\ &\quad - \alpha\mathbf{H}(i-1)\mathbf{z}(i)e(i)/r_e(i)\\ &= \mathbf{H}(i)\begin{bmatrix}\alpha\mathbf{a}(i-1) - \alpha\mathbf{z}(i)e(i)r_e^{-1}(i)\\ \alpha\rho(i-1)e(i)r_e^{-1}(i)\end{bmatrix}\end{aligned} \tag{43}$$

where $\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{H}(i-1)^T\boldsymbol{\varphi}(i)$ as before. This completes the proof. ∎

Hence, after representing $\boldsymbol{\omega}$ with $\mathbf{a}$, and $\mathbf{P}$ with ρ and $\mathbf{Q}$, we have the following equivalent recursion, which is computable in the input space:

Start with

$$\begin{aligned}\mathbf{a}(1) &= \frac{\alpha d(1)}{\beta\lambda + \kappa(\mathbf{u}(1),\mathbf{u}(1))},\\ \rho(1) &= |\alpha|^2\lambda^{-1} + \beta q, \qquad .\\ \mathbf{Q}'(1) &= \frac{|\alpha|^2\lambda^{-1}}{\beta\lambda + \kappa(\mathbf{u}(1),\mathbf{u}(1))}\end{aligned}$$

Iterate for $i > 1$:

$$\begin{aligned}\mathbf{h}(i) &= \mathbf{H}(i-1)^T\boldsymbol{\varphi}(i)\\ \mathbf{z}'(i) &= \mathbf{Q}'(i-1)\mathbf{h}(i)\\ r_e(i) &= \beta^i + \rho(i-1)\kappa(\mathbf{u}(i),\mathbf{u}(i)) - \mathbf{h}(i)^T\mathbf{z}'(i)\\ e(i) &= d(i) - \mathbf{h}(i)^T\mathbf{a}(i-1)\\ \mathbf{a}(i) &= \alpha\begin{bmatrix}\mathbf{a}(i-1) - \mathbf{z}'(i)r_e^{-1}(i)e(i)\\ \rho(i-1)r_e^{-1}(i)e(i)\end{bmatrix}\\ \rho(i) &= |\alpha|^2\rho(i-1) + \beta^i q\\ \mathbf{Q}'(i) &= \frac{|\alpha|^2}{r_e(i)}\begin{bmatrix}\mathbf{Q}'(i-1)r_e(i) + \mathbf{z}'(i)\mathbf{z}'(i)^T & -\rho(i-1)\mathbf{z}'(i)\\ -\rho(i-1)\mathbf{z}'(i)^T & \rho^2(i-1)\end{bmatrix}.\end{aligned} \tag{44}$$

This is quite sufficient for our purpose but when λ is very small, ρ, $\mathbf{Q}'$, $\mathbf{z}'$ and r_e are all very large, causing possible numerical issues. And also to better understand the meanings of these quantities, we do the following change of variables:

$$\begin{aligned}\mathbf{Q}(i-1) &= \mathbf{Q}'(i-1)/\rho(i-1)\\ \mathbf{z}(i) &= \mathbf{z}'(i)/\rho(i-1)\\ r(i) &= r_e(i)/\rho(i-1)\\ \rho_r(i) &= 1/\rho(i-1).\end{aligned}$$

Therefore, we have Algorithm 4.

Algorithm 4: Extended Kernel Recursive Least Squares for the Tracking Model (31)

Initialize

$$\begin{aligned}\mathbf{a}(1) &= \frac{\alpha d(1)}{\lambda\beta + \kappa(\mathbf{u}(1),\mathbf{u}(1))},\\ \rho_r(1) &= \lambda\beta/\left(|\alpha|^2\beta + \lambda q\right),\\ \mathbf{Q}(1) &= \frac{|\alpha|^2}{[\beta\lambda + \kappa(\mathbf{u}(1),\mathbf{u}(1))]\,[|\alpha|^2 + \beta\lambda q]}\end{aligned}$$

Iterate for $i > 1$:

$$\begin{aligned}\mathbf{h}(i) &= [\kappa(\mathbf{u}(i),\mathbf{u}(1)), \cdots, \kappa(\mathbf{u}(i),\mathbf{u}(i-1))]^T\\ \mathbf{z}(i) &= \mathbf{Q}(i-1)\mathbf{h}(i)\\ r(i) &= \beta^i\rho_r(i-1) + \kappa(\mathbf{u}(i),\mathbf{u}(i)) - \mathbf{h}(i)^T\mathbf{z}(i)\\ e(i) &= d(i) - \mathbf{h}(i)^T\mathbf{a}(i-1)\\ \mathbf{a}(i) &= \alpha\begin{bmatrix}\mathbf{a}(i-1) - \mathbf{z}(i)r^{-1}(i)e(i)\\ r^{-1}(i)e(i)\end{bmatrix}\\ \rho_r(i) &= \rho_r(i-1)/\left(|\alpha|^2 + \beta^i q\rho_r(i-1)\right)\\ \mathbf{Q}(i) &= \frac{|\alpha|^2}{r(i)\left(|\alpha|^2 + \beta^i q\rho_r(i-1)\right)}\\ &\quad\times\begin{bmatrix}\mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i)\\ -\mathbf{z}(i)^T & 1\end{bmatrix}\end{aligned} \tag{45}$$

Notice that throughout the iteration, the input vector $\boldsymbol{\varphi}(i)$ only enters in the calculation of $\mathbf{h}(i)$ and $r(i)$, both in the form of an inner product. The significance of this reformulation is its independence on the data dimensionality. Compared to (33), we replaced the recursion on $\boldsymbol{\omega}(i)$ with the one on $\mathbf{a}(i)$, which is $i \times 1$. Furthermore, we replaced the recursion on $\mathbf{P}(i)$ with the ones on $\rho(i)$ and $\mathbf{Q}(i)$ where $\rho(i)$ is a scalar and $\mathbf{Q}(i)$ is $i \times i$. To summarize, the dimensionality of $\mathbf{a}(i)$, $\rho(i)$ and $\mathbf{Q}(i)$ only depends on the number of training data at time i regardless of the dimensionality of the transformed data $\boldsymbol{\varphi}$.

The learning procedure of EX-KRLS is similar to KRLS as a growing RBF network: it allocates a new unit with $\mathbf{u}(i)$ as the center and $r^{-1}(i)e(i)$ as the coefficient; at the same time all the previous coefficients are updated by a quantity $-\mathbf{z}(i)r^{-1}(i)e(i)$. Finally a scalar α is multiplied on $\mathbf{a}(i)$ according to the state update equation. The major difference is the introduction of ρ, which reflects the uncertainty of the state noise. Moreover, we have the following two special cases.

By setting $\alpha = 1$, we have KRLS for the following random walk model

$$\begin{aligned}\mathbf{x}(i+1) &= \mathbf{x}(i) + \mathbf{n}(i)\\ d(i) &= \boldsymbol{\varphi}(i)^T\mathbf{x}(i) + v(i).\end{aligned} \tag{46}$$

By setting $q = 0$ and $\alpha = 1$, we have the exponentially weighted KRLS. Remember that q is a parameter which is proportional to the variance of the state noise, so setting $q = 0$ essentially excludes the state noise. The derivation is also straightforward by Algorithm 4. However $\rho(i)$ becomes a constant and is no longer needed. After absorbing a factor of β into λ, it is almost the same as Algorithm 2 (KRLS) except that the regularization parameter λ is exponentially weighted by β^{i-1} in $r(i)$. Setting $\beta = 1$, we have the KRLS back.

To summarize, all the algorithms discussed in this section can be regarded as nontrivial extensions of KRLS, with an explicit state transition process. They are very suitable to model nonlinear systems with a slow fading and a small variation in state, improving the tracking ability of KRLS.

A. *Sparsification and Approximate Linear Dependency*

The implementation of these algorithms is straightforward. We need to store $\mathbf{a}(i)$, $\rho(i)$ and $\mathbf{Q}(i)$, so the time and space complexities are both $O(i^2)$ at iteration i. The network size increases linearly with the number of training data, which may pose a big problem for the application of these algorithms. To curb the growing structure, several methods have been proposed including the novelty criterion [24] and approximate linear dependency (ALD) [8]. Sparseness is usually achieved by the creation of a basis dictionary and its corresponding coefficients. Suppose the present dictionary is $\mathcal{C}(i) = \{\mathbf{c}_j\}_{j=1}^{m_i}$ where $\mathbf{c}_j$ is the jth center and m_i is the cardinality. When a new data pair $\{\mathbf{u}(i+1), d(i+1)\}$ is presented, a decision is made immediately whether $\mathbf{u}(i+1)$ should be added into the dictionary as a center.

The novelty criterion introduced by Platt is relatively simple: First, it calculates the distance of $\mathbf{u}(i+1)$ to the present dictionary $\mathrm{dis}_1 = \min_{\mathbf{c}_j \in \mathcal{C}(i)} \|\mathbf{u}(i+1) - \mathbf{c}_j\|$. If it is smaller than some preset threshold, say δ_1, $\mathbf{u}(i+1)$ is not added into the dictionary. Otherwise, it computes the prediction error $e(i+1)$. Only if the prediction error is larger than another preset threshold, say δ_2, $\mathbf{u}(i+1)$ is accepted as a new center.

The approximate linear dependency test introduced in [8] is more computationally involved. When a new data pair $\{\mathbf{u}(i+1), d(i+1)\}$ is presented, it tests the following cost:

$$\mathrm{dis}_2 = \min_{\forall \mathbf{b}} \left\| \boldsymbol{\varphi}(\mathbf{u}(i+1)) - \sum_{\mathbf{c}_j \in \mathcal{C}(i)} \mathbf{b}_j \boldsymbol{\varphi}(\mathbf{c}_j) \right\|$$

which indicates the distance of the new input to the linear span of the present dictionary in the feature space.

By straightforward calculus, it turns out that

$$\mathbf{G}(i)\mathbf{b} = \mathbf{h}(i+1) \tag{47}$$

where $\mathbf{G}(i)$ and $\mathbf{h}(i+1)$ are redefined based on $\mathcal{C}(i)$ as

$$\mathbf{h}(i+1) = [\kappa(\mathbf{u}(i+1), \mathbf{c}_1), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{c}_{m_i})]^T \tag{48}$$

$$\mathbf{G}(i) = \begin{bmatrix} \kappa(\mathbf{c}_1, \mathbf{c}_1) & \cdots & \kappa(\mathbf{c}_{m_i}, \mathbf{c}_1) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{c}_1, \mathbf{c}_{m_i}) & \cdots & \kappa(\mathbf{c}_{m_i}, \mathbf{c}_{m_i}) \end{bmatrix}. \tag{49}$$

If $\mathbf{G}(i)$ is invertible, one has

$$\mathbf{b} = \mathbf{G}(i)^{-1}\mathbf{h}(i+1) \tag{50}$$

$$\mathrm{dis}_2^2 = \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T \mathbf{G}^{-1}(i)\mathbf{h}(i+1). \tag{51}$$

$\mathbf{u}(i+1)$ will be rejected if dis_2 is smaller than some preset threshold δ_3 in ALD. The computational complexity is $O(m_i^2)$.

On the other hand, by (15), one has

$$r(i+1) = \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T (\mathbf{G}(i) + \lambda \mathbf{I})^{-1} \mathbf{h}(i+1) \tag{52}$$

in KRLS (20). Note that when λ is small, $r(i+1)$ is essentially equivalent to dis_2. On the other hand, in the Gaussian processes theory, $r(i+1)$ denotes the prediction variance and data selection based on it is well documented [25]. In other words, $r(i+1)$ can be used to approximate ALD and has strong basis to be used as a data selection criterion.

Furthermore, in EX-KRLS, $r(i)$ in (45) plays a very similar role to $r(i)$ in KRLS. Although its meaning is not as clear as in KRLS, $r(i)$ is at least a very good approximation of the distance especially when α, β are close to 1 and q is very small, which is usually valid in practice. Therefore, ALD is readily applicable for EX-KRLS without extra computation. Of course due to the long-term effect of the state transition model, EX-KRLS can deviate significantly from KRLS when the iteration goes on for a long time, so precaution is needed here. The worst case is to compute dis_2 each iteration which also scales with $O(m_i^2)$.

It is also interesting to note that the nearest distance used in the novelty criterion is an approximation of ALD, which corresponds to a special choice of $\mathbf{b}$ (1 for the closest center and 0 for all the others). When a local kernel function (like Gaussian) is used, picking the closest one is a good strategy to estimate dis_2. If the new example is determined to be not "novel," it is simply discarded in this paper, but different strategies can be employed to utilize the information like in [8] and [24].

V. Extended Kernel Recursive Least Squares With Finite Rank Assumption

As one can expect from the previous discussion, it is difficult to have the recursion computable with a general state transition operator $\mathbf{A}$. Here, we propose a subspace technique to make the recursion feasible. In this section, the state transition operator is assumed to be

$$\mathbf{A} = \sum_{m=1}^{M} \sum_{j=1}^{J} b_{m,j} \boldsymbol{\varphi}(\mathbf{c}_m) \boldsymbol{\varphi}(\mathbf{e}_j)^T. \tag{53}$$

The mathematical meaning of this assumption can be interpreted as follows: Define $\mathrm{nul}(\mathbf{A}) = \{x \in \mathbb{F} : \mathbf{A}x = 0\}$ as the null space of $\mathbf{A}$ and $\mathrm{im}(\mathbf{A}) = \{y \in \mathbb{F} : y = \mathbf{A}x, x \in \mathbb{F}\}$ the image or range of $\mathbf{A}$. Then $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^{M}$ is a set of basis of $\mathrm{im}(\mathbf{A})$ and $\{\boldsymbol{\varphi}(\mathbf{e}_j)\}_{j=1}^{J}$ is a set of basis of the orthogonal complement of $\mathrm{nul}(\mathbf{A})$. The dimensionality of the two subspaces, $\mathrm{im}(\mathbf{A})$ and $\mathrm{nul}(\mathbf{A})^{\perp}$ are assumed to be finite, i.e., $M < \infty$ and $J < \infty$. Actually by the rank-nullity theorem [26], $M = J$. And M is also called the rank of $\mathbf{A}$, thus the name of the assumption. $b_{m,j}$ are a set of real-valued coefficients. For any state vector $\mathbf{x}$, the operator $\mathbf{A}$ transforms $\mathbf{x}$ as a linear combination of $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^M$ with $\sum_{j=1}^J b_{m,j}\boldsymbol{\varphi}(\mathbf{e}_j)^T\mathbf{x}$ as the coefficients:

$$\mathbf{A}\mathbf{x} = \sum_{m=1}^{M} \boldsymbol{\varphi}(\mathbf{c}_m)\left(\sum_{j=1}^{J} b_{m,j}\boldsymbol{\varphi}(\mathbf{e}_j)^T\mathbf{x}\right). \tag{54}$$

Denoting $\mathbf{C} = [\boldsymbol{\varphi}(\mathbf{c}_1),\cdots,\boldsymbol{\varphi}(\mathbf{c}_M)]$, $\mathbf{E} = [\boldsymbol{\varphi}(\mathbf{e}_1),\cdots,\boldsymbol{\varphi}(\mathbf{e}_J)]$ and $\mathbf{B} = [b_{m,j}]_{M\times J}$, we have

$$\mathbf{A} = \mathbf{C}\mathbf{B}\mathbf{E}^T.$$

By this assumption, we can prove the following two theorems like in the previous section.

Theorem 4: By (53), the matrices $\mathbf{P}(j)$ in (30) assume the following form:

$$\mathbf{P}(j) = \rho(j)\mathbf{I} - \mathbf{C}\mathbf{Q}(j)\mathbf{C}^T \tag{55}$$

where $\rho(j)$ is a scalar and $\mathbf{Q}(j)$ is an $M\times M$ real-valued matrix, for all $j > 0$.

Theorem 5: By (53), the optimal state estimate in (30) can be expressed as

$$\boldsymbol{\omega}(j) = \mathbf{C}\mathbf{a}(j) \tag{56}$$

where $\mathbf{a}(j)$ is an $M \times 1$ real-valued vector, for all $j > 0$.

The proofs are very similar to the previous theorems and are included in the Appendixes. By representing $\boldsymbol{\omega}$ with $\mathbf{a}$, $\mathbf{P}$ with $\mathbf{Q}$ and denoting $\mathbf{G_{EC}} = \mathbf{E}^T\mathbf{C}$, $\mathbf{G_E} = \mathbf{E}^T\mathbf{E}$, $\mathbf{h_C}(i) = \mathbf{C}^T\boldsymbol{\varphi}(i)$, and $\mathbf{h_E}(j) = \mathbf{E}^T\boldsymbol{\varphi}(j)$, we have Algorithm 5.

Algorithm 5: Extended Kernel Recursive Least Squares With The Subspace Assumption (53)

```
Initialize
    $\mathbf{a}(1) = \dfrac{\mathbf{B}\mathbf{h_E}(1)d(1)}{\lambda\beta + \kappa(\mathbf{u}(1),\mathbf{u}(1))}$,
    $\mathbf{Q}(1) = -\dfrac{\mathbf{B}\mathbf{G_E}\mathbf{B}^T}{\lambda} + \dfrac{\mathbf{B}\mathbf{h_E}(1)\mathbf{h_E}(1)^T\mathbf{B}^T}{\lambda^2\beta + \lambda\kappa(\mathbf{u}(1),\mathbf{u}(1))}$
Iterate for $i > 1$:
    $r_e(i) = \beta^i + \beta^{i-1}q\kappa(\mathbf{u}(i),\mathbf{u}(i)) - \mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{h_C}(i)$
    $e(i) = d(i) - \mathbf{h_C}(i)^T\mathbf{a}(i-1)$
    $\mathbf{a}(i) = \mathbf{B}\mathbf{G_{EC}}\mathbf{a}(i-1)$
        $+ \mathbf{B}\left[\beta^{i-1}q\mathbf{h_E}(i) - \mathbf{G_{EC}}\mathbf{Q}(i-1)\mathbf{h_C}(i)\right] e(i)/r_e(i)$
    $\mathbf{Q}(i) = -\beta^{i-1}q\mathbf{B}\mathbf{G_E}\mathbf{B}^T - \mathbf{B}\mathbf{G_{EC}}\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T$
        $+ \mathbf{B}\mathbf{h_E}(i)\mathbf{h_E}(i)^T\mathbf{B}^T[\beta^{i-1}q]^2/r_e(i)$
        $+ \mathbf{B}\mathbf{h_E}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\beta^{i-1}q/r_e(i)$
        $+ \mathbf{B}\mathbf{G_{EC}}\mathbf{Q}(i-1)\mathbf{h_C}(i)\mathbf{h_E}(i)^T\mathbf{B}^T\beta^{i-1}q/r_e(i)$
        $+ \mathbf{B}\mathbf{G_{EC}}\mathbf{Q}(i-1)\mathbf{h_C}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)$
        $\times \mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T/r_e(i)$
```

Note that throughout the iteration, $\mathbf{a}(i)$ is an $M\times 1$ vector and $\mathbf{Q}(i)$ is an $M \times M$ matrix. So again the dimension of $\mathbf{a}(i)$ and $\mathbf{Q}(i)$ only depends on the rank of $\mathbf{A}$ regardless of the dimension of $\boldsymbol{\varphi}$. This algorithm can be regarded as a fixed-topology RBF network (comparing with the growing structure of Algorithm 4), and the linear coefficients $\mathbf{a}(i)$ is constantly updated based on the state transition process and the input-output observations. The time and space complexities are both $O(M^2)$ per iteration.

A. Subspace Approximation for the State Noise

If the state noise $\mathbf{n}(i) \in \mathbb{F}$ is further assumed within the span of $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^M$, i.e.

$$\mathbf{n}(i) = \mathbf{C}\tilde{\mathbf{n}}(i),$$

it can be proved that the state $\mathbf{x}(i) \in \mathbb{F}$ also lies in this subspace for $i > 0$. Therefore

$$\mathbf{x}(i) = \mathbf{C}\tilde{\mathbf{x}}(i).$$

Note that $\tilde{\mathbf{n}}(i)$ and $\tilde{\mathbf{x}}(i)$ are M dimensional real-valued vectors.

With all these assumptions, model (25) reduces to

$$\begin{aligned}\tilde{\mathbf{x}}(i+1) &= \mathbf{B}\mathbf{G_{EC}}\tilde{\mathbf{x}}(i) + \tilde{\mathbf{n}}(i)\\ d(i) &= \mathbf{h_C}(i)^T\tilde{\mathbf{x}}(i) + v(i)\end{aligned} \tag{57}$$

and Algorithm 5 reduces to a classical (linear) Kalman recursion on state $\tilde{\mathbf{x}}(i)$, with $\{\mathbf{h_C}(i), d(i)\}$ as the input-output observations. Though this result is not surprising (see [18], [27], and [28]) with the subspace technique, it is for the first time presented in the extended recursive least squares framework and it provides much understanding about the state transition processes in the RKHS. The algorithm itself is quite straightforward and the difficulty is really how to learn the parameters in $\mathbf{A}$. This paper focuses on algorithm derivation and the model learning problem deserves a separate treatment.

VI. Simulations

A. Rayleigh Channel Tracking

We consider the problem of tracking a nonlinear Rayleigh fading multipath channel and compare the performance of the proposed EX-KRLS algorithm to the original KRLS. Also performance of the normalized LMS, RLS, EX-RLS and a particle filter are included for comparison.

The nonlinear Rayleigh fading multipath channel employed here is the cascade of a traditional Rayleigh fading multipath channel from [19] and a saturation nonlinearity. In the Rayleigh multipath fading channel, the number of the paths is chosen as $M = 5$, the maximum Doppler frequency $f_D = 100$ Hz and the sampling rate $T_s = 0.8$ μs (so it is a slow fading channel with the same fading rate for all the paths). All the tap coefficients are generated according to the Rayleigh model but only the real part is used in this experiment. A white Gaussian distributed time series (with unit power) is sent through this channel, corrupted with additive white Gaussian noise (with variance $\sigma^2 = 0.001$) and then the saturation nonlinearity $y = \tanh(x)$ is applied to it, where x is the output of the Rayleigh channel. The whole nonlinear channel is treated as a black box and only the input and output are known.

The tracking task is tested with 5 methods. The first one is the normalized LMS (shown as LMS-2 in the following tables and figures) (regularization factor $\epsilon = 10^{-3}$, step size $\eta = 0.25$); the

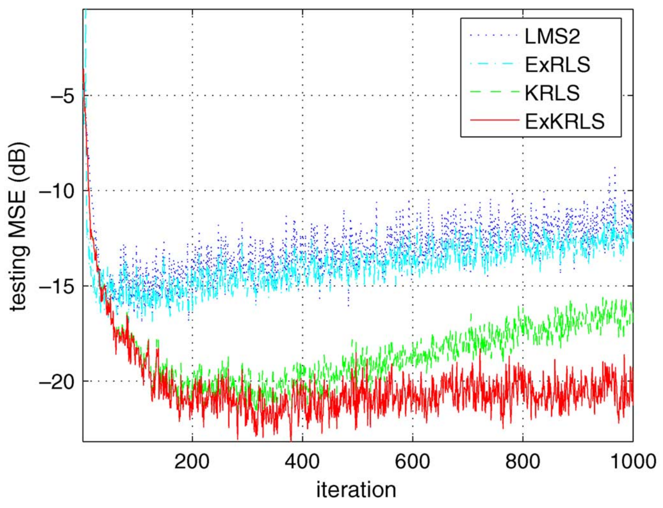


Fig. 1. Ensemble learning curves of LMS-2, EX-RLS, KRLS and EX-KRLS in tracking a Rayleigh fading multipath channel.

TABLE I
PERFORMANCE COMPARISON IN RAYLEIGH CHANNEL TRACKING

Algorithm	MSE (dB)
LMS-2	-11.78 ± 0.94
RLS	-12.02 ± 0.46
EX-RLS	-12.62 ± 0.56
KRLS	-16.63 ± 0.52
EX-KRLS	-20.44 ± 0.75

second is the RLS (with regularization parameter $\lambda = 10^{-3}$); the third one is the EX-RLS ($\alpha = 0.9999999368$, $q = 3.26 \times 10^{-7}$, $\beta = 0.995$, $\lambda = 10^{-3}$ according to [19 , p. 759]. The last two are nonlinear methods, namely the KRLS (regularization parameter $\lambda = 0.01$) and the proposed EX-KRLS ($\mathbf{A} = \alpha\mathbf{I}$, $\alpha = 0.999998$, $q = 10^{-4}$, $\beta = 0.995$, $\lambda = 0.01$). We use a Gaussian kernel in both cases with kernel parameter $a = 1$. Notice that α is very close to 1 and q very close to 0 since the fading of the channel is very slow. All the parameters are selected by scanning for best results.

We generate 1000 symbols for every experiment and perform 200 Monte Carlo experiments with independent inputs and additive noise. The ensemble learning curves are plotted in Fig. 1, which clearly show that the EX-KRLS has a better tracking ability than KRLS. The last 100 values in the learning curves are used to calculate the final mean square error (MSE), which is listed in Table I. It is seen that the nonlinear methods outperform the linear methods significantly, since the channel model we use here is nonlinear. Although the Rayleigh channel is a slow fading channel in this problem, we still enjoy nearly 4-dB improvement by using the EX-KRLS model.

Particle filters are sophisticated nonlinear model estimation techniques. The application of a particle filter requires the knowledge (or the estimation) of the state and observation models. With only the input-output provided and assuming no knowledge about the process, there are no general approaches for using it in state estimation. To provide a reference (rather than a fair comparison), we test its performance with different assumptions.

TABLE II
PERFORMANCE OF PARTICLE FILTER IN RAYLEIGH CHANNEL TRACKING

Algorithm setting	MSE (dB)
full knowledge	-30.73 ± 1.88
random initial	-8.78 ± 5.60
$y = 0.9x + 0.1x^2$	2.06 ± 5.63
$y = tanh(x/2)$	-10.21 ± 5.10
$y = tanh(3x/2)$	-6.20 ± 5.34
$y = x$	2.20 ± 6.00

The following model is assumed throughout the set of simulations

$$\begin{aligned}\mathbf{s}(i+1) &= \mathbf{F}\mathbf{s}(i) + \mathbf{n}(i)\\ d(i) &= h\left(\mathbf{s}(i)^T\mathbf{u}(i)\right) + \mathbf{v}(i)\end{aligned} \tag{58}$$

where $\mathbf{s}$ is regarded as the channel coefficients and h is the static nonlinearity.

In the first setting, $\mathbf{F}$, the variance of $\mathbf{n}$ and $\mathbf{v}$ are estimated from the true channel value. Notice that this is not possible in practice. Nevertheless we want to have the best possible performance from the particle filter. h is assumed known. The number of particles is set to 50. The true channel value at time 0 is used as the initial guess. The maximum a posterior estimation is used here [29]. We also assumed random initial conditions and model mismatch cases. For the random initial condition, we generated a standard normal distribution. For the model mismatch, we assumed various models as listed in Table II.

We compute the mean and standard deviation over 100 tests with independent inputs and initial states. The results are shown in Table II. As we see, the particle filter is able to outperform the extended KRLS with the full knowledge of the model. However it is very sensitive to the initialization and it performs badly with an incorrect model.

In the last simulation, we test the effectiveness of ALD to reduce the complexity in EX-KRLS. We choose ten thresholds in ALD in the range of [0,0.05]. For each threshold, 500 symbols are generated for every experiment and 50 Monte Carlo experiments are conducted with independent inputs and additive noise. The final MSE is calculated with the last 100 values in the ensemble learning curves. The final MSE versus the threshold is plotted in Fig. 2. The final MSE versus the final network size is plotted in Fig. 3. The regularization parameter is set as 0.01.

B. Lorenz Time Series Prediction

The Lorenz attractor, introduced by Edward Lorenz in 1963 [30], is a dynamic system corresponding to the long-term behavior of a chaotic flow, noted for its butterfly shape. The system is nonlinear, three-dimensional and deterministic. In 2001, it was proven by Warwick Tucker that for a certain set of parameters the system exhibits chaotic behavior and displays what is today called a strange attractor [31].

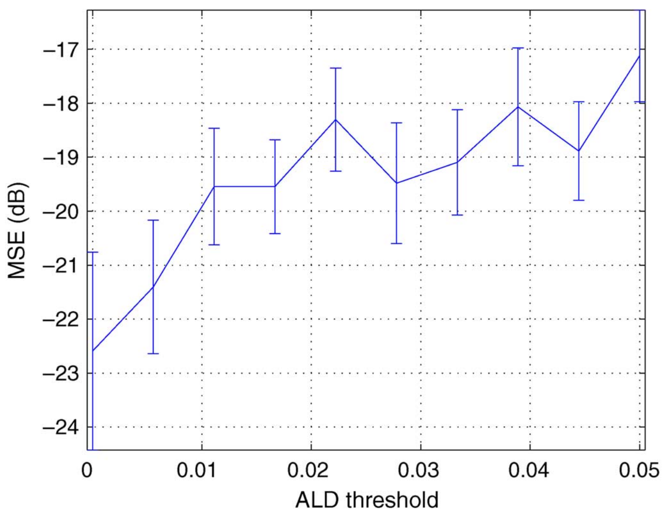


Fig. 2. Effect of approximate linear dependency in EX-KRLS.

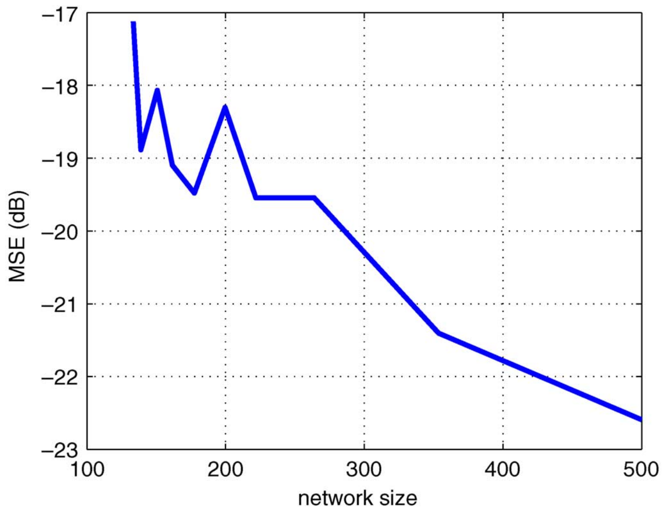


Fig. 3. Network size versus performance in EX-KRLS with ALD.

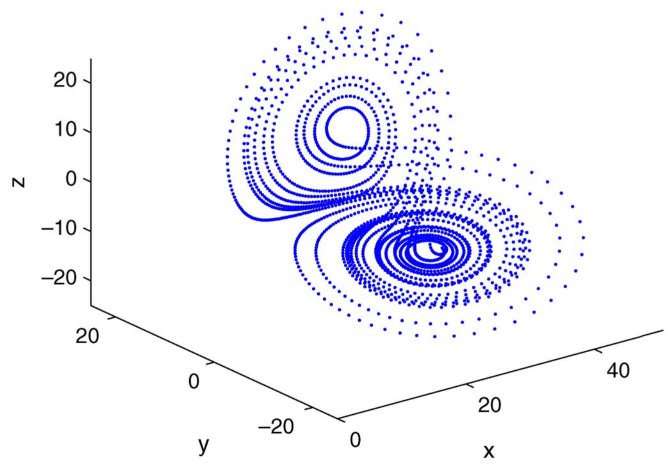


Fig. 4. State trajectory of Lorenz system for values $\beta = 8/3$, $\sigma = 10$ and $\rho = 28$.

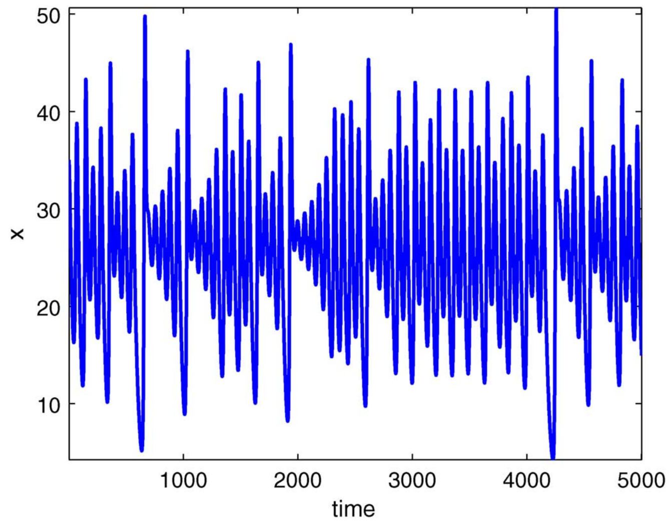


Fig. 5. Typical waveform of the first component from the Lorenz system.

The following set of differential equations dictates how the state of Lorenz system evolves over time in a complex, non-repeating pattern.

$$\begin{aligned}\frac{dx}{dt} &= -\beta x + yz \\ \frac{dy}{dt} &= \sigma(z-y) \\ \frac{dx}{dt} &= -xy + \rho y - z\end{aligned} \tag{59}$$

Setting $\beta = 8/3$, $\sigma = 10$ and $\rho = 28$, the state evolution pattern is plotted in Fig. 4. The first order approximation is used with a step size 0.01.

We pick the first component, namely x here for the short term prediction task. The first component is plotted in Fig. 5. The short term prediction task can be formulated as follows: using 5 past data $\mathbf{u}(i) = [x(i-5), x(i-4), \cdots, x(i-1)]^T$ as the input to predict $x(i+T)$ which is the desired response here. T is the prediction step. The larger T is, the more nonlinearity the system exhibits and hence the harder the problem is. The signal is preprocessed to be zero mean and unit variance before the modeling.

We test LMS-2, RLS, EX-RLS, and EX-KRLS on this task. We pick 20 prediction steps in the range of [1, 20]. For each prediction step, 50 independent simulations are run with different segments of the signal. The length of each segment is 1000 points. The final MSE is calculated from the last 100 points in the ensemble learning curves which are averaged over all the simulations. The regularization factor $\epsilon = 10^{-3}$, step size $\eta = 0.2$ in LMS-2. The regularization parameter $\lambda = 10^{-3}$ in RLS. $\alpha = 1$, $q = 0.01$, $\beta = 0.99$, $\lambda = 10^{-3}$ in EX-RLS. $\lambda = 10^{-3}$ in KRLS and $\mathbf{A} = \alpha\mathbf{I}$, $\alpha = 1$, $q = 0.01$, $\beta = 0.99$, $\lambda = 10^{-3}$ in EX-KRLS. We use a Gaussian kernel with kernel parameter $a = 1$. The performance is plotted in Fig. 6. It can be seen that the extended models exhibit better performance since the Lorenz system is switching between two attractors as shown in Fig. 4. With small prediction steps, the linear model perform equally well because the sampling rate is quite high and the signal becomes very smooth.

It is noticed that there are a few parameters to tune in each algorithm. We experiment with different parameters and

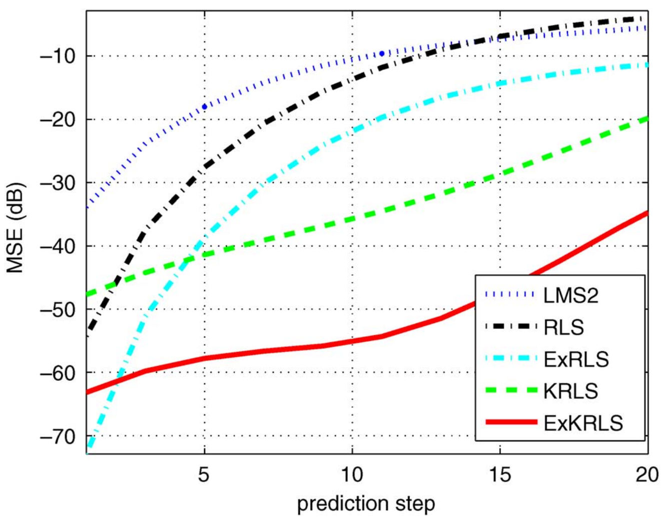


Fig. 6. Performance comparison of LMS-2, RLS, EX-RLS, KRLS, and EX-KRLS in Lorenz system prediction.

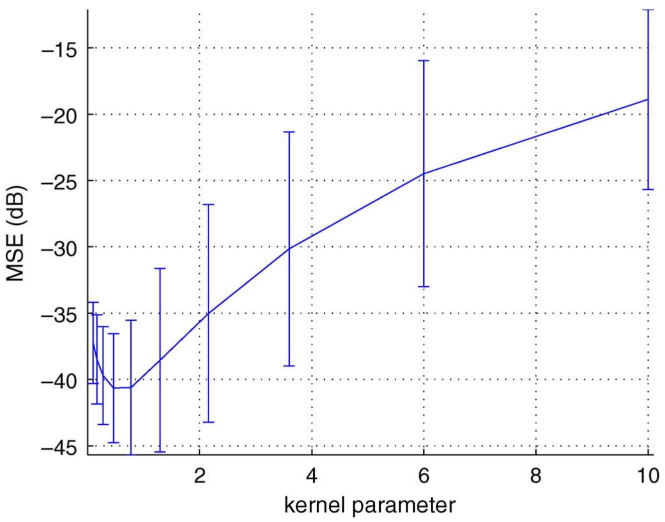


Fig. 7. Effect of kernel parameter on EX-KRLS in Lorenz system prediction.

choose the best through scanning. The effect of the kernel parameter on the performance of EX-KRLS is shown in Fig. 7 (prediction step $= 10$).

Next in Fig. 8, the ensemble learning curves are shown with 200 independent simulations on different input data. The prediction step is fixed at 10. Note that the larger fluctuations in EX-KRLS are due to the log plot, not due to the method. The final MSE is also listed in Table III.

Then we compare the performance of the SW-KRLS proposed in [9] with EX-KRLS. Since the modeling and tracking performance of SW-KRLS depends on the window length, we pick 20 different window lengths in the range of [10, 400]. For each window length, 50 independent simulations are run with different segments of signal. Each segment has 500 points. The prediction step is fixed at 10. The performance is plotted in Fig. 9. Clearly, EX-KRLS outperforms SW-KRLS significantly in this example. The trend of improving performance

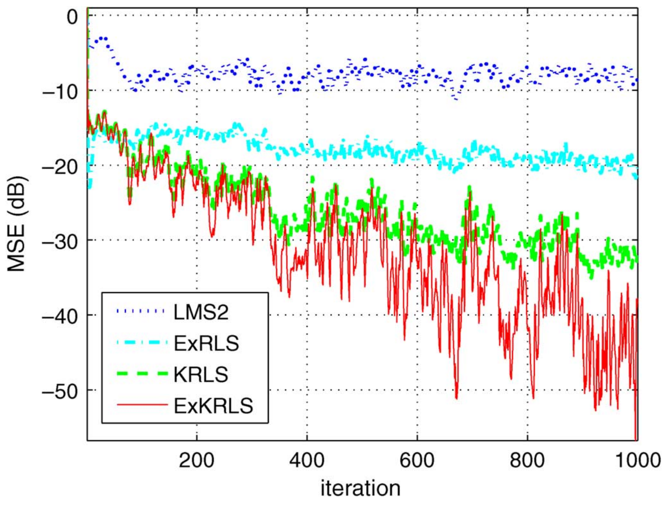


Fig. 8. Ensemble learning curves of LMS-2, RLS, EX-RLS, KRLS, and EX-KRLS in Lorenz system prediction.

TABLE III
PERFORMANCE COMPARISON IN LORENZ SYSTEM PREDICTION

Algorithm	MSE (dB)
LMS-2	-8.12 ± 0.91
RLS	-11.83 ± 0.71
EX-RLS	-19.89 ± 0.77
KRLS	-32.53 ± 1.13
EX-KRLS	-44.92 ± 4.80

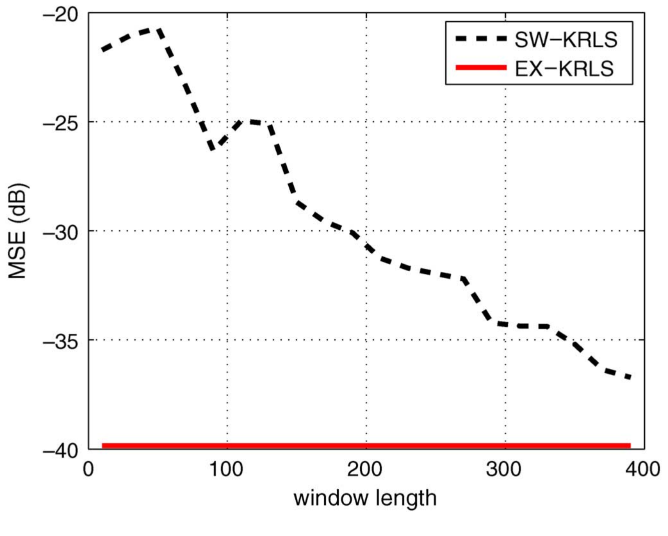


Fig. 9. Performance comparison of SW-KRLS and EX-KRLS in Lorenz system prediction.

with longer window length shows that SW-KRLS fails to improve the tracking ability since it eventually defaults to KRLS with the best performance.

VII. Discussion and Conclusion

The extended kernel recursive least squares algorithm is presented focusing on two special cases of the state transition matrix. Exponentially-weighted KRLS and random-walk KRLS are also discussed as special cases and all of them can be regarded as nontrivial extensions to KRLS, with an explicit state transition process. This algorithm is very suitable to model nonlinear systems with a slow fading and a small variation in state. Compared with the existing KRLS algorithms, EX-KRLS is a step closer to kernel Kalman filters. Preliminary results of this algorithm are promising, which suggests it can have a wide applicability in nonlinear extensions of EX-RLS.

Examples of Rayleigh channel tracking and Lorenz system prediction are illustrated with comparison among EX-KRLS, KRLS, SW-KRLS, particle filter, and linear methods. They show that EX-KRLS outperforms them in terms of its modeling and tracking ability. And more strikingly, the performance boost is achieved based on a simple tracking model. The particle filter is able to outperform EX-KRLS only with full knowledge of the model which is usually impractical.

Apparently how to learn the state transition model itself is very important, if such information is absent from domain knowledge. Though conventional methods such as maximum likelihood and expectation-maximization algorithm [18], [32] seem appropriate to the problem, a special treatment should and will be given in a separate work, to make this paper concise and precise.

All the kernel methods need to choose the kernel type and parameters. The most popular method so far is by cross validation [33]–[35]. The nearest neighbor method is also used in the resource-allocating networks [24] which allow the adaptation of the kernel size during the learning. With a close relation to Gaussian process theory, maximum marginal likelihood [28] is also applicable. Besides, the general kernel selection problem has been examined from a more rigorous viewpoint, formulating the problem as a convex optimization through parameterization of the kernel function family [36]–[38]. It will be interesting to investigate this problem specifically for the extended kernel recursive least squares.

APPENDIX A
PROOF OF THEOREM 4

Proof: First notice that by (30)

$$\begin{aligned}\mathbf{P}(0) &= \lambda^{-1}\mathbf{I},\\ r_e(1) &= \beta + \lambda^{-1}\kappa(\mathbf{u}(1),\mathbf{u}(1)),\\ \mathbf{P}(1) &= \mathbf{CBE}^T\left[\lambda^{-1}\mathbf{I} - \frac{\boldsymbol{\varphi}(1)\boldsymbol{\varphi}(1)^T}{\lambda^2 r_e(1)}\right]\mathbf{EB}^T\mathbf{C}^T + \beta q\mathbf{I}\\ &= \mathbf{C}\left[\frac{\mathbf{BE}^T\mathbf{EB}^T}{\lambda} - \frac{\mathbf{BE}^T\boldsymbol{\varphi}(1)\boldsymbol{\varphi}(1)^T\mathbf{EB}^T}{\lambda^2 r_e(1)}\right]\mathbf{C}^T + \beta q\mathbf{I}.\end{aligned}$$

Denote $\mathbf{G_E} = \mathbf{E}^T\mathbf{E}$ and $\mathbf{E}^T\boldsymbol{\varphi}(j) = \mathbf{h_E}(j)$, which are both computable by the kernel trick. So the claim is valid for $j=1$, namely,

$$\begin{aligned}\rho(1) &= \beta q,\\ \mathbf{Q}(1) &= -\frac{\mathbf{BG_EB}^T}{\lambda} + \frac{\mathbf{Bh_E}(1)\mathbf{h_E}(1)^T\mathbf{B}^T}{\lambda^2 r_e(1)}.\end{aligned}$$

Then by induction, the proof for all j follows. Assume it is true for $j = i-1$, i.e.,

$$\mathbf{P}(i-1) = \rho(i-1)\mathbf{I} - \mathbf{CQ}(i-1)\mathbf{C}^T \tag{60}$$

and $r_e(i)$ is computable using the kernel trick.

By substituting it into the last equation of (30), one has

$$\begin{aligned}\mathbf{P}(i) =& \mathbf{CBE}^T\left[\mathbf{P}(i-1) - \frac{\mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)}{r_e(i)}\right]\\ &\times \mathbf{EB}^T\mathbf{C}^T + \beta^i q\mathbf{I}\\ =& \beta^i q\mathbf{I} - \mathbf{C}\\ &\times\big[-\rho(i-1)\mathbf{BG_EB}^T - \mathbf{BG_{EC}Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\\ &+ \mathbf{Bh_E}(i)\mathbf{h_E}(i)^T\mathbf{B}^T\rho(i-1)^2/r_e(i)\\ &+ \mathbf{Bh_E}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\rho(i-1)/r_e(i)\\ &+ \mathbf{BG_{EC}Q}(i-1)\mathbf{h_C}(i)\mathbf{h_E}(i)^T\mathbf{B}^T\rho(i-1)/r_e(i)\\ &+ \mathbf{BG_{EC}Q}(i-1)\mathbf{h_C}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\\ &\times\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T/r_e(i)\big]\,\mathbf{C}^T\end{aligned}$$

where $\mathbf{G_E} = \mathbf{E}^T\mathbf{C}$ and $\mathbf{h_C}(i) = \mathbf{C}^T\boldsymbol{\varphi}(i)$, which are both computable using the kernel trick. Therefore,

$$\begin{aligned}\rho(i) &= \beta^i q,\\ \mathbf{Q}(i) &= -\rho(i-1)\mathbf{BG_EB}^T - \mathbf{BG_{EC}Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\\ &\quad+ \mathbf{Bh_E}(i)\mathbf{h_E}(i)^T\mathbf{B}^T\rho(i-1)^2/r_e(i)\\ &\quad+ \mathbf{Bh_E}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\rho(i-1)/r_e(i)\\ &\quad+ \mathbf{BG_{EC}Q}(i-1)\mathbf{h_C}(i)\mathbf{h_E}(i)^T\mathbf{B}^T\rho(i-1)/r_e(i)\\ &\quad+ \mathbf{BG_{EC}Q}(i-1)\mathbf{h_C}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\\ &\quad\times\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T/r_e(i).\end{aligned}$$

■

APPENDIX B
PROOF OF THEOREM 5

Proof: Notice that by (30)

$$\begin{aligned}\mathbf{k}_p(1) &= \frac{\mathbf{CBh_E}(1)}{\lambda r_e(1)},\\ \boldsymbol{\omega}(1) &= \mathbf{C}\left[\frac{\mathbf{Bh_E}(1)d(1)}{\lambda r_e(1)}\right] = \mathbf{Ca}(1).\end{aligned}$$

Thus, the claim is valid for $j=1$. Then we use induction to prove it is valid for any j. Assume it is true for $i-1$. By the recursion (30) and the result from theorem 4, we have

$$\begin{aligned}\boldsymbol{\omega}(i) &= \mathbf{CBE}^T\boldsymbol{\omega}(i-1) + k_p(i)e(i)\\ &= \mathbf{CBG_{EC}a}(i-1) + \mathbf{CBE}^T\mathbf{P}(i-1)\boldsymbol{\varphi}(i)e(i)/r_e(i)\\ &= \mathbf{CB}\{\mathbf{G_{EC}a}(i-1)\\ &\qquad+\mathbf{E}^T\left[\rho(i-1)\mathbf{I} - \mathbf{CQ}(i-1)\mathbf{C}^T\right]\\ &\qquad\times\boldsymbol{\varphi}(i)e(i)/r_e(i)\}\\ &= \mathbf{CB}\{\mathbf{G_{EC}a}(i-1)\\ &\qquad+\left[\rho(i-1)\mathbf{h_E}(i) - \mathbf{G_{EC}Q}(i-1)\mathbf{h_C}(i)\right]\\ &\qquad\times e(i)/r_e(i)\}.\end{aligned}$$

Therefore

$$\begin{aligned}\mathbf{a}(i) = \mathbf{B}\{&\mathbf{G_{EC}a}(i-1) + [\rho(i-1)\mathbf{h_E}(i)\\ &-\mathbf{G_{EC}Q}(i-1)\mathbf{h_C}(i)]\,e(i)/r_e(i)\}.\end{aligned}$$

This completes the proof. ■

REFERENCES

[1] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[2] B. Schölkopf, A. Smola, and K. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, pp. 1299–1319, 1998.

[3] W. Liu, P. Pokharel, and J. C. Príncipe, "The kernel least mean square algorithm," *IEEE Trans. Signal Process.*, vol. 56, no. 2, pp. 543–554, Feb. 2008.

[4] W. Liu and J. C. Príncipe, "Kernel affine projection algorithms," *EURASIP J. Adv. Signal Process.*, vol. 2008, Article ID 784292, 12 pp., 2008, doi:10.1155/2008/784292.

[5] T.-T. Frieb and R. F. Harrison, "A kernel-based ADALINE," in *Proc. Eur. Symp. Artificial Neural Networks*, Apr. 1999, pp. 245–250.

[6] A. Kuh, "Adaptive kernel methods for CDMA systems," in *Proc. Int. Joint Conf. Neural Networks*, Washington, DC, 2001, vol. 4, pp. 2404–2409.

[7] Y. Engel, S. Mannor, and R. Meir, "Sparse online greedy support vector regression," in *Proc. 13th Eur. Conf. Machine Learning*, 2002, pp. 84–96.

[8] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2275–2285, Aug. 2004.

[9] S. V. Vaerenbergh, J. Via, and I. Santamaría, "A sliding-window kernel RLS algorithm and its application to nonlinear channel identification," in *Proc. Int. Conf. Accoustics, Speech, Signal Processing* , May 2006, pp. 789–792.

[10] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME—J. Basic Eng.*, vol. 82, pp. 35–45, 1960.

[11] S. Haykin, *Adaptive Filter Theory*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[12] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[13] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Proc. Int. Symp. Aerospace/Defense Sensing, Simul. Controls*, 1997, pp. 182–193.

[14] E. A. Wan and R. van der Merwe, "The unscented Kalman filter for nonlinear estimation," presented at the Proc. IEEE Symp. (AS-SPCC), Lake Louise, AB, Canada, 2000.

[15] S. Arulampalam, S. Maskell, N. J. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

[16] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 2009.

[17] L. Ralaivola and F. d'Alché Buc, "Dynamical modeling with kernels for nonlinear time series prediction," *Adv. Neural Inf. Process. Syst.*, vol. 16, 2004.

[18] L. Ralaivola and F. d'Alche Buc, "Time series filtering, smoothing and learning using the kernel Kalman filter," in *Proc. 2005 IEEE Int. Joint Conf. Neural Networks*, 2005, pp. 1449–1454.

[19] A. Sayed, *Fundamentals of Adaptive Filtering*. New York: Wiley, 2003.

[20] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, pp. 337–404, 1950.

[21] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[22] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[23] I. Steinwart, "On the influence of the kernel on the consistency of support vector machines," *J. Mach. Learn. Res.*, vol. 2, pp. 67–93, 2001.

[24] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, no. 2, pp. 213–225, 1991.

[25] L. Csato and M. Opper, "Sparse online Gaussian processes," *Neural Comput.*, vol. 14, pp. 641–668, 2002.

[26] G. Williams, *Linear Algebra With Applications*. Boston, MA: Jones & Bartlett, 2007.

[27] J. A. K. Suykens, T. V. Gestel, J. Brabanter, B. D. Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. Singapore: World Scientific, 2002.

[28] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.

[29] A. Doucet, J. F. G. de Freitas, and N. J. Gordon, *Sequential Monte Carlo Methods in Practice*, 3rd ed. New York: Springer-Verlag, 2000.

[30] E. N. Lorenz, "Deterministic nonperiodic flow," *J. Atmospher. Sci.*, vol. 20, pp. 130–141, 1963.

[31] W. Tucker, "A rigorous ODE solver and Smale's 14th problem," *Found. Comput. Math.*, vol. 2, pp. 53–117, 2002.

[32] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc., Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[33] J. Racine, "An efficient cross-validation algorithm for window width selection for nonparametric kernel regression," *Commun. Stat.: Simulat. Comput.*, vol. 22, pp. 1107–1114, 1993.

[34] G. C. Cawley and N. L. C. Talbot, "Efficient leave-one-out cross-validation of kernel Fisher discriminant classifiers," *Pattern Recogn.*, vol. 36, pp. 2585–2592, 2003.

[35] S. An, W. Liu, and S. Venkatesh, "Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression," *Pattern Recogn.*, vol. 40, pp. 2154–2162, 2007.

[36] C. Micchelli and M. Pontil, "Learning the kernel function via regularization," *J. Mach. Learn. Res.*, vol. 6, pp. 1099–1125, 2005.

[37] A. Argyriou, C. A. Micchelli, and M. Pontil, "Learning convex combinations of continuously parameterized basic kernels," in *Proc. 18th Conf. Learning Theory*, 2005, pp. 338–352.

[38] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Mach. Learn.*, vol. 46, pp. 131–159, 2002.

Weifeng Liu (M'09) received the B.S. and M.S. degrees in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2003 and 2005, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, in 2008.

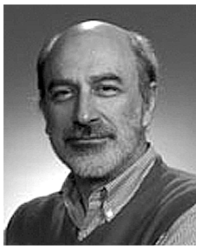
Upon graduation, he joined the forecasting team at Amazon.com, Seattle, WA. His research interests include adaptive signal processing, machine learning and data analysis.

Il (Memming) Park received the B.S. degree in computer science from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2004 and the M.S. degree in electrical and computer engineering from the University of Florida, Gainesville, in 2007. He is currently working towards the Ph.D. degree in biomedical engineering at the University of Florida, Gainesville.

He has been working in the Computational NeuroEngineering Laboratory at the University of Florida under the supervision of Dr. J. C. Principe since 2005. His research interests include neural computation, point processes, signal processing, and machine learning.

Yiwen Wang (M'08) received the B.Eng. and M.Eng. degrees from the University of Science and Technology of China, Hefei Anhui, China, in 2001 and 2004, respectively, and the Ph.D. degree in electrical engineering from the University of Florida, Gainesville, in 2008.

She was a research assistant at the University of Florida and has worked for Siemens Corporation Research, Princeton NJ. She is currently a Research Associate at the Hong Kong University of Science and Technology, Hong Kong. Her research interests are in brain–machine interfaces (BMIs), neuromorphic engineering, and adaptive signal processing.



José C. Príncipe (M'83–SM'90–F'00) is Distinguished Professor of electrical and biomedical engineering at the University of Florida, Gainesville, where he teaches advanced signal processing and artificial neural networks (ANNs) modeling. He is a BellSouth Professor and Founder and Director of the University of Florida Computational NeuroEngineering Laboratory (CNEL). He is involved in biomedical signal processing, in particular the electroencephalogram (EEG) and the modeling and applications of adaptive systems. He has more than 150 publications in refereed journals, 15 book chapters, and over 350 conference papers. He has directed over 60 Ph.D. dissertations and 61 Master's degree theses.

Dr. Príncipe is Past Editor-in-Chief of the IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, Past President of the International Neural Network Society, and former Secretary of the Technical Committee on Neural Networks of the IEEE Signal Processing Society. He is an AIMBE Fellow and a recipient of the IEEE Engineering in Medicine and Biology Society Career Service Award. He was also a member of the Scientific Board of the Food and Drug Administration, and a member of the Advisory Board of the McKnight Brain Institute at the University of Florida.