KALMAN FILTERING IN REPRODUCING KERNEL HILBERT SPACES

By
PINGPING ZHU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2013
© 2013 Pingping Zhu
ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Dr. José C. Príncipe, for his support, encouragement and patience in guiding the research. His openness and support to my investigative ideas had help me to become a better research. I would like to thank Dr. Dapeng Wu, Dr. John M. Shea, and Dr. Murali Rao for being serving on my committee and for their helpful advice. Their valuable comments and constructive criticism help improve the quality of this work greatly. Many thanks are due to former lab member, Dr. Badong Chen, for his advice and many discussions that made my research a lot easier and happier. Thanks are due to all of current lab members for their support. Finally, I would like to thank my parents for their encouragement and support with me.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>3</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>7</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>8</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>10</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>12</td>
</tr>
<tr>
<td>1.1 Estimation, prediction and smooth</td>
<td>12</td>
</tr>
<tr>
<td>1.2 Background</td>
<td>13</td>
</tr>
<tr>
<td>1.3 Requirements of proposed algorithms</td>
<td>14</td>
</tr>
<tr>
<td>2 RELATED WORK</td>
<td>16</td>
</tr>
<tr>
<td>2.1 Algorithms in the Input Space</td>
<td>16</td>
</tr>
<tr>
<td>2.1.1 Bayesian Filter</td>
<td>16</td>
</tr>
<tr>
<td>2.1.2 Kalman Filter</td>
<td>17</td>
</tr>
<tr>
<td>2.1.3 Nonlinear Kalman Filter</td>
<td>19</td>
</tr>
<tr>
<td>2.1.3.1 Extended Kalman Filter</td>
<td>20</td>
</tr>
<tr>
<td>2.1.3.2 Unscented Kalman Filter</td>
<td>21</td>
</tr>
<tr>
<td>2.1.3.3 Cubature Kalman Filter</td>
<td>22</td>
</tr>
<tr>
<td>2.1.4 Dual Kalman Filter</td>
<td>25</td>
</tr>
<tr>
<td>2.1.5 Adaptive filters</td>
<td>25</td>
</tr>
<tr>
<td>2.1.5.1 Recursive Least Squares</td>
<td>27</td>
</tr>
<tr>
<td>2.1.5.2 Extended Recursive Least Squares</td>
<td>28</td>
</tr>
<tr>
<td>2.2 Algorithms in RKHS</td>
<td>29</td>
</tr>
<tr>
<td>2.2.1 Foundations of RKHS</td>
<td>30</td>
</tr>
<tr>
<td>2.2.2 Kernel Recursive Least Squares</td>
<td>32</td>
</tr>
<tr>
<td>2.2.2.1 KRLS-ALD</td>
<td>33</td>
</tr>
<tr>
<td>2.2.2.2 QKRLS</td>
<td>34</td>
</tr>
<tr>
<td>2.2.3 Extended Kernel Recursive Least Squares</td>
<td>35</td>
</tr>
<tr>
<td>2.2.4 Revisiting the Ex-KRLS Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>2.2.4.1 Existence of State Transition Operator</td>
<td>39</td>
</tr>
<tr>
<td>2.2.4.2 Value of State Transition Parameter</td>
<td>41</td>
</tr>
<tr>
<td>2.2.4.3 Relationship with the Kalman Filter</td>
<td>42</td>
</tr>
<tr>
<td>2.2.4.4 Conclusions about Ex-KRLS algorithm</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5 Kernel Kalman Filter</td>
<td>43</td>
</tr>
<tr>
<td>2.3 Summary of Related Algorithms</td>
<td>44</td>
</tr>
</tbody>
</table>
# 3 A NOVEL EXTENDED KERNEL RECURSIVE LEAST SQUARES

3.1 Overview ................................. 46
3.2 Extended Kalman Filter-Kernel Recursive Least Squares Algorithm ........ 47
3.3 Formulation of the Novel Extended Kernel Recursive Least Squares .......... 49
3.4 Experiments and Results .......................... 53
   3.4.1 Vehicle tracking .................................. 53
   3.4.2 Rayleigh channel tracking ...................... 57
3.5 Discussion and Conclusion ................................ 62

# 4 LEARNING NONLINEAR GENERATIVE MODELS OF TIME SERIES WITH A KALMAN FILTER IN RKHS

4.1 Overview ..................................... 66
4.2 Conditional Embeddings in RKHS .................................. 69
   4.2.1 Hilbert Space Embeddings ....................... 69
   4.2.2 Conditional embedding operator ................... 72
   4.2.3 Dynamical System Model with a Conditional Embedding operator .... 74
4.3 Kernel Kalman Filter based on Conditional Embedding Operator ............ 75
   4.3.1 Construction of Kalman Filtering in the RKHS ............... 76
   4.3.2 Kalman Filter Predictor in RKHS .................. 83
4.4 Simplifying the conditional embedding operator ................................ 85
   4.4.1 Down Sampling Training Data ...................... 85
   4.4.2 Quantizing Training Data .......................... 85
   4.4.3 Training Data for Online Operation .................. 88
4.5 Experiments and Results .................................. 90
   4.5.1 Estimation of noisy time-series ..................... 90
      4.5.1.1 Discussion of noise parameters ............... 91
      4.5.1.2 Comparison of estimation results ............. 93
   4.5.2 Prediction of noisy time-series .................... 96
   4.5.3 Prediction of Sunspot data set .................... 100
4.6 Discussion and Conclusion ................................ 102

# 5 KERNEL KALMAN FILTER BASED ON CONDITIONAL EMBEDDINGS

5.1 Overview ..................................... 106
5.2 State-Space Model in RKHS .................................. 107
5.3 Derivation of Full-blown Kernel Kalman Filter ............................... 109
   5.3.1 Recursive Equations with Given Training Data .......... 110
   5.3.2 FKKF algorithm with generated training data ........ 112
5.4 Experiment and Result .................................. 115
5.5 Discussion ..................................... 118
5.6 Conclusion ..................................... 120
6 CONCLUSION AND FUTURE WORK ................................................................. 121

6.1 Conclusion and Discussion ................................................................. 121

6.2 Future work ......................................................................................... 124
   6.2.1 Noise model in RKHS ................................................................... 124
   6.2.2 Operator Design .......................................................................... 124
   6.2.3 Application of Kalman filter in Kernel Adaptive Algorithms ....... 124

APPENDIX

A SQUARE-ROOT CUBATURE KALMAN FILTER ........................................ 126

B PROOFS OF THEOREM 4.1 AND THEOREM 4.2 ................................... 128

C PROOFS OF THEOREM 4-3 ................................................................. 131

D PROOFS OF THEOREM 5.1 AND THEOREM 5.2 ................................. 132

E KERNEL RECURRENT SYSTEM TRAINED BY REAL-TIME RECURRENT
   LEARNING ALGORITHM ........................................................................ 135
   E.1 Introduction .................................................................................... 135
   E.2 Kernel recurrent networks ............................................................. 136
   E.3 On-line recurrent system learning .................................................. 138
      E.3.1 Kernel real-time recurrent learning algorithm ......................... 138
      E.3.2 Teacher forcing ...................................................................... 141
   E.4 Experiments and results ................................................................. 143
   E.5 Conclusion ..................................................................................... 144

REFERENCES ......................................................................................... 146

BIOGRAPHICAL SKETCH ........................................................................ 154
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>MSE of distance $\rho$ and slope $k$</td>
<td>56</td>
</tr>
<tr>
<td>3-2</td>
<td>MSE of position</td>
<td>57</td>
</tr>
<tr>
<td>3-3</td>
<td>Performance comparison in Rayleigh fading channel tracking with maximum Doppler frequency $f_D = 100$ Hz</td>
<td>60</td>
</tr>
<tr>
<td>3-4</td>
<td>Performance comparison in Rayleigh fading channel tracking with maximum Doppler frequency $f_D = 500$ Hz</td>
<td>62</td>
</tr>
<tr>
<td>3-5</td>
<td>Computational complexity analysis</td>
<td>64</td>
</tr>
<tr>
<td>4-1</td>
<td>MSE of estimation of IKEDA data for different algorithms ($\alpha = 1.6$ for $\alpha$ stable noise)</td>
<td>94</td>
</tr>
<tr>
<td>4-2</td>
<td>Prediction MSE with $k = 50$</td>
<td>99</td>
</tr>
<tr>
<td>4-3</td>
<td>Prediction MSE with $k = 100$</td>
<td>99</td>
</tr>
<tr>
<td>4-4</td>
<td>Time complexities</td>
<td>104</td>
</tr>
<tr>
<td>5-1</td>
<td>NMSE of State estimation</td>
<td>117</td>
</tr>
<tr>
<td>6-1</td>
<td>Algorithm requirements check list</td>
<td>123</td>
</tr>
<tr>
<td>E-1</td>
<td>Using $x(1)$ as inputs to predict $x(1)$, $x(2)$ and $x(3)$</td>
<td>144</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Relationship of estimation, prediction and modeling</td>
<td>13</td>
</tr>
<tr>
<td>2-1</td>
<td>Dual estimation problem</td>
<td>26</td>
</tr>
<tr>
<td>2-2</td>
<td>Relationship between algorithms</td>
<td>45</td>
</tr>
<tr>
<td>3-1</td>
<td>Trajectory of the vehicle with background</td>
<td>53</td>
</tr>
<tr>
<td>3-2</td>
<td>Trajectory of the vehicle</td>
<td>54</td>
</tr>
<tr>
<td>3-3</td>
<td>Trajectories of the true position and the predictions of the EKF, KRLS, and Ex-KRLS-KF algorithms</td>
<td>57</td>
</tr>
<tr>
<td>3-4</td>
<td>Comparison of the EKF, KRLS, and Ex-KRLS-KF algorithms</td>
<td>58</td>
</tr>
<tr>
<td>3-5</td>
<td>Ensemble learning curves of the LMS-2, RLS, EX-RLS, KRLS, EX-KRLS and EX-KRLS-KF in tracking a Rayleigh fading multipath channel with maximum Doppler frequency $f_D = 100\text{Hz}$</td>
<td>60</td>
</tr>
<tr>
<td>3-6</td>
<td>Ensemble learning curves of the LMS-2, RLS, EX-RLS, KRLS, EX-KRLS and EX-KRLS-KF in tracking a Rayleigh fading multipath channel with maximum Doppler frequency $f_D = 500\text{Hz}$</td>
<td>61</td>
</tr>
<tr>
<td>3-7</td>
<td>MSE versus maximum Doppler frequency. Error bars mark one standard deviation above and below the mean</td>
<td>63</td>
</tr>
<tr>
<td>4-1</td>
<td>Generated time series</td>
<td>66</td>
</tr>
<tr>
<td>4-2</td>
<td>Trajectory of the first 200 points of the series IKEDA</td>
<td>90</td>
</tr>
<tr>
<td>4-3</td>
<td>MSE of estimation results (a) with respect to $q$ and $r$ (b) with respect to $\theta$ and $\rho$</td>
<td>91</td>
</tr>
<tr>
<td>4-4</td>
<td>MSE of estimation results versus $r/q$. Error bars mark one standard deviation above and below the mean</td>
<td>93</td>
</tr>
<tr>
<td>4-5</td>
<td>SNR of estimation results versus input signal. Error bars mark one standard deviation above and below the mean.</td>
<td>95</td>
</tr>
<tr>
<td>4-6</td>
<td>State trajectory of Lorenz system for parameters $\beta = 8/3$, $\sigma = 10$ and $\rho = 28$</td>
<td>97</td>
</tr>
<tr>
<td>4-7</td>
<td>10-step Prediction MSE with different $k$</td>
<td>98</td>
</tr>
<tr>
<td>4-8</td>
<td>Prediction MSE with $k = 50$</td>
<td>99</td>
</tr>
<tr>
<td>4-9</td>
<td>Prediction MSE with $k = 100$</td>
<td>100</td>
</tr>
<tr>
<td>4-10</td>
<td>Normalized annual sunspot data over the period 1700-2011</td>
<td>101</td>
</tr>
</tbody>
</table>
Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

KALMAN FILTERING IN REPRODUCING KERNEL HILBERT SPACES

By

Pingping Zhu

May 2013

Chair: José C. Príncipe
Major: Electrical and Computer Engineering

There are numerous dynamical system applications that require estimation or
prediction from noisy data, including vehicle tracking, channel tracking, time series
denoising, prediction, estimation, and so on. Many linear algorithms have been
developed to deal with these problems under different assumptions and approximations,
such as the Kalman filter, the recursive least squares algorithm, and the least mean
squares algorithm. However, these linear algorithms cannot solve nonlinear problems
that often occur in real life. To address these nonlinear problems, some nonlinear
algorithms have been recently proposed, like kernelized version of the linear algorithms.
Our research follows this line and seeks to develop novel algorithms using kernel
methods to deal with nonlinear problems. Specifically, our goal is to derive the Kalman
filter in the reproducing kernel Hilbert space (RKHS), which is a space of functions, to
implement signal denoising, prediction and estimation.

In this dissertation, we first analyze and discuss in depth the extended kernel
recursive least squares algorithm, and point out the limitation of this algorithm and its
close relationship with the Kalman filter in RKHS. Next, we develop a novel extended
kernel recursive least squares based on the nonlinear Kalman filters and kernel
recursive least squares algorithms, to improve tracking and prediction performance.
However, the nonlinear Kalman filter is implemented in the input space, not the RKHS.
Then, we introduce the concepts of embeddings in RKHS and conditional embedding
operators, and develop an algorithm to learn nonlinear generative models of time series based on the embeddings and conditional embedding operators to denoise and predict time series. We present a detailed validation of the method, including an analysis of how to speed up the calculations. This algorithm is a Kalman filter in RKHS with a trivial measurement model, in which the estimated measurement embeddings as the hidden states in RKHS are propagated and estimated. Because the embeddings represent distributions of random variables, the models describe the measurement dynamics. Finally, a Kalman filter in RKHS with non-trivial measurement model is proposed, named full-blown kernel Kalman filter (FKKF), in which the state embeddings are treated as the hidden states in RKHS. Because of the non-trivial measurement model and the state embeddings, this algorithm is supposed to be able to handle more complicated dynamical system problems. We present a simple example that shows its performance.

In conclusion this dissertation establishes the framework to advance the theory of state models in RKHS and shows the potential and also identifies the difficulties that needs to be conquered to establish practical implementations of nonlinear state models.
CHAPTER 1
INTRODUCTION

This dissertation addresses the problem of modeling a dynamical system for estimation and prediction. Applications including vehicle tracking, channel tracking, and time series denoising and prediction. Generally speaking, time series filtering problems like estimation and prediction, require system modeling, which is usually described in terms of mathematical equations. Here we consider the dynamical system, which contains state and measurement models defined as:

\[ x_{i+1} = f_i(x_i, u_i) + n_i \]  \hspace{1cm} (1–1)

\[ y_i = h_i(x_i, u_i) + v_i \]  \hspace{1cm} (1–2)

where \( x_i \in \mathbb{R}^{n_x} \) is the state of the dynamic system at discrete time \( i \), which is defined as the minimal model that is sufficient to uniquely describe the unforced dynamical behavior of the system, but usually unobservable; \( u_i \in \mathbb{R}^{n_u} \) is the input of the system; \( y_i \in \mathbb{R}^{n_y} \) is the measurement, which is measurable or observable; \( f_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x} \) and \( h_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_y} \) are some functions used to describe the dynamics of the system; \{\( n_i \)\} and \{\( v_i \)\} are independent process and measurement noise sequences with zero means and covariances \( Q_i \) and \( R_i \), respectively. Here noises are added to the models to reflect the uncertainty of the model and the measurement noise.

1.1 Estimation, prediction and smooth

With the above model, we can approximate the hidden state \( x_j \) using all available data \{\( y_k \)\} and \{\( u_k \)\} up to current time \( i \). When the time \( j \leq i \), the operation is called estimation, while when the time \( j > i \), it is prediction. Here the tasks of estimation and prediction are done with the assumed known model. On the other hand, the task that identifies the underlying model of the system using all information is modeling.

It is clear that these tasks are strongly interdependent. For instance, when the accurate model of the system or clean (noiseless) signal generated by the model
is available, one can obtain the other by modeling or estimation. Furthermore, if an accurate model and good signal estimates are available, good predictions can be generated by using the estimates as inputs to the model. (see Figure 1-1)

![Figure 1-1. Relationship of estimation, prediction and modeling](image)

**1.2 Background**

However, in real cases, neither the accurate model nor noise free signal are available, hence the problems are much more challenging. In these cases, some approximated models are assumed based on the prior knowledge and noises are added to reflect the uncertainty of the assumed model. Many algorithms are developed under thus assumptions, like the particle filters (PF) \[18, 19\] and the (nonlinear) Kalman filters (KF) \[25–29, 33–35, 38, 39\].

Although noises are introduced to these algorithms to compensate the inaccuracy of the model, it is still not satisfactory. Therefore, the dual estimation is studied by some researchers. The task of the dual estimation problem is to estimate hidden states from noisy data and approximate the underlying system model, like the dual Kalman filter (DKF) \[34, 43–45\].

Another methodology is to simplify the system model as a linear model and ignore the state model, normally referred as adaptive filtering, like the recursive least squares (RLS) \[1\] algorithm and the least mean squares (LMS) \[1\] algorithm. But because of the simplification, these algorithms cannot be applied to estimate or predict the hidden...
states. These algorithms just focus on system’s inputs and outputs and try to achieve the best linear relationship between them.

Recently, some nonlinear algorithms have also been proposed based on these linear algorithms, all of which are supported by the kernel theory, like the kernel least mean squares (KLMS) [15], the kernel recursive least squares (KRLS) [16] and the extended kernel recursive least squares (Ex-KRLS) [17] algorithms. However, most of these algorithms still just focus on the relationship between inputs and outputs, not on the states of the system. Even for the Ex-KRLS algorithm, just a simple random walk model is used to describe the state model in RKHS.

1.3 Requirements of proposed algorithms

Kernel methods express nonlinear functions in an inner product form in the RKHS, therefore linear adaptive filter algorithms can be constructed in the reproducing kernel Hilbert space (RKHS) to solve nonlinear adaptive filter problems. Since there is a close relationship between linear adaptive filters and the Kalman filter, we seek to develop a linear dynamical model algorithm, the Kalman filter, in the RKHS to solve nonlinear dynamical problems. Because of the properties of the kernel method, we expect that our proposed algorithms can achieve the following requirements:

1. Implement nonlinear models in the input space, using a linear state model framework in the RKHS and learn the system model from data directly and represent it in the RKHS.

2. Estimate or predict the hidden state and the output of the system model with evaluation in the input space.

3. Be robust in a non-Gaussian noise environment.

Of course, going to the RKHS also brings disadvantages because we lose access to the state (now a function in RKHS) and the calculations become much more involved and time consuming. In fact, we still do not know if it is possible to design realistic state space models in RKHS. This dissertation is one of the first attempts to do so. We hope the Kalman filter in the RKHS could be another methodology besides the nonlinear
Kalman filters and the particle filter to solve nonlinear dynamical problems. In addition, because the novel algorithms should survive in a non-Gaussian noise environment, we expect them to outperform other existing algorithms for some non-Gaussian cases.

In order to develop the algorithms achieving all of these requirements, we study some existing related algorithms and obtain some preliminary achievements. Furthermore, these algorithms will bring us more ideas to design new algorithms.

The rest of the dissertation is organized as follows. Some existing related algorithms are reviewed in Chapter 2, including the algorithms in the input space and the algorithms in the reproducing kernel Hilbert space. Especially, the extended kernel recursive least squares algorithm is analyzed and discussed deeply in Section 2.2.3. The limitation of this algorithm and its close relationship with the Kalman filter in RKHS is stated. Then a novel extended kernel recursive least squares algorithm is proposed in Chapter 3. In Chapter 4, a Kalman filter with trivial measurement model are developed in RKHS. After that, a more complicated Kalman filter in RKHS is proposed in Chapter 5. Finally, the conclusion and future works are given in Chapter 6.
In this Chapter, we review some existing algorithms that are applied to noisy time series. These algorithms are categorized into two groups according to the space where the algorithm is developed. The review of these algorithms also gives the motivation of this dissertation.

### 2.1 Algorithms in the Input Space

#### 2.1.1 Bayesian Filter

When the system model in (1–1) and (1–2) is assumed to be known, the Bayesian filters can be applied to estimate or predict the states. In the Bayesian filtering framework, the posterior density of the state is maintained by system dynamics and new measurement, which provides a complete statistical description of the state of the dynamic system. Maintaining the posterior density includes two steps: prediction and measurement update [19].

*Step 1: Prediction.*

\[
p(x_i|D_{i-1}) = \int p(x_i|x_{i-1}) p(x_{i-1}|D_{i-1}) \, dx_{i-1} \tag{2–1}
\]

where \(D_{i-1} = \{y_{i,j}\}_{j=1}^{k-1}\) denotes the history of the measurements up to time \((i - 1)\), and \(p(x_{i-1}|D_{i-1})\) is the posterior density at the last time \((i - 1)\). The state transition density \(p(x_i|x_{i-1})\) is obtained from (1–1).

*Step 2: Measurement update.*

\[
p(x_i|D_i) = \frac{p(y_i|x_i) p(x_i|D_{i-1})}{p(y_i|D_{i-1})} \tag{2–2}
\]

The denominator is calculated as

\[
p(y_i|D_{i-1}) = \int p(y_i|x_i) p(x_i|D_{i-1}) \, dx_i \tag{2–3}
\]

where the measurement likelihood function \(p(y_i|x_i)\) is obtained from (1–2).
The Bayesian filtering framework provides a conceptual unified recursive approach for nonlinear filtering problems to maintain the posterior distribution of the hidden state. However, it is practically inefficient, because it involves multi-dimensional integrals and the conditional probability density functions (pdf) are also needed. Furthermore, it is not easy to express arbitrary distribution functions.

The Particle filter (PF) algorithm is a sequential Monte Carlo method to implement the Bayesian filter [18–23]. The PF algorithm represents the required pdf as a set of important samples (or "particles"), rather than a function over the state space. The algorithm recursively propagates and updates these samples for the discrete time problem, and can estimate the hidden state using these samples. However, to represent the pdf, the PF algorithm has to generate a large number of particles for each iteration and resample them, which results in high computational complexity and memory requirements.

### 2.1.2 Kalman Filter

For the sake of simplification, many algorithms are developed under the Gaussian assumption, i.e. in the space of Gaussian functions. Under these conditions, the predictive density \( p(x_i|D_{i-1}) \) and the filter likelihood density \( p(y_i|x_i) \) are both Gaussian, which leads to a Gaussian posterior density \( p(x_i|D_i) \). The Gaussian distribution is determined by its mean and covariance. Under the Gaussian approximation, the functional recursion of the Bayesian filter reduces to an algebraic recursion operating only on the means and covariances of various conditional densities encountered in the time and the measurement updates.

Because the Gaussian family is closed under linear transformation, when the system is linear, we have the celebrated Kalman filter (KF) proposed by R. E. Kalman in 1960 [25]. The KF provides a recursive optimal solution to the linear filtering problem, and is widely applied to stationary as well as non-stationary linear
dynamical environments to estimate the current hidden states or predict hidden states or measurements in the future.

For the linear system, the dynamical model (1–1) and (1–2) can be rewritten as:

\[ x_{i+1} = F_i x_i + n_i \]  \hspace{1cm} (2–4)
\[ y_i = H_i x_i + v_i \]  \hspace{1cm} (2–5)

where \( F_i \) is the transition matrix taking the state \( x_i \) from time \( i \) to time \( i + 1 \), and \( H_i \) is the measurement matrix. The process noise \( n_i \) is assumed to be zero-mean, additive, white, and Gaussian, with the covariance matrix defined by

\[
E[n_in_n^T] = \begin{cases} 
Q_i & \text{for } i = j \\
0 & \text{for } i \neq j 
\end{cases} \hspace{1cm} (2–6)
\]

Similarly, the measurement noise \( v_i \) is assumed to be zero-mean, additive, white, and Gaussian, with the covariance matrix defined by

\[
E[v_iv_j^T] = \begin{cases} 
R_i & \text{for } i = j \\
0 & \text{for } i \neq j 
\end{cases} \hspace{1cm} (2–7)
\]

Suppose that a measurement on a linear dynamical system, described by (2–4) and (2–5), has been made at time \( i \). The requirement is to use the information contained in the new measurement \( y_i \) to update the estimation of the unknown hidden state \( x_i \). Let \( \hat{x}_i^- \) denote a priori estimate of the state, which is already available at time \( i \). In the Kalman filter algorithm, the hidden state \( x_i \) is estimated as a linear combination of \( \hat{x}_i^- \) and the new measurement \( y_i \), in the form of

\[
\hat{x}_i = \hat{x}_i^- + G_i(y_i - H_i\hat{x}_i^-) \]  \hspace{1cm} (2–8)

where the matrix \( G_i \) is called Kalman gain.
The Kalman filter algorithm is summarized in Algorithm 2-1. The details can be found in [25, 26].

**Algorithm 2-1: Kalman filter**

**Initialization:** For \( i = 0 \), set

\[
\hat{x}_0 = E[x_0] \\
\mathbf{P}_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T]
\]

**Computation:** For \( i = 1, 2, \ldots \), compute:

- **State estimate propagation**
  \[
  \hat{x}_i = \mathbf{F}_i \hat{x}_{i-1}
  \]

- **Error covariance propagation**
  \[
  \mathbf{P}_i = \mathbf{F}_i \mathbf{P}_{i-1} \mathbf{F}_i^T + \mathbf{Q}_{i-1}
  \]

- **Kalman gain matrix**
  \[
  \mathbf{G}_i = \mathbf{P}_i - \mathbf{H}_i \mathbf{P}_i - \mathbf{H}_i^T \left[ \mathbf{H}_i \mathbf{P}_i - \mathbf{H}_i^T + \mathbf{R}_i \right]^{-1}
  \]

- **State estimate update**
  \[
  \hat{x}_i = \hat{x}_i - \mathbf{G}_i (y_i - \mathbf{H}_i \hat{x}_i)
  \]

- **Error covariance update**
  \[
  \mathbf{P}_i = (I - \mathbf{G}_i \mathbf{H}_i) \mathbf{P}_i - \mathbf{G}_i \mathbf{P}_i - \mathbf{H}_i^T
  \]

Here, \( \mathbf{P}_i^- \) and \( \mathbf{P}_i \) are a priori covariance matrix and a posteriori covariance matrix, respectively, defined as

\[
\mathbf{P}_i^- = E[(x_i - \hat{x}_i^-)(x_i - \hat{x}_i^-)^T], \quad (2–9) \\
\mathbf{P}_i = E[(x_i - \hat{x}_i)(x_i - \hat{x}_i)^T]. \quad (2–10)
\]

### 2.1.3 Nonlinear Kalman Filter

The Kalman filter has been applied to a wide range of fields like optimal controls, statistical signal processing, and econometrics. However, there are many applications, in which the assumption of linearity is not satisfied. When the system is nonlinear, specifically, the functions \( f \) and \( h \) in (1–1) and (1–2) are nonlinear, the integral below is involved in the prediction and update of the means and covariances.[38]

\[
I = \int f(x)p(x)\,dx \quad (2–11)
\]

where \( f(\cdot) \) denotes a nonlinear function and \( p(x) \) is a Gaussian distribution with respect to \( x \). In order to deal with (2–11), many algorithms are proposed, including extended
Kalman filter (EKF), unscented Kalman filter (UKF) and cubature Kalman filter (CKF). All of these algorithms are still developed under the assumptions of the known system and observation models. When they are not known, there is a need to learn them from data. Specifically, when the system and observation are described by parameterized models and these parameters are unknown, the dual Kalman filter (DKF) can be applied to solve the problem, which is discussed in section 2.1.4.

2.1.3.1 Extended Kalman Filter

To develop the extended Kalman filter (EKF), the basic idea is to linearize the state-space model of (1–1) and (1–2) at each time instant around the most recent state estimate, which is taken to be either \( \hat{x}_i \) or \( \hat{x}_i^- \), depending on which particular function is being considered. Once a linear model is obtained, the standard Kalman filter can be applied. Here the noise pdf is still considered Gaussian. The algorithm of EKF is summarized in Algorithm 2-2. The details can be found in [27–31].

---

Algorithm 2-2: Extended Kalman filter

Initialization: For \( i = 0 \), set
\[
\hat{x}_0 = E[x_0] \\
P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T]
\]

Computation: For \( i = 1, 2, \ldots \), compute:

Definitions: linearize the state-space model
\[
F_i = \frac{\partial f(x)}{\partial x}|_{x=\hat{x}_{i-1}} \\
H_i = \frac{\partial h(x)}{\partial x}|_{x=\hat{x}_i^-}
\]

State estimate propagation
\[
\hat{x}_i^- = f(\hat{x}_{i-1})
\]

Error covariance propagation
\[
P_i^- = F_i P_{i-1} F_i^T + Q_{i-1}
\]

Kalman gain matrix
\[
G_i = P_i^- H_i^T \left[H_i P_i^- H_i^T + R_i\right]^{-1}
\]

State estimate update
\[
\hat{x}_i = \hat{x}_i^- + G_i (y - h(\hat{x}_i^-))
\]

Error covariance update
\[
P_i = (I - G_i H_i) P_i^-.
\]

For the EKF, the nonlinear functions are linearized at the estimate values \( \hat{x}_i \) and \( \hat{x}_i^- \). However, the linearization of nonlinear functions results in an inaccurate propagation of
the pdf, and degrades the performance. In order to improve the propagation of the pdf through the nonlinear functions, the unscented Kalman filter (UKF) and the cubature Kalman filter (CKF) are proposed, which are described in section 2.1.3.2 and 2.1.3.3, respectively.

2.1.3.2 Unscented Kalman Filter

The unscented Kalman filter (UKF) is a straightforward extension of the unscented transformation (UT) to the KF. The UT is a method for calculating the statistics of a random variable which undergoes a nonlinear transformation. Consider propagating a \( n_x \)-dimensional random variable \( x \) through a nonlinear function \( y = f(x) \). Assume \( x \) has a symmetric prior density \( \pi(x) \) with mean \( \bar{x} \) and covariance \( P_x \), within which the Gaussian is a special case. To calculate the statistics of \( y \), we form a set of \((2L + 1)\) samples points named sigma vectors and weights, \( \{ \mathcal{X}_n, \omega_n^{(m)}, \omega_n^{(c)} \}_{n=0}^{2L} \) according to the following:

\[
\begin{align*}
\mathcal{X}_0 &= \bar{x}, \\
\mathcal{X}_n &= \bar{x} + \left( \sqrt{(L + \lambda)}P_x \right)_n, \quad n = 1, \ldots, L, \\
\mathcal{X}_n &= \bar{x} - \left( \sqrt{(L + \lambda)}P_x \right)_n, \quad n = L + 1, \ldots, 2L, \\
\omega_0^{(m)} &= \frac{\lambda}{L + \lambda}, \\
\omega_0^{(c)} &= \frac{\lambda}{L + \lambda} + 1 - \alpha^2 + \beta, \\
\omega_n^{(m)} &= \frac{1}{2(L + \lambda)}, \quad n = 1, \ldots, 2L.
\end{align*}
\]  

(2–12)

(2–13)

where \( L \) is set as \( n_x \), and \( \lambda = \alpha^2(L + k) - L \) is a scaling parameter. The constant \( \alpha \) determines the spread of the sigma points around \( \bar{x} \), and is usually set to a small positive value (e.g., \( 1 \leq \alpha \leq 10^{-4} \)). The constant \( k \) is a second scaling parameter, which is usually set to \( 3 - L \), and \( \beta \) is used to incorporate prior knowledge of the distribution of
x (for Gaussian distributions, $\beta = 2$ is optimal). $\left(\sqrt{(L+\lambda)P_x}\right)_n$ denotes the $i$th column of the matrix square root.

The sigma points and weights satisfy the following moment-matching conditions:

$$\bar{x} = \sum_{n=0}^{2L} \omega_n^{(m)} x_n,$$
$$P_x = \sum_{n=0}^{2L} \omega_n^{(c)} (x_n - \bar{x}) (x_n - \bar{x})^T.$$ (2–14)

These sigma vectors are propagated through the nonlinear function

$$y_n = f(x_n), \quad n = 0, \ldots, 2L,$$ (2–15)

and the mean and covariance for $y$ can also be approximated as

$$\bar{y} \approx \sum_{n=0}^{2L} \omega_n^{(m)} y_n,$$
$$P_y \approx \sum_{n=0}^{2L} \omega_n^{(c)} (y_n - \bar{y}) (y_n - \bar{y})^T.$$ (2–16)

Using these sigma vectors $\{x\}_{n=0}^{2L}$ and $\{y\}_{n=0}^{2L}$, we can estimate the cross-covariance, and use them to implement the Kalman filter. The algorithm of UKF is summarized in Algorithm 2-3. The details can be found in [33–37].

2.1.3.3 Cubature Kalman Filter

Similarly to the UKF, the CKF is another approximate Bayesian filter built in the Gaussian domain, but uses a completely different set of deterministic weighted points, the cubature points $\{\xi_n\}$, which rests on applying a numerical method known as the cubature rule [40, 41]. The cubature points are located at the intersection of the sphere and its axes. When the hidden state $x_i \in \mathbb{R}^{n_x}$, the cubature points and weights are defined as

$$\xi_n = \begin{cases} \sqrt{n_x} e_n & n = 1, 2, \ldots, n_x \\ -\sqrt{n_x} e_{n-L} & n = n_x + 1, n_x + 2, \ldots, 2n_x \end{cases}.$$ (2–17)
Algorithm 2-3: Unscented Kalman filter

Initialization: For $i = 0$, set
\[
\hat{x}_0 = E[x_0] \\
P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T]
\]

Computation: For $i = 1, 2, \ldots$, setting $L = n_x$

calculate the sigma points and weights
\[
\mathcal{X}_{i-1} = [\hat{x}_{i-1} \quad \hat{x}_{i-1} + \gamma \sqrt{P_{i-1}} \quad \hat{x}_{i-1} - \gamma \sqrt{P_{i-1}}]
\]
calculate the weights $\left\{\omega_n^{(m)}, \omega_n^{(c)}\right\}_{n=0}^{2L}$ according to (2–13)

**Time Update**
\[
\mathcal{X}_{il}^{*} = f(\mathcal{X}_{i-1}, u_{i-1}), \\
\hat{x}_i = \sum_{n=0}^{2L} \omega_n^{(m)} \mathcal{X}_{n, il}^{*} \\
P_i = \sum_{n=0}^{2L} \omega_n^{(c)} \left( \mathcal{X}_{n, il}^{*} - \hat{x}_i^n \right) \left( \mathcal{X}_{n, il}^{*} - \hat{x}_i^n \right)^T + Q_{i-1}
\]
regenerate the sigma points and weights
\[
\mathcal{X}_{i} = [\mathcal{X}_{il}^{*} \quad \mathcal{X}_{0i}^{*} + \gamma \sqrt{P_{i-1}} \quad \mathcal{X}_{0i}^{*} - \gamma \sqrt{P_{i-1}}]
\]
setting $L \rightarrow 2L$

recalculate the weights $\left\{\omega_n^{(m)}, \omega_n^{(c)}\right\}_{n=0}^{2L}$ according to (2–13)
\[
\mathcal{Y}_{i}^{*} = h(\mathcal{X}_{i}^{*}), \\
\hat{y}_i = \sum_{n=0}^{2L} \omega_n^{(m)} \mathcal{Y}_{n, i}^{*}
\]

**Measurement Update**
\[
P_{y_i,y_i} = \sum_{n=0}^{2L} \omega_n^{(c)} \left( \mathcal{Y}_{n, il}^{*} - \hat{y}_i^n \right) \left( \mathcal{Y}_{n, il}^{*} - \hat{y}_i^n \right)^T + R_{i-1}
\]
\[
P_{x_i,y_i} = \sum_{n=0}^{2L} \omega_n^{(c)} \left( \mathcal{X}_{n, il}^{*} - \hat{x}_i^n \right) \left( \mathcal{Y}_{n, il}^{*} - \hat{y}_i^n \right)^T
\]
\[
G_i = P_{x_i,y_i} P_{y_i,y_i}^{-1}
\]
\[
\hat{x}_i = \hat{x}_i^n + G_i (y_i - \hat{y}_i^n)
\]
\[
P_i = P_i^n - G_i P_{y_i,y_i} G_i^T
\]
where $\gamma = \sqrt{L + \lambda}$

\[
\omega_i = \frac{1}{n_x}
\]  

(2–18)

where $e_n$ is $\begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 1 \end{bmatrix}$, here 1 is the $n$th component.

Using these cubature points, (2–11) can be calculated as

\[
I_N(f) = \int_{\mathbb{R}^{nx}} f(x) \mathcal{N}(x; 0, I) dx \approx \sum_{n=1}^{2n_x} \omega_n f(\zeta_n)
\]  

(2–19)

With the cubature points, we can present the CKF algorithm in Algorithm 2-4. More details of the CKF algorithm are discussed in [38, 39].
Algorithm 2-4: Cubature Kalman filter

Initialization: For $i = 0$, set
\[
\hat{x}_0 = E[x_0] \\
\mathbf{P}_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T]
\]

Computation: For $i = 1, 2, \ldots$,

**Time Update**

Factorize $\mathbf{P}_{i-1} = \mathbf{D}_{i-1}\mathbf{D}_{i-1}^T$.
Evaluate the cubature points ($n = 1, 2, \ldots, m$)
\[
\mathbf{X}_{n,i-1} = \mathbf{D}_{i-1}\xi_n + \hat{x}_{i-1}, \quad \text{where} \quad m = 2n_x
\]
Evaluate the propagate cubature points ($n = 1, 2, \ldots, m$)
\[
\mathbf{X}^*_{n,i} = f(\mathbf{X}_{n,i-1}, u_{i-1})
\]
Estimate the predicted state
\[
\hat{x}_i^- = \frac{1}{m} \sum_{n=1}^{m} \mathbf{X}^*_{n,i}
\]
Estimate the predicted error covariance
\[
\mathbf{P}^-_i = \frac{1}{m} \sum_{n=1}^{m} \mathbf{X}^*_{n,i} \mathbf{X}^*_n - \hat{x}_i^- \hat{x}_i^-^T + \mathbf{Q}_{i-1}
\]

**Measurement Update**

Factorize $\mathbf{P}^-_i = \mathbf{D}^-_i\mathbf{D}^-_i^T$.
Evaluate the cubature points ($n = 1, 2, \ldots, m$)
\[
\mathbf{X}_{n,i}^- = \mathbf{D}^-_{i-1}\xi_n + \hat{x}_i^-,
\]
Evaluate the propagated cubature points ($n = 1, 2, \ldots, m$)
\[
\mathbf{Y}_{n,i}^- = h(\mathbf{X}_{n,i}, u_{i})
\]
Estimate the predicted measurement
\[
\hat{y}_i^- = \frac{1}{m} \sum_{n=1}^{m} \mathbf{Y}_{n,i}^-
\]
Estimate the innovation covariance matrix
\[
\mathbf{P}_{yy}^- = \frac{1}{m} \sum_{n=1}^{m} \mathbf{Y}_{n,i}^- \mathbf{Y}_{n,i}^-^T - \hat{y}_i^- \hat{y}_i^-^T + \mathbf{R}_i
\]
Estimate the cross-covariance matrix
\[
\mathbf{P}_{xy}^- = \frac{1}{m} \sum_{n=1}^{m} \mathbf{X}_{n,i}^- \mathbf{Y}_{n,i}^-^T - \hat{x}^-_i \hat{y}^-_i^T
\]
Estimate the Kalman gain
\[
\mathbf{G}_i = \mathbf{P}_{xy}^- \mathbf{P}_{yy}^-^{-1}
\]
Estimate the update state
\[
\hat{x}_i = \hat{x}_i^- + \mathbf{G}_i (y_i - \hat{y}_i^-)
\]
Estimate the corresponding error covariance
\[
\mathbf{P}_i = \mathbf{P}_i^- - \mathbf{G}_i \mathbf{P}_{yy} \mathbf{G}_i^T
\]

There are two basic properties of an error covariance matrix: i) symmetry and ii) positive definiteness, which are preserved in each update cycle. However, in practice, because of errors introduced by arithmetic operators performed on finite word-length digital computers, these two properties are often lost. In order to avoid this problem, a square-root version of the CKF is proposed, named square-root cubature Kalman filter (SCKF). The algorithm of SCKF is given in APPENDIX A. This is exactly the algorithm
we applied in later chapters. But for simplicity, we still name it as CKF if no confusion occurs.

2.1.4 Dual Kalman Filter

The (nonlinear) Kalman filter not only provides an efficient method for generating approximate maximum-likelihood estimates of the state of a discrete-time dynamical system, but also involves estimating the parameters of a nonlinear model given clean training data of inputs and outputs. In some cases, we need to estimate the state of a system with a parameterized model, which is not known exactly. When the noiseless state is not available, a dual estimation approach is required, in which both the states of the dynamical system and its parameters are estimated simultaneously, given only noisy observation.

To be more specific, we consider the problem of learning both the hidden states $x_i$ and parameters $w$ of a discrete-time nonlinear dynamical system, modeled as follows:

$$x_{i+1} = f_i(x_i, u_i, w) + n_i$$  \hspace{1cm} (2–20)

$$y_i = h_i(x_i, u_i, w) + v_i$$  \hspace{1cm} (2–21)

where $w$ is the parameter in the state transition and measurement functions, which is used to determine the system model.

Essentially, to estimate both hidden states and parameters, two (nonlinear) Kalman filters are run concurrently. At each time step, a (nonlinear) Kalman state filter estimates the state using the current model estimate $\hat{w}_i$, while the other (nonlinear) Kalman filter estimates the parameters using the current estimate $\hat{x}_i$. The system is shown schematically in Figure 2-1 [42–46].

2.1.5 Adaptive filters

Reconsider the dynamical model in (1–1) and (1–2), where we fix the hidden state $x_i$, but assume it is still unobservable. In addition, the measurement is equal to the inner product between the hidden state $x_i$ and input $u_i$. Under such assumption we have the
We can apply a linear adaptive filter to solve this kind of problem. Adaptive filters are self-designing filters, which rely on a recursive algorithm to adapt their weights (the hidden state \( x \)). Adaptive filters are commonly classified as linear or nonlinear. If its input-output map obeys the principle of superposition, the filter is said to be linear. Otherwise, the filter is nonlinear.

For linear adaptive filters, basically, there are two distinct approaches for deriving recursive algorithms for their operation. The first approach is the stochastic gradient approach for instance the least mean squares algorithm (LMS); the second approach is based on the method of least squares, for instance the recursive least squares algorithm (RLS). The former is simple but is sensitive to eigenvalue spread of the input, which makes convergence slow; the latter is computationally more expensive, but follows the Wiener solution at each iteration. Because of the close relationship between RLS and KF, here we just focus on the RLS algorithm and its extension.
2.1.5.1 Recursive Least Squares

With a sequence of training data \( \{u_j, y_j\}_{j=1}^i \) up to time \( i \), the RLS algorithm estimates the weight \( x_i \) by minimizing the following cost

\[
\min_{x_i} \sum_{j=1}^{i} |y_j - u_j^T x_i|^2 + \lambda \|x_i\|^2
\]

where \( u_j \) is the \( n_u \times 1 \) regressors input, \( y_j \) is the desired response (which is assumed a scalar here, but extension to multi-dimensional output is quite straightforward), and \( \lambda \) is the regularization parameter. The advantage of the RLS algorithm is to be able to calculate the weight \( x_i \) recursively from the previous estimate \( x_{i-1} \) without solving (2–23) directly. The standard RLS is summarized in Algorithm 2-5, and more details can be found in [1, 2].

<table>
<thead>
<tr>
<th>Algorithm 2-5: Recursive Least Squares (RLS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization:</strong> for ( i = 0 ), set</td>
</tr>
<tr>
<td>( x_0 = 0 ), ( P_0 = \lambda^{-1} I_m )</td>
</tr>
<tr>
<td>where ( m = n_u ), ( I_m ) is ( m \times m ) identical matrix</td>
</tr>
<tr>
<td><strong>Iterate:</strong> for ( i \geq 1 )</td>
</tr>
<tr>
<td>( r_{ei} = 1 + u_i^T P_{i-1} u_i )</td>
</tr>
<tr>
<td>( G_i = P_{i-1} u_i / r_{ei} )</td>
</tr>
<tr>
<td>( e_i = y_i - u_i^T x_{i-1} )</td>
</tr>
<tr>
<td>( x_i = x_{i-1} + G_i e_i )</td>
</tr>
<tr>
<td>( P_i = [P_{i-1} - P_{i-1} u_i u_i^T P_{i-1} / r_{ei}] )</td>
</tr>
</tbody>
</table>

The RLS estimation may be viewed as a special case of the Kalman algorithm. A distinguishing feature of the Kalman filter is the notion of state, which provides a dynamical description of the system at a specific instant of time. Generally speaking, adaptive filters can be treated with a constant hidden state \( x \), and the optimal weights of the filters, which need to be learned from the training data as \( \{u_i, y_i\} \) at the \( ith \) iteration. But there is an exception: the extended recursive least squares algorithm (Ex-RLS), which has state model and variant hidden states. Because of this, the Ex-RLS algorithm has much closer relationship with the Kalman filter.
2.1.5.2 Extended Recursive Least Squares

Although the RLS algorithm has a faster rate of convergence than the least mean squares (LMS) algorithm [1, 2], another well-known linear regression algorithm in a non-stationary environment, the LMS algorithm exhibits a better tracking behavior than the RLS algorithm.

In order to improve the tracking behavior of the RLS algorithm, the Ex-RLS was developed by Haykin [3] based on the state-space model below:

\[ x_{i+1} = F_i x_i + n_i \]
\[ y_i = u_i^T x_i + v_i \]  
(2–24)

where \( F_i \) is known transition matrix, \( \{y_i\}_{i=0}^{N} \) and \( \{u_i\}_{i=0}^{N} \) are \( N + 1 \) desired response and inputs, respectively. For given positive-definite matrices \( \{\Pi_0, \bar{Q}_i, \bar{R}_n\} \), and an initial guess \( \bar{x}_0 \), we pose the problem of estimating the initial state vector \( x_0 \) and the signals \( n_0, n_1, \ldots, n_N \) in a regularized least-squares manner by solving

\[ \min_{\{x_0, x_1, \ldots, x_N\}} J(x_0, n_1, \ldots, n_N) \]  
(2–25)

subject to the state-equation constraint

\[ x_{i+1} = F_i x_i + n_i \]  
(2–26)

Here cost function \( J \) is quadratic in its arguments and is given by

\[
J = (x_0 - \bar{x}_0)^T \Pi_0^{-1} (x_0 - \bar{x}_0) + \sum_{n=0}^{N} n_i^T \bar{Q}_i^{-1} n_i + \sum_{n=0}^{N} (y_i - u_i^T x_i)^T \bar{R}_i^{-1} (y_i - u_i^T x_i)
\]  
(2–27)
The solution of (2–27) shown in [3] leads to an iterative procedure that provides recursive estimates of the successive weight vectors \(x_i\), denoted by \(\hat{x}_i\). The algorithm of the Ex-RLS is summarized in Algorithm 2-6. More details can be found in [1, 2].

### Algorithm 2-6: Extended Recursive Least Squares (Ex-RLS)

<table>
<thead>
<tr>
<th>Initialization: for (i = 0), set (x_0 = 0, P_0 = \Pi_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterate: for (i \geq 1)</td>
</tr>
<tr>
<td>(R_{e,i} = \hat{R}<em>i + u_i^T P</em>{i-1} u_i)</td>
</tr>
<tr>
<td>(G_i = F_{i-1} P_{i-1} u_i / re_i)</td>
</tr>
<tr>
<td>(e_i = y_i - u_i^T \hat{x}_{i-1})</td>
</tr>
<tr>
<td>(\hat{x}<em>i = F</em>{i-1} \hat{x}_{i-1} + G_i e_i)</td>
</tr>
<tr>
<td>(P_i = F_{i-1} P_{i-1} F_{i-1}^T + Q_{i-1} - G_i R_{e,i} G_i^T)</td>
</tr>
</tbody>
</table>

Actually, this algorithm can be regarded as the Kalman filter mentioned in the previous subsection with the following (statistical) assumptions on the noise sequences: First, \(\{n_i\}\) and \(\{v_i\}\) are both assumed zero-mean white noise sequences with covariance matrices \(\bar{Q}\) and \(\bar{R}\); Second, the initial state-vector \(x_0\) is assumed random with mean \(\bar{x}_0\), and covariance matrix \(\Pi_0\); Third, the random variables \(\{n_i, v_i (x_0 - \bar{x}_0)\}\) are assumed uncorrelated. Simply put, the Ex-RLS algorithm solves a deterministic estimation problem, unlike the Kalman filter that solves a stochastic estimation problem. As long as we treat the weight matrices \(\bar{Q}\) and \(\bar{R}\) in the cost function (2–27) as the covariance matrices \(Q\) and \(R\) to describe the noises, we can utilize the Kalman filter to solve the Ex-RLS problem.

### 2.2 Algorithms in RKHS

In this section, we review some algorithms that are developed in the reproducing kernel Hilbert space (RKHS). The RKHS is a function space, in which some nonlinear problems in the input space are represented as linear forms. Because of this property, some linear algorithms are extended as nonlinear algorithms, such as the celebrated support vector machine (SVM) [5–10], kernel principle component analysis (KPCA) [11, 12], and kernel independent component analysis (KICA) [13, 14]. Here, we just focus on some kernel versions of adaptive filters, including the kernel recursive least
square (KRLS) [16] and the extended kernel recursive least squares (Ex-KRLS) [17].

In addition, we present a kernel Kalman filter [56, 57], which derive a Kalman filter in a subspace of the RKHS.

### 2.2.1 Foundations of RKHS

In order to present the kernel method more clearly, we first provide some foundations of reproducing kernel Hilbert spaces (RKHS).

A reproducing kernel Hilbert space is a special Hilbert space associated with a kernel such that it reproduces (via an inner product) each function in the space [48–54]. Let $\mathcal{H}$ be a Hilbert space of real-valued functions on a domain $\mathcal{X}$, equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and a real-valued bivariate function $k(x, y)$ on $\mathcal{X} \times \mathcal{X}$. Any nonnegative definite bivariate function is a reproducing kernel because of the following fundamental theorem [49].

**Theorem 2.1** (Moore-Aronszajn). *Given any nonnegative definite function $k(x, y)$, there exists a uniquely determined (possibly infinite dimensional) Hilbert space $\mathcal{H}$ consisting of functions on $\mathcal{X}$ such that*

1. $\forall x \in \mathcal{X}, k(x, \cdot) \in \mathcal{H}$
2. $\forall x \in \mathcal{X}, \forall f \in \mathcal{H}, f(x) = \langle f, k(x, \cdot) \rangle_{\mathcal{H}}$.

*Then $\mathcal{H} := \mathcal{H}_k$ is a reproducing kernel Hilbert space associated with a kernel function $k(x, y)$. Property II is called the reproducing property of $k(x, y)$ in $\mathcal{H}_k$.*

According to the Mercer’s theorem [90], there are countably many nonnegative Eigenvalues $\{\lambda_i : i \in \mathbb{N}\}$ and corresponding orthonormal Eigenfunctions $\{\psi_i : i \in \mathbb{N}\} \in \mathcal{H}_k$ such that

$$
k(x, y) := \sum_{i \in \mathbb{N}} \lambda_i \psi_i(x) \psi_i(y)^T, \quad (x, y) \in \mathcal{X} \times \mathcal{X}
$$

(2–28)

where the primer $T$ denotes the transpose. The series above converges absolutely and uniformly on $\mathcal{X} \times \mathcal{X}$. We have a feature map $\varphi : \mathcal{X} \rightarrow \ell^2(\mathbb{N})$ at each $x \in \mathcal{X}$ and $j \in \mathbb{N}$ as

$$
\varphi(x)(j) := \sqrt{\lambda_j} \psi_j(x).
$$

(2–29)
Therefore, Mercer’s representation in (2–28) establishes for each \(x, y \in \mathcal{X}\) that
\[
k(x, y) = \langle k((x, \cdot), (y, \cdot)) \rangle_{\mathcal{H}_k} = \langle \varphi(x), \varphi(y) \rangle_{\ell^2(\mathbb{N})}.
\] (2–30)

It is easy to check that \(\ell^2(\mathbb{N})\) is essentially the same as the RKHS \(\mathcal{H}_k\) by identifying \(\varphi(x) = k(x, \cdot)\) [89]. By slightly abusing the notation, we do not distinguish \(\ell^2(\mathbb{N})\) and \(\mathcal{H}_k\) in this dissertation. Alternatively, \(k(x, \cdot)\) can be viewed as the feature map \(\varphi(x)\), which maps each \(x \in \mathcal{X}\) into the high dimensional (or infinite dimensional) space \(\mathcal{H}_k\), whose dimensionality \(n_k\) is determined by the number of strictly positive eigenvalues. For example, for the Gaussian kernel, which is defined as
\[
k(x, y) = \exp \left( -\frac{\|x - y\|^2}{2\sigma^2} \right) = \exp \left( -\varrho \|x - y\|^2 \right)
\] (2–31)
where \(\sigma\) is the kernel size and \(\varrho\) is the kernel parameter, the dimensionality of the feature space \(n_k\) is infinite. In other words, we can treat \(\varphi(x)\) as an element in \(\mathcal{H}_k\), a function, or an \(n_k \times 1\) column vector, especially, an infinite-dimensional vector for the Gaussian kernel. (2–30) is also known as “kernel trick”, which makes the inner product in the infinite dimensional space computable.

According to the Representer Theorem [87, 88], functions have the following representation
\[
f(\cdot) = \sum_{j \in \mathbb{N}_n} \alpha_j k(x_j, \cdot), \quad x_j \in \mathcal{X}.
\] (2–32)
where \(\mathbb{N}_n := \{1, 2, \ldots, n\}\) and \(\{\alpha_j : j \in \mathbb{N}_n\} \subset \mathbb{R}\) are parameters typically obtained from training data, \(\{x_j : j \in \mathbb{N}_n\}\). The Representer Theorem has widely applicability [52, 54].

Furthermore, if the kernel associated with the RKHS \(\mathcal{H}\) is universal [78], then given any prescribed compact subset \(\mathcal{X}_{\text{sub}}\) of \(\mathcal{X}\), any positive number \(\varepsilon\) and any function \(f \in C(\mathcal{X}_{\text{sub}})\), there is a function \(g \in K(\mathcal{X}_{\text{sub}})\) such that \(\|f - g\|_{\mathcal{X}_{\text{sub}}} \leq \varepsilon\), where \(C(\mathcal{X}_{\text{sub}})\) is the space of all continuous complex-valued functions from \(\mathcal{X}_{\text{sub}}\) to \(\mathbb{C}\) equipped with
maximum norm $\| \cdot \|_{\mathcal{X}_{sub}}$ and $K(\mathcal{X}_{sub})$ is the space defined by

$$K(\mathcal{X}_{sub}) := \text{span}\{ k(x, \cdot) : x \in \mathcal{X}_{sub} \}. \tag{2–33}$$

In other words, with an universal kernel, we can approximate any real-valued target function defined on a compact space arbitrarily well as the number of summands increases without bound. The Gaussian kernel is universal, therefore is widely applied in many kernel methods and will be applied in all algorithms proposed in this dissertation.

### 2.2.2 Kernel Recursive Least Squares

We consider a sequence of input and output samples $D_i = \{(u_1, y_1), \ldots, (u_i, y_i)\}$, arising from some unknown source. In the prediction problem, one attempts to find the best predictor $\hat{y}_i$ for $y_i$ given $D_{i-1} \cup \{u_i\}$. In this context, one is often interested in on-line applications, where the predictor is updated following the arrival of each new sample.

The KRLS algorithm assumes a functional form, e.g. $\hat{y}_i = f(u_i)$ and minimizes the cost function $J$

$$J = \min_f \left[ i \sum_{j=1}^i |y_j - f(u_j)|^2 + \lambda \|f(\cdot)\| \right]. \tag{2–34}$$

where $\lambda$ is regularization factor.

In the reproducing kernel Hilbert space (RKHS) denoted by $\mathcal{H}$, a space of functions, a function $f(\cdot)$ is expressed as an infinite dimensional vector, denoted by $x \in \mathcal{H}$, and the evaluation of the function $f(u)$ is expressed as the inner product between $x$ and $\varphi(u)$, as below:

$$f(u) = \langle x|\varphi(u) \rangle = x^T \varphi(u). \tag{2–35}$$

where $\varphi(\cdot)$ maps $u$ into $\mathcal{H}$ [48–50]. So the cost function is rewritten as

$$J = \min_x \left[ i \sum_{j=1}^i |y_j - x^T \varphi(u_j)|^2 + \lambda \|x\| \right]. \tag{2–36}$$
The KRLS algorithm solves the cost function (2–36) recursively and estimate \( x \) as a linear combination of \( \{ \varphi(u_j) \}_{j=1}^{i} \),
\[
x = \sum_{j=1}^{i} a_j \varphi(u_j) = \Phi_i a_i \tag{2–37}
\]
where
\[
\Phi_i = [\varphi(u_1), ..., \varphi(u_i)]
\]
and
\[
a_i = [a_1, ..., a_i]^T.
\]

The KRLS algorithm is summarized in Algorithm 2-7. The details can be found in [16, 17].

<table>
<thead>
<tr>
<th>Algorithm 2-7: Kernel Recursive Least Squares (KRLS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize: ( S_1 = (\lambda + k(u_1, u_1))^{-1} ), ( a_1 = S_1 y_1 )</td>
</tr>
<tr>
<td>Iterate for ( i &gt; 1 )</td>
</tr>
<tr>
<td>( h_i = [k(u_i, u_1), ..., k(u_i, u_{i-1})]^T )</td>
</tr>
<tr>
<td>( z_i = S_{i-1} h_i )</td>
</tr>
<tr>
<td>( r_i = \lambda + k(u_i, u_i) - z(i)^T h_i )</td>
</tr>
<tr>
<td>( S_i = r(i)^{-1} \begin{bmatrix} S_{i-1} r(i) + z(i) z(i)^T - z_i^T &amp; -z_i^T &amp; 1 \end{bmatrix} )</td>
</tr>
<tr>
<td>( e_i = y_i - h_i^T a_{i-1} )</td>
</tr>
<tr>
<td>( a_i = \begin{bmatrix} a_{i-1} - z_i r_i^{-1} e_i &amp; r_i^{-1} e_i \end{bmatrix} )</td>
</tr>
</tbody>
</table>

Although the KRLS can online approximate the underlying nonlinear function based on the training data set, the computation complexity \( O(i^2) \) increases with the number of training data. To reduce the time and space complexities, some online sparsification algorithms are proposed, including KRLS with approximate linear dependency (ALD), quantized KRLS (QKRLS) and so on.

2.2.2.1 KRLS-ALD

According to (2–37), the approximated underlying function \( \hat{f}(. \cdot) = x \) is the linear combination of mapped inputs \( \{ \varphi(u_j) \}_{j=1}^{i} \). To reduce the complexity, we need to reduce
the number of the mapped inputs. The idea is to sequentially test whether a new feature input is approximately linearly dependent on the dictionary vectors, which is a set of already selected feature inputs. If not, we add it to the dictionary. We denote the dictionary as \( \{ \varphi(\tilde{u}_j) \}_{j=1}^{m_{-1}} \) and new feature input as \( \varphi(u_i) \) at iteration \( i \).

To avoid adding the training sample to the dictionary, we need to find coefficients \( \tilde{c} = [\tilde{c}_1, \ldots, \tilde{c}_{m_{-1}}] \) satisfying the approximate linear dependence (ALD) condition

\[
\delta_i := \min_{\tilde{c}} \| \sum_{j=1}^{m_{-1}} \tilde{c}_j \varphi(\tilde{u}_j) - \varphi(u_i) \|^2 \leq \delta_{ALD}
\]  

(2–38)

where \( \delta_{ALD} \) is the ALD threshold parameter determining the level of sparsity. If the ALD condition in (2–38) holds, \( \varphi(u_i) \) can be approximated within a squared error by some linear combination of current dictionary members. Expanding (2–38), we have

\[
\delta_i = \min_{\tilde{c}} \left\{ \sum_{k,j=1}^{m_{-1}} \tilde{c}_k \tilde{c}_j k(\tilde{u}_k, \tilde{u}_j) - 2 \sum_{j=1}^{m_{-1}} \tilde{c}_j k(\tilde{u}_k, u_i) + k(u_i, u_i) \right\}
\]

(2–39)

where \( \tilde{\Phi}_i = [\varphi(\tilde{u}_1), \ldots, \varphi(\tilde{u}_{m_{-1}})] \) and \( \tilde{K}_i = \tilde{\Phi}_i^T \tilde{\Phi}_i \). Solving (2–39) yields the optimal \( \tilde{c} \), and the ALD condition

\[
\tilde{c}_i = \tilde{K}_i^{-1} \tilde{\Phi}_i^T \varphi(u_i)
\]  

(2–40)

\[
\delta_i = k(u_i, u_i) - \tilde{c}_i^T \tilde{\Phi}_i^T \varphi(u_i)
\]  

(2–41)

respectively. If \( \delta_i > \delta_{ALD} \), then we must expand the current dictionary and \( m_{-1} = m_{i} + 1 \). More details and discussion about the KRLS-ALD can be found in [16].

### 2.2.2.2 QKRLS

Unlike the KRLS-ALD algorithm, which throws away the input when the input is approximately linearly dependent on the dictionary vectors, the QKRLS algorithm quantizes and keep all inputs [84]. To implement the online vector quantization (VQ) for the KRLS algorithm, a simple online (sequential) VQ method is applied [83], in which the
Algorithm 2-8: Online Vector Quantization

Initialization: \( i = 1 \)
Select quantization size \( \varepsilon \) and initialize codebook \( C_1 = u_1 \).

Computation: \( i > 1 \)
1) Compute the distance between \( u_i \) and \( C_{i-1} \):
\[
\text{dis}(u_i, C_{i-1}) = \| u_i - C_{j^*} \|
\]
where \( j^* = \min_{1 \leq j \leq |c_i|} \| u_i - C_j \| \)
2) If \( \text{dis}(u_i, C_{i-1}) \leq \varepsilon \), keep the codebook unchanged:
\( C_i = C_{i-1} \)
and quantize \( u_i \) to the closest code vector:
\( Q[u_i] = C_{j^*} \).
3) Otherwise, update the codebook:
\( C_i = C_{i-1} - \frac{u_i}{\text{dis}(u_i, C_{i-1})} \), where \( \| \cdot \| \) denotes the Euclidean norm and \( C_{j^*} \) denotes the \( j \)th element of the codebook \( C_{i-1} \).

The codebook is trained sequentially from the input data. The VQ algorithm is presented in Algorithm 2-8.

Here \( Q[\cdot] \) denotes a vector quantization operator, which maps the input into \( N \) bins. In the \( n \)th bin, there is one input \( c_n \) which represents other inputs in the same bin. Therefore, using the online VQ method, we can quantize each new \( \varphi(u_i) \) to \( c_n \). Then, at iteration \( i \) we have the codebook \( C_i = [c_1, \ldots, c_N] \) representing \( N \) bins. The number of inputs in the \( n \)th bin is \( M_n \). Thus, we have \( \sum_{n=1}^{N} M_n = i \).

Actually, the QKRLS algorithm attempts to solve the following cost function recursively
\[
\min_{\Omega \in H} \left[ \sum_{j=1}^{i} (y_j - \Omega^T \varphi(Q[u_j]))^2 + \varsigma \| \Omega \|_H^2 \right] \tag{2-42}
\]
where \( \varsigma \) is the regularization term. The QKRLS is summarized in Algorithm 2-9.

Here, \( \Phi_i = [\varphi(c_1), \ldots, \varphi(c_N)] \), \( \Phi_i^T \Phi_i \), and \( \Lambda = \text{diag}[M_1, \ldots, M_N] \).

2.2.3 Extended Kernel Recursive Least Squares

In the previous subsection, the function \( f(\cdot) \), or we say \( x \in H \) is considered static. However, for a non-linear time variant system, the function should be time varying, as the weight vectors in the Ex-RLS algorithm. Therefore, like the Ex-RLS algorithm, a kernelized version of the Ex-RLS algorithm, named as extended kernel recursive least squares (Ex-KRLS) algorithm is proposed by Liu et al. [17].
Algorithm 2-9: Quantized Kernel Recursive Least Squares

**Initialization:** \( i = 1 \)
Select quantization size \( \varepsilon \), regularization term \( \varsigma \),
initialize \( C_1 = u_1, A_1 = 1 \), \( S_1 = \left[ k(u_1, u_1) + \varsigma \right]^{-1} \), and \( a_1 = S_1 y_1 \).

**Computation:** \( i > 1 \)
1) If \( \text{dis}(u_i, C_{i-1}) \leq \varepsilon \):
   Keep the codebook unchanged: \( C_i = C_{i-1} \),
   update the matrix \( A_i \):
   \( A_i = A_{i-1} + \xi_i \xi_i^T \),
   update \( S_i \) and \( a_i^* \):
   \( a_i^* = a_{i-1} + \frac{S_{i-1} y_i}{1 + \xi_i^T S_{i-1}} \).
2) If \( \text{dis}(u_i, C_{i-1}) > \varepsilon \):
   update the codebook: \( C_i = C_{i-1}, u_i \),
   update the matrix \( A_i \):
   \( A_i = A_{i-1} \),
   update \( h_i, z_i, \Lambda_i, S_i, a_i^* \):
   \( h_i = k(C_{i-1}, u_i), \ldots, k(C_{i-1}, u_i) \) \( T \),
   \( z_i = S_{i-1} h_i, z_{\Lambda, i} = S_{i-1} A_{i-1} h_i \),
   \( r_i = \varsigma + k(u_i, u_i) - h_i^T z_{\Lambda, i} \),
   \( S_i = r_i^{-1} \left[ S_{i-1} r_i + z_{\Lambda, i} z_{\Lambda, i}^T - z_{\Lambda, i}^T 1 \right] \),
   \( a_i^* = \left[ a_{i-1}^* - z_{\Lambda, i} r_i^{-1} e_i \right] \),
where \( e_i = y_i - h_i^T a_{i-1}^* \).

In [17] the Ex-KRLS algorithm was derived as a general nonlinear state-space model in RKHS, according to the following theorem.

**Theorem 2.2.** *(Theorem 1 in [17])* Assume a general nonlinear state-space model

\[
\begin{align*}
  s_{i+1} &= g(s_i) \\
  d_i &= h(u_i, s_i) + v_i
\end{align*}
\]

(2–43)

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

\[
x(s_{i+1}) = Ax(s_i)
\]

\[
d_i = \varphi(u^T)x(s_i) + v_i
\]

and

\[
\varphi(u_i)^T \varphi(u_j) = k(u_i, u_j)
\]

with \(k(\cdot, \cdot)\) a suitable kernel function.

The proof is given in [17].

Using this theorem, the Ex-KRLS algorithm is developed based on two special cases of the state transition operators. First, the state transition operator is assumed as \(A = \alpha I\) for the tracking model where \(\alpha > 0\) is a scaling factor and \(I\) is an identity operator. Second, the state transition operator is constructed with finite rank assumption. In [17] only the first case of the state transition operator is implemented. Therefore, we will only focus on the first assumption, which results in the following tracking model:

\[
x_{i+1} = \alpha x_i + n_i
\]

\[
d_i = \varphi(u_i)^T x_i + v_i.
\]

where we use \(x_i\) instead of \(x(s_i)\) for short and \(n_i\) is state noise in the RKHS. It is noteworthy that this noise is omitted in Theorem 1, since the noise term makes the proof very complicated. The state-space model in the RKHS is supposed to solve the following problem:

\[
s_{i+1} = \tilde{a}s_i + \tilde{n}_i
\]

\[
d_i = h(u_i, s_i) + v_i
\]
Algorithm 2-10: Extended Kernel Recursive Least Squares for the Tracking Model

Initialize $i = 1$
\[ a_{1} = \alpha d_{1} \]
\[ \rho_{1} = \lambda \beta / (\|\alpha\|^{2} \beta + \lambda q) \]
\[ Q_{1} = \frac{[\beta \lambda + k(u_{1}, u_{1})][\|\alpha\|^{2} \beta + \lambda q]}{\|\alpha\|^{2}} \]

Iterate $i = 2, 3, ...$
\[ h_{i} = [k(u_{i}, u_{1}), \ldots, k(u_{i}, u_{i-1})]^{T} \]
\[ z_{i} = Q_{i-1} h_{i} \]
\[ r_{i} = \beta \rho_{i-1} + k(u_{i}, u_{i}) - h_{i}^{T} z_{i} \]
\[ e_{i} = d_{i} - h_{i}^{T} a_{i-1} \]
\[ a_{i} = \alpha \begin{bmatrix} a_{i-1} - z_{i} r_{i-1} e_{i} \\ r_{i-1} e_{i} \end{bmatrix} \]
\[ \rho_{i} = \frac{\rho_{i-1}}{\|\alpha\|^{2} \beta + \lambda q \rho_{i-1}} \]
\[ Q_{i} = \frac{[\|\alpha\|^{2}]}{r_{i}([\|\alpha\|^{2} \beta + \lambda q \rho_{i-1}])} \begin{bmatrix} Q_{i-1} r_{i} + z_{i} z_{i}^{T} & -z_{i} \\ -z_{i}^{T} & 1 \end{bmatrix} \]

where $\tilde{\alpha}$ is a scalar which is very close to 1 and $\tilde{n}_{i}$ is the state processing noise in the state space.

Then, the Ex-KRLS algorithm for the tracking model is proposed by implementing the Ex-RLS algorithm [1] in the RKHS, which actually minimizes the following least squares cost function in RKHS:

\[
\begin{align*}
&\min_{\{x_{i}, n_{1}, \ldots, n_{N}\}} \left[ \sum_{i=1}^{N} \beta^{N-i} \|d_{i} - \varphi(u_{i})^{T} x_{i}\|^{2} \\
&+ \lambda \beta^{N} \|x_{1}\|^{2} + q^{-1} \sum_{i=1}^{N} \beta^{N-i} \|n_{i}\|^{2} \right],
\end{align*}
\]

subject to $x_{i+1} = \alpha x_{i} + n_{i}$

where $\beta$ is the exponential weighting factor on the past data, $\lambda$ is the regularization parameter to control the initial state-vector norm and $q$ provides trade-off between the modeling variation and measurement disturbance. The Ex-KRLS is summarized in Algorithm 2-10. The details can be found in [17].

2.2.4 Revisiting the Ex-KRLS Algorithm

In this section we re-think the Theorem 2.2 and the Ex-KRLS algorithm. After that three important statements are given. The first one is about the existence of the state
transition operator \( A \). The second one is about the value of the parameter \( \alpha \). Finally, this Ex-KRLS algorithm is connected with the Kalman filter algorithm.

### 2.2.4.1 Existence of State Transition Operator

With the kernel trick, the measurement function \( h(s_i, u_i) \) in (2–43) is expressed as the inner product in the RKHS as

\[
\begin{align*}
  h(s_i, u_i) &= \langle h(s_i, \cdot), \varphi(u_i) \rangle_{H_u} \\
  &= \langle x_i, \varphi(u_i) \rangle_{H_u}
\end{align*}
\]

where the function \( h(s_i, \cdot) \) is denoted by \( x_i \) or \( x(s_i) \), which is a parameterized function of \( s_i \). In addition, the dynamics of the function \( h(s_i, \cdot) \) are described by the operator \( A \) in (2–44). In [17], \( x_i \) and \( x_{i+1} \) are constructed, then the authors state that the operator \( A \) exists.

However, the existence of the parameterized functions cannot guarantee the existence of the operator \( A \). First, we give a counterexample. Considering the state transition function \( g(s) = s + 1 \), the measurement function \( h(u, s) = s^2 u \), and the current hidden state \( s_i = \frac{1}{2} \), we have \( s_{i-1} = -\frac{1}{2} \) and \( s_{i+1} = \frac{3}{2} \). Then the parameterized measurement functions are \( h(s_{i-1}, u) = \frac{1}{4} u \), \( h(s_i, u) = \frac{1}{4} u \), and \( h(s_{i+1}, u) = \frac{9}{4} u \). Because \( h(s_{i-1}, u) \) and \( h(s_i, u) \) are the same, one does not know the parameterized function at next step. Therefore, there exists no operator \( A \) in this case.

Then, we formally discuss the existence condition of this operator through constructing the operator \( A \). Observing (2–44), one can find that the operator \( A \) just describes the dynamics of \( x(s_i) \), but does not describe the dynamics of the hidden state \( s_i \) explicitly. If we want to describe the dynamics of the hidden state \( s_i \) explicitly in the model, we have to construct the operators that specify the state and measurement
models in the RKHS:

\[
\phi(s_{i+1}) = G\phi(s_i) \tag{2–51}
\]

\[
y_i = \langle H\phi(s_i), \varphi(u_i) \rangle_{H_u} \tag{2–52}
\]

where \( \phi(s_i) \in \mathcal{H}_s \) and \( \varphi(u_i) \in \mathcal{H}_u \), \( G \) and \( H \) are both operators, mapping \( \mathcal{H}_s \mapsto \mathcal{H}_s \) and \( \mathcal{H}_s \mapsto \mathcal{H}_u \), respectively. It is noteworthy that \( \mathcal{H}_u \) and \( \mathcal{H}_s \) are different RKHS. Comparing \((2–45)\) and \((2–52)\), one can find that

\[
x_i = H\phi(s_i) \in \mathcal{H}_u \tag{2–53}
\]

and

\[
x_{i+1} = H\phi(s_{i+1}) = HG\phi(s_i) \tag{2–54}
\]

The operators \( G \) and \( H \) are determined by the functions \( g \) and \( h \) in \((2–43)\), respectively. Actually, the existences of \( x_i \) and \( x_{i+1} \) have been proven by constructing the operators \( G \) and \( H \) in [17]. However, the existence of the operator \( A \) was not proven.

According to \((2–53)\) and \((2–54)\), the operator \( A \) exists, if and only if the mapping \( H : \phi(s_i) \mapsto x_i \) is injective, and the operator \( A \) should be

\[
A = HGT. \tag{2–55}
\]

Here, we denote the mapping from \( x_i \) to \( \phi(s_i) \) by \( T \). Moreover, we have

\[
\phi(s_i) = Tx_i = TH\phi(s_i) \tag{2–56}
\]

holding for all \( s_i \). It means

\[
TH = I \tag{2–57}
\]

In sum, the operator \( A \) exists if and only if \( T \) exists.
2.2.4.2 Value of State Transition Parameter

Let us consider the value of the parameter $\alpha$ in the tracking model under the assumption that the state transition operator exists. In such case, the state transition operator is assumed as $A = \alpha I$. Because of (2–55) we have

\[ \text{HGT} = \alpha I. \]  

(2–58)

Next, according to (2–57), the following equation is obtained.

\[ G = \alpha I. \]  

(2–59)

Then, substituting (2–59) into (2–51), we have

\[ \phi(s_{i+1}) = G\phi(s_i) = \alpha\phi(s_i). \]  

(2–60)

Furthermore, according to the property of the translation invariant kernels [86], including the Gaussian kernel, we know that the norm of $\phi(s)$ is constant, such as

\[ \|\phi(s_i)\|_{\mathcal{H}_s} = \|\phi(s_{i+1})\|_{\mathcal{H}_s} \]  

(2–61)

Finally, substituting (2–60) into (2–61), we have

\[ \alpha^2 = 1 \]  

(2–62)

Therefore, the parameter $\alpha$ should be 1, not a free parameter. This means that theoretically we cannot model the state transition operator as the form of $A = \alpha I$ for general cases. This state model can only be used for the identity state model where $\alpha$ is set as 1. The Ex-KRLS algorithm proposed in [17] is actually a random walk KRLS algorithm, which is described by the following state-space model in the RKHS:

\[ x_{i+1} = x_i + n_i, \]

\[ d_i = \phi(u_i)^T x_i + \nu_i. \]  

(2–63)
That is to say the problem described by the state space model in (2–48) cannot be solved by the Ex-KRLS algorithm, unless $\alpha$ is equal to 1.

### 2.2.4.3 Relationship with the Kalman Filter

Because of the close relationship between the Ex-RLS and the Kalman filter, it is easy to note that the Ex-KRLS algorithm can be explained as a Kalman filter in the RKHS. The cost function (2–49) can be rewritten as

$$
\begin{align*}
&\min_{\{x_1, n_1, \ldots, n_N\}} [x_1^T \Pi^{-1} x_1 + \sum_{i=1}^{N} n_i^T Q_i^{-1} n_i \\
&+ \sum_{i=1}^{N} (d_i - \varphi(u_i) x_i)^T R_i^{-1} (d_i - \varphi(u_i) x_i)]
\end{align*}
$$

subject to $x_{i+1} = x_i + n_i$ \hfill (2–64)

where $\Pi = \lambda^{-1} I$, $Q_i = q\beta^i I$ and $R_i = \beta^i$.

According to the arguments about the Ex-RLS and the Kalman filter in Section 12. A and 12. B in [1], the Ex-KRLS algorithm summarized in Algorithm 1 is equivalent to a Kalman filter in the RKHS, with the state space model defined in (2–63). For this Kalman filter in the RKHS, the initial posterior state covariance is $\Pi = \lambda^{-1} I$, state process noise $n_i \in H_u$ and measurement noise $v_i \in \mathbb{R}$ are both independent, zero mean, Gaussian processes with covariance $Q_i = q\beta^i I$ and $R_i = \beta^i$.

### 2.2.4.4 Conclusions about Ex-KRLS algorithm

In this section, we analyzed the EX-KRLS algorithm proposed in [17] and point out the following: First, the state transition operator $A$ in the RKHS does not exist in the general case. The proof of theorem 1 in [17] is not always correct. It only exists when the mapping from the hidden state $s_i$ to the parameterized measurement function $h(s_i, \cdot)$ is injective. Second, even if the operator $A$ exists and is assumed as $\alpha I$ for the tracking model, theoretically, the parameter $\alpha$ has to be set as 1, which is not a free parameter. Therefore, the random walk KRLS model is not a special case of this algorithm, but the only case. Furthermore, the general state transition operator should not be modeled in the form $\alpha I$ in the RKHS, which is very different from the Ex-RLS algorithm. Thus,
the problem which is described by (2–48) cannot be solved by this Ex-KRLS algorithm. Finally, the Ex-KRLS algorithm is connected with the Kalman filter and explained as a special Kalman filter in the RKHS with an identity state model.

### 2.2.5 Kernel Kalman Filter

As mentioned in the previous subsection, it is very challenging to estimate the measurement function of two arguments. Therefore, consider the measurement function of only one argument, the hidden state $x_i$. No input $u_i$ is considered in the problem.

There exists a kernel Kalman filter algorithm (KKF) proposed by L. Ralaivola and F. d’Alché Buc [56, 57] to tackle the nonlinear time series processing $\{y_i\}_{i=1}^T$, which is built in a high dimensional subspace generated by the training data $\{y_j^0\}_{j=1}^N$.

The idea of the KKF algorithm is described as following: First, obtain a set of orthonormal basis $B = [b_1, \cdots, b_l]$ from the $l$ principle axes of the mapped training data set $\{\varphi(y_j^0)\}_{j=1}^N$ by Kernel PCA (KPCA) [12] method; next, project the mapped measurement $\varphi(y_i)$ into the subspace $H_B$ with the basis $B$, where the subspace $H_B$ is spanned by $B$; then, construct state-space model in the subspace $H_B$ and implement the Kalman filter to estimate or predict the hidden state in such subspace. Finally, map the hidden state back to the input space.

More specifically, the state-space model of this algorithm is expressed as follows:

$$s_i^\varphi + 1 = Fs_i^\varphi + \mu_s^\varphi + \nu_s^\varphi$$  \hspace{1cm} (2–65)  

$$y_i^\varphi = s_i^\varphi + \mu_y^\varphi + \nu_y^\varphi$$  \hspace{1cm} (2–66)

where $F$ is a matrix or operator in $H_B \times H_B$; $s_i^\varphi$ is the hidden state and $y_i^\varphi$ is the projection of $\varphi(y_i)$ in the subspace $H_B$; $\mu_s^\varphi$ and $\mu_y^\varphi$ are both vectors in $H_B$; $\nu_s^\varphi$ and $\nu_y^\varphi$ are assumed to be zero-mean Gaussian noise vectors of covariances $\sigma_s^2 I$ and $\sigma_y^2 I$, respectively. One can find that the measurement model in the KKF algorithm is an identity function of the hidden state.
Because \( B \) is the orthogonal basis of the subspace \( \mathcal{H}_B \), \( F \), \( \mu_s^\varphi \) and \( \mu_y^\varphi \) can be expressed as \( BF^TB \), \( B\tilde{\mu}_s \) and \( B\tilde{\mu}_y \), respectively, where \( F \) is an \( l \times l \) matrix, and \( \tilde{\mu}_s \) and \( \tilde{\mu}_y \) are both \( l \times 1 \) vectors. Therefore, a set of kernel Kalman filtering and kernel smoothing equations can be derived from the traditional Kalman filter and Kalman smoother. Then, the model parameters \( \{\tilde{F}, \tilde{s}_0, \tilde{\mu}_s, \tilde{\mu}_y, \sigma_s, \sigma_y\} \) in the kernel Kalman filter and the Kalman smoother are approximated using an Expectation-Maximization (EM) algorithm \([58]\) from the training data. Finally, we find the hidden states in the input space by solving a PreImage problem \([57, 59]\). The details of the KKF algorithm can be found in \([56, 57]\).

In sum, although the kernel Kalman filter is implemented using the kernel map, it is not the real Kalman filter in the RKHS. It just expresses the Kalman equations in a higher dimensional subspace \( \mathcal{H}_B \subset \mathcal{H}_y \), where \( \mathcal{H}_y \) is the RKHS defined on the measurement domain. Moreover, because of the very complicated and time consuming off-line learning procedure of model parameters, these model parameters cannot be updated simultaneously, which results in a bad tracking performance for a time variant system, especially when the range of the training data cannot cover the test data.

### 2.3 Summary of Related Algorithms

In the previous sections, we review a series of related algorithms involved in dynamical model problems. To clearly present the framework of these algorithms, we summarize them in Figure 2-2. From the figure, one can find that the linear adaptive filter algorithms mentioned in the dissertation are all expanded to the RKHS as new nonlinear algorithms, except the KKF algorithm, which is just implemented in a high dimensional subspace of RKHS. Therefore, we intend to reformulate the Kalman filter in RKHS to obtain a new nonlinear Kalman filter algorithm. This is the goal of our research and is marked red in Figure 2-2. In the following chapters, we will present some algorithms that we develop along this research line.
Figure 2-2. Relationship between algorithms
3.1 Overview

In Chapter 2, we have presented the KRLS and Ex-KRLS algorithms. For both, the measurement functions are constructed in the feature space $\mathcal{H}$ in an inner product form; while from the input space point of view, the measurement functions are nonlinear, unknown in advance, and must be learned from the data. Compared with the KRLS algorithm, the Ex-KRLS algorithm has a more flexible state model in RKHS. However, as analyzed and discussed in Section 2.2.3, the Ex-KRLS is just a random walk KRLS algorithm, which cannot reflect the real and full system states. In order to have a better description of the time variant dynamic system, we have to use a non-trivial transition operator $A$. Although in [17], the algorithm has been derived based on a non-trivial transition operator obtained by subspace projection, how to estimate this non-trivial transition operator is still an open question.

Here we take a different and hybrid approach of building the state model in the input space and the observation model in RKHS. The solution describes the system states better, and still use the state vector $x_i$ to represent the known system state at time $i$. Similar to the KRLS and Ex-KRLS algorithms, the measurement function $h(\cdot, \cdot)$ is assumed unknown nonlinear and will be learned from the data in RKHS. The state space model is presented as

\[
\begin{align*}
    x_{i+1} & = f(x_i) + n_i, & \text{known model} \\
    y_i & = h(x_i, u_i) + v_i, & \text{unknown model}
\end{align*}
\]

where the state noise $n_i$ and $v_i$ are both in the original spaces. Once the unknown measurement function is obtained, we can use the (nonlinear) Kalman filter formulation to solve this problem just like the Ex-RLS algorithm mentioned before as long as we assume corresponding positive definite matrices $\{\Pi_0, Q, R\}$. It is noteworthy that the
measure function can also be a vector-value function \( h(\cdot, \cdot) \) with vector noise \( v_i \), which can be treated as vectors of scalar-value functions and learned like several scalar-value functions. Therefore, in the chapter we do not distinguish scalar-value and vector-value measurement functions, which are not denoted by \( h(\cdot) \) or \( h(\cdot) \).

Since the overall state space model is nonlinear, the nonlinear Kalman filter is necessary. In [47] we have proposed a similar algorithm, EKF-KRLS algorithm. The only difference here is that the measurement function is chosen with two arguments, the input \( u_i \) and the system state \( x_i \). Therefore, we first introduce the EKF-KRLS algorithm, then present the novel extended kernel recursive least squares algorithm, which is a combination of KRLS and nonlinear KF algorithms, named as the extended kernel recursive least squares based on the Kalman filter (Ex-KRLS-KF). Finally, the experiments and conclusion are presented.

### 3.2 Extended Kalman Filter-Kernel Recursive Least Squares Algorithm

The problem we want to solve using the extended Kalman filter-kernel recursive least squares (EKF-KRLS) algorithm can be modeled as

\[
\begin{align*}
x_{i+1} &= f(x_i) + n_i \\
y_i &= h(x_i) + v_i,
\end{align*}
\]

(3–3)

Here we assume the transition function \( f(\cdot) \) is known, but the nonlinear function \( h_i(\cdot) \) is unknown. The assumptions on noise \( n_i \) and \( v_i \) are the same as mentioned before.

In order to obtain a nonlinear observation model, it will be constructed in the RKHS \( \mathcal{H} \). The whole system is reformulated as

\[
\begin{align*}
x_{i+1} &= f(x_i) + n_i; \ (input \ space) \\
y_i &= \langle h, \phi(x_i) \rangle_{\mathcal{H}} + v_i \ (\mathcal{H} \ space)
\end{align*}
\]

(3–4)

We review the EKF algorithm in Algorithm 2-2 to derive the EKF-KRLS algorithm. Because the state model is in the input space, the things we need to consider are the
Kalman gain matrix $G_i$ and the error covariance update equations. In light of [1], the Kalman gain is defined as

$$G_i = E \left[ x_i+1 \tilde{e}_i^T \right] R_{\tilde{e},i}^{-1}$$  \hfill (3-5)

where $E[.]$ denotes the expectation operator, $\tilde{e}_i = y_i - \hat{y}_i = y_i - h(\hat{x}_i^-)$ and $R_{\tilde{e},i} = E \left[ \tilde{e}_i \tilde{e}_i^T \right]$. According to the orthogonality principle, we have

$$R_{\tilde{e},i} = E[\tilde{e}_i \tilde{e}_i^T] = E \left[ (h(x_i) - h(\hat{x}_i^-)) (h(x_i) - h(\hat{x}_i^-))^T \right] + E [v_i v_i^T]$$  \hfill (3-6)

$$E \left[ x_i+1 \tilde{e}_i^T \right] = F_i E \left[ x_i \tilde{e}_i^T \right] + E \left[ n_i \tilde{e}_i^T \right]$$  \hfill (3-7)

where $F_i = \frac{\partial f(x)}{\partial x} \big|_{x=\hat{x}_i}$. The terms $E \left[ x_i \tilde{e}_i^T \right]$ and $E \left[ n_i \tilde{e}_i^T \right]$ are given by

$$E \left[ x_i \tilde{e}_i^T \right] = E \left[ (x_i - \hat{x}_i^- + \hat{x}_i^-) \tilde{e}_i^T \right]$$  \hfill (3-8)

$$= E \left[ (x_i - \hat{x}_i^-) \tilde{e}_i^T \right] \text{ since } \hat{x}_i^- \perp \tilde{e}_i^T$$

$$= E \left[ (x_i - \hat{x}_i^-) (h(x_i) - h(\hat{x}_i^-)) \right] \text{ since } (x_i - \hat{x}_i^-) \perp v_i,$$

$$E \left[ n_i \tilde{e}_i^T \right] = E \left[ n_i (h(x_i) - h(\hat{x}_i^-)) + v_i \right]$$  \hfill (3-9)

$$= E \left[ n_i v_i^T \right] \text{ since } n_i \perp (h(x_i) - h(\hat{x}_i^-))^T$$

$$= 0 \text{ assume } n_i \perp v_i.$$

Like the error covariance $P_i^- = E[(x_i - \hat{x}_i^-)(x_i - \hat{x}_i^-)^T]$ in the Kalman filter algorithm, we also need to construct $E \left[ (x_i - \hat{x}_i^-) (x_i - \hat{x}_i^-)^T \right]$. We employ a first-order Taylor approximation of the nonlinear function $h(\cdot)$ around $\hat{x}_i^-$. Specifically, $h(x_i)$ is
approximated as follows

\[
h(x_i) = h(\hat{x}_i^- + (x_i - \hat{x}_i^-)) \\
\approx h(\hat{x}_i^-) + \frac{\partial h_i}{\partial \hat{x}_i^-} (x_i - \hat{x}_i^-). \tag{3-10}
\]

With the above approximate expression, we can approximate \(G_i\) by substituting from (3–6) to (3–7) into (3–5),

\[
G_i \approx P_i^- H_i^T [H_i P_i^- H_i^T + R_i]^{-1} \tag{3–11}
\]

where \(H_i = \frac{\partial h}{\partial \hat{x}_i^-}\). The error covariance update equation is the same with the approximated Kalman gain.

Once \(H_i\) is estimated at time \(i\), we can apply the Kalman filter algorithm to solve the prediction problem. Until now, the derivation is very similar to the EKF algorithm, except that the hidden state model is still linear and the function \(h(\cdot)\) is assumed unknown.

In order to obtain \(H_i = \frac{\partial h}{\partial \hat{x}_i^-}\), we will use the KRLS algorithm mentioned in Chapter 2 to estimate the function \(h(\cdot)\) based on the predicted hidden states \(\hat{x}_j (j \leq i)\). At every time step, the EKF algorithm estimates the state using the current estimated function \(\hat{h}_i(\cdot)\), while the KRLS algorithm estimates the unknown function \(\hat{h}_{i+1}(\cdot)\) using all available estimated hidden states. We concatenate these two algorithms to obtain a novel algorithm, EKF-KRLS algorithm, summarized in Algorithm 3-1.

Here \(a_0\) is an \(1 \times n_y\) vector if \(n_y > 1\), and \(a_i\) for \(i > 1\) is a \((i+1) \times n_y\) matrix.

Like the KRLS algorithm, \(\Phi_{i-1}\) is scaled by a forgetting factor \(\zeta (0 \ll \zeta \leq 1)\) to get \(\Phi_i\). The reason is that we estimate the measurement function \(h_i(\cdot)\) based on the estimated hidden states \(\{\hat{x}_j\}_{j=1}^i\). However, the hidden states are not trusted at the beginning of filtering. So we use the forgetting factor to get rid of the impact of wrong early estimates.

### 3.3 Formulation of the Novel Extended Kernel Recursive Least Squares

As discussed in section 2.2.3, it is not easy to build a state model in RKHS, which involves operator learning. To avoid this problem, we still maintain the measurement
Algorithm 3-1: EKF-KRLS

Initialize: for $i = 0$, set:

$\hat{x}_0 = E[x_0], \Phi_0 = [k(\hat{x}_0, \cdot)]$

$P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T]$

$Q(0) = (\lambda + k(\hat{x}_0, \hat{x}_0))^{-1}, a(0) = 1_{\mathcal{H}}$

Filtering for $i = 1, 2, ...$

State filtering (EKF)

$F_i = \frac{\partial f(x)}{\partial x}|_{x = \hat{x}_{i-1}}$

$\hat{x}_i^- = F_{i-1}\hat{x}_{i-1}$

$\hat{y}_i = a(i - 1)^T\Phi_{i-1}^T\varphi(\hat{x}_{i-1})$

$P_i^- = F_{i-1}P_{i-1}F_{i-1}^T + S_{i-1}$

$H_i = \left(\frac{\partial \Phi_{i-1}}{\partial \hat{x}_{i-1}}a(i - 1)\right)^T$

$G_i = P_i^-H_i^T\left[H_iP_i^-H_i^T + R_i\right]^{-1}$

$\hat{x}_i = \hat{x}_i^- + K_i(y_i - \hat{y}_i)$

$P_i = (I - K_iH_i)P_i^-$

Measurement function update (KRLS)

$\Phi_i = [\zeta\Phi_{i-1}, k(\hat{x}_i, \cdot)]$

$h(i) = \Phi_{i-1}^T\varphi(\hat{x}_i)$

$z(i) = Q(i - 1)h(i)$

$r(i) = \lambda + k(\hat{x}_i, \hat{x}_i) - z(i)^T h(i)$

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

$e(i) = y_i - a(i - 1)^T h(i)$

$a(i) = \begin{bmatrix} a(i-1) - r(i)^{-1}z(i)e(i)^T \\ r(i)^{-1}e(i)^T \end{bmatrix}$

Model in the input space, but learn the measurement function in RKHS. It is similar to the EKF-KRLS algorithm presented in the previous section. To deal with measurement function in (3–2) with two arguments input $u_i$ and hidden state $x_i$, we combine them as a new input $z_i = [u_i^T x_i^T]^T$ and express the measurement function in the RKHS in inner product form as

\[ h(x_i, u_i) = \tilde{h}(z_i) = \langle \tilde{h}(\cdot), \varphi(z_i) \rangle_{\mathcal{H}} \]  

(3–12)
Algorithm 3-2: Extended Kernel Recursive Least Squares with Kalman filter (Ex-KRLS-KF)

Initialization for $i = 0$:
- Initialize measurement function $h_0(\cdot, \cdot)$
- Set $\xi_0$, $\Pi_0$, $Q$ and $R$

Filtering for $i = 1, 2, \ldots$

State filtering (Nonlinear KF)
- Using any nonlinear Kalman filter to estimate the hidden state $x_i$ based on the initialized or approximated measurement function $h_{i-1}(\cdot, \cdot)$ and input $u_i$ and measurement $y_i$

Measurement function update (KRLS)
- Using KRLS to approximate and update the measurement function $h_{i-1}(\cdot, \cdot)$ based on all the available inputs $\{u_j\}_{j=1}^i$, outputs $\{y_j\}_{j=1}^i$, and estimated hidden states $\{\hat{x}_j\}_{j=1}^i$

Therefore, as long as we have corresponding measurements $\{y_i\}$, hidden states $\{x_i\}$ and inputs $\{u_i\}$, we can approximate the measurement function $h(\cdot, \cdot)$ using the KRLS algorithm.

It is observed that the EKF algorithm is not the unique choice for state filtering. Actually, any nonlinear Kalman filters described in Section 2.1.3 can be applied to estimate the hidden states. Now, the novel algorithm can be presented below:

In practice, for this algorithm the exponentially-weighted KRLS should be employed. The reason is that we estimate the measurement function $h(x_i, u_i)$ based on the estimated hidden states $\{\hat{x}_j\}_{j=1}^i$. However, the hidden states are not trustable at the beginning of filtering because of the incorrect initial measurement function. So we use the forgetting factor to attenuate the impact of incorrect early estimates. Furthermore, we can consider discarding some beginning estimated states or just computing several recent estimated states in a running window.

Although our algorithm is the combination of the nonlinear Kalman filter and KRLS algorithms, the computational complexity of the Ex-KRLS-KF is comparable with the KRLS and Ex-KRLS algorithm, which is equal to $O(i^2)$ at iteration $i$. Moreover, we
should also consider sparsification and approximate linear dependency (ALD) \cite{16,17} to restrict computational complexity.

In this algorithm, the measurement function is approximated based on the inputs, measurements and estimated hidden states. On the other hand, the hidden states are estimated based on the approximated measurement function. Actually, it is the information about the state model in (2–48) that is applied to approximate the unknown measurement function.

We use the KRLS algorithm to approximate the measurement function $h(\cdot, \cdot)$, which is a function of two arguments. However, we can also treat the function as a parameterized function of one argument input $u_i$. When treating the hidden state $x_i$ as the parameter of the function, we have

$$h(x_i, u_i) = h_{x_i}(u_i)$$  \hspace{1cm} (3–13)

which can be rewritten as the inner product in the RKHS

$$h(x_i, u_i) = \langle h_{x_i}(\cdot), \varphi(u_i) \rangle_H. \hspace{1cm} (3–14)$$

Therefore, we have the following two special cases:

By setting the state transition function $f(\cdot)$ as an identical function, we have the KRLS algorithm for the following random walk model:

$$x_{i+1} = x_i + n_i$$  \hspace{1cm} (3–15)

$$y_i = h_{x_i}(u_i) + v_i$$  \hspace{1cm} (3–16)

Furthermore, by setting the covariance matrix $Q$ as zero matrix, we have the KRLS algorithm.

The parameterized measurement function $h_{x_i}(\cdot)$ is the counterpart of $x_i$ in the Ex-KRLS algorithm, which is also unknown in advance. However, the transition behavior
of the function $h_{x_i}(\cdot)$, which is determined by its parameter $x_i$, is described better than by $x_i$ alone. Therefore this algorithm is more general.

### 3.4 Experiments and Results

In this section, two experiments are given to evaluate the tracking performance of the Ex-KRLS-KF algorithm. One is vehicle tracking, and the other is Rayleigh fading channel tracking. The performances of our algorithms are compared with other existing algorithms. To demonstrate the generality of the novel algorithm, the EKF and CKF algorithms are applied in our algorithm, respectively.

#### 3.4.1 Vehicle tracking

Because this algorithm is the combination of the nonlinear Kalman filter and the KRLS algorithm, we compare the tracking performances of our algorithm with the nonlinear Kalman filter and the KRLS algorithm to track the vehicle in a popular open surveillance dataset, PETS2001, which is available at [http://www.hitech-projects.com/euprojects/cantata/dataset/cantata/dataset.html](http://www.hitech-projects.com/euprojects/cantata/dataset/cantata/dataset.html). The celebrated extended Kalman filter (EKF) is selected as the nonlinear Kalman filter in this experiment.

Since our goal is to evaluate the tracking performance of these algorithms, we mark manually the vehicle which we want to track. The figures below show the frames and the trajectory of the vehicle.

![Figure 3-1. Trajectory of the vehicle with background](image)

In Figure 3-1 the red line is the trajectory of the right front light of the vehicle. One can see that the vehicle travels straight first, then backs up, and finally parks. The
Figure 3-2. Trajectory of the vehicle

The trajectory is presented alone in Figure 3-2. There are 410 frames in the surveillance video. In the figure the vehicle positions \( P(\epsilon, \eta) \) are in a Cartesian coordinate system. The kinematics of the vehicle can be modeled by the Ramachandra’s model:

\[
x_{i+1} = \begin{pmatrix}
1 & T & \frac{T^2}{2} & 0 & 0 & 0 \\
0 & 1 & T & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & T & \frac{T^2}{2} \\
0 & 0 & 0 & 0 & 1 & T \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
x_i + n_i, \quad (3-17)
\]

where the state of the model \( x = [\epsilon \ \dot{\epsilon} \ \eta \ \dot{\eta}]^T \); \( \epsilon \) and \( \eta \) denote position, and \( \dot{\eta} \) and \( \dot{\epsilon} \) denote velocities of the vehicle in the \( x \) and \( y \) directions, respectively; \( T \) is the time-interval between two consecutive frames; the process noise \( n_i \sim \mathcal{N}(0, Q) \) with a covariance matrix \( Q = \text{diag} [0 \ 0 \ a_\epsilon \ 0 \ 0 \ a_\eta] \). The scalar parameters \( a_\epsilon \) and \( a_\eta \) are related
to process noise intensities. \( n_i \) is the plant noise that perturbs the acceleration and accounts for both maneuvers and other modeling errors.

A radar is fixed at the origin of the plane and equipped to track the vehicle. Our measurements are the distance \( \rho \) between the vehicle and the origin \( P(0, 0) \) and the slope \( k \) with respect to the origin. The measurement equation is expressed as

\[
\begin{pmatrix}
\rho_i \\
k_i
\end{pmatrix} = \begin{pmatrix}
\sqrt{\varepsilon_i^2 + \eta_i^2} \\
\frac{\eta_i}{\varepsilon_i}
\end{pmatrix} + v_i \tag{3–18}
\]

where the measurement noise \( v \sim \mathcal{N}(0, \mathbf{R}) \) with a covariance matrix \( \mathbf{R} \).

For these three different algorithms, we have the following setting:

**Nonlinear Kalman filter**: Using the state model and measurement model given in (3–17) and (3–18), we employ the EKF algorithm to predict the next step measurement. The parameters are set as: \( a_\varepsilon = a_\eta = 1000 \) and \( \mathbf{R} = \text{diag}(\[1, 0.1\]) \).

**KRLS**: We use the KRLS algorithm to predict the next measurement based on the previous \( N \) measurements. For the 2-D vehicle tracking, actually we use \( 2N \) data to predict the distance and slope, respectively. The parameter \( N \) is set as 6 to obtain the best performance of the KRLS algorithm. We choose the Gaussian kernel in the KRLS algorithm. Here the kernel size is set as 100 through trials. The forgetting factor \( \beta \) is 0.85 to improve the tracking performance of the KRLS algorithm.

**Ex-KRLS-KF**: For the Ex-KRLS-KF algorithm we only need to know the transition matrix. The function \( h(\cdot) \) can be learned from the data. We choose the same transition matrix (3–17) as the EKF to track the vehicle.

Comparing (3–18) with (3–2), one can find that there is no input in the measurement model in this application. In order to employ the Ex-KRLS-KF algorithm, we can set all the inputs as constants. Actually, if we set all inputs as zeros, we have the EKF-KRLS algorithm \([47]\) in Section 3.2.
In the Ex-KRLS-KF algorithm, the unknown measurement function $h(\cdot, \cdot)$ is learned using the KRLS algorithm with the previously estimated hidden states, while the hidden states themselves are estimated by the previous approximated function $\hat{h}(\cdot, \cdot)$. The beginning estimated hidden states are not trustable. Therefore, we need a forgetting factor $0 < \beta < 1$ to make the current hidden states more important with larger weights. We also use the running window to control the updating of the function $h(\cdot)$, which means that we learn the current $h(\cdot)$ only based on the previous $m$ estimated hidden states. We set these parameters as $\beta = 0.64$ and $m = 35$. We set the parameters in the covariance matrix of process noise as $Q = \text{diag} \begin{bmatrix} 0 & 0 & 0 & a_{\xi} \\ a_{\eta} & 0 & 0 & a_{\eta} \end{bmatrix}$ where $a_{\xi} = a_{\eta} = 10^{-7}$, and the covariance matrix of measurement noise as $R = rI$ where $r = 0.1$. We also use the Gaussian kernel in the algorithm and the kernel size is set to be 1000.

All parameters above are chosen to obtain the best performances.

Because the ranges of the distance $\rho$ and the slope $k$ are quite different, we compare their performances separately. Considering that these algorithms have different convergence time, we compare their performances from the 50th frame to the end. Table 3-1 summarizes the prediction performances of the distance $\rho$ and the slope $k$.

### Table 3-1. MSE of distance $\rho$ and slope $k$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>EKF</th>
<th>KRLS</th>
<th>Ex-KRLS-KF</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance $\rho$</td>
<td>0.3180</td>
<td>0.3654</td>
<td>0.1709</td>
</tr>
<tr>
<td>slope $k$</td>
<td>0.0024</td>
<td>0.0001</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

In order to compare their performances more correctly and visually, we transform the distance $\rho$ and the slope $k$ to the position $P(\epsilon, \eta)$ using the equations below:

$$\epsilon = \frac{\rho}{\sqrt{1 + k^2}}.$$  \hspace{1cm} (3–19)

$$\eta = k\epsilon.$$  \hspace{1cm} (3–20)

The trajectories and errors are plotted in Figure 3-3 and Figure 3-4. The Table 3-2 summarizes the prediction performances of these three algorithms. The results are the MSE between predicted position and true position from the 100th frame to the
These results clearly show that the proposed algorithm has the fastest rate of convergence and the best tracking performance; and the superiority is very obvious.

Figure 3-3. Trajectories of the true position and the predictions of the EKF, KRLS, and Ex-KRLS-KF algorithms

Table 3-2. MSE of position

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>EKF</th>
<th>KRLS</th>
<th>Ex-KRLS-KF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>1.7130</td>
<td>1.0156</td>
<td>0.5467</td>
</tr>
</tbody>
</table>

3.4.2 Rayleigh channel tracking

We consider the problem of tracking a nonlinear Rayleigh fading multipath channel and compare the performance of our novel Ex-KRLS-KF algorithm with some existing algorithms, including the normalized LMS, RLS, Ex-RLS, KRLS and Ex-KRLS algorithms. In this experiment, we employ the CKF in our algorithm instead of the EKF in the Ex-KRLS-KF algorithm.

The nonlinear Rayleigh fading multipath channel employed here is the cascade of a traditional Rayleigh fading multipath channel and a saturation nonlinearity. In the Rayleigh multipath fading channel, the number of the paths is chosen as $M = 5$, the
same maximum Doppler frequency $f_D$ is used for each channel and the sampling rate $T_s = 0.8 \mu s$. According to [1], the maximum Doppler frequency $f_D$ is related to the speed of the mobile user $v$, and to the carrier frequency $f_c$, as $f_D = vf_c/c$, where $c$ denotes the speed of light, $c = 3 \times 10^8 m/s$. Assume a carrier frequency of $f_c = 900 MHz$, then verify that the Doppler frequency that corresponds to a vehicle moving at the speed of $v = 120 Km/h$ is $100 Hz$ (so it is a slow fading channel with the same fading rate for all the paths).

The Rayleigh fading multipath channel is simulated by Matlab Communication System Toolbox [91]. The signal is a unit power white Gaussian process which is sent through this channel. Then the real part of the output of the Rayleigh channel is corrupted with additive white Gaussian noise with variance $\sigma^2 = 0.001$. Finally the saturation nonlinearity $y = \tanh(x)$ is applied to the noisy output of the Rayleigh channel. The whole nonlinear channel is treated as a black box and only the input and output are known.
In this experiment, six algorithms are employed to track the time variant multipath channel. The first one is normalized LMS algorithm (shown as LMS-2 in the following figures and tables) (with regularization factor \( \epsilon = 10^{-3} \), step size \( \eta = 0.25 \)); the second is the RLS algorithm (with regularization parameter \( \lambda = 10^{-3} \), forgetting factor \( \beta = 0.995 \)); the third is the EX-RLS algorithm (with state transition matrix \( F = \text{diag}[\alpha_1, \ldots, \alpha_M] \), and state noise covariance \( Q = \text{diag}[q_1, \ldots, q_M] \), the parameters \( \alpha_i \) and \( q_i \) \((i = 1, \ldots, M)\) are both determined by the maximum Doppler frequency \( f_D = 100 \text{Hz} \), forgetting factor \( \beta = 0.995 \), regularization parameter \( \lambda = 10^{-3} \)). All parameters for these two algorithms are set according to [1] (on page 759). The forth is the KRLS algorithm (forgetting factor \( \beta = 0.995 \), regularization parameter \( \lambda = 0.01 \)); the fifth is the Ex-KRLS algorithm (\( A = \alpha I, \alpha = 1, q = 10^{-2}, \beta = 0.995, \lambda = 0.01 \)). The last one is the proposed algorithm Ex-KRLS-KF algorithm (with the same state model and state noise with the Ex-RLS algorithm, forgetting factor \( \beta = 0.995 \), regularization parameter \( \lambda = 0.01 \), and measurement noise covariance \( R = 10^{-6} \)). For all the kernel methods, the Gaussian kernel is applied with the kernel parameter equal to 1. All the parameters are selected 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 Figure 3-5 which clearly shows that our algorithm has the best tracking performance. Because the tracking performances of the RLS and Ex-KRLS are too close to be distinguished in the figure, we just plot the Ex-RLS curve. The last 100 values in the learning curves are used to calculate the final mean square error (MSE), which is listed in Table 3-3. From Table 3-3, the proposed algorithm outperforms other existing algorithms statistically significantly. Observing Figure 3-5 more carefully, one can find that the KRLS, Ex-KRLS and Ex-KRLS-KF algorithms perform very similarly at beginning stage of tracking (for \( i \leq 100 \)) because of slow fading channel, when these algorithms are learning the nonlinear channel. Then, the performances of these kernel methods...
algorithms are different (for $i > 100$). The KRLS algorithm cannot always track the dynamical system and the MSE of the KRLS stats to increase from about the 500th iteration. The Ex-KRLS algorithm can still track and learn this system slowly because of its random walk state model and state noise. The Ex-KKRLS-KF algorithm can track and learn this system better because of more precise state model and well-learned measurement model. Therefore, even for this slow fading channel we still obtain more than 5dB improvement compared with the Ex-RKLS.

![Graph](image)

Figure 3-5. Ensemble learning curves of the LMS-2, RLS, EX-RLS, KRLS, EX-KRLS and EX-KRLS-KF in tracking a Rayleigh fading multipath channel with maximum Doppler frequency $f_D = 100Hz$

Table 3-3. Performance comparison in Rayleigh fading channel tracking with maximum Doppler frequency $f_D = 100Hz$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSE (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS-2</td>
<td>-10.8764±0.79662</td>
</tr>
<tr>
<td>RLS</td>
<td>-11.8035±0.54729</td>
</tr>
<tr>
<td>Ex-RLS</td>
<td>-11.8043±0.54741</td>
</tr>
<tr>
<td>KRLS</td>
<td>-16.1064±1.2613</td>
</tr>
<tr>
<td>Ex-KRLS</td>
<td>-18.3312±1.5165</td>
</tr>
<tr>
<td>Ex-KRLS-KF</td>
<td>-23.6590±0.82174</td>
</tr>
</tbody>
</table>
In the previous experiment, the maximum Doppler frequency is set as $f_D = 100Hz$ for a slow fading channel. In order to show the different behaviors of these algorithms, we increase the maximum Doppler frequency as $f_D = 500Hz$, which corresponds to a vehicle moving at the speed of $v = 600Km/h$ (the speed of the fastest train in the world is about $570Km/h$), to obtain a faster fading channel and repeat the experiment. Considering that the increased maximum Doppler frequency will result in larger state noise for Ex-KRLS algorithm, we reset the parameter $q = 0.1$ in this algorithm. The other parameters are maintained or modified by the new maximum Doppler frequency.

The performances of these algorithms are plotted in Figure 3-6. The last 100 values in the learning curves are used to calculate the final mean square error (MSE), which is listed in Table 3-4.

![Figure 3-6. Ensemble learning curves of the LMS-2, RLS, EX-RLS, KRLS, EX-KRLS and EX-KRLS-KF in tracking a Rayleigh fading multipath channel with maximum Doppler frequency $f_D = 500Hz$](image)

From the results in Table 3-4, it is clear that the proposed algorithm achieves the best performance and it is also statistically significant. The KRLS algorithm does
Table 3-4. Performance comparison in Rayleigh fading channel tracking with maximum Doppler frequency \( f_D = 500 \text{Hz} \)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSE (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS-2</td>
<td>-10.9638±0.85232</td>
</tr>
<tr>
<td>RLS</td>
<td>-9.54140±0.44459</td>
</tr>
<tr>
<td>Ex-RLS</td>
<td>-9.82010±0.44807</td>
</tr>
<tr>
<td>KRLS</td>
<td>-5.91970±1.1839</td>
</tr>
<tr>
<td>Ex-KRLS</td>
<td>-14.0677±1.1863</td>
</tr>
<tr>
<td>Ex-KRLS-KF</td>
<td>-21.5085±1.1051</td>
</tr>
</tbody>
</table>

not work in this cases. Even the Ex-KRLS algorithm cannot obtain good tracking performance and the MSE increases slightly with iteration increasing.

To analyze our algorithm theoretically, we plot the MSE of the Ex-KRLS-KF for different maximum Doppler frequencies (from 100Hz to 1500Hz) in Figure 3-7. One can find that the MSEs are larger than -10 when the maximum Doppler frequencies are higher than 1000Hz. The reason is that when the maximum Doppler frequency is high the hidden state changes faster which results in that the KRLS algorithm is not able to learn the measurement function well. Further, with the inaccurate measurement function, the Ex-KRLS-KF cannot estimate proper hidden states and track the signal well. Therefore, the Ex-KRLS-KF is just suitable to tracking slow fading channel.

### 3.5 Discussion and Conclusion

In this chapter, a new extended kernel recursive least squares algorithm, Ex-KRLS-KF, is proposed based on the Kalman filter. The KRLS and Ex-KRLS algorithms are also mentioned and compared with the proposed algorithm in this chapter. The KRLS algorithm constructs its model as the linear form in the RKHS, which is an unknown non-linear model from the view of the input space. The Ex-KRLS algorithm goes along this line, and constructs its state model in the RKHS as well to achieve better tracking performance. However, since the estimation of the transition operator in the RKHS for a general case is difficult, Ex-KRLS is just implemented practically with a very simple state model, which can be used as exponentially-weighted KRLS and random-walk KRLS (see details in Section 2.2.3). In order to develop an algorithm which can deal
with a complicated state model, we proposed the Ex-KRLS-KF algorithm which is a combination of the KRLS and Kalman filter. The state model is preserved in the original data space, while the measurement model is constructed in the RKHS. Therefore, this algorithm can have complicated state model and is suitable for any unknown measurement model.

Two applications of this algorithm are presented. In the vehicle tracking problem, our algorithm is compared with KRLS and EKF algorithms which are the two parts of our algorithm to track the vehicle in a surveillance. The application is a special case for our algorithm: the inputs are set as zeros. The tracking performances show that our algorithm outperforms its components. The nonlinear Rayleigh fading channel problem illustrates that our algorithm is the best one among other existing algorithms, including the LMS-2, RLS, Ex-RLS, KRLS, and Ex-KRLS algorithms.

All the kernelized algorithms included in this chapter have better tracking performances than other algorithms in the data space. The superiority is achieved at the cost of higher
computational complexity. In Table 3-5, we summarize the computational complexity of all algorithms implemented in this chapter.

Table 3-5. Computational complexity analysis

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Computational complexity per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLS</td>
<td>$O(n_x^2)$</td>
</tr>
<tr>
<td>Ex-RLS</td>
<td>$O(n_x^2)$</td>
</tr>
<tr>
<td>EKF</td>
<td>$O(n_x^3)$</td>
</tr>
<tr>
<td>UKF</td>
<td>$O(n_x^3)$</td>
</tr>
<tr>
<td>CKF</td>
<td>$O(n_x^2)$</td>
</tr>
<tr>
<td>KRLS</td>
<td>$O(i^2)$</td>
</tr>
<tr>
<td>Ex-KRLS</td>
<td>$O(i^2)$</td>
</tr>
<tr>
<td>Ex-KRLS-KF</td>
<td>$O(i^2) + O(n_x^2)$</td>
</tr>
</tbody>
</table>

Here $n_x$ denotes the dimension of the weight vector or the state vector. The computational complexity of KRLS and Ex-KRLS algorithms grow as the square of the iteration $i$; and the computational complexity of the non-kernelized algorithms grow as the square or the cube of the state dimension $n_x$. Because the proposed algorithm is the combination of KRLS and nonlinear Kalman filter algorithms, the computational complexity of the proposed algorithm is the sum of their computational complexity. Generally, since the sample number is much larger than the state dimension, the computational complexity of the proposed algorithm is also comparable with the KRLS and Ex-KRLS algorithms. There are strategies to reduce the computational complexity of KRLS (see for instance ALD [16, 17] and surprise [55]) and they can also be applied to the Ex-KRLS-KF algorithm. Further work towards constant budget algorithm is a promising line of research.

Because of the close relationship between the Ex-RLS and the Kalman filter, nonlinear Kalman filter algorithms (such as EKF, UKF, and CKF) are employed to solve the nonlinear Ex-RLS problem. Using the nonlinear Kalman filter with the approximated measurement function, we can predict the outputs of the measurement function. However, how to estimate the hidden state is still an interesting problem. Since the approximation of the measurement function is based on the estimated hidden states,
while the hidden states are estimated based on the measurement function, how to implement this dual learning task still needs more research.
4.1 Overview

We study a novel methodology to design a learning nonlinear generative model using a Bayesian filtering framework in reproducing kernel Hilbert spaces (RKHS). The goal of this algorithm is to estimate and/or predict the time series from the noisy measurements generated by the following underlying unknown autonomous system in Figure 4-1, which can be non-linear, non-convergent and dynamical system,

\[ x_i, y_i, v_i \]

where \( \{x_i\} \) are the time series in \( \mathbb{R}^n \), which are usually unobservable, and \( \{y_i\} \) are the measurements in \( \mathbb{R}^n \), which are the noisy version of the time series \( \{x_i\} \) corrupted by zero-mean measurement noise \( \{v_i\} \). The noisy \( \{v_i\} \) is independent with signal \( \{x_i\} \) and no Gaussian assumption is required about the noise. Because only the measurements are observable, we model a new dynamical system about measurements in RKHS, which is learned from the given training measurement data, and use the Kalman filter in the RKHS to estimate and/or predict the measurement distribution. Further, we can estimated and/or predicted time series from the corresponding measurement distribution.

This model is very popular and applied to study many phenomena in science, engineering and economy fields, and many methodologies are proposed to solve this
kind of problems. We can categorize them into two groups: direct models and generative models.

Direct approaches make no explicit attempt to model the underlying distributions of the variables and features and are only interested in optimizing a mapping from the inputs to the desired model outputs. For the autonomous model in Figure 4-1, \( y \), or the embedding past measurements \( y_{E,i} = [y_{i-(n_E-1)\tau}, y_{i-(n_E-2)\tau}, \ldots, y_{i-\tau}, y_i] \), where \( n_E \) and \( \tau \) are the embedding dimension and the embedding delay respectively according to Takens’ embedding theorem [64], are used to predict \( y_{i+1} \), by approximating the function \( f : y_{i+1} = f(y_i) \) or \( f : y_{i+1} = f(y_{E,i}) \). These direct approaches include: neural networks [71], support vector regression (SVR)[65–67], and kernel adaptive filter algorithms such as kernel recursive least squares (KRLS) [16] and kernel least mean squares (KLMS) [15]. These approaches usually have good performance to predict the noiseless time series. However, they can not represent the underlying structure of the time series, and the performance is always affected by different kinds of noise environments.

Generative approaches produce a probability density model over all variables in a system and manipulate it to compute classification and regression functions. The celebrated Kalman filter [25] is one of the generative approach, which is widely used to estimate hidden states for linear system models. The problem is that the Kalman filter is a linear system so cannot cope with the diversity of models that generate time series of interest in science and engineering. Some of nonlinear versions of the Kalman filter were developed to solve nonlinear dynamical system problems, such as the extended Kalman filter (EKF)[27–29], unscented Kalman filter (UKF) [33, 34] and cubature Kalman filter (CKF) [38, 39] algorithms. More recently, there are other generative approaches which are also applied to deal with these nonlinear filtering problems, such as the kernel Kalman filter (KKF) [56, 57], the Dynamical System Model with a Conditional Embedding (DSMCE) operator [61] and the kernel Bayes’ rule (KBR) [82] algorithms. For the KKF, the Kalman filter is implemented in a high dimensional subspace obtained by Kernel
PCA algorithm [11, 12]. The DSMCE and KBR algorithms are both developed based on the conditional embedding concept in RKHS. All of these generative approaches treat the time series \( \{x_i\} \) or their mappings in the feature space as the hidden states and attempt to describe the dynamics of the hidden states by the assumed system state-space model or the given hidden state training data set. However, for most of the time series problems, the accurate system state-space model or clean hidden state training data set are not available. Therefore, accurate estimation and prediction performances cannot be obtained by these algorithms.

In order to avoid the description of the unavailable hidden states dynamics, we use the estimated embedding of the noisy measurements \( \{y_i\} \) in RKHS as the hidden state, instead of the embedding of \( \{x_i\} \). Then we construct a new state space model in a RKHS using the estimated conditional embedding operator and implement the Kalman filtering in this space.

This approach learns the state of the nonlinear dynamic system in a self-organized manner by processing the given measurements. As such, it does not require an assumed state space model, unlike the Kalman filtering and its variants. In addition, the embedding represents the distribution of the measurements, and the conditional embedding operator describing the dynamics can be estimated using the training measurement data alone. Therefore, the Kalman filter in the RKHS propagates and updates the measurement distribution, not just the mean and covariance as in input space counterpart, which enhances the filtering capability to withstand non-Gaussian noise environments. Furthermore, the state and measurement model uncertainty is assumed to be a Gaussian noise in the RKHS, which can represent much wider noise types in the input space [12, 57]. Therefore our solution in the RKHS preserves most of the features of the Bayesian filter, although the recursive update structure of the Kalman filter fully utilized with the advantage of efficient computation. To distinguish our work
from previous attempts, we name our novel algorithm the kernel Kalman filter based on the conditional embedding operator (KKF-CEO).

The rest of the chapter is organized as follows. In Section II, the Hilbert space embeddings and conditional embeddings are introduced. In Section III, the conventional Kalman filter is reviewed and our KKF-CEO algorithm is derived. Some optimization techniques are discussed in Section IV. Then experiments are presented to compare our novel algorithm with some existing algorithms in Section V. Finally, the discussion and conclusion are given in Section VI.

4.2 Conditional Embeddings in RKHS

In this section, we introduce the Hilbert space embeddings and the conditional embedding operator in the RKHS, which can be estimated from given training measurements. The estimated conditional embedding operator will be applied in our novel algorithm as the state transition operator.

4.2.1 Hilbert Space Embeddings

The kernel functions mentioned before can either map a deterministic value into the RKHS, or a random variable (R.V.) into the space. In other words, we can represent probability distributions by elements in a reproducing kernel Hilbert space with a Hilbert space embedding. We will summarize briefly here the concepts of Hilbert space embeddings [61, 62, 82]. In the following, we denote by $X$ and $Y$, random variables with domain $\mathcal{X}$ and $\mathcal{Y}$, and refer to instantiations of $X$ and $Y$ by the lower case character, $x$ and $y$. We endow $\mathcal{X}$ and $\mathcal{Y}$ with some $\sigma$-algebra $\mathcal{A}$ and $\mathcal{B}$, and denote the space of all probability distributions (with respect to $\mathcal{A}$ and $\mathcal{B}$) on $\mathcal{X}$ and $\mathcal{Y}$ by $\mathcal{P}_X$ and $\mathcal{P}_Y$, respectively.

**Embedding distribution.** We first focus on a single R.V. $X$ with probability distribution $P_X(x)$ on $\mathcal{X}$. Considering $x$ is mapped into a RKHS $\mathcal{F}$ associated with the kernel function $k_\mathcal{F}$ by $\varphi(\cdot)$, the expectation of $f(X)$ can be expressed as an inner
where $f(\cdot) \in \mathcal{F}$ and $E[\cdot]$ denotes the expectation operator. Here, the term $\mu_X$ is the embedding in the RKHS with respect to the probability distribution $P_X(x)$, which is defined as [61]

$$
\mu_X := E_X [\varphi(X)].
$$

The embedding is also in the RKHS $\mathcal{F}$ as long as $E[k_\mathcal{F}(X, X)] < \infty$, and is referred as the mean map. Its empirical estimate is

$$
\hat{\mu}_X = \frac{1}{m} \sum_{i=1}^{m} \varphi(x_i)
$$

where $D_X = \{x_1, \ldots, x_m\}$ is a training set assumed to have been drawn i.i.d. from $P_X(x)$. With the embedding $\mu_X$, we can guarantee that distinct distributions can be mapped to distinct points in a RKHS for the characteristic kernel class. [79].

**Definition 1 (Characteristic kernel)** Let $\mathcal{X}$ be a topological space, $P_X \in \mathcal{P}_X$ be a Borel probability distribution on $\mathcal{X}$, $\mathcal{F}$ is a RKHS associated with a measurable, bounded kernel $k(\cdot, \cdot)$ on $\mathcal{X}$, $\mu_X \in \mathcal{F}$ is the embedding with respect to the distribution $P_X$. Then $k(\cdot, \cdot)$ is said to be characteristic if the map $\mathcal{M}$ by

$$
\mathcal{M} : \mathcal{P}_X \rightarrow \mathcal{F}, \ P_X \mapsto \mu_X
$$

is injective.
There are some close relationships between the characteristic kernels and the universal kernels \([77, 86]\). Specifically, the Gaussian kernel is universal and characteristic. Therefore, in this chapter the Gaussian kernel is selected to develop our novel algorithm. From now on only RKHS associated with Gaussian kernels are used.

**Cross-covariance operator.** Next, we consider two R.V. \(X\) and \(Y\) with marginal and joint probability distributions \(P_X\), \(P_Y\), and \(P_{XY}\), respectively. Like \(\phi(\cdot)\) which maps the R.V. \(X\) into the RKHS \(\mathcal{F}\), \(\phi(\cdot)\) maps the R.V. \(Y\) into the RKHS \(\mathcal{G}\) associated with the kernel function \(k_G\), with the mean map \(\mu_Y\) with respect to \(P_Y\). Likewise, the expectation of \(f(X)g(Y)\) can also be expressed as an inner product in RKHS,

\[
E[f(X)g(Y)] = E[(f, \phi(X))_\mathcal{F} (g, \phi(Y))_\mathcal{G}] \\
= E[(f \otimes g, \phi(X) \otimes \phi(Y))_{\mathcal{F} \otimes \mathcal{G}}] \\
= \langle f \otimes g, E[\phi(X) \otimes \phi(Y)]_{\mathcal{F} \otimes \mathcal{G}} \rangle \\
= \langle f \otimes g, C_{XY} \rangle_{\mathcal{F} \otimes \mathcal{G}} \\
= \langle f, C_{XY} g \rangle_{\mathcal{F}}
\]

where \(\otimes\) denotes the tensor product operator and \(\mathcal{F} \otimes \mathcal{G}\) is also a RKHS associated with the kernel function \(k_Fk_G\). The term \(C_{XY}\) is the uncentered cross-covariance operator, which is defined as

\[
C_{XY} = E[\phi(X) \otimes \phi(Y)] \\
= \int \phi(x) \otimes \phi(y) dP_{XY}(x, y) \in \mathcal{F} \otimes \mathcal{G}
\]

and \(C_{XX}\) is also defined as

\[
C_{XX} = E[\phi(X) \otimes \phi(X)] \\
= \int \phi(x) \otimes \phi(x) dP_X(x) \in \mathcal{F} \otimes \mathcal{F}
\]
The cross covariance operator is a linear operator $C_{XY} : \mathcal{G} \rightarrow \mathcal{F}$. Since the uncentered cross covariance operator is only determined by the joint probability distribution $P_{XY}(x, y)$ on $\mathcal{X} \times \mathcal{Y}$ when the kernel functions are given, it can be treated as the joint distribution embedding $\mu_{XY}$ in the tensor product RKHS $\mathcal{F} \otimes \mathcal{G}$ [82].

Given $m$ pairs of training examples $D_{XY} = \{(x_1, y_1), \cdots, (x_m, y_m)\}$ drawn i.i.d. from $P_{XY}(x, y)$, the uncentered covariance operator $C_{XY}$ can be estimated as

$$\hat{C}_{XY} = \frac{1}{m} \sum_{i=1}^{m} \varphi(x_i) \otimes \phi(y_i) = \frac{1}{m} \gamma \Phi^T$$

(4–8)

where we denote the feature matrices by $\gamma = [\varphi(x_1), \cdots, \varphi(x_m)]$ and $\Phi = [\phi(y_1), \cdots, \phi(y_m)]$.

### 4.2.2 Conditional embedding operator

In [61] the conditional embedding is introduced, which embeds conditional distribution of the form $P_{Y|X}$ into a RKHS. Assume that the conditional expectation $E_{Y|X}[g(Y)|X = :] \in \mathcal{F}$ for all $g \in \mathcal{G}$. We have the following relation provided by Fukumizu in [62],

$$C_{XX}E_{Y|X}[g(Y)|X = :] = C_{XY}g.$$  

(4–9)

Here the cross covariance operators $C_{XX}$ and $C_{XY}$ map the functions $E_{Y|X}[g(Y)|X = :] \in \mathcal{F}$ and $g \in \mathcal{G}$ into $\mathcal{F}$, respectively.

Using (4–9), the conditional expectation of $g(Y)$ given $X = x$ can be expressed as an inner product, [62, 82]

$$E_{Y|X}[g(Y)|X = x] = \langle g, \mu_{Y|X} \rangle_{\mathcal{G}}$$

$$= \langle E_{Y|X}[g(Y)|X = :], \varphi(x) \rangle_{\mathcal{F}}$$

$$= \langle C_{XX}^{-1}C_{XY}g, \varphi(x) \rangle_{\mathcal{F}}$$

$$= \langle g, C_{YY}C_{XX}^{-1}\varphi(x) \rangle_{\mathcal{G}}$$

(4–10)

where $C_{XX}$ is assumed to be injective, i.e., if the function $f \in \mathcal{F}$ with $C_{XX}f = C_{XY}g$ is unique.
Here the term $\mu_{Y|x}$ is the conditional embedding of R.V. $Y$ given $X = x$, which is defined as

$$\mu_{Y|x} := E[\phi(Y)|X = x].$$

(4–11)

According to (4–10), the conditional embedding $\mu_{Y|x}$ can be expressed as

$$\mu_{Y|x} = C_{YX}C_{XX}^{-1}\varphi(x).$$

(4–12)

It is noteworthy to remark that the assumption of $E_{Y|X}[g(Y)|X = \cdot] \in \mathcal{F}$ always holds for finite domains with characteristic kernels, does not necessarily hold for continuous domains. In the case where it does not hold, we use $C_{YX}C_{XX}^{-1}\varphi(x)$ as an approximation of $\mu_{Y|x}$ [61].

Using the law of total expectation in the RKHS, we have

$$\mu_Y = E_X[\mu_{Y|x}] = C_{YX}C_{XX}^{-1}\mu_X.$$

(4–13)

Now, we can give the following definition of the conditional embedding operator $\mathcal{U}_{Y|X}$.

**Definition 2** The conditional embedding operator $\mathcal{U}_{Y|X}$ is defined as

$$\mathcal{U}_{Y|X} := C_{YX}C_{XX}^{-1}.$$

(4–14)

The conditional embedding operator satisfies three properties:

1. $\mu_{Y|x} := E_{Y|X}[\phi(Y)|x] = \mathcal{U}_{Y|X}k(x, \cdot)$;
2. $E_{Y|X}[g(Y)|x] = \langle g, \mu_{Y|x} \rangle_G$;
3. $\mu_Y = \mathcal{U}_{Y|X}\mu_X$.

The conditional embedding operator can be estimated as

$$\hat{\mathcal{U}}_{Y|X} = \hat{C}_{YX}\hat{C}_{XX}^{-1}$$

$$= \frac{1}{m}\phi\gamma^T \left( \frac{1}{m}\gamma\gamma^T + \varsigma I \right)^{-1}$$

$$= \phi\gamma^T \left( \gamma\gamma^T + \varsigma mI_m \right)^{-1}$$

(4–15)
further by the matrix inversion lemma [81],

\[ \mathcal{U}_{Y|X} = \Phi (K + \varsigma m \mathbf{I})^{-1} \mathbf{1} \tag{4-16} \]

where \( K = \mathbf{1}^T \mathbf{1} \), \( \mathbf{I} \) is an identity operator, \( \mathbf{I}_m \) denotes an \( m \times m \) identity matrix, and \( \varsigma \) is a regularization term. We must emphasize that the significance of the transition from (4–15) to (4–16). First, \( K = \mathbf{1}^T \mathbf{1} \) is computable by the kernel trick. Second, the estimated conditional embedding operator can be applied conveniently to calculate \( \mu_{Y|X} \) and \( \mu_Y \) in (4–12) and (4–13), respectively.

The conditional embedding operator links the feature map \( \varphi(x) \) and the conditional embedding \( \mu_{Y|X} \), which represents the conditional distribution \( P_{Y|X}(y) \) in the RKHS. Therefore, several Bayesian algorithms are developed using this operator [61, 82], which are constructed by hidden states and training measurement data, to estimate the hidden state embeddings. We take the DSMCE algorithm [61] for example to show how to use the conditional embedding operators to develop dynamical system algorithms.

### 4.2.3 Dynamical System Model with a Conditional Embedding operator

Consider that a dynamical system generates hidden states and corresponding measurements \( \{x_i, y_i\} \). The goal of the DSMCE algorithm is to recursively maintain the embedding \( \mu_{x_i} \) given a new measurement \( y_i \). Therefore the following conditional embedding operator is constructed

\[ \mathcal{U}_{x_{i+1}|x_i y_{i+1}} = c_{x_{i+1}|x_i y_{i+1}}^{-1} (c_{x y_{i+1}}^{-1}) (c_{x y_{i+1}}) \tag{4-17} \]

which takes both the previous belief state and an measurement as inputs, and outputs the predictive embedding for \( P(x_{i+1}|x_i y_{i+1}) \). That is

\[ \mu_{x_{i+1}|x_i y_{i+1}} = \mathcal{U}_{x_{i+1}|x_i y_{i+1}} \omega((x_i, y_{i+1})) \tag{4-18} \]
where $\omega(\cdot)$ is the joint feature map for $x_i$ and $y_{i+1}$. In [61] the joint feature map is approximated by the concatenation of the feature map $\varphi(x_i)$ and $\psi(y_{i+1})$; that is

$$\omega((x_i, y_{i+1})) = (\varphi(x_i)^T, \phi(y_{i+1})^T)^T.$$ (4–19)

With this approximated joint feature map, the conditional embedding operator $U_{x_{i+1}|x_i y_{i+1}}$ can be equivalently viewed as the concatenation of two operators $U_{x_{i+1}|x_i y_{i+1}} = (T_1, T_2)$ and the conditional embedding $\mu_{x_{i+1}|x_i y_{i+1}}$ is expressed as

$$\mu_{x_{i+1}|x_i y_{i+1}} = T_1 \varphi(x_i) + T_2 \psi(y_{i+1}).$$ (4–20)

Next, taking the expectation of $\mu_{x_{i+1}|x_i y_{i+1}}$ with respect to $P(x_{i+1}|y_{i+1})$, one can obtain

$$\mu_{x_{i+1}|y_{i+1}} = T_1 \mu_{x_i|y_{i+1}} + T_2 \psi(y_{i+1})$$
$$\approx T_1 \mu_{x_i|y_i} + T_2 \psi(y_{i+1})$$ (4–21)

where the distribution $P(x_i|y_{i+1})$ is approximated by a less confident one $P(x_i|y_i)$. (4–21) is the solution to update the embedding $\mu_{x_i}$ given a new measurement $y_i$. Furthermore, given training samples the operators can be approximated easily. Finally, $\hat{x}_i \in \mathbb{R}^{n_x}$ is determined as the estimated hidden state by performing the following optimization problem

$$\hat{x}_i = \arg\min_x \|\varphi(x) - \hat{\mu}_{x_i|y_i}\|_H^2$$ (4–22)

The DSMCE algorithm is summarized in Algorithm 4-1.

**4.3 Kernel Kalman Filter based on Conditional Embedding Operator**

In this section, we construct the Kalman filter in the RKHS, using the estimated conditional embedding operator mentioned in Section II as the state transition operator.

The concept of state is fundamental in the Kalman filter [25]. The state vector or simply state, denoted by $x_i$, is defined as the minimal model that is sufficient to uniquely describe the autonomous dynamical behavior of the system; the subscript $i$ denotes discrete time. Typically, the state $x_i$ is unobservable. To estimate it, we use a set of
Algorithm 4-1: Dynamical System Model with Conditional Embedding (DSMCE)

Learning:
Given training samples: \( D = [(x^0_0, y^0_0), \ldots, (x^0_{m+1}, y^0_{m+1})] \)
\( T = [\varphi(x^0_0), \ldots, \varphi(x^0_m)] \), \( \Phi = [\varphi(x^0_0), \ldots, \varphi(x^0_{m+1})] \) and \( \Psi = [\varphi(y^0_0), \ldots, \varphi(y^0_{m+1})] \)
Compute matrices: \( \mathbf{K} = T^T T \), \( \mathbf{U} = \Phi^T \Phi \) and \( \mathbf{G} = T^T \Phi \)
Compute matrices: \( \tilde{T}_2 = (\mathbf{K} + \mathbf{U} + \lambda \mathbf{I}_m)^{-1} \) and \( \tilde{T}_1 = \tilde{T}_2 \mathbf{G} \)

Filtering:
for \( i = 0 \): Let \( \beta = \frac{1}{m} \mathbf{1} \),
for \( i > 0 \):
Update estimated embedding: \( \tilde{\mu}_{x_i|y_i} = \Phi \beta_i \) and \( \beta_i = \tilde{T}_1 \beta_i + \tilde{T}_1 \Psi T \psi(y_i) \)
Estimate hidden state: \( \hat{x}_i = \arg \min_x \| \varphi(x) - \tilde{\mu}_{x_i|y_i} \|^2_{\mathcal{H}_x} \)

measurement data, denoted by the vector \( y_i \). The model can be expressed in (2–4) and (2–5) where \( F_i \) is the given transition matrix taking the state \( x_i \) from time \( i \) to time \( i + 1 \), and \( H_i \) is the given measurement matrix. The process noise \( n_i \) and measurement noise \( v_i \) are both assumed to be zero-mean, additive, white, and Gaussian noise, with covariance matrices \( Q_i \) and \( R_i \), respectively. In Section 2.1.2 we present the traditional Kalman filter equations in our notation.

4.3.1 Construction of Kalman Filtering in the RKHS

In this subsection, we develop an algorithm for the special case of tracking a noisy time series signal, which is generated by the system in Figure 4-1. The system model is written as:

\[
\begin{align*}
x_{i+1} &= f(x_i) + n_i \quad (4\text{–23}) \\
y_i &= x_i + v_i \quad (4\text{–24})
\end{align*}
\]

where \( f \) is an unknown nonlinear function, \( n_i \) is the state processing noise reflecting the state model uncertainty and \( v_i \) is the zero mean measurement noise. Here a simple additive noise measurement model is considered.

Because the state transition function is unknown, the traditional Kalman filter or nonlinear Kalman filters cannot be applied in this case. Likewise, because the hidden state \( \{x_i\} \) are not available in many signal processing cases, we cannot use the DSMCE
[61] and KBR [82] algorithms, in which the hidden states training data are necessary to construct the state transition operators.

In order to avoid the description of the unavailable hidden states dynamics, we use the estimated embedding of the noisy measurements \( \{y_i\} \) as the hidden state in RKHS, instead of the embedding of \( \{x_i\} \). Therefore, we can construct the new state transition operators only using the noisy measurements, to describe the dynamics of the measurement embeddings and develop a Kalman filter in the RKHS. For the rest of this paper, the estimated measurement embedding is denoted by \( \hat{\mu}_i \).

For convenience, we denote the estimated conditional embedding operator by \( F_i \) instead of \( \tilde{U} \), which is the state transition operator, not a matrix now. Assuming that we have the time series \( D = \{y_1^0, \ldots, y_{m+1}^0\} \) as the training data set, a block of samples taken from time series \( \{y_i\} \), which are mapped into the RKHS \( H_y \), the estimated conditional embedding operator \( F_i \) can be expressed as

\[
F_i = \Phi (K + \varsigma m I_m)^{-1} \gamma^T 
\]

where \( \gamma = [\varphi(y_1^0), \ldots, \varphi(y_{m+1}^0)] \), \( \Phi = [\varphi(y_2^0), \ldots, \varphi(y_{m+1}^0)] \), \( K = \gamma^T \gamma \). Here the superscript 0 refers to the training data.

We can connect the current estimated measurement embedding \( \hat{\mu}_i \) with the next step estimated measurement embedding \( \hat{\mu}_{i+1} \) by

\[
\hat{\mu}_{i+1} = F_i \hat{\mu}_i. \tag{4–26}
\]

Because the operator is estimated based on limited data and on noisy dynamics, we need to add some noise \( \xi_i \) in the RKHS to the predicted next step embedding as

\[
\hat{\mu}_{i+1} = F_i \hat{\mu}_i + \xi_i. \tag{4–27}
\]

Here, (4–27) is the state model in the RKHS.
When we have a new measurement $y_i$, we have to build the measurement model in the RKHS to link the measurement and the hidden state, estimated embedding $\hat{\mu}_i$. Then, we map the new measurement $y_i$ into the RKHS as $\varphi(y_i) \in \mathcal{H}_y$. Because $\hat{\mu}_i$ is the estimate of $E[\varphi(Y_i)]$ and $\varphi(y_i)$ is an instantiation of the R.V. $\varphi(Y_i)$, we have

$$\varphi(y_i) = \hat{\mu}_i + \nu_i$$

(4–28)

where $\nu_i$ is the measurement noise in the RKHS.

Therefore, we have the dynamical state-space model of measurements in the RKHS

$$\hat{\mu}_{i+1} = F_i \hat{\mu}_i + \xi_i$$

$$\varphi(y_i) = \hat{\mu}_i + \nu_i.$$  

(4–29)

It is noteworthy that $\hat{\mu}_i$ is a R.V. in the RKHS which does not correspond to any R.V. in the input space.

Here $\xi_i$ and $\nu_i$ are both noises in the RKHS, which represent the uncertainty in the estimated state model in (4–27) and measurement noise (difference between estimated expectation $\hat{\mu}_i$ and measurement instantiation $\varphi(y_i)$) in (4–28). In the RKHS we can treat the noises $\xi_i$ and $\nu_i$ as $n_k \times 1$ vectors. For the Gaussian kernel, $n_k$ is infinite. If we assume that each component of these vectors is independent zero-mean Gaussian noise process with covariance

$$Q_i = qI$$

(4–30)

$$R_i = rI$$

(4–31)

like the assumption of the Ex-KRLS algorithm [17], we can use the Kalman filter to estimate the hidden state $\hat{\mu}_i$ in the RKHS. Although, (4–30) and (4–31) cannot accurately describe the covariances of the noise in the RKHS, these two scalars $q$ and $r$ reflect the noise intensity. A smaller $q$ is applied when the estimated conditional
embedding operator is approximated more accurately, while a larger \( r \) is used when the measure noise in the input space is larger. In Section V we will find that it is the ratio \( q/r \) that is more important rather than \( q \) and \( r \).

Comparing the traditional model in (2–5) with the model in the RKHS (4–29), we can see that, \( F_i = \Phi (K + \zeta ml_m)^{-1} T^T, H_i = I, Q_i = qI \) and \( R_i = rI \). Therefore, the recursion to estimate \( \mu_i \) in the RKHS will be written as follows:

Start with \( \hat{\mu}_0 = E[\varphi(y_0)] \) or \( \hat{\mu}_0 = \varphi(y_0), P_0 = \lambda I \) like the Kalman filter [26],

\[
\begin{align*}
\hat{\mu}_i^- & = F_{i-1}\hat{\mu}_{i-1} \quad (4–32) \\
P_i^- & = F_{i-1}P_{i-1}F_{i-1}^T + Q_{i-1} \quad (4–33) \\
G_i & = P_i^- \left[ P_i^- + R_i \right]^{-1} \quad (4–34) \\
\hat{\mu}_i & = \hat{\mu}_i^- + G_i \left( \varphi(y_i) - \hat{\mu}_i^- \right) \quad (4–35) \\
P_i & = (I - G_i) P_i^- \quad (4–36)
\end{align*}
\]

where \( \hat{\mu}_i^- \) denotes a priori estimate of the measurement embedding.

Because the estimated measurement embedding \( \hat{\mu}_i \) and the mapped measurement \( \varphi(y_i) \) both lie in a possible infinite dimensional space \( \mathcal{H}_y \), \( P_i^- \), \( P_i \) and \( G_i \) are all operators in \( \mathcal{H}_y \times \mathcal{H}_y \), which can all be treated as \( n_k \times n_k \) matrices (\( n_k \) is equal to infinity for the Gaussian kernel). Then, this recursive approach cannot be computed directly using normal matrix calculations. However, by the representer theorem in RKHS [87, 88], the solution always exists in the subspace spanned by the data, which in this case is of dimension \( m \). The issue becomes to find a way to write the operators in such a way that \( \infty \times \infty \) matrices are avoided. Inspired by [89] the recursion approach can still be implemented due to the following theorems we have proved.

**Theorem 4.1.** *The operator \( P_i^-, P_i, G_i \) are of the following forms*

\[
P_i^- = \Phi \tilde{\Phi}^T \Phi + qI \quad (4–37)
\]
\[
\mathbf{P}_i = \Phi \tilde{\mathbf{P}}_i \Phi^T + \frac{qr}{q + r} \mathbf{I} \quad (4-38)
\]
\[
\mathbf{G}_i = \frac{r}{q + r} \Phi \tilde{\mathbf{G}}_i \Phi^T + \frac{q}{q + r} \mathbf{I} \quad (4-39)
\]

where \( \tilde{\mathbf{P}}_i \), \( \tilde{\mathbf{P}}_i \), and \( \tilde{\mathbf{G}}_i \) are all \( m \times m \) matrices. Furthermore, these matrices can be computed recursively as follows:

\[
\tilde{\mathbf{P}}_1 = \lambda (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \tau [(\mathbf{K} + \varsigma \mathbf{m}_m)^{-1}]^T \quad (4-40)
\]
\[
\tilde{\mathbf{G}}_i = \left[ (q + r) \mathbf{l}_m + \tilde{\mathbf{P}}_i \Phi^T \Phi \right]^{-1} \tilde{\mathbf{P}}_i \quad (4-41)
\]
\[
\tilde{\mathbf{P}}_i = \frac{r}{q + r} \tilde{\mathbf{P}}_{i-1} - \frac{r}{q + r} \tilde{\mathbf{G}}_i \Phi^T \Phi \tilde{\mathbf{P}}_{i-1} - \frac{qr}{q + r} \tilde{\mathbf{G}}_i \quad (4-42)
\]
\[
\tilde{\mathbf{P}}_{i+1} = \left[ (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \right] \Phi \tilde{\mathbf{P}}_i \Phi^T \left[ (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \right]^T + \frac{qr}{q + r} \left[ (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \right] \left[ (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \right]^T \quad (4-43)
\]

**Theorem 4.2.** The predicted embeddings \( \hat{\mu}_i \) and the estimated embeddings \( \hat{\mu}_i \) are both combinations of the mapped measurements \( \{ \varphi(y_i^0) \}_{i=2}^{m+1} \)

\[
\hat{\mu}_i = \Phi a_i \quad (4-44)
\]
\[
\hat{\mu}_i = \Phi b_i + \frac{q}{q + r} \varphi(y_i) \quad (4-45)
\]

where \( a_i = [a_1, \ldots, a_m]^T \) and \( b_i = [b_1, \ldots, b_m]^T \) are both \( m \times 1 \) real-valued vectors.

Furthermore, these vectors can also be computed recursively as follows:

\[
a_1 = (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \mu_0 \quad (4-46)
\]
\[
b_i = \left[ \frac{r}{q + r} \mathbf{I} - \frac{r}{q + r} \tilde{\mathbf{G}}_i \Phi^T \Phi \right] a_i + \frac{r}{q + r} \tilde{\mathbf{G}}_i \Phi^T \varphi(y_i) \quad (4-47)
\]
\[
a_i = (\mathbf{K} + \varsigma \mathbf{m}_m)^{-1} \tau^T \left[ \Phi b_{i-1} + \frac{q}{q + r} \varphi(y_{i-1}) \right] \quad (4-48)
\]

Proofs of Theorem 4.1 and Theorem 4.2 are presented in APPENDIX B. By these theorems, the operators are all expressed in term of \( m \times m \) matrices and feature
matrices, such as $\Phi$ and $\Gamma$, which can both be treated as $n \times m$ matrices. Therefore, the calculations from (4–32) to (4–36) only involve normal matrices calculations and inner product operations between feature matrices and vectors like $\Gamma^T \Gamma$ and $\Phi^T \varphi(y_i)$, which can be computed by the kernel trick readily.

The goal of this algorithm is to estimate the signal $x_i$ in the input space from the noisy measurements. Considering the definition of the embeddings in the RKHS, we propose an approach to estimate the filtered signal $\hat{x}_i$ from the embedding $\hat{\mu}_i$, which is similar to the Pre-Image problem [59], but much more effective. From (4–24), we have

$$
\hat{x}_i = E[x_i] = E[x_i + v_i] = E[y_i] = E[\langle f_i, \varphi(y_i) \rangle] = \langle f_i(\cdot), E[\varphi(y_i)] \rangle = \langle f_i(\cdot), \hat{\mu}_i \rangle = \left\langle f_i(\cdot), \Phi b_i + \frac{q}{q + r} \varphi(y_i) \right\rangle
$$

(4–49)

where $f_i = [f_1, \ldots, f_n]$ is a row vector of functions and $f_j(y) = y^{(j)} (j = 1, \ldots, n_y)$. $y^{(j)}$ is the $j$th component of the vector $y$. Although the function $f_j(\cdot)$ may not belong to the RKHS, we can learn and approximate them from the training data $Y^0$, where $Y^0 = [y^0_2, \ldots, y^0_{m+1}]$. The learned row vector of functions $\hat{f}_i(\cdot)$ can be expressed as

$$
\hat{f}_i(\cdot) = \Phi(\Phi^T \Phi)^{-1} Y^0 T.
$$

(4–50)

Then the above equation can be approximated as

$$
\hat{x}_i = b_i^T \Phi^T \Phi (\Phi^T \Phi)^{-1} Y^0 T + \frac{q}{q + r} y_i = Y^0 b_i + \frac{q}{q + r} y_i.
$$

(4–51)

At this point, we have the Kernel Kalman filter algorithm based on the estimated conditional embedding operator, which is summarized in Algorithm 4-2.

The space complexity is $O(m^2)$, while time complexity is $O(m^3)$ because the complexity is dominated by the multiplication and inversion of $m \times m$ matrices. This
Algorithm 4-2: Kernel Kalman Filtering based on conditional embedding operator (KKF-CEO)

**Initialization:** For $i = 0$, set
\[
\begin{align*}
\Upsilon &= \left[ \phi(y_1^0), \ldots, \phi(y_m^0) \right], \\
\Phi &= \left[ \phi(y_2^0), \ldots, \phi(y_{m+1}^0) \right], \\
Y^0 &= [y_2^0, \ldots, y_{m+1}^0], \\
K &= \Upsilon^T \Upsilon, \\
M &= \Phi^T \Phi, \\
T &= \Upsilon^T \Phi, \\
L &= (K + \varsigma m I)^{-1}, \\
\mu_0 &= \phi(y_0), \\
P_0 &= \lambda I, \\
a_{i+1} &= L^T \mu_0, \\
\tilde{P}_{-i+1} &= \lambda LKL^T
\end{align*}
\]

**Filtering:** For $i = 1, \ldots$, compute
\[
\begin{align*}
\tilde{G}_i &= \left[ (q + r) I_m + \tilde{P}_{-i} M \right]^{-1} \tilde{P}_{-i}, \\
\tilde{P}_{i} &= \frac{r}{q+r} \tilde{P}_{-i} - \frac{r}{q+r} \tilde{G}_i M \tilde{P}_{-i} - \frac{aq}{q+r} \tilde{G}_i, \\
b_i &= \left[ \frac{r}{q+r} I_m - \frac{r}{q+r} \tilde{G}_i M \right] a_i + \frac{r}{q+r} \tilde{G}_i \phi^T (y_i), \\
\hat{x}_i &= Y^t b_i + \frac{q}{q+r} y_i, \\
a_{i+1} &= L^T \phi b_i + \frac{q}{q+r} L^T \phi (y_i), \\
\tilde{P}_{i+1} &= L^T \tilde{P}_{i} T^T L^T + \frac{q}{q+r} LKL^T
\end{align*}
\]

Complexity increases when more training data are required to construct the estimated conditional embedding operators. Therefore, simplifying techniques are used to reduce the size of the estimated conditional embedding operators, which will be discussed in Section 4.4.

Although, our algorithm and Fukuminzu’s algorithms [61, 82] are all developed based on the embedding concept, they are totally different methods. The hidden state in our algorithm is the estimated measurement embedding, not the embedding of the unobservable signal, and the transition operator is estimated only from measurements. Therefore, our algorithm can be implemented using only the measurement data $\{y_i\}$, while Fukuminzu’s algorithms both require the clean signal data $\{x_i\}$ and the corresponding measurement data $\{y_i\}$ as training data to construct the respective state transition and measurement operators. Furthermore, comparing with the KKF algorithms [56, 57], our algorithm obtains the state transition operator by constructing the estimated conditional embedding operator directly from noisy measurements, while the KKF
algorithms only implement the Kalman filter in a high dimensional subspace (not the RKHS), the state transition operator has to be learned from the training measurement data by the EM algorithm, which is more complicated, and the Pre-image technique is required to estimate the signal $\hat{x}_i$.

4.3.2 Kalman Filter Predictor in RKHS

In the previous subsection, the KKF-CEO algorithm is presented to denoise measurements based on the available noisy data. In this subsection, we present a predictor based on this algorithm.

Using the KKF-CEO algorithm, we can estimate the current measurement embedding. Instead of utilizing the estimated conditional embedding operator $F_i$ repeatedly to predict the corresponding measurement embedding, we construct a new conditional embedding operator from the training data set which links the current and future measurement embeddings directly. For the $\ell$-step prediction, the predictive conditional embedding operator $F_P$ is estimated as

$$F_P = \psi (K + \varsigma m I_m)^{-1} \gamma^T$$  \hspace{1cm} (4–52)

where $\psi = [\varphi(y_{i+\ell}^0), \ldots, \varphi(y_{m+\ell}^0)]$. With this conditional embedding operator the measurement embedding at time $i + \ell$ can be expressed as

$$\mu_{i+\ell} = F_P \mu_i$$

$$= \psi (K + \varsigma m I_m)^{-1} \gamma^T \left( \Phi b_i + \frac{q}{q + r} \varphi(y_i) \right)$$

$$= \psi c_{i+\ell},$$  \hspace{1cm} (4–53)

where

$$c_{i+\ell} = (K + \varsigma m I_m)^{-1} \gamma^T \left( \Phi b_i + \frac{q}{q + r} \varphi(y_i) \right).$$  \hspace{1cm} (4–54)

Like (4–49), the predicted $\hat{x}_{i+\ell}$ is

$$\hat{x}_{i+\ell} = Z^0 c_{i+\ell}$$  \hspace{1cm} (4–55)
where $Z^0 = [y_{1+\ell}^0, \ldots, y_{m+\ell}^0]$.

The form of the predictive conditional embedding operator $F_P$ is similar to the KRLS algorithm when the system has been learned well. Recalling the KRLS algorithm [16], the predicted output is

$$o_i = \omega^T \varphi(u_i)$$  \hspace{1cm} (4–56)

where $\omega$ is the weight in the RKHS which is learned from the training data, and $u_i$ is the input which is used to predict the output $o_i$. If the weight $\omega$ is learned from the training input set $[y_1^0, \ldots, y_m^0]$ and desired set $Z^0 = [y_{1+\ell}^0, \ldots, y_{m+\ell}^0]$, the weight should be

$$\omega = \Upsilon (K + \lambda I_m)^{-1} Z_0^T$$  \hspace{1cm} (4–57)

and the predicted output given input $y_i$ is

$$o_i = Z^0 (K + \lambda I_m)^{-1} \Upsilon^T \varphi(y_i).$$  \hspace{1cm} (4–58)

The approximated prediction conditional embedding operator $F_P$ has very similar form to the weight $\omega$, but with different regularization parameters.

However, they are different algorithms. The approximated prediction conditional embedding operator $F_P$ describes how the embeddings is propagated from time $i$ to time $i + \ell$, which operates on the current estimated embedding $\hat{\mu}_i$, while the weight $\omega$ specifies an underlying function, whose variable is the current input $u_i$. If and only if $F_P$ and $\omega$ are both trained based on the same training data, regularization parameters $\lambda = \zeta m$, and the measurement noise parameter $r = 0$, these two algorithms can have the same approximated prediction, because $b_i = 0$ and

$$c_{i+\ell} = (K + \zeta m I_m)^{-1} \Upsilon^T \varphi(y_i).$$  \hspace{1cm} (4–59)

By setting $r = 0$, it means that there is no difference between $\mu_i$ and the $\varphi(y_i)$ and the time series are not contaminated by noise. Therefore, compared with the KRLS
algorithm, the KKF-CEO algorithm is more appropriate for non-stationary and noisy environments.

4.4 Simplifying the conditional embedding operator

In the previous section, the KKF-CEO algorithm is developed to estimate and predict the time series from the noisy data. Like the traditional Kalman filter, this algorithm is implemented by maintaining some covariance matrices. However, the sizes of these matrices are all $m \times m$, where $m$ is the size of the training data that are used to estimate the state transition operator $F_i$, not the dimensionality of the hidden state. Therefore, the time and space complexities of this algorithm are $O(m^3)$ and $O(m^2)$, respectively. The time complexity increases cubically with the number of the training data, which may pose a big problem for the application of these algorithms. To reduce the time and space complexities, we have to reduce the number of the training samples in the proposed algorithm in a reasonable way. Several techniques are proposed in this section, including down sampling, quantizing, and online selecting the training data. These techniques can also be applied to other algorithms with similar operators, like the DSMCE [61] and KBR [82] algorithms.

For convenience, $y^0_{i+\ell}$ is denoted by $z^0_i$ in this section. We get $F_i$ when $\ell = 1$, while get $F_P$ when $\ell > 1$.

4.4.1 Down Sampling Training Data

According to (4–8) and (4–16), the conditional embedding operator is estimated based on $i.i.d.$ training data. Under this assumption, we can down sample uniformly the training data to reduce the size of the estimated conditional embedding operator. With the down sampling rate $L_s$, the sizes of the matrices become $k = \lfloor m/L_s \rfloor$ and the time and space complexities of filtering are $O(k^3)$ and $O(k^2)$, respectively.

4.4.2 Quantizing Training Data

Quantization techniques can also be applied to reduce the size of the conditional embedding operators. The idea is very similar to the QKLMS [83], where it has
demonstrated that for the Gaussian kernel good results are obtained with much fewer prototypes than samples.

Because the learning of the conditional embedding operators is not necessarily implemented online, we use the kernel K-means algorithm [11] to quantize the training data set in the RKHS, instead of using the online vector quantization (VQ) method mentioned in [83], which is implemented in the original training data set domain. Comparing with the online VQ method, there are several advantages by using the kernel K-means method. First, the number of the quantized data set is easy to control by the parameter \(k\), while for the online VQ method the number cannot be determined in advance. Second, the kernel K-means algorithm obtains the code book by updating based on the whole data set which can get better quantization performance, while the online VQ method generate the code book incrementally and the old quantized centers are not updated when new data arrive. Third, the kernel K-means algorithm is implemented in the RKHS based on the Gram matrix, which is the natural space to train the conditional embedding operator, while the online VQ method is controlled by a distance threshold parameter (quantization size \(\varepsilon\)) defined in the input space, which is not easy to choose.

Considering that estimated conditional embedding operators are constructed based on the mapped measurements \(\{\varphi(y_i^0)\}\) and \(\{\varphi(z_i^0)\}\), the kernel K-means algorithm should be implemented based on the pairs \(\{(\varphi(y_i^0), \varphi(z_i^0))\}\). There are two options to implement the quantization.

First, implement the kernel K-means algorithm in a direct product space. We use the concatenation of the mapped measurements \(\varphi(y_i^0)\) and \(\varphi(z_i^0)\) as a new vector \(\nu((y_i^0, z_i^0)) = (\varphi(y_i^0)^T, \varphi(z_i^0)^T) \in \mathcal{H}_y \oplus \mathcal{H}_y\), where \(\oplus\) denotes the direct product operator.
The inner product of these new vectors can be calculated as

\[
\langle \nu((y_i^0, z_i^0)), \nu((y_j^0, z_j^0)) \rangle_{H_y \oplus H_y} = \langle \varphi(y_i^0), \varphi(y_j^0) \rangle_{H_y} + \langle \varphi(z_i^0), \varphi(z_j^0) \rangle_{H_y}.
\] (4–60)

The gram matrix of this new vector \( \nu((y_i^0, z_i^0)) \) is denoted as \( K_\oplus \), which can be expressed as

\[
K_\oplus = K_1 + K_2
\] (4–61)

where \( K_1 \) and \( K_2 \) are the gram matrices of \( \varphi(y_i^0) \) and \( \varphi(z_i^0) \). Therefore, the gram matrix \( K_\oplus \) is nonnegative definite and the space \( H_y \oplus H_y \) is a RKHS.

Second, implement the kernel K-means algorithm in a tensor product space. The direct product RKHS \( H_y \oplus H_y \) is not an universal space, even if the RKHS \( H_y \) and \( H_y \) are universal. Now we can consider the tensor product RKHS \( H_y \otimes H_y \). The inner product in this RKHS is

\[
\langle \varphi(y_i^0) \otimes \varphi(z_i^0), \varphi(y_j^0) \otimes \varphi(z_j^0) \rangle_{H_y \otimes H_y} = \langle \varphi(y_i^0), \varphi(y_j^0) \rangle_{H_y} \langle \varphi(z_i^0), \varphi(z_j^0) \rangle_{H_y}.
\] (4–62)

and the gram matrix for this RKHS can be expressed by \( K_1 \) and \( K_2 \) as

\[
K_\otimes = K_1 \ast K_2
\] (4–63)

where \( \ast \) denotes the element-by-element product operator.

Because the kernel K-means algorithm can cluster data only based on the Gram matrix, we just use the new gram matrices \( K_\oplus \) or \( K_\otimes \) instead of the original gram matrix \( K \), then cluster the data into \( k \) bins. To quantize the data set, we can select one data pair \( (\varphi(y_i^0), \varphi(z_i^0)) \) as the code center to represent the data in this bin. For the \( k \) bins, we have the code book \( \{(\varphi(y_i^0), \varphi(z_i^0))\}_{i=1}^k \).

Based on the clustering results, we can estimate the quantized conditional embedding operator \( F_Q \).
Theorem 4.3. Given $m$ pairs of training examples $\mathcal{D}_{YZ} = \{ (y_1^0, z_1^0), \ldots, (y_m^0, z_m^0) \}$, which are quantized to $\mathcal{D}_{YZ}^Q = \{ (y_1^q, z_1^q), \ldots, (y_k^q, z_k^q) \}$, the quantized conditional embedding operator can be estimated as

$$F_Q = \Phi_Q \Lambda (K_Q \Lambda + \varsigma m I_k)^{-1} \Gamma_Q^T \tag{4–64}$$

where $\Gamma_Q = [\varphi(y_1^q), \ldots, \varphi(y_k^q)]$ and $\Phi_Q = [\varphi(z_1^q), \ldots, \varphi(z_k^q)]$. Here $\Lambda = \text{diag} [\lambda_1^q, \ldots, \lambda_k^q]$ is a diagonal matrix, where $\lambda_j^q (1 \leq j \leq k)$ is the number of the samples clustered in the $j$th bin by the kernel K-means algorithm and $K_Q = \Gamma_Q^T \Gamma_Q$.

The proof of Theorem 4 is given in APPENDIX C.

Assuming that in each bin the number of the samples clustered is the same, say $L_q = m/k$. Then the diagonal matrix $\Lambda$ is rewritten as $\Lambda = L_q I_k$. The quantized conditional embedding operator can be rewritten as

$$F_Q = \Phi_Q L_q I_k (K_Q L_q I_k + \varsigma m I_k)^{-1} \Gamma_Q^T$$

$$= \Phi_Q (K_Q + \varsigma I_k)^{-1} \Gamma_Q^T, \tag{4–65}$$

which is the same as the down sampling conditional embedding operator. It shows that the down sampling conditional embedding operator is a special case of the quantized conditional embedding operator.

According to the property of the kernel K-means algorithm, the sizes of the matrices should be equal to or less than the parameter $k$. Therefore, the time and space complexities of filtering are equal to or less than $O(k^3)$ and $O(k^2)$, respectively.

4.4.3 Training Data for Online Operation

In the previous subsections, the samples are selected from the whole data set in advance to construct the conditional embedding operators, and the operators are fixed during the filtering procedure. However, not all selected samples are important to construct the operators. The information of the current estimated embedding $\hat{\mu}_i$ should be considered to select the samples to construct the operators. Since the conditional
embedding operators are constructed directly from the data, we can update these operators with respect to the current estimated embedding $\hat{\mu}_i$, not fix them.

In order to online select the training data set to construct the estimated conditional embedding operator $F_{O,i}$ at iteration $i$, we can calculate the distance $d_n$ between the current estimated embedding $\hat{\mu}_i$ and all the training samples $\{y_n^0\}_{n=1}^m$ in the RKHS. Then we select the closest $k$ samples $\{y_{i,j}^0\}_{j=1}^k$ and the corresponding measurements $\{z_{i,j}^0\}_{j=1}^k$ to construct the conditional embedding operator at iteration $i$. The squared distance $d_n^2$ is calculated by substituting (4–45),

$$
\begin{align*}
    d_n^2 &= ||\hat{\mu}_i - \varphi(y_n^0)||^2 \\
    &= ||\Phi_{i-1}b_i + \frac{q}{q+r}\varphi(y_i) - \varphi(y_n^0)||^2 \\
    &= b_i^T\Phi_{i-1}\Phi_{i-1}b_i + \left(\frac{q}{q+r}\right)^2||\varphi(y_i)||^2 + ||\varphi(y_n^0)||^2 \\
    &\quad + \frac{2q}{q+r}b_i\Phi_{i-1}\varphi(y_i) - \frac{2q}{q+r}\varphi(y_i)^T\varphi(y_n^0) \\
    &\quad - 2b_i^T\Phi_{i-1}\varphi(y_n^0) (1 \leq n \leq m) \\
\end{align*}
$$

(4–66)

where $\Phi_{i-1} = [\varphi(z_{i-1,1}^0), \ldots, \varphi(z_{i-1,k}^0)]$ at iteration $i - 1$. Actually, to find the $k$ closest distances for different $y_n^0$, we just need to calculate

$$
\begin{align*}
    \Delta d_n^2 &= \frac{q}{q+r}\varphi(y_i)^T\varphi(y_n^0) + b_i^T\Phi_{i-1}\varphi(y_n^0) (1 \leq n \leq m) \\
\end{align*}
$$

(4–67)

and select $k$ largest $\Delta d_n^2$. Then we use the corresponding training measurement data $\{\varphi(y_{i,j}^0), \varphi(z_{i,j}^0)\}_{j=1}^k$ to construct new feature matrices $\gamma_i$, $\Phi$, and the online selected conditional embedding operator $F_{O,i}$ at iteration $i$.

For the sample selection part, the time and space complexities are $O(mk^2)$ and $O(m)$, respectively. For the filtering part, the time and space complexities are the same. Therefore, the time and space complexities of this method are $O(mk^2) + O(k^3)$ and $O(m) + O(k^2)$, respectively.
4.5 Experiments and Results

In this section two synthetic experiments and real-world dataset simulation are presented to evaluate the estimation and prediction performances of the novel algorithm KKF-CEO compared with some other existing algorithms.

4.5.1 Estimation of noisy time-series

The experimental data is the IKEDA chaotic dynamical system [68], which is related to laser dynamics. This time series is defined from a starting point \( x_0 = [x_0(1), x_0(2)]^T \) by

\[
\begin{align*}
    w_i &= c_1 - \frac{c_3}{1 + x_i^2(1) + x_i^2(2)} \\
    x_{i+1}(1) &= c_4 + c_2 (x_i(1) \cos(w_i) - x_i(2) \sin(w_i)) \\
    x_{i+1}(2) &= c_2 (x_i(1) \sin(w_i) + x_i(2) \cos(w_i))
\end{align*}
\]

where \( c_1, c_2, c_3 \) and \( c_4 \) are all real valued parameters. We set \( c_1 = 0.4, c_2 = 0.84, c_3 = 6.0, c_4 = 1.0 \) and \( x_0 = [1, 0]^T \). The first 200 points of the series IKEDA are plotted in Figure 4-2.

![Figure 4-2. Trajectory of the first 200 points of the series IKEDA](image)
In this experiment, we add noise to the clean IKEDA data set to obtain the noisy data \( \{ y_i \} \). We want to estimate the noiseless data from the noisy data. For all data sets, the data are split into \( n_{\text{train}} = 201 \) training data and \( n_{\text{test}} = 200 \) test data, including the noisy measurements and the corresponding clean signal.

The KKF-CEO algorithm presented in Section 4.3.1 is applied to the estimation problem, and its estimation performance is compared with the performances of the EKF, UKF, CKF and DSMCE algorithms.

### 4.5.1.1 Discussion of noise parameters

In order to select suitable parameters in our algorithm to obtain best estimation performances, we have to discuss how these parameters affect the estimation results. Among these parameters, \( q \) and \( r \) are the most important parameters, which reflect the state and measurement noises in the RKHS. Taking the Gaussian noise as example, we plotted the estimation results of 3 dB noisy signal for different pairs of \( q \) and \( r \) in Figure 4-3A. Here, the mean squares error (MSE) is calculated by

\[
MSE = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \| x_i - \hat{x}_i \|^2.
\]  

\[(4–69)\]

Figure 4-3. MSE of estimation results (a) with respect to \( q \) and \( r \) (b) with respect to \( \theta \) and \( \rho \)
The other algorithmic parameters, $\varsigma$ and $\lambda$ are set as $10^{-3}$ and $10^{-4}$, respectively. The Gaussian kernel is used here and the kernel size $\sigma$ is the median value of the pairwise distance of the training data set.

From Figure 4-3A one can find out that the estimation results are similar when the ratios between $q$ and $r$ are similar. Therefore, we re-plot the results with respect to the $\rho = \sqrt{q^2 + r^2}$ and $\theta = \tan^{-1}(r/q)$ in Figure 4-3B.

The figure clearly shows that only the ratio between the $r$ and $q$ affects the estimation results. Although this statement cannot be verified mathematically by Algorithm 4-2 because of the term $qr/(q+r)$, it is verified practically as long as the parameters $q$ and $r$ are both large enough to meet the following approximations (for $i = 1$)

$$
\tilde{G}_i = \left[ (q + r) I_m + \tilde{P}_i M \right]^{-1} \tilde{P}_i
$$

$$
\approx [(q + r) I_m]^{-1} \tilde{P}_i
$$

$$
\tilde{P}_i = \frac{r}{q + r} \tilde{P}_i - \frac{r}{q + r} \tilde{G}_i M \tilde{P}_i - \frac{qr}{q + r} \tilde{G}_i
$$

$$
\approx \frac{r}{q + r} \tilde{P}_i
$$

$$
b_i = \frac{r}{q + r} \left[ \left( I_m - \tilde{G}_i M \right) a_i + \tilde{G}_i \Phi^T \varphi(y_i) \right]
$$

$$
\approx \frac{r}{q + r} a_i
$$

$$
\tilde{P}_{i+1} = L \tilde{P}_i T^T L^T + \frac{qr}{q + r} LKL^T \approx \frac{qr}{q + r} LKL^T.
$$

Because the elements in $\tilde{P}_i$ and $M$ are all very small, this approximations always hold when $q, r > 10^{-4}$. With these approximations, one can show mathematically that for $i > 1$ the vectors $a_i$ and $b_i$ only depend on the ratio $r/q$. Even if approximations do not hold for some extreme cases (for example, $q, r < 10^{-20}$), the ratio $r/q$ still primarily controls the quality of the results. To present how the ratio affects the estimation results, we plot the MSE versus the different ratios $r/q = \{1/5, 1/4, 1/3, 1/2, 1, 2, 3, 4, 5\}$ in Figure 4-4. The ratio is a balance on how much the system trusts the estimated state.
transition operator and measurements, which should be determined based on prior knowledge.

Figure 4-4. MSE of estimation results versus $r/q$. Error bars mark one standard deviation above and below the mean.

These results are plotted based on 20 runs. They show that we can obtain the best estimation performance when $r/q$ is selected between 2 and 3.

4.5.1.2 Comparison of estimation results

For the EKF, UKF, and CKF algorithms, the system models are all assumed known, which should provide an advantage to these algorithms. No training data are required to learn the models. For the DSMCE algorithm, because the hidden state in the RKHS $\mu_x$ is the embedding of the hidden state $x$ and the conditional embedding operators involve the hidden states $\{x_i\}$ and measurements $\{y_i\}$, the training hidden states and the measurements are all required to construct the operators. While for our algorithm, because the hidden state in the RKHS $\hat{\mu}_y$ is the estimated measurement embedding, only the training measurements are required to construct the state transition operator.
In addition, the simplifying techniques discussed in Section IV are not utilized in this experiment.

We study how these different algorithms respond to various noise models. In particular, we scale and add four different noises to get the noisy signals with $3dB$ signal noise ratio (SNR), such as the Gaussian noise, Laplacian noise, zero-mean uniform noise and alpha stable noise [69, 70]. In the experiments, the parameters $q$ and $r$ are always set as 1 and 3, respectively, which makes the ratio $r/q = 3$.

For each trial, the algorithms are run 20 times to validate the mean and variance of the results. The MSE between the estimations $\hat{x}_i$ and real positions $x_i$ of the system are listed in Table 4-1. All the results in these tables are in the form of “mean ± standard deviation”, including Table 4-2 and 4-3.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Normal</th>
<th>Laplacian</th>
<th>zero-mean uniform</th>
<th>$\alpha$ stable noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF</td>
<td>0.3630±0.0448</td>
<td>0.3897±0.0407</td>
<td>0.3843±0.0308</td>
<td>0.3021±0.1232</td>
</tr>
<tr>
<td>UKF</td>
<td>0.2639±0.0218</td>
<td>0.2719±0.0217</td>
<td>0.2696±0.0213</td>
<td>0.2465±0.0969</td>
</tr>
<tr>
<td>CKF</td>
<td>0.2374±0.0176</td>
<td>0.2574±0.0199</td>
<td>0.2427±0.0193</td>
<td>0.2580±0.0991</td>
</tr>
<tr>
<td>DSMCE</td>
<td>0.3918±0.0502</td>
<td>0.3555±0.0626</td>
<td>0.3945±0.0800</td>
<td>0.2319±0.1335</td>
</tr>
<tr>
<td>KKF-CEO</td>
<td>0.2253±0.0168</td>
<td>0.2121±0.0202</td>
<td>0.2384±0.0180</td>
<td>0.1461±0.0413</td>
</tr>
</tbody>
</table>

It is clear that the estimation results of the KKF-CEO algorithm outperforms the other algorithms in all noisy environments. Especially, in the $\alpha$ stable noisy environment the advantage of our algorithm is very obvious. Even in the normal noisy environment, our algorithm can still obtain the best estimation performance. That is because the system is nonlinear and the Gaussian noise assumption is not satisfied in this case.

In the previous experiments, the SNR of the input signal is always 3dB. In order to figure out how these algorithms respond in more noisy environments, we use the signal contaminated by the different normal noises with SNRs from -15dB to 3dB to test these algorithms. For the KKF-CEO algorithm, the noise parameter $q = 1$ and $r$ is set the
different values \{256, 128, 60, 40, 12.5, 6, 3\} to obtain good performances according to the different SNRs. The results are plotted in Figure 4-5.

![Figure 4-5. SNR of estimation results versus input signal. Error bars mark one standard deviation above and below the mean.](image)

From Figure 4-5 one can find that the KKF-CEO algorithm has the best estimation performances when \(\text{SNR} \geq -12\) dB. Even in low SNR environment (SNR = -12dB), this algorithm can still obtain a good result. Although the EKF, UKF and CKF algorithms can obtain acceptable performance when the input signal is less noisy (SNR = 3dB), these algorithms cannot work well in the larger noise environment. For the DSMCE algorithm, its performances are always inferior to the performances of the KKF-CEO algorithm (except SNR = -15dB), especially in less noisy environment. In such noisy environment (SNR = -15dB), the system dynamics is distorted seriously and cannot be learned well from the noisy measurement using our algorithm, while the DSMCE, which uses the clean hidden state training data to construct the operators which preserve the dynamics of the signal, can learn more about the system dynamics.
4.5.2 Prediction of noisy time-series

To evaluate the prediction performance of the algorithm presented in Section 4.3.2, our algorithm is applied to predict the noisy Lorenz time-series. The Lorenz attractor, introduced by Edward Lorenz in 1963 [74], is a dynamical system corresponding to the long-term behavior of a chaotic flow and is well recognized by its butterfly shape. The system is nonlinear, three-dimensional and deterministic, which is described by three ordinary differential equations known as the Lorenz equations:

\[
\begin{align*}
\frac{dx(1)}{dt} &= -\beta x(1) + x(2)x(3) \\
\frac{dx(2)}{dt} &= \sigma(x(3) - x(2)) \\
\frac{dx(3)}{dt} &= -x(1)x(2) + \rho x(2) - x(3).
\end{align*}
\]

It is proven by Tucker that for a certain set of parameters the system exhibits chaotic behavior and displays what is today called a strange attractor [75]. Here, the experimental data are generated by setting $\beta = 8/3$, $\sigma = 10$ and $\rho = 28$. The first order approximation is used with a step size 0.01 to obtain the signal $x_i = [x_i(1), x_i(2), x_i(3)]^T$. The Lorenz data are plotted in Figure 4-6.

Like the previous experiment, we add Gaussian white noise to the clean Lorenz time-series signal $\{x_i\}$ to obtain the noisy signal $\{y_i\}$ with -15dB SNR, and want to predict the noiseless data in the future from the current noisy data and the training data. For all the data sets, the data are split into $n_{\text{train}} = 1500$ training data $\{y_i^0, y_{i+1}^0, y_{i+\ell}^0\}$ and $n_{\text{test}} = 200$ test data $\{y_i, x_{i+\ell}\}$, where $\ell$ is the prediction step size.

The KRLS algorithm achieves a very good performance in the prediction problem [76], outperforming the LMS, RLS, EX-RLS and KLMS algorithms, and the Ex-KRLS algorithm, which is just a random walk KRLS algorithm, has very similar behavior to the KRLS algorithm. Therefore, we compare the prediction performance of our novel algorithm only with KRLS. For the KRLS algorithm, we just need to learn the underlying function, which is used to predict the future signal from the current noisy measurement.
However, for our algorithm, we construct the conditional embedding operator as the state transition operator in the RKHS using the training data. If we use the whole training data directly, then the size of the matrices that we have to maintain in the algorithm is $m \times m$, where $m = 1500$. The time and space complexities are so high that the algorithm is not practical. Therefore, the simplifying techniques mentioned in Section 4.4 are applied to estimate the current embedding $\mu_i$. We still use all the training data set to construct the prediction operator $F_P$. In other words, the KRLS algorithm uses the current measurement $y_i$ to predict $x_{i+\ell}$, while we use the current estimated embedding $\hat{\mu}_i$ to predict $x_{i+\ell}$. For the KKF-CEO$_Q$ algorithm, $K_\otimes$ in (4–63) is used to implement the kernel K-means algorithm.

For the KRLS algorithm, the regularization factor is set at $\zeta = 10^{-5}$, $\lambda = 10^{-3}$, and forgetting factor $\beta = 1$. For our algorithm, $\zeta = 10^{-5}$ and $\lambda = 10^{-3}$, the noise parameters $q = 0.2$ and $r = 0.5$, which makes the ratio $r/q = 2.5$. The Gaussian kernel is applied for all algorithms and the kernel parameters are all set as $10^{-3}$.
To figure out how the size of the maintaining matrices $k$ affects the prediction performances, we use different $k$ values to predict $x_{i+\ell}$ when $\ell = 10$. The results are plotted in Figure 4-7, by setting $k = [30, 40, \ldots, 100]$. Here the MSE is defined as the error between predicted $\hat{x}_{i+\ell}$ and the corresponding clean signal $x_{i+\ell}$.

![Figure 4-7. 10-step Prediction MSE with different $k$](image)

Here KKF-CEO$_D$, KKF-CEO$_Q$ and KKF-CEO$_O$ denote the algorithms with the down sampling training data, the quantized training data and online selected training data, respectively.

The results show: First, our algorithms with different simplifying techniques all outperform the KRLS algorithm. With the same network size $k$, the KKF-CEO$_O$ algorithm has best prediction performance. Even for small network size $k = 30$, the KKF-CEO$_O$ algorithm works very well. Second, the differences among these three algorithms decrease as $k$ increases. It makes sense, because, when $k = m$, then these three algorithms should have the same behavior.
The prediction performances for different prediction steps are also presented. By setting $k = 50$ and $k = 100$, the prediction performances of these algorithms are plotted in Figure 4-8 and Figure 4-9 and tabulated in Table 4-2 and 4-3.

![Figure 4-8. Prediction MSE with $k = 50$](image)

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>1-Step prediction</th>
<th>10-Step prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRLS</td>
<td>5.4081±0.75559</td>
<td>7.6784±1.3396</td>
</tr>
<tr>
<td>KKF-CEO_D</td>
<td>4.9681±0.94691</td>
<td>6.3667±1.2723</td>
</tr>
<tr>
<td>KKF-CEO_Q</td>
<td>4.2606±0.67791</td>
<td>5.8732±1.1518</td>
</tr>
<tr>
<td>KKF-CEO_O</td>
<td>3.8708±0.47701</td>
<td>5.6403±0.93988</td>
</tr>
</tbody>
</table>

Table 4-3. Prediction MSE with $k = 100$

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>1-Step prediction</th>
<th>10-Step prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRLS</td>
<td>5.4081±0.75559</td>
<td>7.6784±1.3396</td>
</tr>
<tr>
<td>KKF-CEO_D</td>
<td>4.2162±0.62487</td>
<td>5.5421±0.91026</td>
</tr>
<tr>
<td>KKF-CEO_Q</td>
<td>3.9040±0.54787</td>
<td>5.4294±0.95342</td>
</tr>
<tr>
<td>KKF-CEO_O</td>
<td>3.5771±0.45949</td>
<td>5.253±0.88234</td>
</tr>
</tbody>
</table>

The prediction result of the KKF-CEO_O algorithm outperforms all the other algorithms for all the prediction steps when the network sizes $k$ are the same.
4.5.3 Prediction of Sunspot data set

In this subsection, we apply our algorithm to predict the real-world data, the sunspot data, and the prediction performance is compared with the KRLS-ALD [16] and QKRLS [84] algorithms.

The Wolf annual sunspot count time series has been the subject of interest in physics and statistics communities and has also been the benchmark data in time series prediction problem [85]. The normalized annual sunspot data over the period 1700-2011 is plotted in Figure 4-10.

The Sunspot data are split into training data over the period 1700-1979 and test data over the period 1980-2011. The training data set is used to train the kernel systems to implement 1-step prediction tested on the testing data set. Because the Sunspot data \( \{x_i\} \) are 1-Dimensional time series, we have to use the embedding vector \( x_i = [x_{i-(n_E-1)\tau}, \ldots, x_{i-\tau}, x_i] \) to predict \( x_{i+1} \) according to Takens’ embedding theorem [64]. In this experiment, these parameters are set as \( n_E = 9 \) and \( \tau = 1 \), respectively.
Like the previous experiments, the Gaussian kernel is applied to these three kernel algorithms with the same kernel parameter $\varrho = 1$. For the KRLS-ALD and QKRLS algorithms, the same forgetting factor $\beta = 1$ and regularization term $\zeta = 1e^{-3}$ are applied. The ALD threshold $\delta_{\text{ALD}}$ and the quantized size $\varepsilon$ are set as different values to obtain different network sizes. For the KKF-CEO algorithm, we chose the same samples in the sparsification dictionary as the KRLS-ALD to construct the state transition operator $F$ and 1-step prediction operator $F_P$. The regularization term is set as same as the other algorithms $\zeta = 1e^{-3}$. The noise parameters are set as $q = 1$ and $r = 2$ for all the network size, which makes the ratio $r/q = 2$.

The prediction normalized mean squared error (NMSE) for different network sizes are plotted in Figure 4-11.

One can find that that the advantage of the KKF-CEO algorithm over the other two algorithms is obvious when larger network size is used. When the network size is small, the transition operator constructed by less samples in the KKF-CEO algorithm cannot
describe the system dynamics sufficiently. When the training data are sufficient to
describe the dynamics, the KKF-CEO algorithm has more advantage. That is because
real-world data like the Sunspot data are typically contaminated with outliers and
require treatment by distributions that have longer and/or thicker tails than the Normal
(Gaussian) distribution. The KKF-CEO algorithm, therefore, is more suitable in such
cases.

4.6 Discussion and Conclusion

In this chapter, the Kalman filter for denoising and prediction is implemented
in the RKHS to deal with nonlinear estimation problems. The embeddings of the
measurements are used as the hidden states in the RKHS which are propagated by the
conditional embedding operator and updated by the new transformed measurements.
The propagation of the embeddings, or we can say the propagation of the pdf, results in
the survival of the filtering algorithm in non-Gaussian noise environments.
Unlike the classical nonlinear Kalman filter, such as the EKF, UKF, and CKF, which require the knowledge of the system model, the transition operator is constructed from the observations directly. Therefore, no hidden states information or system model prior knowledge are required. Furthermore, because the Gaussian noise is also modeled in the RKHS, the model can cover a very wide range of input space noises, not only the Gaussian noise. In addition, the propagation of the pdf is described using the data directly, not from assumed system models, which makes the algorithm more suitable for complicated nonlinear system functions or low SNR cases.

The novel algorithm is applied to estimate the noisy IKEDA time-series. The estimation performances under different noise environments are compared with some existing algorithms and outperform them, especially for the non-Gaussian noise. There are three reasons to explain the advantage. First, the distributions of the measurements are propagated very well via the transition operator, which is constructed from the noisy measurements, no matter what kind of noise. Second, the distributions of the measurements are used to calculate the estimates, not just the first and second order of the statistics like the EKF, UKF and CKF. Third, noises in the RKHS are brought into the algorithm to reflect the uncertainty of the transition operator and measurement noise in the RKHS.

The algorithm is also utilized to predict the noisy Lorenz time-series. The prediction performances outperform the KRLS algorithm. For the Lorenz time-series data set, more training data are required to learn the system than the IKEDA data set, which incurs high complexities. In order to bound the complexities, three techniques, including down sampling (D), quantization (Q) and online operation (O), are proposed. All of them can offer better prediction performances even with a smaller size of the transition operator. The advantage primarily results from the fact that the embeddings of the current measurements are used to predict the output in the future, not the measurement itself. However, compared with the KRLS algorithm, our algorithm incurs higher complexities.
to maintain the embeddings and calculate the prediction. The time complexities are summarized in Table 4-4.

Table 4-4. Time complexities

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Estimation of embeddings</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRLS</td>
<td>N/A</td>
<td>$O(m^2)$</td>
</tr>
<tr>
<td>KKF-CEO_D</td>
<td>$O(k^3)$</td>
<td>$O(m^2) + O(mk)$</td>
</tr>
<tr>
<td>KKF-CEO_Q</td>
<td>$O(k^3)$</td>
<td>$O(m^2) + O(mk)$</td>
</tr>
<tr>
<td>KKF-CEO_O</td>
<td>$O(k^3) + O(mk^2)$</td>
<td>$O(m^2) + O(mk)$</td>
</tr>
</tbody>
</table>

From this table, one can find out that all the KKF-CEO algorithms have similar time complexities of prediction to the KRLS algorithm, because $m > k$. In addition, although the KKF-CEO_O algorithm can outperform other algorithms when $k$ is the same, it has the highest computational complexity.

The advantage of this algorithm is also shown by the real-world data application, the Sunspot data set short term prediction. The prediction performance of the KKF-CEO algorithm outperforms the KRLS-ALD and QKRLS algorithms when enough training data are applied.

The Kalman filter is constructed in the RKHS to solve nonlinear estimation and prediction problems. However, because the measurement embeddings are treated as the hidden states in the RKHS, not the hidden states of the physical model, the algorithm is not able to precisely model the dynamics. Although, the uncertainty in the model can compensate for this, it will always have limitations to deal with complicated system. Therefore, This methodology need to be extended in the future to develop the Kalman filter in RKHS with a general measurement model to implement hidden states estimation, which will require the hidden states as training data to approximate the state transition and measurement operators.

Another difficulty that should be further researched deals with the constrained control of the noise model in RKHS. In fact so far only the power of the noise is under control with $q$ and $r$ and this affects all the dimensions of the RKHS. In addition, the ratio $r/q$ has to be determined based on the prior knowledge of the noise in the time series to
obtain good results. In many practical cases, one would like to have control the noise in each dimension of the RKHS, and this can only be done with a tensor product kernel in the current approach.
CHAPTER 5
KERNEL KALMAN FILTER BASED ON CONDITIONAL EMBEDDINGS

5.1 Overview

In Chapter 4 we develop a novel Kalman filter algorithm in the RKHS based on the conditional embedding operator to deal with nonlinear dynamic system. However, as discussed in Chapter 4, the measurement embeddings are treated as the hidden states in the RKHS, not the hidden states of the physical model, so the algorithm is not able to precisely model the dynamics. Moreover, the measurement function in that algorithm is assumed to be an identity function, which is too simple for some applications. Therefore, in this chapter we will propose another algorithm to deal with nonlinear dynamic systems with a complex measurement model, which is assumed to be any nonlinear function.

The state-space model is assumed to be known as

\[
\begin{align*}
x_{i+1} &= f(x_i) + n_i \\
y_i &= h(x_i) + v_i
\end{align*}
\] (5–1)

where \( f(\cdot) \) and \( h(\cdot) \) are known state transition and measurement functions, respectively; \( n_i \) and \( v_i \) are state noise and measurement noise, respectively.

This algorithm is also developed based on the conditional embedding operators, but the hidden state in RKHS will be the state embeddings, instead of the measurement embeddings. We need the hidden states as training data to construct the state transition and measurement operators, which are both conditional embedding operators. However, in many cases this hidden states are not available as training data, but as an assumed system model. Therefore, the new algorithm is developed under the assumption that system model is given. We can use the system model to generate hidden states and corresponding measurements as training data to construct the operators. With these operators, we can estimate the state embeddings directly from system measurements as done in Chapter 4.
Since the RKHS algorithm developed under the assumption that the system model (generally the state-space model) is assumed to be known, coincides with the assumption of the Kalman filter and nonlinear Kalman filters, we can compare these algorithms more fairly. Therefore, we name this algorithm the full-blown kernel Kalman filter (FKKF).

The rest of the chapter is organized as follows. In Section 5.2 the state-space model in RKHS is constructed with the conditional embedding operators. In section 5.3 the state transition and measurement operators are estimated and the FKKF algorithm is proposed. Then, an experiment is presented in Section 5.4. Following the experiment, the results are discussed in Section 5.5. Finally, the conclusion is given in Section 5.6.

5.2 State-Space Model in RKHS

In this section, we assume that the training data set including hidden states and corresponding measurements are available as training data set to construct the state-space model in RKHS.

Given the training set \( D = \{ (x_1^0, y_1^0), \ldots, (x_{m+1}^0, y_{m+1}^0) \} \), generated by the system modeled by following equations

\[
\begin{align*}
x_{i+1} &= f(x_i) \quad (5–3) \\
y_i &= h(x_i), \quad (5–4)
\end{align*}
\]

we map these training data into two RKHS \( \mathcal{H}_x \) and \( \mathcal{H}_y \) as \( \varphi(x_i) \) and \( \phi(y_i) \). Then, we can estimate the conditional embeddings operators \( F_i \in \mathcal{H}_x \times \mathcal{H}_x \) and \( H_i \in \mathcal{H}_y \times \mathcal{H}_x \) for the state model and measurement model as below:

\[
\begin{align*}
F_i &= \Phi (K + \varsigma_f m l) \Gamma^T \quad (5–5) \\
H_i &= \Psi (T + \varsigma_h m l) \Phi^T \quad (5–6)
\end{align*}
\]
where \( \gamma = [\varphi(x_0^0), \ldots, \varphi(x_m^0)], \Phi = [\varphi(x_0^2), \ldots, \varphi(x_{m+1}^0)], \Psi = [\psi(y_0^0), \ldots, \psi(y_{m+1}^0)], \)
\( K = \gamma^T \gamma, \) and \( T = \Phi^T \Phi. \)

We denote the state and measurement embeddings by \( \mu_x \) and \( \mu_y, \) respectively, and the estimates are denoted by \( \hat{\mu}_x \) and \( \hat{\mu}_y. \) Like the conditional embedding operator defined in Chapter 4, these two operator can also link these estimated embeddings as

\[
\hat{\mu}_{x,i+1} = F_i \hat{\mu}_{x,i} \tag{5-7}
\]
\[
\hat{\mu}_{y,i} = H_i \hat{\mu}_{x,i}. \tag{5-8}
\]

Since the samples generated by (5–3) and (5–4), the operators \( F_i \) and \( H_i \) reflect how the state transition and measurement functions \( f(\cdot) \) and \( h(\cdot) \) propagate the embeddings \( \hat{\mu}_{x,i} \) and \( \hat{\mu}_{y,i}, \) respectively.

Considering the state and measurement noise we add noises in (5–7) and (5–8) and obtain following equations

\[
\hat{\mu}_{x,i+1} = F_i \hat{\mu}_{x,i} + \xi_i \tag{5–9}
\]
\[
\hat{\mu}_{y,i} = H_i \hat{\mu}_{x,i} + \nu'_i \tag{5–10}
\]

where \( \xi_i \in \mathcal{H}_x \) and \( \nu'_i \in \mathcal{H}_y \) are both noise in RKHS.

Comparing with (5–2), one can find that \( \hat{\mu}_{y,i} \) is supposed to be the measurement in the RKHS state-space model. However, \( \hat{\mu}_{y,i} \) is the estimated measurement embedding at iteration \( i, \) which is unobservable. Only the mapped measurement \( \psi(y_i) \) can be observed, which is an instantiation of the measurement embedding \( \mu_{y,i}. \) We link the estimate \( \hat{\mu}_{y,i} \) and the instantiation \( \psi(y_i) \) by

\[
\psi(y_i) = \hat{\mu}_{y,i} + \nu''_i. \tag{5–11}
\]
where $\nu''_i$ is also noise in $H_y$. Then, we have the RKHS state-space model as

$$
\hat{\mu}_{x,i+1} = F_i \hat{\mu}_{x,i} + \xi_i \tag{5–12}
$$

$$
\psi(y_i) = H_i \hat{\mu}_{x,i} + \nu'_i + \nu''_i. \tag{5–13}
$$

For convenience, we rewrite the above equation as

$$
\hat{\mu}_{i+1} = F_i \hat{\mu}_i + \xi_i \\
\psi(y_i) = H_i \hat{\mu}_i + \nu_i \tag{5–14}
$$

where $\hat{\mu}_i = \hat{\mu}_{x,i}$ and $\nu_i = \nu'_i + \nu''_i$. These noises in RKHS are also treated as high dimensional (or infinite dimensional) vectors, and can be assumed to be zero-mean Gaussian with covariance

$$
Q_i = q I \tag{5–15}
$$

$$
R_i = r I. \tag{5–16}
$$

(5–14) is the RKHS state-space model, which is used to develop the FKKF algorithm in this chapter. For the rest of this chapter, we denote the estimated state embedding by $\hat{\mu}_i$, which is the hidden state of the RKHS state-space model. In the next section, we will derive the FKKF algorithm to estimate the hidden state $\hat{\mu}_i$, then obtain hidden state estimate $\hat{x}_i$ from $\hat{\mu}_i$.

### 5.3 Derivation of Full-blown Kernel Kalman Filter

Like the derivation of the KKF-CEO algorithm, we develop the FKKF algorithm based on the RKHS state-space model (5–14) in this section. First, we derive the recursive equations following the traditional Kalman filter based on the given training data. Then, we propose the FKKF algorithm with generated training data from the state transition and measurement functions at each iteration.
5.3.1 Recursive Equations with Given Training Data

Comparing the traditional model in (2–5) with the RKHS state-space model in (5–14), we can see that \( F_i = \Phi (K + \varsigma_f m) \gamma_i^T \) and \( H_i = \Psi (T + \varsigma_h m) \Phi_i^T \), \( Q_i = q I \) and \( R_i = r I \). Therefore, the recursion to estimate \( \mu_i \) in the RKHS will be written as follows:

Start with \( \hat{\mu}_0 = E[\varphi(x_0)] \) or \( \hat{\mu}_0 = \varphi(x_0) \), \( P_0 = \lambda I \) like the Kalman filter [26],

\[
\hat{\mu}_i^- = F_{i-1}\hat{\mu}_{i-1} \tag{5–17}
\]

\[
P_i^- = F_{i-1}P_{i-1}F_{i-1}^T + Q_{i-1} \tag{5–18}
\]

\[
G_i = P_i^-H_i^T [H_i P_i^- H_i^T + R_i]^{-1} \tag{5–19}
\]

\[
\hat{\mu}_i = \hat{\mu}_i^- + G_i (\varphi(y_i) - H_i \hat{\mu}_i^-) \tag{5–20}
\]

\[
P_i = (I - G_i H_i) P_i^- \tag{5–21}
\]

where \( \hat{\mu}_i^- \) denotes a priori estimate of the state embedding. Here we have operators \( P_i^- \in \mathcal{H}_x \times \mathcal{H}_x \), \( P_i \in \mathcal{H}_x \times \mathcal{H}_x \), and \( G_i \in \mathcal{H}_x \times \mathcal{H}_y \). As discussed in Chapter 4, this recursive approach cannot be computed directly using normal matrix calculations. However, like the KKF-CEO algorithm, the recursion approach can still be implemented due to the following theorems we have proved.

**Theorem 5.1.** The operator \( P_i^- \), \( P_i \), \( G_i \) are of the following forms

\[
P_i^- = \Phi_i \Phi_i^T + q I \tag{5–22}
\]

\[
P_i = \Phi_i \Phi_i^T + q I \tag{5–23}
\]

\[
G_i = \Phi G_1 \Psi^T \tag{5–24}
\]

where \( \Phi_i \), \( \Phi \), and \( G_i \) are all \( m \times m \) matrices. Furthermore, these matrices can be computed recursively as follows:

\[
\Phi_i = \Phi_i \Phi_i^T + q I \tag{5–25}
\]

\[
G_i = \left[ r I_m + \left( \Phi_i \Phi_i^T + q H_i^T \right) \Psi^T \Psi \Phi_i^T \Phi_i \right]^{-1} \left( \Phi_i \Phi_i^T \Phi_i^T + q H_i^T \right) \tag{5–26}
\]
\[ \tilde{P}_i = \tilde{P}_i - q\tilde{G}_i\psi^T\psi\tilde{H} - \tilde{G}_i\psi^T\psi\tilde{H}\phi^T\phi\tilde{P}_i \]  
\[ \tilde{P}_{i-1} = \tilde{F} \left( \gamma^T\phi\tilde{P}_{i-1}\phi^T\gamma + q\gamma^T\gamma \right) \tilde{F}^T \]

where \( \tilde{F} = (K + \varsigma f\mu I)^{-1}, \tilde{H} = (T + \varsigma h\mu I)^{-1}, \tilde{F} = \phi \tilde{F}^T \text{ and } H = \psi \tilde{H}^T \). 

**Theorem 5.2.** The predicted embeddings \( \tilde{\mu}_i \) and the estimated embeddings \( \tilde{\mu}_i \) are both combinations of the mapped states \( \{\varphi(x_i^0)\}_{i=2}^{m+1} \)

\[ \tilde{\mu}_i = \Phi a_i \]  
\[ \tilde{\mu}_i = \Phi b_i \]

where \( a_i = [a_1, ..., a_m]^T \) and \( b_i = [b_1, ..., b_m]^T \) are both \( m \times 1 \) real-valued vectors. 

Furthermore, these vectors can also be computed recursively as follows:

\[ a_1 = \tilde{F}\gamma^T\mu_0 \]  
\[ b_i = \left( a_i + \tilde{G}_i\psi^T\psi(y_i) - \tilde{G}_i\psi^T\psi\tilde{H}\phi^T\phi a_i \right) \]  
\[ a_i = \tilde{F}\gamma^T\phi b_{i-1} \]

Proofs of **Theorem 5.1** and **Theorem 5.2** are presented in APPENDIX D. By these theorems, the operators are all expressed in term of \( m \times m \) matrices and feature matrices, such as \( \Phi \) and \( \gamma \), which can both be treated as \( n_k \times m \) matrices. Therefore, the calculations from (5–17) to (5–21) only involve normal matrices calculations and inner product operations between feature matrices and vectors, which can be computed readily by the kernel trick.
Once we estimate the state embedding $\mu_i$, like (4–49), we can compute the hidden state estimate $\hat{x}_i$ by

$$
\hat{x}_i = E[x_i] \\
= E[(f_i, \varphi(x_i))] \\
= \langle f_i(\cdot), E[\varphi(x_i)] \rangle \\
= \langle f_i(\cdot), \hat{\mu}_i \rangle \\
= \langle f_i(\cdot), \Phi b_i \rangle 
$$

(5–34)

where $f_i = [f_{x,1}, \ldots, f_{x,n_x}]$ is a row vector of functions and $f_{x,j}(x) = x^{(j)} (j = 1, \ldots, n_x)$. $x^{(j)}$ is the $j$th component of the vector $x$. Although the function $f_{x,j}(\cdot)$ may not belong to the RKHS, we can learn and approximate them from the training data $X^0$, where $X^0 = [x_0^0, \ldots, x_{m+1}^0]$. The learned row vector of functions $f_i(\cdot)$ can be expressed as

$$
\hat{f}_i(\cdot) = \phi(\phi^T\phi)^{-1}X^0T. 
$$

(5–35)

Then the above equation can be approximated as

$$
\hat{x}_i = b_i^T\phi^T\phi(\phi^T\phi)^{-1}X^0T = X^0b_i. 
$$

(5–36)

At this point, we can estimate the hidden state $x_i$ using the given training data set $D$, which are generated by the state transition and measurement functions.

### 5.3.2 FKKF algorithm with generated training data

To implement the FKKF algorithm, we need the conditional embedding operators $F_i$ and $H_i$ to construct the state and measurement model in RKHS. Furthermore, these operators have to be learned from training data, which are generated from the known state-space model functions. The simplest way to generate samples from the state transition and measurement functions is to find the fastest rate of change of the function and select the samples with the appropriate range. However, this would be wasteful in all the other parts of the domain. Therefore, we propose here to use the local
information about the current estimated state embedding to generate new samples and construct the state transition and measurement operators at each iteration. This idea of samples generation is similar to the idea of the online selected sampling technique, which we proposed in Section 4.4.3. The difference is that we generate samples around the previous estimate hidden state $\hat{x}_{i-1}$, not select samples from the estimated embedding.

At iteration $i$, we generate $m$ i.i.d. samples $X_{\text{Pre},i}^0 = \{x_{\text{pre},i,j}^0\}_{j=1}^m$ around estimated hidden state $\hat{x}_{i-1}$; then, feed $X_{\text{Pre},i}^0$ to the state transition function $f(\cdot)$ to obtain $X_i^0 = f(X_{\text{Pre},i}^0) = \{x_{i,j}^0\}_{j=1}^m$; finally, feed $X_i^0$ to the measurement function $h(\cdot)$ to output $Y_i^0 = h(X_i^0) = \{y_{i,j}^0\}_{j=1}^m$. Using these generated training data, we can construct the state transition and measurement operator $F_i$ and $H_i$ respectively at iteration $i$ by

$$F_i = \Phi_i \tilde{F}_i \Gamma_i$$

(5–37)

$$H_i = \Phi_i \tilde{H}_i \Phi_i$$

(5–38)

where $\Gamma_i = [\phi(x_{\text{pre},i,1}^0), \ldots, \phi(x_{\text{pre},i,m}^0)], \Phi_i = [\phi(x_{i,1}^0), \ldots, \phi(x_{i,m}^0)], \Psi_i = [\psi(y_{i,1}^0), \ldots, \psi(y_{i,m}^0)]$.

The $m \times m$ matrices $\tilde{F}_i$ and $\tilde{H}_i$ are expressed as

$$\tilde{F}_i = (K_i + \varsigma f m I)^{-1},$$

(5–39)

$$\tilde{H}_i = (T_i + \varsigma h m I)^{-1}$$

(5–40)

where $K_i = \Gamma_i^T \Gamma_i$ and $T_i = \Phi_i^T \Phi_i$.

To generate samples $X_{\text{Pre},i}^0$, we assume a Gaussian distribution with mean $\hat{x}_{i-1}$ and covariance $\Sigma_i$. It is noteworthy that the covariance $\Sigma_i$ is affected by the state transition function $f(\cdot)$. If the the distribution propagation property of this function changes slowly, we can choose a larger $\Sigma_i$, if not a smaller $\Sigma_i$ is required.
Algorithm 5-1: Full-blown Kernel Kalman Filtering

Initialization: For $i = 0$, Set
regularization parameters: $\varsigma_f$ and $\varsigma_h$
number of generated training samples: $m$
initial hidden state: $x_0$ and $\mu_0 = \varphi(x_0)$
set $\Phi_0 = \varphi(x_0)$ and $b_0 = 1$, then $\mu_0 = \Phi_0 b_0$
initial covariance: $P_0 = \lambda I$,

Filtering: For $i > 0$
Generate training data for $i$ iteration:
generate $m$ i.i.d. samples: $X_{\text{Pre},i} \sim \mathcal{N}(\tilde{x}_{i-1}, \Sigma_i)$
feed $X_{\text{Pre},i}$ to (5–3) and (5–4) to generate samples: $X_i = \{x_i\}_{j=1}^m$, and $Y_i = \{y_i\}_{j=1}^m$
feature matrices: $\gamma_i$, $\Phi_i$ and $\psi_i$
Compute: $K_i = (\gamma_i^T \gamma_i + \varsigma_f m I) ^{-1}$ and $\tilde{H}_i = (\gamma_i^T + \varsigma_h m I) ^{-1}$
a $i = \tilde{F}_i \gamma_i^T \Phi_j b_j$, and $b_i$
if $i = 1$, $\tilde{P}_i = \lambda \tilde{F}_i \kappa_i \tilde{F}_i^T$, else $\tilde{P}_i = \tilde{F}_i \left( \gamma_i^T \Phi_i \tilde{P}_{i-1} \Phi_i^T \gamma_i + q \kappa_i \right) \tilde{F}_i^T$
$G_i = \left[ r I_m + \left( \tilde{P}_i^T \tilde{H}_i^T + q \tilde{H}_i^T \right) S_i \tilde{H}_i \tilde{T}_i \right] ^{-1} \left( \tilde{P}_i^T \tilde{H}_i^T + q \tilde{H}_i^T \right)$
b $i = \left( a_i + G_i \psi_i^T \psi(y_i) - G_i S_i \tilde{H}_i \tilde{T}_i a_i \right)$
$\tilde{P}_i = \tilde{P}_i - q G_i S_i \tilde{H}_i G_i - G_i S_i \tilde{H}_i \tilde{T}_i \tilde{P}_i$
Estimate hidden state: $\hat{x}_i = X_i b_i$

Because the state embedding at iteration $i$ is obtained by using $F_i$ and $H_i$, $\hat{\mu}_i$ and $\hat{\mu}_i$ are both expressed by $\Phi_i$. Therefore, (5–29) and (5–29) should both be rewritten as

$$\hat{\mu}_i = \Phi_i a_i \quad (5–41)$$

$$\hat{\mu}_i = \Phi_i b_i \quad (5–42)$$

Likewise, with these operators $F_i$ and $H_i$, constructed by the generated training data at each iteration, these modifications are also needed in the other recursive equations presented in THEOREM 5.1 and THEOREM 5.2.

Finally, we have the FKKF algorithm with the generated training data at each iteration. The FKKF algorithm is summarized in Algorithm 5-1.

The computation complexity of the FKKF algorithm is $O(m^3)$ because of the matrix calculations, same as the KKF-CEO estimation part.
5.4 Experiment and Result

The FKKF proposed algorithm in Algorithm 5-1 is our first attempt of developing the full blown Kalman filter in RKHS. This algorithm is just preliminary, not complete. In this section, we apply this preliminary algorithm to a simple synthetic system problem to show that it works. However, to obtain a complete algorithm, more studies and modifications are needed.

The performance of the FKKF algorithm is evaluated on a synthetic scalar estimation problem and compared with some existing algorithms including the EKF, UKF, CKF and PF.

We consider a nonlinear dynamical system described by the following state-space model:

\[ x_{i+1} = f(x_i, i) = \frac{x_i}{2} + \frac{25x_i}{1 + x_i^2} + 8 \cos(1.2i) + n_i \]  
\[ y_i = h(x_i) = \frac{x_i^2}{20} + v_i \]  

where \( n_i \) and \( v_i \) are zero mean Gaussian random variables with covariance \( Q_i \) and \( R_i \) to reflect the system uncertainty and measurement noise, respectively. In this experiment, we use \( Q_i = 0.1 \) and \( R_i = 0.01 \). The same state-space model has been analyzed before in many publications \([100–103]\).

Given only the noisy measurements \( \{y_i\} (i = 1, \ldots, 100) \), the different filters are applied to estimate the underlying clean state sequence \( \{x_i\} \). For reference, the true states and the corresponding measurements for 100 iterations in an exemplar run are shown in Figure 5-1.

For the nonlinear Kalman filters, the state-space model and covariance \( Q_i \) and \( R_i \) are used to estimate the hidden states. The UKF parameters are set as \( \alpha = 1, \beta = 0 \) and \( k = 2 \). For the PF, we use the generic PF algorithm \([18]\) (the importance density is assumed to be the transitional prior and systematic resampling \([24]\) is used). In this algorithm 200 particles and systematic resampling are applied to implement the
Figure 5-1. True values of the states and corresponding noisy measurements in an exemplar run
estimation. For the proposed FKKF, the Gaussian kernels are applied to map \( x_i \) and \( y_i \) to RKHS with the same kernel parameter \( \varrho = 1 \). The other parameters are set as: regularization terms \( \varsigma_f = 10^{-4} \) and \( \varsigma_h = 10^{-4} \), initial covariance parameter \( \lambda = 10^{-2} \) and sample generation covariance \( \Sigma_i = 5 \). The most important noise parameters are \( q = 0.1 \) and \( r = 0.01 \). These parameters are selected to obtain best estimation performances for all algorithms.

For each trial, the algorithms are run 100 times to validate the mean and variance of the results. The NMSE between the estimations \( \hat{x}_i \) and real positions \( x_i \) of the system are listed in Table 5-1. All the results in the table are in the form of “mean \( \pm \) standard deviation”.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF</td>
<td>0.290480±0.293210</td>
</tr>
<tr>
<td>UFK</td>
<td>0.177950±0.259540</td>
</tr>
<tr>
<td>CKF</td>
<td>0.832920±0.900670</td>
</tr>
<tr>
<td>PF (( m = 200 ))</td>
<td>0.098076±0.217250</td>
</tr>
<tr>
<td>FKKF (( m = 30 ))</td>
<td>0.087914±0.045131</td>
</tr>
<tr>
<td>FKKF (( m = 200 ))</td>
<td>0.050336±0.032467</td>
</tr>
</tbody>
</table>

From Table 5-1, one can conclude that the FKKF algorithm obtains the best estimation performance with the smallest mean and standard deviation of NMSE. It means that the FKKF algorithm is also most robust in this application. Even only using \( m = 30 \) training samples, the FKKF can outperform the PF with 200 particles.

To analyze how the number of samples \( m \) affects the estimation performance, we use different numbers of samples to construct the state transition and measurement operators to estimate the hidden states. The estimation results are plotted in Figure 5-2, by setting \( m = [5, 10, 20, 30, 50, 100, 200] \). It seems that \( m = 50 \) is a good choice to obtain better performance with less computation complexity.
5.5 Discussion

In the previous section, the FKKF and some other existing algorithms are tested to estimate the hidden states generated by a scalar state-space model from the corresponding noisy measurements. The FKKF outperforms other algorithm with a small number of the training samples and shows its robust property. For other algorithms, in some trials algorithms cannot always track and estimate the signal and result in large estimation errors. In this section, we give more discussions about these algorithms and their performances.

For the nonlinear KF algorithms, the Gaussian approximation is not a sufficient description of the nonlinear and non-Gaussian nature of the example. Once the nonlinear KFs cannot adequately approximate the bimodal nature of the underlying posterior, the Gaussian approximation fails. Especially, the CKF gets poor performance, actually, it does not work in this experiment. This is the first time in the dissertation that the CKF cannot outperform the EKF and UKF. There are several reasons to explain this
result: First, the CKF uses just $2 \times n_x$ cubature points to describe the underlying distributions, while the UKF with $3 \times (n_x + 1)$ sigma points and weights controlled by free parameters $\alpha$ and $\beta$ to describe the underlying distributions. When the hidden state dimensionality is large the difference is very small, but for scalar problem this difference is huge. Furthermore, the pair of cubature points defined in (2–17) are additive inverse and the state transition function (5–43) is even. Therefore, it is more difficult to describe the underlying distributions.

For the PF algorithm, 200 particles and the corresponding weights are used to describe the underlying distributions. Therefore, even if the distributions are not Gaussian, PF can still describe them and obtain good results. However, in some cases these propagated samples and weights cannot work well. For example, the resampling step, which is applied to reduce the effects of degeneracy, probably results in a loss of diversity among the particle as the resultant sample will contain many repeated points. In this case, the PF cannot also track and estimate the hidden states. [18]. This result is expected, because there is no feedback in PF and the importance density affects the performance significantly.

For the FKKF algorithm, the distributions represented by the estimated embeddings are propagated at each iteration. Therefore, the nonlinear and non-Gaussian nature of the dynamical system can be described sufficiently well. Furthermore, although the estimated embedding are also expressed in terms of the training samples and weights, the training samples are not propagated directly by the state transition and measurement functions. It is the estimated embedding in RKHS that is propagated by the conditional embedding operators, which can be treated as the counterpart of the state transition and measurement functions in RKHS. The training samples are just used to construct these operators. Therefore, there are no issues of resampling and loss of diversity. The FKKF can obtain best and most robust estimation performance.
5.6 Conclusion

In this chapter, we propose a full-blown kernel Kalman filter (FKKF) algorithm based on the conditional embedding operators. Compared with the KKF-CEO algorithm presented in the previous chapter, the FKKF algorithm uses the estimated states embeddings as the hidden state in RKHS. In addition, with a non-trivial measurement model in RKHS, the FKKF algorithm can more precisely model the system dynamics with two conditional embedding operators in tensor product RKHS $\mathcal{H}_x \times \mathcal{H}_x$ and $\mathcal{H}_y \times \mathcal{H}_x$, which are constructed by samples generated by the system model functions at each iteration.

The FKKF algorithm is tested to estimate the scalar hidden states generated by a synthetic dynamical system from the noisy measurements and the estimation performance is compared with some existing algorithms, including the EKF, UFK, CKF and PF. The proposed algorithm outperforms these algorithm in precision and robustness.

However, more studies about this algorithm are needed to improve it. For example, how to accurately model the multi-dimensional noise in RKHS. In this chapter, the FKKF is applied to solve a scalar estimation problem with very simple noise model $Q_i = qI$ and $R_i = rI$. But for a multi-dimensional dynamical system, such noise models are not sufficient obviously for some complicated systems. Furthermore, how to construct efficient state transition and measurement operators with less samples is also an interesting problem.
CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion and Discussion

In this work, we intend to develop the Kalman in RKHS to solve nonlinear dynamical system problems. In this research direction, we have proposed three Kalman based algorithms, namely the extended kernel recursive least squares based on Kalman filter (Ex-KRLS-KF), the kernel Kalman filter based on conditional embedding operator (KKF-CEO), and the full-blown kernel Kalman filter (FKKF).

First, we analyze the extended recursive least squares (Ex-KRLS) algorithm and point out its limitation. As a extension of the KRLS algorithm, the Ex-KRLS brings the dynamical state (weight) into the system model in RKHS, constructs the state transition operator to describe the system dynamics in RKHS and implement this algorithm by setting the operator $A = \alpha I$. However, our work shows that this dynamical model in RKHS cannot describe general dynamical systems and the Ex-KRLS is just a random walk KRLS algorithm because the parameter $\alpha$ has to be 1 theoretically.

Next, we develop the Ex-KRLS-KF algorithm, which is a combination of the KRLS and nonlinear Kalman filter. To avoid constructing a complicated state model in RKHS, which involves operator learning, we maintain the state model in the input space, while still construct the measurement model in RKHS. Then, we learn the unknown measurement model in RKHS and use them to estimate hidden state by the nonlinear Kalman filter. Furthermore, we obtain the output by feeding the estimated hidden state to the learned measurement model. Because of the more precise state model, tracking performance is better than the Ex-KRLS algorithm. Although, in this algorithm we obtain the hidden states as the intermediate step to compute the output, we cannot estimate real hidden states. Because the hidden states are estimated based on the approximated measurement model, not the real measurement model. It is impossible to estimate real hidden states only using input and output information.
Then, we present the KKF-CEO algorithm to estimate the hidden state with a trivial measurement model in RKHS. To construct a state model in RKHS, we introduce the concepts about embeddings in RKHS and conditional embedding operators. The estimated measurement embeddings, which represent the distribution of measurements, are applied as the hidden states in RKHS and propagated by the conditional embedding operators. The estimated measurement embedding is linked to the mapped measurement by an additive noise model. In addition, the model uncertainty and measurement error are assumed to be zero mean Gaussian noise in RKHS. Based on this state-space model in RKHS, a Kalman filter in RKHS is developed to estimate the measurement embedding, as well as the hidden states in the input space. Because the Gaussian noise in RKHS can represent much wider noise types in the input space, this algorithm is robust in non-Gaussian noise environments to denoise and predict signals. However, its trivial measurement model and hidden state of measurement embeddings in RKHS limits its applications. The algorithm is well suited as a nonlinear model to predict noisy series as we demonstrate.

Finally, we propose the FKKF algorithm also based on the conditional embedding operators. The algorithm is proposed under the assumption that the state-space model is known, like the nonlinear Kalman filters. Using the state transition and measurement functions, we generate training samples at each iteration, including hidden states and corresponding measurements, to construct the conditional embedding operators, which are counterparts of state transition and measurement functions in RKHS. With these operators, we can have the full-blown state-space model in RKHS with the state embeddings as hidden states. Likewise, the model uncertainty and measurement noise are assumed to be zero mean Gaussian noise in RKHS. Because of the non-trivial measurement model, which is constructed from the generated samples, this algorithm is named full-blown Kalman filter in RKHS. In addition, because training samples are generated and distributions are propagated in this algorithm, there is some similarity
between this algorithm and PF. However, they are totally different algorithms. We have given discussions and explanations about this point in Section 5.5. However, this algorithm is not completed. There is still many interesting issues to be researched.

If we treat the KRLS as a Kalman algorithm with a static state model in RKHS, then we can summarize these studied algorithms in a state-space model framework in Figure 6-1. In Figure 6-1 the hidden state, state transition matrices (operators) and measurement matrices (operators) in the algorithms are all presented.

![Figure 6-1. Research line of algorithms](image)

In Chapter 1, we list some requirements that we try to achieve by our proposed algorithms. After presenting all algorithms, let check whether these algorithm achieve these requirements, or not. The results are listed in Table 6-1.

Table 6-1. Algorithm requirements check list

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Learn system model from data directly</th>
<th>Estimation/Prediction output</th>
<th>Estimation/Prediction of hidden state</th>
<th>Robust in a non-Gaussian noise environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex-KRLS-KF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>KKF-CEO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FKKF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
It is clear that the KKF-CEO and FKKF algorithms achieve all requirements, while the Ex-KRLS-KF just achieves some, because it is not a Kalman filter in RKHS, but a hybrid approach.

6.2 Future work

We have presented several algorithms based on Kalman algorithm in RKHS. However, there are still many issue that we need to address to improve these algorithms or develop new algorithms.

6.2.1 Noise model in RKHS

It should be noted that noises in RKHS in all the algorithms, including the Ex-KRLS, KKF-CEO and FKKF, are all assumed to be zero mean Gaussian distributions. Although, this assumption is simple and effective, this noise model cannot represent some specific noises in complicated dynamical system. Especially, how to describe a multi-dimensional noise with full rank covariance matrix in RKHS is very challenging, and the only suggestion at this time is to use a product kernel that controls some of the axis in RKHS. Furthermore, because the kernel mapping is nonlinear, the noise is actually not additive. How to reflect the non-additivity property of noise is an open problem.

6.2.2 Operator Design

In the KKF-CEO and FKKF algorithms, the conditional embedding operators are used to construct the state-space model in RKHS. It is also an open question to find new operators to implement Kalman filter in RKHS with better properties.

6.2.3 Application of Kalman filter in Kernel Adaptive Algorithms

Since the traditional (nonlinear) Kalman filters are applied in some adaptive algorithms, we can also consider the Kernel Kalman algorithm in kernel adaptive filtering. For example, we have studied the first (to our knowledge) hierarchical kernel filter as a kernel version of recurrent neural networks (RNN), namely kernel recurrent system (KRS). Moreover, we developed a kernel version of real-time recurrent learning
(KRTRL) algorithm to train the KRS. The details can be found in Appendix E. Because the nonlinear Kalman filters have been used to train RNN, it is also possible to develop a kernel version of Kalman filter to train the KRS.
In this APPENDIX, we summarize the square-root cubature Kalman filter (SCKF) algorithm. Before giving the algorithm, we denote a general triangularization algorithm (e.g. the QR decomposition) as $D = \text{Tria}(A)$, where $D$ is a lower triangular matrix. The matrices $A$ and $D$ are related as follows: Let $R$ be an upper triangular matrix obtained from the QR decomposition on $A^T$, then $D = R^T$. We use the symbol “/” to represent the matrix right division operator. When we perform the operation $A/B$, it applies the back substitution algorithm for an upper triangular matrix $B$ and the forward substitution algorithm for a lower triangular matrix $B$. 
Algorithm A-1: Square-root Cubature Kalman filter

Initialization: For $i = 0$, set

$\hat{x}_0 = E[x_0]$

$P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T]$.

Factorize $P_0 = D_0D_0^T$.

Computation: For $i = 1, 2, \ldots,$

**Time Update**

Evaluate the cubature points ($n = 1, 2, \ldots, m$)

$X_{n,i-1} = D_{i-1}\xi_n + \hat{x}_{i-1}$, where $m = 2n_x$.

Evaluate the propagated cubature points ($n = 1, 2, \ldots, m$)

$X_{n,i}^* = f(X_{n,i-1}, u_{i-1})$.

Estimate the predicted state

$\hat{x}_i = \frac{1}{m}\sum_{n=1}^{m} X_{n,i}^*$.

Estimate the square-root factor of the predicted error covariance

$D_i^* = \text{Tria}([X_i^* \; D_{Q,i-1}])$

where $D_{Q,i-1}$ denotes a square-root factor of $Q_{i-1}$ such that

$Q_{i-1} = D_{Q,i-1}D_{Q,i-1}^T$ and the weighted, centered matrix

$X_i^* = \frac{1}{\sqrt{m}} \left[ X_{1,i}^* - \hat{x}_{i}^- \; \cdots \; X_{m,i}^* - \hat{x}_{i}^- \right]$.

**Measurement Update**

Evaluate the cubature points ($n = 1, 2, \ldots, m$)

$X_{n,i}^- = D_{i-1}\xi_n + \hat{x}_{i}^-.$

Evaluate the propagated cubature points ($n = 1, 2, \ldots, m$)

$Y_{n,i}^- = h(X_{n,i}, u_i)$.

Estimate the predicted measurement

$\hat{y}_i = \frac{1}{m}\sum_{n=1}^{m} Y_{n,i}^-$.

Estimate the square-root of the innovation covariance matrix

$D_{yy,i} = \text{Tria}([Y_i \; D_{R,i}])$

where $D_{R,i-1}$ denotes a square-root factor of $R_{i-1}$ such that

$R_{i-1} = D_{R,i-1}D_{R,i-1}^T$ and the weighted, centered matrix

$Y_i = \frac{1}{\sqrt{m}} \left[ Y_{1,i}^- - \hat{y}_i^- \; \cdots \; Y_{m,i}^- - \hat{y}_i^- \right]$.

Estimate the cross-covariance matrix

$P_{xy} = X_iY_i^T$.

where the weighted, centered matrix

$X_i = \frac{1}{\sqrt{m}} \left[ X_{1,i} - \hat{x}_{i}^- \; \cdots \; X_{m,i} - \hat{x}_{i}^- \right]$.

Estimate the Kalman gain

$G_i = (P_{xy}/D_{yy,i})/D_{yy,i}$.

Estimate the update state

$\hat{x}_i = \hat{x}_i^- + G_i(y_i - \hat{y}_i^-)$.

Estimate the corresponding error covariance

$D_i = \text{Tria}([X_i - G_iY_i \; G_iD_{Q,i}])$. 

127
APPENDIX B
PROOFS OF THEOREM 4.1 AND THEOREM 4.2

Because of their close relationship, we prove them together.

Proof. Given that $F_i = \phi (K + \varsigma m l_m)^{-1} \gamma^T$, $H_i = I$, $Q_i = q I$ and $R_i = r I$. Starting with $\mu_0 = \varphi(y_0)$, $P_0 = \lambda I$ and following (4–32) to (4–36), we have the first iteration as:

First, we predict the $\mu_{i-1}$

$$
\mu_{i} = F_0 \mu_0 = \phi (K + \varsigma m l_m)^{-1} \gamma^T \mu_0 = \phi a_1. \tag{B-1}
$$

where $a_1 = (K + \varsigma ml_m)^{-1} \gamma^T \mu_0$.

Then we express $P_{i-1}$ as

$$
P_{i-1} = F_0 P_0 F_0^T + q I
= \lambda \phi (K + \varsigma ml_m)^{-1} \gamma^T \gamma \left[(K + \varsigma ml_m)^{-1}\right]^T \phi^T + q I
= \phi \tilde{P}_{i-1} \phi^T + q I. \tag{B-2}
$$

where $\tilde{P}_{i-1} = \lambda (K + \varsigma ml_m)^{-1} \gamma^T \gamma \left[(K + \varsigma ml_m)^{-1}\right]^T$.

For Kalman gain $G_1$,

$$
G_1 = P_{i-1} H_i^T \left[H_i P_{i-1} H_i^T + R_i\right]^{-1}
= P_{i-1} \left[P_{i-1} + r I\right]^{-1}
= \left[\phi \tilde{P}_{i-1} \phi^T + q I\right] \left[\phi \tilde{P}_{i-1} \phi^T + (q + r) I\right]^{-1}
= \left[q \over q + r \left[\phi \tilde{P}_{i-1} \phi^T + (q + r) I\right] + r \over q + r \phi \tilde{P}_{i-1} \phi^T\right]
\left[\phi \tilde{P}_{i-1} \phi^T + (q + r) I\right]^{-1}
= r \over q + r \phi \tilde{P}_{i-1} \phi^T \left[\phi \tilde{P}_{i-1} \phi^T + (q + r) I\right]^{-1}
+ q \over q + r \left[\phi \tilde{P}_{i-1} \phi^T + (q + r) I\right]^{-1}. \tag{B-3}
$$
Further by the matrix inversion lemma, we have

\[
G_1 = \frac{r}{q+r} \Phi \left[ (q+r) I_m + \tilde{P}_1^{-1} \Phi^T \Phi \right]^{-1} \tilde{P}_1^{-1} \Phi^T + \frac{q}{q+r} I
\]

where \( \tilde{G}_1 = \left[ (q+r) I_m + \tilde{P}_1^{-1} \Phi^T \Phi \right]^{-1} \tilde{P}_1^{-1} \).

Next we update covariance matrix

\[
P_1 = (I - G_1) P_1^{-}
\]

\[
= \Phi \left[ \frac{r}{q+r} \tilde{P}_1^{-} - \frac{r}{q+r} \tilde{G}_1 \Phi^T \tilde{P}_1^{-} - \frac{qr}{q+r} \tilde{G}_1 \right] \Phi^T
\]

\[
+ \left( q - \frac{q^2}{q+r} \right) I
\]

\[
= \Phi \tilde{P}_1 \Phi^T + \frac{qr}{q+r} I. \quad (B-5)
\]

where \( \tilde{P}_1 = \frac{r}{q+r} \tilde{P}_1^{-} - \frac{r}{q+r} \tilde{G}_1 \Phi^T \tilde{P}_1^{-} - \frac{qr}{q+r} \tilde{G}_1 \).

Finally, we update the state \( \mu_1 \) as

\[
\mu_1 = \mu_1^{-} + G_1 (\varphi(y_1) - \mu_1^{-})
\]

\[
= \Phi a_1 + \left( \frac{r}{q+r} \Phi \tilde{G}_1 \Phi^T + \frac{q}{q+r} I \right) (\varphi(y_1) - \Phi a_1)
\]

\[
= \Phi \left[ \left( \frac{r}{q+r} I - \frac{r}{q+r} \tilde{G}_1 \Phi^T \Phi \right) a_1 + \frac{r}{q+r} \tilde{G}_1 \Phi^T \varphi(y_1) \right]
\]

\[
+ \frac{q}{q+r} \varphi(y_1)
\]

\[
= \Phi b_1 + \frac{q}{q+r} \varphi(y_1). \quad (B-6)
\]

where \( b_1 = \left[ \frac{r}{q+r} I - \frac{r}{q+r} \tilde{G}_1 \Phi^T \Phi \right] a_1 + \frac{r}{q+r} \tilde{G}_1 \Phi^T \varphi(y_1). \)

Here, we have proven Theorem 2 and Theorem 3 for \( i = 1 \) iteration, then we prove them for \( i > 1 \). Assuming that Theorem 2 and Theorem 3 are both satisfied at \( i - 1 \).
iteration, we have the following equations at $i$ iteration.

\[ \mu_i^- = F_{i-1} \mu_{i-1} \]
\[ = \Phi \left( K + \varsigma m I_m \right)^{-1} \gamma^T \left[ \Phi b_{i-1} + \frac{q}{q + r} \varphi(y_{i-1}) \right] \]
\[ = \Phi a_i \]  \hspace{1cm} (B–7)

where $a_i = \left( K + \varsigma m I_m \right)^{-1} \gamma^T \left[ \Phi b_{i-1} + \frac{q}{q + r} \varphi(y_{i-1}) \right]$

and

\[ P_i^- = F_{i-1} P_{i-1} F_{i-1}^T + qI \]
\[ = F_{i-1} \left[ \Phi \tilde{P}_{i-1} \Phi^T + \frac{qr}{q + r} I \right] F_{i-1}^T + qI \]
\[ = \Phi \tilde{P}_i^- \Phi^T + qI. \]  \hspace{1cm} (B–8)

where

\[ \tilde{P}_i^- = (K + \varsigma m I_m)^{-1} \gamma^T \Phi \tilde{P}_{i-1} \Phi^T \gamma \left[ (K + \varsigma m I_m)^{-1} \right]^T \]
\[ + \frac{qr}{q + r} (K + \varsigma m I_m)^{-1} \gamma^T \gamma \left[ (K + \varsigma m I_m)^{-1} \right]^T \]  \hspace{1cm} (B–9)

Moreover, following from (B–2) to (B–6) other equations in Theorem 2 Theorem 3 can be obtained for $i > 1$. \hfill \Box
APPENDIX C
PROOFS OF THEOREM 4-3

Proof. Let \( e_j = [0, 0, \ldots, 1, \ldots, 0]^T \) (\( 1 \leq j \leq k \)) be a \( k \)-Dimensional standard basis (canonical basis). If \((y_0^j, z_0^j)\) is quantized to \((y_q^j, z_q^j)\), \(\varphi(y_0^j)\) and \(\varphi(z_0^j)\) are quantized as \(\gamma_Q e_j\) and \(\Phi_Q e_j\), respectively. For convince, we define \(c[i] = j\). Then the feature matrices \(\gamma\) and \(\Phi\) are quantized as \(\gamma_Q E\) and \(\Phi_Q E\), where \(E = [e_{c[1]}, \ldots, e_{c[m]}]\).

Substituting the quantized \(\gamma\) and \(\Phi\) into (4–25), we have

\[
F_Q = \Phi_Q E (E^T K_Q E + \varsigma ml)^{-1} E^T \gamma_Q^T.
\]  

(C–1)

Further by the matrix inversion lemma [81], we have

\[
F_Q = \Phi_Q E E^T (K_Q E E^T + \varsigma ml)^{-1} \gamma_Q^T
= \Phi_Q \Lambda (K_Q \Lambda + \varsigma ml)^{-1} \gamma_Q^T
\]  

(C–2)

where \(\Lambda = EE^T\) is a diagonal matrix whose \(j\)th diagonal element \(\lambda_j^Q\) is the number of the samples clustered in the \(j\)th bin by the kernel K-means algorithm. \(\square\)
APPENDIX D
PROOFS OF THEOREM 5.1 AND THEOREM 5.2

Because of their close relationship, we prove them together.

Proof. Given that $F_i = \Phi \tilde{F}^T$, $H = \Psi \tilde{H}^T$, $Q_i = qI$ and $R_i = rI$. Starting with $\mu_0 = \varphi(x_0)$, $P_0 = \lambda I$ and following (5–17) to (5–21), we have the first iteration as:

First, we predict the $\mu_1$

$$\mu_1 = F \mu_0$$
$$= \Phi \tilde{F}^T \mu_0$$
$$= \Phi \tilde{F}^T \mu_0$$
$$= \Phi a_1$$  \hspace{1cm} (D–1)

Then we express $P_1$ as

$$P_1 = F_i P_0 F_i^T + qI$$
$$= \lambda F_i F_i^T + qI$$
$$= \lambda \Phi \tilde{F}^T \gamma \tilde{F}^T \Phi + qI$$
$$= \Phi \tilde{P}_1 \Phi^T + qI$$  \hspace{1cm} (D–2)

where $\tilde{P}_1 = \lambda \tilde{F}^T \gamma \tilde{F}^T$. For Kalman gain $G_1$,

$$G_1 = P_1^{-1} H^T [HP_1^{-1} H^T + R_1]^{-1}$$
$$= \left( \Phi \tilde{P}_1 \Phi^T + qI \right) \left( \Psi \tilde{H} \Phi^T \right)^T$$
$$\left[ \left( \Psi \tilde{H} \Phi^T \right) \left( \Phi \tilde{P}_1 \Phi^T + qI \right) \left( \Psi \tilde{H} \Phi^T \right)^T + rI \right]^{-1}$$
$$= \left( \Phi \tilde{P}_1 \Phi^T \Phi \tilde{H}^T \Psi^T + q \Phi \tilde{H}^T \Psi^T \right)$$
$$\left[ \left( \Psi \tilde{H} \Phi^T \right) \left( \Phi \tilde{P}_1 \Phi^T \Phi \tilde{H}^T \Psi^T + q \Phi \tilde{H}^T \Psi^T \right) + rI \right]^{-1}$$
$$= \Phi \left( \tilde{P}_1 \Phi^T \Phi \tilde{H}^T + q \tilde{H}^T \right) \Psi^T \left[ \Psi \tilde{H} \Phi^T \Phi \left( \tilde{P}_1 \Phi^T \Phi \tilde{H}^T + q \tilde{H}^T \right) \Psi^T + rI \right]^{-1}$$  \hspace{1cm} (D–3)
Further by the matrix inversion lemma, \((A - BD^{-1}C)^{-1}BD^{-1} = A^{-1}B(D - CA^{-1}B)^{-1}\), let \(A = I_m\), \(B = (\tilde{P}_1^{-1} \Phi \tilde{H}^T + q \tilde{H}^T)\) \(\psi^T\), \(C = -\psi \tilde{H} \Phi^T \psi\), and \(r I\), where \(I_m\) is a \(m \times m\) identical matrix. we have

\[
G_1 = \Phi [r I_m + (\tilde{P}_1^{-1} \Phi \tilde{H}^T + q \tilde{H}^T) \psi^T \psi \tilde{H} \Phi^T \psi]^{-1} (\tilde{P}_1^{-1} \Phi \tilde{H}^T + q \tilde{H}^T) \psi^T
\]

\[
= \Phi \tilde{G}_1 \psi^T.
\]

(D–4)

where \(\tilde{G}_1 = [r I_m + (\tilde{P}_1^{-1} \Phi \tilde{H}^T + q \tilde{H}^T) \psi^T \psi \tilde{H} \Phi^T \psi]^{-1} (\tilde{P}_1^{-1} \Phi \tilde{H}^T + q \tilde{H}^T)\).

Next we update covariance matrix

\[
P_1 = (I - G_1 H) P_1^{-1}
\]

\[
= P_1^{-1} - G_1 H P_1^{-1}
\]

\[
= \left( \Phi \tilde{P}_1^{-1} \Phi^T + q I \right) - \Phi \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T \psi \left( \Phi \tilde{P}_1^{-1} \Phi^T + q I \right)
\]

\[
= \Phi \tilde{P}_1^{-1} \Phi^T - q \Phi \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T - \Phi \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T \Phi \tilde{P}_1^{-1} \Phi^T + q I
\]

\[
= \Phi \left( \tilde{P}_1^{-1} - q \tilde{G}_1 \psi^T \psi \tilde{H} - \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T \Phi \tilde{P}_1^{-1} \right) \Phi^T + q I
\]

\[
= \Phi \tilde{P}_1^{-1} \Phi^T + q I.
\]

(D–5)

where \(\tilde{P}_1 = \tilde{P}_1^{-1} - q \tilde{G}_1 \psi^T \psi \tilde{H} - \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T \Phi \tilde{P}_1^{-1}\).

Finally, we update the state \(\mu_1\) as

\[
\mu_1 = \mu_1^{-1} + G_1 (\psi(y_1) - H \mu_1^{-1})
\]

\[
= \Phi a_1 + \Phi \tilde{G}_1 \psi^T (\psi(y_1) - \psi \tilde{H} \Phi^T a_1)
\]

\[
= \Phi \left( a_1 + \tilde{G}_1 \psi^T \psi(y_1) - \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T a_1 \right)
\]

\[
= \Phi b_1.
\]

(D–6)

where \(b_1 = \left( a_1 + \tilde{G}_1 \psi^T \psi(y_1) - \tilde{G}_1 \psi^T \psi \tilde{H} \Phi^T a_1 \right)\).

Here, we have proven \(Theorem 5.1\) and \(Theorem 5.2\) for \(i = 1\) iteration, then we prove them for \(i > 1\). Assuming that \(Theorem 5.1\) and \(Theorem 5.2\) are both satisfied at
At the $(i-1)$ iteration, we have the following equations at $i$ iteration.

\[
\mu_i^- = F \mu_{i-1} \\
= \Phi \tilde{F} \gamma^T \Phi b_{i-1} \\
= \Phi a_i,
\]  

where $a_i = \tilde{F} \gamma^T \Phi b_{i-1}$ and

\[
P_i^- = FP_{i-1}F^T + qI \\
= \Phi \tilde{F} \gamma^T \left( \Phi \tilde{P}_{i-1} \Phi^T + qI \right) \gamma \tilde{F} \Phi^T + qI \\
= \Phi \tilde{F} \left( \gamma^T \Phi \tilde{P}_{i-1} \Phi^T \gamma + q\gamma^T \gamma \right) \tilde{F} \Phi^T + qI \\
= \Phi \tilde{P}_i^- \Phi^T + qI
\]

where $\tilde{P}_i^- = \tilde{F} \left( \gamma^T \Phi \tilde{P}_{i-1} \Phi^T \gamma + q\gamma^T \gamma \right) \tilde{F}^T$.

Moreover, following from $(D-2)$ to $(D-6)$ other equations in Theorem 2 Theorem 3 can be obtained for $i > 1$. $\square$
APPENDIX E
KERNEL RECURRENT SYSTEM TRAINED BY REAL-TIME RECURRENT LEARNING ALGORITHM

E.1 Introduction

In this appendix, we propose a novel hierarchical kernel adaptive filter algorithm for time series prediction with recurrent hidden state model, which is learned from the processing data and is able to describe the dynamics of the data.

Since the success of the support vector machine (SVM) [7], the kernel methodology has been applied to many algorithms of machine learning and adaptive filters. Utilizing the famed kernel trick, these linear methods have been recast in high dimensional reproducing kernel Hilbert spaces (RKHS) [48–51] to yield more powerful nonlinear extensions, including kernel principal component analysis (KPCA) [11, 12] and kernel independent component analysis (KICA) [13, 14]. Recently, some on-line kernel adaptive filter algorithms are also developed to solve many nonlinear regression problems, such as kernel recursive least squares (KRLS) [16], kernel least mean squares (KLMS) [15] and kernel recurrent gamma network (KRGN) [92] algorithms, etc.

The KRLS and KLMS algorithms are able to discover the underlying input-output mapping very well for stationary cases. However, because of absence of hidden states, these algorithms cannot describe the underlying dynamics of the processing data for non-stationary cases. Therefore, they are not able to achieve good performance in such cases. In the KRGN algorithm, although the recursive Gamma filter is implemented into the RKHS, the specific topology only allows local recursion on the past of the input, which is easy to control stability but does not allow a full recurrent state. To solve global recursiveness, the extended kernel recursive least squared (Ex-KRLS) algorithm [17] was proposed. But, it can only implement a random walk model in the hidden state. Then, another extended kernel recursive least squares algorithm (Ex-KRLS-KF) was proposed based on the Kalman filter (KF) [76], which can be applied to any known linear or nonlinear hidden state model cases. For both of these algorithms (Ex-KRLS and
Ex-KRLS-KF), however, the hidden state model has to be known in advance, which may not be available in many signal processing problems.

To construct and learn the underlying hidden state model, the idea of recurrent neural networks (RNNs) [96] is adopted. In kernel recurrent systems, the KLMS algorithm is applied instead of the original perceptron algorithm, while making the topology recurrent, as in Jordan and Elman’s networks [97–99]. Furthermore, a kernel version of real-time recurrent learning (RTRL) algorithm [94, 95] is derived to learn this recurrent network. To obtain a stable system, the teacher forcing [93, 94] technique is applied to the KRTRL which substitutes the dynamics of the hidden state using the desired response to avoid system instability.

The rest of the appendix is organized as follows. In Section 2, the kernel recurrent system (KRS) are described. Next, the kernel RTRL (KRTRL) algorithm is derived in Section 3. Then, the experiment of Lorenz time series prediction is presented to evaluate the proposed algorithm in Section 4. Finally, discussions and conclusion are given in Section 5.

### E.2 Kernel recurrent networks

In this section, we present a hierarchical general kernel recurrent network architectures, namely the state-space model and point out that any state-space model can be subsumed by a specific state-space model with a kernel state model and a linear measurement model. Then, in the next section, we will develop a kernel RTRL algorithm to train this specific state-space model.

Fig. E-1 shows the block diagram of a state-space model. \( u_i \in \mathbb{R}^{n_u}, x_i \in \mathbb{R}^{n_x} \) and \( y_i \in \mathbb{R}^{n_y} \) are inputs, hidden states and output at time \( i \), respectively. The state-space model is

\[
x_{i+1} = f(x_i, u_i) \quad \text{(E–1)}
\]

\[
y_i = h(x_{i+1}) \quad \text{(E–2)}
\]
Figure E-1. State-space model, the feedback part is shown in red.

where \( f_i(x_i, u_i) = [f^{(1)}(x_i, u_i), \ldots, f^{(n)}(x_i, u_i)]^T \)

and \( h_i(x_i) = [h^{(1)}(x_{i+1}), \ldots, h^{(n)}(x_{i+1})]^T \). In this paper the superscript \((k)\) denotes the \(k\)th component of a vector or the \(k\)th vector of a matrix.

In order to simplify the computation and reduce learning time, we modify this state-space model as

\[
\begin{bmatrix}
  x_{i+1} \\
  y_i
\end{bmatrix}
= \begin{bmatrix}
  f(x_i, u_i) \\
  h \circ f(x_i, u_i)
\end{bmatrix}
\]  

\[ (E-3) \]

\[
y_i = \begin{bmatrix}
  0 & I_{n_y}
\end{bmatrix}
\begin{bmatrix}
  x_{i+1} \\
  y_i
\end{bmatrix}
\]  

\[ (E-4) \]

We denote \( \begin{bmatrix}
  x_{i+1} \\
  y_i
\end{bmatrix} \) by \( s_{i+1} \) as a new hidden state, and \( \begin{bmatrix}
  0 & I_{n_y}
\end{bmatrix} \) by \( W_m \) as a known measurement matrix, where \( I_{n_y} \) is an \( n_y \times n_y \) identity matrix, and \( \circ \) is the composition operator. Furthermore, let \( g(s_i, u_i) = f(x_i, u_i) \).

We map \( s_i \) and \( u_i \) into the RKHS \( H_s \) and \( H_u \) as \( \varphi(s_i) \in H_s \) and \( \phi(u_i) \in H_u \), respectively. Then the new non-linear state transition weights \( W_{H} = \begin{bmatrix}
  g(\cdot, \cdot) \\
  h \circ g(\cdot, \cdot)
\end{bmatrix} \) are in the same RKHS \( H_{su} = H_s \otimes H_u \), where \( \otimes \) is the tensor operator. We denote \( \varphi(s_i) \otimes \phi(u_i) \in H_{su} \) by \( \psi(s_i, u_i) \).
At this point, we can express general state-space recurrent networks using a special kernel state-space model as

\[
\begin{align*}
s_{i+1} &= W_H^T \psi(s_i, u_i) \\
y_i &= W_m s_{i+1}
\end{align*}
\tag{E–5}
\tag{E–6}
\]

where \(W_H\) are weights in RKHS and \(W_m\) is a known linear matrix.

E.3 On-line recurrent system learning

In the previous section, nonlinear recurrent systems are reformulated in (E–5) and (E–6). To perform on-line learning in this network, we develop the kernel real-time recurrent learning (KRTRL) algorithm in this section and teacher forcing technique is also introduced to avoid instability during training.

E.3.1 Kernel real-time recurrent learning algorithm

The kernel real-time recurrent learning (KRTRL) is developed based on the RTRL algorithm [94], which derives its name from the fact that adjustments are made to the weights of a fully connected recurrent network in real time. The KRTRL algorithm can also update the weights in RKHS, which are functions actually, while the network continues to perform its signal-processing function. The KRTRL is derived with respect to (E–5) and (E–6). Without loss of generality, all kernels involved in the algorithm are Gaussian kernels. For the RKHS \(\mathcal{H}_u\) and \(\mathcal{H}_s\), the kernels are \(k_{\sigma_u}(x, y)\) and \(k_{\sigma_s}(x, y)\) with kernel parameters \(\sigma_u\) and \(\sigma_s\), respectively.

To complete the description of this learning process, we need to calculate the gradient of the error surface with respect to \(\omega_k \in \mathcal{H}_{su}\), which is the \(kth\) component of \(W_H\). To do this, we first use (E–6) to define the \(n_y \times 1\) error vector:

\[
e_i = d_i - y_i,
\tag{E–7}
\]
and the instantaneous sum of squared errors at time-step $i$ is defined in term of $e_i$ by

$$E_i = \frac{1}{2} e_i^T e_i.$$  \hfill (E–8)

To implement this on-line learning algorithm, we use the method of steepest descent, which requires knowledge of the gradient matrix $\frac{\partial E_i}{\partial \omega_k}$, written as

$$\frac{\partial E_i}{\partial \omega_k} = \frac{\partial e_i^T e_i}{2\partial \omega_k} = -e_i^T \frac{\partial y_i}{\partial \omega_k} = -e_i^T \frac{\partial y_i}{\partial s_{i+1}} \frac{\partial s_{i+1}}{\partial \omega_k}.$$ \hfill (E–9)

According to (E–5) and (E–6), we have

$$\frac{\partial y_i}{\partial s_{i+1}} = W_m,$$ \hfill (E–10)

and

$$\frac{\partial s_{i+1}}{\partial \omega_k} = \frac{\partial W_H^T \psi(s_i, u_i)}{\partial \omega_k} = W_H^T \frac{\partial \psi(s_i, u_i)}{\partial \omega_k} + I_n s_{\psi(s_i, u_i)}^T$$ \hfill (E–11)

where $I_n$ is the $n_s \times n_s$ identity matrix and $I_n^{(k)}$ is the $kth$ column of the identity matrix.

According to representation theory, we can express $\omega_{k,i}$ the weight at time $i$ as a linear combination of $\{\psi(s_j, u_j)\}_{j=0}^{i-1}$ such as

$$\omega_{(k),i} = \Psi_i c_{k,i}$$ \hfill (E–12)

where $\Psi_i = [\psi(s_0, u_0), \ldots, \psi(s_{i-1}, u_{i-1})]$ and $c_{k,i} \in \mathbb{R}^i$, and $W_H$ can be expressed as

$$W_H = \Psi_i C_i$$ \hfill (E–13)

where $C_i = [c_{1,i}, \ldots, c_{n_s,i}] \in \mathbb{R}^{i \times n_s}$. 

139
Then the first term of the last line in (E–11) is calculated by

\[ W^T \frac{\partial \psi(s_i, u_i)}{\partial \omega_k} = C^T \frac{\partial \psi^T(s, u)}{\partial s_i} \frac{\partial s_i}{\partial \omega_k} \]

\[ = 2\sigma_x C^T D_i S_i^T \frac{\partial s_i}{\partial \omega_k} \]

\[ = \Gamma_i \frac{\partial s_i}{\partial \omega_k} \]  

(E–14)

where \( D_i = \text{diag}(\Psi^T \psi(s_i, u_i)) \),

\( S_i = [(s_0 - s_i), ..., (s_{i-1} - s_i)] \), and \( \Gamma_i = \frac{\partial s_i}{\partial s_i} = 2\sigma_x C^T D_i S_i^T \). Substituting (E–14) into (E–11), we have the following recursive equation:

\[ \frac{\partial s_{i+1}}{\partial \omega_k} = \Gamma_i \frac{\partial s_i}{\partial \omega_k} + l_n^{(k)} \psi(s_i, u_i)^T. \]

(E–15)

If we assume that \( \frac{\partial s_i}{\partial s_i} = 0 \), then we can express \( \frac{\partial s_i}{\partial \omega_k} \) as

\[ \frac{\partial s_i}{\partial \omega_k} = M_{k,i} \psi^T, \]

(E–16)

where \( M_{k,i} \) is an \( n_s \times i \) matrix and \( M_{k,1} = [0, 0, ..., 0]^T \). Furthermore, (E–15) can be rewritten as

\[ \frac{\partial s_{i+1}}{\partial \omega_k} = \Gamma_i M_{k,i} \psi^T + l_n^{(k)} \psi(s_i, u_i)^T \]

\[ = [\Gamma_i M_{k,i}, l_n^{(k)}] \psi_i^T \]

\[ = M_{k,i+1} \psi_i^T \]

(E–17)

where

\[ M_{k,i+1} = [\Gamma_i M_{k,i}, l_n^{(k)}]. \]

(E–18)

Substituting (E–10) and (E–16) into (E–9), we have

\[ \frac{\partial E_i}{\partial \omega_k} = -e_i^T W_m M_{k,i+1} \psi_i^T. \]

(E–19)

and can update \( \omega_k \) by

\[ \omega_{k,(i+1)} = \omega_{k,i} + \eta \psi_{i+1} (W_m M_{k,i+1})^T e_i \]

(E–20)
where $\eta$ is the learning rate parameter.

Therefore, the feature matrix $\Psi_i$ and coefficient matrix $C_i \in \mathbb{R}^{i \times n_s}$ should be updated by

$$\Psi_{i+1} = [\Psi_i, \psi(x_i, u_i)]$$  \hspace{1cm} (E–21)

$$C_{i+1}^{(k)} = c_{k, i+1} = [c_{k, i}, 0]^T + \eta_1 M_{k, i+1}^T W_m e_i$$ \hspace{1cm} (E–22)

$$(k = 1, 2, \ldots, n_x)$$

If we substitute (E–18) into (E–22), we have

$$M_{k, i+1}^T W_m e_i = \left[ \Gamma_i M_{k, i}, I_{n_x}^{(k)} \right]^T W_m e_i$$

$$= \left[ (M_{k, i}^T \Gamma_i^T W_m e_i), (W_m^T e_i)^{(k)} \right]^T$$  \hspace{1cm} (E–23)

At this point, the learning procedure is complete and is summarized in Algorithm E-1.

The computational complexity of KRTRL algorithm is $O(n_x n_y i + n_x^2 i)$, considering that $D_i$ is a diagonal matrix. The computational complexity increases linearly with the sample number $i$, like the KLMS algorithm.

### E.3.2 Teacher forcing

The KRTRL algorithm is a gradient based approach applied to a recurrent system as can be seen in Fig. E-1. However, unlike the RTRL algorithm in which the activation functions are fixed in advance and only the weights are updated to adjust the network, the KRTRL constructs the expected KRS by adjusting the functions themselves.

Therefore, the stability of the learning system is a big issue for this learning algorithm, and unfortunately, the stability analysis of recurrent functions is so complicated that there is no general method to easily specify stability conditions. Specifically, the updated functions and calculated gradients both depend on the inputs $u_i$ and hidden states $s_i$, which are functions in this case. In the KRTRL algorithm the gradient of hidden states with respect to the functions $\frac{\partial s_{i+1}}{\partial \omega_k}$ is propagated forward by (E–15) to (E–18), so the propagated gradient cannot always reflect the real contribution of $\omega_k$ to the current cost.
Algorithm E-1: Kernel Real-Time Recurrent Learning

**Initialization:** For $i = 0$.
- Input Dim.: $n_u$, State Dim.: $n_s$ and Output Dim.: $n_y$
- randomly set $u_0$, $s_0$ and $s_1$.
- Gaussian kernel size: $\sigma_s$ and $\sigma_u$
- feature matrices: $\Psi_{i+1} = [\psi(s_i, u_i)]$
- Coeff. matrix: randomly set $C_{i+1} \in \mathbb{R}^{1 \times n_s}$
- Meas. matrix: $W_m \in \mathbb{R}^{1 \times n_y}$
- Gradient matrix: $M_{k,i+1} = 0_{n_s \times 1}$ for $k = 1,\ldots,n_s$
- Learning rate parameters: $\eta$

**Learning:** For $i = 1,\ldots$.
- Forward pass:
  \[
s_{i+1} = C_i^T \psi_i^T \psi(s_i, u_i), \quad y_i = W_m s_{i+1}
  \]
  \[
e_i = d_i - y_i
  \]
- Backward pass:
  \[
  S_i = [(s_0 - s_i),\ldots,(s_{i-1} - s_i)]
  \]
  \[
  D_i = \text{diag}(\psi_i^T \varphi(s_i, u_i))
  \]
  \[
  \Gamma_i = 2\sigma_s C_i^T D_i S_i^T
  \]
  \[
  M_{k,i+1} = \begin{bmatrix} \Gamma_i M_{k,i} & 1^{(k)} \end{bmatrix} \text{ for } k = 1,\ldots,n_s
  \]
- Update weights in RKHS $W_H$:
  \[
  \psi_{i+1} = [\psi_i, \psi(s_i, u_i)]
  \]
  \[
  C_{i+1}^{(k)} = C_i^{(k)} + \eta M_{k,i} \Gamma_i W_m^T e_i \text{ for } k = 1,\ldots,n_x
  \]
  \[
  C_{i+1} = \begin{bmatrix} C_{i+1}^T, \eta (W_m^T e_i)^T \end{bmatrix}^T
  \]

function, because if the dynamics are unstable the weight update will be wrong. The update made based on the gradient does not guarantee that the function is modified in a proper way, whatever initial condition or how small learning rate.

Fortunately, we can apply the teacher forcing technique to avoid this problem and train the KRS in a stable manner. The idea of the teacher forcing technique is to replace, in the training procedure, the actual hidden state $s_{i+1}^{(j)}$ by the corresponding desired response $d_{i+1}^{(j)}$ in subsequent computation of the behavior of the network.

This can be done at every iteration or periodically to avoid divergence of states. The teacher forcing technique uses the desired response, which is assumed bounded and stable, to constrain and guide the learning dynamics of the hidden states by following the sequential desired responses and to force the KRS to converge in a proper way, avoiding system instability.
In order to derive a learning algorithm for this situation, we once again differentiate the cost function with respect to \( \omega_k \). We find only the term \( M_{k,i} \) need to be modified. In (E–16), \( \frac{\partial s_i}{\partial \omega_k} = M_{k,i} \Psi_i^T \). If \( s^{(j)}_{i+1} \) is replaced by \( d^{(j)}_i \), then \( \frac{\partial s^{(j)}_{i+1}}{\partial \omega_k} \) should be equal to zero. That is to say the \( j \)th row of the matrix \( M_{k,i} \) is set as zeros. If all the hidden states \( s^{(j)}_{i+1} (j = 1, \ldots, n_k) \) are all substituted by \( d_i \), the system training procedure degrades to a MIMO KLMS algorithm. In addition, the teacher forcing is applied once every \( t \) steps.

**E.4 Experiments and results**

To evaluate the proposed KRTRL algorithm, we chose to predict the Lorenz time series and the performance is compared with the KLMS algorithm. The system is nonlinear, three-dimensional and deterministic, defined by the set of differential equations given in [17]. Here, the experimental data are generated by the coefficients \( \beta = 8/3, \sigma = 10 \) and \( \rho = 28 \).

The first order difference approximation is used with a step size 0.01 to obtain the signal \( x_i = [x_i(1)x_i(2)x_i(3)]^T \). We pick the first component \( x(1) \) as the input signal to implement a short prediction of the other components. The short term prediction task can be formulated as follows: using 10 past data \( u_i = [x(1)i_{-10}, \ldots, x(1)i_{-1}]^T \) as the input to predict \( x(k), k = 1, 2, 3 \). For each prediction only the input signal \( x(1) \) and desired signal \( x(k) \) are available. We do not have the whole three signals at the same time. In the first 3000 time series, we randomly obtain 2000 time-steps to learn the filters, and use the next 200 time-steps to test these algorithms. To compare the prediction performances in a fair way, we run 50 independent simulations and the mean squared error (MSE) and signal noise ratio (SNR) are recorded.

All kernel parameters involved in this experiment are set as 1. For the KLMS algorithm the learning rates are set as 0.1 and 0.01. For our KRTRL algorithm, the learning rate is set as 0.1 and the dimension of the hidden states is 2. The teacher forcing technique is applied at every step \( (t = 1) \). \([x_i(k), x_{i-1}(k)]\) is used as teacher
signal to train the KRS. For testing, the hidden state corresponding to \( x_i(k) \) is chosen as the system output. The prediction performances are tabulated as below:

Table E-1. Using \( x(1) \) as inputs to predict \( x(1) \), \( x(2) \) and \( x(3) \)

<table>
<thead>
<tr>
<th>Des.</th>
<th>Algorithms</th>
<th>SNR</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x(1) )</td>
<td>KLMS(( \eta = 0.1 ))</td>
<td>25.741±1.7716(dB)</td>
<td>0.00096218±0.00083813</td>
</tr>
<tr>
<td></td>
<td>KLMS(( \eta = 0.01 ))</td>
<td>12.193±2.1166(dB)</td>
<td>0.021771±0.013692</td>
</tr>
<tr>
<td></td>
<td>KRS(( \eta = 0.1 ))</td>
<td>24.3963±1.6035(dB)</td>
<td>0.0012755±0.00097267</td>
</tr>
<tr>
<td>( x(2) )</td>
<td>KLMS(( \eta = 0.1 ))</td>
<td>-2.223±0.25915(dB)</td>
<td>1.6834±0.12373</td>
</tr>
<tr>
<td></td>
<td>KLMS(( \eta = 0.01 ))</td>
<td>3.483±0.27802(dB)</td>
<td>0.44998±0.026276</td>
</tr>
<tr>
<td></td>
<td>KRS(( \eta = 0.1 ))</td>
<td>23.4077±2.4422(dB)</td>
<td>0.0051275±0.002389</td>
</tr>
<tr>
<td>( x(3) )</td>
<td>KLMS(( \eta = 0.1 ))</td>
<td>-2.5039±0.22352(dB)</td>
<td>1.5296±0.11027</td>
</tr>
<tr>
<td></td>
<td>KLMS(( \eta = 0.01 ))</td>
<td>3.0712±0.25711(dB)</td>
<td>0.42038±0.024646</td>
</tr>
<tr>
<td></td>
<td>KRS(( \eta = 0.1 ))</td>
<td>17.9362±1.7771(dB)</td>
<td>0.014737±0.005885</td>
</tr>
</tbody>
</table>

From these results, one can conclude that both algorithms can obtain good prediction performances while using \( x(1) \) to predict \( x(1) \). But the KLMS algorithm get poor performances when predicting \( x(2) \) or \( x(3) \) time series using \( x(1) \). While, our algorithm can obtain good prediction performances in predicting the there dimensional system. This is because the mappings from \( x(1) \) to \( x(2) \) or \( x(3) \) are more complex and the error increases with iterations proportionally to the largest Lyapunov exponent of the system that produced the time series, which is positive in this case (Chaotic system).

The KRS learned by the KRTRL algorithm is designed to describe the system dynamics, which can quantify better the data structure to implement the prediction task. Therefore, there is no surprise that our proposed algorithm can still be successful.

E.5 Conclusion

In this appendix, we proposed for the first time a hierarchical kernel filter called the kernel recurrent system (KRS) which can describe general state-space model recurrent networks and implement the nonlinear computations in the RKHS. Furthermore, a kernel real-time recurrent learning (KRTRL) algorithm is developed to train the KRS. The teacher forcing technique is applied to modify the KRTRL algorithm to improve the
learning tasks. The KRTRL algorithm is applied to Lorenz time series prediction, and the prediction performances outperforms the KLMS algorithm when the input to output mappings are non-stationary.

The computational complexity of the KRTRL algorithm is similar to the KLMS increasing linearly with the sample number $i$. However, because of the recurrent architecture, the whole coefficients are updated when an new sample arrives, unlike the KLMS algorithm, which only updates current coefficient. This property also enhance the learning capability of this algorithm. In the experiment, the teacher forcing is applied at every step. We can also increase the interval $t$, like $t = 2$ or $3$, to learn the state dynamics better, which however requires the smaller learning rate and longer training time.
REFERENCES


[18] B. Ristic, S. Arulampalam, N. Gordon, Beyond the Kalman Filter: Particle Filters


February 1999.


1–25, 1996.


[29] G. Welch, G. Bishop, An Introduction to the Kalman Filter. Technical report,


[31] P. Maybeck, Stochastic Models, Estimation and Control, vol. 1, New York:


[51] Martin Sewell, Kernel Methods, Department of Computer Science, University College London, UK.


Pingping Zhu was born in Wuhan, China. He received the B.S. degree in electrical and information, the M.S. degree in pattern recognition and intelligent system in 2006 and 2008 respectively, from Huazhong University of Science and Technology, Wuhan, China. He also received the M.S. and the Ph.D. degree in electrical and computer engineering in 2010 and 2013, respectively, from University of Florida, Gainesville, USA. He studied and developed the fast (inverse) discrete cosine transform (DCT/IDCT) algorithms in video compression (MPEG-4) and fast modified (inverse) discrete cosine transform (MDCT/IMDCT) algorithms in audio codec (G.711) in Institute of Pattern Recognition and Artificial Intelligence, Huazhong University of Science and Technology. In 2009, he joined the Computational NeuroEngineering Laboratory at the University of Florida as a Ph.D. student. His research focuses on statistical signal processing, adaptive filtering, and machine learning.